acoustic_spaces

User manual

Version 4.2.1

https://naelolaiz.github.io/acoustic-spaces/

acoustic_spaces contributors, GPL-3.0-or-later

Contents

User manual

User manual 6

- Contents 6
- First steps 6
- Building a scene 7
- Simulating rooms 8
- Listening in space 9
- Advanced workflows 10
- How far to trust the results 11
- Reference 11

Getting started 12

- Install 12
- Install from source 12
- Where the results go 18
- Language 18

Graphical editor 20

- Drawing rooms 25
- Outdoor scenes 26
- Exporting to Blender 26

Example scenes 28

- Rooms 28
- Halls and churches 28
- Sound systems 29
- Outdoors 30
- Models from CAD and BIM files 31
- Measurements and material libraries 31

Scene files 32

- Geometry 32
- Materials 35
- Walls, doors and openings 40
- Audience seating 42
- Sources and directivity 43
- Walks 49
- Air, response length and seed 51

Simulation 52

- Scattering 56
- Frequency bands 58
- Edge diffraction 58
- Low frequencies: early reflections in pressure 63

Low frequencies: the wave solver 65
Performance 70
Validation 76

Spatial audio and listening 77
Spatial outputs 77
Listening: convolution and real-time playback 79

Coverage maps 89
Uncertainty 91

Outdoor scenes 92
The ground and the sky 92
Terrain 93
Buildings and barriers 95
Buildings from a map 97
Ground impedance 97
Ground zones 99
Wind and temperature 100
Turbulence 112
Vegetation 115
What the simulation does outdoors 118
Results outdoors 119
Levels over a long term 119
Passing vehicles 121
Aircraft by ECAC Doc 29 124
Vibration through the ground 127
In the editor 128
What is not modelled yet 129

Demos 132
Running a demo 132
A demo of any scene 134
An orchestra in the concert hall 137
A choir and an organ in the church 139
A concert in the stadium 142
A concert across a lake 145
A walk through the gallery 148
Vehicles passing by 149
Trains passing by 154
A helicopter flying over 158
A helicopter over the stadium concert 160
Road works in the street 162
Aircraft taking off and landing 164

Measurements 166
Measuring impulse responses 166
Comparing with measurements 168
Fitting materials to measurements 170

Running on another computer 173
Packs 173
The container image 174
A server over SSH 174
Hugging Face Jobs 175

Reference

Validation 176
Checks against theory, on every change 177
Cross-check against pyroomacoustics 184
Measured rooms (BRAS) 189
Edge diffraction against the exact solution 200
Reflections that fade at edges against the exact solution 209
Ground reflection against the exact solution 218
Outdoor propagation against measurements (Hannover) 220
Turbulence against a parabolic equation through random eddies 222
Vegetation against a particle simulation 224
Leaves against closed forms and ISO 9613-2 225
The wave solver in wind and temperature against a parabolic equation 227
Moving air in the wave solver 228
Outdoor results against exact solutions 232
Outdoor sound in wind against a fast field program 235
The wide-angle parabolic equation 243
Terrain against closed forms and the exact wedge 251
Outdoor scenes on the GPU 254
Weather zones against a Crank-Nicholson parabolic equation 256
Barriers and buildings in wind against a full-wave solution 259
Rows of roofs and barriers against a wave calculation 266
Levels over a long term 276
Moving sources against the exact field 277
Helicopter rotors against theory and a certified level 282
Sirens and road vehicles against their rules 285
Trains against Schall 03 and the EU’s limits 287
Road works against Defra’s rows and the EU’s limits 288
Aircraft noise against Doc 29 Volume 3 289
Flight profiles against Doc 29 Volume 3 290
An aircraft in the engine against Doc 29 291
Ground-borne vibration against the FTA’s manual 292
Edge diffraction in whole cities 293
Wave-based solver 294
Real-time convolver 309
Known issues 309
Not validated yet 322
Reproducing 325

API reference **327**
Scenes and rooms 328
Source directivity 350
Geometry 357
Materials 361
Simulation 363
Coverage maps 382
Edge diffraction 385
Insertion loss 396
Early reflections in pressure 396
Leak finder 403
Audience seating 405
Wave-based solver 407
The late stage on a GPU 419
Parameters, air and impulse responses 428
Spatial outputs 440
Rendering and playback 447
Walkthroughs 449
Aircraft noise (ECAC Doc 29) 475
Ground-borne vibration 497
Sound through a façade 504
Head tracking 505
Measurement 509
Comparing with measurements 512
Fitting materials to measurements 514
Audio files and results 521
Example scenes 522
3D view and plots 524
Export to Blender (glTF) 533

User manual

This manual is a path through the documentation, from a first simulation of a plain room to the advanced workflows. Each chapter names an example scene to try and a command or an editor menu to try it with, and links to the page that describes it in full; the pages themselves are the reference, so the manual repeats none of their detail. Read it in order the first time, or jump to a chapter from the contents.

The examples come with the package. `acoustic-spaces examples --copy examples` copies them into a folder `examples` that you can edit, so the commands below run as written, from an installed copy as from a source checkout. The example scenes page lists them all, in the groups the editor's **File > Open example** shows.

Contents

1. First steps: installing, a first simulation, looking at the room, the editor and listening to the result.
2. Building a scene: the geometry, from dimensions, a floor plan or a CAD or BIM model; materials; sources and listeners; rooms joined by walls and doors; audience seating; the air and the seed.
3. Simulating rooms: how a run works and its settings, sound round corners, low frequencies and the wave solver, how far a result may be off, coverage maps, and speed and memory.
4. Listening in space: binaural, Ambisonics, loudspeakers and microphones, mixing several sources, real-time playback and head tracking, walks through a scene and moving sources.
5. Advanced workflows: sound systems, outdoor noise, demos, measuring a real room and fitting its model, long runs on another computer, and scripting in Python.
6. How far to trust the results: the validation report and the known issues.
7. Reference: every page of the documentation.

First steps

Installing

```
pip install "acoustic-spaces[viz,gui]"    # with the 3D view and the graphical editor
pip install "acoustic-spaces[viz,gui,measure,sofa,gpu,ifc,sketchup]"  # every optional part
acoustic-spaces info                      # the version, the compiled core and the GPU
```

What each extra adds is in Getting started, and installing from a source checkout below it.

A first simulation

The `shoebox` example is an 8 × 6 × 3 m room from an OBJ file, with a talker and one seat: the simplest case every method must get right.

```
acoustic-spaces examples --copy examples
acoustic-spaces validate examples/shoebox/scene.yaml   # the scene's checks
acoustic-spaces inspect examples/shoebox/scene.yaml    # geometry, materials, reverberation
estimates
acoustic-spaces simulate examples/shoebox/scene.yaml --out results --plot
```

`simulate` prints the ISO 3382-1 parameters per octave band, the level, the STI and the echo criteria, and writes the impulse responses in each format the scene lists, with `parameters.json` and, with `--plot`, the echogram and the arrival directions. The printout, why its T30 is longer than the Sabine and Eyring estimates, and which file holds what are in Getting started and Where the results go.

Looking at the room

```
acoustic-spaces view examples/shoebox/scene.yaml --paths --max-order 2     # reflection paths
in 3D
acoustic-spaces animate examples/lecture_hall/scene.yaml                   # sound particles
acoustic-spaces export examples/concert_hall/scene.yaml hall.glb --paths   # for Blender
```

These need the `viz` extra. The particles are described under Getting started, and the export under Exporting to Blender.

The editor

`acoustic-spaces` alone opens the graphical editor on a new box room, and `acoustic-spaces edit examples/studio/scene.yaml` on a scene file. Open an example with **File > Open example**, run it with **Simulation > Run** (F5) and pick a source and a listener in the **Results** panel; **Listen** plays a dry recording through the response (with the `measure` extra). The editor works on the same scene files as the command line, so everything below can be done in either.

Listening to the result

A dry (anechoic) recording run through the binaural response is what the seat sounds like on headphones. Dry recordings to try lists free ones; with one saved as `speech.wav`:

```
acoustic-spaces convolve speech.wav results/talker__seat-a.binaural.wav --out wet.wav
```

The binaural output uses a spherical head model unless the scene names a SOFA file; `acoustic-spaces hrtf` lists free measured sets to download (HRTF sets to download).

Building a scene

Starting a scene

A scene is a YAML file: the geometry, the materials, which surface gets which, and where the sources and listeners are (Scene files). Start from a copy of an example, or in the editor from **File > New box room**, **New room from a floor plan** or **New from model file**. A rectangular room needs only its dimensions (`geometry: {box: [8, 6, 3]}`), and the `lecture_hall` example is raised from its floor plan, no CAD file needed (Geometry). The editor's **Plan** tab draws rooms from above (Drawing rooms).

Models from CAD and BIM files

OBJ is the recommended exchange format; DXF, glTF, STL, IFC, Collada and SketchUp files are read too, with their units and up axis (Geometry). Try `rehearsal_room` (a Collada file in inches, as SketchUp exports it) and `bim_office` (an IFC model read as one room, with the `ifc` extra; BIM models). A model that is not closed lets rays escape; `leaky_classroom` has two such mistakes for the leak finder to find (Finding leaks):

```
acoustic-spaces validate examples/leaky_classroom/scene.yaml --leaks
```

Materials

Each surface gets a material: absorption and scattering per band, from the built-in database (`acoustic-spaces materials carpet` searches it), values of your own, or the material libraries of CATT-Acoustic and ODEON, as in the `material_libraries` example. A material can say how far its absorption may be off, which the simulation turns into an uncertainty of every parameter (Materials).

Sources and listeners

A source has a position, an orientation, a directivity (a pattern, a measured balloon or an array of elements), a sound power or a level, and a spectrum; it can also be a line or a surface (Sources and directivity). `acoustic-spaces directivity hypercardioid --plot polar.png` shows a pattern. Try `studio` (stereo loudspeakers at ±30°) and `workshop` (a conveyor as a line source and a machine casing as a surface source). Sources and listeners sit in groups, and any of them can be left out of simulations without being deleted (Groups, and sources left out).

Rooms, walls and doors

A model can hold several rooms, with walls that let part of the sound through, doors and openings: `apartment` hears a television through a wall and a door, closed or open (Walls, doors and openings). Sound that runs low over rows of seats loses more than their absorption says; a material marked `audience: true` adds that loss, as the seats of `town_hall_pa` are (Audience seating).

The air and the seed

The air's temperature, humidity and pressure set the speed of sound and the air's absorption, and `simulation.ir_length_s` the responses' length. The rays are drawn anew on each run unless the scene sets `simulation.seed`, with which the CPU's results repeat from run to run (Air, response length and seed).

Simulating rooms

How a simulation works

Each source and listener pair is simulated in two parts: the early reflections, exact specular paths found by ray tracing or image sources, and the late reverberation, from stochastic ray tracing by default. Their settings are under `simulation` in the scene and in the editor's **Simulation** entry: Simulation, with how surfaces scatter and when to use third-octave bands.

Sound round corners

Sound that bends round an edge is added to the reflections. In `gallery` one listener stands round the corner from the guide, and in `archway` a diner hears a singer over both corners of an archway's jamb in a wall 40 cm thick (Edge diffraction):

```
acoustic-spaces view examples/gallery/scene.yaml --paths --listener "round the corner"
```

Low frequencies

In the lowest bands the early reflections add in pressure, with their phases, so a loudspeaker near a wall gains or loses bass, as `listening_room` shows (Early reflections in pressure). For the modes of a small room a wave solver can take over the lowest bands, on the CPU or a GPU, as in `room_modes` (The wave solver).

How far a result may be off

With the late rays in batches (10 by default) every parameter gets a standard error: how much it would change on a run with another seed. `simulate` also propagates the materials' absorption uncertainty (plus or minus 0.1 for a built-in material unless the scene says otherwise) and the rays' error into each parameter's combined uncertainty and 95 % interval, by Monte Carlo (GUM Supplement 1), for runs in octave bands that meet what Materials lists. It prints the largest of both after the parameters and keeps them all in `parameters.json`; the editor gives them in the results' tips (Simulation, Materials).

Coverage maps

A coverage map gives the parameters, the A-weighted level and the STI on a grid over the audience area, and with `--spread` the uncertainty of each (Coverage maps); in the editor, **Simulation > Run coverage map** (F6).

```
acoustic-spaces map examples/lecture_hall/scene.yaml --plot sti.png --parameter sti
```

Speed and memory

Rays, bands, the wave solver and maps set how long a run takes and how much memory it needs. The late stage and the wave solver can run on a GPU with the `gpu` extra, and `acoustic-spaces tune` finds the fastest device on a computer (Performance, Memory, Tuning the devices for a computer).

Listening in space

Spatial formats and microphones

The scene's `outputs` list picks the formats: mono, binaural (spherical head or a SOFA file), Ambisonics up to the seventh order, and loudspeaker layouts from stereo to 7.1.4 or of your own; `simulate --output` picks them for one run. A listener can also pick up its mono output through a microphone, a first-order pattern or a stereo pair (Spatial outputs, Microphones).

Mixing several sources

`acoustic-spaces auralize` mixes every source's own dry signal as heard at one listener, in any format the simulation wrote (Listening). The `orchestra` example takes this to 61 sources, one for each track of an anechoic recording of a symphony (An orchestra in the concert hall).

Real-time playback and head tracking

`acoustic-spaces play` runs a recording, or the microphone with `--live`, through a response on the sound card in real time (the `measure` extra). With a head tracker and an Ambisonics response, `--head-tracking` keeps the sound in place while the head turns (Head tracking).

Walking through a scene

A listener or a source can follow a walk, drawn in the editor's **Movement** panel, and be heard as it goes. In `gallery` a visitor walks away from the guide and round the corner (Walkthroughs); a fast source, such as a passing car, also drops in pitch (Moving sources and Doppler).

Advanced workflows

Sound systems

Loudspeaker arrays add up box by box, with each feed's gain, equalizer and delay. The examples go from simple to large, each with scripts that print what a system designer checks (Example scenes):

- `town_hall_pa`: two steered columns for speech and two delay loudspeakers; `delays.py` maps the STI with and without the delays.
- `concert_stage`: two line arrays, subwoofers, fills and the stage's monitors; `align.py`, `feedback.py` (gain before feedback) and `near_field.py`.
- `stadium_concert`: a touring system with delay towers and the same monitors, mapped over the pitch and the stands.

Outdoor noise

A scene is outdoors when its geometry has a ground (Outdoor scenes; the editor's **File > New outdoor scene**, outdoor scenes in the editor). The outdoor examples add one effect at a time:

- `street`: buildings and a noise barrier, whose insertion loss `simulate --without noise_barrier` gives (Buildings and barriers).
- `roadside`: the ground's impedance, grass against a paved yard (Ground impedance).
- `tree_belt`: foliage and trunks (Vegetation).
- `hillside`: a terrain from a grid of heights and houses from a map (Terrain, Buildings from a map).
- `open_air_stage`: wind, downwind and upwind, and turbulence (Wind and temperature, Turbulence).
- `lakeside_concert`: weather zones over a lake and a village (Weather zones).
- `railway`, `helicopter_flyover` and `airport`: trains by Schall 03, a helicopter from its rotors and aircraft by ECAC Doc 29 (Passing vehicles, Aircraft by ECAC Doc 29).

What a listener gets outdoors (the level relative to free field, insertion losses, levels over a long term) is in Results outdoors and Levels over a long term; how much traffic and works shake a building, in Vibration through the ground.

Demos

`acoustic-spaces demo my_room/scene.yaml` (or **Simulation > Run demo…**) has each source play its recording, mixes what every listener hears and makes a video that tours them (A demo of any scene). Several examples come with demos of their own, run step by step (Running a demo); each shows one space and one kind of sound, not the simulator as a whole.

Measuring a real room and fitting its model

`acoustic-spaces measure` records sine sweeps through a loudspeaker and a microphone and analyses them; `compare` sets a simulated response beside a measured one in just noticeable differences; `calibrate` fits the materials' absorption to the measurements (Measurements). Without a sound card, the `measurement` example records a sweep in a simulated room, and the `fitting` example makes up measurements of a seminar room to compare and fit.

Long runs on another computer

`acoustic-spaces pack` (**File > Pack for another computer**) puts a scene and every file it names in one zip, which every command that reads a scene takes in its place. A container image runs it on a GPU server over SSH or as a cloud job (Running on another computer).

Scripting in Python

Everything the command line does is in the Python package: `Room.from_yaml` reads a scene, `simulate` runs it, and each response gives its parameters and its impulse responses in every format. The short tour is at the end of Getting started, and every class and function is in the API reference.

`acoustic_spaces.examples.example_path("church")` gives an example's scene file.

How far to trust the results

The validation report compares the results with theory, with pyroomacoustics and with measured rooms (BRAS), with the numbers and the scripts to reproduce them; its known issues say what is still wrong. Read it before relying on a result for a decision.

Reference

- Getting started: installing, the command line and the Python API, where the results go, the language.
- Graphical editor: editing scenes, drawing rooms, outdoor scenes, export to Blender.
- Example scenes: every bundled scene and what it shows.
- Scene files: geometry, materials, walls and openings, seating, sources, walks, air.
- Simulation: the methods and their settings, diffraction, low frequencies, performance.
- Spatial audio and listening: output formats, convolution, real-time playback, walkthroughs, Doppler, head tracking.
- Coverage maps: parameters and the STI over the audience.
- Outdoor scenes: ground, terrain, buildings, weather, vehicles, aircraft, vibration.
- Demos: every demo, its recordings and options.
- Measurements: measuring, comparing and fitting.
- Running on another computer: packs, the container image, servers and cloud jobs.
- Validation report and API reference.
- Roadmap, development, translating and the changelog.

Getting started

Install

```
pip install acoustic-spaces                # simulation, rendering and the command line
pip install "acoustic-spaces[viz,gui]"     # with the 3D view and the graphical editor
pip install "acoustic-spaces[viz,gui,measure,sofa,gpu,ifc,sketchup]"  # every optional part
```

PyPI has wheels with the compiled core inside for Linux (x86-64), macOS (Apple silicon and Intel) and Windows (x86-64), CPython 3.10 to 3.14; elsewhere pip builds the package from source, which needs a C++17 compiler. Every GitHub release has the same wheels, to `pip install` with the extras after the file name, such as `pip install "acoustic_spaces-4.2.1-cp312-cp312-win_amd64.whl[viz,gui]"`. Between releases, install from source.

Extras

The extras in brackets add the optional parts:

Extra	Adds	For
`viz`	matplotlib, PyVista, imageio, imageio-ffmpeg	plots of responses and coverage maps (`--plot`), the 3D view (`view`, `--screenshot`), sound particles and walks as GIF or MP4 videos
`gui`	PySide6, pyvistaqt, PyVista, matplotlib	the graphical editor
`measure`	sounddevice	the sound card: measuring a room with sweeps (`measure`, `devices`) and real-time playback (`play`, the editor's Listen)
`sofa`	h5py	HRTFs and source directivities from SOFA files
`gpu`	wgpu	the late stage's rays, the wave solver and `tune` on a GPU, through Vulkan, Metal or Direct3D 12
`ifc`	IfcOpenShell, mapbox-earcut	BIM models: IFC files
`sketchup`	openskp	SketchUp files (experimental)
`test`	pytest, h5py	running the tests
`dev`	pytest, h5py, ruff, mypy, types-PyYAML	the tests, the linter and the type checks (CONTRIBUTING.md)
`docs`	Sphinx, Furo, MyST-Parser, pypdf	building this documentation and its PDF

A command or function that needs an extra that is not installed says which one to install.

The documentation (these pages, the validation report and an API reference generated from the docstrings) builds with Sphinx; see CONTRIBUTING.md.

Install from source

Requirements: Python 3.10+, a C++17 compiler and CMake 3.18+.

```
git clone https://github.com/naelolaiz/acoustic-spaces
cd acoustic-spaces
```

With uv, `uv sync` builds the package into `.venv` together with the `dev` tools (pytest, h5py, ruff, mypy), and `--extra` adds the extras:

```
uv sync --extra viz --extra gui        # the 3D view and the editor
uv sync --all-extras                   # every extra in the table, docs included
```

`uv sync` installs exactly what it is asked for, so name every extra each time. It rebuilds the compiled core when the C++ sources change.

With pip, in a virtual environment, the extras go in brackets after the `.`:

```
python -m pip install -e ".[dev,viz,gui]"   # the development tools, the 3D view and the
editor
```

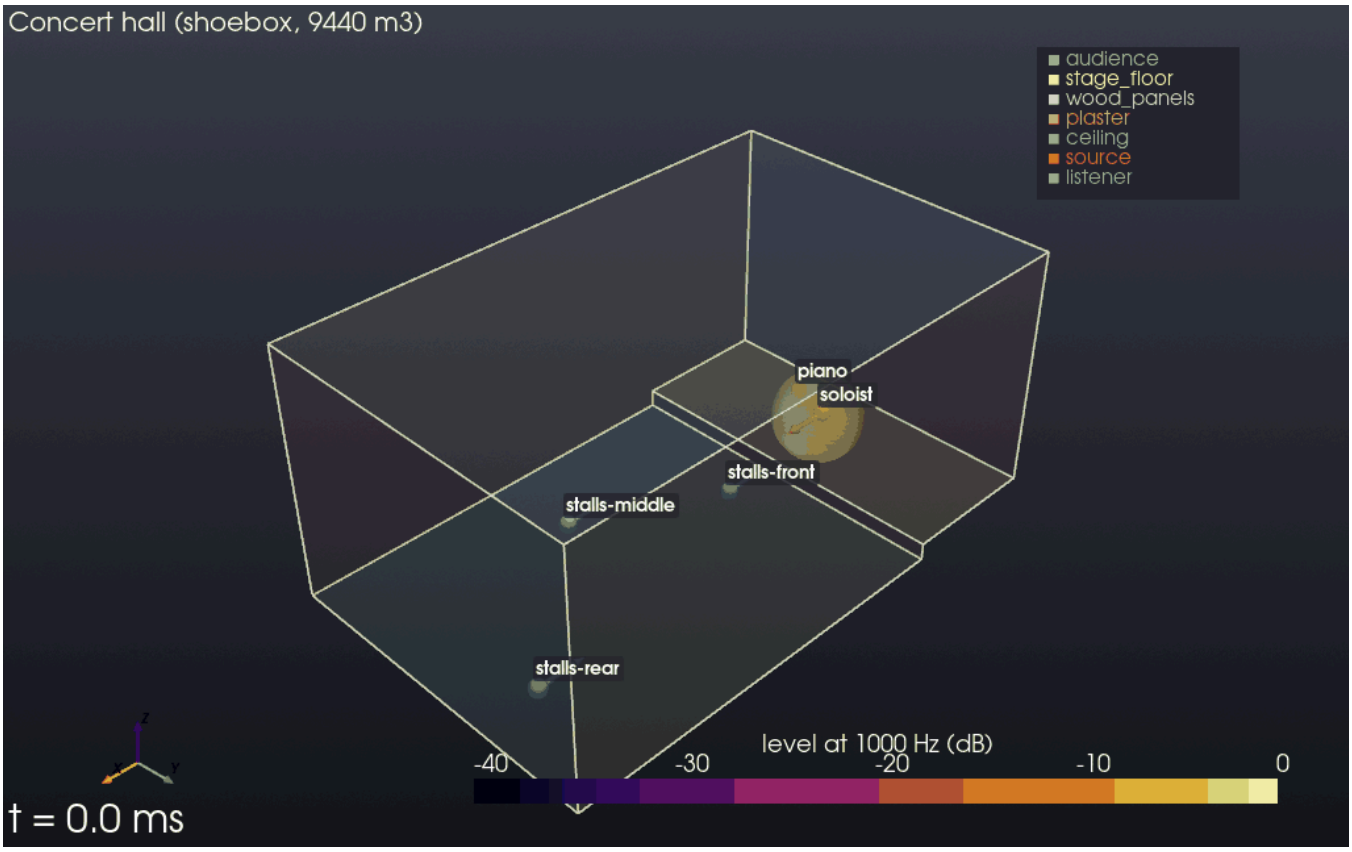
Check the installation and look at the example scene. The commands run in the environment you installed into: activate it (`source .venv/bin/activate`, or `.venv\Scripts\activate` on Windows), or with uv put `uv run` in front of each.

```
acoustic-spaces info
acoustic-spaces validate examples/shoebox/scene.yaml   # errors exit with status 1
acoustic-spaces validate examples/leaky_classroom/scene.yaml --strict  # warnings too
acoustic-spaces validate examples/leaky_classroom/scene.yaml --leaks  # where rays escape
acoustic-spaces inspect examples/shoebox/scene.yaml    # geometry, materials, RT estimates
acoustic-spaces view examples/shoebox/scene.yaml       # 3D window; --screenshot out.png
acoustic-spaces view examples/shoebox/scene.yaml --paths --max-order 2  # reflection paths
acoustic-spaces view examples/gallery/scene.yaml --paths --listener "round the corner"  #
diffraction
acoustic-spaces animate examples/lecture_hall/scene.yaml               # sound particles
acoustic-spaces export examples/concert_hall/scene.yaml hall.glb --paths --particles  #
Blender
acoustic-spaces materials carpet                       # search the material database
acoustic-spaces directivity hypercardioid --plot polar.png  # a source directivity
acoustic-spaces simulate examples/shoebox/scene.yaml --out results --plot
acoustic-spaces map examples/lecture_hall/scene.yaml --plot sti.png --parameter sti
acoustic-spaces walkthrough examples/gallery/scene.yaml guide=speech.wav --out walk.wav  # a
visitor walks
acoustic-spaces demo my_room/scene.yaml                # each source plays its signal: a
video of every listener
acoustic-spaces examples                               # the bundled rooms, and where they
are
acoustic-spaces hrtf                                   # free HRTF sets to download, with
their licences
acoustic-spaces pack examples/hillside/scene.yaml hillside.zip  # one zip to run elsewhere
acoustic-spaces tune --default auto                    # with the gpu extra: run on the
fastest device here
python -m pytest                                       # the test suite
```

`pack` puts a scene and every file it names in one zip, which every command that reads a scene takes in its place, to run on a server or a cloud job: see Running on another computer.

Errors start with `error:` and warnings with `warning:`, in red and yellow in a terminal (`NO_COLOR=1` turns colours off).

`animate` shows sound particles leaving a source and spreading through the room, each reflecting off the walls (turned from the specular direction by as much as the surface scatters; see Scattering) and losing the absorbed share of its energy at every hit, coloured by level or by reflection order. It opens a window with a time slider, or saves a GIF (`--out particles.gif`) or an MP4 video (`--out particles.mp4`).

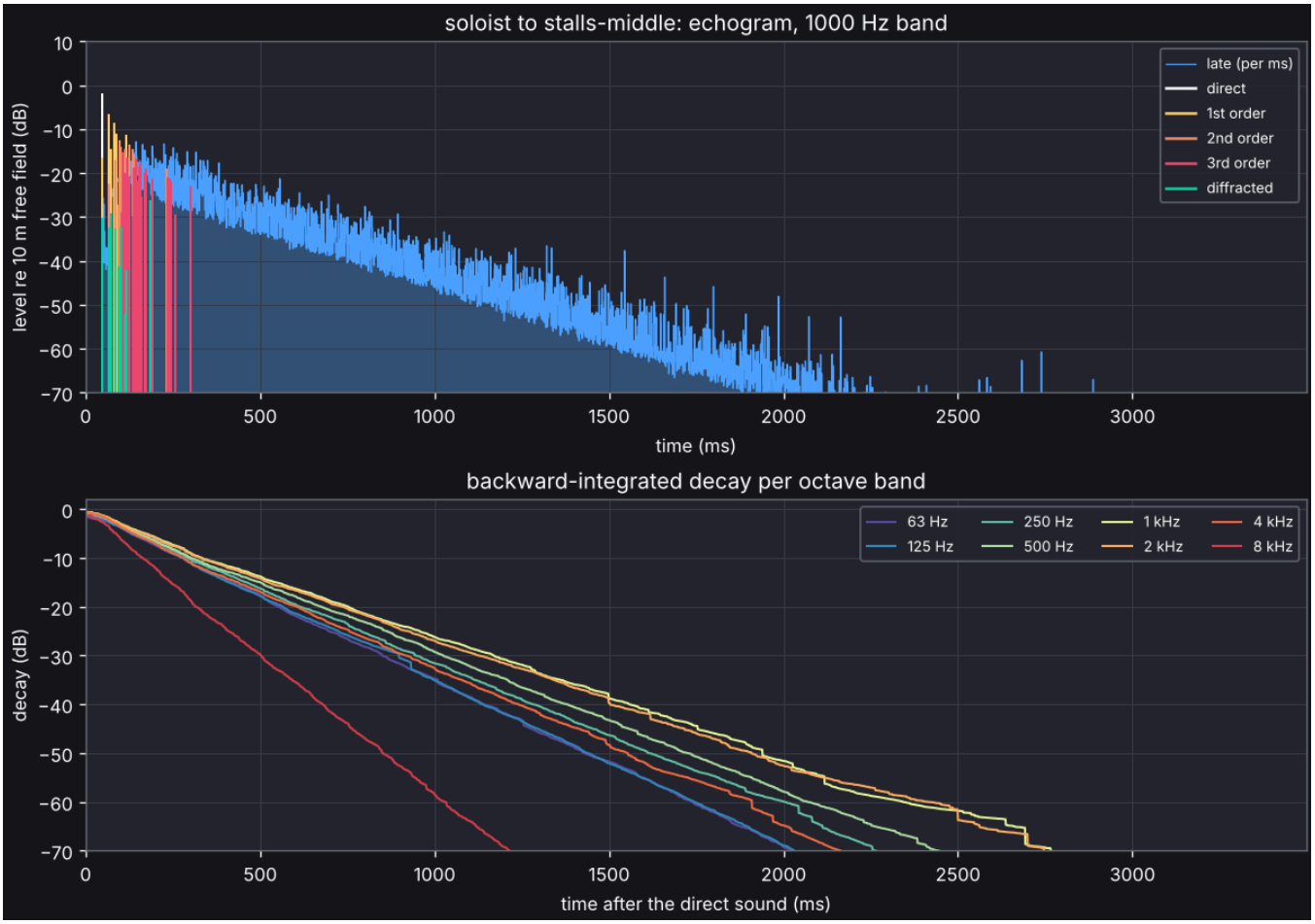


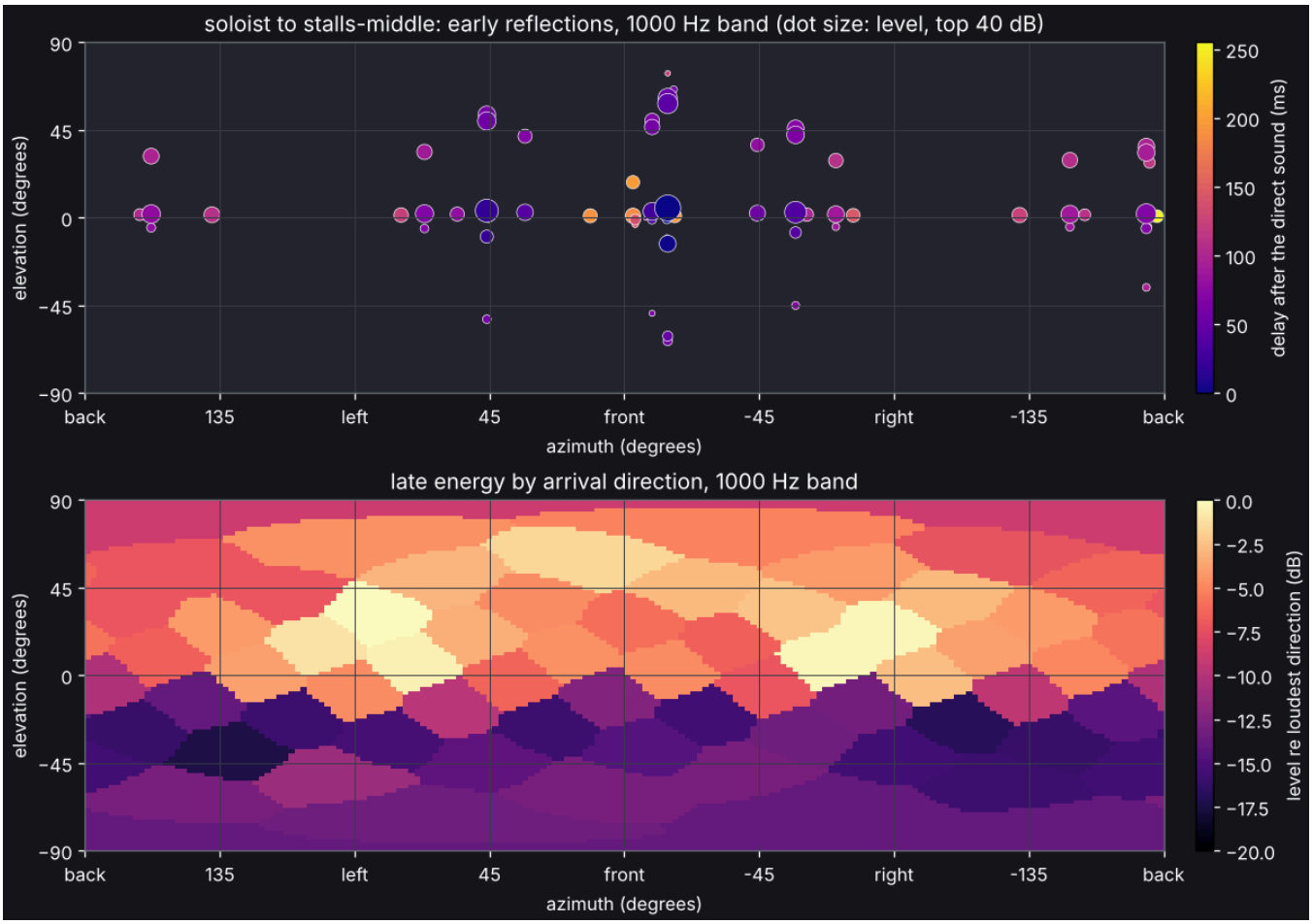
`simulate` writes impulse responses for every source and listener in each format the scene's `outputs` list (`results/talker__seat-a.wav` in mono, `.binaural.wav`, `.ambix3.wav`, `.5.0.wav` and so on, 32-bit float, scaled by one gain to a -1 dBFS peak with `--normalize`, `ir_gain` in `parameters.json`; names that would give one file name, as `A/B` and `A_B` do, get `-2`, `-3`… after the later ones), `parameters.json` with the room acoustic parameters of every response per octave band, its STI and its echo criteria, and with `--plot` an echogram, decay curves and a map of arrival directions as PNG. The files go into the folder once all of them are written, `parameters.json` last, so a run that cannot write them all (a full disk, say) leaves the folder as it was; the editor's Export does the same. A file where the folder should be stops `simulate` before its run. Files of an earlier run whose names this one does not write stay in the folder. `--output ambisonics:3` (repeatable) picks formats on the command line. It prints the parameters as it goes:

```
talker -> seat-a: direct sound at 11.7 ms, 63 early paths
                63 Hz  125 Hz  250 Hz  500 Hz   1 kHz   2 kHz   4 kHz   8 kHz
EDT (s)         0.680   0.519   0.703   0.764   0.921   1.004   0.942   0.605
T20 (s)         0.712   0.781   0.876   1.173   1.353   1.369   1.169   0.764
T30 (s)         0.708   0.841   0.925   1.292   1.378   1.359   1.174   0.778
C50 (dB)          2.7     5.8     2.8     3.1     3.1     2.7     3.0     5.3
C80 (dB)          6.1     8.7     6.1     6.1     5.7     5.3     5.6     8.6
D50             0.652   0.793   0.655   0.670   0.669   0.650   0.664   0.771
Ts (s)          0.049   0.037   0.050   0.052   0.056   0.058   0.054   0.036
G (dB)           22.0    24.1    20.6    19.8    18.5    17.8    17.0    15.6
LF              0.170   0.093   0.300   0.304   0.330   0.343   0.345   0.328
LFC             0.254   0.138   0.433   0.432   0.446   0.454   0.451   0.430
LJ (dB)          10.4    10.7     9.5     9.4     8.9     8.6     7.4     3.7
IACC_E          0.998   0.989   0.817   0.586   0.495   0.739   0.604   0.611
IACC_L          0.973   0.947   0.577   0.183   0.298   0.122   0.078   0.067
T30_X (s)       0.707   0.787   0.927   1.265   1.363   1.341   1.146   0.775
T30_Y (s)       0.708   0.952   0.940   1.341   1.398   1.372   1.209   0.784
T30_Z (s)       0.707   0.821   0.895   1.083   1.279   1.265   1.154   0.753
ξ_T20 (‰)         0.0     0.9     2.7     5.5     3.8     1.8     0.6     0.9
ξ_T30 (‰)         0.0     2.9     2.2     6.5     2.1     1.4     0.8     0.5
Curvature (%)    -0.6     7.7     5.7    10.1     1.9    -0.8     0.4     1.7
standard errors from 10 batches of rays: at most 1.58 JND from 125 Hz to 4 kHz (T30 at 500
Hz); more rays (simulation.late.rays) would change the parameters less from seed to seed
uncertainty from the absorption and the rays (10 batches), 400 draws: at most 5.23 JND from
125 Hz to 4 kHz (T20 at 1 kHz: u = 0.354 s (absorption 0.34, rays 0.097), 95 % within 0.678
to 1.71 s, 95 % at least 0.707 s and 95 % at most 1.67 s); built-in materials of unknown
source taken at plus or minus 0.1 (simulation.uncertainty.presets): painted_concrete,
carpet_on_concrete, acoustic_ceiling
SPL 60.0 dB(A) (u = 0.7, 95 % within 58.4 to 61.0, 95 % at most 60.8); per octave band, dB:
52.8 66.9 63.4 58.9 51.6 44.9 38.1 30.7
STI 0.66 (u = 0.041, 95 % within 0.63 to 0.74, 95 % at least 0.63), STIPA 0.66, echo
criterion 0.65 for speech (echo above 1.0), 0.69 for music (echo above 1.8)
```

Unless the scene sets `simulation.seed`, its rays and the noise of its rendered late part are drawn anew on each run, so these numbers move from run to run: IACC most, as it is read from one rendered response, and the others about as much as the standard errors say.

With `--plot`, each response gets an echogram (the late energy per millisecond and the early paths by order) with the decay curves of every band, and a map of where the sound comes from:


soloist to stalls-middle: echogram, 1000 Hz band
level re 10 m free field (dB)
time (ms)
late (per ms)
direct
1st order
2nd order
3rd order
diffracted
backward-integrated decay per octave band
decay (dB)
time after the direct sound (ms)
63 Hz
125 Hz
250 Hz
500 Hz
1 kHz
2 kHz
4 kHz
8 kHz


soloist to stalls-middle: early reflections, 1000 Hz band (dot size: level, top 40 dB)
elevation (degrees)
azimuth (degrees)
delay after the direct sound (ms)
late energy by arrival direction, 1000 Hz band
level re loudest direction (dB)
back
left
front
right

From 500 Hz up, T30 comes out two to more than four times the Sabine and Eyring estimates that `inspect` prints, and that is the room, not a fault. Its absorption sits on the ceiling and the carpet while the walls are hard and smooth (scattering 0.05), so sound running between the walls meets little absorption and dies away slowly. Sabine and Eyring assume a diffuse field, where sound meets every surface alike; `inspect` says when a room is far from that and adds the Arau-Puchades estimate, which allows for absorption that faces one way. With every surface fully diffusing, the same room decays in Eyring's 0.37 s at 1 kHz; with every surface scattering 0.3, in about 0.53 s. The validation report checks both kinds of room.

From Python:

```
from acoustic_spaces import Room, simulate
from acoustic_spaces.audio import write_wav
from acoustic_spaces.engine import early_paths, trace_particles
from acoustic_spaces.spatial import Ambisonics, Binaural, Loudspeakers
from acoustic_spaces.viz import animate_particles, plot_directions, plot_response, show_room

room = Room.from_yaml("examples/shoebox/scene.yaml")
print(room.report.summary())
print(room.problems(), room.warnings())
print(room.reverberation_time("eyring"))  # per octave band, seconds
show_room(room, paths=early_paths(room, max_order=2))
animate_particles(room, trace_particles(room, duration=0.1), "particles.gif")

result = simulate(room)
response = result["talker", "seat-a"]
print(response.parameters().table())
write_wav("talker__seat-a.wav", response.impulse_response(48_000), 48_000)
plot_response(response, "talker__seat-a.png")

binaural = response.spatial_impulse_response(Binaural(), 48_000)  # (samples, 2)
ambix = response.spatial_impulse_response(Ambisonics(order=3), 48_000)  # 16 channels
surround = response.spatial_impulse_response(Loudspeakers.layout("5.0"), 48_000)
plot_directions(response, "talker__seat-a.directions.png")
```

Where the results go

A simulation gives one response for each source and listener in use, all side by side: none of them is *the* result until a mix or the editor's results panel picks a listener. Sources and listeners left out of simulations (`enabled: false`) get none.

- `acoustic-spaces simulate scene.yaml --out DIR` (`--out` has no default) writes every pair into `DIR`, as above: `SOURCE__LISTENER.wav` for the mono output, `SOURCE__LISTENER.binaural.wav`, `.ambix3.wav`, `.5.0.wav`… for the others, and `parameters.json` with the parameters of every pair. The stadium concert, with 19 sources and 12 listeners in use and a mono output, gives 228 WAV files; `ls DIR/*__mix_position.wav` lists one listener's. A listener with a microphone (`pattern`) has its mono files picked up through it, which `parameters.json` names (`microphone`).
- The editor's **Simulation > Run** (F5) keeps the results in memory: the results panel shows one source and one listener at a time, picked above it or by selecting them in the scene. **Export…** there, or **File > Export results…**, writes them all into a folder you pick, as `simulate` does.
- `acoustic-spaces auralize DIR SOURCE=WAV... --out mix.wav` mixes the sources as heard at one listener, which `--listener` names when the results have more than one; `acoustic-spaces walkthrough --all --out walk.wav` writes one file per listener, `walk_LISTENER.wav`.
- `acoustic-spaces demo scene.yaml` (*This scene* in the editor's **Simulation > Run demo…**) writes into `demo/` beside the scene (`--work` elsewhere): the simulation in `results/`, one mix per listener in `seats/` (`moving/` when something walks), named `LISTENER.binaural.wav`, and `tour.wav`, the stills in `stills/`, `thumbnail.png` and `tour.mp4`, which visits every listener in use in the scene's order (`--listeners` picks them). An example's own demo writes its scene and simulation beside its script (the stadium's `scene_band.yaml` and `results_band/`) and the rest into `demo/` there (Demos).
- The editor runs a demo in the open scene's folder, where the command line runs it, but for the examples of an installed copy (opened from **File > Open example**) and a folder it cannot write: those run in a folder of the same name in `~/acoustic-spaces/` (*This scene* in its `demo/`), so that nothing goes among the package's files, which an upgrade replaces. The **Run demo** window says where (*Runs in …*).

Language

The editor and the command line show their text in English, Spanish, Catalan, Italian, German or Dutch, following the computer's language unless told otherwise:

- **View > Language** in the editor picks one for this user, or **System** to follow the computer again. It applies from the next start of the editor and of the command line.
- `ACOUSTIC_SPACES_LANG=nl` (or `de`, `it`, `ca`, `es` or `en`) in the environment sets it over both, for one shell or one script. In the editor, choose **Nederlands** for Dutch, **Deutsch** for German, **Italiano** for Italian or **Català** for Catalan.

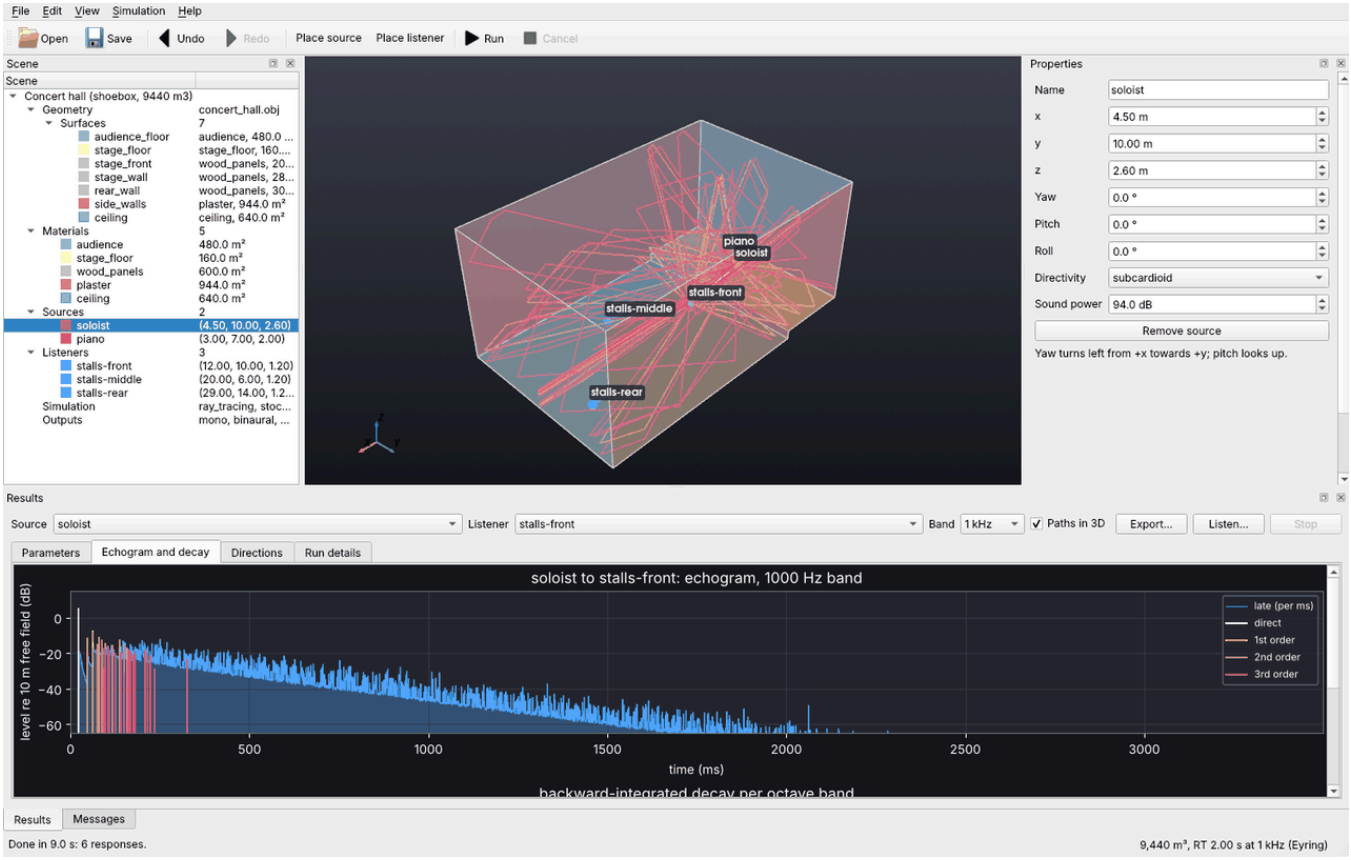
`acoustic-spaces info` says which language is in use and what chose it. Scene files, command and option names and the files the commands write stay in English whatever the language, so scripts keep working. The editor writes numbers as the computer's regional settings do (0,85 in Spain, 0.85 in Mexico) and reads them typed either way; the command line keeps the decimal point. The Spanish, Catalan, Italian, German and Dutch translations come in steps: the editor is translated, with its plots and the examples' descriptions; the command line and the library's messages (which the editor shows too, such as the room's checks) come next (see the roadmap); text not yet translated shows in English. Translating explains how the catalogs work and the terms the translations use.

Graphical editor

The editor needs the `gui` extra (Extras): `pip install "acoustic-spaces[gui]"`, or `uv sync --extra gui` in a source checkout and then `uv run acoustic-spaces`.

```
acoustic-spaces                                  # alone, it opens the editor on a new box
room
acoustic-spaces edit examples/studio/scene.yaml  # or open a scene file
```

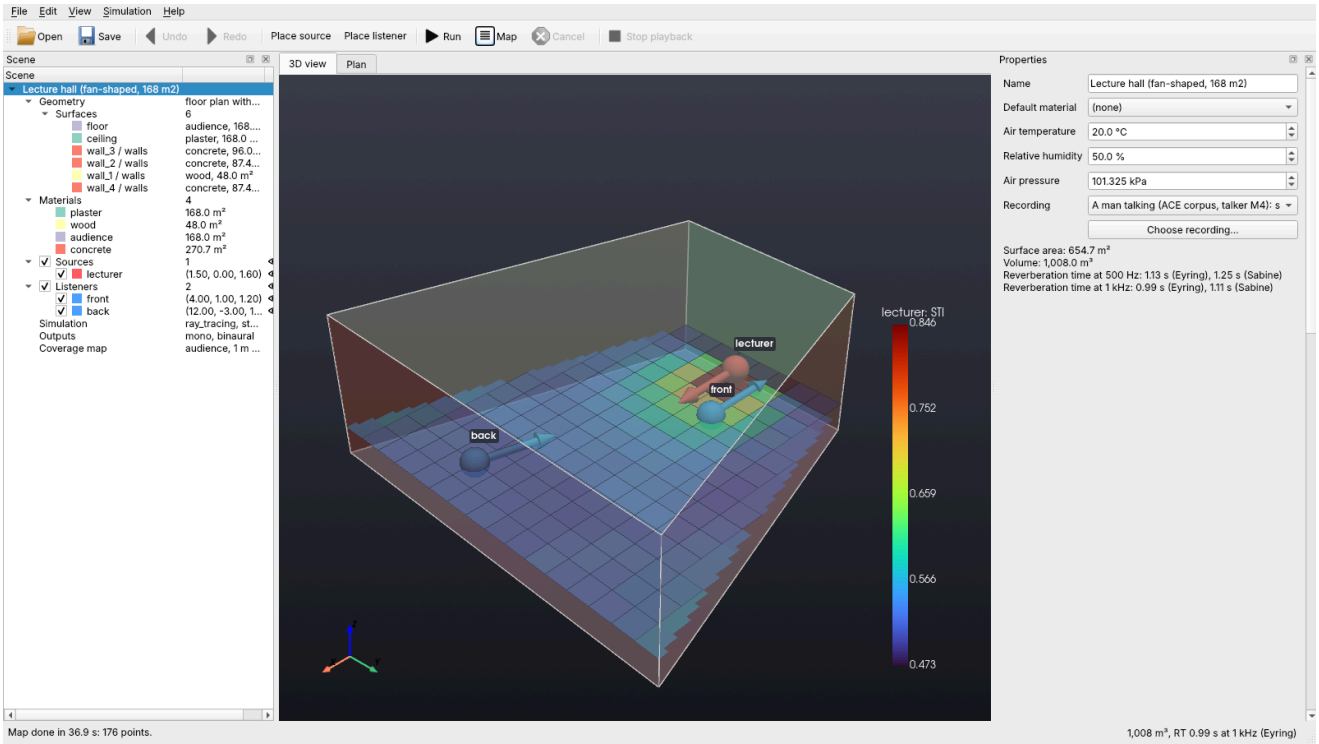
`acoustic-spaces` with no command opens the editor; `acoustic-spaces -h` lists the commands. Without the `gui` extra, or with no display to show the editor on (over SSH, say), it prints the short usage instead, with the reason.



The editor works on the same scene files as the command line.

- **File > Open example** opens the bundled rooms, in the groups of Example scenes. Saving one asks where to keep your copy and takes along the files it uses from the example's folder (its model, terrain, map of buildings, material files, directivities with the patterns their array files name, the sources' signals, its recording and HRTFs); an example whose model needs an extra that is not installed is greyed out, and its tip says how to install it. **New box room**, **New room from a floor plan**, **New from model file** (OBJ, STL, PLY, OFF, glTF, DXF, IFC, Collada or SketchUp) and **New outdoor scene** (a meadow 200 m across, to build on; see Outdoor scenes) start from scratch. For an IFC model, the geometry's form picks one room of the building (**IFC room**) or every element.
- The **Plan** tab, next to the 3D view, shows the room from above and draws rooms (see below).
- The **scene tree** lists the geometry and its surfaces, the materials, sources, listeners, the wind outdoors, simulation settings and output formats. Selecting an entry, there or by clicking it in the 3D view, shows its form under **Properties**. Every change is checked against the scene schema; a refused change says why under **Messages**, and every change can be undone.
- Sources and listeners sit in their **groups** (the form's **Group** field; groups nest with `/`, as PA/Delays). Each source, listener and group has a tick: unticked, it is left out of simulations, maps and checks, and is saved as `enabled: false` (the form's **Use in simulations** does the same). The eye beside it hides it in the 3D view and the plan, or shows it again, without changing the scene. What a scene leaves out starts hidden (the stadium's helicopter, band and singer's microphone), and an eye follows its tick when that changes, by a click, undo or redo; shown again by its eye, an item left out is drawn faded. See Groups, and sources left out.
- **Place source** and **Place listener**, then click a surface: on a floor the new point stands at ear or mouth height, on a wall or ceiling it goes half a metre into the room. In the plan, the click places it above the floor there. Sources and listeners move by dragging, in the 3D view or the plan, and keep their height above the floor (up a raked floor too); with Shift in the 3D view they move up and down. Surfaces take materials from their form, or from a new material picked from the database (**Edit > Add material from the database**) or from a CATT-Acoustic or ODEON library (**Edit > Add material from a library**).
- A material's form sets its absorption and scattering per band (24 columns when it has third-octave data), whether it **Lets through** nothing, part of the sound (with its sound reduction index per band in the table) or everything (an opening), **Audience seating** for the seats (see Walls, doors and openings), and under **Absorption uncertainty** how far its absorption may be off, for the spread of the results (see Materials).
- **Edit > Edit movement…**, or **Edit movement…** in a source's or listener's form, opens the **Movement** panel on the selected point source's or listener's walk: a table of keyframes, where it is and how it faces at each time, or, for a walk through points at one speed, a row per point and the speed. Drag a keyframe's or a point's handle in the plan or the 3D view, or type its values; every change can be undone. The slider and **Play** move every mover along its walk, nothing simulated. **Moves** keeps a walk but leaves its mover standing, and **Doppler** has it play as a passing vehicle does (`doppler: true`; it then moves alone). A line or a surface source does not move. **Render movement…** simulates every mover on one clock and writes what each listener hears as a WAV file, as `walkthrough --all` does (see Walkthroughs).
- The scene tree's **Simulation** entry has the settings of Simulation. The scene's own entry sets the air's temperature, humidity and pressure, and its **Recording** is what the sources with a **Channel** play in a demo of the scene: one of the free recordings or a file of yours.

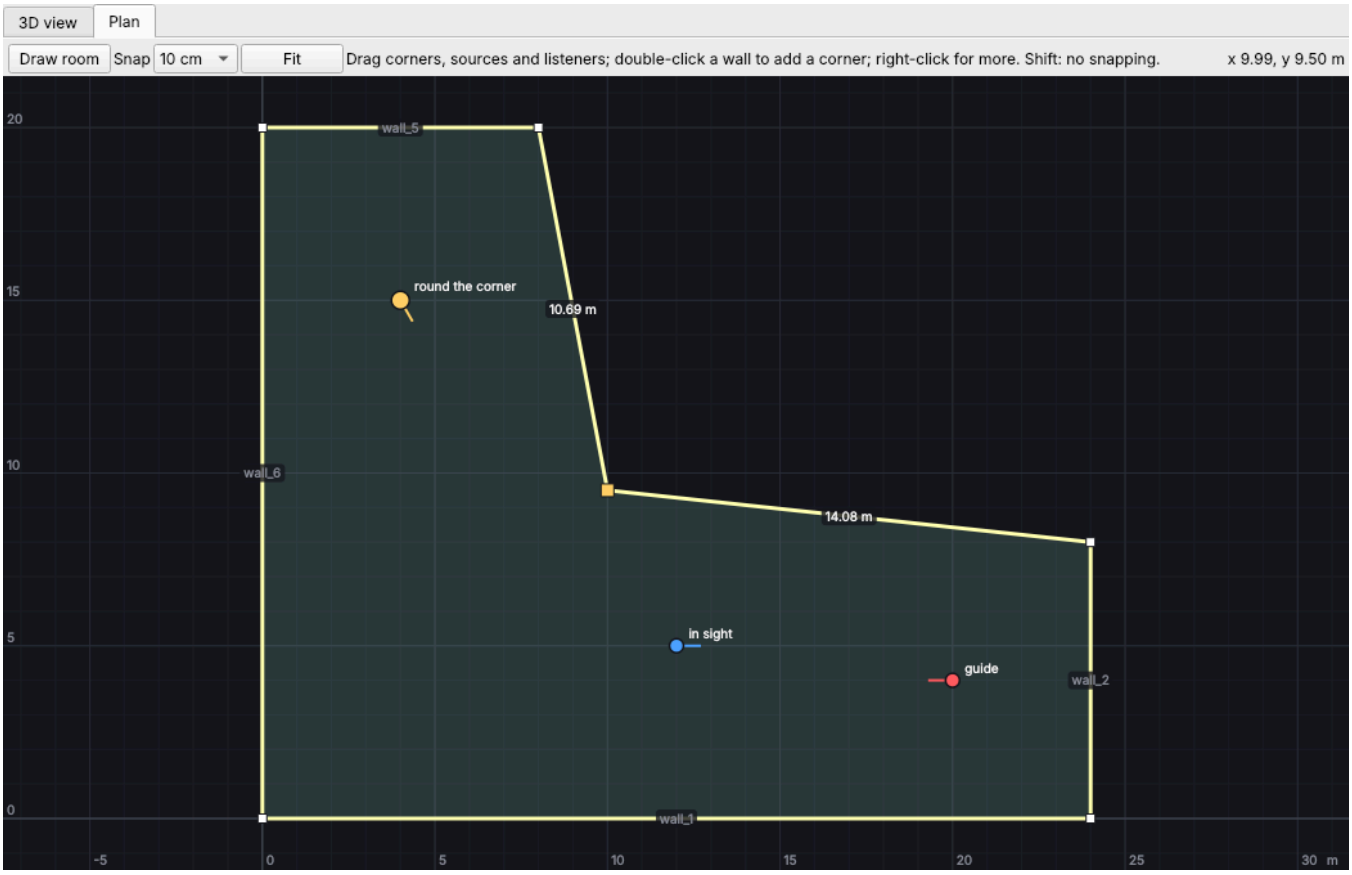
- **Simulation > Tune CPU and GPU…** times the CPU and the GPU on the open scene (or on the concert hall example when the scene cannot run as it is), as `acoustic-spaces tune` does (see Tuning the devices for a computer; needs the `gpu` extra). It runs in the background, once per computer, and then shows the table of times with a check box, **Make auto the default on this computer**: scenes whose **Run on** is *default* then run on the fastest device found, in the editor and on the command line.
- **Simulation > Run demo…** runs a demo of the open scene in the background, with what it prints under **Messages** and **Cancel** to stop it: *This scene* (a demo of any scene), and for an example's scene its own demos and scripts. See Running a demo for the window.
- The scene tree's **Outputs** entry chooses what a run writes: mono, binaural, Ambisonics or loudspeakers. For binaural, **HRTF file…** takes a SOFA file from the disk, **Spherical head** goes back to the model, and **Download HRTFs…** offers the free sets with their licences and fetches the one chosen in the background (see HRTF sets to download).
- **Simulation > Run coverage map** (F6) maps every source over the audience area (see Coverage maps). The **Coverage map** panel shows any parameter and band from above, for one source or all of them together, and **Map in 3D** lays it over the room, see-through so that the sources and listeners stay in sight; **Save** writes the picture or every map as JSON. The scene tree's **Coverage map** entry sets the surfaces, height, spacing, rays and background noise, and with the box under **Uncertainty** ticked the map also gives its values' uncertainty, which the panel's **Show** draws.



- **Simulation > Run** (F5) works in the background. The **Results** panel shows the ISO 3382-1 parameters (outdoors also the level relative to free field, the insertion loss and the level over a long term, see Results outdoors and Levels over a long term), the echogram and decay curves and the arrival directions of any source and listener, and draws their reflection paths in the room. With the late rays in batches, each parameter's tip gives its standard error, how much it would change between runs with other seeds (Simulation), and the value is amber where that is over half a JND: more rays per band would make it smaller. **Export** writes the same files as `simulate`, in the background (**Cancel** stops it); **Listen** plays a dry WAV file through the response on the sound card, binaurally (through the HRTF of the scene's binaural output if it has one, else a spherical head; needs the `measure` extra), or head-tracked with **Head tracking** ticked (see Head tracking). While it plays, the status bar says so, with **Stop playback** (also in the toolbar and the Simulation menu, or Esc). Picking another source or listener (in the scene or the panel), or ticking **Head tracking**, plays on from the same point with that response, at the same level so the difference stays audible. What plays does not follow changes to the scene (moving a source, say): the status bar marks it as from before them, and Listen again plays them.
- Change a material's absorption after a run and the results follow at once: the run counted each ray's reflections off each material, and is re-weighted for the new values instead of traced again (as `calibrate` does). Where the early reflections add in pressure (up to 125 Hz indoors by default, every band outdoors), their phases turn with the absorption too, as a new run has them; only the faces of an edge that sound bends round keep what they did to it as traced. The panel says so; **Run** traces afresh. Other changes, and runs with the statistical method, third-octave bands or the wave solver, leave the results marked out of date until the next run. So does another absorption of an outdoor ground without an impedance where the solver for whole rows carries a row's sound, which depends on it.
- A source's form sets its **Shape** (a point, a line or a surface, with its length, height and spacing), its directivity (a pattern, a SOFA or CSV file of measured data, or an array file, whose elements **Add up the array's elements one by one** sums near its listeners), its sound power per band, its **Spectrum** (flat, or male or female speech) and the **Gain**, **Equalizer** and **Delay** of its feed. It shows the directivity index per band, opens the polar patterns, and the selected source shows its balloon in the 3D view. **Plays** says what the source plays in a demo of the scene: **Choose WAV…** picks a dry recording, **Silent** takes it away, and without one it plays its **Channel** (left, right or mono) of the scene's **Recording**. The channel of a file of its own is picked the same way; *(none)* mixes it down.
- A listener's form sets its **Microphone**, what its mono output picks up through, aimed where it faces: omni, a first-order pattern, or an XY pair (with its **Angle** and **Capsules**), a Blumlein pair or an MS pair (with its **Mid** capsule and its **Side**'s gain), which write two channels (microphones). A pattern per band, set in the scene file, shows as *per band: …* in the list (and in **Capsules** or **Mid** for a pair's) and stays until another is picked. The selected listener shows its microphone's balloon in the 3D view (at 1 kHz), one for each capsule of a pair.
- The status bar keeps the volume and the Eyring reverberation time at 1 kHz up to date as you edit, with the number of problems in red.
- **Messages** shows warnings in amber and errors in red; so does the scene's own form for its problems and warnings. Its **Item** list keeps only the messages about one source, listener or material, and with them every error and the line that ends a run (done, cancelled, a demo's).

- **View > Scene YAML** shows the scene as YAML text that follows every edit. Edited text is applied with **Apply** (Ctrl+Return), checked as every other change is, in one step that **Undo** takes back; text the checks refuse stays as typed, with the reason. The text is made from the scene, not read from its file, so the file's comments do not show (Save does not keep them either). A material's absorption, scattering and source show only where its preset, file or other settings do not give them, so another preset or file written there brings its own.
- Runs, coverage maps, tunings and exports go on in the background, with **Cancel** in the status bar.
- **View > Follow the run** (on by default) rings, in the 3D view and the plan, the sources and listeners that the latest step of a run (a demo's too) names, hidden ones included. While a walking source is simulated place by place (a pass-by demo's vehicle, **Render movement…**), every walker of the run is drawn where it is at that place's time, on the walks the run simulates: a demo's own scene when it has one, the vehicle at `--speed`, say. When only listeners walk, each source is traced to all their places at once, so the views do not move them. While a demo encodes its video, the views show what the video shows at that moment: the listener of each pass of the tour in turn, the sources that play, and the walkers where the video has them. A head a demo turns (the gallery walk's look round) faces as it does then, as it is rendered and in the video; arrows and directivity balloons turn with it.
- **View > Show leaks** traces rays from the sources and shows where they escape from a model that is not closed: its open edges, the escaping rays and a marker on each gap, with the report in the messages (see Finding leaks). It traces again when the geometry, the materials or the sources change.
- **View > Show wind** draws the wind of an outdoor scene in the 3D view and the plan, with the weather zones and their winds, and **View > Show wind profile** stands a mast with the wind at every height in each weather (see Wind and temperature).
- **View > Language** shows the editor and the command line in English, Spanish, Catalan, Italian, German or Dutch, or in the computer's language, from their next start (see Language). Numbers in the editor follow the computer's regional settings: 0,85 where the decimal mark is a comma, with lists of numbers (an equalizer's gains) apart by semicolons there.
- **File > Pack for another computer** writes the scene and every file it names to one zip file, as `acoustic-spaces pack` does, to run on a server or a cloud job (see Running on another computer).
- **File > Export to Blender** writes the room with the reflection paths on show, and **File > Export particle animation to Blender** the sound particles leaving the selected source, asking first how much of the sound to animate, how slowly and how many particles (see below).

Drawing rooms



The **Plan** tab shows the room from above on a grid in metres. In a floor plan or box room every corner is a handle:

- Drag a corner to move it; the two walls it joins show their lengths as it moves. A box becomes a floor plan with the same walls, so its materials stay where they were.
- Double-click a wall to add a corner there, or right-click a corner to remove it. Clicking a wall selects that surface, to give it a material.
- **Draw room** (or **Edit > Draw floor plan**) draws a new floor plan: click its corners in turn and close it on the first one, with a double-click or with Enter. Backspace takes the last corner back and Esc stops. The new room keeps the old one's height (change it under **Geometry**). Sources and listeners that would end up outside move inside, keeping their arrangement. Over a model file, the editor asks before replacing it.
- Corners and points snap to the grid chosen under **Snap** (10 cm unless changed); hold Shift to place them freely. A move that would make walls cross is refused, with the reason under **Messages**.
- The wheel zooms, dragging empty space pans and **Fit** shows the whole room again, with every source and listener and the places their walks go through.

Rooms from model files show their outline, and their sources and listeners move the same way. Every change can be undone.

Outdoor scenes

File > New outdoor scene starts an outdoor scene: a flat meadow 200 m across under an open sky, with a source and a listener 50 m apart. The **Geometry** entry of an outdoor scene has an **Outdoors** group instead of the room's: the ground's height, the margin, the sky and the **Extent** (tick it to set where the ground ends; unticked, the ground reaches the margin past what stands on it), then the buildings, barriers, vegetation and ground zones in tables, with their heights, which can be changed, and buttons to add and remove them:

- **Add building**, **Add zone** and **Add vegetation** with *belt of trees* chosen draw an outline in the plan, like a floor plan: click its corners in turn and close it on the first one, with a double-click or Enter. **Add barrier** and **Add vegetation** with *hedge* chosen draw a path: click its points and finish with a double-click or Enter. A *lone tree* stands where the next click lands. Backspace takes a point back and Esc stops. A new building is 10 m high, a barrier 3 m, a belt of trees 12 m, a hedge 2 m high and 1.5 m wide, a tree 10 m high with a crown 8 m across; change the heights in the tables and the rest in the scene file. A zone takes the ground's material until its surface is given one in the scene tree.
- **Terrain from a file** makes the ground follow the heights in an Esri ASCII grid or a table of x y z points (see Terrain); the form then describes it (its points, heights and grid), and **Remove terrain** flattens the ground again.
- **Buildings from a map** adds the buildings of a GeoJSON file of footprints (see Buildings from a map), after asking which of its properties holds their heights (or one height for all of them), the height of those without it, and which property names them. They stand in the buildings table after the scene's own, where **Remove building** leaves one out of the file, and **Remove the map's buildings** drops the file.

The scene's own entry then has a **Wind and temperature** group: the weather, the wind, the temperature over the ground and their **Eddies** (see Turbulence), with **Show the profile** to plot them. **Weather of** picks whose weather the fields show: the scene's, a weather zone's (**Add weather zone** draws one in the plan) or a weather class's (**Add weather class**), each class with its share of the time in each period of the day. A material's page has a **Ground impedance** group, to make the material a ground: ticking it keeps the material's scattering unless that is 0.1 in every band, as a new material's is, since a ground scatters nothing unless its material says so, and unticking it gives a ground that scatters nothing 0.1 in every band (see Ground impedance). The **Simulation** entry adds **Barriers and buildings** (the solver for whole rows or the paths over the edges, see Sound over rows of barriers and buildings), the **Parabolic equation** (narrow-angle or wide-angle), **Frequencies behind barriers** (each choice with its largest error at 8 kHz), **Frequencies over open ground** (each choice with its largest error at 1 and 2 kHz) and, with buildings, barriers or vegetation, an **Insertion loss** list: ticking one runs the scene again without it (see Results outdoors). **Draw room** replaces an outdoor scene with a floor plan, after asking.

Exporting to Blender

`acoustic-spaces export` writes a scene as glTF 2.0 (`.glb`) for Blender or another 3D tool; the editor does the same from its File menu.

```
acoustic-spaces export examples/concert_hall/scene.yaml hall.glb              # the room
acoustic-spaces export examples/concert_hall/scene.yaml hall.glb --paths      # + reflection
paths
acoustic-spaces export examples/concert_hall/scene.yaml hall.glb --particles  # + particle
animation
```

In Blender, use **File > Import > glTF 2.0 (.glb/.gltf)** and pick the file. The axes are the scene's (+z up, metres).

- Each surface is an object named as in the editor, with a see-through material per scene material; the material's custom properties hold its absorption and scattering per octave band. Sources (red) and listeners (blue) are spheres turned the way they face, with an arrow along the aim. Outdoors, each tree, hedge or belt of trees is a see-through green object under `vegetation`, and the sky is left out.
- `--paths` adds the early reflection paths from `--source` to `--listener` up to `--max-order`, as long after the direct sound as a simulation keeps them (`transition_ms`, or the mixing time with the statistical late method, see Simulation), one object of edges per order. Edges do not render: to render them, use Object > Convert > Curve and give the curve a bevel depth.
- `--particles` adds the particles of `animate` as small spheres that move with them and shrink with their level, gone 60 dB down: 1000 of them (`--count`) over the first 100 ms (`--duration`), 40 times slower than the sound (`--slow-motion`). If the timeline ends sooner, set its end frame, then press play. The editor asks for the same three: **Particles**, **Sound to animate** and **Slow motion**; outdoors it starts with sound enough to reach the farthest listener. Blender draws a dashed line from each particle to the object that holds them; **Overlays > Relationship Lines** hides them.

Example scenes

The examples come in groups, the same ones the editor's File > Open example shows, each from the simplest.

Rooms

Example	What it is
`shoebox`	An 8 × 6 × 3 m room from an OBJ file, the simplest case every method must get right.
`room_modes`	The bass in a small bare room from the wave solver, in third-octave bands, on the CPU or a GPU, against rays alone.
`studio`	A 6 × 4.5 × 3 m control room with side panels, a diffuser and stereo loudspeakers at ±30°.
`listening_room`	A living room with a loudspeaker against the front wall or a metre out: the wall's reflection adds to the bass in pressure, or takes from it (early reflections in pressure); `bass.py` prints how much.
`apartment`	Two rooms of a flat, a television in one and a bed in the other: how loud the television is through the wall and the door, closed or open.
`archway`	A bar and a dining room joined by an archway in a wall 40 cm thick: a diner behind the wall hears the singer over both corners of the arch's jamb (`second_order` diffraction).
`workshop`	Noise in a workshop hall: a conveyor 20 m long (a line source), a vibrating machine casing (a surface source) and a compressor, each with its own sound power spectrum. Map the A-weighted level over the floor, then with another ceiling.

Halls and churches

Example	What it is
`lecture_hall`	A fan-shaped lecture hall built from its floor plan, no CAD file needed.
`concert_hall`	A 32 × 20 × 15 m shoebox hall with a raised stage and upholstered seats; two sources on stage, three seats in the stalls.
`orchestra`	The concert hall with a full orchestra on its stage, one source for each of the 61 tracks of an anechoic recording, and the scripts that mix them and make a video.
`church`	A stone nave with an apse, 18 m high, with about six seconds of reverberation; `church_demo.py` plays its choir and organ, left out of the church's own simulations until ticked, as are the places it tours, and makes a video.
`gallery`	An L-shaped gallery: one listener sees the guide, one stands round the corner and hears the guide by edge diffraction, and a visitor walks away from the guide and round the corner, for a walkthrough; `walk_demo.py` makes its video, where the visitor then looks round from the listener `standing`, left out of the gallery's own simulations until ticked. With the walk turned off the visitor stands where it starts.

Sound systems

Example	What it is
`town_hall_pa`	A stone town hall of 4000 m³ with a sound system for speech: two columns (coherent arrays steered down onto the seats) beside the stage, and two delay loudspeakers further down the hall that play later, quieter and with less bass. `delays.py` maps the STI with and without the delays: they help the back rows and cost the front a little.
`concert_stage`	A concert stage on open ground with its monitors: wedges for the singer, a drum fill and side fills, beside the two main line arrays of 12 boxes in a J (coherent arrays of a generic box's balloon, added up box by box near the players), the subwoofers and the front fills. `align.py` shows each player hears their own monitor loudest; `feedback.py` finds the gain before feedback of the singer's microphone, a supercardioid listener in front of the singer's voice, both left out of the scene's simulations until ticked, at the mix position and at the singer's ears, and renders it ringing; `near_field.py` prints how far the arrays' far-field patterns would be off at each listener.
`stadium_concert`	A concert in a football stadium, with its sound system as tours hang it and the stage's monitors: the two main line arrays, a block of eight cardioid subwoofers timed to arrive with them at the mix position, out-fills, in-fills for the pitch's centre, front fills, two delay towers on the pitch and four delays under the roof's edge, each arriving 6 to 20 ms after the main arrays at its seats, and on stage `concert_stage`'s wedges, drum fill and side fills. Every array adds up its boxes one by one. Map the level over the pitch and the stands; `align.py` prints when and how loud each loudspeaker arrives at each listener on the pitch, the stands and the stage, and which would be heard as an echo; `near_field.py` prints how far the arrays' far-field patterns would be off; `feedback.py` finds the singer's microphone's gain before feedback. Its sound system plays alone with the Stage and Monitors groups left out. `stadium_demo.py` takes in the band, left out of the concert's simulations as the singer's microphone is, plays a song through it all and makes a video, and `helicopter_demo.py` then flies the scene's helicopter, left out of the concert's simulations, 150 m over the stadium halfway through the song (a helicopter over the stadium concert).

Outdoors

Example	What it is
`street`	Outdoors: traffic along a street between a row of houses and an office block, heard on the pavement, on a balcony and in a garden behind a noise barrier, where the sound comes over the barrier and round its ends (outdoor scenes); `--without noise_barrier` gives the barrier's insertion loss. A car (with its horn, if wanted), a bus, a motorcycle, a police car with its siren on and a fire engine with its siren and its air horn, left out until ticked, drive along the street at 50 km/h with Doppler, and a tram runs down its middle at 50 km/h: `passby_demo.py --vehicle` plays one (vehicles passing by). Road works in the lane on the houses' side, a breaker on an excavator, a generator and a dumper reversing with its alarm on, are left out too: `works_demo.py` plays them, outdoors or, with `--indoors`, in the houses and the offices at the pavement and the balcony (road works in the street). `vibration.py` prints how much the tram, the breaker and the dumper shake the houses and the offices, against the US FTA's limits and damage criteria (vibration through the ground).
`roadside`	Outdoors: traffic on a road between a meadow and a paved yard, heard over the grass at 7.5, 25 and 50 m and over the yard at 25 m: the ground dip over the grass at a few hundred hertz, which the hard yard does not have. A car at 50 km/h and a truck at 70 km/h, left out until ticked, drive along the road with Doppler: `passby_demo.py --vehicle` plays one (vehicles passing by).
`railway`	Outdoors: a railway line on ballast across a meadow, heard where the EU's rule for trains' noise measures a pass-by (7.5 m from the track, 1.2 m above the rail top), 25 m away over the meadow, 25 m away behind a noise barrier 3 m high and 2 m in front of a house 20 m from the track; the line source is the track's traffic on average, one regional train an hour at 160 km/h, its sound power per metre and its directivity round the track Schall 03's (Germany's method for railway noise). A train, left out until ticked, runs past with Doppler as three sources along one walk (at the rail top, on the roof and at the pantograph): `passby_demo.py --train` runs a regional train, an S-Bahn, an intercity or a freight train past, sounding its warning horn with `--horn blasts`, heard outdoors or, with `--indoors`, in the house (trains passing by). `vibration.py` prints how much each train shakes the house, or one where each other listener is, and how loud it rumbles in its rooms, against the US FTA's limits, and from how far they hold (vibration through the ground).
`tree_belt`	Outdoors: traffic on a road heard from two gardens 55 m away, one behind a belt of trees 30 m deep and one across an open meadow: the trees' foliage and trunks take up to about 8 dB from the highest bands and almost nothing from the lowest, and scatter some of the sound into the late part, which at 500 Hz and 1 kHz fills the ground's dip: there the garden behind them is louder than it would be without them.
`hillside`	Outdoors: a works yard at the foot of a hill, heard 160 m away in the hill's shadow, on the slope facing the yard and on the crest, over a terrain read from a grid of heights, with two houses from a map: the sound bends over the crest to the listener behind it, which takes 10 dB off the lowest band and nearly 40 dB off the highest, and reflects off the mean plane of the slope to the one facing the yard.
`open_air_stage`	Outdoors: a stage heard 100 and 300 m away across a park in a 5 m/s wind, downwind and upwind: the sound bent down to the ground downwind, sooner and louder in the low bands, and up over the listeners' heads upwind, into a shadow 300 m away that the wind's eddies scatter sound into.
`lakeside_concert`	Outdoors: a summer evening concert on a lawn by a lake, heard by the audience, from a boat and in a village 550 m away across the water, with two weather zones: over the lake the air lies cool and the breeze blows stronger, both bending the sound down to the water, and among the village's houses the breeze drops. The editor's **View > Show wind** and **Show wind profile** draw the three weathers; `lakeside_demo.py` plays a song across the lake through the stage's sound system, left out of the scene's own simulations until ticked, in that weather and in still air, and makes a video.

Example	What it is
`helicopter_flyover`	Outdoors: a helicopter, an AS350 B3, flying over a meadow 150 m up at 63 m/s, heard right under its track, where its noise certification's microphone stands, and 150 m to its left and right. Its two rotors' sound comes from their model, tones and broadband noise, the tones louder to its left, where the main rotor's blades advance; a light wind against it bends the sound it sends ahead up and the sound behind it down, and eddies make it flicker. `flyover_demo.py` plays it past each listener with Doppler (a helicopter flying over).
`airport`	Outdoors: an aerodrome's runway across a meadow, and Doc 29's hypothetical twin jet JETW (ECAC Doc 29 Volume 3) taking off and landing on it, heard 1.2 m above the grass under its departure 4 km from its start of roll, beside the runway 450 m from its centre line (on the line along which an aircraft's noise certification measures its take-off from the side) and under its approach 2 km before the threshold (where the certification measures its approach). Each flies its procedure as Doc 29 does (its heights, speeds and thrust, Appendix B), with the sound power its NPD curves give at its highest thrust and its engines' installation as directivity (an aircraft in a scene). The example's own simulation hears the departure at its start of roll, at take-off thrust; the arrival, as loud as its reverse thrust after touchdown, is left out until ticked. `flight_demo.py` flies either past each listener with Doppler (aircraft taking off and landing).

Models from CAD and BIM files

Example	What it is
`rehearsal_room`	A band's rehearsal room under a pitched roof from a Collada file in inches, as SketchUp exports it.
`bim_office`	An office from a BIM model in IFC, read as one room from its space boundaries, its surfaces named by element, material and IFC class (BIM models; the `ifc` extra).
`leaky_classroom`	A classroom model with two mistakes that leave it open, a missing window pane and a ceiling 3 cm short of a wall, for the leak finder to find.

Measurements and material libraries

Example	What it is
`fitting`	A seminar room that measured longer than it was designed, as the panels on its back wall were left out: `make_measurements.py` makes up the measurements, then compare the model with them and fit the back wall to them.
`material_libraries`	A seminar room whose materials come from a CATT-Acoustic file and an ODEON library (material libraries).
`measurement`	No scene, a script: it records a sweep in a simulated room, as a microphone would, to try the measurement commands without a sound card.

`acoustic-spaces examples` lists them with where they are installed, and `acoustic-spaces examples --copy my-rooms` copies them to a folder you can edit. From Python, `acoustic_spaces.examples.example_path("church")` gives a scene file.

Scene files

A scene is a YAML file that names the geometry, the materials, which surface gets which material, and where the sources and listeners are. See `examples/shoebox/scene.yaml` and the schema in `src/acoustic_spaces/scene/model.py`. Frequency-dependent data uses eight octave bands from 63 Hz to 8 kHz, or 24 third-octave bands from 50 Hz to 10 kHz (see Frequency bands).

Geometry

Format	Read with	Names used for material assignment
OBJ	built-in reader	`usemtl` material, `g` groups, `o` object
DXF	ezdxf (3DFACE, MESH, polyface and polygon meshes, closed 3D polylines, block references)	layer
glTF / GLB	trimesh	material, node and mesh names
STL, PLY, OFF	trimesh	file name
IFC (`.ifc`)	IfcOpenShell, the `ifc` extra	element name, type name, materials, IFC class
Collada (`.dae`)	built-in reader (triangles, polylists, polygons, nodes and instanced nodes)	material, geometry name
SketchUp (`.skp`), experimental	openskp, the `sketchup` extra	material

Simple rooms need no CAD file: `geometry: {box: [8, 6, 3]}` builds a rectangular room from the origin, and a floor plan is raised to a height (see `examples/lecture_hall/scene.yaml`):

```yaml
geometry:
  floor_plan: [[0, -4], [0, 4], [14, 8], [14, -8]]  # corners, in order, any shape
  height: 6
assignments:
  floor: audience
  ceiling: plaster
  wall_1: wood       # the edge from the first corner to the second
  walls: concrete    # every other wall
```

OBJ is the recommended exchange format: SketchUp, Rhino, Revit and Blender all export it with material names. Set `geometry.units` (`m`, `cm`, `mm`, `in`, `ft`) and, when the file is +y up (Blender's OBJ default, glTF always), `geometry.up_axis: y`. Without `units`, a DXF file is read in the units its header records (`$INSUNITS`), an IFC or Collada file in the units it records, and any other file in metres; a Collada file's up axis is the one it records (X_UP, Y_UP or Z_UP). SketchUp's Collada export (`.dae`) is read as it is (see `examples/rehearsal_room`).

SketchUp files can also be read as they are, with the `sketchup` extra: openskp reads them without SketchUp, in metres. That reader is young. Faces are matched to materials by colour, so give each material its own colour; a face painted differently on its two sides takes its front's material; and some files do not read at all (the BRAS auditorium, for one). An OBJ export from SketchUp is then the way.

Polygons are triangulated, duplicate vertices merged, degenerate faces, copies of a face (the same corners, turning the same way) and the vertices no face uses dropped, and faces that meet along an edge are turned to agree with each other (each part of the model keeps the orientation most of its area has), so a few faces flipped by the exporter do not spoil the volume or the inside checks. Where a face's edge runs past the corners of faces on its other side (a T-junction, which SketchUp stitches with faces of zero area), the edge is split at those corners, so the model stays closed; also where two surfaces and a third face meet along the line it runs past. A stitch whose own corner lies on its edge (within 10 µm) is dropped there, as the split closes the gap. A model with no faces left is refused, as is one with coordinates that are not finite numbers or a face that names a vertex the file does not have. `validate` then reports surfaces without a material, sources or listeners outside the room or on a surface (within 1 µm of it, or within 2e-9 of the model's diagonal where that is more, twice the least distance at which a ray traced on the CPU meets a face; an opening, with air on both sides, does not count) and listeners within 1 cm of a source; it warns about models that are not closed (rays would escape), listeners so close to a surface or an opening that the late part comes out noisy, and assignments that match nothing. Errors exit with status 1, and with `--strict` warnings too. Normals may point either way: the room is the space the faces close off, found from the faces themselves (see Walls, doors and openings), and a panel standing in the room, with air on both sides, reflects and absorbs on both.

Outdoors

`geometry.ground` makes a scene outdoors: a flat ground under an open sky, with `buildings` (footprints raised to a height), `barriers` (thin screens along a path), `vegetation` (belts of trees, hedges and lone trees, which sound passes through) and the objects of a model file standing on it. See Outdoor scenes.

```
geometry:
  ground:
    margin: 30
    zones:                        # parts of the ground with materials of their own
      - {name: road, polygon: [[-40, -4], [60, -4], [60, 4], [-40, 4]]}
  buildings:
    - {name: hall, footprint: [[0, 10], [20, 10], [20, 22], [0, 22]], height: 9}
  barriers:
    - {path: [[30, 6], [30, 20]], height: 3}
  vegetation:
    - {name: hedge, path: [[30, 24], [50, 24]], width: 1.5, height: 2}
```

The ground can be a terrain, its heights on a grid from a file or given in the scene (`ground.terrain`), and the buildings can come from a GeoJSON map (`geometry.buildings_file`); see Terrain and Buildings from a map:

```
geometry:
  ground:
    terrain: {file: hills.asc}      # or {heights: [[...], ...], spacing: 10}
    map_origin: [552000, 5806000]   # subtracted from the map files' coordinates
  buildings_file: {file: town.geojson, height: height, default_height: 9}
```

Outdoors, `atmosphere` sets the wind and the temperature over the ground, which bend sound down or up, into shadow zones (see Wind and temperature), their eddies, which blur the ground dip, scatter sound into shadows and make it flicker (see Turbulence), and `zones`, parts of the scene with weather of their own (see Weather zones):

```yaml
atmosphere:
  preset: windy                 # neutral, windy, sunny_day or clear_night
  wind: {speed_m_s: 8, direction_deg: 90}   # 8 m/s 10 m up, blowing towards +y
  turbulence: moderate          # weak, moderate or strong, or its structure parameters;
                                # {preset: moderate, height_m: 2}: weaker with height
  zones:                        # polygons in the plan; what they leave out is the scene's
    - {name: lee, polygon: [[120, -60], [300, -60], [300, 60], [120, 60]], wind: {speed_m_s:
1}}
```

`long_term` runs an outdoor scene in several weather classes, each holding for a share of the day, the evening and the night, for the level over a long term and L_den (see Levels over a long term):

```yaml
long_term:
  classes:                      # a class takes the scene's weather where it says nothing
    - {name: downwind, share: {day: 0.4, evening: 0.3, night: 0.2}}
    - {name: upwind, wind: {direction_deg: 180}, share: {day: 0.3, evening: 0.2, night: 0.1}}
    - {name: inversion, preset: clear_night, share: {day: 0, evening: 0.3, night: 0.6}}
    - {name: still, preset: neutral, share: {day: 0.3, evening: 0.2, night: 0.1}}
  # periods: [{name: day, hours: 12}, {name: evening, hours: 4, penalty_db: 5},
  #           {name: night, hours: 8, penalty_db: 10}]   # the default, L_den
```

BIM models (IFC)

IFC files from Revit, ArchiCAD, Vectorworks or any other BIM program are read with the `ifc` extra (IfcOpenShell). There are two ways, and `examples/bim_office` shows both:

```yaml
geometry:
  file: office.ifc
  space: Office        # one room, from its space boundaries; leave out for every element
assignments:
  Plaster on brick: plaster   # an IFC material
  IfcWindow: glass            # every element of a class
  Door: wood                  # one element, by name
```

- With `space`, only that room (an IfcSpace, by name, long name or GlobalId) is read: the faces of the room that each element bounds, as BIM programs export them as space boundaries (Revit: export with space boundaries, first or second level). Each face is named after its element; one with no element behind it, an opening to another room, is a `virtual boundary`. This gives a closed room with one face per surface, which is what the simulation wants. A file without space boundaries gives the space's own solid, cut into triangles of at most 50 cm, each named after the element behind its middle: windows and doors then have ragged edges, so export boundaries where you can.
- Without `space`, every building element is read as IfcOpenShell builds it: walls, slabs, roofs, windows, doors, stairs and furniture, with their openings cut. Openings, virtual elements and building services (ducts, pipes, lights) are left out. Walls then have both sides and their thickness, and solids that only touch do not share edges, so the model reads as not closed even when no ray escapes (the leak finder tells); the volume and the inside checks are skipped.

Each face's names, for assignments, are its element's name, its type's name, its materials (a layer set's name, then each layer's material) and its IFC class (`IfcWallStandardCase` also as `IfcWall`); the first that an assignment names gives its material.

Finding leaks

A model that is not closed lets rays escape, and its reverberation comes out too short. The warning counts its open edges; the leak finder shows where the holes are:

```
acoustic-spaces validate examples/leaky_classroom/scene.yaml --leaks
acoustic-spaces view examples/leaky_classroom/scene.yaml --leaks
```

```
leaks: 5000 rays, up to 100 reflections each: 4084 escaped (81.7 %)
  1. 3813 rays (76.3 %) through a gap of about 3.6 m2 (3.00 x 1.20 m) by east_wall
     at (9.00, 3.51, 1.59) m
  2. 271 rays (5.4 %) through a gap of about 0.27 m2 (9.00 x 0.03 m) by ceiling,
     north_wall, west_wall
     at (4.08, 6.99, 3.00) m
```

It traces rays from the sources (5,000 by default, `--rays` sets how many), reflecting them diffusely off every surface whatever its material, so that they reach every corner, for up to 100 reflections each. Each ray that finds no surface ahead has escaped, and is put down to the gap it went through: the open edges round a hole, with open edges that come within 2 % of the model's size of each other counted as one gap (a ceiling that stops short of a wall leaves open edges on both). For each gap the report gives the rays that escaped through it, its area and size (from its open edges, which run round it), where the rays crossed it and the surfaces next to it, those with the longest open edges first. In the example the window glass is missing from the east wall and the ceiling stops 3 cm short of the north wall. `view --leaks` draws the open edges in red, the escaping rays from their last reflection to past the hole, and a numbered marker on each gap; in the editor, **View > Show leaks** does the same and writes the report to the messages.

Rays that leave through a surface letting sound through (an opening to the outside, see Walls, doors and openings) are listed as such, not as a leak. A closed model keeps every ray; one that loses some all the same has faces that meet without closing, and the report names them. From Python, `acoustic_spaces.engine.leaks.find_leaks(room)` gives the same as a `Leaks` object, and `acoustic_spaces.viz.plot_leaks` draws it.

Materials

Each entry in `materials` takes coefficients from one of:

- `preset: <id>`: the built-in database (`acoustic-spaces materials` lists it);
- `file: <csv>`: a table of rising centre frequencies in Hz, absorption and optionally scattering, one row each, at any band resolution (the layout the BRAS benchmark uses);
- `file: <library>` with `entry: <name>`: a material of a CATT-Acoustic or ODEON material library (see below);
- `absorption: [...]` with eight octave values or 24 third-octave values, which also overrides a preset or file;
- `impedance:` a porous ground's impedance (`grass`, `snow`, `asphalt`… or a flow resistivity; `reaction: extended` for one whose impedance changes with the angle), which gives the absorption and reflects as a ground does (see Ground impedance).

`scattering` is a number, eight or 24 values, or the id of a built-in scattering entry (`acoustic-spaces materials --kind scattering` lists them; `theatre_audience`, measured over rows of seated people, rises from 0.3 at 125 Hz to 0.7 from 1 kHz up, and stands for the rows of an audience area modelled as one raked plane, as CATT-Acoustic and ODEON models do). Without it a material scatters 0.1 in every band, and a ground with an `impedance` nothing (see Ground impedance). File data with more than one value in some octave (third-octave tables) stays in third-octave bands; other data is averaged onto octaves. Bands with no data (63 Hz in most tables) take the nearest band's value and the material's `source` says which. Coefficients run from 0 to 1: values written in the scene must lie in that range, and a file's values outside it (absorption measured in a reverberation room can exceed 1) are taken as 0 or 1, with a warning that names the material, the file and the bands. Surfaces get a material through `assignments` (any geometry name to a material), by having the same name as a material, or through `default_material`. Names match whatever their case (`brick_rough` names SketchUp's `Brick_Rough`), though a name written the same way wins over one in another case.

`uncertainty` (**Absorption uncertainty** on the material's form in the editor) says how far a material's absorption may be off: a standard uncertainty (`distribution: normal`, the default) or a tolerance of plus or minus the value (`distribution: uniform`), in absorption coefficient units, one value or one per octave band, with an optional `source`. A material's bands are taken as moving together, and the materials as independent of one another.

```yaml
materials:
  plaster:
    absorption: [0.02, 0.02, 0.02, 0.03, 0.04, 0.05, 0.05, 0.05]
    uncertainty: {absorption: 0.02, source: lab report}
```

A built-in material (`preset`, with no absorption values of its own) that gives no uncertainty takes `simulation.uncertainty.presets`, as the built-in database does not say where its values come from (**Built-in materials** in the editor's simulation settings):

```yaml
simulation:
  uncertainty:
    presets: tolerance        # tolerance (default), iso_12999_2 or none
```

`tolerance` is plus or minus 0.1 in every band; `iso_12999_2` is twice the reproducibility standard deviation that ISO 12999-2:2020 gives for absorption measured to ISO 354, sigma_R = m alpha + n, as a standard uncertainty: from 250 Hz to 4 kHz the standard's octave bands' (Table 2, for the practical absorption coefficient of ISO 11654), elsewhere the mean of its third octaves' (Table 1; the 63 Hz octave from the 63 and 80 Hz rows, the 8 kHz octave, past the table, the 5 kHz row's); `none` gives them none. The standard's data come from absorbers, and below 100 Hz from outside ISO 354's range, so for hard surfaces and the 63 Hz octave it is taken beyond them. Plus or minus 0.1 is much for a hard surface: the shoebox example's painted concrete, 0.01 to 0.02, is drawn anywhere from 0 to 0.12, so that example's traced T30 at 1 kHz, 1.28 s, lies near the top of its 95 % interval, 0.74 to 1.49 s. For an absorber ISO 12999-2's reproducibility is largest at low frequencies (for alpha = 1, 0.41 at 63 Hz, 0.20 at 125 Hz and 0.040 at 1 kHz), so twice it spreads the lowest bands far. From 500 Hz to 4 kHz it is the same for every absorption (0.040, and 0.050 at 4 kHz), and a hard surface gets the least in the lowest bands (for alpha = 0.02, 0.023 at 63 Hz and 0.019 at 125 Hz). The spread names the materials that took the default (`assumed`).

`acoustic_spaces.engine.uncertainty.material_spread` gives the uncertainty this leads to in each listener's ISO 3382-1 parameters: the mean, combined standard uncertainty and 95 % coverage interval of many draws, a Monte Carlo propagation as GUM Supplement 1 (JCGM 101:2008) describes. Each draw takes the absorption from its distribution and re-weights the traced run instead of tracing again, so the run must be simulated with the materials' face groups (`acoustic_spaces.engine.reweight.groups_of`, then `simulate(..., groups=...)`), in octave bands, with stochastic ray tracing for the late part, without the wave solver and with at most 16 materials on its surfaces. A band in which a material absorbs everything (an absorption of 1, as some absorbers have in some bands) is not drawn, as no energy is left in it to re-weight; the spread names these (`held`). A draw does not go below what the material lets through, which counts as absorbed. An absorption uncertainty beside a ground's impedance is left out, with a warning: the impedance gives the ground's reflection, which does not follow the absorption as other surfaces' do. On a ground with zones, a reflection near a zone's edge mixes the grounds on both sides, and a draw takes it as off the ground it meets alone. To each draw it adds the rays' random error: the standard error of the late rays' batches (`simulation.late.batches`, 4 or more) times a deviate of Student's t with one degree of freedom fewer than the batches, as GUM Supplement 1 takes a quantity known from that many observations. The budget gives each input's standard uncertainty alone (`absorption`, `rays`), and the combined one is the root of the sum of their squares: the rays' error is added to each draw on its own, so its variance adds exactly, where the standard deviation of draws that hold it would be unsteady with few batches. An end of the interval, or a one-sided bound, past what the parameter can be (a decay time below 0, a D50 above 1) is held at that limit, as the Analytical Methods Committee recommends near a natural limit (AMC Technical Brief 26A). The parameters from the late sound's directions (LF and the like) have no standard error and take the absorption's alone. Not counted: the scattering and the method, and a parameter that some draw leaves undefined (a decay too slow for the response's length, say) gets none in that band. Of the method's errors, those over edges are listed where a response has paths over edges (`not_counted`), each with the largest difference of the level from the exact solution that Validation finds in each band, as the band adds its paths in pressure or in energy: over one edge, over two (`second_order`), a path reflected on both sides of an edge left out (`one_side`), and where a path reaches an edge's end (a solid block's corners; a thin screen's edges fade out past their ends). Where the solver for whole rows carries a row's sound, its own largest difference from the wave calculation is listed in those bands (`whole rows`) and the edges' in the others. That sound stays as it was traced in every draw, though it depends on the absorption of a ground without an impedance. The method's errors are not drawn, as they are measured for whole levels in a few geometries, not for each path. In the lowest bands, where the early paths add in pressure, a reflection's phase follows its face's admittance, which is fitted to the absorption; the draws take it between fits at nine absorptions of each material over their range rather than a fit of their own: in the shoebox example G within 0.17 dB and C80 within 0.20 dB of a fit for each draw at 63 Hz, where a fit of its own moves them by as much for a change of 0.001 in the absorption (0.18 and 0.23 dB in one of the draws furthest off). `acoustic-spaces simulate` does this itself when a material on the surfaces has an uncertainty, its own or a built-in material's: it prints the largest combined standard uncertainty in JNDs from 125 Hz to 4 kHz, with its budget, 95 % interval, one-sided bounds and the errors over edges, and `parameters.json` holds them all (`uncertainty`). With settings that cannot re-weight (third-octave bands, say) it leaves the spread out and says what the spread needs, as a warning when a material gives an uncertainty of its own; so does the editor, in its messages. The editor shows each run's results at once, draws the spread after them and gives it in the results table's tips. The draws add to a run: in the concert hall example (five built-in materials, six responses) 400 draws take 21 s in four worker processes (below; 72 s in the run's own process) after a 12 s run on four cores, about half of their work the lowest bands' paths added in pressure for each draw; `presets: none` leaves the built-in materials out.

`acoustic-spaces simulate` and the editor work the draws out in worker processes, as many as the run has threads (`simulation.performance.threads`, `--threads`) and the cores it may use, with the same results as in the run's own process and the same messages. Each process takes a run of one response's draws at a time and holds that response, as traced and as a draw re-weights it, what the run's draws share (up to 128 MiB of the coherent bands' delay phases and band grids) and its own interpreter, some 80 MiB; the processes take at most half of the computer's memory (or of its container's limit) together, so fewer of them for large responses. There is one process for each 100 draws of all the responses together at most, so a spread of fewer than 200 draws in all stays in the run's own process, as does every spread with `--threads 1`. The admittances the draws take in the lowest bands are fitted before the draws, each once. The processes inherit the run's environment and the cores it may use and set nothing of the linear algebra's threads: a draw's products are added in a fixed order, and the library's solvers (the admittances' fits) run in them as in the run's own process. The results come out the same to the last bit, except on an Intel Mac, where Apple's Accelerate rounds a sum by where its numbers lie in memory and the admittances' fits may differ in their last digits as they may from one fit to the next in one process. In Python, `material_spread` and `with_spread` take `workers` (1, the default, for none; 0 for every core). The draws stay in the calling process in a daemonic process (a `multiprocessing.Pool`'s). The processes are started by spawning, which imports the main module again in each, so a script that gives `workers` must do so under `if __name__ == "__main__":`, or the call ends with an error.

`simulation.uncertainty.draws` is the number of draws: 400 unless the scene says otherwise, 0 for no spread (`--spread-draws` on the command line, **Draws** in the editor's simulation settings). For a normal spread, 400 draws leave the standard uncertainty good to about 7 % and each end of the 95 % interval to about a quarter of the standard uncertainty (twice their own standard deviation). With `draws: adaptive` the draws go on as the adaptive procedure of GUM Supplement 1 has them (JCGM 101:2008, 7.9): batch after batch until, for every result in every band of every response, twice the standard deviation of the mean, of the standard uncertainty and of each end of the interval (and of each one-sided bound, below) is within the numerical tolerance of the standard uncertainty, half a unit of its first digit. A batch is 100 draws, not the Supplement's 10 000 (a number it calls arbitrary), as each draw re-weights the whole run; the test is first made after ten batches, as two this small can pass by chance; and the draws stop at 20 000, the Supplement's own first test. `stable` in `parameters.json` says whether the test was met (null for a number of draws given beforehand, which is not tested). The test asks for many more draws than the default: from 9 000 to 14 500 in runs of the shoebox example.

For a limit on one side, the spread gives each result's one-sided bounds from the same draws (`at_least` and `at_most` in `parameters.json`): the value the result is at least with a probability of 95 %, and the one it is at most with that probability; for a normal spread, the mean less and plus 1.645 standard uncertainties, where the 95 % interval's ends lie 1.96 away and leave 2.5 % out on each side. `simulation.uncertainty.one_sided` gives another probability between 0.5 and 1, 0.9 say (`--one-sided` on the command line, **One-sided bounds** in the editor's simulation settings).

The same draws give the spread of each listener's levels and speech results (`uncertainty.levels` in `parameters.json`, with the same parts): the sound pressure level per octave band and A-weighted, the STI, the STIPA, U50 and, outdoors, the level relative to free field. Each draw finds them for the source's sound power and the scene's background noise, and their rays' error comes from the same batches. A band's level is its G and the source's power, so its spread is G's; in a band whose G some draw leaves undefined, the level is that of the energy the response holds and has a spread of its own. `simulate` prints the A-weighted level's with the level it is at most and the STI's with the STI it is at least, and the editor's tips give the level's in each band. A coverage map gets the uncertainty of its values by tracing it again for each material (see Coverage maps); a listener's insertion loss and long-term levels have no spread.

Material libraries of CATT-Acoustic and ODEON

Materials kept for CATT-Acoustic or ODEON can be used as they are. `file` names the library and `entry` the material in it: its name (in any case), or for ODEON its number or description.

```
materials:
  carpet:
    file: catt_materials.geo   # CATT-Acoustic: ABS lines, in a geometry or library file
    entry: carpet
  ceiling_tiles:
    file: odeon_materials.li8  # ODEON: Material.Li8 or a library of the same layout
    entry: 1001
    scattering: 0.1
```

- CATT-Acoustic writes `ABS name = <a125 a250 a500 a1k a2k a4k> L <s125 ... s4k> {r g b}`: absorption and scattering in percent from 125 Hz to 4 kHz, 8 and 16 kHz optional (`<a125 ... a4k : a8k a16k>`), `;` before a comment; `ABS1` gives them as fractions from 0 to 1. `ABS name = other` gives a material another's values. Any file with such lines will do (a `.geo` file with its geometry, or a library). Scattering that CATT estimates itself (`L <estimate(...)>`) is not read, so give `scattering`.
- ODEON's library has each material on two lines, its number and description, then eight absorption coefficients from 63 Hz to 8 kHz (six, from 125 Hz to 4 kHz, in older libraries). Where the description has ODEON's `{Product name*=...}` and `{Category=...}` fields, as in its own `Material.Li8`, the product name is the material's name, and `materials --file` lists the materials by category. Materials made with ODEON's Material calculator are read by their octave values, which ODEON simulates with; their third-octave values are not read. ODEON gives scattering to surfaces, so give it with `scattering` (0.1 otherwise).

The 63 Hz band takes the 125 Hz value where a library has none, 8 kHz the 4 kHz value, and 16 kHz is left out. `acoustic-spaces materials --file <library>` lists a library's materials, with words to search for as in the built-in database, and in the editor **Edit > Add material from a library** picks one. The `material_libraries` example takes its carpet and walls from a CATT file and its ceiling from an ODEON one.

Four rooms of the BRAS benchmark, converted from its SketchUp models, are in `validation/bras` as scenes with their surface data and measurements (the coupled rooms twice, with the door between them at two angles).

Walls, doors and openings

A model can hold several rooms. Sound passes between them through surfaces whose material lets some through: `transmission`, the share that passes (1 for an opening), or `transmission_loss`, the sound reduction index R in dB (the share is 10^(-R/10)), one value or one per band.

```yaml
materials:
  partition:                 # 12 cm of brick
    absorption: [0.02, 0.02, 0.03, 0.04, 0.05, 0.07, 0.07, 0.07]
    transmission_loss: [30, 35, 40, 46, 52, 57, 60, 60]
  door:
    absorption: [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]
    transmission_loss: 25
  doorway: {transmission: 1}  # an opening: all of it passes
assignments:
  wall_between: partition
  door_leaf: door
  door_hole: doorway
```

- What passes through counts as absorbed on the side it comes from, so the absorption must be at least the transmission in every band. A material without absorption values, preset or file absorbs what it lets through: an opening absorbs 1. What a surface lets out of the rooms goes into no room and does not come back in through another: the late part's rays and the rain of their reflections stop there.
- The rooms are found from the faces: every space the faces close off on all sides is one, whichever way the normals point. The space inside a thick wall or a column, with rooms all round it, is solid, as is the inside of a closed shell standing on the floor or in a corner, its faces lying on the room's. Such a solid holds air when a source or listener stands in it and a surface that lets sound through joins it to a room (a booth with glass walls), and so does any space it opens into that way; a source or listener inside any other solid is reported as inside a closed solid, and `simulate` refuses to run. When none stands in the model's rooms (walls drawn as a closed shell inside another), the spaces they stand in are the rooms. A material with `transmission: 0` lets nothing through. The faces between two rooms have a room on both sides and reflect and absorb in both. `inspect` lists the rooms ("closed, 2 rooms of 48.00, 72.00 m3"), the editor too, and the volume is theirs together. When the sources and listeners stand in air that the model's rooms do not include (walls drawn as a closed shell inside another), `inspect` also gives that "air volume", the one its reverberation times use.
- Early reflections pass through those surfaces, keeping the share that passes at each; through openings they pass unchanged. A reflected path through more than two partitions is left to the late part, which with the stochastic method counts it (the statistical method takes what the early paths carry out of its own energy, so there the early part keeps any it finds). The rays of the late part that hit a partition go on as two: the reflection, and a copy behind it with the share that passes, so even a 60 dB wall has rays behind it. Each room's rays are thinned and split for the level that room is expected to reach (from the rooms' power balance in a diffuse field), so a quiet room behind a wall is traced as finely as the loud one. The detector spheres are sized for the volume of all the rooms, which the rays spread over: sized from the volume of a side room behind a 10 or 40 dB wall, they gave the same G, C80 and T30 there with half as much again spread between seeds. Sound goes straight through, in the direction it came.
- Coverage maps cover the audience areas of every room and find at each point what a listener there would. The statistical late method follows coupled-room theory: each room's diffuse field decays as the energy the rooms exchange through the faces between them has it, with Eyring's absorption in each, so a room behind a partition holds less and a dry room opening into a live one decays in two slopes; the early paths end at the mixing time of the listener's room. The statistical reverberation times of `inspect` count each face between two rooms on both its sides, as for one room.
- For diffraction an opening is a hole: the edges of a doorway are the free edges of the wall round it, and sound bends round them into the next room. Diffraction through a wall is left out, as it is far weaker than what passes straight through.

- The wave solver lets its air flow through openings; it does not model walls that let part of the sound through, and says so. Particles in animations pass through openings, and through partitions as often as the sound does.
- A listener in a room that no surface joins to the source's room is warned about.
- In the editor, a material's **Lets through** chooses nothing, part of the sound (the sound reduction index per band in the table) or everything (an opening).

The tests check the level behind a 30 and a 50 dB partition between two rooms against diffuse-field theory (within 0.3 dB), two rooms joined by an opening the size of the wall against one room of both their sizes, and the double-sloped decay of a dry room opening into a live one against the two-room model (within 1 dB down to -40 dB). The statistical method follows that model in both rooms, and maps over both rooms match listeners at their points. In four rooms in a row, the fourth room's response is the same whether the early part follows reflections to order 3 or not at all, and ray tracing finds the same early paths as image sources in every room.

The `apartment` example has a wall and a door, closed or open, between the living room and the bedroom of a flat.

Audience seating

Sound that reaches a listener low over rows of seats loses much more than the seats' absorption says, as it passes heads and seat backs on the way. Mark the seating's material with `audience: true` (in the editor, **Audience seating**):

```
materials:
  seats: {preset: audience_wooden_chairs_1_m2, scattering: theatre_audience, audience: true}
```

- The attenuation is the fit of Jurkiewicz, Rummler, Green, Wulfrank and Kahle ("Impact of grazing incidence attenuation on early reflection design", Proc. IOA 47 Pt. 1, 2025) to scale-model measurements over a seated audience: sound arriving at an elevation φ above the audience, having run d metres over it, keeps min(1, (3.40 sin²φ + 2.17 sin φ + 0.02)^((d - 1)/10)) of its energy. That is 17 dB less at grazing incidence after 11 m, 0.6 dB per metre at 5°, and nothing lost from 17.8° up.
- It applies to the last leg of every early path (the direct sound, reflections and diffracted sound) to listeners and map points up to 2 m above audience surfaces: φ is the leg's elevation above the surface under the listener, and d how far the leg runs over audience surfaces no more than 2 m higher than the listener's ears are above theirs, so the stalls do not shade a balcony seat.
- Only in the 500 Hz to 2 kHz octaves (third octaves from 400 Hz to 2.5 kHz), the bands of the fit, where it hardly changed with frequency. The seat dip lower down depends on the seats themselves, and the fit has no data higher up. The late part is left as it is, and the statistical method does not make up what the seats take from the early paths. Leaving out the leg from the source to the first reflection, as the fit does, errs on the side of too little attenuation.
- In a flat lecture hall with the talker at 1.6 m, it takes 10 dB off the direct sound at the back row in those bands and 1.3 dB off C50 there.

The `town_hall_pa` example marks its seats as audience.

Sources and directivity

A source has a position, an orientation (`yaw`, `pitch` and `roll` in degrees; all zero faces +x with +z up), a sound power level (`level_db`, dB re 1 pW) and a `directivity`:

```yaml
sources:
  - name: talker
    position: [2.0, 3.0, 1.5]
    orientation: {yaw: 30}
    directivity: cardioid   # or omni, subcardioid, supercardioid, hypercardioid, figure8
  - name: monitor
    position: [0.7, 1.3, 1.2]
    directivity:            # a pattern, or the a of a + (1 - a) cos(theta), per octave band
      pattern: [omni, omni, omni, subcardioid, subcardioid, cardioid, cardioid, 0.3]
  - name: loudspeaker
    position: [0.7, 3.3, 1.2]
    directivity: {file: speaker.sofa}   # measured data: SOFA or CSV
```

- A directivity says where the sound power goes, not how much: its weights average 1 over the sphere in every band. It shapes the direct sound, each early path (by the direction it leaves the source in), the rays of the late part and the particles of `animate`.
- **SOFA** files follow the AES69 `FreeFieldDirectivityTF` convention (spectra measured by microphones around the source, as published for loudspeakers and instruments), or hold impulse responses in `Data.IR`. Directions are the microphones' positions as the source sees them (`SourceView` and `SourceUp`). `measurement: 3` picks one measurement, for example one note of an instrument; by default all are averaged. Reading SOFA files needs the `sofa` extra (h5py).
- A **CSV** table has the header `azimuth, elevation` followed by frequencies in Hz, then one row per direction with levels in dB (any reference; only differences matter). Azimuth turns from the front towards the source's left and elevation is positive upwards, in degrees. Any set of directions works. Bands are averaged into octaves, and octaves with no data copy the nearest one; in third-octave runs each octave's pattern holds for its thirds. From Python, `acoustic_spaces.directivity.write_directivity_csv(path, balloon)` writes a directivity (a `Balloon`) as such a table, its octave bands on its grid.
- The directions of a SOFA or CSV file are interpolated onto a 5 degree grid, each of its points from the four directions nearest to it, or from all of them where the file has fewer. Where several are equally near and not all of them are among the four, the point takes the nearer ones alone, or all of them if none is nearer. A direction given more than once (a table's -180 and 180 degrees of azimuth, its rows straight up and down) counts once, with the mean of its energies.
- CLF (Common Loudspeaker Format) files are not read: their specification is not public. Convert them to SOFA or CSV.

`acoustic-spaces directivity speaker.sofa --plot polar.png` prints the directivity index and front-to-back ratio per octave band and draws the polar patterns; `acoustic-spaces inspect` lists every source's directivity index.

A coherent array, such as a column loudspeaker or a line array, is a source whose directivity is the sum of its elements' sound with its phase:

```yaml
sources:
  - name: column
    position: [0.5, 3.0, 2.0]
    directivity:
      array:
        element: cardioid    # each element's pattern, along its aim (default omni)
        count: 8             # a straight column along the up axis, centred...
        spacing_m: 0.15      # ...and this far apart
        steer_deg: -8        # delays that tilt the beam 8 degrees down
  - name: flown
    position: [6.0, 3.0, 6.0]
    directivity: {file: line_array.yaml}   # the same fields in a YAML file
```

```
# line_array.yaml: elements one by one, in the source's frame (x forward, y left, z up)
element: {file: cabinet.sofa}   # or a pattern, as for a source
elements:
  - {position: [0, 0, 0.0], orientation: {pitch: 0}}
  - {position: [0, 0, -0.35], orientation: {pitch: -3}, gain_db: -1, delay_ms: 0.05}
  - {position: [-0.02, 0, -0.70], orientation: {pitch: -7}, gain_db: -2}
  - {position: [0, 0, -1.05], invert: true}   # polarity inverted
```

- The elements' pressures add up with their path differences, their feeds' gains, delays and polarities, the steering, and each element's pattern along its aim (first-order patterns with their sign, measured ones by magnitude). The energy is the exact mean over each third-octave band, on a grid of 1 to 5 degrees fine enough for the beam, and with the room's speed of sound. A column of 8 takes about a second.
- The result is the array's **far-field** pattern, a balloon like a measured one, so the source stays one point: right beyond about the array's length squared over the wavelength, $L^2 f / c$ (16 m for a 2 m array at 1.4 kHz). Closer to a long array, its sound spreads as a line's and its beam has not formed yet. The room's warnings name every listener nearer to an array than that distance at 1 kHz: `validate`, `inspect`, `simulate` and `map` print them, and the editor lists them under **Properties** when the scene itself is selected. Higher up the far field starts farther still:

array length	far field from, at 1 kHz	at 4 kHz
0.5 m	0.7 m	2.9 m
1 m	2.9 m	11.7 m
2 m	11.7 m	46.6 m

- `near_field: true` (next to `array` or `file`) adds up the elements' sound one by one instead, each from where it is, with its own distance, direction, feed and pattern, wherever an early path leaves the array: the direct sound and the reflections (from the image of the listener the path unfolds to), the ground's reflection and refracted rays outdoors, and the sound that reaches an edge or a crest before it bends round it. It is exact over each band as the pattern is, becomes the pattern far away, and the warnings leave the array out. The late reverberation keeps the pattern, which carries the same power. In the editor it is "Add up the array's elements one by one" under the source's directivity.

  ```
  directivity: {file: line_array.yaml, near_field: true}
  ```

  A 3.75 m column loses about 3 dB per doubling of the distance on axis within its far field's start, as a line does, and 6 dB beyond it, as a point does.

- As for any source, `level_db` is the array's sound power in all; the pattern only says where it goes. A single element's array has the element's own pattern.
- The tests check a column against the analytic sums of a line array (the directivity index within 0.05 dB in every band, the vertical pattern within 0.3 dB), steering against delays given element by element, a dipole made of two elements in opposite polarity, and splayed cardioids against their exact sum.
- In the editor, pick the YAML file with "Measured data or an array from a file…"; `acoustic-spaces directivity line_array.yaml --plot polar.png` shows its pattern.

How loud a source is, and how it is fed:

```yaml
sources:
  - name: talker
    position: [2.0, 3.0, 1.5]
    level_db: 78            # sound power in all, dB re 1 pW
    spectrum: speech_male   # or flat (default) or speech_female
  - name: delay-speaker
    position: [12.0, 3.0, 2.5]
    level_db: [95, 98, 100, 100, 100, 99, 97, 92]   # or per band (8 or 24 values)
    gain_db: -6                       # gain of the feed
    eq_db: [0, 0, -3, 0, 0, 0, 2, 4]   # equalizer per band, dB
    delay_ms: 18.5                    # plays this long after emission starts
    signal: sounds/announcement.wav    # what it plays in a demo of the scene
    channel: mono                     # of it: left, right or mono (both mixed down)
```

The scene's `recording` is what its sources with a `channel` and no `signal` play, so a stage's loudspeakers play the two channels of a band's stereo master:

```yaml
recording: spicy_funk_cake   # a free song, downloaded the first time; or a file
sources:
  - {name: house_left, position: [0, 6, 6], channel: left}   # on the audience's left
  - {name: house_right, position: [0, -6, 6], channel: right}
  - {name: subs, position: [0, 0, 0.5], channel: mono}
```

- `level_db` is the sound power level, in all or per band. A level in all is spread by `spectrum`: evenly over the octave bands (`flat`), or as the long-term spectrum of male or female speech (IEC 60268-16), for which the STI then takes the male or female weighting. The standard's normal voice, 60 dB(A) at 1 m, is about 78 dB of sound power with either speech spectrum and an omnidirectional source. It gives no level at 63 Hz, nor at 125 Hz for female speech; those bands are taken 12 dB below the next one up, an assumption that does not reach the STI.
- `gain_db` and `eq_db` are the gain and equalizer of the source's feed. Both act on the sound power and on the impulse responses, so a mix made with `auralize` hears them. G stays the room's: it is measured against the same source, fed the same way, in free field. The impulse responses are per unit of sound power, and `auralize` scales each source's signal by its `level_db` against the loudest source's, so a mix keeps the balance between sources of different power too.
- `delay_ms` delays everything the source sends. Its impulse responses start that much later, and in the coverage map of all sources together its sound arrives that much later, as when delay loudspeakers are aligned with a stage.
- `signal` is the file the source plays in a demo of the scene (`acoustic-spaces demo`, see a demo of any scene): a dry recording, relative to the scene's folder, played at the source's sound power. A WAV file is read as it is; another format (FLAC, MP3, M4A…) needs ffmpeg on the PATH. `channel` picks what of it the source plays: `left`, `right`, or `mono`, both mixed down; without a channel it is mixed down to mono.
- The scene's `recording` is the file a source with a `channel` and no `signal` plays: a file relative to the scene's folder, or the name of one of the free recordings (`spicy_funk_cake`, `great_british_summer`, `potassium`, `speech_male`, `speech_female`; see a demo of any scene), which the demo downloads the first time and keeps for every scene. A name without a suffix must be a free recording's or a file that is there, so a mistyped free name is refused. Left and right are the recording's, so the loudspeaker on the audience's left plays `left`. A source with neither a signal nor a channel is silent in the demo. The simulation reads neither.
- `simulate` reports the sound pressure level per octave band and A-weighted (`spl_db` and `spl_a_db` in `parameters.json`), the editor's results show it per band, and `inspect` lists each source's sound power. In a band whose G is undefined, the level is that of the energy the response holds, as for the pass-by levels, and null where no sound arrives. In free field, the level 4 m from a source comes out as its sound power less 10 log10(4 π r²) = 23 dB and the air's absorption over the 4 m, to within 0.01 dB.

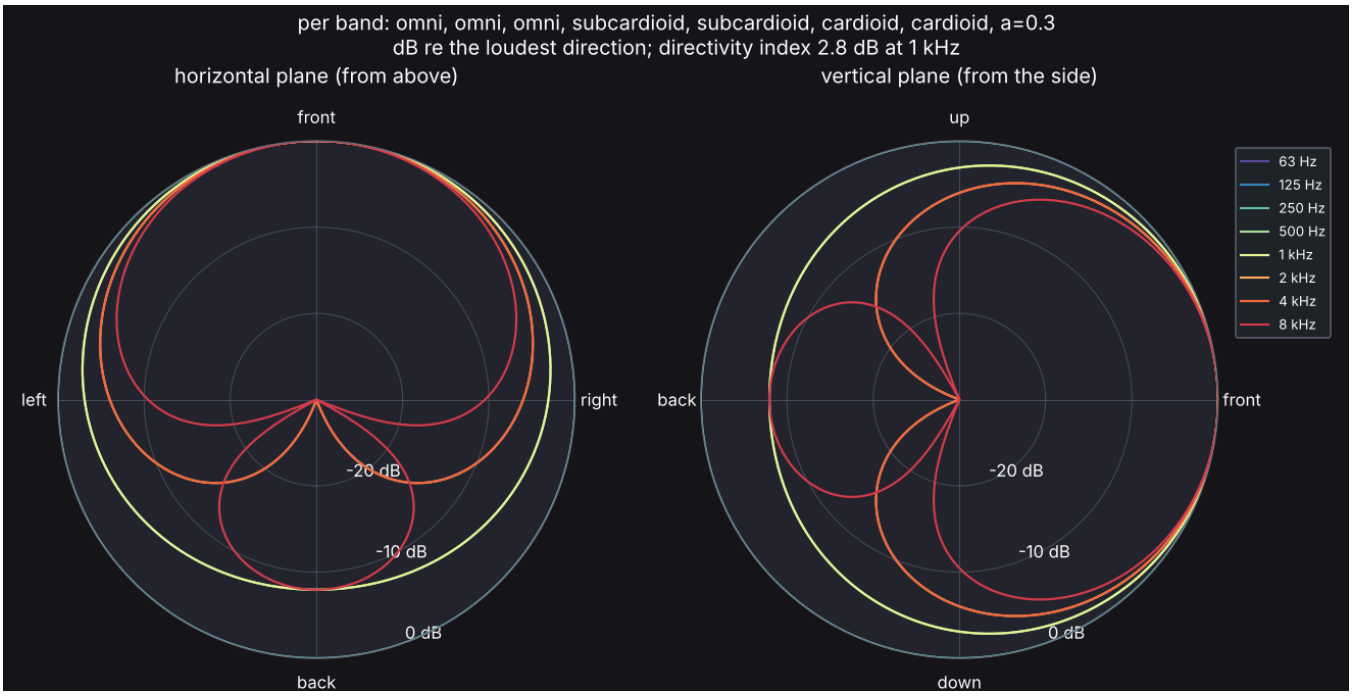
The `town_hall_pa` example feeds two delay loudspeakers with their own gain, equalizer and delay, behind two columns that are coherent arrays.

Line and surface sources, such as a road, a row of loudspeakers fed alike, a choir or a vibrating wall:

```yaml
sources:
  - name: traffic
    position: [0.0, 20.0, 0.5]
    orientation: {yaw: 90}   # the line runs along the source's left axis: here -x
    line_m: 40
  - name: panel
    position: [6.0, 0.05, 1.5]
    orientation: {yaw: 90}   # facing +y, into the room
    surface_m: [2.0, 1.2]    # width along the left axis, height along the up axis
    spacing_m: 0.25          # at most this far apart (default 0.5 m)
    directivity: lambert     # a vibrating surface's: in proportion to the cosine
```

- A line or surface is made of uncorrelated point sources, at most `spacing_m` apart, each in the middle of an equal share of it. They share the source's sound power and its rays equally, each with a seed of its own, and keep its directivity, orientation and feed. A source is made of at most 400 of them; their sound adds up in energy, as for incoherent sources such as traffic or a crowd. Close to a long line, the level falls about 3 dB per doubling of distance rather than a point source's 6 dB (3.3 dB between 2 and 4 m from a 40 m line in the tests).
- `lambert` radiates as a vibrating surface, in proportion to the cosine of the angle to the forward axis and nothing behind (a directivity index of 6 dB).
- All of a line or surface must be inside the room; `inspect` lists its length or size and its number of points. The wave solver takes point sources only. The editor draws lines and surfaces in the 3D view and the plan, and the Blender export has them too.

The `workshop` example maps the noise of a conveyor (a line), a machine's casing (a surface) and a compressor (a point).



Groups, and sources left out

Sources and listeners can sit in groups, and can be left out of a run without being removed:

```yaml
sources:
  - name: main_left
    group: PA/Mains          # the group Mains in the group PA
    position: [-40, 19, 14]
  - name: vocal_wedges
    group: Monitors
    enabled: false           # left out of simulations, maps and checks
    position: [-43.3, 0, 2.25]
listeners:
  - name: singer
    group: Stage
    position: [-44.8, 0, 3.7]
```

- `group` sorts the scene tree of the editor; groups nest with `/`, and a source and a listener may share a group's name. It changes no result.
- `enabled: false` leaves a source or listener out of `simulate`, `map`, `walk` (the sources that do not walk) and the scene's checks, as if it were not there, but keeps it in the scene; the others get the same responses as in a run of all, since seeds follow the scene's order. Named on the command line (`--source`), it is simulated all the same, and checked. `inspect` marks it "left out". With every source, or every listener, left out, `validate`, `pack`, `simulate` and the editor refuse the scene, as there is nothing to run (a map needs no listener).
- In the editor's scene tree each source, listener and group has a tick (in simulations or left out, saved as `enabled: false`) and an eye that shows or hides it in the 3D view and the plan. Hiding is how the editor shows the scene, not part of it: it is not saved and changes no result. Sources and listeners left out start hidden, and an eye follows its tick when that changes; shown by their eye, they are drawn faded.

Walks

A listener or a point source can walk through the room, for walkthroughs:

```yaml
listeners:
  - name: visitor
    position: [17.0, 4.0, 1.6]   # where the walk starts
    walk:
      to: [[4.0, 4.0, 1.6], [4.0, 16.0, 1.6]]   # the corners it goes through
      speed_m_s: 1.2       # walking pace (default 1.2 m/s)
      face_forward: true   # turn to face the way it goes (default); false keeps its yaw
      turn_deg_s: 90       # how fast it turns through a corner (default 90 degrees/s)
      doppler: false       # pitch shifts as it goes: cars, trains (default false)
```

- A walk goes in straight lines from the position through each point of `to`, at an even pace, and stops at the last one. Facing forward sets the yaw along each leg and keeps the pitch and roll; on a leg straight up or down the yaw stays as it was. At a corner the walker turns the shorter way round at `turn_deg_s`, half of the turn before the corner and half after it (no further than half-way along either leg), so a sound to one side moves round the head instead of jumping.
- `doppler: true` plays the early sound through delays that change as the walker goes, so that its pitch shifts: for vehicles and aircraft, which go faster than anyone walks (see Moving sources and Doppler). It needs a speed below the speed of sound and no wave solver.
- Everything else (`simulate`, `map`, the editor) sees the walker standing where its walk starts; `inspect` lists how far each walk goes and for how long. Lines and surfaces do not walk.

A walk may instead be timed, through `keyframes`:

```yaml
sources:
  - name: speaker
    position: [2.0, 3.0, 1.7]        # where it is at 0 s
    orientation: {yaw: 0}            # and how it faces then
    walk:
      keyframes:
        - {time_s: 4.0, position: [6.0, 3.0, 1.7]}                       # 1 m/s, facing as
before
        - {time_s: 6.0, position: [6.0, 3.0, 1.7], orientation: {yaw: 90}} # turns on the
spot
        - {time_s: 10.0, position: [6.0, 7.0, 1.7]}                      # on, facing +y
      enabled: true                  # false keeps the walk but leaves it standing (default
true)
```

- Each keyframe says where the walker is at `time_s` (seconds from the start, in order, each later than the last) and, if given, how it faces there. Between two keyframes it moves and turns evenly, so each leg's speed is its distance over its time: it can wait, turn on the spot, or go at any pace. A keyframe without an `orientation` keeps the facing of the one before (at first, the walker's own); the keyframes say how it faces, so `face_forward` and `turn_deg_s` do not apply. A turn goes the shorter way round.
- `enabled: false` keeps a walk, of either kind, but leaves its walker standing still: `walkthrough --all` and the editor's movement leave it out, and `walkthrough` picks the one walk that is enabled. Giving it by name (`--walker`) is refused.
- Several listeners and sources may each have a walk; `walkthrough --all` plays them all together on one clock (see Walkthroughs). The editor's Movement panel edits keyframes in a table and along a timeline (see Graphical editor).

Air, response length and seed

```yaml
air:
  temperature_c: 20          # default 20 °C
  humidity_percent: 50       # default 50 %
  pressure_kpa: 101.325      # default 101.325 kPa
simulation:
  ir_length_s: 2.0           # length of the impulse responses (s), default 2
  sample_rate: 48000         # of the impulse responses (Hz), default 48000
  seed: 1                    # random seed, 0 or more; unset, every run draws its own
  insertion_loss_of: [noise_barrier]  # outdoors: run again without these parts
```

- `air` sets the speed of sound and the air absorption per band (ISO 9613-1), for the rays, the wave solver and the statistical reverberation times alike. ISO 9613-1:1993 states the absorption's accuracy (±10 % to ±50 %, as the humidity goes) for any humidity and temperature a scene takes, but only for frequency over pressure from 0.0004 to 10 Hz/Pa: a pressure outside 0.794 to 157.7 kPa in octave bands (1 to 125.3 kPa in third-octave bands) is warned about. Outdoors, `temperature_c` is the temperature at the ground, and an `atmosphere` changes it with height (see Wind and temperature).
- Outdoors, `ir_length_s` counts from the latest direct sound of the run, as sources and listeners can stand far apart (see Outdoor scenes).
- A response that ends before the room has decayed is extended by its fitted decay rate when the parameters are computed (see Simulation); a longer `ir_length_s` makes longer files, and more time steps for the wave solver. `--sample-rate` on the command line overrides `sample_rate` for one run.
- With a `seed`, the late rays, the scattering and the coverage maps draw the same random numbers every run, so a scene's results repeat from run to run (`--seed` sets it for one run). A GPU draws random numbers of its own, and its results can vary slightly from run to run, as do those of the CPU and a GPU sharing the rays (see The late stage on a GPU). The editor's **Simulation** entry has these under **Response**, and the scene's own entry the air.
- Outdoors, `insertion_loss_of` names buildings, barriers or vegetation to run the scene again without, for their insertion loss (see Results outdoors); `--without` on the command line overrides it for one run, and the editor's **Simulation** entry lists the parts to tick.

Simulation

Each source and listener pair is simulated in two parts that add up without a crossfade.

Early reflections (`simulation.early`) are exact specular paths up to `max_order` reflections, with delay, arrival direction and energy per band. Candidate image sources come from deterministic specular rays (`method: ray_tracing`, the default; cost grows with `rays`, not with the number of surfaces) or from every sequence of planes (`method: image_sources`; complete, but grows as planes to the power of the order). With edge diffraction the rays' first reflections come with the faces of every diffracting edge that the source stands in front of, in its sight or not: a ray misses a face the source sees nearly edge-on, and a reflection that an obstacle cuts off still fades at the obstacle's edges. Every candidate is then traced back from the listener, so the paths found are exact either way, including paths that reflect exactly on an edge or in a corner: a face reflects as far as its edges, to within a billionth of the model's diagonal. Sound that bends round edges is added to them (see Edge diffraction).

Late reverberation (`simulation.late`) is either:

- `stochastic_ray_tracing` (default): `rays` rays per band (50,000 by default) leave the source and reflect off the surfaces as Scattering says, losing energy to absorption and air. Specular paths the early part already computed are left out (with partitions, those through two at most; the early part keeps no reflected path through more). How energy reaches the listener is set by `reception`:
  - `diffuse_rain` (default): the scattered part of every reflection radiates to each listener it can see; the rest counts when a ray crosses a detector sphere (`receiver_radius`, 0.5 to 1 m by default and kept clear of walls). Kept clear, the sphere of a listener closer than 5.6 cm to a surface (an opening or an outdoor scene's sky too) is under a tenth of that size and catches a hundredth of the rays or fewer, which the scene's warnings say. With rays that go straight (no wind or temperature gradient), the scattered sound of each reflection straight from the source is also found from the listener's side: 16384 directions from each listener, turned at random by the seed, find the surfaces it sees, and the rays and those directions share that sound by how densely each finds it (the balance heuristic of multiple importance sampling): the rays near the source, the listener's directions near the listener. The expected energy is the same, but a ray that lands next to a listener (above a ground that scatters, an aircraft overhead) brings it no burst of scattered sound.
  - `secondary_sources`: all reflected energy radiates that way, as in ODEON's late method. Much smoother tails, but it treats specular energy as diffuse: in the example box (smooth walls, absorbing ceiling) EDT comes out about 20 % shorter.
  - `detector`: classic ray tracing, everything counted by the sphere.

  In rooms with absorption in one place and smooth walls, most late energy rides on the few rays that keep clear of the absorber, and the tail breaks up where those run out. With `splitting` (on by default), a first tenth of the rays (at least 1,000, or half of them when there are fewer than 2,000) measures how much energy a ray usually carries over time; the rest are split into copies when they carry more than 4 times that and thinned out by Russian roulette when they carry less than a quarter (a weight window). The expected result is unchanged; with random scattering, the example box's late tail (0.5 to 2.5 s) misses a 1-million-ray reference by 0.5 to 2 dB per band instead of 1 to 8 dB at 100k rays (1 to 4 dB instead of 4 to 17 dB at 10k), for about 1.5 times the run time.

The rays are traced in `batches` (10 by default), each with the random streams of its share of the rays: the same rays as at once, so the same result but for rounding (which can tip a split or a roulette the other way). From 2 batches on, the parameters get standard errors, one standard deviation of how much each would change between runs with other seeds: a jackknife leaves out each batch in turn and takes the spread of the parameters. `acoustic-spaces simulate` prints the largest in JNDs from 125 Hz to 4 kHz and suggests more rays when it is over half a JND; `parameters.json` holds them all (`standard_errors`), and the editor shows each in the results table's tips, amber over half a JND. The early paths and the wave solver's bands do not depend on the seed and add none. Each batch takes its share of the listeners' gather of the first reflections' scattered sound (every tenth of its directions with 10 batches), so the standard errors hold its error too (over a ground plane, in 10 ms slots, 1.1 to 1.4 times its spread between seeds at the median). The parameters from the directions the late energy arrives from (LF, LFC, LJ, the decay times along the listener's axes) get none. Over 60 seeds in a diffuse box (2,000 rays), the root mean square of the standard errors of EDT, T30, C80 and G came within 0.90 to 1.17 of the spread between the runs from 125 Hz to 4 kHz, and within 0.8 to 1.7 over 20 seeds in the shoebox example at its 50,000 rays. With few rays in a room whose late tail rides on a few rays, one strong arrival (a ray through the listener's sphere long after the rest) comes in some runs and not in others: the standard errors are then right on average over runs but low in most of them (those of T20 and T30 about 0.5 to 0.9 of the spread in the shoebox at 2,000 rays). Batches cost time, as each batch's run sums its threads' copies of the results: 10 made the late part 10 to 16 % slower in the shoebox and concert hall examples on 4 cores, and no slower on one thread; `batches: 1` traces the rays at once, without standard errors.

Nearly all of the run time goes into ray queries: which surface a ray meets next, and whether each listener can see a reflection (see Performance).

- `statistical`: the reflected energy of Barron's revised theory, an exponential decay at the Eyring reverberation time T from the direct sound on, less what the early paths carry. Early paths end `transition_ms` after the direct sound (by default the mixing time, the square root of the volume of the listener's room in milliseconds), so the late part holds the scattered share of every early reflection, the orders above `max_order` before the transition and everything after it. The total is Barron's, c T / (13.8 V) e^(-13.8 t_d / T) for a unit source whose direct sound arrives at t_d (Barron and Lee's 31200 T / V e^(-0.04 r / T) where the direct sound is 100 / r²): the Sabine form 4 / A with Eyring's absorption area and the same exponential, which reads 0.2 dB above the classical 4 (1 - α) / (α S) at a mean absorption of 0.1 and 0.8 dB above at 0.3.

The two parts hand over by reflection order by default. Set `transition_ms` to keep early paths only that long after the direct sound.

Impulse responses are built per band: early paths become fractional-delay impulses (in the lowest bands with the phase of their reflections, see Low frequencies: early reflections in pressure), the late energy shapes a Dirac sequence with the density of reflections in the room, and a zero-phase crossover bank sums the bands. From 50 ms after the direct sound the late energy is spread over 10 ms first, so a single strong ray late in the tail does not become a click; none of it moves ahead of those 50 ms (into C50's early part) or past the end. The direct sound at r metres is an impulse of height 1/r, so levels compare between files and the energy of the direct sound at 10 m (0.01) is the reference for strength G.

Parameters follow ISO 3382-1 per octave band: EDT, T20, T30, C50, C80, D50, Ts and G. `Response.parameters()` computes them from the simulated energies; `acoustic_spaces.parameters.from_impulse_response` does the same for any mono IR (see Measuring impulse responses). A response that ends before it has decayed is extended by its fitted decay rate; one that ends before it has decayed by 20 dB leaves too much to that estimate, and all its parameters are NaN. Outdoors, where a response may hold no decay at all, G is then that of the energy it holds, and so are the levels and the STI that take it; the other parameters stay NaN (see Results outdoors). A response that ends in silence holds all its sound: its energies are those it holds, and its decay curve drops to nothing after the last sound. That drop is the end of the data, not a decay, so no decay time is fitted through it: EDT, T20 or T30 is NaN when the curve does not fall through its range before the last sound.

Each T20 and T30 comes with the checks of ISO 3382-2 (annex B) on the line it is read from: its non-linearity $\xi = 1000\ (1 - r^2)$ ‰, r being the correlation of the decay curve with time over the fit range, and the curvature of the decay, $C = 100\ (T30 / T20 - 1)$ % (`xi_t20`, `xi_t30` and `curvature`). Past 10 ‰ (`NON_LINEARITY_LIMIT`) the decay is not straight and its decay time is not to be relied on; a curvature of 0 to 5 % is usual, and past 10 % (`CURVATURE_LIMIT`) the decay is curved, likely with two slopes as in coupled rooms, which T20 and T30 each average differently. The editor's results table shows them in amber past those limits, a ξ with the T20 or T30 it puts in doubt. Air absorption follows ISO 9613-1 for the scene's temperature, humidity and pressure.

The directions the sound arrives from give more of them:

- LF, LFC and LJ (ISO 3382-1, annex A), from a figure of eight with its null on the source: the share of the early energy that arrives from the sides, which widens the source, and the level of the late lateral sound, which envelops the listener.
- IACC early and late (annex B), from the binaural response when the scene writes one (through the spherical head or a SOFA file; the first, if it writes several).
- Stage support ST early (Gade's ST1) and ST late (annex C), how well musicians hear themselves and each other, for listeners 0.9 to 1.1 m from a source; elsewhere they are empty. The standard measures them 1 m from the source, and the direct sound in their first 10 ms changes them by up to 20 lg r dB at r metres: 0.9 dB at most within that span, 6 dB at 0.5 m.
- T30 of figures of eight facing forward, left and up from the listener: where sound keeps running between two hard walls, the decay along that axis is longer.
- STI and STIPA (IEC 60268-16) and U50, at the level the source's `level_db` gives and the background noise of `coverage.noise_db`, as the coverage maps take them (NaN, like G, when a band from 125 Hz up is undefined); and the echo criteria of Dietsch and Kraak for speech and music, from the response filtered to the 1 kHz octave for speech and to the two octaves about 1.4 kHz (0.7 to 2.8 kHz) for music. ISO 3382-3's quantities for open-plan offices are not computed (the spatial decay rate D2,S, the speech level at 4 m Lp,A,S,4m, the distraction distance rD and the privacy and comfort distances): listeners on a line from an omnidirectional source given the standard's normal speech spectrum as its `level_db` per band (57.4 dB(A) at 1 m in free field) give the A-weighted level and the STI, with `coverage.noise_db` as the background noise, to read them from.

The echogram adds the arrivals in energy, as ODEON and CATT do, except the early reflections in the lowest bands, which add in pressure (see below). A figure of eight on the centre line of a symmetric room hears the two side walls' reflections arrive together with opposite signs, and they cancel in the rendered Ambisonics response. LF read from that response with `analyze --ambisonic` is then lower than the echogram's; off the centre line the two agree on average, band by band within the interference of reflections close in time.

Scattering

A surface's scattering coefficient (ISO 17497-1) is the share of the reflected energy that leaves it diffusely rather than specularly (that of a ground with an impedance is 0 unless its material gives one, see Ground impedance). `simulation.scattering` says how rays carry it out:

```yaml
simulation:
  scattering:
    method: vector     # or random
    from_size: true    # small, distant and grazed surfaces scatter more
    by_band: true      # each band follows its own scattering
```

- `vector` (default): every reflection sends the ray in the direction of the sum of its specular direction and a random Lambert direction, weighted by the scattering coefficient, as ODEON and pyroomacoustics do. The ray keeps all the reflected energy: the diffuse rain sends the scattered share to the listeners, and the detector spheres and coverage maps count the rest along the exact specular direction, on a leg of its own up to the next face, as random scattering's specular rays do, so it reaches only where specular paths go. Where the bands' coefficients differ, each band is traced with rays of its own, so a wall that scatters the bass and reflects the treble does both; one direction for all bands (pyroomacoustics takes their mean) made the example box's treble decay twice as fast as it should. For the same rays per band, tracing band by band takes 3 to 5 times as long as one ray for all bands; at the default 50,000 per band a simulation takes 1.5 to 4 times as long as version 2's 100,000 shared rays. `by_band: false` goes back to one ray for all bands, turned by their mean scattering.
- `random`: the ray reflects specularly or, with the coefficient as probability, in a Lambert direction. This was the only model up to version 2. With smooth walls a few rays then keep running between them, and the decay comes out longer. Those few rays also make the results noisy. In BRAS's chamber music hall (see Validation), at 50,000 rays per band, a few single rays keep long legs across the hall for up to 1.5 s, one of them between its end walls, and hold up to 22,800 times a typical ray's energy. The weight window cannot spread them, as its copies stay on the same path until they scatter. The decay times' standard errors then reach 1 to 2.8 JNDs at two of its ten source and listener pairs; with 1 million rays per band the largest is 0.33 JND. Where the bands' coefficients differ, each band is traced with rays of its own here too; in the example box and the lecture hall a simulation then takes about two thirds of vector's time. `by_band: false` traces one ray for all bands, as random did up to 4.0.1: it goes in a Lambert direction with the bands' mean coefficient as probability, and each band's energy is re-weighted by its own coefficient over that mean. That is right on average, but over many reflections a band's weight drifts towards zero and its energy rides on a few rays. In the BRAS rooms, with the scattering of the surfaces' size, T30's standard errors came out 15 to 180 times vector's, and the coupled rooms' reverberation chamber decayed far too fast (from its first loudspeaker, T20 at 63 Hz 3.1 s against 8.9 s measured).
- `from_size` (default on): a surface also scatters for its size, after Christensen and Rindel (2005), as in ODEON. The specular share falls below $f_w = c\,d / (2\,(w \cos\theta)^2)$ and again below $f_l = c\,d / (2\,l^2)$, where w and l are the surface's width and length, d the distance the sound came from (from the source or the last reflection, through openings and partitions it passed) and θ its angle of incidence. A large wall reflects specularly down to low frequencies, while a small panel, a distant one or one met at a grazing angle scatters. Each flat surface (the faces of one plane joined at their edges, of any material) counts as the rectangle with its area and perimeter. Early reflections lose that share too and the late part picks it up. Reflections that edge diffraction already completes keep their own coefficient: the first two orders with `first_order` diffraction (the default) or `second_order`, the first with `direct`.

In the two measured rooms of the validation report, the default model brings the reverberation time of a chamber music hall from 20 to 32 % long down to 7 to 20 %, and keeps a seminar room within 2 % from 500 Hz to 4 kHz. Without the surfaces' sizes, neither model fits both rooms.

Frequency bands

Simulations run in the eight octave bands from 63 Hz to 8 kHz by default, or in the 24 third-octave bands from 50 Hz to 10 kHz:

```
simulation:
  bands: third_octave   # or octave (default)
```

- Everything that depends on frequency follows the bands: materials, air absorption, source directivity, the scattering of a surface's size, the rays (`rays` per band) and the filter bank the impulse response is built with. With the wave solver on, it computes the third octaves of the octaves up to `max_band_hz`.
- Materials hold 8 or 24 values. An octave value holds for each of its three third octaves, and an octave simulation takes the mean of a third-octave material's three.
- Room parameters, the STI, coverage maps, plots and the editor's results stay in octave bands: each octave's energy is the mean of its three thirds'. A response's `energy()` has all 24 bands and `octave_energy()` the octaves.
- Third octaves pay where coefficients change within an octave: absorbers narrower than an octave (a Helmholtz resonator, a slot absorber), and the 8 kHz octave, across which the air absorbs about twice as much at 10 kHz as at 6.3 kHz. Its decay is then no single slope, and in the shoebox example its T30 comes out 8 % longer in third octaves. In the measured seminar room of the validation report they bring the reverberation time at 250 Hz and 8 kHz from 13 and 18 % short to within 4 %. They trace three times the rays and take about three times as long.
- `acoustic-spaces simulate --bands third_octave` sets them for one run, and the editor's simulation page has them. A material with third-octave data shows its 24 values in the editor, and `inspect` lists its octave averages.

Edge diffraction

Sound bends round the edges of walls, screens and balconies: a listener round a corner still hears the source, quieter and duller, and the level does not drop off a cliff where a wall cuts a path off. `simulation.early.diffraction` adds this to the early part with the uniform theory of diffraction (UTD, Kouyoumjian and Pathak), averaged over each octave band:

- `first_order` (default): the direct sound and the first-order reflections bend round edges, with the reflection before or after the edge. Sound that passes two edges in a row is left out: there is none behind a thick wall; where the source and the listener see each other just over both edges of a flat roof, or along a wall in line with both, the level is too low; and behind the roof with the source above its plane it is up to 6 dB too high (see the validation report).
- `second_order`: as `first_order`, and the direct sound bends round two edges in turn: over a thick wall or barrier, round both corners of a doorway's jamb, over two screens in a row. A path over two edges reflects nowhere, so over a thick barrier on a hard ground the reflections off the ground before and after it are missing, and the level comes out 1.7 to 5.7 dB low from 500 Hz up (see the validation report).
- `direct`: only the direct sound.
- `none`: specular paths only.

With `first_order` and `second_order`, `simulation.early.diffraction_reflections` says where a path over one edge may reflect: `one_side` (the default), before or after the edge, off the ground on the way to a barrier or off a façade behind it; or `both_sides`, before and after, which adds the path reflected off the ground on the way to a barrier and again behind it, the fourth of the four paths over a barrier on the ground. Over a hard ground that path carries a quarter of the sound over the barrier, and without it the insertion loss comes out up to 0.8 dB off from 1 kHz up (mostly too high) and a few decibels off in the bands below, where the four paths interfere; with it the engine is within 0.15 dB of the exact solution (see the validation report). Over grass and other soft grounds, whose reflections near grazing are weaker, the path should matter less, but that is not checked. `both_sides` pairs every reflection before an edge with every one after it, so it costs more where many faces reflect (in the `street` example about a sixth more diffracted paths and a few tens of milliseconds per listener); `--diffraction-reflections` sets it for one run of `simulate` or `map`, and the editor has it as **Reflections over an edge** under **Early reflections**.

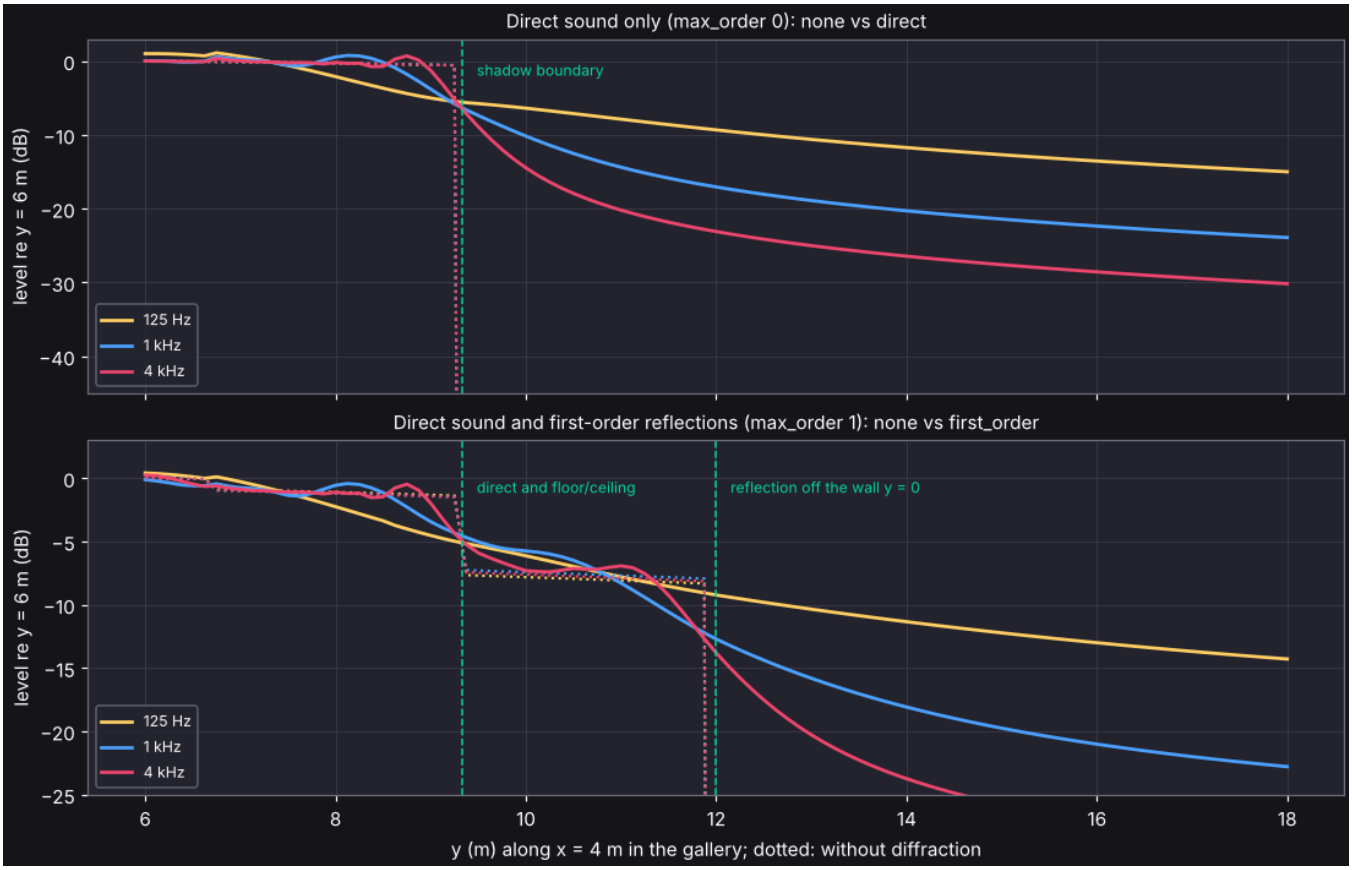
Outdoors, where sound passes over barriers and buildings on its way to a listener, paths over one edge and two do not add up to the field. There `simulation.early.rows: solver` (the default) takes the sound in the vertical plane through the source and the listener from a solver for the whole row, in the octave bands up to 500 Hz that add their paths in pressure; `rows: edges` leaves the paths over the edges alone (see Sound over rows of barriers and buildings).

The edges are found in the model: corners where two faces meet with more than 195° of open air round them (the inside corner of an L-shaped room, the front of a stage or a balcony) and the free edges of thin screens. Each source and listener get the shortest path over each edge they can both see. In the shadow of the edge that path carries the diffracted sound; on the lit side, near the boundary, it changes the energy of the specular path it borders. Across a boundary the level is continuous: the tests cross one in an L-shaped room in millimetre steps, with first-order reflections, and the level changes by less than 0.25 dB where specular paths alone drop by more than 5 dB; where a reflection off one of the corner's walls ends and the direct sound carries on, by less than 0.05 dB. Over a ground with an impedance a term reflects off it as its geometric wave does near its boundary (the whole path's spherical wave, over the grounds round that path's reflection) and as the edge's wave does farther off, so where a barrier's top or end cuts off a reflection off grass the level changes by less than 0.02 dB across it. The diffraction coefficient, its terms added in pressure, matches the exact solutions for a half-plane and for wedges (within 0.25 dB); where the paths add in energy (indoors from 250 Hz up), the level near a shadow boundary is below the exact solution: on the boundary of a right-angled corner by 1.7 dB at 250 Hz and 1 dB at 1 kHz, and more behind blunter edges (2.2 dB at 250 Hz for one open over 225°; see the validation report). A thin screen comes within 1.5 dB of Maekawa's chart in the tests' geometry (the chart, an empirical one, is itself up to 2.1 dB from the exact solution behind a barrier 40 to 90 m away).

A path over the edge of a thin screen takes the part of its Fresnel zone that the edge covers, the Fresnel integral from each end as in BTM's integral over a finite edge, so near a free end its sound fades to half at the end and out past it, over a few widths of the zone, instead of stopping there. Where two such edges meet at a right angle (a screen's top and its side) they share the fade; at a free end, where they meet at another angle, or where a screen's top bends, each of an edge's terms takes its own part of the edge, as BTM's integrand weights it near that term's boundary. An end that stands on the ground, or on a face across the edge, goes on as its image there. Past the corner of a thin screen, square or not, and beside the bend of a screen's top, the level is within 0.2 dB of the exact solution and steps as it does, where it stepped by up to 2.8 dB past a square corner, 3.0 dB past one of 45° and 16 dB beside a bend (see the validation report). The edges of solid wedges still stop at their ends: near a block's corners, where three meet, the level steps by up to 2.4 dB at 63 Hz and is up to 2.3 dB from the exact solution, and fading them as at a screen's corner put it up to 1.25 dB above it before the corner, where stopping is within 0.36 dB. Paths over two edges (`second_order`) still stop at the ends.

Behind a thick wall or barrier, whose two corners share its top, no edge sees both the source and the listener: `first_order` finds nothing there, or only a way round one corner of a doorway. `second_order` follows the shortest path over each pair of edges that see each other. The first edge's wave reaches the second as the UTD has it, with the second edge's distance parameter from the curvature of that wave where it gets there, since near the first edge's boundaries it is still curved like the source's. Where one of the two edges' waves is lit, the path continues a path over the other edge alone and shares its energy with it, so the level is continuous where a path over one edge appears: as the listener rises past the top of a barrier, it changes by at most 0.1 dB from 1 kHz up, and 0.01 dB at 63 Hz, where the waves add in pressure (0.7 dB in energy). Where both are lit, the source and the listener in sight of each other just over both edges, each edge's terms take their part off the direct sound, and the path over both gives back what they take twice: up to 500 Hz the level over a roof 10 m wide is then within 0.06 dB of the exact solution, where `first_order` is 6.4 dB low at 63 Hz half a metre above the roof's plane and 12.6 dB a centimetre above it. Against the exact solution (BTM, in the validation report), the level over barriers 0.3 to 3 m thick is within 0.3 dB from 500 Hz up, and over two screens in a row within 0.45 dB; at 63 and 125 Hz, in pressure, within 0.6 dB over the barriers and 0.13 dB over the screens (1.7 and 1.9 dB with the waves in energy).

```
acoustic-spaces simulate examples/gallery/scene.yaml --out gallery
acoustic-spaces view examples/gallery/scene.yaml --paths --listener "round the corner"
acoustic-spaces simulate examples/gallery/scene.yaml --out plain --diffraction none
acoustic-spaces view examples/archway/scene.yaml --paths --listener "behind the wall"
acoustic-spaces simulate examples/archway/scene.yaml --out archway --diffraction first_order
```



The figure walks round the corner of the `gallery` example, away from the guide. Without diffraction (dotted) the direct sound and each reflection stop at the line where the corner hides them; with it, the level falls smoothly, by more at high frequencies.

Paths with more reflections than that fade at edges too. Every specular path, found or stopped by one event, takes the UTD's term of its own boundary at each edge near each of its events: where a leg of it passes an edge, and where the point it reflects at nears the edge of its face, which a narrow face reflects only in part at low frequencies. The terms of one event add, each edge on its own side of its boundary (so both edges of a gap count where a path goes through it), and the events of a path multiply, as the path must get past each in turn. A path that is not found is followed where one edge explains each run of the events that stop it, or two edges explain them between them, in at most two runs: a reflection past the end of its face with a corner in the way of a leg next to it, say. An edge explains a reflection only in front of its face, and stands on a plane only where the plane goes on round its end on its open side, so no path reflects off a roof's plane from below beside the building. Reflections off perpendicular planes, such as the floor and a wall, give the same image in either order, and the path over an edge may take another order than the path itself: each path takes the terms of every such order, and each order is tried where the early stage found only one. Each term is a path over its edge, and each product of terms is a path of its own, which arrives later by the detours of all its terms; the terms the edge diffraction adds already are left out (`engine.edge_fade` in the API reference). Against the exact solution, past the end of a screen, the corner of a building, a strip 2 m wide and a gap 4 m wide on a hard ground, each path with its copy off the ground, the level is within 0.08 dB in every band (0.19 dB at 8 kHz on the strip, an error of the exact level's average over too few frequencies), where it was up to 7 dB off without it, and steps between listeners 1 cm apart by no more than the exact level does (see the validation report). Walking round the gallery's corner 1 cm at a time, with reflections up to third order, the level now steps by at most 0.13 dB in any band, where it stepped by up to 3.8 dB without the fade (see the validation report, which says what is left). Where the edge diffraction bends every path the early stage follows, as with `max_order: 0` and `direct` or `max_order: 1` and `first_order`, the fade changes the level by less than 0.05 dB. Steps are left where a term still stops at the end of its edge, as the fade's terms of reflections that the edge diffraction has no path for do: driving the `street` example's car past the barrier 1 cm at a time, up to 1.11 dB in one band and 0.55 dB(A). And indoors, where paths add in energy from 250 Hz up, the sound bent round edges is kept as waves (`engine.clusters` in the API reference): a term continues its geometric wave as far as it is coherent with it over the band (on the dark side of its boundary, where that wave is gone, near the boundary only), and the rest of it is part of its event's diffracted wave, whose terms add up as one wave, so the two edges of a narrow face nearly cancel as they do in pressure. A wave's energy is that of its pieces added up in pressure and averaged over the band, never below nothing. Behind the `archway`'s wall, where the reflection off the arch's jamb, 40 cm deep, appears, the level steps by at most 0.08 dB from 250 Hz to 1 kHz. The known issues list these, with what is checked and what is not. The fade comes with any `diffraction` but `none`. The late part is unchanged, since its scattered rays already find their way round corners.

`view --paths` and the Blender export draw diffracted paths in their own colour, the response's `early.diffracted` marks them, and `simulate` reports how many there are. The cost grows with the number of edges and of the paths near them, as each term of the fade is a path over its edge: in a hall with 16 edges, the early stage takes 540 ms per source and listener on four cores with `first_order`, 630 ms with `second_order`, which looks at pairs of edges, and 38 ms without diffraction (see the validation report).

Paths that reflect before or after an edge pair every edge with every plane the source or the listener faces, which in a whole city grows with the square of the buildings. Most of those pairs cannot carry a path, because the plane's faces lie nowhere near where a ray from the edge would meet the plane: the engine leaves them out before it traces a ray, on `simulation.performance.threads` threads, and the results are the same bit for bit. A city of 900 buildings simulates in about 13 s on four cores, and 3,600 in about 48 s, of which the fade's terms take about 10 and 24 s (see Outdoor scenes and the validation report).

Low frequencies: early reflections in pressure

At low frequencies reflections that arrive a few milliseconds apart do not add in energy, as an echogram adds them, but in pressure. Next to a hard floor the reflection arrives in phase and doubles the pressure of the direct sound (+6 dB, where energy gives +3 dB); a loudspeaker a metre in front of a wall loses bass where the wall's reflection arrives half a period late. A band's filter rings for longer than such reflections lie apart, so the impulse response shows that interference, and an echogram that adds the paths in energy misses it. Up to `simulation.early.coherent_max_band_hz` the early paths add in pressure:

```yaml
simulation:
  early:
    coherent_max_band_hz: 125   # 63, 125 (default), 250, 500, or null for energy throughout;
                                # outdoors up to 8000, the default there
```

- Each reflection turns the phase as a locally reacting surface with the material's absorption does: R = (cos θ - Y) / (cos θ + Y) at the angle of incidence θ, with the admittance Y fitted to the material's absorption in every band, as the wave solver fits it but without its grid, so the phase in a band does not depend on how many bands add in pressure. A hard surface reflects in phase at every angle; an absorbing one turns the phase of sound that grazes it, by half a turn at grazing incidence. Not so the sound that runs along a face from one edge to the next (`second_order`, over a roof or a thick wall): that is a point source's wave along the surface, which the plane wave's half turn would cancel; it takes the face as rigid and the ground wave's loss over the way along it, 2 F(w) against a rigid face's 2 (Weyl-van der Pol, see Ground impedance), in every band: 0.07 dB at 63 Hz to 0.76 dB at 8 kHz over a 10 m roof absorbing 0.02. Above those bands each reflection keeps the phase it has at their top, so the bands join without a step. A path over an edge does too where it arrives alone; where other paths' sound reaches it, a rendered response turns it by a phase of its own, drawn from the path and the response's seed, as far as that sound reaches it. The pieces a diffracted wave is cut into carry energies that already hold their interference, so they render their energy on average over seeds rather than adding in pressure a second time. A surface whose material has an impedance (a ground's) reflects with its spherical wave's factor instead, in every band (see Ground impedance).
- Outdoors the paths are few, and the ground dip spans several bands: the early paths add in pressure up to 8000 Hz by default there. Wind and temperature gradients bend them, and up to 2 kHz a parabolic equation gives the sound over the open ground, a narrow-angle one by default and a wide-angle one with `simulation.early.parabolic_equation: wide_angle` (see Wind and temperature). Above 500 Hz they add pair by pair, each pair's cross term as far as the band leaves the two coherent; a room takes 500 Hz at most, as its paths are many.
- Sound bent round an edge carries the whole complex UTD coefficient, whose terms add to the specular paths in pressure, as the theory has them, rather than through the cross terms that stand for that interference in the higher bands.
- The echogram reads those bands' early energy from the paths' pressure, through the band filters the analysis of measured responses uses, and so do coverage maps and the material fit. Those filters let in a little of the band above, so there each path over an edge takes the mean of its rendered phases over seeds, which needs no seed. An impulse response and the echogram it came from give the same parameters in those bands: in a 7.3 x 5.1 x 3 m box without edge diffraction, G within 0.01 dB and C50 within 0.05 dB, where adding the paths in energy leaves them up to 2.1 and 1.6 dB apart. With paths over edges they agree on average over seeds, one response scattering about the echogram.
- Outdoors in turbulence the echogram takes each two paths' coherence factor, the average over the eddies, and an impulse response is one moment of them, whose levels scatter about the echogram's (see Turbulence).
- The late part still adds in energy: its rays stand for many paths with random phases. So do the points of a line or surface source, which are uncorrelated.
- At a listener, the wave solver's bands take their place when it runs up to the same band or higher. Coverage maps do not run it, and keep the paths in pressure whatever `simulation.wave` says (see Coverage maps).

```
python examples/listening_room/bass.py
acoustic-spaces simulate examples/listening_room/scene.yaml --out energy --coherent-band none
```

In the `listening_room` example, the early part at the sofa is 5.7 dB louder at 63 Hz from a loudspeaker against the front wall than from one a metre out, and the whole response 2 dB louder; added in energy, the two are within 0.7 dB. `--coherent-band` sets it for one run of `simulate` or `map`, and the editor's simulation page has it as "In pressure up to".

Low frequencies: the wave solver

Rays treat sound as rays, which holds while a room is large next to the wavelength. At low frequencies it is not: a small room rings at its modes, so the bass booms at one seat and vanishes at the next, and sound bends round everything. With `simulation.wave`, the lowest octave bands come from the sound field itself:

```yaml
simulation:
  wave:
    enabled: true
    max_band_hz: 125           # the highest band it computes: 63, 125 (default), 250 or 500
    points_per_wavelength: 8   # grid cells per wavelength at the top of that band
    scheme: auto               # the grid's stencil: auto (default), isotropic or standard
    walls: cell_faces          # cell_faces (default), or fitted: where the model has them
    wind: effective            # outdoors: effective (default) speed of sound, or moving air
    device: cpu                # where it steps the grid: cpu, gpu or auto; left out: this
                               # computer's default (cpu unless tune made it auto)
```

- The sound field is computed with finite differences in time on a cubic grid over the room (an explicit leapfrog scheme just inside its stability limit), in the compiled core, on all cores. Up to the 125 Hz band the cells are 20 cm.
- Waves on the grid travel a little slower the shorter they are. In closed rooms the grid takes the interpolated isotropic stencil (19 points): 0.67 % slow at 8 cells per wavelength, the same in every direction within 0.03 %. Round open models it takes the standard one (7 points), the only one with the matched layer below: up to 1.7 % slow, along the grid's axes. `scheme` picks one or the other; with fitted walls (below) closed rooms take the standard one too.
- Fewer cells per wavelength make the waves slower still, along the grid's axes most: at 5 (the least `points_per_wavelength` takes), 1.85 % with the isotropic stencil and 4.56 % with the standard one. Below the default 8, the scene's warnings (`validate`, `simulate` and the editor's) say how slow they are at the top of the highest band. Each run's details give it as `wave_slow_percent`, for the grid the run took (in wind and temperature, for the slowest sound on it).
- With `device: gpu` the grid is stepped on the GPU instead, through WebGPU: Vulkan, Metal or Direct3D 12, so AMD, NVIDIA, Intel and Apple GPUs alike, with the `gpu` extra (wgpu); `auto` takes a hardware GPU when wgpu finds one, and the CPU otherwise. Left out, `device` is this computer's default (see Tuning the devices for a computer). Once `acoustic-spaces tune` (below) has timed both on the computer, `auto` takes whichever steps the run's grid as many times clearly faster: a GPU costs more to set up and less per step, so it can lose on short responses. The grid is laid out on the CPU either way, and the two agree to within about 1e-5 of the largest pressure (single-precision rounding). `acoustic-spaces info` names the GPU it finds.
- Surfaces absorb as their materials do in each band: each material becomes a locally reacting surface whose admittance changes with frequency (a constant and resonant mass-spring-damper branches), fitted so that its random-incidence absorption matches the material's in every wave band. A material with an impedance (a ground's, see Ground impedance) is fitted to its admittance itself, phase and all, with branches without mass whose corner frequencies spread an octave apart, as a ladder of resistors and capacitors makes a porous ground's admittance, which grows with frequency at a nearly constant phase: over grass the level matches the exact solution within 0.12 dB from 40 to 315 Hz. One run covers all the bands.
- Closed rooms, open models and thin screens all work. Round an open model the grid grows by a perfectly matched layer of 8 cells, which takes in the sound leaving it: what comes back is 72 to 74 dB down, against 7 to 14 dB from the grid's edge alone. Sound that runs along the layer it sends back far more, so the grid reaches past the source and the listeners, on each side the model leaves open, by 15 % of the farthest a listener lies from the source along that side (none under an outdoor ground, and none on a side that the model's own surfaces close round them).

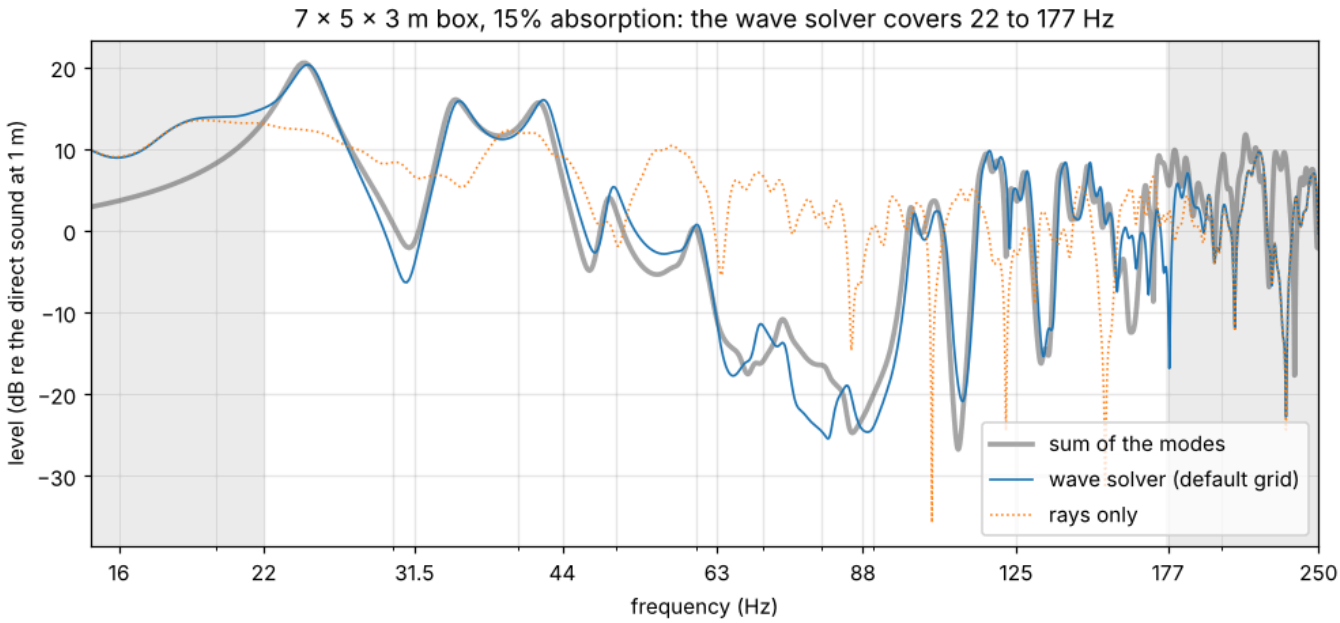
- A surface that passes through a plane of the grid’s cell centres, or within 1e-10 cells of one, acts on every link between cells and every line of sight as if it lay an infinitesimal step further along -x (else -y, else -z): a wall there stays a wall on either stencil. A source or a listener exactly on a surface, an edge or a corner is read a hair off it, on the side that step gives; where the source’s air on that side reaches no listener, or a listener’s side has no air, on a side that does (a source on a pillar’s face, a listener in a room’s corner); on an edge or in a corner, into the middle of the air round it, and on a screen’s free rim or corner, straight out past it. The points round a listener its gradient is read at move with it. A source whose only listeners lie on the plane of a surface through it, reached from both sides, is refused. Surfaces about 1e-10 cells off the centres can still be placed wrong, from a few links to a closed object read as air: see Known issues.

- With `walls: fitted` the walls lie where the model has them instead of on the cell faces nearest them: the model's faces are cut exactly through the grid's cells, and each cell they cross is stepped with the volume of its air, the open area of each of its faces and its walls' area for each material (a finite-volume form of the same update). It takes the standard stencil, indoors and in still air only (`scheme: isotropic` and outdoor scenes are refused), on the CPU or the GPU. In a rigid 4.13 x 3.37 x 2.71 m box on the 125 Hz band's grid, its 16 lowest modes come within 0.05 % of the exact ones (the grid's dispersion taken out) with the walls along the grid and within 0.38 % with the box turned by up to 14 degrees, wherever the box lies against the grid; walls on cell faces put them up to 5.2 % and 4.2 % off. The level 1.5 to 6.5 cells in front of a rigid wall is within 0.04 dB of the exact one along the grid and 0.18 dB at 3 to 14 degrees to it, where walls on cell faces are up to 1.8 dB off (8 to 16 cells per wavelength). Its limits, as measured (the figures, and how, under Walls in Validation):
  - Steeper walls: turned by 22.5 to 45 degrees, the box's modes come out up to 4.3 % high on the 125 Hz band's grid and 1.8 % on the 250 Hz band's (1.9 % elsewhere on that grid in a one-off check), where walls on cell faces put them up to 7.1 % and 3.5 % low; the level in front of a wall at 45 degrees is 0.26 to 0.40 dB off (0.32 to 0.59 dB on cell faces).
  - A room's concave edge: in an L-shaped room the lowest mode comes out from 1.4 % below to 1.0 % above the exact one on the 125 Hz band's grid (0.5 % below to 0.4 % above on the 250 Hz band's), by where the edge falls within its cell, and jumps by 2.3 % (0.9 %) as the edge crosses a plane of cell centres; on cell faces it is 1.4 % (0.5 %) below wherever the room lies.
  - Free edges: in the cell that a surface's free edge ends in, the surface is left out, so a free-standing panel comes out up to a cell shorter and an opening in a wall of a single surface up to a cell wider on each side; the scene's warnings name such edges and their length. Behind a thin screen whose edge lies inside a cell the level was up to 1.2 dB too high (rms over 8 to 16 cells per wavelength), 0.15 dB with the edge on a cell face, where walls on cell faces were 0.3 to 0.7 dB off (one-off checks).
  - Absorbing walls: a wall's reflection still depends on where the wall falls in its cell. The end of a duct whose admittance is 1 reflects up to 0.077 of the wave instead of nothing (0.044 on cell faces), and the end of a resonant material up to 0.025 more or less than the grid's own reflection off a wall there. In the studio turned 17 degrees the 125 Hz band's T20 came out 6 % longer than on a grid twice as fine (3 % with the studio along the grid), where walls on cell faces gave it 9 to 23 % shorter (one-off checks).
  - A model is refused, with the faces named, where part of its air reaches the source only past an edge that one face alone has (a crack or a T-junction), within a hundredth of a cell of it; where two faces less than 7.2 degrees apart lie within 0.0175 of a cell of each other with air on the sides that face each other (faces laid on one another, or a layer that thin); or where it comes within a cell of the matched layer. A gap narrower than a hundredth of a cell that both of its sides reach anyway is left open.
  - In one-off checks at 125 Hz, a run took 1.1 to 1.4 times as long as with walls on cell faces and the default stencil in seven of the examples, and up to 2.6 times in a small room holding a sphere of 5,120 faces, where fitting the model took most of the extra time; the fit took 0.03 to 1.1 s on the examples and 13 to 18 s on a grid of 37 million cells, and the set-up's memory peaked at 2.7 to 4.1 times that with walls on cell faces.
  - Where a face crosses a plane of cell centres, where the fit gives a piece of air that holds no cell centre to one of its neighbours, or where a solid's convex edge meets a face just off the grid's planes, the response can change suddenly as the model moves by a hair (see Known issues).

- Outdoors in wind and temperature (an atmosphere in which sound bends), the speed of sound changes from cell to cell: the still air's at the cell's height, plus the wind's component along the horizontal from the source. The grid is then fine enough for the slowest sound and steps in time for the fastest, on the CPU or on the GPU alike (see How the simulation bends sound). With `wind: moving` (`simulate --wave-wind moving`) the air moves instead, so that sound a building sends back towards the source goes against the wind, which flows round the model's solids; it takes four to five times as long over open ground, more round buildings, and runs on the CPU.
- The listener records the pressure and its gradient, so the low end keeps its direction in Ambisonics, binaural and loudspeaker outputs (to first order). The source and each point a listener samples lie between the grid's cell centres (a cell is 0.2 m on the grid up to the 125 Hz band) and take four cells a side, weighted as a cubic through their centres. Next to a floor, a wall or flat ground along the grid they leave out the cells behind it and read across it from the two cells round the point; next to a surface at an angle to the grid, an edge or an opening, and for a listener's points nearer the source than three cells, they take the eight cells round the point that it sees, weighted trilinearly. From four cells from the source on, the level does not depend on where the two lie within their cells (within 0.04 dB with the 19-point stencil). Nearer the source its sound on the grid is not a point's: the pressure is up to 2.9, 0.7 and 0.32 dB off a point source's one, two and three cells away, and the gradient's ratio to it up to 7.3, 5.4, 1.7 and 0.43 dB off, its direction up to 15, 15, 8 and 3 degrees, one to four cells away (each stencil's figures, and the 7-point one's own gain along the grid's diagonals, under Free field in Validation).
- The result replaces the rays from 22 Hz up to the top of the highest wave band (177 Hz for 125), through the same crossovers the impulse response synthesis uses. The parameters of those bands come from it too.
- The solver steps 0.2 s further than the responses last (0.4 s with third-octave bands): the bands' filters spread each arrival both ways in time, so they read a response's last 50 ms from the run after it too (The end of a response in Validation).

```
acoustic-spaces simulate examples/concert_hall/scene.yaml --out hall --wave
acoustic-spaces simulate examples/studio/scene.yaml --out studio --wave --wave-band 250
acoustic-spaces simulate examples/church/scene.yaml --out church --wave --wave-device gpu
acoustic-spaces simulate examples/room_modes/scene.yaml --out modes   # wave solver on
acoustic-spaces simulate examples/room_modes/scene.yaml --out rays --no-wave
```

The editor's Simulation page has the same switch, where to run it, and Fitted walls. Each octave higher costs about 16 times as much: on 4 cores, up to the 125 Hz band, the studio takes 0.1 s, the concert hall (9,400 m³, 3.5 s responses) 7 s, and the church (7 s responses) 13 s per source; 6 s and 11 s with `scheme: standard`, at the cost of its dispersion. Grids of more than 40 million cells are refused, and on a GPU so is a grid with a buffer larger than the GPU binds: the field's or, round an open model, the matched layer's, of nine values a cell. `python validation/wave_gpu.py` times your GPU against your CPU; Mesa's lavapipe, a Vulkan driver that runs on the CPU (and what CI tests with on Linux), took 7.5 to 9 times as long as the CPU solver in a whole run with an earlier version of the script (the validation report has the times).



In a 7 x 5 x 3 m box with 15 % absorption, the wave solver follows the sum of the box's modes from 22 Hz up; rays give the average level instead. In the 63 and 125 Hz bands, G is within 1 dB of the modes and T20 within 5 %; rays are up to 5.2 dB and 13 % off.

What it leaves out: surfaces react locally, so their absorption changes with the angle only as an impedance's does, and absorb at most 95 % at random incidence; next to a much more absorbing band, a band absorbs somewhat more than its material says (the fit reports how much, as `wave_absorption_miss`); near the top of the highest band the grid's walls absorb a little less than their admittance (0.025 less at 500 Hz with 14 cells per wavelength); with walls on cell faces (the default; `walls: fitted`, above, puts them where the model has them) the grid puts every wall on cell faces, a wall along the grid up to half a cell from the model's and one at an angle to it in steps: in a rigid 4.13 x 3.37 x 2.71 m box moved within a cell or turned against the grid (seven placements), that moves the modes of the grid's highest band by up to 11 % on the 63 Hz band's grid, 6.4 % on the 125 Hz band's (the default's) and 2.8 % on the 250 Hz band's (more `points_per_wavelength` moves them less), and from 8 to 16 points per wavelength the level 1.5 to 6.5 cells in front of a wall is 1.5 dB rms off with the wall 0.4 cell off a cell face and up to 1.2 dB rms with it at 3 to 25 degrees to the grid (19-point stencil; validation report); waves on the grid travel slower the shorter they are, at the top of the highest band with 8 points per wavelength by 0.67 % on the isotropic stencil closed rooms take and up to 1.7 % on the standard one, which open models and fitted walls take (along the grid's axes, not at all along its body diagonals; `wave_slow_percent` in the run's details), which nothing corrects, so the modes there come out up to that much low; in single precision the 19-point stencil's rounding lets a closed room's mean pressure creep, far below the bands (see The end of a response in Validation), and with fitted walls a rigid closed room's mean pressure rings, as slowly (once in 100,000 steps or more); sources radiate equally in all directions in the wave bands; and air absorption, negligible there, is left out.

Performance

`simulation.performance` says how a run uses the computer; neither setting changes the results.

```yaml
simulation:
  performance:
    threads: 0        # CPU worker threads; 0 (default) uses every core
    ray_tree: sah     # sah (default) or median
```

- `threads`: the worker threads of the ray tracer (early candidates, the late part, coverage maps, calibration), of the wave solver on the CPU, of writing the responses' files (`simulate --out`, the editor's Export), and the worker processes of the spread's draws (see Materials). Fewer leave the computer free for other work while a long run goes on; more than the computer's cores run as its cores (see Memory). The responses' files are rendered side by side, and the threads the responses leave free render the channels of each side by side too. A channel in progress holds its band signals and their spectra, which the run counts as 67 MiB for 2.5 s at 48 kHz in octave bands and 192 MiB in third-octave bands (more for a response of tens of thousands of early paths), so the channels in progress at once are held to a quarter of the computer's memory (or of its container's limit), shared between the responses rendered at once, one channel for each at least. Results do not depend on it beyond rounding: each thread sums its share of the rays, and the shares add up in another order (in the shoebox example, the late energy moved by 7×10^{-16} of its peak between 1, 2 and 4 threads, and the early paths were the same to the bit); the files are rendered to the same samples on any number of threads.
- `ray_tree`: the bounding volume hierarchy that finds the surface a ray meets, and whether a listener can see a reflection, which is nearly all of the run time. `sah` splits it by the surface area heuristic and walks the nearer box first; `median` splits it at the middle face along the longest side, as before, for comparison. Both find the same surfaces at the same distances; where a ray meets two faces at exactly the same point (a shared edge) the other tree may take the other one. In BRAS's chamber music hall (2,309 triangles, five listeners) one band of 50,000 rays takes 6.2 s on four cores with `sah` and 21 s with `median`.

`simulate`, `map` and `calibrate` take `--threads N` and `--ray-tree sah|median` to override them, `view`, `export` and `walkthrough` `--threads`; the editor's Simulation page has them under Performance. `acoustic-spaces info` prints how many cores there are. Where the wave solver steps its grid (CPU or GPU) is `simulation.wave.device`, above.

```
acoustic-spaces simulate validation/bras/CR3/scene.yaml --out cr3 --threads 3
```

A run's results do not depend on how many threads the linear-algebra library behind numpy is given, which is as many as the computer has cores unless `OPENBLAS_NUM_THREADS` says otherwise: the package adds its long sums itself, each in a fixed order (the early paths' spectra and signals, the IACC's correlations, the decay fits and the like; `acoustic_spaces.products`), and the example scenes give the same results, bit for bit, with one, two and four of those threads (all were run but `bim_office`). What it leaves to the library are the solvers (the eigenvalues behind turbulence's delays, a binaural decoder's pseudo-inverse, the fits of calibration and of a ground's impedance), whose results can round with the number of threads where their matrices are large (the eigenvalues of a 200 by 200 matrix did; nothing in the examples does), and live head-tracked playback.

Memory

Before a simulation or a coverage map starts, the memory of its late part is estimated (`engine.run_size`): the energy at every listener (or map point), band and millisecond of the response, and at a listener, every 10 ms, for each arrival direction (`late.directions`) and as each batch of rays gives it (`late.batches`), for every source until the run ends; and while a source is traced, the tracer's own. The places of a walk between those traced for their late part hold none of it: theirs is mixed from the traced places on either side when it is read, and the estimate counts the traced ones alone. On the CPU the tracer's own is a copy of the results for each thread that sums rays, 256 MiB at most together (one thread at least, however large the copy): a run whose copies would take more runs on fewer threads, and more slowly. The examples' copies take 1.6 MiB a thread at most, as the default vector-based scattering traces one band at a time; with one ray for all eight octave bands, 20 listeners and responses of 10 s would take 90 MiB a thread, so two threads. On a GPU it is the GPU's buffers, which a GPU built into the processor takes from the computer's memory. With `late.device` gpu, both or auto (auto only where there is a hardware GPU), the estimate takes the worst of the CPU, the GPU and the two together. So the memory grows with `simulation.ir_length_s`, the listeners, `late.directions`, `late.batches`, the sources and their points, and the map points (`coverage.spacing`). On the CPU the rays cost time, not memory. The 10 batches of the default add about an eighth to the late part's memory with 64 arrival directions: in the estimate, the late part of `stadium_concert` at 8 s went from 896 to 1,014 MiB and `orchestra`'s at 6 s from 668 to 758 MiB; measured on 4 threads, the whole runs went from 890 to 1,020 MiB and from 724 to 815 MiB. `orchestra`'s grew as estimated, `stadium_concert`'s by 11 MiB more, as the batches' many smaller arrays leave the memory allocator more to hold (with glibc's mmap threshold fixed at 128 KiB, that run grew by 124 MiB against the estimate's 118).

A run that would need more memory than the computer has (or than its container may use: on Linux, the lowest limit on the process's control groups and on those above them) is refused before it starts, with a message that gives what a simulation would need with each setting lowered on its own (more `late.batches` need more, not less), and the editor's messages (`Room.warnings()`) note one that would need more than half of it. A map is checked again once its first source point is traced, with the early paths found there at every point, which grow with the points too (`workshop`'s map: 311 MiB of early paths beside 495 MiB of late energy). The editor's note counts a map's late part only. `threads` above the computer's cores run as its cores: more would only take turns on them, each with its copy.

The estimate leaves out the model, a simulation's early paths, a GPU driver's own memory and the writing of the responses' files (see `threads` under Performance). The early paths take little beside the late part at a few listeners, but not in a room of many facets with edge diffraction, where the edge fade keeps the pieces of the waves round the edges in single precision: about 17 million at the third listener of BRAS CR3 in third-octave bands. Its run peaks at 10.0 GiB (11.8 GiB in third-octave bands), nearly all of it at one listener: the edge fade hands it back before the next, and in third-octave bands each listener's starts from less than 0.9 GiB. `python validation/late_memory.py` measures the estimate against what runs take (on Linux): on the CPU it came within 2 MiB of the late part's peak in `shoebox` at 1, 4 and 64 threads (4 cores), with responses of 30 to 480 s and both kinds of scattering. Whole runs, counted from the memory the process held before them, came near it or a little above: `stadium_concert` at 8 s (19 sources, 12 listeners) at 1,020 MiB against 1,014, `orchestra` at 6 s at 815 MiB against 758 (890 against 896 and 724 against 668 in one batch). Traced on a software GPU (SwiftShader's Vulkan) at 120 s, the late part peaked 59 to 66 MiB above the GPU's estimate, the driver's own; with both, at 30 and 120 s, 58 to 117 MiB above the estimate, which takes the CPU's part and the GPU's at their peaks at once.

The late stage on a GPU

The late stage's rays can be traced on a GPU, with the `gpu` extra, through WebGPU like the wave solver: Vulkan, Metal or Direct3D 12, so AMD, NVIDIA, Intel and Apple GPUs alike.

```
simulation:
  late:
    device: cpu       # cpu, gpu, both or auto; left out: this computer's default
```

- Left out: this computer's default, `cpu` unless `acoustic-spaces tune --default auto` (or OK in the editor's tune window) made it `auto` (see below). A scene that names a device keeps it on every computer.
- `cpu`: the compiled core, on `performance.threads`, as before.
- `gpu`: one compute shader traces what the CPU does (detector spheres and rain, both kinds of scattering, the scattering of a surface's size, air absorption, the early stage's paths left out, arrival directions, the weight window, and outdoors the rays bent by wind and temperature, those through vegetation and the open sky); the listeners' side of the first reflections' scattered sound is found on the CPU. Rays start where the CPU's do, then draw random numbers of their own, in single precision, so the two agree statistically, not ray for ray. Its lengths are coarser than the CPU's: a straight ray meets no face nearer its start than a millionth of the model's size (the CPU's, a billionth), and a reflection starts a hundred-thousandth of it off its face, so the CPU traces the runs with a source or a listener right by a surface (see below). The tests trace main's stored runs on the GPU and check that they land within the spread of the CPU's own results over seeds. The shader is compiled once per session with only what the scene has of the outdoors (bent rays, vegetation), in seconds. On Direct3D 12 (Windows) its compiler (FXC) takes a minute or two over either, without its optimizer, which got the vegetation wrong; it would take more than an hour over both, so the GPU leaves a scene with rays bent by the air and vegetation to the CPU there, which the run says.
- `both`: the CPU (on all its threads but one, which keeps the GPU busy) and the GPU share the rays, each taking the next part as soon as it is done with the last, so the faster one traces more (if the GPU declines a part at run time, the CPU traces it and the rest on all its threads). Which rays each traces depends on how fast each goes, so two runs differ within the noise of the ray count. `cpu` gives the same result every time, and so does `gpu` until the weight window fills a family of copies (see below).
- `auto`: the fastest of `cpu`, `gpu` and `both` on this computer, for the run's response length (a longer response keeps each ray going longer, which can change which is fastest). It takes the choice `acoustic-spaces tune` kept for that length (below). Without one, a short trial on the scene's own model traces 8,192 rays, for the run's own length, on the CPU and on the GPU, picks the fastest and keeps it for that range of lengths (up to 0.5, 1, 2, 4 s and so on). It keeps the CPU, the safe choice, when wgpu finds no GPU or only a software one, or when neither the GPU nor both together would be at least 20 % faster. `--no-device-trial` (`simulate` and `edit`) skips the trial: `auto` then takes the choice kept for the nearest length, or the CPU when there is none.

Tuning the devices for a computer

`acoustic-spaces tune` times every option and keeps the fastest for `auto`, once per computer (again after changing the GPU or its driver):

```
acoustic-spaces tune                            # the concert_hall example
acoustic-spaces tune my-hall.yaml --rays 50000  # your own scene, at your ray count
acoustic-spaces tune --lengths 1 3 8            # other response lengths (s)
acoustic-spaces tune --default auto             # time, then make auto this computer's
default
acoustic-spaces tune --default cpu              # back to the CPU, without timing anything
acoustic-spaces info                            # the default, and what auto takes now
```

- The late stage traces the scene's first source to its listeners (but those too near a surface for the GPU, see below) with the scene's settings (or `--rays` per band) on `cpu`, `gpu` and `both`, for responses of 0.5, 1, 2 and 4 s (`--lengths`), and keeps the fastest for each: the CPU unless another is at least 10 % faster. Each length stands for its range (up to 0.5, 1, 2, 4 s and so on). `auto` then takes the choice for the run's range, or for the nearest longer one, else for the longest, with no trial; `both` shares the rays by the speeds timed there.
- The wave solver steps the scene's wave grid (as set in its wave settings, whether the scene uses the solver or not) for two short responses on the CPU and on the GPU. The two times give each device's cost per cell to set up and per step, so `auto` can compare them for any run's step count; `--no-wave` leaves it out.
- It prints a table of the times and what `auto` will take, and `acoustic-spaces info` shows the choices later.
- A software GPU (lavapipe, SwiftShader) is timed too, but `auto` never takes it.
- `--default auto` then makes `auto` this computer's default: scenes that leave `simulation.late.device` and `simulation.wave.device` out run on the fastest device found. `--default cpu` switches back. Without `--default`, tune keeps the timings and leaves the default as it is (`cpu` at first).
- The choices are kept per computer (the GPU's name and the CPU's thread count) in `acoustic-spaces/devices.json` in the user cache folder (`~/.cache`, `~/Library/Caches` or `%LOCALAPPDATA%`), with the times and the default. Deleting the file forgets them.
- In the editor, Simulation > Tune CPU and GPU… times the open scene, shows the table, and asks with a check box whether to make `auto` this computer's default (ticked). OK saves the timings and that choice; Cancel saves nothing, not even the timings.
- A scene that names its devices keeps them on every computer, and with `cpu` the seed repeats a run exactly on a computer (`gpu` and `both` may vary a little); between computers the last digits can differ (the linear-algebra library's solvers, for one, differ from one processor to the next). Scenes saved by the editor before 3.1.0 name `cpu`: set **Run on** to *default* (or leave the tune window's second check box ticked for the open scene) to let them follow the default.

Sound through partitions and openings (an outdoor scene's sky, which lets sound out into no room, is not one), coverage maps and vegetation of more than 16 media are traced on the CPU whatever the setting, and so are runs whose results the GPU's buffers cannot hold (many listeners and arrival directions over a long response), and runs with a source nearer a surface than 2 millionths of the model's size, a listener so near one that the start of the GPU's reflections would cost it more than 0.1 dB of the surface's scattered sound (with the late stage's own spheres: nearer than about 47 millionths of the model's size, or than 0.6 to 1.2 mm in a model of 2 to 25 m; in a model of more than 12 km it can be less than those 47 millionths), or either nearer both sides of an inner edge than 50 millionths where the two surfaces meet at a right angle (up to 550 at sharper angles, fewer at wider ones and none from about 96 degrees on). `simulate --late-device` overrides the setting; the editor's Simulation page has it as "Run on" under Late reverberation. When it is not `cpu`, the progress messages say where the rays went and why (`talker: late rays traced on the CPU (no hardware GPU)`).

With splitting, a ray's copies on the GPU advance together, where the CPU follows each to its end before it takes the next. Where the weight window asks for more copies than a family may have (`max_family`), the GPU so makes them earlier and traces more reflections in the late tail (up to twice as many in the example box): the same expected energies, with a tail that varies less. Which of a family's copies split before it is full then depends on the order the GPU's threads come in, so two `gpu` runs of such a scene differ in the late tail, within the noise of the ray count.

`python validation/late_gpu.py` times the late stage on each device in BRAS's two rooms and two example halls, and shows how far T30, EDT and C80 come out from the CPU's next to how far a second CPU run with the next seed does; `python validation/outdoor_gpu.py` does the same for the outdoor examples (the stage in the wind, the tree belt and the hillside), with the late sound's level and centre time per listener and band. The GPU's chords through the atmosphere are tested against the CPU's ray for ray: the same turns, the same curve to the millimetre. `python validation/rays_gpu.py` times the ray searches alone, the face a ray meets next and whether a listener can see a reflection, and checks that the GPU finds the CPU's faces. The GPU works in single precision, so its boxes and triangles are widened slightly to keep rays from slipping between faces that meet; it finds the CPU's face for all but rays that meet two faces at nearly the same point.

Validation

The tests check the simulation against theory on Linux, macOS and Windows: exact image sources in a box, Eyring’s reverberation time and Barron’s revised theory in a diffusing box, ISO 3382-1 parameters of known decays, spatial encoders against closed forms, the convolvers against direct convolution, and sweep measurements of synthetic rooms. A cross-check against pyroomacoustics in four rooms agrees within 5 % on T30 with the same scattering model, from purely specular to fully diffusing surfaces. Against measurements in four rooms of the BRAS benchmark, a seminar room comes out within 2 % on reverberation time from 500 Hz to 4 kHz and within one JND on clarity, and a chamber music hall and an auditorium within about one JND on clarity and 6 to 10 % and 12 to 21 % long on reverberation time (from 250 and 125 Hz to 4 kHz), which comes within about one JND once their absorption is fitted with this package’s own rays. In two-room diffuse theory BRAS’s two coupled rooms are joined by about 1 m² more than their model from 250 Hz to 2 kHz, with the door ajar (10 to 16 times the model’s slit) and open alike, and IACC_L comes within one JND of binaural responses in 19 to 26 of 30 values per room. Edge diffraction is checked against the exact solutions for a half-plane and for wedges, against Maekawa’s chart, and over two edges against the exact BTM solution. The wave solver matches the frequencies of a box’s modes within 0.25 % and their decay rates within 2 % with walls whose admittance changes with frequency, the reflection and absorption of walls, and the free-field level within 0.04 dB from four cells from the source on (the 19-point stencil, wherever the source and the listener lie within their cells), and round an open model what comes back from the grid’s edge is 72 dB down or more. Sound through a partition reaches the level behind it that diffuse-field theory gives within 0.3 dB, and coupled rooms decay in the two slopes of the two-room model. The validation report has the details, the numbers and the scripts to reproduce them.

Spatial audio and listening

Spatial outputs

The scene's `outputs` list says which formats `simulate` writes (mono by default):

```
outputs:
  - mono
  - binaural                              # spherical head model
  - {format: binaural, hrtf: kemar.sofa}  # measured HRTFs
  - {format: ambisonics, order: 3}        # AmbiX: ACN order, SN3D, orders 0 to 7
  - {format: loudspeakers, layout: "5.1"} # stereo, 5.0, 7.0, 5.0.4, 7.0.4, and .1 with LFE
  - format: loudspeakers                  # or any layout, in channel order
    speakers: [{name: L, azimuth: 30}, {name: R, azimuth: -30}, {name: Top, azimuth: 0,
elevation: 60}]
```

Directions are those of the listener's head as its `orientation` turns it: azimuth counts to the left, elevation up, as in SOFA and AmbiX. Every early reflection keeps its exact arrival direction. For the late part the stochastic tracer sorts arriving energy into 64 directions (`simulation.late.directions`) every 10 ms; each impulse of the synthesized tail then comes from one of them, drawn in proportion to that energy. The statistical tail arrives evenly from all around.

- **Ambisonics** channels are the real spherical harmonics with SN3D normalisation, so W is the mono response exactly.
- **Loudspeakers** get vector base amplitude panning (VBAP, unit power). Layouts with height but nothing below the listener get an imaginary speaker straight down whose share is dropped. A gap of more than 90 degrees between the speakers on the horizon, as behind the listener in 5.x.4, gets imaginary speakers on the horizon that the two speakers either side play, so arrivals there pan as in 5.0. Four or more speakers on one plane, as the height ring of 5.x.4 and 7.x.4 and the rear corners of 7.x.4, get one at their middle that they all play, so a symmetric layout pans symmetrically and straight up plays from the whole ring. Named layouts carry their WAV channel mask; the LFE channel of the `.1` layouts stays silent, leaving bass management to the playback system.
- **Binaural** gives every reflection and each late direction an HRTF interpolated between the measured directions: the measured directions are triangulated as for VBAP, and the responses at the corners of the triangle a direction falls in are aligned at their onsets, mixed with its barycentric weights and delayed by the same mix of their onsets, so the interaural time difference follows the direction smoothly even with a coarse set. A response's onset is the lag at which it best matches its minimum-phase version, to a fraction of a sample, so responses delayed by fractions of a sample still add in phase and the mix keeps their spectrum (within 1 dB). Without a SOFA file it uses Brown and Duda's spherical head model: interaural time and level differences, but no pinna cues. `read_sofa` reads SimpleFreeFieldHRIR files, which most HRTF databases (for example those listed by sofaconventions.org) provide; it needs h5py.

Levels match the mono response: an arrival from straight ahead has the same height in W, in the speaker it is panned to, and, below the head's influence, at both ears.

Microphones

A listener can be a microphone: its `pattern` is what its mono output picks up through, aimed where the listener faces.

```yaml
listeners:
  - name: vocal_mic
    position: [0.0, 0.3, 1.6]
    orientation: {yaw: 180, pitch: -10}
    pattern: supercardioid        # omni (the default), subcardioid, cardioid,
                                  # supercardioid, hypercardioid or figure8
  - name: main_pair
    position: [6.0, 0.0, 3.0]
    orientation: {yaw: 180, pitch: -15}
    pattern: {pair: xy, angle_deg: 90, capsule: cardioid}
  - name: blumlein
    position: [8.0, 0.0, 3.0]
    orientation: {yaw: 180}
    pattern: {pair: blumlein}     # two figure8s aimed 45 degrees to each side
  - name: ms
    position: [10.0, 0.0, 3.0]
    orientation: {yaw: 180}
    pattern: {pair: ms, mid: cardioid, side_db: 0}
  - name: handheld
    position: [0.0, 0.5, 1.6]
    orientation: {yaw: 180}
    # A pattern or a value of a per octave band (63 Hz to 8 kHz), or 24 per third octave.
    pattern: [subcardioid, subcardioid, cardioid, cardioid, supercardioid, supercardioid,
0.3, 0.3]
```

- A first-order pattern picks up a + (1 - a) cos θ of sound arriving θ off its aim, with the `a` the sources' patterns take (0.7, 0.5, 0.37, 0.25 and 0 from subcardioid to figure8): 1 straight ahead, the same at every frequency, as an ideal capsule.
- A real capsule's pattern changes with frequency: a list gives one per octave band (8) or third-octave band (24), as names or values of `a`, for a single capsule, an XY pair's `capsule` or an MS pair's `mid` (a Blumlein pair's figure8s stay as they are). Its `a` goes from one band to the next across the crossovers of the bands the responses are synthesised in, so in the middle of a band whose neighbours agree the microphone picks up as that band's pattern. The response is rendered in two parts, what the capsule picks up whatever `a` is (cos θ) and what `a` scales (1 - cos θ), the second through a filter of gain `a` in each band; both take the same paths and impulses, so the sum is exact, and the same pattern in every band is rendered once, as a single name. Third octaves given to an octave simulation are averaged three by three, as a source's are. The report names it (`"microphone": "per band: ..."`), and the editor draws its balloon at 1 kHz. The on-axis response stays flat (1 straight ahead in every band), and measured patterns (files) are for sources only.
- A coincident pair writes two channels, left and right, in the mono output's file: `xy`, two capsules of one pattern (cardioids by default) aimed `angle_deg` apart (90 by default), the left one to the left of the front; `blumlein`, two figure8s aimed 45 degrees to each side; `ms`, a `mid` capsule aimed forward and a figure8 aimed left, decoded to M + gS on the left and M - gS on the right, g the side's gain `side_db` (0 dB by default; more is wider). The report's mono output names the microphone (`"microphone": "xy"`, say) and its channels.
- Each early path is picked up from the direction it arrives from, and each impulse of the late part from the one it is drawn from, so a pattern's response is the first order of the field the simulation gives (to the 64 directions the tracer sorts the late energy into), and the wave solver's bands are their first-order field too. A pair's channels hold the same late impulses, as two capsules in one place hear the same reverberation.

- The pattern changes what the mono output holds, in `simulate`'s files, `auralize`, walkthroughs, movements and the editor's listening; not the parameters (`parameters.json` and the editor's results stay those of an omnidirectional listener, as ISO 3382 measures them), the echo check, or the binaural, Ambisonics and loudspeaker outputs. `validate` warns about a pattern in a scene whose outputs leave mono out.

Spaced pairs (AB, ORTF, a Decca tree) are a listener for each capsule, each with its pattern and its orientation. Their direct sound and early reflections arrive with the delays between the capsules' places, but each listener's late part is drawn from noise of its own, so their reverberation does not correlate at any frequency, where real capsules d apart correlate below about c / 2d (1 kHz for 17 cm): the bass of such a pair sounds wider than it would.

HRTF sets to download

A measured head places sounds better than the spherical head model, front from back above all. `acoustic-spaces hrtf` lists the free sets this program knows, with their licences, and downloads the one named into the user's data folder (`--folder` puts it elsewhere, `--url` fetches any other SOFA file):

```
acoustic-spaces hrtf                # the sets, their licences and sizes
acoustic-spaces hrtf ku100          # download one; it prints the file
acoustic-spaces hrtf viking --member subj_K.sofa   # one head of a set of many
acoustic-spaces simulate scene.yaml --out results --hrtf ~/.local/share/acoustic-
spaces/hrtf/ku100.sofa
```

Set	Ears	Licence	Download	What it holds
`ku100`	a Neumann KU100 dummy head	CC BY 3.0	3.4 MB	2702 directions, 48 kHz, 128 taps
`kemar`	a KEMAR dummy head (SADIE II)	Apache 2.0	38 MB	8802 directions, 48 kHz, 256 taps
`viking`	a KEMAR dummy head with one of 20 pairs of replica pinnae, or none	CC BY 4.0	58.4 MB	1513 directions, 48 kHz, 128 taps

The sets are other people's measurements: `ku100` is Benjamin Bernschütz's compilation at TH Köln (https://doi.org/10.5281/zenodo.3928297), `kemar` the University of York's SADIE II database (https://doi.org/10.5281/zenodo.12092466) and `viking` the Viking dataset v2 of Spagnol, Miccini and Unnthórsson (Aalborg University and the University of Iceland, https://doi.org/10.5281/zenodo.4160401), measured at the University of Iceland, which holds one file per pair of pinnae: replicas moulded from lifelike model heads (A to S), KEMAR's own large pinnae (T) and flat baffles with no pinnae (Z). Each licence asks for its credit in a work that uses it; the command and the editor both print it before downloading. A dummy head measures the same way every time; a person's own ears place sounds better for that person and worse for everybody else, and none of these sets is a person's.

In the editor, **Download HRTFs…** on the **Outputs** page (and in the **Run demo…** window beside the HRTF field) shows the same sets with their licences, downloads the one chosen in the background and puts it in the scene's binaural output. Any other SOFA file goes in by its address in the same window, or with **HRTF file…** from the disk.

Listening: convolution and real-time playback

Run a dry (anechoic) recording through a simulated or measured response:

```
acoustic-spaces convolve speech.wav results/talker__seat-a.binaural.wav --out wet.wav
acoustic-spaces auralize results talker=speech.wav piano=piano.wav --format 5.0 --out mix.wav
acoustic-spaces play results/talker__seat-a.binaural.wav speech.wav   # real time
acoustic-spaces play results/talker__seat-a.binaural.wav --live      # microphone in
```

`convolve` takes any response WAV, with any number of channels; `auralize` reads a `simulate` folder and mixes every source's own signal as heard at one `--listener`, in one of the formats it wrote (`mono`, `binaural`, `ambix3`, `5.0`, ...), keeping the levels the simulation gives each source. The responses carry each source's feed (`gain_db`, `eq_db`, `delay_ms`) but are per unit of sound power, so each signal is scaled by its source's `level_db` against the loudest one's in the mix (from the sound powers `parameters.json` lists): a front fill 18 dB below the main loudspeakers plays 18 dB softer, each signal taken at the same level. `--equal-power` mixes them as if all had the loudest one's power; results written by an older version, which lack the powers, are mixed that way, with a note. `walkthrough` does the same. A binaural output through a SOFA file is written as `binaural-<file name>` (`talker__seat-a.binaural-kemar.wav` for `kemar.sofa`), so that it and the spherical head's can be written together; `--format binaural` finds it when it is the only binaural one. Dry signals are mixed down to mono, as sources are points, and resampled to the response's rate. The results are 32-bit float WAV files (RF64 past 4 GiB); `--normalize` scales them to a -1 dBFS peak.

`play` convolves in real time on the sound card (it needs the `measure` extra). The compiled convolver adds no latency beyond one `--block` of frames (256 by default, 5.3 ms at 48 kHz), and each block is written into buffers made before playback starts. It cuts the response into partitions that grow along it, from the block size up to 8192 frames, and spreads the work of the long ones evenly over the blocks in between (non-uniform partitioning), so long and multichannel responses stay cheap: a 3 s, 16-channel third-order Ambisonics response in blocks of 128 frames takes about 0.3 ms of the 2.7 ms each block lasts, on one current x86-64 core, where uniform partitions took 3.6 ms. By default the response is scaled to unit energy so the result plays about as loud as the dry signal; `--absolute` keeps the simulated level.

From Python:

```
from acoustic_spaces.render import Convolver, auralize, convolve, play, resample
from acoustic_spaces.spatial import Binaural

mix = auralize(result, {"talker": (speech, 44_100)}, "seat-a", Binaural())  # (samples, 2)
wet = convolve(speech_48k, binaural)  # offline, float64
convolver = Convolver(binaural, block=256)  # block by block, float32
out = convolver.process(next_block)  # (256, 2)
convolver.process(next_block, out=out)  # into an array of your own, as a callback would
play(binaural, 48_000, speech_48k)  # through the sound card
```

Dry recordings to try

These anechoic recordings are free to download, and each one tests something different:

- **BRAS anechoic audio** (BRAS, folder `4 Additional data/Audio/AnechoicAudio`; CC BY-SA 4.0; WAV, 32-bit float, 44.1 kHz): noise pulses and a string quartet with one instrument per channel. Next to them, `Audio/Auralization` holds the authors' renderings through the measured BRIRs of each room (quartet channels 1 to 4 from LS3 to LS6, the noise pulses from LS7, all heard at MP6), so a simulation of the same room, sources and seat can be heard side by side with the real one. The noise pulses make the decay and its colour easy to hear.
- **Beethoven, Symphony No. 8** (TU Berlin; Böhm, Ackermann and Weinzierl; CC BY 4.0; WAV, 32-bit float, 48 kHz): movements 1, 2 and 4 of a full orchestra, 61 mono tracks per movement (zips of 590 to 740 MB), for `auralize` with one source per instrument on a stage, as in the `orchestra` example.
- **Speech**: Cicero's 3rd Catiline Oration in Italian, Latin and German, and Demosthenes' 1st Olynthic Oration in German (TU Berlin; CC BY-NC-SA 4.0, non-commercial; mono WAV, 48 kHz), for clarity, intelligibility and echoes in lecture rooms and churches.
- **University of Iowa Musical Instrument Samples** (free for any use; AIFF, 16-bit 44.1 kHz, or 24-bit 96 kHz for those recorded since 2011): single notes of orchestral instruments, all but the piano recorded in an anechoic chamber. Short attacks (xylophone, marimba, pizzicato) make discrete echoes and any click at the convolver's partition boundaries easy to hear.
- **Aalto anechoic symphonic recordings** (Aalto University; Pätynen, Pulkki and Lokki, 2008; free for academic research; MP3, PCM on request): orchestral excerpts recorded one instrument at a time.

The commands read WAV files only (16, 24 or 32-bit PCM, or float; RF64 and BW64 too), so convert AIFF, FLAC or MP3 first, for example with `ffmpeg -i note.aiff -c:a pcm_f32le note.wav`. A dry file at another rate is resampled to the response's, with a note. A file with several channels is averaged to mono, so split a multitrack file into one file per source first; channels that are present but silent also lower the level (the BRAS files have five channels, of which the noise pulses use one and the quartet four):

```
from acoustic_spaces.audio import read_wav, write_wav

quartet, rate = read_wav("StringQuartet_4channel.wav")
for n in range(4):
    write_wav(f"quartet_{n + 1}.wav", quartet[:, n], rate)
```

A unit impulse at the response's rate checks the convolution itself: `convolve` gives the response back to float precision.

Demos

Three demos mix dry recordings through example scenes and make a video that tours their listeners, with the scripts that do it: an orchestra in the concert hall (the Beethoven recording above), a choir and an organ in the church, and a band with its sound system and monitors in the stadium.

Walkthroughs

A listener or a source with a walk can be heard as it goes. In the `gallery` example a visitor walks away from a guide, down the long arm of the L-shaped gallery and round the corner, while the guide talks (any dry speech recording does as `speech.wav`; see dry recordings to try); the gallery's two other listeners stand still, and with the visitor's walk turned off (**Moves**, or `enabled: false`) the visitor stands too, so one scene serves both:

```
acoustic-spaces examples --copy . gallery && cd gallery
acoustic-spaces walkthrough scene.yaml guide=speech.wav --out walk.wav --video walk.mp4
```

- Responses are simulated every `--step` along the path (25 cm by default, 101 places for the gallery's 25 m). The early sound, which changes quickly from place to place, is simulated at every one of them; the late field changes slowly, so the late part, by far the costliest to trace, is traced only every `--late-step` (2 m by default, 14 places), and the places in between mix the late parts of the two on either side in proportion to how near they are (with the statistical late method, which costs little, every place has its own). Along the gallery, late parts mixed from places 4 m apart, twice the default, came within 0.3 dB on average of those traced at the places themselves, band by band, and within 1.3 dB next to the corner, where the field changes fastest (with 10,000 rays per band, whose own noise is part of that). The gallery's walk takes about five minutes on 4 cores, six times as long as one response.
- The dry signals play through the response of the place the walker is nearest to, with a `--crossfade` (50 ms by default) half-way between two places, so the sound changes smoothly as it walks. In each octave band the power goes from one place's to the next's along a raised cosine, however alike the two sound: they fade along a quarter period of a cosine and of a sine, equal power for unlike sounds, and both are turned up or down by how alike they are over the crossfade (their correlation coefficient), up by at most 6 dB. A tone heard a quarter or a third of a turn later at the next place keeps its level to within 0.1 dB; in antiphase it dips over the middle 46 % of the crossfade, to nothing half-way: there the two cancel to less than half the amplitude unlike sounds would have, which 6 dB cannot make up. One response at every place gives exactly the convolution with it, and between places that differ only in level the gain changes no faster than π/2 times a linear crossfade's mean slope (the tests). Every place's late part is drawn from the same random sequence, so the tails that cross fade differ only in their level and decay, not in their noise. Once the walk is over, the walker stays where it ends for as long as the signals last.
- A walking listener hears every source in use; a walking source is heard at `--listener`, even one left out (`enabled: false`), by default the only listener in use, with the other sources in use where they stand. `--format` takes the formats of `simulate --output` (by default the scene's first output, or binaural with `--hrtf`, which is for binaural only) and `--video` saves the room, the path and the walker going along it, with an arrow and its yaw for the way it faces, as a GIF or an MP4 (both with the viz extra) at `--fps`. An MP4 carries the sound when the format has one or two channels (mono, stereo, binaural) and then lasts as long as the sound; with more channels, play the WAV alongside it.
- A walker faces the way it goes (unless its walk says `face_forward: false`) and turns through each corner at 90 degrees per second. The gallery's visitor has the guide behind it all down the long arm, and at the corner turns over a second towards the short arm, which brings the guide round to its right. The default binaural head is a sphere, which tells left from right but hardly front from behind (it has no outer ears), so the guide behind sounds much as one ahead would; a measured HRTF (`--hrtf`) tells them apart.
- At walking speed the Doppler shift is under half a percent (8 cents), and the crossfades leave it out. A car, a train or an aircraft takes `doppler: true` in its walk (see Moving sources and Doppler).
- A walk along keyframes (timed places and facings, so the walker can wait, turn on the spot or go at any pace) is placed every `--step` along each leg and every `--turn-step` (20° by default) of a turn on the spot, and at every keyframe.
- Outdoors in wind or a temperature gradient, the places of a walking source share the parabolic equation's runs, with the results of a run per place: an aircraft's flyover in the `windy` preset runs in about a third of the time (see How the simulation bends sound).

From Python:

```
from acoustic_spaces import Room
from acoustic_spaces.spatial import Binaural
from acoustic_spaces.viz import animate_walk
from acoustic_spaces.walkthrough import render_walk, simulate_walk

room = Room.from_yaml("scene.yaml")
walk = simulate_walk(room)  # the only listener or source with a walk
sound = render_walk(walk, {"guide": (speech, 44_100)}, Binaural())  # (samples, 2)
animate_walk(room, walk, "walk.gif")
```

Several movers at once

When more than one listener or source has a walk, `walkthrough --all` plays them all together, on one clock, in one simulation:

```
acoustic-spaces walkthrough scene.yaml band=song.wav speaker=talk.wav --all --out heard.wav
```

- Every walk that is `enabled` moves (the default; `enabled: false` leaves a walker standing where it starts), and every listener in use hears every source in use, still or moving, so the balance between listeners is the scene's. One WAV is written per listener (`heard_<listener>.wav`; `--listener NAME` for one of them, as `--out` names it).
- The moments simulated are every mover's places: every `--step` along its walk, every `--turn-step` of a turn on the spot, and its keyframes or corners. At each moment each mover stands as a copy of itself where its walk has it then, one copy serving every moment it stands still at (once its walk is over, say). All the copies go into one simulation with the still sources and listeners, so a moving listener costs what a walk of its own does, and the late part is traced every `--late-step` along each walk, the places between mixing theirs, as for one walk. A scene where both a source and a listener move simulates every copy of the one against every copy of the other, which grows with the square of the moments: keep `--step` coarse there, or let one of them stand.
- A pair that moves crossfades from each moment's response to the next's, as a walk does; a still pair plays through its one response. `--video` draws every mover along its path with an arrow the way it faces, and the still sources and listeners where they stand; an MP4 carries the sound when one listener is rendered.
- A walk with Doppler moves alone: `--all` plays it when it is the only walk that is on, every listener hearing each early path through its own changing delay as `--walker` does, from one simulation, and refuses it beside another mover. `--levels` is for one walk (`--walker`). The demo of any scene and the editor's **Render movement…** play a walk with Doppler the same way (demos).

From Python: `simulate_movement(room)`, `render_movement(movement, signals, fmt)` (a dict of each listener's sound), `save_movement(rendered, "heard.wav", rate, fmt)` and `animate_movement(room, movement, "heard.mp4")`. In the editor, the Movement panel edits the keyframes, previews every mover along one timeline and renders the movement in the background (see Graphical editor).

Moving sources and Doppler

A source that passes quickly, a car, a train or an aircraft, drops in pitch as it goes by, and so does a source that a listener passes quickly. With `doppler: true` in its walk, the walker's early sound follows it:

```yaml
sources:
  - name: car
    position: [-60.0, 0.0, 0.5]
    power_db: 94
    walk: {to: [[60.0, 0.0, 0.5]], speed_m_s: 25, face_forward: false, doppler: true}
```

```
acoustic-spaces walkthrough scene.yaml car=engine.wav --listener kerb --out pass.wav --levels
pass.csv
```

- Each early path (the direct sound, each reflection, the sound bent round each edge) is followed from place to place (at the next place it is the path off the same surfaces over the same edge whose delay carries on from its own, changed by no more than the time sound takes over the walker's step, with a margin) and played through a delay that changes smoothly as the walker goes: a cubic curve through the path's delay at each place, read through a windowed sinc that cuts off below the Nyquist frequency as the pitch rises, so that nothing aliases. A walking source is heard at the retarded time, when the sound it sent from each place arrives; a walking listener hears what reaches it at each place. So each path shifts in pitch as its own length changes: the ground's reflection shifts a little differently from the direct sound, and their interference sweeps as it does by a road. Each path keeps its filter (the ground's, the air's, a barrier's, a microphone's pattern, the ears' in binaural), which goes in a straight line from one place's to the next's as their sound is heard, however fast the walker passes them, but for its spreading: each sample is divided by the path's delay then, so the level follows one over the distance from sample to sample, as the pitch does. In turbulence the eddies drift past as in any walkthrough, the delay they give each path in its delay. In binaural, where each ear's response starts goes into that ear's delay instead of its filter: measured ears start a few samples apart from one direction to the next (KEMAR's far ear by up to 11 samples between places 1 m apart along a road 25 m away), and two places' filters that start apart, mixed, would cancel the treble between them. A walking listener's ear is that much further along the path; a walking source's sound reaches the ear that long after it passes the head's centre, so the time between the two ears is the head's own whatever the pitch does, and each ear's sound is divided by one plus its onset's slope (the ear's Doppler factor over the centre's).
- The late part is crossfaded from place to place by weights that add up to one, its pitch unshifted: the places' tails, drawn from one random sequence, are alike (in a shoebox room, neighbouring places 25 cm apart, each traced, correlate by 0.97 or more in every band, a one-off check), so its power runs on. A walking source's sound goes from one place's late part to the next's as it is sent, so that each place's late part follows its early paths at the retarded time; a walking listener hears the late part of the place it is at. Outdoors it is far below the early sound.
- Against the exact field of a source passing 10 m from a listener over a hard ground, the sound is within 45 dB of it at 30 m/s, with places 1 m apart, and its level within 0.02 dB in every 20 ms; a train at 80 m/s is within 39 dB and 0.42 dB at 4 kHz, most of that the air's absorption, which the engine takes at the pitch heard and that field at the pitch sent. A source passing 1 m from the listener's line at 70 m/s, its places 25 cm apart, keeps its level within 0.01 dB of the field's at 1 kHz in every 5 ms, and within 0.03 dB through a cardioid microphone and at the ears. Crossfading from place to place instead misses the level by 2.7 to 4.1 dB at most, and by 12 dB for the train at 4 kHz, where the field's two paths cancel for a moment and the places' do not (see Validation). The places should be no more than 1 m apart (the default `--step` is 25 cm): 2 m apart the sound is within 38 dB of the field, 4 m apart only within 18 dB. What depends on the direction a path arrives from (a microphone's pattern, the ears, panning, Ambisonics) goes in a straight line between two places, so a walker that passes close wants them close as the listener sees them: 7° apart left third-order Ambisonics 0.3 % from the field over a pass, 14° apart 1.3 % (a one-off check, see Validation).

- The source is quasi-static: its sound power and directivity are those of a source standing at each place. A moving source's convective amplification, which makes it louder ahead of it and quieter behind, is left out. Its size depends on the kind of source: a point monopole's far-field pressure goes as $(1 - M \cos\theta)^{-2}$, $M \cos\theta$ its Mach number towards the listener as the sound leaves, so its level as $40 \lg 1/(1 - M \cos\theta)$, which would add up to 1.6 dB straight ahead at 30 m/s and 4.0 dB at 70 m/s, and take 1.5 and 3.2 dB straight behind.
- **The sound sent each way.** From Python, the walker (or, for a walking listener, a source it passes) may send an `Emission` (`acoustic_spaces.emission`) instead of one signal: the sound it sends in every direction of its own frame, as a rotor sends each harmonic of its blades with a pattern and a phase of its own. Each early path then carries what is sent the way it leaves the source (its first leg, turning from one place's way to the next's) at the time it leaves, and the source's directivity is taken out of the path's filter; the late part plays the emission's sound averaged over the directions. `Mixture` makes one of signals that do not correlate, each with an amplitude per direction; `Periodic` one of a sound that repeats, one period of it in each direction, which may turn with a rotor and be sent from off the source's place (a helicopter's tones, see Helicopters); `Sum` adds emissions up. The directivity and the sound power still give the band levels the simulation and the pass-by report, so give the source the emission's own: its `directivity()` (written for a scene file by `acoustic_spaces.directivity.write_directivity_csv`) and its `sound_power_db(density, sound_speed)`, which the flyover demo does (a helicopter flying over). Both are in IEC 61260-1's octaves (or third octaves), taken as ideal filters between their band edges: what is sent below the 63 Hz octave's 44.7 Hz or above the 8 kHz octave's 11.2 kHz plays, but is in no band. A tone sent louder ahead ($1 + 0.8 \cos\theta$ in amplitude) and turned in phase with the height follows the exact field within 0.09 dB in level at 30 m/s and 0.42 dB for the train at 4 kHz, as one signal does there, with a cardioid directivity on the source or without (see Validation). An emission measured or computed in a moving source's frame may carry its convective amplification.
- The walker must go slower than sound, as every walk must, with Doppler or without: shock waves and sonic booms are not simulated. The wave solver's bands do not follow a walk with Doppler: with `simulation.wave.enabled` it stops with an error.

The **pass-by levels** come with it: the walkthrough of a walk with Doppler prints the listener's L_Amax and when it is reached, the sound exposure level (SEL) and L_Aeq over the pass-by, and `--levels CSV` (for any walk) saves the level over time, A-weighted and in each octave band. Each is the level of the source's sound power at a place, as `simulate` gives it, at the time its sound arrives, so its maximum and its exposure are those of noise mapping's quasi-static source; `slow_a` gives them as a sound level meter reads them with slow time weighting, as ECAC Doc 29 measures an aircraft: L_ASmax and the SEL over the time the level is within 10 dB of it (`parameters.event_levels` does so for any level over time). In a band where a place's G is undefined, as where the few rays that reach a far place carry no energy in it, the place's level is that of the energy its response holds, as outdoors (-inf where none arrives). Through the walk's sound over grass, each band's energy came within 0.26 dB of them, 0.8 dB in the band of the ground's dip, where the band levels are averages over the band and the sound is read through band filters, whose skirts fill a dip (see Validation).

```
from acoustic_spaces.walkthrough import pass_by

walk = simulate_walk(room)  # the car, with doppler: true
sound = render_walk(walk, {"car": (engine, 48_000)})
passing = pass_by(walk)  # passing.max_a, .exposure_a, .equivalent_a, .levels_a, .slow_a

# Or the sound it sends each way: signals that do not correlate (here a tone's sine
# and cosine), each with an amplitude per direction on a grid of elevations from -90
# to 90 degrees and azimuths from 0 (ahead) up to 360, turning towards the left.
from acoustic_spaces.emission import Mixture

sent = Mixture(signals, weights, 48_000)  # (K, samples), (elevations, azimuths, K)
sound = render_walk(walk, {"car": sent})
```

Head tracking

With headphones and a head tracker, a binaural rendering stays put in the room while the head turns. `play --head-tracking` takes an Ambisonics response (`--output ambisonics:3` writes `.ambix3.wav`), runs the dry signal through it in real time, turns the sound field against the head once per block and decodes it to the ears:

```
acoustic-spaces simulate examples/church/scene.yaml --out church --output ambisonics:3
acoustic-spaces play church/organ__nave-middle.ambix3.wav organ.wav --head-tracking
```

The head's orientation comes from OSC messages on UDP port 9000 (`--osc-port`) of this computer (`--osc-host 0.0.0.0` takes them from the network too, as from a phone): `/ypr` with yaw, pitch and roll in degrees, `/quaternion` (or `/quaternions`, `/quat`) with w, x, y and z, or `/yaw`, `/pitch` and `/roll` on their own, under any prefix (`/SceneRotator/ypr` too). Yaw turns left, pitch looks up and roll tilts the head to the right; `--invert yaw,pitch` flips the axes a tracker counts the other way round. Each turn crossfades over one block, so the head moves without clicks, and the latency is still the block.

The decoder fits the HRTF (a SOFA file with `--hrtf`, or the spherical head model) with the spherical harmonics up to the response's order. Above 600 Hz per order it fits the magnitudes only (MagLS, after Schörkhuber, Zaunschirm and Höldrich, 2018): with the spherical head at third order, every octave from 250 Hz to 16 kHz keeps its level within 0.1 dB, and the level difference between the ears is within 0.5 dB rms over all directions, where a plain least-squares fit (`--decoder ls`) is 16 dB short at 16 kHz.

In the editor, tick **Head tracking** in the Results panel: the yaw and pitch sliders turn the head by hand, a tracker sending to the **OSC port** turns it as you move (**Centre** takes your current orientation as straight ahead), and **Listen** plays through the head-tracked rendering.

From Python:

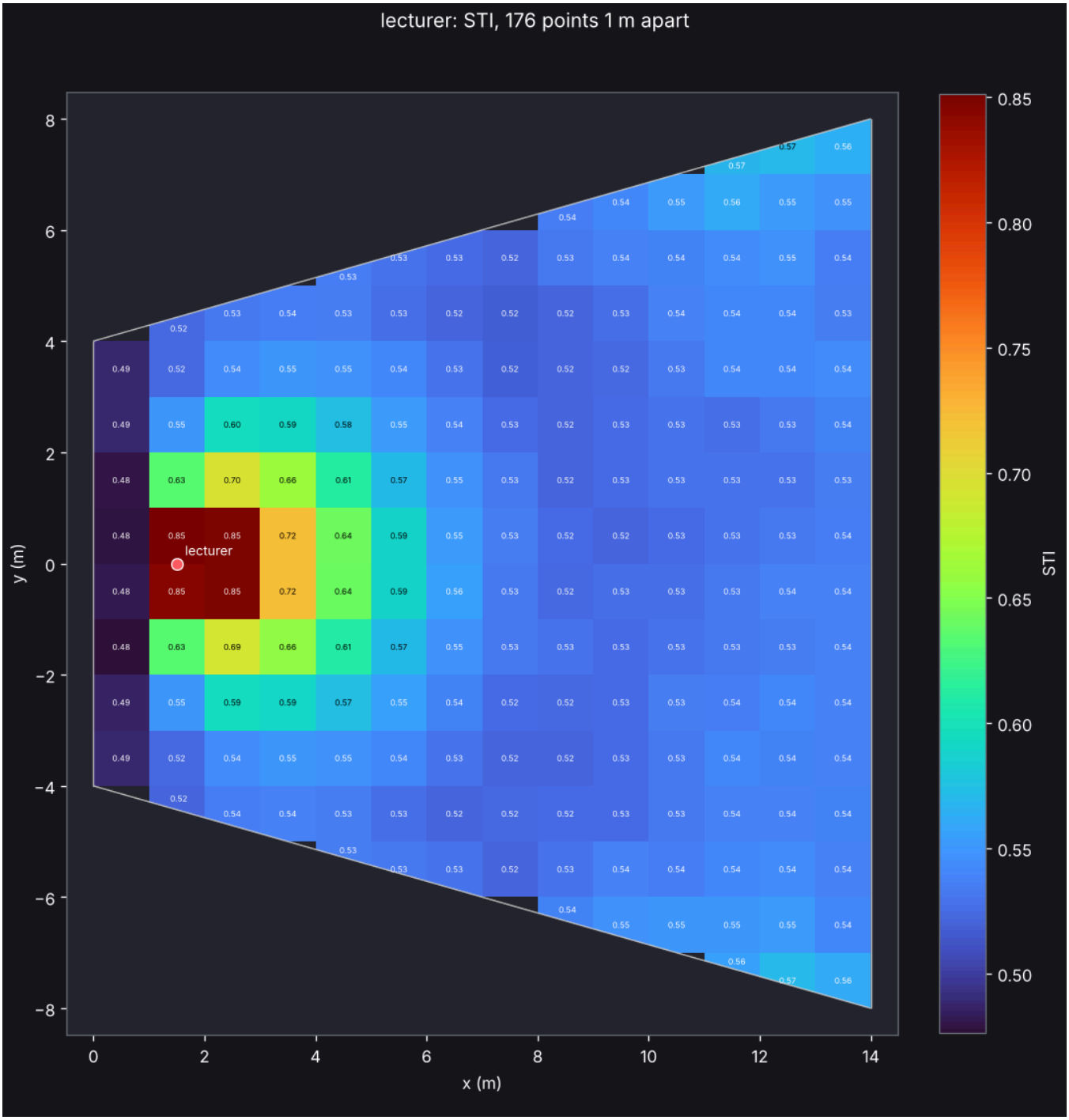
```
from acoustic_spaces.headtracking import HeadTrackedBinaural, OscReceiver
from acoustic_spaces.render import play
from acoustic_spaces.spatial import Ambisonics

ambix = response.spatial_impulse_response(Ambisonics(order=3), 48_000)
renderer = HeadTrackedBinaural(ambix, 48_000, block=256)
renderer.orientation.look(yaw=30)  # by hand, or let a tracker turn it:
with OscReceiver(renderer.orientation, port=9000):
    play(renderer, 48_000, speech_48k)
```

Coverage maps

A coverage map gives the parameters on a grid over the audience area, the way CATT and ODEON draw them: strength G and sound pressure level, C50, C80, D50, EDT, T20, T30 and Ts per octave band, and the A-weighted level and speech transmission index; outdoors also the level relative to free field, and the insertion loss of the buildings, barriers or vegetation the map was also run without (see Results outdoors).

```
acoustic-spaces map examples/lecture_hall/scene.yaml --plot sti.png --parameter sti
acoustic-spaces map examples/concert_hall/scene.yaml --spacing 0.5 --json map.json \
    --screenshot c80.png --parameter c80 --band 1000
```



The map covers the surfaces named in `coverage.surfaces` (by default those whose names contain “audience”, else the floor; outdoors, the ground, leaving out the points inside buildings) with square columns `spacing` metres wide, with a point at the listening `height` above the surface in each (above its top, when a slab’s underside is in the same group); a balcony over the stalls gets a second level of points. Columns along a slanted wall keep the part over the surface. A model that holds several rooms is mapped in every room (with the sound that passes between them), and a column that a wall between two rooms crosses gets a point on each side that holds at least a fifth of it. The early part is exact at every point, as at a listener. The late part comes from one ray tracing run per source in which every ray adds its path length through a slab of air around each point (a track-length estimate, with a vector reflection’s specular share on a leg of its own, as at a listener: see Simulation), counting only the rays in the point’s room and only the part of the slab in its air: what lies past a wall, in the next room or inside a building is left out. A map of hundreds of points costs about as much as a few listeners: the concert hall example (480 points, two sources, 50,000 rays per band) takes under two minutes on 4 cores. In the shoebox example, full simulations at three of the map’s points give G within 0.3 dB of the map and T30 within 5 % from 125 Hz to 4 kHz (the tests, with one seed); at other points and seeds T30 strays further now and then, as both carry the late rays’ noise. With the statistical late method each point gets the late part a listener there would, and G agrees within 0.01 dB.

Each source radiates its sound power per band (its `level_db`, `spectrum`, `gain_db` and `eq_db`; see Sources and directivity), which sets the sound pressure levels. The STI (IEC 60268-16, with the male weighting, or the female one for a `speech_female` source) takes that as the speech level, with auditory masking and the reception threshold, and `noise_db` as the background noise. In the map of all sources together, each source’s sound arrives its `delay_ms` later (the `town_hall_pa` example’s `delays.py` maps its sound system with and without its delay loudspeakers). Line and surface sources add up the sound of their points and arrays radiate their beams, as they do at a listener (the `workshop` example maps the A-weighted noise of both). `map` prints the range of every parameter, `--plot` saves the top view, `--screenshot` the 3D view and `--json` every value of every point. Settings live in the scene:

```yaml
coverage:
  surfaces: [audience_floor]   # default: names with "audience", else "floor"
  height: 1.2                  # listening height above the surface, m
  spacing: 1.0                 # distance between points, m
  rays: 50000                  # rays per source and band (default: simulation.late.rays)
  noise_db: 35                 # background noise for the STI, dB SPL (or one per band)
  spread: false                # also the values' uncertainty (see Uncertainty below)
```

From Python:

```python
from acoustic_spaces.engine import coverage_map
from acoustic_spaces.viz import plot_map, show_room

result = coverage_map("examples/concert_hall/scene.yaml")
both = result.get()  # all sources together; result.get("soloist") for one
c80 = both.value("c80", band=4)  # one value per point, 1 kHz
plot_map(both, "c80.png", parameter="c80", room=result.room)
show_room(result.room, coverage=both, parameter="sti")
```

Maps do not run the wave solver: whatever `simulation.wave` says, their lowest bands come from the early paths, in pressure up to `simulation.early.coherent_max_band_hz` (see Low frequencies), and the rays. For a small room's modes, put listeners where they matter and run them with the wave solver. Nor do maps give ISO 3382-3's quantities for open-plan offices (D2,S, Lp,A,S,4m, the distraction distance rD and the privacy and comfort distances); listeners on a line from an omnidirectional source with the standard's speech spectrum and level give the A-weighted level and the STI they are read from (see Simulation).

Uncertainty

With `spread: true` in the map's settings (`--spread` on the command line, the tick box under **Uncertainty** on the coverage map's page in the editor) the map's values get the standard uncertainty that the materials' absorption uncertainty leads to (see Materials), an insertion loss and the long-term levels too. A map keeps no response at its points to re-weight, as a listener's spread does, so it follows the law of propagation of uncertainty (GUM, JCGM 100:2008, 5.1.2): for each material with an uncertainty the map is traced twice more, on the same points and along the rays of the map itself, with its absorption one standard uncertainty up and down. Half the difference of the two maps is that material's part, and the root of the sum of the parts' squares the standard uncertainty of each value at each point. A map with M such materials takes about 2 M + 1 times as long. A value that one of the runs leaves undefined (a decay too short for a T30, say) has no uncertainty.

`map` prints the least, the median and the largest standard uncertainty of every parameter beside its range, and `--json` writes every point's with each material's part (`uncertainty` in each map, null where a value has none). `--show standard` draws the standard uncertainty in place of the value, `--show low` or `high` an end of the 95 % interval and `--show at_least` or `at_most` a one-sided bound (of 95 %, or of `simulation.uncertainty.one_sided`, `--one-sided` on the command line); the editor's map panel has them under **Show**. The interval and the bounds take the result as normal about the map's value: the value less and plus 1.96 standard uncertainties, or 1.645 for a one-sided 95 %, held to what the parameter can be. From Python, `acoustic_spaces.engine.sensitivity.coverage_map(scene, spread=True)` gives each map its `spread`, and the map's `spread_of(parameter, part, band)` the values that `--show` draws.

The law is first order, so where a value bends over a material's uncertainty the standard uncertainty comes out low. In a box of 8 by 6 by 3.5 m, against the Monte Carlo of a listener at one of the map's points (2 000 draws): with standard uncertainties of 0.02 to 0.03 on its three materials every parameter's is within 6 %; with a tolerance of 0.1 on each, the A-weighted level's is 4 % low, the STI's 3 %, D50's 5 %, a band's level's, G's, C50's and C80's up to 8 %, T30's up to 13 % from 250 Hz (below that the listener's draws leave T30 undefined) and T20's, EDT's and Ts's up to a quarter, at 63 Hz.

The rays' random error is not counted, as it is in a listener's spread, nor what a listener's spread leaves out. The uncertainty has that error itself: where a value changes from one seed to the next, so does its uncertainty, and a material's small part of it more. Between two seeds of the lecture hall example's map, T30 at 1 kHz differs by 0.12 s (root mean square over its points) and its standard uncertainty, 0.23 s at the median, by 0.05 s; the standard uncertainty of the A-weighted level, the STI, C50 and G differs by 1 to 3 %, and EDT's by 5 to 12 %. With four times the rays T30's two figures fall to 0.05 s and 0.03 s, and the others' to about 1 %, EDT's to 3 to 10 %.

Outdoor scenes

Sound outdoors: a street, a square, a courtyard, a stage in a park, a noise barrier by a road. A scene is outdoors when its geometry has a `ground`: a flat ground under an open sky, with buildings, barriers, trees and hedges and the objects of a model file standing on it.

```yaml
geometry:
  ground: {margin: 30}          # flat, at z = 0, 30 m past everything on it
  buildings:
    - name: houses
      footprint: [[-30, 6], [10, 6], [10, 16], [-30, 16]]   # corners, in order, any shape
      height: 9
  barriers:
    - name: noise_barrier
      path: [[14, 6], [44, 6]]  # points along the ground
      height: 3
assignments:
  ground: asphalt
  houses: brick
  barriers: concrete
```

See `examples/street/scene.yaml`: traffic along a street between two buildings, a balcony, and a garden behind a noise barrier.

The ground and the sky

The ground, four walls of sky round its edge and the sky overhead close a box, so the open air is a room like any other: its volume is known, sources and listeners are checked to stand in it, and the tracers know which side of every face the air is on.

- `ground.height` is the ground's height (z), 0 by default.
- `ground.extent: [[x_min, y_min], [x_max, y_max]]` is where the ground ends. Without it, the ground reaches `ground.margin` (30 m by default) past the buildings, barriers and the model file's objects. A scene with none of them needs an extent.
- `ground.sky` is how high the sky is over the ground; without it, `margin` over the highest building, barrier or object.

The sky is the material `sky`, an opening: everything that reaches it passes through and leaves the scene, as sound leaves into the open. Nothing reflects off it, its edges diffract nothing and its edge with the ground does not either, and the ground scatters nothing for its size (as a surface of limited size does at low frequencies, see Scattering), so the ground reads as if it went on for ever: a source over the ground has its direct sound and one reflection off the ground, and nothing else. A scene may give the sky a material of its own (`assignments: {sky: ...}`); otherwise the scene file leaves it out.

The ground takes a material as any surface does. A material with an `impedance` (grass, soil, snow, gravel, asphalt) reflects sound as a real ground does, with the phase and the ground wave that make the dip over soft ground (see Ground impedance); one with absorption coefficients alone reflects as a floor does. Parts of the ground can have materials of their own: a road, a lawn, a car park (see Ground zones).

The ground and the sky are the surfaces `ground` and `sky`.

Terrain

The ground need not be flat: `ground.terrain` gives its heights on a square grid, from a file or in the scene, and everything stands on it.

```
geometry:
  ground:
    terrain: {file: hills.asc}        # an Esri ASCII grid, or x y z points on a grid
    map_origin: [552000, 5806000]     # the map's point that becomes the scene's (0, 0)
    sky: 60
```

- `terrain.file` is an Esri ASCII grid (`.asc`, what GIS tools write and `gdal_translate -of AAIGrid` makes of a GeoTIFF or any other raster) or a table of `x y z` points on a regular grid (`.xyz`, `.txt` or `.csv`). Its coordinates must be in the geometry's units on a plane (a projected system such as UTM), not degrees of longitude and latitude; `ground.map_origin` is subtracted from them, so that the scene's coordinates stay small, and from a buildings file's (see Buildings from a map).
- `terrain.heights` gives the heights in the scene instead: rows of values, each row along x from `origin`, the rows along y, `spacing` apart.
- `ground.height` is added to the terrain's heights.

Between the grid's nodes the ground is bilinear; past the grid's edge it goes on at the height of the edge. The scene's ground is the grid sampled over the ground's extent and cut into triangles: the whole grid unless `extent` is set, which must then lie inside it (the terrain sets the ground's size, not the margin). The sky is `sky` over the terrain's highest point, or `margin` over the highest building, barrier or object. Ground zones take the triangles whose centres lie in them. Buildings and barriers stand on the terrain however it runs: a building's walls go down a little below the lowest ground under them and its roof lies its height over the terrain's mean height at the footprint's corners; a barrier's top follows the terrain along its path; vegetation stands on the terrain under its middle. The model file's objects are taken as they are, and one that reaches below the terrain's lowest point is refused. A terrain with wind or temperature gradients is not supported yet: the scene is refused.

`acoustic-spaces inspect` describes the terrain (its points, its heights and its grid) and gives each source's and listener's height over the ground under it. Coverage maps follow the terrain: their points stand at the map's height over the ground under them.

How the simulation takes the terrain into account

- **The late rays** bounce off the terrain's triangles as off any surface.
- **The early paths** do not reflect off its triangles: a facet a few metres across would send a 5 m wave back as a plain does, and as a listener moves the reflections off neighbouring facets would come and go in jumps. In sight of the source the ground reflects once instead, off the **mean plane** of the terrain's profile between the source and the listener, as the outdoor standards (CNOSSOS-EU, Nord2000) have it: the plane through the least-squares line of the profile, level across it, with the reflection factor of the ground under the reflection point (its material, or the grounds of the zones its Fresnel zone covers, as on flat ground). The line is fitted to the whole profile, segment by segment, as CNOSSOS-EU fits it, so it follows the ground's shape, not where the edges of the terrain's triangles cross the profile. Over a plain or a plane slope that is the ground itself, and the reflection is the one flat ground gives.
- **Crests** between the source and the listener bend the sound over them. The ground's profile in the vertical plane between the two is drawn, and the string pulled taut over it from one to the other bends at the crests in the way: each is a wedge whose faces run from it down to the ground under its neighbours on the string (the source's foot, the listener's or the next crest), so a ridge of two slopes is the wedge it is and a rounded hill the wedge of its chords. The sound over them is the uniform theory of diffraction's, wedge after wedge: the direct wave over each edge and the ground's reflections before and after it, off the profile's segments between the crest and its neighbours on the string, each weighted by the share of its Fresnel zone the segment holds (Nord2000's Fresnel-zone weights, so that at low frequencies they add up to the one reflection off the side's mean plane and at high ones the segments that mirror the sound to the edge carry it all), with the ground's reflection factor (the spherical wave's over a ground with an impedance). Near the crest a reflection is a term of the wedge's coefficient, as in the exact solution; far from it, a path of its own, as in the four-path model of a barrier, and the uniform theory's transition over its extra path bridges the two. Where the string runs along the ground over a flat or rounded top, the wedge sits where the string's two legs onto it would meet above it (the equivalent edge of a thick screen), and the crest screens more the wider it is, by ISO 9613-2's factor for a thick screen (up to 4.8 dB more at the highest bands): its width is the ground within an eighth of a wavelength of the string, less what a sharp wedge would have, so a ridge has none and a rounded hill a few metres at the highest bands. The path over the crests arrives from the last crest after the string's length, with the air's attenuation over it.
- **In sight of the source**, the crest the ground comes nearest the line of sight still diffracts: its wedge's waves are added to the direct sound and to the reflection off the slope before it (where the listener sees the source in that slope), with the uniform theory's transition at each boundary, so the level is continuous across the edge of a hill's shadow and ripples in the lit zone as the exact field does. The mean plane's reflection takes over from the slope's as the crest drops below the line of sight, by half the longest wavelength of path difference.
- Buildings, barriers or model objects standing on the string stop the path: the sound bent over a hill and a building in turn is not modelled. Edges buried in the terrain do not diffract, nor do its own ridges, which bend too little at each grid line: the sound over the crests takes their place.

A ridge agrees with the exact solution for the wedge within 0.1 dB, a flat terrain and a plane slope with the flat ground and the image source, and a smooth hill over a soft ground lets 1.4 to 1.8 dB more over its crest than the thin screen's attenuation (see Validation); a rounded crest, which the sound creeps over, has no exact solution, and is an engineering model here as in the standards.

Buildings and barriers

A building is its `footprint` (corners in order, as a floor plan) raised to its `height` over the ground: walls and a flat roof, open underneath, where it stands on the ground. Its surfaces are named as a floor plan room's walls are, after the building:

Surface	What it is
`<name>_roof`	its roof
`<name>_wall_1`, `<name>_wall_2`, …	the wall from the footprint's first corner to its second, and so on
`<name>_walls`, `<name>`	its walls; all of it
`walls`, `roofs`, `buildings`	those of every building

A barrier is a thin screen along a `path` of two or more points, from the ground to its `height`, reflecting on both sides: surfaces `<name>` and `barriers`. Buildings and barriers without a `name` are `building_1`, `barrier_1` and so on. The names `ground`, `sky`, `walls`, `roofs`, `buildings` and `barriers` are taken, and no two may share one.

A model file (`geometry.file`, with its `units` and `up_axis`) adds what it holds: buildings of any shape, a stage, a bandshell, terrain features. Its objects stand on the ground and must not reach below it; faces lying on the ground (a building's floor, the file's own ground plane) are left out, so they are open underneath too. With an `extent`, everything must lie inside it. The file's objects are taken as they are: a point inside one is not reported, and the inside corners of a building drawn in the file may diffract, which those of `buildings` do not; give buildings as `buildings` where you can.

Sources and listeners must stand in the open air: above the ground, under the sky, inside the extent and not inside one of the `buildings`; `validate` and the editor say which one is not.

Sound over rows of barriers and buildings

Where sound passes close over several tops in a row (the roofs along a street, a barrier and the house behind it), the paths over one edge and two do not add up to the field, and a path over two edges reflects nowhere (see Known issues). With `simulation.early.diffraction` on and reflections of the first order or higher, the early stage takes the sound in the vertical plane through a source and a listener from the solver for whole rows instead, in the octave bands up to 500 Hz that add their paths in pressure: `simulation.early.rows: solver`, the default (`--rows` for one run of `simulate`, **Barriers and buildings** under **Early reflections** in the editor). `rows: edges` leaves the paths over the edges alone. The solver marches a parabolic equation over the row, with the ground and the roofs in it, and adds what the row's rigid edges shed; against a wave calculation, two flat roofs on a rigid ground are within 0.74 dB in a simulation that takes it, where the paths over the edges are up to 10.7 dB off (see Validation).

- **What gives way.** The row's own paths: the direct sound, its reflections off the ground, the roofs and the walls that the way crosses between the two, and the paths over tops that cross the way between them. One path carries the solver's sound in their place, along the string pulled taut over the tops, and counts among the diffracted paths. The other paths stay: round the ends of barriers and the corners of buildings, off walls behind the source, past the listener or beside the way, over a model file's objects, and through a barrier or a wall that lets sound through. The solver marches one way, so sound that goes to and fro between the row's walls is not in its field.
- **Near the ends of a top.** The solver's row is the same all along its edges. Where the way crosses a top, or its line past its end, closer to an end than the edges' own terms follow that end, the solver's sound and the row's paths are added in pressure, each with its share, and closer than two thirds of that distance the paths stand alone: for a top across the way, 10 m from the source and 20 m from the listener, that is within 9.8 m of an end in the 63 Hz band and 3.5 m in the 500 Hz band, and the solver's sound is whole from 14.8 and 5.2 m on. There a barrier or a building less than 19.6 m long keeps its paths alone in the 63 Hz band, and one 30 m long has the solver's sound whole in every band near its middle only. In the `street` example, whose noise barrier is 30 m long, no band's level changes by more than 0.01 dB with it.
- **Materials.** The solver's barriers and roofs are rigid, whatever their materials absorb or let through, and its ground has each zone's impedance. A roof on the way whose material has an `impedance` leaves the row to the paths. A ground given by its absorption alone has the admittance that gives that absorption, as in the weather's runs, where the paths reflect off it with one magnitude at every angle: 2 to 90 m behind a long barrier 3 m high, 10 m from a source 1 m up, the solver's level is within 0.28 dB of the paths' over a rigid ground, 0.35 dB with an absorption of 0.02 and 1.8 dB with 0.1 (give an outdoor ground an impedance, see Ground impedance).
- **In wind and temperature gradients** the solver's sound is that in still air, and the weather's effect behind the barriers and buildings scales it as it scales the paths in the plane (see How the simulation bends sound).
- **Where the paths stand as they are.** Over a terrain; with an object of a model file in the straight line between the two; in the bands above 500 Hz; in each octave band in which the solver does not take the row (what it does not hold is listed under Validation); in coverage maps, with thousands of points; and along the walk of a source or a listener, with a place every 0.25 m, where the level would step between a place that the solver takes and the next that it does not.

- **Time.** Behind one long barrier the solver takes 1.5 to 1.9 s for a source and a listener 25 to 280 m apart (four octave bands, one core). The `railway` example, whose listeners behind the barrier take the solver's sound for part of the line source, takes about a sixth longer.

Buildings from a map

`geometry.buildings_file` adds the buildings of a GeoJSON file (`.geojson` or `.json`) to the scene's own: each `Polygon` feature, and each part of a `MultiPolygon`, is a building with its outer ring as the footprint and its height in a property.

```
geometry:
  ground:
    map_origin: [552000, 5806000]
  buildings_file:
    file: town.geojson
    height: height          # the property with each building's height, or one height for all
    default_height: 9       # for buildings whose property is missing
    name: name              # the property with each building's name; else building_<n>
    without: [old_mill]     # buildings of the file to leave out, by name
```

The coordinates must be in the geometry's units on a plane (a projected system such as UTM); a file in degrees of longitude and latitude is refused, with a note to project it. `ground.map_origin` is subtracted from them, as from a terrain's. Holes in a footprint (courtyards) are built over, and the report says which footprints had them. The file's buildings come after the scene's own in the lists, in `inspect` and in the editor's buildings table, where **Remove** leaves one out (it goes into `without`), and they count for insertion losses like any other building.

A whole town or a district from a map runs as it is. Edge diffraction looks for paths that reflect off a wall, a roof or the ground before or after bending over an edge, and it skips the walls and roofs that no such path can reach before it traces a ray, with the same results as testing them all. On four cores 900 buildings simulate in about 13 s and 3,600 in about 48 s (see Validation). About three quarters of that time for 900, and half for 3,600, goes to the terms that make reflections fade at the edges (Simulation). The work still grows with the square of the buildings: for tens of thousands, cut the map to the area round the sources and listeners, or set `simulation.early.diffraction: direct`, whose cost grows with the edges alone. `second_order` looks at every pair of edges that see each other, which in a city is most of them: keep it for a street or a site. A coverage map pays the cost of a listener at each of its points.

Ground impedance

A material with an `impedance` is a porous ground. Its impedance follows from its flow resistivity σ (kPa s/m², how hard it is to push air through it), by a preset or a value of its own:

```yaml
materials:
  lawn: {impedance: grass}                    # a preset (see below)
  field:
    impedance: {flow_resistivity: 150, model: delany_bazley}
  snow_on_ice:
    impedance: {preset: snow, thickness: 0.1} # a layer 10 cm thick on a hard backing
  powder:
    impedance: {flow_resistivity: 5, reaction: extended}  # the impedance changes with the
angle
  pasture:
    impedance: {model: two_parameter, flow_resistivity: 100, porosity_rate: 20}
assignments:
  ground: lawn
```

- `model` is the empirical model of porous media the impedance comes from: `miki` (Miki, 1990; the default), which stays passive and causal at every frequency; `delany_bazley` (Delany and Bazley, 1970), as most of the outdoor literature quotes it; or `two_parameter` (Attenborough, 1985), a porosity that falls off with depth at `porosity_rate` (1/m) under an effective flow resistivity, which fits measured grass and soil better than one parameter does.
- `thickness` (m) makes the ground a layer of that thickness on a hard backing: snow on frozen ground, a thin soil over rock. Not with `two_parameter`.
- `reaction` is how the ground meets sound at an angle: `local` (the default), the same impedance at every angle, right for grass and harder grounds; or `extended`, sound also travelling sideways inside the ground, so that a reflection meets the impedance of its own angle, Zc n / sqrt(n² - sin² θ) with n the ground's complex index (and the layer's with it; Attenborough 1985), which tells for soft snow, moss, forest floors and thin layers: a layer of snow 10 cm deep on ice reflects up to 3.7 dB differently 20 m away, fresh snow 1.0 dB, grass 0.1 dB. The early paths take it at each reflection's angle and the parabolic equation at grazing incidence; the wave solver, whose ground is one impedance for all angles, reacts locally. Not with `two_parameter`, which has no index (see Validation).
- The presets are the ground classes of Nord2000, with their flow resistivities. Real grounds spread over a factor of two or more round them.

`preset`	σ (kPa s/m²)	Ground
`snow`	12.5	very soft: new snow, moss
`forest_floor`	31.5	soft forest floor, short dense heather, thick moss
`loose_soil`	80	uncompacted loose ground: turf, loose soil
`grass`	200	normal uncompacted ground: pasture, lawns
`compacted_soil`	500	compacted field and gravel: trodden lawns, parks
`gravel`	2000	compacted dense ground: gravel roads, car parks
`asphalt`	20000	hard surfaces: most asphalt and concrete
`water`	200000	very hard and dense: dense asphalt, concrete, water

A material with an impedance takes its absorption from it: the random-incidence absorption of its impedance (Paris's formula), in third-octave bands, which the late part's rays use and `inspect` shows (grass absorbs 0.18 at 63 Hz and 0.89 at 8 kHz). It takes no `preset`, `file` or transmission, and any `absorption` given with it is replaced.

A ground scatters nothing unless its material gives a `scattering`: the early paths keep the whole reflection its impedance gives, as the exact solutions for a flat ground do (roughness takes little from sound that grazes a ground: by Kirchhoff's approximation, heights of 2 cm rms take 0.4 % out of a reflection 5° off the ground at 1 kHz). A `scattering` given to a ground is a share lost at every reflection off it in the image sources (over a terrain, the reflections off its mean plane and off the slopes in sight of the source), the curved rays and the paths round edges, which the late rays scatter; the parabolic equation, the wave solver and the waves bent over a terrain's crests keep the whole reflection.

The early paths use the impedance itself. A point source's spherical wave does not reflect off a ground as a plane wave does: near grazing incidence, where the plane wave's reflection factor R tends to -1 and would cancel the direct sound, a ground wave makes up part of it. A reflection off a ground takes the spherical wave's factor Q = R + (1 - R) F(w) (Weyl-van der Pol), with F the ground wave's, which depends on the path's length, the angle of incidence and the frequency. It is the asymptotic form of the exact solution for a locally reacting plane: from 5 m away, the level is within 0.05 dB of the exact solution's over grass and harder grounds, 0.2 dB over snow and 0.4 dB over a layer of snow 10 cm thick, more only in dips over 20 dB deep (see Validation).

Over soft ground the reflection arrives turned in phase and cancels part of the direct sound over a band of frequencies: the ground dip, at a few hundred hertz for a source and a listener a metre or two up over grass, deeper and wider the farther apart they are. Over hard ground the dip lies where the reflected path is half a wavelength longer, often above 1 kHz. In the `roadside` example, 25 m from the road and 1.5 m up, the 500 Hz band is 11.3 dB quieter over the meadow than over the paved yard across the road, and the yard has its own dip at 2 kHz.

So that the dip shows in every band, outdoors the early paths add in pressure in every band by default (`simulation.early.coherent_max_band_hz: 8000`; a room takes 500 at most, see Low frequencies: early reflections in pressure): outdoors the paths are few. Up to 500 Hz they go through the band filters as indoors, and the time parameters there are read from them (in wind and temperature gradients, out of the shadow, the parabolic equation's path from its field at every frequency its runs took, which in the 63 Hz band goes on down to an octave below it; see Validation); higher up each pair of paths adds its cross term as far as the band leaves the two coherent, which it does less the more their delays differ. The points of a line or surface source still add in energy, as they are uncorrelated; the paths from each point add in pressure.

The wave solver fits each ground to its impedance itself, its admittance and phase as well as its absorption (see the wave solver). Any surface may have an impedance, indoors too. `calibrate` (see Fitting materials to measurements) fits absorption coefficients, so it leaves materials with an impedance alone, and outdoor scenes, which do not decay as a room does.

Ground zones

Zones are parts of the ground with a material of their own: a road across a meadow, a lawn in a paved square, a car park.

```yaml
geometry:
  ground:
    margin: 30
    zones:
      - name: road
        polygon: [[-100, -4], [100, -4], [100, 4], [-100, 4]]   # corners, in order
      - name: car_park
        polygon: [[20, 4], [60, 4], [60, 40], [20, 40]]
assignments:
  ground: lawn        # the ground outside the zones, and zones without a material
  road: asphalt
  car_park: asphalt
```

A zone is a polygon on the ground, cut to the ground's extent; zones may touch but not overlap. Its surfaces are `<name>`, `zones` and `ground`, so a zone without an assignment of its own takes the ground's material. Zones without a `name` are `zone_1`, `zone_2` and so on, and the names of zones, buildings and barriers are all different.

A reflection near where two grounds meet takes some of each. The sound that reflects comes from a patch of ground round the reflection point, its first Fresnel zone: the ellipse where a path through the ground is at most half a wavelength longer than the reflected path, small at high frequencies and near normal incidence, long near grazing incidence. The reflection factor is the sum of each ground's (Q for a ground with an impedance, the square root of 1 - absorption for one without), each weighted by the share of the Fresnel zone's area it covers, band by band, as engineering models such as Nord2000 and Harmonoise weight mixed ground. Past the ground's extent the ground goes on as it is at its edge.

Wind and temperature

Outdoors the speed of sound changes with height. The air is warmer or cooler higher up, and sound travels faster in warmer air; the wind, slowed by the ground, blows faster higher up, and sound travels faster with it and slower against it. Sound bends towards where it travels slower:

- **down** when its speed grows with height: downwind, and on a clear night, when the ground cools the air next to it (an inversion). Sound comes back to the ground again and again on its way, and carries further than in still air.
- **up** when its speed falls with height: upwind, and on a sunny day, when the ground warms the air next to it. Past some distance a listener near the ground is in a **shadow** that no ray reaches: only the sound that bends round its edge gets there, less and less with distance and with frequency.

```yaml
atmosphere:
  preset: windy                 # neutral, windy, sunny_day or clear_night
  wind: {direction_deg: 90}     # the preset's 5 m/s 10 m up, blowing towards +y
```

`atmosphere` is for outdoor scenes; in a room the air is still and one temperature, and a room with an atmosphere stops with an error. Its fields:

field	what it sets
`preset`	a weather to start from: `neutral` (still air), `windy` (5 m/s 10 m up), `sunny_day` (the air cooling with height: `temperature_log_k: -0.5`, `temperature_gradient_k_m: -0.0098`) or `clear_night` (the air warming with height: `temperature_log_k: 0.5`). Fields given with it change it.
`wind.speed_m_s`, `wind.height_m`	the wind's speed (m/s) at a height above the ground (m, 10 by default)
`wind.direction_deg`	the way the wind blows to, in degrees counterclockwise from +x seen from above (0: along +x, 90: along +y)
`wind.profile`	how its speed grows with height: `log`, as ln(1 + z / `roughness_m`), the usual profile over open ground (the default); `linear`, in proportion to the height; `constant`
`temperature_gradient_k_m`	how the temperature changes per metre of height (K/m): -0.0098 in air the sun stirs up
`temperature_log_k`	a change of temperature fastest near the ground, in K times ln(1 + z / `roughness_m`): negative on a sunny day, positive on a clear night
`roughness_m`	the ground's roughness length for the log profiles (m, 0.05 by default; 0.01 to 0.1 over grass, up to 1 among buildings and trees)
`table`	a measured profile in place of the formulas: rows of [height above the ground (m), temperature (°C), wind speed (m/s)] by height, linear between them and as at the top above them; the wind blows along `wind.direction_deg`

The scene's `air.temperature_c` is the temperature at the ground (a table gives its own). The wind and the temperature depend on the height only, the same over the whole scene unless weather zones say otherwise: this suits open ground better than streets, where buildings shelter the wind and channel it. See `examples/open_air_stage/scene.yaml`: a stage heard 100 and 300 m away across a park, downwind and upwind.

The 3D view (the editor's, and `acoustic-spaces view`) draws the wind: sky blue arrows over the ground, at the height the wind's speed is given at (`wind.height_m`), pointing the way it blows and as long as it is strong, the strongest a little shorter than the space between them. The editor's plan draws the same arrows from above, with the scene's wind in its top right corner (`wind 5.0 m/s at 10 m`). Arrows keep clear of the sources and listeners, and none stand inside a building taller than them. In the editor, **View > Show wind profile** also stands a mast in each weather where the wind blows, the scene's and each zone's: arrows at heights evenly up to the wind's height and a curve through the wind at every height between, how it grows from the still air at the ground; **Show the profile** plots the same in numbers.

Weather zones

The weather is not the same everywhere: the wind drops in the lee of a row of buildings or a wood, the air lies cooler over a lake and warmer over a car park, and a sea breeze blows along the shore and dies away inland. `atmosphere.zones` draws parts of the scene with weather of their own:

```yaml
atmosphere:
  preset: windy                      # 5 m/s 10 m up, blowing towards +x
  zones:
    - name: lee                      # behind the warehouses the wind drops
      polygon: [[120, -60], [300, -60], [300, 60], [120, 60]]
      wind: {speed_m_s: 1}
    - name: lake                     # cooler air over the water, under an inversion
      polygon: [[300, -200], [600, -200], [600, 200], [300, 200]]
      preset: clear_night
      temperature_c: 12
```

A zone is a polygon in the plan, in the geometry's units, with a `name` (numbered when left out) and any of the atmosphere's fields: `preset`, `wind` (its `speed_m_s`, `direction_deg`, `height_m` and `profile`), `temperature_gradient_k_m`, `temperature_log_k`, `roughness_m` and `table`, and `temperature_c`, the temperature at the ground there. What a zone leaves out it takes from the scene's atmosphere: a zone with `wind: {speed_m_s: 1}` alone has the scene's wind slowed to 1 m/s, blowing the scene's way, with the scene's temperature profile and roughness. A `preset` in a zone starts from that preset instead, its wind blowing the scene's way unless the zone gives `direction_deg`, and the zone's other fields change it; a zone's `table` replaces the formulas, and a zone's gradient or log term drops the scene's table. Outside the zones the scene's weather holds, and where zones overlap the first wins. The eddies (`turbulence`) are the scene's everywhere. A zone that restates the scene's weather changes nothing but rounding. `inspect` names each zone, its corners and its weather; in the editor, **Weather of** chooses between the scene's weather and each zone's, and **Add weather zone** draws one in the plan (see the editor). The views draw each zone with its name and its wind's speed at the scene's wind height (`lee: 1.0 m/s`), with its ground temperature when it has one of its own (`lake: 0.0 m/s, 12.0 °C`), and its own arrows in it; the 3D view raises see-through walls round it up to the wind's height. `examples/lakeside_concert/scene.yaml` has two: a concert on a lawn by a lake, heard across it, with cool air and a stronger breeze over the water and a weaker breeze among the houses on the far shore.

A zone's border is a step: the wind does not die away gradually into the lee (only the wave solver's moving air makes it pass gradually from one zone's to the next's). How each stage of the simulation takes the zones is described below; the parabolic equation and the wave solver go through them, and the early paths, whose arcs need one profile, take the mean along their way.

How the simulation bends sound

Along each direction from a source, sound travels at the effective speed c(z) = c_T(z) + u(z) cos a: the speed of sound in still air at that height, plus the wind's speed times the cosine of the angle a between the wind and the way the sound goes (faster downwind, slower upwind, as in still air across the wind). This speed is taken as linear in height between 160 heights from 1 cm to 1 km, closest together near the ground, where the profiles change fastest, and constant above. Within each layer a ray is an arc of a circle, so its travel time and its length are exact.

- **The direct sound and its reflections off the open ground** are the eigenrays: the curved paths from the source to the listener, straight there or off the ground up to ten times. Downwind there may be several where still air has one; in the shadow there are none. Each comes in with its own travel time and angle, at a level set by how far its neighbouring rays spread apart (1/R² for a straight path). That spread comes from the ranges of the rays either side (for a ray that turns back in the air, fitted over the rays that turn a few layers above and below it, as the gentle corners where a smooth profile is sampled would otherwise make it jump), or, where the ray runs level at the source, the listener, the ground or a corner of the profile, or through a layer whose speed does not change with height, with the parts that grow without bound there taken from the arcs in closed form. Each is worked out again from rays half as far apart, and rules on how much each changes pick one; a path neither gives steadily is left out. Where the speed is the same at every height between the source's and the listener's, the direct path is the straight line. Against the smooth profiles at 20 °C (`validation/curved_rays.py`), from sources 1, 4 and 5 m up to 1300 listeners 0.5 to 12 m up, 100 to 1000 m away where the sound bends down (the `windy` preset downwind, `clear_night`) and 20 to 400 m away where it bends up (`windy` upwind, `sunny_day`), the levels of the paths, straight or off the ground up to three times, are within 1.5 dB, 87 % of them within 0.1 dB, but for three that come in 8.5 dB or more over a straight path's, nearer a caustic (up to 2.4 dB). Those the engine caps or where the smooth profile's is 8 dB or more are up to 9.4 dB off, and four that graze the ground at the shadow's edge are not checked; the sums at the listeners with none of these and no path the smooth profile lacks are within 1.9 dB. A path between ends at one height that turns just above them is within 0.8 dB of the smooth profile's, and one that turns just below them, where the sound bends up, within 1.5 dB (ends 1 to 25 m up, 10 to 600 m apart). Where the air at the source or the listener is nearly level, rounding can rarely leave out a direct path or take one that arrives up to 4 cm from the listener, at its level to 0.003 dB. Where neighbouring rays cross (a caustic) the level is capped at 10 dB over a straight path's, and each caustic advances the phase by a quarter period (a factor j, time going as e^(jωt)); past the caustic the two paths that merged there are gone, with nothing in their place (no transition zone), so in the bands the rays give over the open ground (4 and 8 kHz, and short of half the way to the shadow 1 and 2 kHz with `open_ground_frequencies: rays`) the level steps there by up to 15.5 dB within a centimetre (up to 14.0 dB at 1 and 2 kHz; see Validation). Against an independent ray tracer on the smooth profile of the `windy` preset downwind, the levels of three such paths 100 and 300 m away are within 0.05 dB (the tests allow 0.2 dB); no such check is made near a caustic.

- **In the octave bands up to 2 kHz** rays fail over the ground, as the profiles change most within a wavelength of it, where the direct sound and its reflection meet: at 1 and 2 kHz, 50 to 300 m from sources 0.5, 2 and 5 m up to a listener 1.5 m up, they were up to 3.2 dB off a fast field program in the `windy` preset downwind, 8.5 dB in `clear_night` and 9.6 dB under a gradient of 0.1/s, and stepped by up to 14 dB a centimetre at a caustic. There the sound over the open ground comes from a parabolic equation (the Green's function parabolic equation of Gilbert and Di) marched out from the source in the vertical plane through the listener, over the grounds along the way and their impedances, at eight frequencies an octave and 24 at 1 and 2 kHz, where the ground's fringes come closer in frequency (fewer there, or the rays, with `simulation.early.open_ground_frequencies`, see below), within 0.1 dB of the program in those cases; in third octaves it gives the three of each of those octave bands, to 2.5 kHz. A source's listeners and map points that the sound goes the same way to share a march, each reached by a last step of its own off it and under the absorbing layer of a run for it alone, so that over the open ground it reads the same alone as among others (behind barriers and buildings the others still set the layer, see Known issues). In the shadow, above those bands, the shadow deepens with frequency as it does over the octave below. At 4 and 8 kHz the rays give the sound: against the program, 50 to 300 m from a source 2 m up, they are up to 2.4 dB off at 4 kHz and 0.9 dB at 8 kHz in `windy` downwind and 4.6 and 1.7 dB in `clear_night`, and from a source 5 m up within 0.8 dB at 4 kHz (8 kHz was not checked there). The equation applies from 1.5 times the source's and the listener's heights added away along the ground, where the sound comes in at 34 degrees from the horizontal or flatter; nearer, the atmosphere has had little room to bend it, and the rays give it. In still air the parabolic equation is within 0.2 dB of the exact solution over grass and harder grounds from a source 1 m up, 25 to 400 m away, and within 0.45 dB from a wind turbine's hub 178 m away (see Validation). In wind, against a fast field program on the 1995 benchmark cases with the gradient stopping 100 m up, as the wind's does, it is within 0.35 dB to 1 km downwind, and at 10 km within 3 dB at 100 Hz and 11 dB at 1 kHz (47 dB down); upwind within 0.1 dB until the sound is 20 dB down and up to 6 dB off deeper into the shadow. A gradient that goes on to a kilometre up bends sound back at angles it does not carry, and beyond 1 to 2 km its pure tones stray there by 3 to 20 dB, though the bands a scene reports stay within 0.9 dB of the program to 2 km (see Validation).

- **The wide-angle parabolic equation** takes its place with `simulation.early.parabolic_equation: wide_angle` (`--parabolic-equation wide_angle` for one run of `simulate` or `map`, **Parabolic equation** in the editor). The default, `narrow_angle`, takes the refraction as a phase shift at each step, which holds for sound within some 15 degrees of the horizontal and a speed of sound close to the ground's, so the listeners whose reflection off the ground comes in more steeply get the wide-angle one's level: in 28 cases of wind and temperature gradients, in the octave bands compared (63 to 500 Hz in ten of them, 63 and 125 Hz in the others), within 0.09 dB of a fast field program (0.29 dB per third octave, compared in 24 of them), where the narrow-angle one is up to 0.96 dB off (2.2 dB), and at 1 and 2 kHz within 0.07 and 0.49 dB (a one-off check in the `windy` preset); flatter, the narrow-angle one is up to 1.10 dB off per octave band in those bands, and 5.7 dB in a third octave 14 dB below free field, these two in 10 m/s of wind over a hard ground (see Validation). The wide-angle one (split-step Padé, as in Collins' RAM) marches the field with the exact one-way operator, written as a rational function of ten terms, on finite differences of fourth order in height, with the ground's impedance as its boundary and a starting field of the operator's own; it carries sound at up to 45 degrees from the horizontal through air of any speed. On the benchmark's gradients going on to a kilometre up it is within 0.11 dB of the fast field program to 10 km (0.4 dB at 10 Hz 44 dB deep in a shadow) where the narrow-angle one strays by 3 to 20 dB, and in the weather presets within 0.02 dB per band on average where the narrow-angle one is within 0.07 dB (see Validation). Over open ground it takes five to eight times as long, all of it in the bands the parabolic equation gives, and behind barriers and buildings, where it runs in every band, about ten times.
- **Off façades and barriers** the paths bend the same way, in the vertical plane the reflections fold them into, with the wind's mean component over their legs. A path that would pass over the wall, or through something on the way, is left out. These paths (those over no edge) are eigenrays in every band, without the parabolic equation (see Known issues).
- **Over an edge** (a roof, the top of a barrier) each leg keeps its straight course and takes the travel time of its heights, and past the start of the shadow its level falls as a creeping wave's does.

- **Behind barriers and buildings**, where the listener's way seen from above crosses one, the paths in the vertical plane through the source and the listener (off the ground and the roofs, and over the tops across it) keep their levels in still air, and in every band the weather changes them by its effect in a parabolic equation with the barriers and buildings on the way inside: the band energy marched through the weather over that marched through still air, both from the source along whole degrees of direction. A listener takes the nearest whole degree's effect, and near halfway between two both, weighed in dB by its direction, so that the effect changes smoothly as it moves round; a degree along which more or fewer barriers and buildings stand short of the listener than along its own way, or the listener would be under a roof, counts for nothing, one whose last barrier stands nearer the listener than half as far as along its own way for less, and the listener's own way takes what they lose. Inside the equation a barrier lets nothing through below its top (Salomons 1994), which leaves the level behind it 0.7 to 7.4 dB below a rigid one's 2 to 90 m behind (up to 11 dB 3 to 10 m behind, steeply below the top), and a roof is a rigid floor at its own height, between the equation's heights, over which the field goes on; the ratio cancels most of that. Where the sound reaches a point more steeply than 15 degrees the wide-angle equation runs, in which a roof is a rigid floor on its height steps: behind a building in 10 m/s the effect steps by 0.01 to 0.05 dB from 250 Hz to 1 kHz, 0.30 dB at 125 Hz and 0.35 dB at 2 kHz where the points cross that angle, the narrow-angle equation's error below it (Validation). A source or a listener on a building's roof takes the same runs, with the building inside: they start from the roof, the source's image in it, or end over it. A listener on a barrier's line or a building's wall, seen from above, keeps the paths' straight legs: the equation would take it past them, where nothing gets through below the top. These paths arrive as much sooner or later as the first of them would along its legs, and take the eddies' share of a refraction shadow as the open ground's sound does. The paths round a barrier's end or a building's corner alone, off nothing but the ground, where one of them reflects off the ground, for a listener behind the barrier or in the open beside it alike, change as the open ground's field along their own way unfolded round the edges does (its length seen from above, the wind's component along it, the weathers and grounds along it), from the straight direct sound and ground reflection in still air, against which they were found, to the weather's. In the bands the open ground's parabolic equation gives (every band in a refraction shadow) they become one path, at each frequency of the bands added coherently their pressure times the equation's field over the straight paths' (their band energies so in the others); where they hold no path of a kind the straight ones have (off the ground or not: an end below the line of sight) they keep the field's turn of phase at each frequency and take its change of level over the band. Above those bands each eigenray takes a copy of them of its kind, times its pressure over the straight path's and the share of the edge it gets at its own height there over its straight partner's, at each frequency (which goes to 0 as it rises past a barrier's top; at a building's corner, whose edge goes on into the roof's, it stays whole, and the copies stop where the paths round the corner do, at the roof's height, as in still air; where none of its kind goes round the end, one that passes the end below the top takes the other kind's), as much later or sooner. So at an end's shadow boundary they jump by the weather's field, as the direct sound does, and beside a building they take the open ground's effect with the direct sound (within 0.45 dB from 63 to 500 Hz 3 m beside a wall 60 m away in the `windy` preset, down- and upwind). Behind a building, where nothing else takes a run, this costs a run of the open ground's equation a way, shared by the ways whose profiles agree and whose absorbing layers need the same grid of heights: 85 m from a source 1 m up, behind a building 10 m deep and 20 m wide 20 m from it, a listener took 1.1 s of CPU instead of 0.16 s in the `windy` preset, and a coverage map round it on a 10 m grid 33.7 s instead of 25.6 s (306 runs in its bands instead of 240); at that listener the wind's effect comes within 0.33 dB of the open ground's along the way round the corner from 63 Hz to 2 kHz and 1.04 dB at 4 kHz. Where the source or the listener stands on a building's roof, the paths round an

end take instead, in every band, the effect of the runs with the buildings under the two inside that a listener there takes with nothing else on its way (and come as much sooner or later as the first of them along its legs): the lit side's paths take those runs, so 1.5 m over a roof 45 m downwind and upwind of a source 1 m up (and the two swapped), 1 mm either side of the end's shadow boundary, the weather's effect of the paths but those over the roof's edge steps by at most 0.02 dB (up to 7.1 dB with the open ground's field along the way), and of all of them by up to 0.57 dB at 500 Hz (see Known issues). A row of 11 such listeners across the boundary took 256 runs and 14.4 s of CPU, against 312 runs and 18.9 s with the open ground's runs along the ways: along a whole degree the ways' runs hold what the lit listeners' runs there hold, and share them. Round other edges (over a top and round an end at once, a model file's objects), the paths keep their straight legs as over an edge. Against a full-wave solution of rigid barriers and blocks in wind from 63 to 500 Hz, and against a parabolic equation started behind a rigid barrier from its exact field from 125 Hz to 8 kHz, the weather's effect is within 0.71 dB behind one barrier or building in 5 m/s of wind and 0.74 dB downwind in 10 m/s, where the sound is within 25 dB of its level in still air; upwind in 10 m/s it comes out 0.2 to 1.64 dB low (the narrow-angle equation's), behind a top 45 degrees up from the source it is within 1.0 dB, behind two barriers 10 m apart within 1.73 dB and 40 m apart within 4.95 dB, and behind the started barrier within 0.5 dB (see Validation). Nearer than 1.5 times the source's and the listener's heights added, where an object of a model file stands in the way, or where no path lies in the plane, the paths go over the edges as above, and the bent direct sound and its bent reflection off the ground are there where the straight ones the edges' terms make up for are: walked across a model file's wall's shadow boundaries in 10 cm steps in the `windy` preset, the weather's effect changes by at most 0.2 dB a step from 63 to 500 Hz in octave bands across the direct sound's, and by 0.6 dB across the ground reflection's behind the wall; in third octaves by more. From a source on a roof the effect steps by 0.12 dB where the equation takes over (see Known issues).

- **The direct sound's time** in the results is when the bent direct sound arrives: sooner downwind, later upwind (the time along the ground in the shadow); where nothing clears the way, the earliest path's where it comes sooner than the straight line's (over a barrier downwind).
- **The late rays** follow their arcs through the layers, in chords that stray at most 2 cm from them, reflect where they meet a surface, and leave the scene once they are bent up out of it. The energy they rain onto listeners goes straight, in the time of the heights it crosses, and does not reach a listener in the shadow of where it leaves. They leave out the specular paths the early part holds: off the ground and vertical walls, those with up to ten bounces off the ground, which count as one reflection towards `max_order`; off any other surface, as in still air. The CPU and the GPU trace these rays alike (`simulation.late.device`): the GPU follows the same chords, in single precision, and what reaches each listener in each band, and when, agrees within the noise of the ray count (see Validation). On Direct3D 12 (Windows) the CPU traces them in a scene with vegetation too, which the run says.
- **The wave solver** (`simulation.wave`) steps the sound field itself in its bands, with a speed of sound that changes from cell to cell: at each cell's height, the still air's plus the wind's component along the horizontal from the source, as above, so the rays, the parabolic equation and the wave solver bend sound alike. Its grid is fine enough for the slowest sound and steps in time for the fastest, a few percent more cells and steps than in still air in the `windy` preset. Over grass, 50 to 200 m from the source, in the 63 and 125 Hz bands, it is within 0.60 dB of the parabolic equation, 0.2 dB on average (see Validation). It runs on the CPU or on the GPU alike (`simulation.wave.device`), the two agreeing to rounding. The run's details give the slowest and the fastest speed of sound on the grid (`wave_speed_min_m_s`, `wave_speed_max_m_s`).

- **Moving air in the wave solver.** The effective speed of sound gives each cell one speed, whichever way the sound crosses it: sound that a façade downwind sends back towards the source goes at the speed of sound going away from it. With `simulation.wave.wind: moving` (`simulate --wave-wind moving`, or **Wind** on the editor's Simulation page) the air moves instead: each cell keeps the still air's speed of sound, and the wind carries the sound along with it, so that sound goes at c + v downwind and c − v upwind, whichever way it turned. In a uniform wind of 15 m/s it is within 0.02 dB and 0.1 ms of the exact solution in every direction, where the effective speed of sound is up to 0.84 dB off; the echo of a façade 40 m downwind comes 0.01 ms off ray theory's through the wind round the building, where the effective speed of sound brings it about 10 ms early; in the `windy` preset over grass moving air is within 0.26 dB of the parabolic equation and the effective speed of sound within 0.39 dB, 0.1 and 0.2 dB on average (see Validation). It takes four to five times as long as the effective speed of sound over open ground; round buildings its steps take about eight times as long, after the wind is made to flow round them (37 s on the 6 million cells of the validation's façade). It runs on the CPU only (asked for the GPU, it says so and stays on the CPU), and the run's details say `wave_moving_air`. Its grid is fine enough for sound going against the fastest wind of the atmosphere in the slowest air.
- **The moving air's wind round buildings.** Moving air keeps the sound's energy only where no wind crosses a surface and no cell of the grid gains or loses air; a wind blown through a building as over open ground makes the sound grow there. So where the atmosphere's wind meets a surface of the model (a building, a barrier, terrain) or changes along its own way (weather zones), the solver first makes it flow round the solids, on the faces between its cells: the wind of least change from the atmosphere's, each face's change weighed by one over the wind's speed there, that crosses no surface and leaves every cell as much as it enters it. It slows to nothing before a façade, is faster past a building's edges and over its roof, leaves a room with one opening sheltered whatever the wind's profile, and far from the solids is the atmosphere's wind. Against an ideal fluid's flow past a sphere it is within 1.5 % of the wind (root mean square) from two cells off the surface (see Validation). It is such a fluid's flow: without a wake or eddies (behind a building it closes in as it parted in front), and at a sharp edge it speeds up, the more the finer the grid, 3.3 times the wind at the edges of the validation's building on cells of 38 cm. The run says how fast it gets through a face between two cells (`wave_wind_fastest_m_s` in its details) and is refused if that reaches half a cell a time step. The wind is solved on the wave grid and ends with it: a solid near the scene's edge has its flow cut short there. It takes about 40 bytes a cell while it is solved, and 16 bytes a cell more than over open ground during the run. Over open, flat ground the atmosphere's wind crosses nothing and is carried as it is. The air of the grid's absorbing layer is still. Where the wind crosses its border, each cell's own sound is carried out and silence in, which adds no energy there; and a wave that leaves the moving air for the layer's still air comes back by about half the wind's Mach number: 32 to 40 dB below the sound at 10 m/s, against 74 to 78 dB in still air in the same check (25 to 150 Hz on cells of 30 cm; see Validation). Each run in moving air is watched all the same: once the source's pulse has ended, the pressure squared over the grid's air, averaged over pieces of at least a quarter second, may not rise to twice its lowest level so far, or the run stops with an error.
- **A source shut in a closed building.** No wind blows through closed walls. Where the wave solver's grid is closed (the source and its listeners in a building without an opening), it takes the still air's speed of sound, with either way of taking the wind, and the run says that it left the wind out (`wave_still_inside` in its details).

- **Through weather zones** the parabolic equation marches zone by zone, changing its speed of sound where its path crosses a border, and the wave solver takes each cell's zone. Its moving air then lets no cell gain or lose air (see the wind round buildings above): where the wind changes from one zone to the next along its way, it does so gradually, and what one zone's wind has more than the next's leaves sideways and upwards through the grid's edges, the more gradually the farther the grid reaches from the border. In a one-off check with a wind that drops to a fifth, half of the drop lies 2 m before the border on a grid 6 m wide and 8 m before it on one 36 m wide, four fifths of it at the border on both; this passage is not checked against a reference. The early paths, whose arcs need one profile, take the mean profile along their way, each zone's weighted by the length of the way over it (the equivalent profile of Nord2000 and Harmonoise), with the wind's component along the way from each zone's own direction; so do the paths off façades and barriers, over their legs. The late rays bend through the mean weather over the scene's extent, each zone's weighted by its area. Against a Crank-Nicholson parabolic equation marched through the zones, the engine's parabolic equation is within 0.15 dB on average and 1.2 dB at most to 800 m, over shelter and gusts downwind and upwind, an inversion past a lake shore and cool water downwind. The mean profile, which the rays' bands (4 and 8 kHz) take, is within 0.35 dB over the cool water at 500 Hz to 2 kHz; where the wind changes, or an inversion sets in, it is up to 6.6 dB off (at 2 kHz 800 m downwind of a shelter), mostly giving 1 to 4 dB more sound and in a few places up to 2.7 dB less: one profile cannot both bend the sound down over the first 150 m and let it go straight after, the cost of the equivalent profile the standards take (see Validation).
- **The animation's particles** and **the drawn paths** (`view --paths`, the Blender export) curve as the sound does.

Bending takes time: on one core the `open_air_stage` example's listeners, each simulated alone, take about 26 times as long as in still air and its coverage map on a 40 m grid about 38 times, the parabolic equation running up to 2 kHz for each point (less with `open_ground_frequencies`, see below), and its late rays take about seven times as long; `simulation.performance.threads` shares the work out over the cores. Behind barriers and buildings the parabolic equation runs in every band, in the weather and in still air, for each degree of direction, the runs of a source that differ only past their first obstacle sharing the march over the open ground before it: on one core, the runs for a listener 160 m out take about 7 s, 85 % of it at 4 and 8 kHz; a coverage map over grass in 8 m/s of wind, of 96 points behind two barriers or 90 round a building, 77 to 89 s; and three or four listeners 10 to 20 m up 30 to 60 m from the source behind a barrier or a building, where the wide-angle equation runs (the sound reaching them steeply), about 60 s, against 0.1 to 0.5 s in still air.

Over the open ground the runs take 24 frequencies an octave at 1 and 2 kHz; below, 8. `simulation.early.open_ground_frequencies` takes 8 there too, or `rays`, which leaves those bands to the rays where one reaches the listener short of half the way to the shadow's start (a third-octave run then takes the equation's bands by octave, so its 630 Hz band comes from the equation, and deepens the shadow above them from the octave bands' levels, as an octave run does). Against the fast field program (the band's energy over 24 slices; over 48 it moves by up to 0.04 dB from the source 10 m up), at 290 points within 60 dB of free field in 18 cases (`validation/open_ground_frequencies.py`: `windy` down- and upwind, `sunny_day` and `clear_night`, sources 0.5, 2 and 5 m up over grass and 2 m up over a rigid ground, and 10 m up over grass downwind and on the clear night, listeners 1.5 and 4 m up 50 m to 1 km away, 500 m in the upward cases), the early sound at 1 and 2 kHz is off by, in dB (mean, nine in ten, largest), and a run takes on one core, against the rays, for the `open_air_stage` example's four listeners each simulated alone, its coverage map on a 40 m grid, `helicopter_flyover`'s listener under the track with the helicopter held 300 m short of it (the wide-angle equation from 150 m up; the example itself sets `rays`, which are within 0.11 dB of the default along its track) and the listener 200 m downwind in `validation/long_term.py`'s scene in its own weather:

`open_ground_frequencies`	1 and 2 kHz	the stage	its map	the helicopter	a class
24 (the default)	0.07, 0.14, 0.75	2.0 times	3.4 times	26 times	9.4 times
8	0.17, 0.35, 4.1	1.2 times	1.4 times	9.4 times	3.4 times
`rays`	0.67, 1.7, 8.4	1	1	1	1

With 24 the largest error is the parabolic equation's own, 43 dB deep in the shadow 200 m upwind; 8 is furthest off 1 km downwind, and the rays 200 m away under the clear night's inversion, and they step at a caustic by up to 14 dB a centimetre at 1 kHz (Known issues). The other bands change only in the shadow, where 4 and 8 kHz deepen from the 1 and 2 kHz levels: 8 and `rays` move them there by up to 0.33 dB within 60 dB of free field in these cases, and by up to 4.9 dB deeper (89 dB below it, 8 kHz upwind). Nothing changes in still air.

Through the crossovers between the bands the equation's bands carry its field at every frequency the runs took, and one more run takes the top band on through the crossover above it at the same spacing, so a ground dip at a join is kept (so do the paths round a barrier's end or a building's corner, from their way's runs, each path in still air drawn between its own frequencies, unless the end is below the line of sight); the lowest band's run goes on down to an octave below it, which its band filter reads (out of the shadow: in it, where the bands above the equation deepen from the octave below, the bands keep their own samples) (Validation). On one core these take the equation from 0.75 to 1.27 s for a listener 100 m downwind in the benchmark's case 2, from 4.9 to 6.0 s 300 m away, and in `windy` from 2.0 to 2.9 s 200 m downwind, 22 to 29 s 600 m and 38 to 66 s 1 km downwind (in third octaves from 0.87 to 0.97, 4.0 to 4.6 and 1.6 to 1.9 s in the first three; with 8 an octave from 0.29 to 0.52 s 100 m away in case 2 and from 12 to 21 s 1 km downwind). A coverage map, which reads the bands above 500 Hz from their energies, takes the run above the top band only with `rays`, where its 500 Hz band filter reads that crossover (8 of the 77 runs of the stage's map on a 40 m grid, 0.2 of its 8.6 s).

Behind barriers and buildings the runs take 32 frequencies an octave at 8 kHz and 16 at 4 kHz, where the ground's interference fringes are densest; below, 8, as the early paths do. `simulation.early.barrier_frequencies` takes fewer, 16 or 8 at 8 kHz (and no more than that at 4 kHz), for less time and a weather's effect further off. Against 64 an octave, at 15 listeners 15 to 150 m away behind barriers (one 15 m from the source, one 4 m from a source by a rail line, one with listeners up to 14 m up, one whose top is 60 degrees up from the source) and a building, over rigid ground and grass, in 5 m/s of wind down- and upwind (`validation/barrier_wind.py`'s convergence cases, where the effect is within 25 dB of still air), the weather's effect is off by, in dB (mean, nine in ten, largest), and a coverage map of 96 points over grass behind two barriers in 8 m/s takes, on one core:

`barrier_frequencies`	4 kHz	8 kHz	the map	against straight legs
32 (the default)	0.05, 0.10, 0.25	0.04, 0.09, 0.51	81 s	5.5 times
16	0.05, 0.10, 0.25	0.51, 1.0, 6.4	59 s	4.0 times
8	0.80, 1.7, 9.6	2.1, 3.5, 14	45 s	3.0 times

These come on top of the parabolic equation's own error (see Validation), and the bands below 4 kHz do not change. A source or a listener on a building's roof costs what a listener behind a building does: from a source 1 m over a roof 6 m high, upwind in the `windy` preset, one listener 43 m away takes 4.2 s on one core and one 200 m away 15.7 s (1.0 and 2.8 s when the building was left out), but ones 20 and 25 m away, close behind the edge where the wide-angle equation runs, 18.7 and 25 s (0.1 and 0.2 s), 16 s of the first in its 8 kHz run; a map of 22 points round the building takes 159 s (4.6 s), 105 s with `barrier_frequencies: 16` and 54 s with 8. The places of a walking source share the parabolic equation's runs where they stand at one height over one ground and neither the weather nor the ground has zones: a run from one place serves the listeners of every other that the sound goes the same way to, each reached by a last step of its own under the absorbing layer of a run for it alone, so that over the open ground it gets what a run of its own place would give it (see Validation). Each place keeps its bent paths while it waits for the shared runs, so the memory a walk takes grows with the places that share them. An aircraft's 26 places 64 m apart, 100 m up over 1.6 km of a meadow in the `windy` preset, heard 30 m off the middle of its track, take about four fifths of the time on one core that a run for each place takes, with the same early levels. An atmosphere in which sound goes straight (`neutral`, no wind and no temperature gradient) changes nothing: the results are those of still air, bit for bit.

Without turbulence, shadows come out deeper than measured. The eddies of the wind and of the temperature scatter sound into a shadow: in the validation's shadows, 280 to 500 m out and from 500 Hz up, they keep it 5 to 25 dB below the free field, where without them it is 50 to 70 dB deep. Give the atmosphere its `turbulence` (see Turbulence).

Against measurements of a wind turbine up to 845 m away in ten spells of different weather, from light winds to strong upwind, the bands from 80 Hz to 2 kHz are within 1.8 to 2.7 dB on average, about as close as the published parabolic equation model in the eight cases other than strong upwind, and closer in those two (see Validation).

`acoustic-spaces inspect` describes the atmosphere and gives the speed of sound across the wind, downwind and upwind at a few heights up to the sky. From Python, `acoustic_spaces.viz.plot_atmosphere(scene.atmosphere, scene.air.temperature_c, "profile.png")` draws the temperature, the wind and the speed of sound over the ground.

Turbulence

The wind and the temperature are not steady: eddies of every size, from the gusts of the wind to the warm plumes the sun raises off the ground, drift past, and the speed of sound flickers with them. Outdoors that matters in three ways:

- The paths of one sound that go different ways stop adding in step: the **ground dip**, where the direct sound and its reflection off the ground cancel, fills in at long distances and high frequencies.
- Sound is **scattered into shadows**: a shadow of the wind or of the sun's warmth stays some 5 to 25 dB below the free field instead of 50 to 70 (see Validation).
- The sound of a far source **flickers**, louder and softer by several decibels from one second to the next, as the eddies drift past.

```
atmosphere:
  preset: windy
  turbulence: moderate          # weak, moderate or strong; or the values themselves:
  # turbulence: {temperature_structure: 0.008, wind_structure: 0.12, outer_scale_m: 10}
  # turbulence: {preset: moderate, height_m: 2}   # the strength at 2 m, weaker higher up
```

`turbulence` goes in the `atmosphere`, also in still air (`atmosphere: {turbulence: strong}`). Its fields:

field	what it sets
`preset`	a strength to start from (below); fields given with it change it
`temperature_structure`	the temperature's structure parameter C_T² (K² m^-2/3): about 0.001 to 1 near the ground, the most in sunshine
`wind_structure`	the wind's structure parameter C_v² (m^4/3 s^-2): about 0.01 to 1 near the ground, the more the stronger the wind
`outer_scale_m`	the size of the largest eddies (m, 10 by default): some metres near the ground
`height_m`	the height above the ground (m) the structure parameters hold at, from which the eddies weaken with height as (z / `height_m`)^-`falloff`, taken no nearer the ground than `outer_scale_m`; none by default: the same at every height
`falloff`	how fast they weaken with height, the power of z / `height_m` (2/3 by default): 2/3 in a neutral or stable surface layer, up to 4/3 in strong sunshine

preset	C_T²	C_v²	weather
`weak`	0.001	0.03	overcast, a light wind
`moderate`	0.008	0.12	a breeze and some sun
`strong`	0.08	0.5	a strong wind or a hot sun

These are indicative, for the strength near the ground, and are taken as the same at every height unless `height_m` is given. Real eddies are strongest near the ground and weaken with height: in the surface layer the structure parameters fall as z^(-2/3) in neutral and stable air (the wind's always, the temperature's as z^(-4/3) when the sun drives the air; Monin-Obukhov similarity, Wyngaard, Izumi and Collins 1971). With `height_m` the values hold at that height, where they were measured or estimated (usually 2 m), and fall above it by that power; nearer the ground than the eddies' outer scale the engine takes them as at the outer scale (with `height_m` 2 m and the default 10 m, 0.34 of the given values up to 10 m, then falling), because eddies that size reach that high and the thin layer of stronger ones right at the ground does not tell in the parabolic equation (see Validation). Without `turbulence`, or with both structure parameters 0, the results are those without eddies, bit for bit.

How the simulation takes eddies into account

The eddies make the index of refraction fluctuate, by -T' / (2 T0) for the temperature's and -v' / c0 for the wind's, both with a von Kármán spectrum of outer scale `outer_scale_m`, taken together as one of strength $Cn^2 = C_T^2 / (4\ T0^2) + (11/6)\ C_v^2 / c0^2$ (Ostashev and Wilson 2015).

- **Each two early paths of a sound** add in pressure times their coherence factor exp(-D / 2): D grows with the square of the frequency, with the distance and with how far apart the two paths go, from the Markov approximation along both (Clifford and Lataitis 1983) at 24 points each. Near the ground the eddies that part the direct sound and its reflection span few wavelengths, and a parabolic equation marched through random eddies fills the dip far less than that: two paths that differ only by reflections off the ground take D times 1 / (1 + (32 / k rho)^1.7), rho their largest separation, fitted to it. With it the level of the direct sound and its reflection is within about 1 dB of the parabolic equation's on average over eight grounds and heights (see Validation). The echogram, the parameters and the coverage maps take this average.
- **Eddies weaker with height** (`height_m`) enter where each pair of paths runs: the phase difference two paths gather over a stretch grows with Cn^2 at the mean height of the two points there, so paths aloft part slower than with eddies the same everywhere; the sound scattered into a shadow takes Cn^2 at the height it scatters at, and the shaking of the refraction Cn^2 at the geometric mean of the source's and the listener's heights; all of them no nearer the ground than the eddies' outer scale. Against the parabolic equation through eddies strengthening right down to the ground, the ground dip is within 0.9 dB on average over the soft grounds and 1.5 over the harder ones (0.8 and 1.0 with eddies the same at every height) and the shadows within 1.3 to 2.9 dB (0.7 to 1.2); taking Cn^2 at the mean height itself, right down to the ground, would put the dip 1.7 dB off over the soft grounds (see Validation).
- **In the parabolic equation's bands** (in wind or temperature gradients, see How the simulation bends sound) the eddies blur its sound towards the rays' added in energy, by the rays' coherence factors at each frequency the equation ran at.
- **In a shadow** the eddies add the sound they scatter into it: once, from the air that the source and the listener both see over the shadow's edge (the Born approximation, which the parabolic equation through random eddies bears out at 250 Hz), and at higher frequencies by shaking the refraction near the ground that makes the shadow, a share of the free field's energy that grows as the square of the eddies' speed of sound against the fall of the speed of sound over the heights. That share is fitted to the parabolic equation on a sunny day and upwind in the `windy` preset, the source 2 m and the listener 1.5 m up, and deep in those shadows the engine is within about 1 dB of it on average, 3.5 dB at most; the shadows of other heights and profiles follow the same formula, untested. A barrier's or a building's own shadow takes nothing from the eddies: the sound they scatter over its top into it, which limits a barrier's insertion loss at high frequencies, is not modelled. Behind one, in a refraction shadow, each path in the plane through the source and the listener takes the open ground's share of its own energy, which the barrier has already lowered, so the eddies' share comes down by the barrier's loss too.
- **An impulse response is one moment of the eddies.** Each path is delayed at random, band by band, as the eddies have it at that moment (the seed draws it), so the levels of a response scatter about the echogram's average, the more the stronger the eddies and the higher the band; averaged over many moments they give it. In the parabolic equation's bands a response carries the average.

- **Listening and walkthroughs flicker.** `acoustic_spaces.render.auralize`, the editor's **Listen** and `walkthrough` send the sound through a new moment of the eddies every sixteenth of the time they take to drift past (the outer scale over the wind's speed, at least 1 m/s; at most every 50 ms). Each early path plays through its own delay, which goes from one moment's to the next's (its top band's; how its other bands lag behind it goes into its filter), as along a walk with Doppler: crossfading the moments' responses instead would mix each path at two delays a few microseconds apart, which cancels the treble half-way between them. The moments change as an Ornstein-Uhlenbeck process, so the delays drift rather than jump, and a walker keeps the same eddies from place to place. The `auralize` command and live playback (`play`, head tracking) go through the saved responses, one moment each.

Vegetation

Trees, hedges and belts of trees are no surfaces: sound passes through them, and their leaves, twigs and trunks take some of it on the way, absorbing it or scattering it on. `geometry.vegetation` lists them:

```
geometry:
  ground: {extent: [[-90, -30], [90, 90]], sky: 25}
  vegetation:
    - name: wood                    # a belt of trees: a footprint raised to a height
      footprint: [[-90, 15], [0, 15], [0, 45], [-90, 45]]
      height: 15
      foliage: dense                # ISO 9613-2's dense foliage, the default
      stems: {per_m2: 0.08, diameter: 0.3}
    - name: hedge                   # a hedge: a path of a width
      path: [[10, 50], [40, 50], [40, 60]]
      width: 1.5
      height: 2
    - name: oak                     # a lone tree's crown
      tree: [45, 52]
      crown_diameter: 10
      base: 3                       # where the crown starts
      height: 14
```

field	what it sets
`name`	its name (`vegetation_1`, `vegetation_2` … by default)
`footprint`	a belt of trees or a wood: the (x, y) corners of the ground it covers, raised from `base` to `height`
`path`, `width`	a hedge: the (x, y) points along it and how wide it is (its sides mitred at the bends)
`tree`, `crown_diameter`	a lone tree: where it stands and how wide its crown is; the crown is a spheroid from `base` to `height`
`height`	its top over the ground
`base`	its bottom over the ground (0 by default); for a tree, where the crown starts, the trunk below left out
`foliage`	what the leaves and twigs take: a preset or values (below)
`stems`	the trunks and stems in it: `per_m2` per square metre of ground, `diameter` across (m)

Each has one of `footprint`, `path` or `tree`, in the geometry's units over the ground. A volume may run past the ground's edge (a wood the ground cuts off), and where two overlap, what fills them adds up.

Foliage is a preset or an attenuation per metre:

`foliage`	what it takes
`dense` (the default)	ISO 9613-2's dense foliage (Annex A): 0.02, 0.03, 0.04, 0.05, 0.06, 0.08, 0.09 and 0.12 dB per metre from 63 Hz to 8 kHz, all absorbed
`none`	no foliage: bare stems, a wood in winter (then it needs `stems`)
`{attenuation: [...], albedo: 0.5, asymmetry: 0.3}`	dB per metre per octave or third-octave band; `albedo` is the share of it that the foliage scatters on instead of absorbing (0 by default), `asymmetry` the mean cosine of the angle it scatters through (0 every way alike, towards 1 onwards)
`{leaf_area_per_m3: 0.5, leaf_width: 0.06}`	the leaves themselves: their area (one side's) per cubic metre of the volume and their width (the diameter of a disc of a leaf's area, m); what they scatter and absorb in each band, and how far forward they scatter it, follow from their size (below)

ISO 9613-2's rates are what dense foliage takes from sound passing through it in all, with what it scatters onwards counted in, so the preset absorbs all of it and adds no scattered sound. The standard gives them for 20 to 200 m of foliage, a fixed 0 to 3 dB for 10 to 20 m (none at 63 and 125 Hz, 1 dB from 250 Hz to 2 kHz, 2 and 3 dB at 4 and 8 kHz) and none below 10 m. The engine takes the rates per metre at any depth: a belt under 10 m deep loses 0.02 to 0.12 dB per metre where the standard says nothing, and one 10 to 20 m deep loses 0.2 to 0.6 dB more than the standard at 63 and 125 Hz, 0.2 to 1.8 dB less at 4 and 8 kHz and within 0.6 dB of it in between. It measures the depth along the straight path, where the standard takes an arc of 5 km radius.

Leaves given by their area and width are taken as thin rigid discs of that size, turned every way alike, `leaf_area_per_m3` of them per cubic metre. Each scatters as a disc does: small against the wavelength, as a dipole, from the air it holds back $(16\ (kr)^4 / (81\ \pi^2))$ of its area, k the wavenumber and r its radius), so the low bands pass almost untouched; large against it, twice its mean projected area, half of that the shadow it casts, which goes nearly straight on, and half reflected off its faces, which go every way alike. In between the two limits are bridged smoothly, as the exact cross-section of a disc about a wavelength across is not in closed form. The leaves absorb in the viscous and thermal boundary layers on their two faces, a few per mille of their area, which is what they take at the low frequencies where they scatter nothing. Per metre, the volume takes its leaf area times those cross-sections over a disc's area, averaged over each band (the speed of sound sets the wavelength); the scattered share and its mean cosine (0 for the dipole, towards 1/2 for a large disc) replace `albedo` and `asymmetry`, which leaves do not take. Leaves 5 cm wide at 0.5 m^2 per m^3, a dense canopy, take 0.05 dB per metre at 2 kHz, 0.5 at 4 kHz and 1.6 at 8 kHz out of the early paths, and almost nothing below 1 kHz; a hedge's leaves (2 to 6 m^2 per m^3) take several decibels per metre from 4 kHz up. Much of it comes back scattered, with the late rays, so what a listener loses is far less (see Validation). A real leaf is neither rigid nor a disc, and leaves hang in clumps: measured foliage, where there is any, is the better input, as `attenuation` with an `albedo` and an `asymmetry`.

Stems are upright rigid cylinders. They absorb nothing and scatter, per metre of horizontal travel, their number times a cylinder's scattering width: the exact series for sound at right angles to its axis (Morse and Ingard 1968), averaged over the band. 0.08 stems per m^2 0.3 m across scatter about 0.002 dB per metre at 63 Hz and 0.2 dB at 8 kHz. Sound going up or down meets them along its horizontal share only, and they turn it sideways, keeping its slope: in the low bands mostly back the way it came, in the high bands mostly onwards.

How the simulation takes vegetation into account

- **Early paths** (the direct sound, the reflections, the paths round edges) lose exp(-Σ μ l) of their energy, μ each medium's attenuation per metre (the stems' times the horizontal share of the leg) and l the length of each leg inside each volume: Beer-Lambert. In the parabolic equation's bands (in wind or temperature gradients) the sound over the ground loses what the direct path loses. What the plants scatter out of the early paths comes back in the late part.
- **The late rays** lose what the plants absorb along their way and scatter in them at random, as often as the media scatter. Where a ray scatters in foliage, the foliage sends its share straight to each listener the way its phase function has it (Henyey-Greenstein), as a surface's scattered sound reaches them, and the ray goes on in a direction drawn from it; where it scatters off stems, it turns sideways, keeping its slope, so the stems send a listener nothing straight: they send what the ray's next leg would carry through the listener's detector sphere, averaged over the turns (a quadrature over those that cross it). The rain from every reflection loses what the plants take on its way to the listener. In several bands at once, the rays scatter at the rate of the bands' energy-weighted mean and each band is weighted by what it meets, so each keeps its own expectation. Against a separate particle simulation through a slab of foliage or stems, the tracer agrees within its noise, a few tenths of a percent (see Validation). The CPU and the GPU trace rays through vegetation alike; vegetation of more than 16 media (kinds of foliage and stems in one scene), and on Direct3D 12 (Windows) vegetation in wind or temperature that bends the rays, is traced on the CPU, which the run says.
- **Coverage maps** take the same: the early paths' loss and the late rays'.
- **The wave solver's bands** leave vegetation out: at their low frequencies foliage takes little (0.02 to 0.03 dB per metre in the `dense` preset).

Behind vegetation that scatters (stems, or foliage with an `albedo`), much of the sound comes scattered, with the late rays, and their noise shows in the levels: trace more of them (`simulation.late.rays`) for steadier results. The `tree_belt` example traces 200000, and G behind its trees still varies from run to run, by 0.1 to 0.9 dB (one standard deviation over ten seeds, a one-off check) depending on the band.

`acoustic-spaces inspect` lists the vegetation with what sound loses per metre through each, foliage (or leaves) and stems together. The editor's 3D view, `view` and the export to Blender draw the volumes in see-through green, and coverage maps outline them.

What the simulation does outdoors

- **Early paths** are found as in a room: the direct sound, reflections off the ground, façades and barriers, and (with `simulation.early.diffraction`) the sound bent over roofs, round the corners of buildings and over the top and round the ends of barriers. Edges on the ground (the foot of a wall or a barrier) and the inside corners where two wings of a building meet diffract nothing. `second_order` follows the sound over a building, its front and back edges in turn, along the roof with the ground wave's loss over it (see Simulation); `first_order` bends over one edge at a time, which is enough for a barrier. With `diffraction_reflections: both_sides` a path reflects off the ground both before and after the edge, the fourth of the four paths over a barrier on the ground (see Edge diffraction). The sound over long barriers and rows of buildings comes from a solver for the whole row in the bands up to 500 Hz (see Sound over rows of barriers and buildings).
- **The late part** comes from the stochastic ray tracer: rays reflect off the ground and the buildings, and scatter off those that scatter, until they leave through the sky. Between tall buildings (a street canyon, a courtyard) that is a tail of its own; in the open, almost nothing: over flat ground that scatters nothing, nothing at all, and over a terrain that scatters nothing, the sound its slopes reflect more often than the image sources' order (twice or more, off the hollow at the foot of the `hillside` example's hill). The statistical late method needs a diffuse field, which the open air does not have: outdoors it stops with an error.
- **Response length.** Sources and listeners can stand hundreds of metres apart, so outdoors `ir_length_s` counts from the latest direct sound of a run: a listener 700 m away with `ir_length_s: 1` gets 3 s responses (2 s of travel, then 1 s). The run's details give the length (`ir_length_s`).
- **Coverage maps** cover the ground by default and leave out the points inside buildings (see Coverage maps).
- **The wave solver** sees the sky as an opening and puts a matched layer round the grid, as for any open model, past the source and the listeners above and to the sides by at least 15 % of the farthest a listener lies from the source along that side (where the sky or the ground's edge is nearer), and in wind and temperature takes a speed of sound per cell (see How the simulation bends sound).
- **A terrain** reflects the early sound off the mean plane of its profile and bends it over its crests, and the late rays off its triangles (see How the simulation takes the terrain into account).
- **Air** absorbs per band as indoors (ISO 9613-1, from `air`), along every path.
- **Wind and temperature** bend the paths and the rays (see Wind and temperature), and their eddies blur the paths' interference, scatter sound into shadows and make it flicker (see Turbulence).
- **Vegetation** takes sound from the paths through it and scatters some of it into the late part (see Vegetation).
- **The early paths add in pressure in every band** (see Ground impedance): `simulation.early.coherent_max_band_hz` is 8000 outdoors unless the scene sets it.

`acoustic-spaces inspect` says where the ground and the sky lie, describes the atmosphere, and leaves out the statistical reverberation times, which need a diffuse field.

Results outdoors

Outdoors every response and every map point also gives its **level relative to free field**: how much louder (or, negative, quieter) the source is there than the same source as far away in free field, with the same air between them, per octave band and A-weighted. The ground, the weather, buildings, barriers and vegetation make the difference; with the sign turned it is the excess attenuation of outdoor acoustics. It compares the energy the response holds with the free field's, so it needs no decay to read, and neither does G outdoors: where a response does not decay as a room's does (an open field, a lone barrier), G and the levels that take it are those of the energy it holds instead of undefined; the other parameters stay undefined. `simulate` prints it, `parameters.json` has `re_free_field_db` and `re_free_field_a_db`, maps have the parameter `re_free_field` and the editor's Results panel the row **Re free field**.

The **insertion loss** of buildings, barriers or vegetation is what they take off the level: the level without them less the level with them, per octave band and A-weighted for the source's sound power. Name them in the scene,

```
simulation:
  insertion_loss_of: [noise_barrier]   # buildings, barriers or vegetation, by name
```

or for one run on the command line (`acoustic-spaces simulate scene.yaml --without noise_barrier`, and the same for `map`), and the scene is run as it is and again without them, with the same settings and seed so that both runs trace the same rays; a run takes twice as long. The results of the scene as it is then carry the insertion loss at every listener (`insertion_loss_db` and `insertion_loss_a_db` in `parameters.json`, printed by `simulate`) and at every map point (the map parameters `insertion_loss` and `insertion_loss_a`), and the report says what was left out (`removed`). A part without a name goes by its kind and its place in the list (`building_1`, `barrier_2`, `vegetation_1`). A barrier that shields the listener has a positive insertion loss, a building that sends sound back towards the listener a negative one. In the editor, the **Simulation** entry of an outdoor scene has an **Insertion loss** list with every building, barrier and vegetation to tick, and the Results panel the row **Insertion loss**. From Python, `acoustic_spaces.engine.insertion.simulate` and `coverage_map` take `without=[...]`, and `Scene.without(names)` is the scene without them.

Both are checked against exact solutions: a barrier's insertion loss within 0.07 dB of the Biot-Tolstoy-Medwin solution over an absorbing ground, the level over grass within 0.15 dB of the exact ground reflection outside the band-filtered dip (see Validation).

Levels over a long term

The weather changes from day to day, and so does the level a road or a wind farm makes a kilometre away: noise maps and assessments give the level over a long term, a year, in which each weather counts for the share of the time it holds (ISO 1996-2, CNOSSOS-EU, Nord2000). `long_term` gives the weather classes and their shares:

```yaml
atmosphere:
  preset: windy                       # 5 m/s 10 m up, towards the listeners
long_term:
  classes:
    - name: downwind                  # the scene's own weather
      share: {day: 0.3, evening: 0.25, night: 0.15}
    - name: upwind
      wind: {direction_deg: 180}
      share: {day: 0.3, evening: 0.25, night: 0.15}
    - name: still
      preset: neutral
      share: {day: 0.4, evening: 0.3, night: 0.2}
    - name: inversion                 # clear nights: the air warms with height
      preset: clear_night
      temperature_c: 8
      turbulence: weak
      share: {day: 0, evening: 0.2, night: 0.5}
```

A class has a `name` (`class_1`, `class_2` and so on when left out), its `share` of the time, and any of a weather zone's fields over the scene's weather: `preset`, `wind`, the temperature's gradient and log term, `roughness_m`, `table` and `temperature_c`, the ground's temperature. They resolve as a zone's do (see Weather zones), and a class without them is the scene's own weather. A class may also have `turbulence`, eddies of its own in place of the scene's: weak on a clear night, strong in the sun. The weather zones hold in every class. `share` is one number for the whole day, or one per period of the day, and in each period the classes' shares add up to 1.

The periods are by default the day (12 hours), the evening (4 hours, 5 dB added) and the night (8 hours, 10 dB added) of L_den (Directive 2002/49/EC); `long_term.periods` gives others, each with its `name`, `hours` and `penalty_db`:

```yaml
long_term:
  periods:                            # 15 hours of day, 9 of night with 10 dB (L_dn)
    - {name: day, hours: 15}
    - {name: night, hours: 9, penalty_db: 10}
  classes:
    - {name: downwind, share: {day: 0.5, night: 0.3}}
    - {name: inversion, preset: clear_night, share: {day: 0.5, night: 0.7}}
```

The scene is run as it is, for its impulse responses, and once in each class, all with the same seed so that every run traces the same rays; a class whose weather is the scene's own takes the scene's run, so n classes cost at most n + 1 runs (each made twice with `insertion_loss_of`, with and without what it names). Then, at every listener, per octave band and A-weighted:

- the level in each class;
- the level in each period: the energy mean of the classes' levels, each by its share of that period, L = 10 lg Σ p_k 10^(L_k/10);
- the **long-term level**: the energy mean of the periods' levels by their hours;
- the **rating level**: each period's A-weighted level with its penalty added, averaged by the hours. With the default periods it is L_den, 10 lg ((12·10^(L_day/10) + 4·10^((L_evening + 5)/10) + 8·10^((L_night + 10)/10)) / 24); with others, L_R, a whole-day rating level as ISO 1996-1 defines it.

The long-term level relative to free field, and the long-term insertion loss of what `insertion_loss_of` names (the long-term level without them less that with them, from the runs with and without them in every class), come with them. `simulate` prints them. `parameters.json` has a `long_term` block (the classes, their shares and the periods), and each response a `long_term` entry with `spl_db`, `spl_a_db`, `rating_level_db` and `rating` (L_den or L_R), the levels of each period and each class, and `re_free_field_db` and `insertion_loss_db` with their A-weighted values. A map gains at every point the parameters `long_term` (dB per band), `rating_level` (dB(A)) and, with `insertion_loss_of`, the long-term insertion loss `long_term_insertion_loss` (dB per band) and `long_term_insertion_loss_a` (dB(A)); its other parameters, `spl`, `re_free_field` and `insertion_loss` among them, are those of the scene's own weather. `inspect` lists the classes with their shares and weathers, and the periods. From Python, `acoustic_spaces.engine.long_term.simulate` and `coverage_map` take `long_term=` to run classes the scene does not give, and `Scene.under(weather_class)` is the scene in one class. In the editor, weather classes are added and changed under **Weather of** (see the editor).

The level in each class is as accurate as the simulation in that weather (see How the simulation bends sound). The combination is the standards' formula, exact to rounding; over grass, the correction it gives from the classes favourable to propagation to the long-term level follows ISO 9613-2's within 0.7 dB, with a C_0 of 3.4 dB, inside the standard's 0 to 5 dB (see Validation).

Passing vehicles

A car, a train or an aircraft is a source with a walk and `doppler: true`: `walkthrough` plays it past the listener with its pitch shifting along each path, and prints the listener's L_Amax, sound exposure level and L_Aeq over the pass-by; `--levels pass.csv` saves the level over time, A-weighted and per octave band (see Moving sources and Doppler). Over a hard ground, the sound is within 45 dB of the exact field at 30 m/s (see Validation). A walking listener passes standing sources the same way.

Helicopters

From Python, `acoustic_spaces.rotorcraft` gives the sound a helicopter's two rotors send each way, as an `Emission` for a walk with Doppler (see Moving sources and Doppler):

- `trim` balances a `Helicopter` flying level at an airspeed by momentum theory: the main rotor's thrust holds up its weight and holds back its fuselage's drag, the disc tilted forward between them; Glauert's inflow gives the induced velocity through the disc, and the tail rotor's thrust holds the main rotor's torque.
- `emission` then gives each rotor's **tones**, the harmonics of its blade passing frequency from its blades' steady lift, drag and thickness (the Ffowcs Williams and Hawkings equation far from the rotor, Farassat's formulation 1A, up to the harmonic 90 dB under the loudest), and each rotor's **broadband noise**, louder ahead and quieter behind as a moving source is. It is the pressure 1 m from the main rotor's hub, in pascals, as the far field has it, in the helicopter's frame: x along its flight, y to its left, z up; a walk with `face_forward` turns it along its path.
- The broadband noise comes from one of two models, the helicopter's `broadband`: `"pegg"`, the default, is Pegg's semi-empirical method (NASA TM-80200), a fit to measured rotors' noise, one spectrum for each rotor round a peak set by its thrust and its tip speed. `"bpm"` is each blade section's own noise in the flow past it, by Brooks, Pope and Marcolini's airfoil self-noise (NASA RP-1218, `acoustic_spaces.airfoil_noise`): the turbulent boundary layer at the trailing edge, the laminar one's vortex shedding, the trailing edge's bluntness and the tip's vortex, for 24 sections of each blade at 144 places round the turn, each at its own speed and angle of attack from the blades' trimmed pitch (`rotorcraft.pitch`), its bands moved by its own Doppler shift. A `Rotor` gives the blades' twist, lift slope, trailing edge and tips, and whether their boundary layers are tripped. `rotorcraft.broadband(trim, ..., model="bpm")` picks one model for a call. BPM's sum over the blades does not depend on the noise's length: `rotorcraft.bpm_powers` finds it once, for `broadband(..., bands=...)` to send the noise at any length.
- `AS350_B3` is an Airbus Helicopters AS350 B3 at its maximum mass of 2,250 kg, from its type certificate and NASA's flight tests of the type; its main blades' chord and twist and its tail rotor's chord and thickness are assumed (the source code says which is which).

```
from acoustic_spaces import rotorcraft
from acoustic_spaces.walkthrough import render_walk

trim = rotorcraft.trim(rotorcraft.AS350_B3, 63.0, temperature_c=25.0)  # 63 m/s
sent = rotorcraft.emission(trim, seconds=30.0)  # 48 kHz: tones and broadband noise
sound = render_walk(walk, {"helicopter": sent})  # its walk with doppler: true
```

The tones agree with Gutin's formula for a hovering rotor to within 3 x 10^{-15}, and with Farassat's formulation in time, blade point by blade point, to within 7.5 x 10^{-6} of the loudest harmonic for the AS350 B3's main rotor and 1.6 x 10^{-5} for its tail rotor; Pegg's worked example comes out as he gives it, and the airfoil self-noise matches the two sample outputs of its authors' code to within 0.01 dB (see Validation). Flying the AS350 B3 over a microphone as its noise certification does, 150 m up at 63 m/s, the two models fall on either side of the certified 84.6 dB(A) of sound exposure:

broadband noise	sound exposure level	against the certified 84.6 dB(A)	main rotor’s broadband alone
Pegg’s (`"pegg"`, the default)	88.0 dB(A)	3.4 dB over	87.3 dB(A)
each blade section’s own (`"bpm"`)	80.1 dB(A)	4.5 dB under	68.6 dB(A)

With Pegg’s noise the main rotor’s broadband noise is nearly all of it. Pegg’s method is a correlation of the 1970s’ measurements; in his own worked example the band of the broadband peak comes out 5 dB over the level measured there, though a tone of the tail rotor falls in that band too. The blade sections’ own noise is 19 dB under Pegg’s for the main rotor, and the tail rotor’s tones (79.8 dB(A) on their own) are then nearly all of it: self-noise is the noise of a blade in a smooth flow, and leaves out the blades cutting through the wake of those ahead and through turbulence, which a full-size rotor in flight makes and which Pegg’s fit to measured rotors holds.

A scene’s source carries the sound power and the directivity of what the rotors send, for the levels the simulation and the pass-by report and for the late part: `sent.sound_power_db(trim.density, trim.sound_speed)` gives the one per octave band and `sent.directivity()` the other, which `acoustic_spaces.directivity.write_directivity_csv` writes as a table a scene reads (`directivity: {file: helicopter.csv}`). Both are in IEC octaves, so the main rotor’s tones at its blade passing frequency and at twice it (19.7 and 39.4 Hz for the AS350 B3), below the 63 Hz octave, are in no band (the sound sent each way). The `helicopter_flyover` example has them for an AS350 B3 150 m up at 63 m/s against a light wind, and its flyover demo flies it over a meadow, its rotors trimmed at its airspeed: its speed over the ground less the wind at its height.

What it leaves out: loads that change round the turn (in forward flight the advancing blade carries less lift than the retreating one, and the blades flap), the blades meeting the tip vortices of those ahead (blade-vortex interaction, the slap of a descent or a turn), the engine and the gearbox, and the fuselage’s reflection and shadow. `trim` takes the airspeed it is given, through still air: the flyover demo gives it the speed over the ground less the wind.

Aircraft by ECAC Doc 29

`acoustic_spaces.doc29` gives the noise of one movement of an aircraft, a departure or an arrival, at receivers on the ground by ECAC Doc 29, the European method for noise contours around civil airports (*Report on Standard Method of Computing Noise Contours around Civil Airports*, 5th edition, 2026). Like the vibration below, it is an assessment by the method's own empirical curves and corrections, apart from the scene's sound: nothing of it goes through the simulation.

- **The aircraft** (`doc29.tables`): its NPD curves, the sound exposure level (SEL) and the maximum level (LAmax) at 200 to 25,000 ft from a straight, steady flight at 160 kt for each power, its engines' installation (on the fuselage, under the wings, propellers) and its fixed-point profiles, the height, speed and power along the ground track from the start of roll or to touchdown.
- **The flight path** (`doc29.flight_path`, Volume 2 section 3.6): the profile along a ground track (`doc29.Track`), carried on to the track's ends and the track to the profile's, cut into straight segments: the take-off and landing rolls in steps of 10 m/s, the initial climb and the final approach below 4231 ft at the heights that keep the lateral attenuation from changing more than 1.5 dB along a segment, and airborne segments whose speed changes by more than 20 kt in steps of 10 m/s; the track's nodes joined in, and no source lower than 1 m.
- **Each segment's level** (`doc29.segment_levels`, chapter 4) at each receiver: the NPD curves at its power and distance (no nearer than 30 m), corrected for its speed against 160 kt, the engines' installation round the roll axis, the lateral attenuation of sound passing low over the ground, its finite length and, behind a take-off roll, the start-of-roll directivity. The caller passes the adjustment for the air's impedance at the aerodrome against the NPD curves', `impedance=doc29.impedance_db(temperature_c, pressure_kpa)`; left out, it is nought: the NPD curves' reference day, 25 °C and 101.325 kPa (the standard atmosphere at sea level adds 0.074 dB).
- **The event** (`doc29.event`): the segments' energy summed to the SEL, their greatest level the LAmax. Beyond the NPD curves' powers their levels are extrapolated from the two curves at that end (eq. 4-3); where that takes a segment's level more than 5 dB from the nearest curve's, the event warns (`doc29.NpdExtrapolationWarning`), as Volume 2 section 4.2 cautions.

`doc29.volume3` holds the reference cases of Doc 29 Volume 3: three hypothetical aircraft, which rely on no data of the ANP database (a jet with its engines on the rear fuselage, `JETF`; one with them under the wings, `JETW`; a turboprop, `PROP`), each arriving and departing on a straight and a curved route at a hypothetical aerodrome, and the levels Volume 3 gives at its 18 receptors:

```
import numpy as np
from acoustic_spaces.doc29 import volume3

case = volume3.Case.named("JETFDC")  # JETF departing on the curved route
levels = case.event(np.array([[6500.0, 0.0, 0.0], [8200.0, -1800.0, 0.0]]))
levels.sel, levels.lamax  # dB(A) at each receiver (x, y, z in m)
```

An arrival's profile measures its distances to touchdown and its track from the landing threshold: `flight_path(..., threshold_m=...)` takes the profile's distance at the threshold. Volume 3's reference cases put it at the approach's lowest node above the runway (`doc29.path.lowest_approach_node`), the jets' profile point at 15.2 m, 290.2 m before touchdown, and the turboprop's cut at 17.2 m, 328.1 m before it.

Flight profiles from procedures (`doc29.performance`, Volume 2 Appendix B). An aircraft's fixed-point profile can also be computed from the steps of its procedure (the take-off; climbs at a constant calibrated airspeed; accelerations at a rate of climb or with a share of the thrust; level flight; on approach the descents, level and idle steps, the landing and the decelerations on the runway), its engine and aerodynamic coefficients and its weight, at the aerodrome's elevation, temperature and sea-level pressure and into its headwind (`performance.Aerodrome`; by default sea level in the standard atmosphere into 8 kt, the conditions the coefficients are given for). Where the thrust changes from one step to the next, a point 1000 ft on (or half the step, if that is shorter) takes the change, as Appendix B does. The calculation is in feet, knots and pounds, as Appendix B's is; `performance.as_profile` gives the points in metres for `flight_path`:

```
from acoustic_spaces.doc29 import performance, volume3

plane = volume3.performance()["JETW"]
steps = volume3.departure_procedures()[("JETW", "ICAO_A")]
hot = performance.Aerodrome(elevation_ft=5000.0, temperature_c=40.0, headwind_kt=0.0)
points = performance.departure(plane, steps, plane.weights[("D", "1")], hot)
profile = performance.as_profile(points)  # for doc29.flight_path
```

The ANP database, the method's data for real aircraft (EASA's Aircraft Noise and Performance database), is not part of the package: its terms of use (issue 3, 2024) allow the data only for noise contours of an average fleet and traffic, not for benchmarking aircraft types, and forbid publishing it or passing it on to anyone who has not accepted them. With a copy downloaded from EASA under those terms, `doc29.anp.read_database(folder)` reads its tables from the folder by their files' names (aircraft, NPD, spectral classes, fixed-point profiles, jet and propeller engine coefficients, aerodynamic coefficients, default weights, departure and approach procedural steps), and `Database.profile` flies an aircraft's procedure where the database has it, else its fixed-point profile. A procedure gives the corrected net thrust in lb, so `Database.profile` refuses it for an aircraft whose NPD curves are in another power parameter (shaft horsepower in %, for instance). The readers take the fields Volume 2 Appendix G lists, as Volume 3's workbooks lay them out: a fixed-point profile's distances, heights and speeds in the unit its column's name gives in brackets (m, ft or feet; m/s, kt, kts or knots), the lateral directivity by how it starts (Fuselage, Wing or Prop); anything else is an error. They have not been tried on a download of the database.

Its levels agree with all of Volume 3's reference results: the 99 levels at its receptors within 0.0055 dB, its 12 grids of 37,458 points at 0.003 dB rms, and its 3516 rows of segment results term by term, the segments' places within 7.3 m: the differences come from where Volume 3 puts its cuts by speed and from its conversion of metres to feet (the check). Its flight profiles agree with all 26 of Volume 3's (Part 2: JETW arriving and departing and PROP departing, at sea level and at 2000 and 5000 ft, at 15 and 40 °C, into headwinds of -5 to 8 kt): their 314 points' heights, speeds and thrusts to Volume 3's rounding, their distances within 0.15 ft. Volume 3's profiles differ from Volume 2's printed equations in three places, where the package follows Volume 3 (the check).

The NPD curves in the aerodrome's air (`doc29.atmosphere`, Volume 2 Appendix D). The NPD curves hold the attenuation of the AIR-1845 atmosphere, an average of no one temperature and humidity. `atmosphere.adjusted(curves, spectral_class, temperature_c, humidity_percent, pressure_kpa)` adjusts them to another air by Appendix D's three steps: the spectral class taken back to the aircraft, out to each NPD distance through both atmospheres, and the difference of their A-weighted levels added to the curves. The aerodrome's absorption is ISO 9613-1's, averaged over each third octave. It gives Volume 2's worked example (JETW's spectral classes at 10 °C, 80 % relative humidity and 101.325 kPa, its tables D-3c and D-4) within 0.1 dB, the tables' rounding.

An aircraft in a scene (`doc29.source`). Doc 29 gives an aircraft's sound as levels at receivers, so a source of a scene that sends the same sound is made from them:

- *its sound power* from its spectral class (the spectrum 1000 ft under it at the time of its maximum level), scaled to its NPD curves' maximum level 1000 ft away at its power, with the AIR-1845 attenuation over those 1000 ft added back and the reflection off the soft ground the NPD curves are measured over (1.2 m above it) taken out: grass, its reflection's energy added to the direct sound's (`sound_power_db`, `ground_gain_db`). That maximum level is read with a sound level meter's slow time weighting (1 s), which lags behind the passing aircraft and reads its loudest moment low, so the level 1000 ft away is raised by what it takes off the NPD curves' own pass at 160 kt in the scene's air (`slow_drop_db`; 0.29 dB for JETW at 15 °C and 70 % humidity);
- *its directivity* from its engines' installation correction (eq. 4-15), by the depression angle across the flight path, the same in every band; nought straight down, where the NPD curves are measured, so the sound power is raised to keep the level there (`directivity`, `below_offset_db`);
- *its thrust along the flight*: its sound power changes with the thrust as the NPD curves' maximum level 1000 ft away does, its spectrum kept (`power_change_db`);
- *its flight* as a walk's keyframes (`flight`): the flight path's segments flown as Doc 29 interpolates them (along a segment the square of the speed changes evenly with the distance), so each place is reached when the aircraft gets there, the aircraft facing the way it flies.

The airport example's flight demo flies Volume 3's JETW this way. The NPD curves are measured at 160 kt and give no directivity along the flight path but behind a take-off roll (eq. 4-22): the source keeps none of that either, and the engine reckons the ground, the air and the lateral spreading in its own way, so its levels differ from Doc 29's; how far is measured.

What it leaves out for now: bank angles from the turns (a segment takes the bank it is given; Volume 3's cases fly level, and the performance calculation flies turns as straight), the dispersion of tracks round a backbone, the line-of-sight blockage of terrain, levels summed over many movements (LAeq, Lden) and contours. Volume 3 gives no reference maximum levels, so the LAmax is checked only against Volume 2's equations, and no idle steps of an approach away from sea level in the standard atmosphere, which Appendix B adjusts so that the aircraft slows as it does there: those are checked against that definition only.

Vibration through the ground

Trains, trams and construction plant shake the ground as well as the air: a building near them feels it, and its rooms rumble with the sound their floors, walls and ceilings radiate (ground-borne noise). `acoustic_spaces.vibration` gives both by the general method of the US Federal Transit Administration (FTA, *Transit Noise and Vibration Impact Assessment Manual*, FTA Report No. 0123, 2018). It is an assessment from the manual's curves and tables, apart from the scene's sound: nothing of it goes through the simulation.

- **Trains and trams** (the manual's General Vibration Assessment, section 6.4). A base curve gives the overall rms vibration level of the ground 10 to 300 ft (3.0 to 91.4 m) from the track's centreline, as far as the manual draws it (figure 6-4, the equations of table 6-10): one for trains hauled by locomotives, one for rapid transit and light rail (multiple units and trams too), one for a bus on a smooth road (`vibration.CURVES`; each train of the railway demo has its own, `rail.Train.curve`). The level changes with the speed as 20 log10 of it (eq. 6-4) and, in `vibration.Line`, with the vehicles' suspension and wheels, the track's condition and special trackwork near, a floating slab, ballast mats, resilient fasteners or ties, an elevated structure or a subway's, and the ground: +10 dB for soil that propagates efficiently, +2 to +9 dB for a rock layer (tables 6-11 and 6-12, with the manual's rules for which add up). `vibration.indoor_db` takes it into a building and up to a floor: the foundation's coupling loss, 2 dB a floor up to the fifth and 1 dB a floor up to the tenth, and 6 dB for the floors' resonances (tables 6-12 and 6-13). Where a building's structure is not known, the manual's conservative change from outdoors to indoors stands for all three: +3 dB for a small masonry building or a light wood-frame house's first floor, +6 dB for its second and third, 0 dB for a large building. `vibration.noise_level` gives the rumble in dB(A): the room's vibration level less 50, 35 or 20 dB as the vibration peaks below 30 Hz (most surface track), at 30 to 60 Hz (most subways, surface track on very stiff clayey soil) or above 60 Hz (subways in rock) (table 6-14).
- **Construction plant** (section 7.2). Table 7-4's 14 kinds of plant, from an impact pile driver to a small bulldozer: the peak particle velocity (PPV) and rms level 25 ft away, falling as (25 ft / D)^1.5 and 30 dB a decade (`vibration.plant_ppv`, `vibration.plant_level`, eq. 7-2 and 7-3), against the PPV that may damage four kinds of building (`vibration.DAMAGE`, table 7-5), from 12.7 mm/s for reinforced concrete, steel or timber to 3.05 mm/s for buildings extremely susceptible to it. The street's breaker is the table's hoe ram, its dumper its loaded trucks (`plant.Machine.vibration`).
- **Limits** (tables 6-2 to 6-4, `vibration.criterion`): the indoor vibration and rumble above which there is impact, for buildings with equipment sensitive to vibration, for residences and for institutions used mostly by day, and for concert halls, studios, auditoriums and theatres, by how many events a day: frequent (more than 70), occasional (30 to 70) or infrequent (fewer than 30). For residences with frequent events they are 72 VdB and 35 dB(A).

Levels are in VdB, decibels re 1 micro-inch per second (25.4 nm/s), as the manual gives them, and a PPV in mm/s and in/s.

The railway example's `vibration.py` stands a house where each of the scene's listeners in use is (or those `--listeners` names, in that order), as far from the track, or takes the scene's house a listener stands in front of (`house_18m`'s, its wall 20 m from the track), and prints for each of the demo's trains at its own speed the vibration of the ground outside, of the house's floor and the rumble in its rooms against the limits for residences, and from how far each limit holds. The house is a small masonry building whose structure is not known, the trains the scene's one regional train an hour, 24 a day, which are infrequent events; `--building`, `--floor`, `--ground`, `--peak` and `--trains-per-day` change them. A freight train's locomotive counts as an infrequent event and its wagons, which pass for minutes, as frequent ones, as the manual says; it puts the locomotive "often" 3 to 8 dB over the wagons, and the wagons are taken 3 dB under it. 25 m from the track:

train	speed	ground	indoors	rumble	vibration within its limit from	rumble from
S-Bahn	120 km/h	72.7	75.7	25.7	14 m	3.0 m
regional	160 km/h	86.3	89.3	39.3	63 m	16 m
intercity	200 km/h	88.2	91.2	41.2	76 m	20 m
freight, its locomotive	100 km/h	82.2	85.2	35.2	43 m	9.7 m
freight, its wagons	100 km/h	79.2	82.2	32.2	69 m	18 m
tram	50 km/h	65.1	68.1	18.1	3.5 m	3.0 m

(VdB, the rumble in dB(A); the limits are 80 VdB and 43 dB(A), the wagons' 72 VdB and 35 dB(A); 3.0 m is where the curves start.) The street's `vibration.py` does the same for the tram, 6.0 m from the houses, its passes frequent events as the manual has most rapid transit's: 77.6 VdB indoors, 5.6 dB over the limit, and the rumble 27.6 dB(A), 7.4 dB under it. Its breaker, 4.0 m from the houses, gives them a PPV of 5.94 mm/s, over the damage criteria of the two most fragile kinds of building, and 95.4 VdB, 23.4 dB over the limit for residences; the dumper gives 5.08 mm/s, over the most fragile kind's and 0.1 % under the next one's (0.1998 against 0.2 in/s), and 94.4 VdB. The office block, 8.0 m from both, stays under every damage criterion; the method gives it no limits for annoyance, as it counts offices only where something in them is sensitive to vibration. Both scripts run from the editor's **Run demo** window. The pass-by demos play the rumble in the room of each listener in front of a house, with the train or the tram through a closed window (`--indoors`, see Trains passing by), and the road works demo the breaker's and the dumper's (see Road works in the street); at every listener their captions say how hard the ground or the floor shakes and whether that is felt, and a bass shaker's track plays it.

What the method leaves out: it gives overall levels, not their spectra, so the rooms are not simulated. Its curves are the upper range of measurements of North American systems in good condition, which the manual says ground-borne vibration rarely exceeds by more than 1 or 2 dB unless wheels or rails are worn; European trains and soils may differ. Its table of speeds runs from 20 to 60 mph (32 to 97 km/h); the faster trains take its 20 log10 of the speed further. The manual's detailed method works band by band from a train's force density and a site's transfer mobility, which it says to measure (see the roadmap).

In the editor

The editor builds and edits outdoor scenes too: see Outdoor scenes.

What is not modelled yet

The wave solver's moving air (`simulation.wave.wind: moving`) runs on the CPU only, and leaves out the term of the linearised Euler equations that the wind's change with height puts on the air's own motion, about |dv/dz| / ω of the rest (under 0.3 % at 63 Hz a metre above the ground in the `windy` preset). Round the model's solids its wind is an ideal fluid's (see How the simulation bends sound): no wake and no eddies behind a building, whose lee is as windy as its front, and no separation at a sharp edge, within a few metres of which it is too fast. Between weather zones its wind passes gradually, over a distance that grows with the grid's reach. The air of the grid's absorbing layer is still, so a wave that leaves the moving air comes back from its border by about half the wind's Mach number. By default it takes the wind as the rays and the parabolic equation do, as an effective speed of sound, so sound that a building sends back towards the source travels at the speed of sound going away from it. It leaves out vegetation and the eddies.

Vegetation is volumes filled evenly: a lone tree's trunk is left out, the foliage's attenuation is given, not derived from its leaves' size and density, and stems stand upright at random.

Behind barriers and buildings in wind (see How the simulation bends sound) the parabolic equation leaves out:

- **The barrier's own wind.** The wind blows through barriers and buildings as over the open ground; behind a real barrier it is slower and gustier, which changes how much it screens by 3 dB or more (Salomons 1999; Aballéa et al. 2007), and Nord2000 parts by up to 6.5 dB from a parabolic equation that leaves it out (Plovsing 2006).
- **Sound going back.** The equation marches away from the source only: nothing reflects between two barriers or back towards the source. Behind two barriers 10 m apart the weather's effect is up to 1.73 dB off from 63 to 500 Hz; 40 m apart (3 and 5 m high) up to 4.95 dB at 250 Hz in 8 m/s of wind downwind over rigid ground (2.23 dB in 5 m/s, 1.28 dB over grass), where the full-wave solution's level changes sharply with frequency, as the sound goes back and forth between them.
- **Steep angles upwind.** The narrow-angle equation is exact only near the horizontal: in 10 m/s of wind upwind the effect comes out 0.2 to 1.64 dB low from 63 to 500 Hz, the more the higher the band and the farther the listener (1.0 to 1.4 dB at 250 Hz 120 and 200 m behind a barrier 4 m high 15 m from a source 1 m up, where the wide-angle equation is 0.45 dB low 120 m away); in 5 m/s within 0.71 dB.
- **The third dimension.** The equation runs in vertical planes along whole degrees, so a barrier is a ring round the source; how obliquely the way crosses it changes the level in still air by 0.4 to 0.9 dB from 0 to 45 degrees (Aballéa et al. 2007), which the ratio cancels in part. Against a run along the listener's own way the blend of two degrees is within 0.9 dB 120 m behind a barrier at 14 degrees to the way and 1.8 dB within 3 m behind a wall at 45 degrees. The paths round an end change as the open ground's field along their way does, not as the barrier's, while those over the top take the barrier's, past the end too as they fade out there: across the shadow boundary of a barrier's end the level steps by at most 0.07 dB a millimetre in octave bands 60 m away and 120 m downwind, and by 0.22 dB 120 m upwind, where still air steps by 0.17 dB (0.28 dB in the 50 Hz third octave, where still air steps by 0.30 dB), but at 4 and 8 kHz with eddies by up to 1.08 dB (strong ones, 120 m downwind; in still air the same eddies step there by up to 0.95 dB) and with weather zones by up to 0.76 dB. Where the end stands below the line of sight (a barrier 1 m high), a listener crossing the end's line seen from above (its way crossing the barrier on one side, and on the other the open ground's sound) steps, 1 mm across it, by up to 0.94 dB from 63 to 250 Hz, but from 500 Hz to 2 kHz by up to 4.7 dB 100 m away and 6.0 dB 200 m downwind, where the barrier's runs and the open ground's own field disagree, and 12.8 dB upwind at 8 kHz, where the barrier's runs and the open ground's shadow do (see Validation). No reference in three dimensions checks the paths round ends and corners, for listeners behind a barrier or in the open beside it.
- **Buildings in the wide-angle equation** (at steep points, or with `parabolic_equation: wide_angle`) are two screens at their walls: up to 1.4 dB in the wind's effect behind a building 60 m deep at 500 Hz in 10 m/s.
- **Objects of a model file** are not in the equation: where one stands in the way the paths go over the edges with straight legs.
- **Deep shadows.** Deeper than 25 dB below still air the effect is not bounded: in the checks it is off by up to 4.2 dB 39 dB below still air, 20 dB 52 dB below and 17 dB 74 dB below.

The levels in still air the effect multiplies keep their own errors (see Validation). The standards' corrections for the weather over a barrier (ISO 9613-2's and CNOSSOS-EU's path curved over the edge, Nord2000's refraction-modified wedges) are not offered.

The eddies are the same over the whole scene (weather zones change the wind and the temperature, not the eddies), and the eddies' largest size is the same at every height. A weather zone's border is a step (but for the wave solver's moving air), and the early paths take one mean profile along their way, so a wind that changes on the way tells on them by a few decibels in the bands the rays give (see Weather zones). The ground reacts locally unless `reaction: extended` says otherwise, and the wave solver's ground always does: one impedance for every angle, which holds for grass, soil and snow, less for very soft snow and thin layers.

Over a terrain the early sound reflects off the mean plane of the ground's profile, not off its facets, so a listener high over rolling ground gets one reflection where the real ground glistens with many, and a valley between the source and the listener reflects as its mean plane does, not as a bowl. A rounded crest is a wedge of its chords with a thick screen's extra screening, an engineering model as in the standards, where a ridge is exact. Sound bent over a hill and a building in turn is left out, and a terrain cannot be combined with wind or temperature gradients yet. The editor shows a terrain and the buildings of a map but does not draw either: they are set in the scene file.

Sound that reflects off the ground both before and after bending over an edge (two reflections and a diffraction) is left out unless `simulation.early.diffraction_reflections` is `both_sides`: by default a path reflects before or after the edge, not both. Over a rigid ground that path carries a quarter of the sound over a barrier, so without it the barrier's insertion loss comes out up to 0.8 dB off from 1 kHz up (mostly too high, 0.8 dB too low at 2 kHz 90 m behind) and a few decibels off in the bands below, where the paths interfere; with it the engine is within 0.15 dB of the exact solution (see Validation). Over grass and other soft grounds, whose reflections near grazing are weaker, the path left out should matter less, but that is not checked. Sound that goes over a barrier, down to the ground and over the barrier again is left out either way.

Demos

Five demos run recordings through example scenes, mix them at the scenes' listeners and make a video of the result, a sixth drives a car, a truck, a bus, a motorcycle, a police car with its siren on or a fire engine with its siren and its air horn past the listeners of two outdoor examples, a tram down the street of one of them and a train past those of a third, a seventh flies a helicopter over those of a fourth, an eighth flies it over the stadium's concert, a ninth plays road works in the street and a tenth flies a jet taking off from an aerodrome and landing on it. Each shows one space and one kind of sound: a symphony orchestra on the stage of a shoebox concert hall, a choir and an organ in a reverberant stone church, a band with its sound system and stage monitors in a stadium, the same band's song carried across a lake on a summer evening, a guide talking to a visitor who walks away from them through a gallery and then turns their head, a vehicle passing on a road across a meadow and along a street and a train on a railway across a meadow, its pitch falling as it goes by, a helicopter flying over a meadow and low over the stadium during the song, a breaker, a generator and a dumper reversing with its alarm on at road works in the street, and a twin jet climbing away from a runway and coming down to it. None of them stands for what the simulator does elsewhere; the examples cover far more, and any scene makes a demo of its own with the recordings its sources are given (a demo of any scene).

The scripts come with the examples: `acoustic-spaces examples --copy . NAME` copies them with the scene. The recordings do not: they are other people's work under their own licences, named below with each demo, and the scripts download them from where their authors publish them. The videos need ffmpeg on the PATH and the pictures Pillow (the `viz` extra brings it). The mixes are binaural: use headphones. They go through the spherical head model unless every step and the `simulate` command get `--hrtf me.sofa`, a SOFA file of your own or measured HRTFs (the `simulate` command a script prints carries it; the `sofa` extra reads the file). The videos are in one playlist.

Running a demo

Every demo is a script of steps: it writes its scene, simulates it, mixes what each listener hears and puts the video together. A step is run by name, and the steps of a demo are listed by `python NAME_demo.py --help`:

```
python church_demo.py music    # the recordings it plays, downloaded
python church_demo.py scene    # the scene the demo simulates
python church_demo.py simulate # acoustic-spaces simulate on that scene
python church_demo.py all      # every step in order, leaving out what is done already
```

`all` is the short way through: it skips the steps whose result is in the work folder already (the scene file, the simulation's results, each stop's mix, for the run's `--format` and `--hrtf`), so a run that stopped halfway carries on where it left off, and a change to one step is seen in the video without simulating again. `--again` mixes the stops afresh even where their files exist, `--work FOLDER` puts everything somewhere else, and `--no-mp4` writes the audio and the stills without calling ffmpeg. Where a step needs an earlier one, it says which command to run.

`--listeners a,b` has a demo play those listeners alone: it mixes them and its video tours them, leaving the others out. Every demo with several listeners takes it, with the names the scene gives them (a name the demo cannot play is refused with the list of those it can). The simulation still takes every listener in use, so another choice later needs no new simulation, except in the pass-by demos, which simulate the vehicle along its walk at each listener they play (a pass made already is kept). The demo of any scene and the pass-by demos visit them in the order given; the others keep their own tour's order: the church's tour keeps its times, a place left out giving its time to the one before; the stadium's and the lake's stops there keep their phrases, the last running to the end; the orchestra plays each movement at the seats of its tour among them, or at all of them when its tour has none. A tour's crossfade is never longer than the shorter of the two stops it joins. The gallery walk hears its visitor alone and takes no `--listeners`.

`--input SOURCE=WAV`, once for each source, has a source play a recording of yours instead of what the demo gives it: `python lakeside_demo.py render --input subs=bass.wav`, or a singer of the choir (`--input singer_f1=take.wav`), a player of the orchestra (`--input fl1=flute.wav`, in every movement), any source of the stadium. The file is mixed down to mono, brought to the gated RMS of the feed it replaces (unweighted, so a recording with another spectrum plays louder or quieter in dB(A)), and cut or padded with silence to that feed's length, so the tour's times hold; a name the demo has no source for is refused with the list of them. The mixes are made again when the inputs change. The credit line on the stills names the files that replaced a recording (and only them when every source has one); the captions go on describing the demo's own music. The walk through the gallery takes `--input guide=talk.wav`, played as it is: the video's one gain sets how loud it is heard, and the captions' level and STI stay the scene's voice; the pass-by demos take `--input car=engine.wav`, a dry recording played as the car's sound.

`--format` mixes the stops in another output the simulation wrote, by the label `simulate` gives it (`ambix3`, `5.0`, `7.0.4`, `mono`…), instead of binaural. The tour still plays at the two ears: Ambisonics through the binaural decoder of its order, and a loudspeaker layout as virtual speakers, each speaker's channel through the HRTF of its direction (the LFE channel, which the simulator leaves silent, left out), both through the HRTFs the scene's binaural output names or the spherical head model. A mix of one channel plays in both ears, and one of two (a stereo microphone pair, the stereo layout) as it is.

In the editor, **Simulation > Run demo…** does the same for the open scene: *This scene* (below), and for an example's scene, or a copy of one, the example's own demos and its scripts that are no demos. The window shows the options that change what the demo plays, at the command line's defaults, the file each source plays, the listeners to play and the walkers that move; **Advanced** holds the step, the paths and the command line, to repeat the run in a terminal. A line above **Run** says what the run downloads and how much. The demo reads the scene's file, so the editor asks to save a scene with no file or with changes first. However it closes, the window keeps the options changed in it for the next time, each demo its own: a pick from a list, a number, a tick and a file of yours (the HRTFs, a song with its credit) for any scene, and what names something in the scene (the listeners, the files its sources play, what moves, the format, a tour) for that scene file, which also keeps the demo picked; **Defaults** puts them back. The folders and files a demo writes and the scenes it reads follow the open scene.

Every still says how its sound was made, at the bottom: the version, the early reflections (traced or image sources, their order, diffraction) and the late sound with its rays, as the simulation ran them, the wave solver's bands when it ran, outdoors in moving air the bent paths and the parabolic equation's bands, the HRTFs the mixes went through (or the spherical head model) and the sample rate; and under that the command that makes the video again, with the options it was given and paths by their names (the song, the HRTFs).

A demo of any scene

Any scene makes a demo: each source plays a recording, every listener is heard in turn, and a video tours them. A source names what it plays in the scene file, or with **Plays** on its page in the editor (**Choose WAV…**; **Silent** takes it away):

```
sources:
  - name: talker
    position: [2.0, 3.0, 1.5]
    level_db: 78
    spectrum: speech_male
    signal: sounds/talk.wav   # a dry recording, relative to the scene's folder
```

A sound system plays one recording through all its loudspeakers instead: the scene names it once, as its `recording` (**Recording** on the scene's page in the editor), and each loudspeaker that plays it gets a `channel` (**Channel** on its page): `left` or `right`, or `mono`, both mixed down, as subwoofers and loudspeakers in the middle play. The recording is a file of yours, such as a band's stereo mix (WAV, or anything ffmpeg decodes), or the name of a free one, downloaded the first time a demo plays it:

```
recording: spicy_funk_cake    # or a file of yours: music/our_band.flac
sources:
  - name: house_left          # on the audience's left
    position: [-40.0, -19.0, 14.0]
    channel: left
  - name: subs
    position: [-41.0, 0.0, 0.6]
    channel: mono
```

Free recording	What it is	Licence	Download
`spicy_funk_cake`	"Spicy Funk Cake" by the Rhythmussportgruppe, a nine-piece funk band, its stereo mix (Zenodo)	CC BY 4.0	61 MB
`great_british_summer`	"Great British Summer" by Tom McKenzie, its stereo master (Zenodo)	CC BY 4.0	50 MB
`potassium`	"Potassium" by Software-Entwicklungskit, its stereo master (album)	CC0 1.0	6 MB
`speech_male`	a man talking, anechoic: talker M4 of the ACE challenge corpus (Zenodo)	CC BY 4.0	16 MB
`speech_female`	a woman talking, anechoic: talker F2 of the same corpus	CC BY 4.0	49 MB

They are the songs the lakeside demo plays and the talks of the gallery's. A free recording is kept as one WAV file in the user's data folder (`acoustic-spaces/recordings` under `~/.local/share` on Linux, `~/Library/Application Support` on macOS, `%LOCALAPPDATA%` on Windows; `ACOUSTIC_SPACES_RECORDINGS` names another), so every scene that names it plays the same file and none downloads it again. The stills credit it as its licence asks unless `--credit` says otherwise. A source's own `signal` wins over the recording, and its `channel` then picks a channel of that file; a source with neither stays silent. The examples with a stage, a sound system or a talker name a free recording and give their loudspeakers or talker a channel, so their demo plays out of the box: concert_stage, stadium_concert, lakeside_concert (its stage; ticked, its PA group as lakeside_demo.py feeds it), open_air_stage and studio a song; town_hall_pa, lecture_hall, leaky_classroom, shoebox, bim_office, material_libraries and gallery the male talk. A stereo mix has none of the parts that the monitors or a band on the stage play, so they stay silent; the stadium demo plays its song's stems through them.

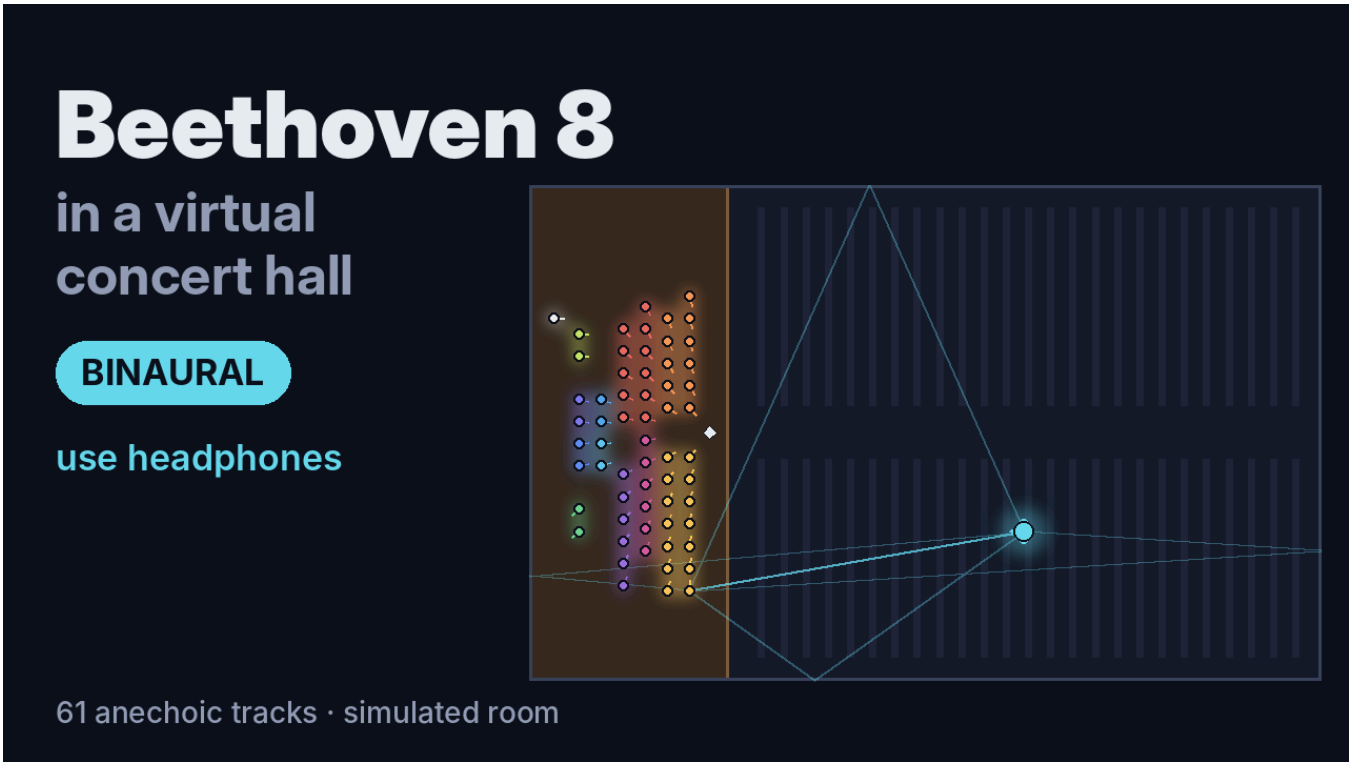
```
acoustic-spaces demo scene.yaml                # every step, into demo/ beside the scene
acoustic-spaces demo scene.yaml recording      # download the scene's free recording
acoustic-spaces demo scene.yaml feeds          # one step: feeds, simulate, render, video
acoustic-spaces demo scene.yaml --seconds 15 --listeners front,back --hrtf me.sofa
acoustic-spaces demo scene.yaml --moving none   # every walker standing where it starts
```

- **recording**: downloads the free recording the scene names, unless it is there or no source plays it; the editor's Run demo window says so above Run, with its size.
- **feeds**: each source's file (its `signal`, or the scene's `recording`), at the scene's sample rate, its `channel` or mixed down to mono. A file mixed down is brought to one gated RMS (unweighted); its left and right channels share one gain that brings their mean square to it, so a recording's stereo balance stays and how loud each source plays against the others is its sound power in the scene (`level_db`, with its feed's gain and equalizer), as the simulation says, and not how loud the recording happens to be. Sources with neither stay silent and are listed as such. `--input SOURCE=WAV` plays another file instead of a source's signal or the recording, without changing the scene (the source's `channel` is taken out of it in the same way), and gives a silent source one.
- **simulate**: `acoustic-spaces simulate` on the scene, its binaural output through the HRTFs the scene's binaural output names, `--hrtf` instead, or the spherical head model when neither names any. A scene changed since its results is simulated again.
- **render**: every listener in use (or those `--listeners` names, in its order; `--tour`, its earlier name, still works) hearing every source that plays, with one gain for all, so that the listeners' differences in level are heard as they are.
- **video**: `tour.wav` hears each listener for `--seconds` (20 by default) while the recordings run on, crossfading between them; a recording too short for every listener shares its length out. Each stop is turned up to the loudness of the loudest, by 40 dB at most and by as much as the still says, the only step that is not the simulation's: K-weighted, with the channels' powers added up as ITU-R BS.1770-4 adds them (the side surrounds of a loudspeaker layout 1.5 dB up, the LFE left out), an Ambisonics mix measured on its W channel alone. The stills show the scene from above, drawn from its own model, with its sources (those that play in colour, with a ray to the listener), the listeners and the one of the moment as a head, and beside it the A-weighted level there of the sources that play, as simulated, when their sound arrives, the reverberation time T30 at mid frequencies (the mean of the 500 Hz and 1 kHz bands) and the nearest source that plays. `--credit` puts a line about the recordings on the stills (who made them, their licence); without it the line gives the free recording's credit, when one plays, and says which source plays which other file.

Movement. Sources and listeners whose walk is on move along it while the recordings play, all on one clock from their start, as `walkthrough --all` plays them (walkthroughs). Then **render** simulates the scene with every walker at each moment of its walk, in one simulation, and mixes every listener hearing the walkers go by, or walking itself, with one gain for all; the still simulation stays for what the stills say where nothing moves. The video is a moving picture: the walks dotted, every walker where it is at that moment, and the level of the sources that play as simulated at that moment; the arrival time is left out where something moves, and T30 is the one where the walks start. A stop hears its listener where the clock has it then, so `--listeners` puts a walker first to hear its walk from the start. `--moving` picks the walks of a run without changing the scene: `--moving visitor,guide` moves those two and leaves every other walker standing where it starts, `--moving none` keeps them all still, so one scene serves both. A walk with Doppler (a car, a train) moves alone: every listener hears each early path through its own delay as it changes, so the pitch shifts as it passes, and the late sound crossfaded from place to place (moving sources and Doppler), all from the one simulation of the moving run, as the editor's **Render movement…** plays it. With other walks on as well, the demo says to leave them standing.

Everything goes to `demo/` beside the scene (`--work` elsewhere): the feeds, the results, the mixes in `seats/` (in `moving/` when something moves), `tour.wav`, the stills, `thumbnail.png` and `tour.mp4`. In Python the demo is `acoustic_spaces.demos.scene_demo` (`python -m acoustic_spaces.demos.scene_demo all --scene scene.yaml`).

An orchestra in the concert hall



`orchestra_demo.py` plays the three movements the TU Berlin's anechoic recording of Beethoven's 8th symphony has (one of the dry recordings to try; 70 to 85 s of each) in the concert hall example, binaurally (use headphones), and makes a video of about four minutes that tours its four seats through them: the first movement from the middle of the stalls, the front rows and the conductor's podium, the second from the back of the hall and the middle, the finale from the podium, the front, the back and the middle again. Each of the 61 tracks is one source on the stage, in German seating: violins on both sides of the conductor, the winds and brass on risers, every player facing the conductor with a directivity that narrows above 1 kHz, horn bells pointing back. The `orchestra` example is that scene, with the four seats from the conductor's podium to the back of the hall.

```
acoustic-spaces examples --copy . orchestra && cd orchestra
python orchestra_demo.py tracks   # demo/tracks: the 61 tracks of each movement (2.0 GB)
acoustic-spaces simulate scene.yaml --out results
python orchestra_demo.py render   # demo/seats: each stop of the tour, binaural (~1 min each)
python orchestra_demo.py video    # demo/tour.wav, demo/stills, demo/tour.mp4,
demo/thumbnail.png
```

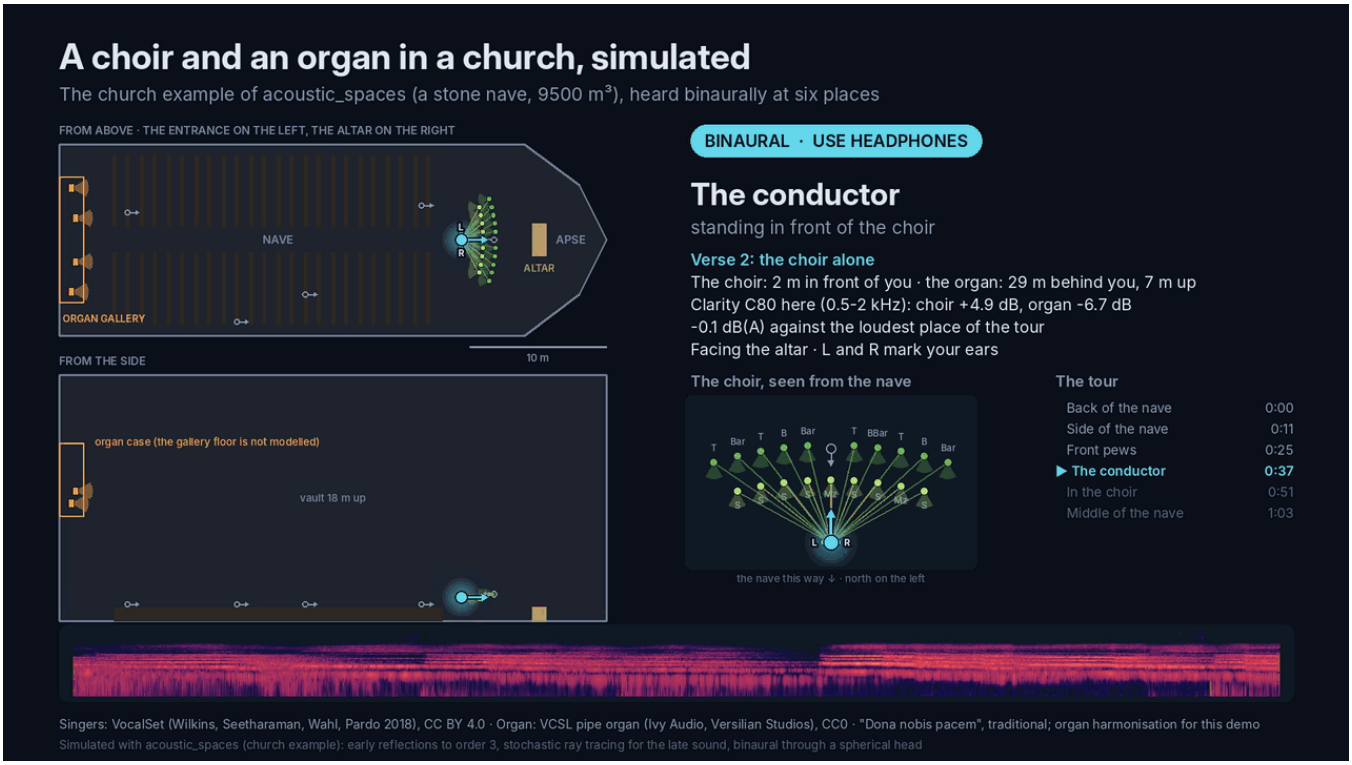
Options: `--tour 1:stalls-middle,1:conductor,2:stalls-rear` (the seats of a movement take equal shares of it), `--listeners conductor,stalls-rear` (each movement at the seats of its tour among them, or at all of them when it has none of them), `--zips FOLDER` for a folder with zips downloaded before, `--movements 1,2` for `tracks`, `--format` for another output the simulation wrote (the tour decodes it to two ears), `--no-mp4` for the audio and stills only. `demo/chapters.txt` lists the stops' times for a video description. `all` runs tracks, the simulation, render and video, leaving out what is there already.

- **The cuts.** Within a movement the seat changes at the quietest moment within 10 s of each equal share's end (the lowest half-second RMS of the first seat's mix), so the cuts fall between phrases, with a 1 s crossfade. Between movements the hall's reverberation dies away (the responses run 3.5 s) and there is a pause of 3 s. Every seat's mix is shifted earlier by the time the orchestra's sound takes to reach it (the peak of the energy of its responses: 10 ms on the podium, 85 ms at the back), so the music runs on across a cut.
- **One gain for all.** All stops keep one gain (the loudest peak at -1 dBFS): the hall's differences are heard as the simulation gives them, from the podium, 2 m from the first violins, to the back row, and each still says how far below the loudest stop it is. Every source has the same power, so the balance between the instruments is the recording's own.
- **The pictures.** `make_images.py` draws the cover (the hall from above with the orchestra, the seat and the first reflections from the leader to it, the sections and their numbers), the seating plan and the thumbnail with Pillow (`viz` extra); the demo adds the stop's facts and a bar across the top with the tour, the movements' numerals and the seats, the current stop bright.

One seat and one movement at a time still work as before: `auralize.sh` maps each source to its track and calls `auralize` (the mix normalized to -1 dBFS), `make_images.py --seat stalls-middle --movement 1` draws the pictures and `make_video.sh` puts a scrolling spectrogram of the mix under the cover.

Recording. Böhm, Ackermann and Weinzierl, *Beethoven, Symphony No. 8*, TU Berlin, https://doi.org/10.14279/depositonce-6729.2, CC BY 4.0: movements 1, 2 and 4, 61 mono tracks each, in a zip with the movement's mix (590 to 740 MB). `tracks` reads the 61 WAV files of each out of its zip on the TU Berlin's server by HTTP range requests, without keeping the zip: 640, 600 and 740 MB, 2.0 GB for the three (nearly the whole zip, as the blocks of 32 MiB it reads of the last tracks hold the mix), 2.6 GB on disk. It reads them out of the zip, or the folder it was unpacked into (`movementN/anechoic`), in `--zips` instead, the example's folder by default. Tracks there already are kept, so a download cut short goes on where it stopped.

A choir and an organ in the church



`church_demo.py` puts a choir of 19 real singers in front of the altar of the `church` example and the organ in the gallery over the entrance, and plays the round "Dona nobis pacem" three times: the organ alone on full stops, the choir alone, then the choir with the organ on quiet stops, slowing into a held last chord and the church's reverberation (1:26 in all). It mixes the piece at six places and makes a video that tours them: under the organ at the back, along the south wall, the front pews, the conductor, the back row of the choir and the middle of the nave. The choir, the organ's pipes and those places are the example's: its Choir, Organ pipes and Choir concert groups, left out of the church's own simulations until ticked in the editor, and its listeners `nave-back` and `nave-middle`.

```
acoustic-spaces examples --copy . church && cd church
python church_demo.py music     # 19 takes, their annotations, 27 organ samples (127 MB)
python church_demo.py scene     # scene_choir.yaml: the church with the choir and the organ
acoustic-spaces simulate scene_choir.yaml --out results_choir --output binaural   # 10-20 min
on 4 cores
python church_demo.py feeds     # demo/feeds: one WAV per singer and per side of the organ
python church_demo.py render    # demo/seats: the whole piece at each listener
python church_demo.py video     # demo/tour.wav, demo/stills, demo/tour.mp4,
demo/thumbnail.png
```

Options: `--tour nave-back:0,conductor:25,...` (listener:start in seconds), `--listeners nave-back,conductor` (those places alone, in `render` and `video`), `--format` for another output the simulation wrote (`ambix3`, `5.0`, …; the tour decodes it to two ears), `--no-mp4` for the audio and stills only. The scene and its simulation go beside the script (`scene_choir.yaml`, `results_choir`), the rest to `./demo` (`--work`). `all` runs every step in turn, leaving out what is done already.

- **The choir.** Each singer sang the tune alone in a recording booth, at their own tempo and a little off pitch here and there. `feeds` moves each take towards the tune (each singer's own offset fully, 60 % of each note's error) and warps it in time so that every note starts on the beat, from the dataset's note annotations; after that the voices start within about 40 ms of each other and sit about 16 cents from the tune on average, as a real choir does. The women stand in the front row, the men on a step behind them, 0.6 m apart, facing the nave (sources with a voice-like pattern: omni in the bass, cardioid from 2 kHz), 88 dB sound power each.
- **The organ.** A four-part harmonisation of the tune, written for the demo, played from a sampled church organ: an open 8' with the same rank an octave up as a 4' and the pedal with its 16' for verse 1, quiet stops 11 dB down for verse 3. As in a real organ, each division's pipes stand in two halves, C side and C# side, so the organ is four sources across the gallery, 9.5 m up for the manual and 8.6 m for the pedal towers, facing the nave (105 dB for each half of the manual, 102 dB for each pedal tower).
- **How loud.** The responses are per unit of sound power and `auralize` scales each source by its `level_db`; the feeds are brought to one gated RMS where they play loudest, so the balance between the singers and the organ is the scene's.
- **One gain for all listeners.** `render` scales the six mixes by one gain (the loudest peak at -1 dBFS), so the level differences between places are the simulation's.
- **The cuts** fall where the music breathes (the rest in the middle of the tune, the gaps between verses), with a 1 s crossfade. Time is the performers': a listener far from the choir hears it up to 80 ms later than one near it, as in the church.
- **The pictures** draw the church from above and from the side, each listener with an arrow the way they face and the current one with L and R at its ears, the sources as dots with see-through cones the way they face, and lines from each source playing to the listener, brighter where it is louder. The caption gives the clarity C80 there.

Limits. A "virtual choir": the takes were recorded separately, so the singers do not hear each other or the room; their timing and tuning are evened out by the script, not by listening. The organ samples hold a little of the room they were recorded in, and the organ is four points, not a case full of pipes. The model is the example's: the floor plan raised 18 m, with no gallery floor under the organ and no columns or furniture; `ir_length_s` is raised to 9 s, as the bass rings for nearly ten. Binaural is through the spherical head model unless `--hrtf` names a SOFA file.

Recordings and licences. `music` writes these credits to `demo/music/SOURCES.txt`.

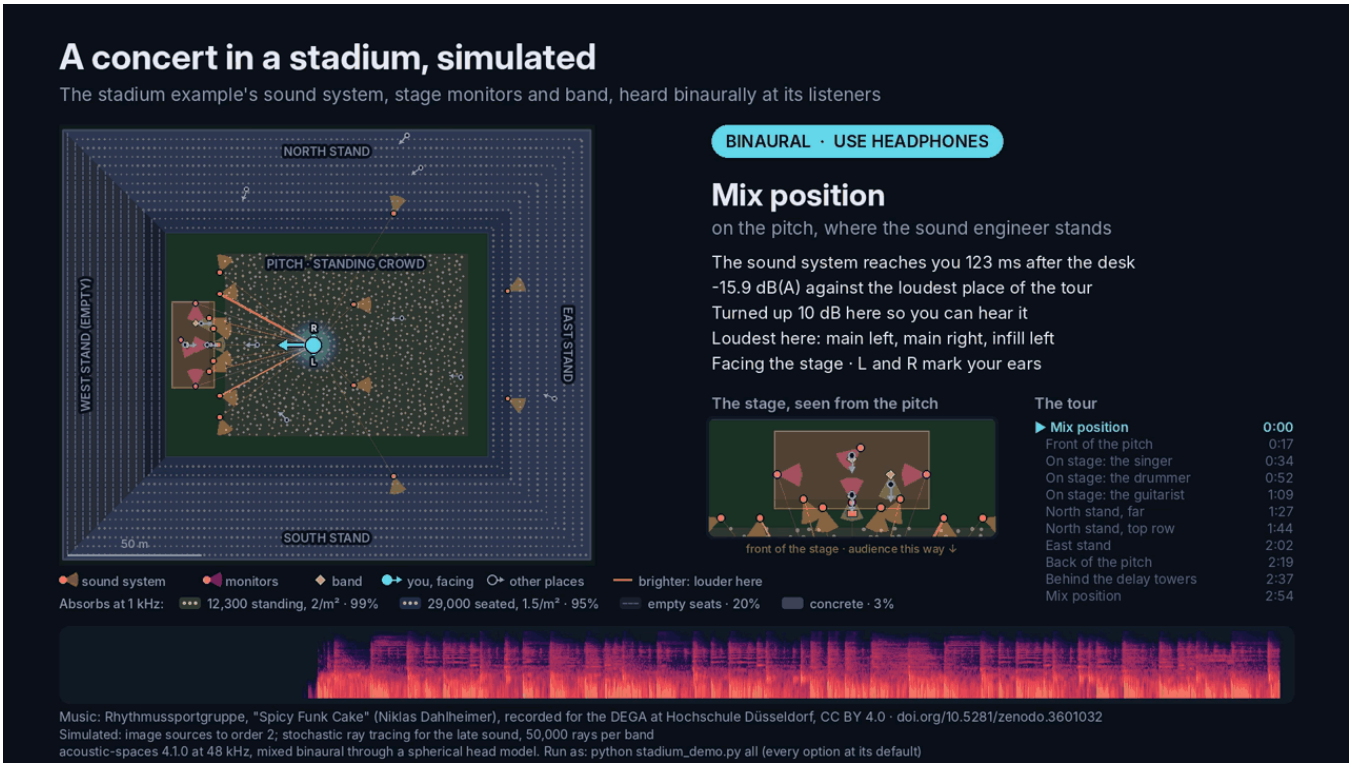
- The singers: Wilkins, Seetharaman, Wahl and Pardo, **VocalSet: A Singing Voice Dataset** (ISMIR 2018), https://doi.org/10.5281/zenodo.1442513, **CC BY 4.0**. The straight-tone takes of "Dona nobis pacem" of 19 of its 20 professional singers, close to a cardioid microphone in a booth (mono, 44.1 kHz). The countertenor is left out: the annotation of that take belongs to a longer take than the one in the archive.
- Where the notes are: Faghih and Timoney, **Annotated-VocalSet** (Applied Sciences 12(18), 2022), https://doi.org/10.5281/zenodo.7061507, **CC BY 4.0**: the start, end and mean F0 of each sung note. Its ground truth for the fourth note says F4; the score and the singers have G4, which the demo uses. Both archives are read by HTTP range requests, so only the parts holding the files needed (69 MB) are downloaded, not the 2.5 GB.
- The organ: the Versilian Community Sample Library's pipe organ, sampled by Simon Dalzell of Ivy Audio, Versilian Studios, https://github.com/sgossner/VCSL (pinned commit), **CC0 1.0**, credited as its notes encourage. Sampled every minor third; each note comes from the nearest sample, retuned by up to two semitones. The samples hold about a second of the room they were recorded in (30 dB down after 0.7 to 0.9 s); the release fade removes most of it.
- The music: "Dona nobis pacem" is a traditional round (public domain). The organ's harmonisation (`ORGAN` in `church_demo.py`) was written for the demo; use it freely.
- Changes, as CC BY asks to say: each take is retuned towards the tune by resampling, time-warped so that its notes start on the beat (WSOLA, which keeps the pitch), cut into two verses, brought to one level and mixed through the simulated church. The organ samples are evened out in level along the keyboard and cut at each key's release with a 90 ms fade.

Whoever shares a render (the video on a video site, say) has to credit VocalSet, link the licence and say that the takes were changed, for example:

> Singers: VocalSet by J. Wilkins, P. Seetharaman, A. Wahl, B. Pardo (CC BY 4.0, doi:10.5281/zenodo.1442513), with note annotations from Annotated-VocalSet by B. Faghih and J. Timoney (CC BY 4.0, doi:10.5281/zenodo.7061507); retuned, time-aligned and placed in a simulated church. Organ: VCSL pipe organ, sampled by Simon Dalzell (Ivy Audio), Versilian Studios (CC0). License: https://creativecommons.org/licenses/by/4.0/

No multitrack recording of a real choir could be used: Dagstuhl ChoirSet, the Choral Singing Dataset, the ESMUC Choir Dataset and Cantoría are all for non-commercial use only (CC BY-NC 4.0).

A concert in the stadium



`stadium_demo.py` plays a song by a band on the stage of the `stadium_concert` example, through its sound system and stage monitors, and mixes it at each of its 12 listeners. The video plays the song once while the listener moves, on phrase boundaries, from the mix position to the front of the pitch, onto the stage (singer, drummer, guitarist), round the stands to the far end and back. It plays one of three songs (`--song`): “Spicy Funk Cake”, funk by a nine-piece band (3:15, ten of the listeners, as the song has eleven phrases), or “Francium” (4:54, all twelve) or “Rubidium” (3:52), electronic songs by Software-Entwicklungskit released with their stems.

```
acoustic-spaces examples --copy . stadium_concert && cd stadium_concert
python stadium_demo.py music     # the reference mix and 7 stems (WAV, 313 MB to download)
python stadium_demo.py band      # scene_band.yaml: the scene with the band on the stage
acoustic-spaces simulate scene_band.yaml --out results_band --output binaural   # ~2.5 min on
4 cores
python stadium_demo.py feeds     # demo/feeds: one WAV per loudspeaker and player
python stadium_demo.py render    # demo/seats: one binaural WAV per listener (~30 s each)
python stadium_demo.py video     # demo/tour.wav, demo/stills, demo/tour.mp4,
demo/thumbnail.png
```

Options: `--song francium` or `rubidium` for `music` and `feeds` (with `--work demo_francium` for a folder of its own), `--tour mix_position:1,singer:0.5,east_stand:1` (listener:phrases of 8 bars, 0.5 for 4; the last one runs to the end), `--listeners mix_position,east_stand` (those listeners alone, in `render` and `video`), `--format` for another output the simulation wrote (the tour decodes it to two ears), `--no-mp4` for the audio and stills only. `all` runs every step in turn, leaving out what is there already (the music, the scene, the simulation).

A song of your own plays in place of the presets: `--music song.wav --bpm 120` for `music` and `feeds` (and `all`), the master as a stereo file (WAV, or anything ffmpeg decodes) and its tempo, which the tour's phrases of 8 bars go by. `--stem vocal=vox.wav --stem drums=drums.wav`, once per stem (drums, bass, guitar, keys, brass, vocal, backing), gives the stems the monitors' mixes and the band's players draw on; a mix none of whose stems is given plays nothing, so without stems the monitors and the band are silent, which `feeds` and the Run demo window's table say. `--downbeat 0.5` is the time of the first downbeat, so that the cuts stay on the beat; `--credit "Music: …"` the line the video shows, the file's name unless given. Nothing is downloaded; the music goes to `demo/music/<file name>`, a folder of its own beside the presets', and `--input` goes on working on top of it.

- **The band.** The players are the example's sources on the stage, in its Band group, which the concert's own simulations leave out until it is ticked; `band` takes them in: a drum kit beside the drummer (116 dB sound power, omni) playing the drums, the singer's voice (98 dB, subcardioid, facing the audience) playing the vocal, and an amplifier behind the guitarist (112 dB, cardioid, tilted up) playing the guitar, each from its own stem. On stage and near it the players are heard from where they stand; further out the sound system covers them.
- **Who plays what.** The sound system plays the song's stereo master: the arrays, fills, towers and delays on the audience's left (-y) take the left channel, those on its right (+y; the scene's `*_left`, named from the stage) the right one, the subwoofers both. The monitors play mixes of the stems: the singer's wedges the vocal over the backing vocals, the keys and a little of the guitar, bass and drums; the drum fill the drums and bass with the vocal, guitar and keys under them; the side fills a stage mix with the vocal 3 dB up and the drums 20 dB down (the kit is loud on stage by itself). A stem a song does not have (brass, hand percussion) is left out of the mixes that name it.
- **How loud.** `feeds` brings every signal to the same gated RMS and then scales it by its `level_db` against the loudest's: the main arrays 0 dB, subwoofers and towers -2, delays -4 and -6, out-fills -4, in-fills -10, side fills -12, front fills -18, wedges and drum fill -19. So the feed files are what each loudspeaker plays, and `render` runs `auralize` with `--equal-power`, so that the levels are applied once.
- **One gain for all listeners, then each stop turned up.** `render` scales the 12 mixes by one gain (the loudest peak at -1 dBFS), so the level differences between places in `demo/seats` are the simulation's: on stage, between the monitors and beside the drum kit, it is 6 to 18 dB(A) louder than in the audience, and the mix position is 16 dB(A) below the singer's place. That is too much to hear both in one video, so the tour turns each stop up to the loudness of the loudest one and its caption gives the real level and how much it was turned up. Loudness is weighted as loudness meters weigh it (K, ITU-R BS.1770), which counts the subwoofers' bass that the A weighting of the levels all but leaves out: on the pitch the stops are turned up less than their dB(A) alone would say.
- **The cuts.** Every listener's mix is shifted earlier by the time the loudest sound takes to reach them (the peak of the energy of their responses, each weighted by how loud its source plays: 0.12 s at the mix position, 0.38 s at the far end), so the beat runs on across each 1 s crossfade; the caption gives that time. The stops last 8 bars from the song's first downbeat, each crossfade ending on a downbeat. "Francium" starts a beat into a bar and its intro is a bar short, so its phrases run from 14.8 s and the first stop takes the intro as well; it has nineteen phrases and a half, so five stops, each at a section's start, last two.

- **The pictures.** The stadium from above, with the loudspeakers as dots and cones the way they point, lines from each to the listener of the moment (brighter where louder), and the crowd where the scene has its audience: a dot for every ten or so people standing on the pitch (two on a square metre) and sitting in the stands (one and a half), and the empty stand behind the stage as rows of seats. A line under the plan gives how many stand and sit and how much of the sound at 1 kHz the crowd, the empty seats and the concrete take up, as the scene's materials have it.

Limits. The band is three point sources at typical sound powers, not measured instruments; the bass, keys and brass on stage come only through the monitors and the sound system, and there is no crowd noise. The reference mix and the stems are a finished studio production, reverberation and all, so a little of the studio plays in the stadium too. Binaural is through the spherical head model unless `--hrtf` names a SOFA file. Some listeners stand nearer the arrays than their far-field patterns hold, which the scene's arrays handle by adding up their boxes one by one (`near_field.py` in the example puts numbers on it).

Recordings and licences. "Spicy Funk Cake" by the Rhythmussportgruppe, a nine-piece funk band (guitar, drums, bass, keys, saxophone, two trumpets, trombone and voice; composed by Niklas Dahlheimer, sung by Juliane Blum; 110 BPM, 3:15), recorded in 2017 for the sound database of the DEGA (German Acoustical Society) at Hochschule Düsseldorf's studio, https://doi.org/10.5281/zenodo.3601032, **CC BY 4.0**. `music` reads the reference mix and the seven processed stems (drums, bass, guitar, keys, brass, vocal, shouts; stereo WAV, 24-bit, 48 kHz) out of the release's 3.5 GB archive by HTTP range requests, so only those are downloaded: 313 MB (450 MB unpacked). The stems start 58 ms before the reference mix; `feeds` lines them up with it by cross-correlation.

"Francium" (127.75 BPM, 4:54) and "Rubidium" (132 BPM, 3:52) by Software-Entwicklungskit, from the album *Everything is Free*, https://github.com/ichbinsoftware/everythingisfree, **CC0 1.0**: electronic songs released with their stems. `music --song francium` downloads the master and the 26 stems (M4A, 42 MB in all; "Rubidium" has 9, 43 MB) from the album's repository at a fixed commit, decodes them with ffmpeg and sums the stems into the groups the mixes use: drums, bass, guitar (the synth lead, which the guitarist's amplifier plays), keys, vocal and backing vocals.

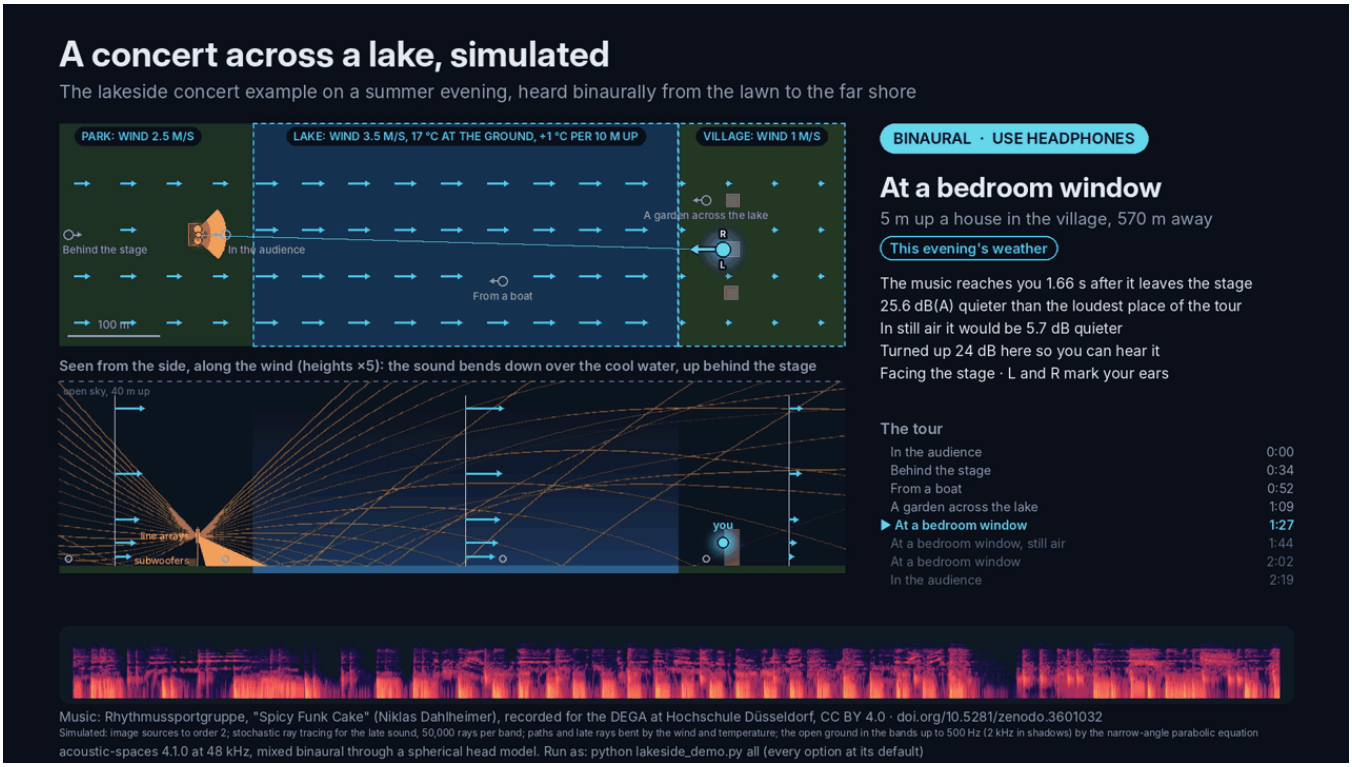
Either way the master plays through the sound system, the stems through the monitors and the band, and the credit goes to `demo/music/<song>/SOURCES.txt`.

Whoever shares a render has to credit the musicians (and the DEGA), link the licence and say what was changed, for example:

> Music: "Spicy Funk Cake" by Rhythmussportgruppe (composed by Niklas Dahlheimer), recorded for the DEGA at Hochschule Düsseldorf, CC BY 4.0, doi:10.5281/zenodo.3601032; the mix and its stems played through a simulated stadium sound system and stage and rendered binaurally. License: https://creativecommons.org/licenses/by/4.0/

Other band multitracks were left out: Cambridge-MT's are for education only, MedleyDB's and MUSDB18's non-commercial or restricted, and the OnAir Music dataset's (CC BY-SA 4.0) share-alike clause would bind the renders too.

A concert across a lake



`lakeside_demo.py` plays a song from the stage of the `lakeside_concert` example on a summer evening and mixes it in the audience on the lawn, 140 m behind the stage, on a boat in the middle of the lake and in the village across it, 550 m away, in a garden and at a bedroom window. Every place is mixed twice: in the evening's weather and in still air. The video tours them, and at the bedroom window plays the song in the evening's weather, then in still air, then in the evening's again, so the difference the weather makes is heard.

```
acoustic-spaces examples --copy . lakeside_concert && cd lakeside_concert
python lakeside_demo.py music    # the song (WAV, 56 MB), a signal for each loudspeaker
python lakeside_demo.py scenes   # scene_demo.yaml, and scene_still.yaml in still air
acoustic-spaces simulate scene_demo.yaml --out results_demo --output binaural    # ~20 s
acoustic-spaces simulate scene_still.yaml --out results_still --output binaural  # ~13 s
python lakeside_demo.py render   # demo/seats: every listener in both weathers
python lakeside_demo.py video    # demo/tour.wav, demo/stills, demo/tour.mp4,
demo/thumbnail.png
```

Options: `--song great_british_summer` or `potassium` for `music` (below), `--music song.wav --bpm 120` for a song of your own in their place (a stereo file, WAV or anything ffmpeg decodes, and its tempo, which the tour's phrases of 8 bars go by; nothing is downloaded, and the video credits the file by its name unless `--credit` says what to show), `--tour bedroom_window:still:1,audience::2` (listener:weather:phrases of 8 bars), `--listeners audience,bedroom_window` (those places alone, in both weathers), `--last-phrases` (how long the last stop lasts, 0 to the end of the song; 2 by default), `--format` for another output the simulation wrote (the tour decodes it to two ears), `--no-mp4` for the audio and stills only. `all` runs every step in turn, leaving out what is there already (the music, the scenes, the simulations).

- **The sound system.** `scenes` plays, in place of the example's single source (120 dB sound power), the example's PA group, left out of its own simulations until ticked: a system as a lawn concert hangs it: two line arrays of eight boxes (`lake_array.yaml`, a generic box's pattern in `box.csv`, both made with the stadium example's `make_model.py`), 6.5 m up either side of the stage, 124 dB each, aimed down the lawn and, box by box, from 5° down at the last rows, where the lawn meets the water, to 36° at the front rows, each added up box by box (`near_field`), playing the song's left and right channels; and a block of four cardioid subwoofers (`sub_array.yaml`, 130 dB) on the ground at the stage front, 2 m nearer the audience and delayed 6 ms so that both arrive together there, playing both channels, rolled off above 125 Hz. The system is tuned at the audience as a system engineer tunes it (`eq_db`: the arrays' low mids up, their top octave up for the air): at 30 m, in the evening's weather, every octave from 500 Hz to 4 kHz is within 1 dB of 1 kHz, 250 Hz 1.3 dB down, 8 kHz 1.9 dB down and 63 and 125 Hz 1.8 to 1.9 dB up. `scenes` also takes in the example's listener 140 m behind the stage, upwind (`behind_stage`, left out as the system is), and lets the responses run 3.5 s past the latest direct sound, the far shore's, 1.6 s late (5.2 s in all).
- **The weather** is the example's: a breeze from the park out over the lake, 2.5 m/s 10 m up, and two weather zones, the lake's (air 17 °C at the water, a degree warmer every 10 m up, and a 3.5 m/s breeze) and the village's (a 1 m/s breeze among the houses). The breeze and the cool air bend the sound down to the water, which gives it back almost whole. `scene_still.yaml` is the same scene with no atmosphere.
- **What it gives.** At the bedroom window the evening's weather makes the song 5.3 dB(A) louder than still air would, in the garden 5.0 dB; behind the stage, upwind, it is 0.7 dB quieter, and on the boat 0.9 dB quieter (the example's `simulation.seed`, 1; with rays of their own, two other runs gave up to 6.0 dB at the window and 0.5 dB behind the stage). These figures are from before the 1 and 2 kHz bands over the open ground came from the parabolic equation, and from before the lawn and the water scattered nothing, and have not been run again since. For the line arrays the first change lowers those two bands in the evening's weather by up to 2.7 dB at the window and 3.5 dB in the garden and raises them by up to 1.6 dB on the boat, and leaves the other bands, the place behind the stage and still air as they were; the second moves what the weather adds by 0.3 dB(A) at most at these four places (every source at its sound power, one run each). The far shore gains most: what the weather bends down comes in off the water as well as straight, and at the window the house's wall gives back each of those again.

- **How loud.** From the audience to the far shore the sound drops by more than 20 dB(A), too much to hear the far places in a video, so each stop is turned up to the audience's loudness and its caption says how much quieter the place really is, in dB(A). The loudness is K-weighted, as meters weigh it, not A-weighted: the far places and the back of the stage keep little but bass (the array boxes radiate every way below 250 Hz and the water gives the low end back whole), which the A weighting all but leaves out, so matching dB(A) would put their bass far above the audience's. The window in still air keeps the gain of the window in the evening's weather, so the difference the weather makes is heard.
- **The cuts.** Each place's mix is shifted earlier by the time the loudest sound takes to reach it (the peak of the energy of its responses: 0.09 s in the audience, 1.66 s at the window), so that the beat runs on across each 1 s crossfade, every 8 bars.
- **The pictures** draw the lake from above with the wind's arrows, the weather zones and their weather, the arrays and the subwoofers with cones the way they aim, and the current listener with L and R at its ears; and, under it, a slice along the line from the stage across the lake, heights drawn five times larger, with the arrays hung from the roof and the wedge from their top box's aim to their bottom one's, the subwoofers on the ground, and rays from the arrays bent by the scene's own profiles of the speed of sound (traced by the script for the picture; the simulation traces its own) and masts with the wind's profile; in still air the rays go straight.

Limits. The arrays are a generic box's pattern, not a product's measured one; there is no crowd, no wind noise and no other sound of the evening. The weather holds still: the example has no turbulence, so the sound does not come and go with the gusts as it does across a real lake. The song is a finished studio mix, played as it is. Binaural is through the spherical head model unless `--hrtf` names a SOFA file.

Recordings and licences. `music` writes the credit to `demo/music/song.json`, and the video shows it.

- `spicy_funk_cake` (the default): "Spicy Funk Cake" by the Rhythmussportgruppe, the recording of the stadium demo, **CC BY 4.0**; its reference mix (a 56 MB WAV file, 61 MB to download) is read out of the release's preview archive by HTTP range requests.
- `great_british_summer`: "Great British Summer" by Tom McKenzie, from the album *Extraordinary* in AlbumDB, https://doi.org/10.5281/zenodo.19683001, **CC BY 4.0**, read out of its archive of masters the same way.
- `potassium`: "Potassium" by Software-Entwicklungskit, from the album *Everything is Free*, https://github.com/ichbinsoftware/everythingisfree, **CC0 1.0**, decoded from M4A with ffmpeg.

A shared render of either CC BY song has to credit it, link the licence and say what was changed, for example:

> Music: "Spicy Funk Cake" by Rhythmussportgruppe (composed by Niklas Dahlheimer), recorded for the DEGA at Hochschule Düsseldorf, CC BY 4.0, doi:10.5281/zenodo.3601032; played through a simulated outdoor sound system across a lake and rendered binaurally. License: https://creativecommons.org/licenses/by/4.0/

A walk through the gallery

`walk_demo.py` is the smallest demo and the only one of speech: a guide stands in the L-shaped gallery of the `gallery` example and talks while a visitor walks away from them, down the long arm and round the corner, where the guide is out of sight and only the sound bent round the corner arrives; then the visitor stands 4 m in front of the guide and turns their head a quarter at a time, so the voice moves to the right ear, behind, to the left ear and back in front. The video follows the visitor from above, with the head and its ears, the guide, and a bar of the levels.

```
acoustic-spaces examples --copy . gallery && cd gallery
python walk_demo.py all   # or speech, scene, walk, turn, video one at a time
```

Options: `--talker M4` picks one of the corpus's 14 talkers, `--voice` how loud the guide speaks (relaxed, normal, raised, loud), `--stand 16,4` where the visitor stands for the turn (by default where the scene's `standing` listener does, left out of the gallery's own simulations), `--turn` the angles it holds, `--hrtf me.sofa` a measured head.

- **The walk** is the example's visitor's: responses every 25 cm along it (`acoustic-spaces walkthrough`), the talk played along them so that the voice changes as the visitor moves. The speech level at the ears falls from 56.0 dB(A) three metres in front of the guide to 48.7 dB(A) round the corner, and the speech transmission index with it, from 0.73 to 0.54: the gallery is quiet and bare, so what is lost is the direct sound, not the clarity. The gallery's two other listeners stand where they are and are left out. Edit the visitor's walk (the Movement panel, or the scene file) and `all` makes the scene and the walk again; with the walk turned off (**Moves**) the demo says there is no walk to play.
- **The turn** goes through the engine's head tracker, scripted rather than played live: the place's response in Ambisonics of order 7, the sound field turned against the head every 256 samples and decoded to the ears (head tracking). It comes within half a decibel, in each ear and octave, of rendering the ears again at every angle.
- **Front and back.** Through the spherical head model the guide behind sounds much as the guide in front: a sphere has no outer ears. This is the demo where a measured HRTF (`--hrtf`, or a set to download) is heard most plainly.
- **The voice** is IEC 60268-16's: the long-term spectrum of male speech at 60 dB(A) 1 m in front for a normal voice, with a voice's directivity, all round in the bass and towards the front in the treble. One gain plays the whole video, so the voice is as much quieter round the corner as the simulation says.

Recording. The ACE challenge corpus (Eaton, Gaubitch, Moore and Naylor, Imperial College London), **CC BY 4.0**: an anechoic talk, a man describing his way to work through Delft; `speech` reads the corpus's 148 MB archive as it comes, up to the talk: 16 MB for that talker, up to 145 MB for another (`--talker`). A talk shorter than the walk and the turn need, as some talkers' are, is said again from its start, 2 s after it ends.

Vehicles passing by

`passby_demo.py` comes with two outdoor examples, `roadside` and `street`: a vehicle drives past and each listener hears one pass in turn, its pitch falling as it goes by. The video follows the vehicle on the plan, with the level heard over the pass drawn under it. `--vehicle` picks what goes past: in the roadside a passenger car (`car`) or an articulated truck (`truck`), in the street the car (with `car_horn`, its horn, when `--car` says), a city bus (`bus`), a motorcycle (`motorcycle`), a police car with its siren on (`police_car`, the car itself, and `police_car_siren`), a fire engine with its siren on and its air horn in blasts (`fire_engine`, the vehicle itself, `fire_engine_siren` and `fire_engine_horn`) or a tram (`tram`, see trains passing by).

```
acoustic-spaces examples --copy . roadside && cd roadside   # or street
python passby_demo.py all --vehicle truck   # or sound, scene, walk, video one at a time
```

Options: `--vehicle car` what goes past (the car by default), `--speed 70` how fast it drives (km/h; without it, the speed of its walk in the example's scene: 70 km/h for the truck, 50 km/h for the others), `--rpm 2000` how fast its engine turns, for its made sound (by default its own: the car's four cylinders at 2000 rpm, the truck's six at 1200, the bus's and the fire engine's six at 1500 and the motorcycle's two at 4000), `--siren wail` the police car's siren (`wail`, `yelp` or `two_tone`), `--police-car car_and_siren` what the police car plays (the car itself and its siren, or `siren` alone), `--fire-engine truck_siren_and_horn` what the fire engine plays (all three, `truck_and_siren`, `siren_and_horn` or `siren` alone), `--car car` what the street's car plays (the car alone, `car_and_horn` with its horn, or `horn` alone), `--step 1` how far apart the vehicle is simulated along its walk (m), `--hrtf me.sofa` a measured head, `--input NAME=WAV` a dry recording of yours for the vehicle NAME (`police_car_siren=WAV` for the police car's siren, `fire_engine_horn=WAV` for the fire engine's horn, `car_horn=WAV` for the car's), `--listeners verge,yard_25m` the listeners it is simulated and heard at (every one in use by default), in that order. The vehicle drives the length of its walk at that speed, so the pass lasts the length divided by the speed: the 150 m of the roadside take 10.8 s at 50 km/h, 7.7 s at 70 km/h. The **Run demo** window gives the three under its options as they change, and ticks the vehicle (the police car's two sources, the fire engine's three, the car's with its horn) in its Movement box.

- **The road vehicles** are the example's sources of their names: point sources 0.05 m above the road, where CNOSSOS-EU puts a vehicle's sound, each with a walk along its lane with Doppler, left out of the example's own simulations until ticked (there the traffic stays a still line source). A walk goes through points at a speed, or through keyframes timed for one steady speed, since CNOSSOS-EU gives a vehicle's sound at a steady speed: keyframes that speed up or slow down by more than 1 % are refused, unless `--speed` drives the vehicle through their places at one speed. The demo's scene (`scene_demo.yaml`) has the vehicle alone along its walk at that speed, with CNOSSOS-EU's sound power of its category at that speed on the method's reference road surface, level (the method's corrections for road gradients, other surfaces and studded tyres are left out), in the scene's air temperature: rolling noise A_R + B_R lg(v/70) + K (20 - t) and propulsion noise A_P + B_P (v - 70)/70 per octave band, with the coefficients of Table F-1 of Directive 2002/49/EC, Annex II, as amended by Commission Delegated Directive (EU) 2021/1226 (`acoustic_spaces.demos.road`). At their speeds and 20 °C:

vehicle	CNOSSOS-EU category	speed	sound power	A-weighted
`car`	1, light motor vehicles	50 km/h	102.0 dB	98.4 dB(A)
`bus`	2, medium heavy vehicles (two axles)	50 km/h	109.4 dB	104.5 dB(A)
`truck`	3, heavy vehicles (three axles or more)	70 km/h	114.1 dB	110.2 dB(A)
`motorcycle`	4b, motorcycles (propulsion noise alone)	50 km/h	104.1 dB	97.5 dB(A)

  The amendment has the vehicle's point radiate alike every way, where Directive (EU) 2015/996 had it radiate into the half space above the road, and its coefficients A are higher: the car's sound power is 2.1 to 2.7 dB(A) higher than by 2015/996's table from 30 to 130 km/h, 2.5 dB(A) at 50 km/h (96.0 dB(A) before acoustic-spaces 4.1.0; `validation/urban_sources.py`). The table is used as published: no measured pass-by is compared with it here. Measured pass-bys of medium heavy vehicles in France found the 2015 table's levels 3 to 6 dB(A) low (Euronoise 2018, paper 210).

- **Their sound** is made to that power. A recording of a vehicle going past would not do: it holds its own Doppler shift, the road and the distance, and the simulation would add them again. Tyre noise is noise with the rolling noise's octave spectrum; the engine is a pulse each time one of its cylinders fires (every 15 ms for the car's four at 2000 rpm, 17 ms for the truck's six at 1200, 13 ms for the bus's six at 1500 and 15 ms for the motorcycle's two at 4000, the cylinders a little unequal and the firing a little uneven) with the propulsion noise's spectrum; the two are at the ratio of their sound powers in each band, so the octave levels are CNOSSOS-EU's and the time structure is made up, as the stills say. The motorcycle has the engine alone, as the method gives two-wheelers no rolling noise. The sound fades in and out over its first and last second, where the vehicle is farthest. `--input car=engine.wav` plays a dry recording instead (on board, or at the wheel), mixed down to mono and at least as long as the pass; the captions still give CNOSSOS-EU's vehicle.
- **The police car** (in the street) is two sources along one walk: the car itself (`police_car`), a passenger car 0.05 m above the road with CNOSSOS-EU's sound power and a sound made as the car's (its engine four cylinders at 2000 rpm, `--rpm`), and its siren (`police_car_siren`), the walk's points at the height of its roof, 1.5 m up, facing the way it drives (its walk's `face_forward`). Each is simulated along the walk alone, and the two are heard together: their sounds added, each at its own sound power, and the energies of their levels band by band. The demo refuses a siren whose walk goes elsewhere in plan or at other times than the car's. `--police-car siren` plays the siren alone. The siren has the level and the directivity of a commercial 100 W electronic siren the National Bureau of Standards measured in an anechoic room (NBS Special Publication 480-28, 1978): 3 m away, its yelp averaged over 20 s, 118 dB(A) on its axis, 112 dB(A) 50 degrees off it and 107 dB(A) at 90 degrees. The scene's `siren.csv` goes through those three, in dB linearly with the angle off the axis, alike round the axis and behind as at 90 degrees: NBS measured neither the vertical nor behind, so both are assumptions (`acoustic_spaces.demos.siren`). Its sound is a square wave at the siren's fundamental, as the siren's Class D amplifier drives its loudspeaker: the wail and the yelp sweep it from 500 to 1500 Hz and back 12 and 180 times a minute (California's rule for sirens asks 10 to 30 and 150 to 250), the two tones are 440 and 586.7 Hz, a fourth apart, 1.5 s each (the Swiss rule for priority vehicles asks base frequencies of 360 to 630 Hz in the ratio 3:4 and a cycle of 2.5 to 3.5 s). The loudspeaker is taken to send every harmonic alike and the same power in every pattern, the yelp's: 130.9 dB in all, 130.6 dB(A) for the wail and the yelp and 128.6 dB(A) for the two tones, which sit lower where the ear weighs less. At 50 km/h the car's sound power is 32 dB(A) under the wail's and 30 under the two tones', but a square wave sends its odd harmonics alone, nothing under its fundamental: in the octaves of 63 to 250 Hz the police car's sound is the car's, the rumble of it going by (the siren's `level_db` gives those octaves 44 to 55 dB under its whole power). Against the rules (`validation/urban_sources.py`): 3.0 m away the wail and the yelp give 118.0 dB(A) on the axis and 112.0 dB(A) 50 degrees off it, above the least levels of California's class B siren at every angle of its table and 1 to 2 dB under those of class A (the rule takes the average of five peaks of the sweep, NBS the energy over 20 s), loudest in the 1 kHz octave band, as the rule asks (1 or 2 kHz), and the wail's level in 125 ms falls 2.2 dB(A) within its cycle, where the rule allows 10. The two tones give 119.4 dB(A) 2 m away on the axis in a free field (the Swiss rule's test bench: 116 to 129 dB(A)) and 108.1 to 110.8 dB(A) 7 m in front of the car over the street's asphalt, 0.5 to 1.5 m up (on the vehicle: 100 to 115 dB(A)).

- **The fire engine** (in the street) is three sources along one walk: the vehicle itself (`fire_engine`), a medium heavy vehicle with two axles (CNOSSOS-EU's category 2, as the bus is) 0.05 m above the road, with a sound made as the bus's (six cylinders at 1500 rpm, `--rpm`); its siren (`fire_engine_siren`) on its cab's roof, 3.0 m up; and its air horn (`fire_engine_horn`), 1.2 m up. The siren and the horn face the way it drives, with the directivity of the siren NBS measured (`siren.csv`), and the three are heard together, each at its own sound power (`--fire-engine` leaves out the horn, the vehicle itself, or both). The siren plays the police car's two tones (440 and 586.7 Hz, 1.5 s each), 6.0 dB louder: as loud as the Swiss rule lets two tones be on a vehicle, 114.9 dB(A) 7 m in front of the siren (taken at the vehicle's front) at the loudest of 0.5 to 1.5 m up, over the street's asphalt (the rule: 100 to 115 dB(A)), 134.6 dB(A) of sound power. The horn is as loud as the rule for vehicles' horns (Council Directive 70/388/EEC, Annex I) lets one be on its test bench: 118 dB(A) 2 m in front of it on its axis, in a free field (the rule: 105 to 118). The rest of the horn is assumed, as no measured air horn of a fire engine was found (`acoustic_spaces.demos.horn`): two trumpets sounding together at 150 and 180 Hz, a minor third apart as a car's pair of horns is (about 405 to 420 Hz and 500 Hz), within the 125 to 180 Hz Wikipedia's article on vehicle horns gives trucks' air horns, without a source; each a sawtooth (all its harmonics, each 1/k of the fundamental); two blasts of 0.6 s, 0.3 s apart, every 3 s, each at its level 0.05 s after it starts (the rule: within 0.2 s); the height of the rule's bench; and the siren's directivity. Its sound power in the scene is its average over its blasts and pauses, 122.7 dB(A), as its made sound has them: during a blast it is 4.5 dB louder (127.1 dB(A)) than that. The captions' levels have the horn as loud as its sound plays around each place: its level there is the average's times how loud its sound is around that place (as **the pass** below says) against its average, faded in and out with it. A recording given for it (`--input fire_engine_horn=WAV`) plays its blasts as loud as the made horn's, whatever its own pauses, and the captions follow its own blasts. Against the rules (`validation/urban_sources.py`): the siren gives 125.4 dB(A) on the Swiss rule's bench (116 to 129) and 112.3 to 114.9 dB(A) on the vehicle; the horn 107.9 to 108.6 dB(A) 7 m in front of the vehicle, 0.5 to 1.5 m up (the rule: 93 at least), and, on its bench, 110.2 dB(A) between 1800 and 3550 Hz (the rule: 105 at least), its loudest component there 6.0 dB over any above 3550 Hz (the rule: over).
- **The car's horn** (in the street, `--car car_and_horn`, or `horn` alone) is a pair of disc horns behind the car's front bumper (`car_horn`), 0.5 m up along the car's walk, facing the way it drives, with the siren's directivity, in two blasts of 0.25 s, 0.15 s apart, every 3 s from the start of the pass, each at its level 0.02 s after it starts (the blasts, the height and the directivity assumed). The horns are Hella's S77, approved to UN Regulation 28: 360 and 430 Hz, 110 +/- 3 dB(A) 2 m in front of them (Hella's sheet for its S70, S77 and S98), within the rule's 105 to 118. A disc horn's diaphragm strikes its magnet's core every cycle and its tone disc rings, its main components near 3 kHz, harmonics of the diaphragm's frequency (US patent 8,689,723): each horn's harmonics are struck alike and shaped by a resonance at 3 kHz with a Q of 2 (assumed), which leaves the two fundamentals 24 and 23 dB under it. Its sound power is 119.1 dB(A) during a blast and 110.9 dB(A) averaged over its blasts and pauses, as the scene has it; the captions follow its blasts as the fire engine's horn's do. Against the rule (`validation/urban_sources.py`): 109.9 dB(A) on its bench, 107.6 dB(A) between 1800 and 3550 Hz (105 at least), its loudest component there 1.9 dB over any above 3550 Hz, and 100.4 to 101.3 dB(A) 7 m in front of the car, 0.5 to 1.5 m up (93 at least). Made as the air horn is, a sawtooth at each fundamental, it would give only 102.8 dB(A) between 1800 and 3550 Hz, under the rule. `--car car`, the default, plays the car alone, as it plays without a horn.

- **The pass**: the vehicle every metre along its walk (`--step`), simulated at each listener in turn (`acoustic-spaces walkthrough`), its sound played through those places with each early path through its own delay as it changes (moving sources and Doppler), the late sound crossfaded from place to place without a shift of its own. Through places 1 m apart a car's sound at 1 kHz follows the exact field's waveform to within -58 dB (-94 dB a quarter of a metre apart; validation). The level at each moment, the loudest (L_Amax) and the sound exposure level (SEL) come from the same simulation, each place's as if the vehicle stood there (a horn's as loud as it sounds around that place: its sound's power averaged over 125 ms, the time constant of a sound level meter's fast time weighting, so that the beats between its notes do not count, then over the time from halfway to the place before to halfway to the next; places farther apart than a blast lasts less those 125 ms read its blasts low, the car's horn at 50 km/h by up to 0.2 dB 2 m apart and 1.3 dB 4 m apart); the stills give them, with how close the vehicle comes and how far the pitch of its direct sound falls from where it starts to where it ends. One gain makes every pass, and each is turned up to the loudness of the loudest, by as much as the still says.
- **The roadside**: 150 m of road, the car in the lane on the meadow's side at 50 km/h, the truck in the lane on the yard's side the other way at 70 km/h, heard at the verge, 23 and 48 m away over the grass and 27 m away over the paved yard (from the car as it passes; the truck passes 4 m farther from the meadow's listeners and nearer the yard's). The car's pass peaks at 76.4 dB(A) at the verge, 5.7 m away (SEL 76.3 dB(A)), 54.0 and 42.7 dB(A) over the grass and 63.4 dB(A) over the yard; the truck's at 84.6 dB(A) at the verge, 9.6 m away (SEL 85.5 dB(A)), 67.4 and 56.1 dB(A) over the grass and 76.9 dB(A) over the yard, 23.0 m away. Over the grass the ground takes away much of the middle and the treble, the more the farther away, while over the yard its reflection adds to the direct sound. At the verge the engine's tone falls 1.4 semitones as the car passes and 1.9 as the truck does, as the geometry says.
- **The street**: 125 m of street, the car, the motorcycle and the fire engine in the lane on the offices' side, the bus and the police car in the lane on the houses' side the other way, the tram down the middle of the street (see trains passing by), all at 50 km/h, heard on the pavement (6.2 m from the car, the motorcycle and the fire engine's siren as they pass, 2.5 m from the bus and 2.0 m from the police car's siren), on a balcony of the office block and in the garden behind the noise barrier. The passes peak at (L_Amax, dB(A)):

	car	bus	motorcycle	police car (wail)	fire engine	tram
pavement	79.4	90.3	78.1	114.8	113.0	95.6
balcony	74.0	80.3	73.4	104.5	111.0	91.1
garden	62.7	71.3	61.5	95.1	97.2	73.7

  The police car's column is its siren's, which the car itself does not change at this precision: the car goes where the bus does, with a sound power at least 4.6 dB under the bus's in every octave band, so its passes peak at least 28 dB(A) under the siren's at every listener and add 0.01 dB at most. It is heard under the siren's pitch, which dB(A) weighs little. The fire engine's column is its three sources together, its siren much the loudest: in the garden the siren alone peaks at 97.2 dB(A), the horn's blasts at 89.3 and the truck at 68.4, and the fire engine peaks there as its siren does, between two of the horn's blasts; on the pavement a blast comes with the siren's peak and adds 0.7 dB to it (113.0 dB(A), the siren alone 112.3). The balcony hears the fire engine nearly as loud as the pavement does, as its siren, 3.0 m up in the lane on the offices' side, passes 7.6 m from the balcony and 6.2 m from the pavement. With its horn (`--car car_and_horn`) the car peaks at 97.0 dB(A) on the pavement, 93.9 on the balcony and 80.5 in the garden, its horn's loudest blast 17.5 to 19.7 dB over the car's own loudest.

The garden hears the car loudest where the car sees it straight through the 4 m gap between the houses and the barrier (14 m short of the gap), less while the barrier stands between them, and more again once the car comes out past the barrier's far end. The police car, coming the other way, the garden hears loudest 16 m before the barrier's far end, before the barrier stands between them, the garden some 20 degrees off the siren's axis; the pavement hears it loudest 2 m before it passes, 45 degrees off the axis, rather than as it passes, 90 degrees off. Where a reflection through the gap comes or goes, it fades in or out over a few metres, as in a real field: every path, also the copy of a reflection off the road (which the car, so close to it, makes nearly as strong and in step with it) and the second-order reflections, fades at the edges that cut it off. From one metre of road to the next the garden's early sound from the car changes by at most 3.1 dB(A), smoothly where paths go in and out of step; 1 cm apart, the steps left stand out by 0.55 dB(A) at most (see Validation). The late sound comes from rays drawn at random, few of which reach the garden, so two runs of the car's pass put its level up to 2.1 dB(A) apart at one place (its L_Amax within 0.01 dB, the balcony's 0.17 dB apart) unless the scene has a `seed` (see Scenes); the levels above are one run's of each vehicle. Reflections off the road, the facades and the barrier and the sound diffracted round their edges each shift in pitch their own way. Between the buildings the late sound, which keeps the engine's pitch, is strong in the bass, so the engine's tone falls less plainly than by the open road.

Trains passing by

`passby_demo.py` comes with the `railway` example too, and runs a train past its listeners; in the street it runs a tram down the middle of the street (`--vehicle tram`). Each listener hears one pass in turn, its pitch falling as it goes by, and the video follows the train on the plan, with the level heard over the pass drawn under it.

```
acoustic-spaces examples --copy . railway && cd railway
python passby_demo.py all --train freight   # or sound, scene, walk, video one at a time
```

Options: `--train regional` the train (`regional`, an electric locomotive and five coaches, 151 m long, at 160 km/h; `s_bahn`, a four-car electric multiple unit, 67 m, at 120 km/h; `intercity`, the locomotive and twelve coaches, 336 m, at 200 km/h; `freight`, the locomotive and 24 wagons with composite brake blocks, 5 of them tank wagons, 446 m, at 100 km/h), `--speed` how fast it runs (km/h; by default the train's own above, whatever the speed of the example's walk: the Run demo window's **Speed** reads *the train's own* at zero), `--track ballast` the track it runs on (`ballast`, sleepers in ballast; `slab`, slab track; `slab_absorber`, slab track with absorbers; `level_crossing`), `--horn off` the train's warning horn (`blasts` or `chord` sounds it, see below), `--indoors off` where it is heard (see below), `--step 1` how far apart the train is simulated along its walk (m), `--hrtf me.sofa` a measured head, `--input train=WAV` a dry recording of yours (`train_roof=WAV` and `train_pantograph=WAV` for the sources up on the train, `train_horn=WAV` for its horn). The tram has no options of its own but `--indoors`: a low-floor tram with air conditioning, 30 m long, on its track set in the street, at the speed of its walk (50 km/h) unless `--speed` says.

- **Their sound power** is that of Schall 03 (Germany's method for the noise of railways and tramways, Anlage 2 of the 16. BImSchV, 2014; `acoustic_spaces.demos.rail`): for each vehicle unit, from its data sheet (Beiblatt 1 for trains, 2 for trams), each of its sources' A-weighted sound power per metre of track for one unit an hour, per octave band, at its speed and on its track (equation 1), with the speed factors of Tabelle 6 and 14 and the track's corrections of Tabelle 7 and 15. A unit going by at v km/h once an hour has the sound power of that level plus 10 lg(1000 v) dB. The sources sit at the rail top (rolling noise, the bogies' aerodynamic noise, the fans under the floor, the motors), 4 m up (on the roof its aerodynamic noise and its fans, and a tank wagon's body, which its rolling noise makes ring), and at the pantograph, 5 m up, each radiating louder across the track than along it, by 10 lg(0.22 + 1.27 sin² δ) dB with δ the angle to the track (equation 8; the example's `train.csv`, the street's `tram.csv`). Schall 03 takes a fifth of a freight train's wagons as tank wagons unless its make-up is known (4.2), so 5 of the freight train's 24 are, every fifth from the third: 4 m up it has 116.4 dB(A) of sound power, its locomotive's roof alone 111.8. At their speeds on ballast (the tram on its street track):

train	units	length	speed	sound power
`regional`	locomotive (Fz 7) + 5 coaches (Fz 9)	151 m	160 km/h	130.6 dB(A)
`s_bahn`	electric multiple unit (Fz 5)	67 m	120 km/h	121.7 dB(A)
`intercity`	locomotive + 12 coaches	336 m	200 km/h	136.8 dB(A)
`freight`	locomotive + 24 wagons (Fz 10), 5 of them tank wagons	446 m	100 km/h	131.9 dB(A)
`tram`	low-floor tram (Fz 21)	30 m	50 km/h	115.0 dB(A)

  The tram's sound power is 16.6 dB(A) over the car's at 50 km/h (CNOSSOS-EU's, 98.4 dB(A)): Schall 03's tram, with air conditioning on its roof (+8 dB on that source), on a track set in the street, whose rolling noise Tabelle 15 makes 1 to 8 dB louder per octave band than on ballast.

- **A train as long as it is.** Scene sources walk as points, so the example has three: `train` at the rail top, `train_roof` and `train_pantograph`, each with a walk along the track (the street's `tram` and `tram_roof`). The demo shares each unit's sources at a height out between points along it: at the rail top its bogies, from 15 to 85 % of its length (a coach's two bogies 18.5 m apart, as a UIC coach's pivots are), on the roof one over each car, and the pantograph in the locomotive's middle. Each source plays the train's front point at its height along its walk, and every other point there along the same walk as far behind as it is: the same places, that much later, so the regional train passes for 151 m. Each point has its own sound (noise with the octave spectrum of its sources there) at its own sound power, each heard through the same simulated places with its own Doppler, and the levels add each point's energy that much later. The units' lengths and the points' places are assumptions; Schall 03 gives only each unit's sound power per metre of track. Its sound's time structure (steady noise) is made up too; its spectrum per octave band is Schall 03's.
- **The horn** (`--horn blasts`, the example's `train_horn`, at the train's front 2.0 m above the rail top, along its walk): its high note (660 Hz) and its low note (370 Hz) in turn, the notes of the EU's rule for locomotives and passenger trains (TSI LOC&PAS, point 4.2.7.2.1), a second each, 0.2 s apart, every 4 s, each as loud as EN 15153-2 lets it be, 109 dB(C) 25 m ahead of the train over an open, flat site (`acoustic_spaces.demos.horn.TRAIN_HORN`). Each note is an air horn's sawtooth, rich in harmonics, its sound power taken from that level with the site's ground reflection taken out, harmonic by harmonic: 140.2 dB(A) while the high note sounds and 139.2 while the low one does, 136.4 dB(A) averaged over its blasts and pauses, 5.8 dB(A) over the regional train's. `--horn chord` sounds the two notes together in the same blasts, which EN 15153-2 allows, the two 109 dB(C) together (its level for a group of horns sounded at once; `TRAIN_CHORD`): 139.7 dB(A) while it sounds, 136.4 averaged. Its blasts, its height on the train, the measuring site's ground and microphone height, and its directivity (the measured siren's) are assumptions; see Validation.

- **Indoors** (`--indoors`; `acoustic_spaces.demos.indoors`): a listener 0.5 to 2 m in front of a building's wall, where ISO 1996-2 measures in front of a façade, hears the pass in the room behind it, and the others hear it outdoors. The railway's `house_18m` stands 2 m in front of a house 20 m from the track; the street's pavement and balcony stand in front of its houses and its offices. `train_and_rumble` plays the train through the room's closed window and the rumble the ground's vibration makes in it, `train` the train alone, `rumble` the rumble alone (`tram_and_rumble` and `tram` for the tram). The room is EN ISO 12354-3's, not the building's geometry: AGC's 4-16-4 double glazing as measured by EN ISO 10140, Rw (C; Ctr) = 30 (-1; -4) dB, 1.82 m² of it into a room of 50 m³. Each octave band in the room is the simulated level 2 m in front of the wall, the façade's reflection in it (EN ISO 12354-3's L_1,2m), less D_2m,nT = R + 10 lg(V / (6 T0 S)), the window's sound reduction index plus 9.6 dB (`acoustic_spaces.facade`); for traffic's spectrum that is Rw + Ctr + 9.6 = 29 - 4 + 9.6 = 34.6 dB, with ISO 717-1's rating of AGC's values in octave bands, 29 (-1; -4) dB (AGC's 30 is rated from third octaves); the S-Bahn's, higher in frequency, loses 39.6 dB(A) at the railway's house (87.2 dB(A) outdoors, 47.6 in the room). That level is the listener's own where it stands 2 m out, as `house_18m` and the pavement do; nearer, where the wall's reflection makes the 63 to 250 Hz bands louder or quieter, the vehicle is simulated once more, as one more listener (`balcony_2m`, the street example's, ticked off), at the point 2 m in front of the wall, square to it, at the listener's height: for the balcony, 1 m out from it. The room hears the listener's sound through the window, each octave band changed at each moment by as much as the level 2 m out is over the listener's. The rumble is the FTA's general method (see Vibration through the ground) at each moment of the pass: the train's curve at the distance from its nearest wheels to the room's wall behind the listener, the floor 3 dB over the ground, its rooms' rumble 50 dB under that in dB(A), played as noise in the third octave at 25 Hz, the same in both ears: in the railway's house 27.5 dB(A) for the S-Bahn and 41.3 for the regional train. The room's own reverberation is not added, and the sound keeps the direction it comes from at the listener.
- **What is felt**: each listener's caption gives how hard the floor of its house, or outdoors the ground under it, shakes at most during the pass, in VdB (the FTA's curves, which are the ground's), and whether that is felt, from about 65 VdB (the manual's threshold of perception). Outdoors nothing rumbles: the manual counts ground-borne noise only inside buildings. Each listener's vibration is also a track for a bass shaker, `walk/<listener>.shaker.wav`, and the tour's `tour.shaker.wav`: noise in the third octave at 25 Hz whose RMS is the vibration's velocity, every listener at one gain, its full scale in `walk.json` (`shaker`).
- **Against Schall 03's own sum** (`validation/rail_sources.py`): in a free field, the engine's pass of the S-Bahn at 80 km/h and of the regional train at 160 km/h, 7.5 m from the track 1.2 m above the rail top and 25 m away 3.5 m above it, gives the same SEL as Schall 03 summing the same stretch of track in pieces of 0.1 m, within 0.01 dB in every octave band. Over the railway example's ground the engine gives 3.0 to 4.1 dB(A) less than Schall 03 with its own terms for the ground, which reckon the ground's reflection otherwise (see Validation, which also sets Schall 03's units beside the EU's limits for new trains).
- **The railway**: 300 m of track, each train at its own speed on ballast, heard 7.5 m from the track 1.2 m above the rail top (`tsi_7m5`), 25 m away over the meadow (`meadow_25m`), 25 m away on the other side, behind the barrier (`behind_barrier_25m`), and 2 m in front of a house 20 m from the track (`house_18m`, 2 m in front of the house's wall, its reflection included). The passes peak at (L_Amax, with the SEL after it, dB(A)):

	regional, 160 km/h	S-Bahn, 120 km/h	intercity, 200 km/h	freight, 100 km/h
7.5 m	94.4 (98.4)	88.6 (90.7)	97.3 (103.6)	90.3 (101.6)
meadow, 25 m	85.3 (90.6)	79.6 (82.8)	88.5 (96.0)	81.8 (93.6)
behind the barrier, 25 m	78.2 (79.2)	66.2 (69.8)	82.3 (83.3)	74.8 (82.6)
house, 18 m	92.5 (97.8)	87.2 (90.1)	95.4 (103.0)	89.1 (100.9)

The scene fixes no seed, so each run draws its own late rays: over five runs the peaks by the house vary by up to 0.2 dB and their SELs by 0.04 dB, over four runs those on the meadow by 0.05 dB, and at 7.5 m and behind the barrier two runs agree within 0.01 dB.

The longer a train, the more its SEL is over its L_Amax: the freight train, 446 m long, takes 16 s to pass at 100 km/h, the S-Bahn, 67 m, 2 s at 120 km/h. The barrier, 3 m high and 4 m from the track, hides the rail top from the listener behind it, but not the roof (4 m up) or the pantograph (5 m up): behind it the regional train peaks 7.1 dB(A) under its level on the meadow's side, the S-Bahn, whose rail top is 20 dB over its roof in sound power (the regional train's 16 dB), 13.3 dB(A) under.

- **The tram in the street**: the tram runs down the middle of the street at 50 km/h, 4.0 m from the pavement's listener, and peaks at 95.6 dB(A) on the pavement (SEL 99.3), 91.1 on the balcony (95.4) and 73.7 in the garden (80.3): louder than the bus, which passes 2.5 m from the pavement's listener at 90.3 dB(A), as Schall 03's tram, on its track set in the street, has 10.5 dB(A) more sound power than CNOSSOS-EU's bus at 50 km/h.

A helicopter flying over

`flyover_demo.py` comes with the outdoor example `helicopter_flyover`: an Airbus Helicopters AS350 B3 flies over a meadow 150 m up at 63 m/s, against a light wind, and each listener hears one pass in turn: right under its track, 1.2 m above the grass, where the microphone of its noise certification stands, and 150 m to its left and to its right. It is the road vehicles' demo with the helicopter in the car's place (`acoustic_spaces.demos.helicopter`): the same steps, options and video, but for the car's sound.

```
acoustic-spaces examples --copy . helicopter_flyover && cd helicopter_flyover
python flyover_demo.py all   # or helicopter, scene, walk, video one at a time
```

Options: `--speed 200` how fast it flies over the ground (km/h; without it, the speed of its walk, 226.8 km/h), `--broadband bpm` its rotors' broadband noise from each blade section's own noise instead of Pegg's method (Helicopters), `--step 4` how far apart it is simulated along its walk (m), `--hrtf me.sofa` a measured head, `--input helicopter=WAV` a dry recording of yours, `--listeners under_track` the listeners it is simulated and heard at. Its 900 m of flight take 14.3 s at 63 m/s.

- **The helicopter** is the example's source `helicopter`, a point at the main rotor's hub, with a walk 150 m up with Doppler, facing the way it flies. Its rotors are trimmed for level flight (Helicopters), so a walk that climbs or descends by more than 0.5 m is refused. They are trimmed at its airspeed: its speed over the ground less the wind of the scene's own weather at its height, as vectors, in the scene's air. In the example the wind blows against it at 4.5 m/s 150 m up (3 m/s at 10 m), so it flies through the air at 67.5 m/s. It faces along its walk, so it does not turn into a wind from the side, as a helicopter flying crabwise would: a wind along its track, or none, suits it best. The demo's scene (`scene_demo.yaml`) has the helicopter alone with the sound power per octave band and the directivity of its rotors' sound at that airspeed, the directivity in a table beside it (`scene_demo_helicopter.csv`): 137.7 dB, 132.9 dB(A) in the example, which its own scene has too.
- **Its sound** is what the rotors send each way: each harmonic of each rotor's blade passing frequency with a pattern and a phase of its own, and broadband noise. Each early path plays what is sent the way it leaves the helicopter, at the time it leaves (the sound sent each way); a late sound, which the meadow alone does not give, plays it averaged over the directions. It fades in and out over its first and last second. `demo/helicopter/below.wav` is what they send straight down, dry, to listen to. `--input helicopter=WAV` plays a dry recording instead, mixed down to mono, at least as long as the pass, sent every way with the rotors' directivity; the captions still give the rotors' model.
- **The pass**: the helicopter every 4 m along its walk, simulated at each listener in turn, as the car's. From 150 m up, places 4 m apart follow the exact field of a flyover to within 55 dB at 4 kHz, more closely than the car's places 1 m apart (validation). Over the grass the bands up to 500 Hz come from the wide-angle parabolic equation where the sound comes in at 34° from the horizontal or flatter (`parabolic_equation: wide_angle`, as the narrow-angle one holds within some 15°), and the rays give the rest: the example sets `open_ground_frequencies: rays`, as its sound comes down steeply and the rays' 1 and 2 kHz bands are within 0.11 dB of the equation's along its track (how the simulation bends sound). The wind bends the sound the helicopter sends ahead up and the sound it leaves behind down, and the eddies (`moderate` at 2 m, weaker higher up) make it flicker in the bands the rays give.

What it leaves out: the rotors' model is louder than the helicopter's noise certificate. Flown over the microphone as the certification does, it gives a sound exposure level of 88.0 dB(A) where the AS350 B3 is certified at 84.6 dB(A), nearly all of it the main rotor's broadband noise by Pegg's method; with `--broadband bpm` it gives 80.1 dB(A), its tail rotor's tones then nearly all of it (see Validation). The helicopter keeps one trim along its walk, as in steady level flight: the slap of the blades meeting the vortices of those ahead, as a helicopter makes descending or turning, and the loads that change round each turn are left out. The eddies flicker at the pace of their drift in the wind, not that of the helicopter sweeping the paths through them, and they are checked for sources up to 10 m up, not 150 m.

A helicopter over the stadium concert

`helicopter_demo.py`, the stadium example's second demo, flies the flyover's helicopter over the stadium concert halfway through the song, and adds its pass to each listener's mix of the concert, both at their simulated levels. The helicopter is in the stadium's scene, left out of the concert's own simulations: the editor shows its walk, the Run demo window lists it as what the demo moves, and what Edit movement… changes there the demo flies. It builds on the stadium demo: run that one first, in the same work folder, up to its `render` step, with the `--format` and the `--hrtf` this one gets.

```
acoustic-spaces examples --copy . stadium_concert && cd stadium_concert
python stadium_demo.py all        # the concert first (see above)
python helicopter_demo.py all     # or flight, helicopter, scene, walk, mix, video one at a
time
```

Options: `--height 330` how high it flies (m, at every place of its walk; by default as the scene has it), `--at 180` when it is over the centre spot, in seconds into the song as the sound system plays it (by default halfway through); `--speed`, `--broadband`, `--step` and `--input helicopter=WAV` as in the flyover demo, and `--tour` and `--listeners` as in the stadium demo (the helicopter is then simulated at those listeners alone, over the concert's mixes there). At the mix position the walk takes 2 minutes on 4 cores, half of it playing the 301 places through with Doppler, so some 25 for the 12 listeners.

- **The flight.** The stadium's scene has the helicopter, left out of the concert's own simulations, flying 1200 m along the stadium's long axis at 63 m/s, from behind the stage over the centre spot to past the east stand, with its rotors' sound power and directivity (`helicopter.csv`) at that speed through still air. `flight` writes `scene_flight.yaml`: the stadium's scene with the helicopter at `--height` when that is given. As the scene has it, it flies 150 m up, as a police or a television helicopter given leave may. Without leave the EU's rules of the air keep a flight by sight (VFR) over an open-air assembly of persons 300 m above the highest obstacle within 600 m (SERA.5005(f)), here the stadium's roof, 30 m up: `--height 330`. The ground and the sky are widened to take the flight; the responses run the scene's 2.0 s past the latest direct sound, as every outdoor response does (4.0 s in all, against the concert's 2.7 s), so each keeps the stadium's decay after the helicopter's sound arrives.
- **The helicopter and its pass** are the flyover demo's steps on that scene: `helicopter` its rotors' sound, `scene` the helicopter alone (`scene_overhead.yaml`) with the sound power and the directivity of its rotors at 63 m/s through still air (137.1 dB, 132.8 dB(A)), and `walk` the helicopter every 4 m along its flight at each of the 12 listeners, with Doppler. At the mix position it peaks between 81 and 82 dB(A) (about 74 from 330 m), where the sound system's loudspeakers, simulated at their sound power, give 97 dB(A): it is some 15 dB under the music, heard most in the song's quiet passages. High in the north stand, under the roof, it peaks 10 to 12 dB lower (four runs): the sound there is almost all late sound, which a few rays can lift by several decibels at one place.
- **The mix.** `mix` adds the helicopter's pass at each listener to the concert's mix there (`demo/seats`), the helicopter over the centre spot at `--at`. The feeds stand for the loudspeakers' sound power, a gated RMS of 0.1 for the loudest one's (the main arrays', 134 dB), and the helicopter's sound is in pascals 1 m from its hub, so the mix scales it by 0.1 over the pressure 1 m from a source of the main arrays' sound power, in the scene's air. The mixes get one gain for all (the loudest peak at -1 dBFS), into `demo/crossing/seats`, and the feeds' table goes to `demo/crossing/feeds` with the song lasting until the helicopter has gone.
- **The video** is the stadium demo's, made from those mixes into `demo/crossing`: the same tour, each stop turned up to the loudness of the loudest, the helicopter named in the title and in the credit. The tour goes on as the stadium demo's does: halfway through each of the three songs it is in the north stand, at its far seats and then at its top row, under the roof, where the helicopter is quietest. `--at 10` brings the helicopter over during the first stop, at the mix position.

What it leaves out: the stadium's air is still, so no wind or eddies bend the helicopter's sound. The rotors' downwash does not reach the crowd: flying past at this speed, their wake trails behind them and sinks a few metres a second (1.6 m/s through the main rotor's disc in its trim). The rotors' model is 3.4 dB louder than the helicopter's noise certificate, as in the flyover. The stills draw the stadium as the stadium demo's do, without the helicopter, and the concert's mixes are taken as they are: one rendered with other HRTFs than `--hrtf` names is not caught.

Road works in the street

`works_demo.py` comes with the `street` example too: road works in the lane on the houses' side, 20 m along from the pavement's listener, heard on the pavement, on the balcony and in the garden in turn, binaurally, and a video that tours them.

```
acoustic-spaces examples --copy . street && cd street
python works_demo.py all   # or scene, feeds, simulate, render, video one at a time
```

Options: `--blows 480` how many blows a minute the breaker strikes (Epiroc's MB 1650, a 1650 kg breaker, strikes 320 to 640), `--seconds 20` how long each listener is heard, `--listeners pavement,garden` which listeners and in which order (every one in use by default; `--tour`, its earlier name, still works), `--indoors off` where they are heard (see below), `--hrtf me.sofa` a measured head, `--input NAME=WAV` a dry recording of yours instead of a made sound (`breaker`, `generator`, `dumper` or `dumper_alarm`). The scene with the works alone goes to `scene_works.yaml`, the same heard 2 m in front of the offices to `scene_works_2m.yaml` (see Indoors below), everything else to `demo_works/`. In the editor, **Simulation > Run demo…** on the street's scene has it beside the pass-by demo.

- **The machines** are rows of Defra's noise database for construction sites (*Update of noise database for prediction of noise on construction and open sites*, Hepworth Acoustics for Defra, 2005, the data BS 5228-1 Annex C gives; `acoustic_spaces.demos.plant`): each one's octave-band levels 10 m away, its sound power 28 dB over them, a point over hard ground radiating into a half sphere.

source	Defra's row	height	10 m away	sound power
`breaker`	a 1650 kg breaker on a 15 t excavator of 121 kW, breaking up brick foundations (table 1, row 9)	1.0 m	90 dB(A)	118.3 dB(A)
`generator`	a 6 kVA diesel generator at 3000 rpm (table 4, row 87)	0.5 m	65 dB(A)	92.9 dB(A)
`dumper`	a 9 t dumper of 75 kW, the loudest moment of its drive-by (table 4, row 4)	1.5 m	76 dB(A)	103.5 dB(A)

  The heights are assumed. Their sound is made to that power: the breaker's blows, a burst of noise each at `--blows`, and the engines' firing, the generator's at its 3000 rpm (one cylinder, assumed) and the dumper's at 2200 rpm (four cylinders, assumed), each brought to its row's octave spectrum. Defra's levels are averages over an activity, its pauses too; the made sounds are steady.

- **The reversing alarm** (`dumper_alarm`), at the dumper's back, 1.2 m up, facing the works: a tonal alarm at 1200 Hz, 107 dB(A) 1.2 m in front of it on its axis (SAE J994 Type B; Hella's 009148021 is one), on and off for as long, 1.5 times a second (J994 asks for 1 to 2; the rate is assumed within that), as directional as the siren NBS measured (assumed, as for the fire engine's air horn). Its sound power is 111.7 dB(A) while it sounds and 108.6 dB(A) on average over its beeps, which the scene holds. The pavement's listener is in front of the dumper, behind its alarm, 11 dB under its axis.
- **What each place hears** (the simulated level of each source, dB(A)):

	breaker	generator	dumper	alarm	all
pavement	91.0	65.4	77.8	79.7	91.5
balcony	89.0	62.9	75.7	77.3	89.5
garden	72.1	48.0	57.7	58.5	72.5

  The houses stand between the works and the garden, which hears them over and round the houses and the barrier, 19 dB(A) under the pavement.

- **A tone is heard as the street makes it at its frequency.** The table's levels are each octave band's, as a sound spread over the octave has them. The alarm's beeps are one tone, where the reflections off the road and the facades add up or cancel: they come out 2.9 dB under the table's level on the pavement, 0.7 dB over it on the balcony and 5.3 dB under it in the garden (the simulated responses at 1200 Hz, which change a few tens of centimetres away). The engines' firing, a row of tones, comes out within 1.1 dB of the table's; the breaker's blows, noise, within 0.4 dB. The captions give the octaves' levels.

- **Indoors** (`--indoors`; `acoustic_spaces.demos.indoors.StillHouses`): as in Trains passing by, the pavement's listener, 2 m in front of the houses, and the balcony's, 1 m in front of the offices, hear the works in a room behind that wall, and the garden hears them outdoors: `works_and_rumble` the works through the closed window and the rumble of the breaker's and the dumper's vibration, `works` the window alone, `rumble` the rumble alone. The offices' room takes its level outdoors at the point 2 m in front of their wall, 1 m out from the balcony: the `scene` step writes that point (`balcony_2m`) to `scene_works_2m.yaml`, which `simulate` simulates too. The vibration is each machine's level at its distance from the room's wall by the FTA's eq. 7-3 from table 7-4 (a hoe ram, 87 VdB at 25 ft; loaded trucks, 86 VdB), their energies added, with no adjustment for the building, as the manual compares plant's level with its limits as it is; the rumble is 50 dB under it in dB(A), as the trains'. It is 29.8 dB(A) in the houses (their floor at 79.8 VdB) and 25.5 in the offices (75.5), and steady, as the breaker's blows come faster than noise at 25 Hz can swell and fade. The garden's ground shakes at 66.1 VdB, just felt. The captions give what is felt at each listener, and each one's vibration is a bass shaker's track (`<listener>.shaker.wav` beside the mixes, `tour.shaker.wav`), as the trains' are.

What it leaves out: the dumper stands still while its alarm sounds, as if it reversed on the spot; the machines' bodies are not in the scene, so nothing of the dumper shields the alarm, and Defra's levels are taken as coming from one point at each machine's height. See Validation for the rows against their own levels and the EU's limits for outdoor machines.

Aircraft taking off and landing

`flight_demo.py` comes with the outdoor example `airport`: Doc 29's hypothetical wing-engined twin jet JETW (ECAC Doc 29 Volume 3) takes off from a runway across a meadow, or lands on it, and each listener hears its pass in turn, binaurally: under the departure 4 km from its start of roll, beside the runway 450 m from its centre line (2.5 km from the start of roll) and under the approach 2 km before the threshold, all 1.2 m above the grass. It is the road vehicles' demo with the jet in the car's place (`acoustic_spaces.demos.aircraft`): the same steps (the first one `aircraft`), options and video, but no `--speed`, since Doc 29 times the flight.

```
acoustic-spaces examples --copy . airport && cd airport
python flight_demo.py all                    # the departure
python flight_demo.py all --vehicle arrival   # the arrival
```

Options: `--vehicle arrival` the landing instead of the take-off, `--step 4` how far apart the jet is simulated along its flight (m), `--hrtf me.sofa` a measured head, `--input departure=WAV` a dry recording of yours, `--listeners lateral_450m` the listeners it is simulated and heard at.

- **The flight.** The example's sources `departure` and `arrival` are points whose walks give the ground track, along the runway's centre line in the example. The demo flies JETW along it as Doc 29's Appendix B does, at Volume 3's default weights in the scene's air (15 °C at sea level, ANP's reference headwind of 8 kt), and writes the flight into the demo's scene as the walk's keyframes, its pitch falling as it passes (Doppler). The departure flies the ICAO-A noise abatement procedure: from standing at its start of roll, it lifts off 1493 m down the runway at 301 km/h, climbs at take-off thrust to 1500 ft (457 m, 4.0 km from the start of roll), cuts back to climb thrust by 4.3 km and is 740 m up at the end of its walk, 6.5 km out, after 95.0 s. The arrival comes down a 3° approach from 225 m up at 253 km/h, 120 m up over the approach's listener, touches down 290 m past the threshold, puts on reverse thrust and rolls out to 50 km/h 1.6 km down the runway, after 92.7 s. A walk of your own gives another ground track (Edit movement…); the departure's starts where its walk does, and the arrival touches down where its walk first comes down to the runway, at most 1 m up.
- **Its sound** is noise with the octave spectrum of its sound power, which comes from its NPD curves and spectral class (an aircraft in a scene): 156.0 dB(A) at the departure's highest thrust (25,000 lb, at its start of roll) and 145.7 dB(A) at the arrival's (its reverse thrust, 10,000 lb), louder and softer as the thrust changes along the flight: the departure's climb thrust 8.0 dB under its start of roll, the arrival's approach 2.3 dB under its reverse thrust and the end of its roll-out 3.5 dB under. Its directivity is its engines' installation correction (under the wings), in a table beside the demo's scene (`scene_demo_departure.csv`). `demo/departure/engines.wav` is the made sound, dry, as long as the flight. `--input departure=WAV` plays a dry recording instead, at least as long as the flight, its level following the recording's.
- **The pass**: the jet every 4 m along its flight at each listener in turn, through the direct sound and its reflection off the ground. Places 4 m apart follow the exact field of a flyover closely from 100 m away and more (validation). A flight of 6.5 km is some 1600 places at each listener, which takes a while.

The captions give the pass's L_Amax, the highest level of the jet's places, each as heard with the jet standing there, and its SEL over the whole flight. They also give it as Doc 29 measures it: L_ASmax, the highest level through a sound level meter's slow time weighting, which lags about a second behind the passing jet, and the SEL over the time that level is within 10 dB of its highest.

While the jet is in the air, what each listener hears is within 1.0 dB of Doc 29's own sum over the same flight path's segments, but for the arrival's SEL at approach_2000m, 2.0 dB under. While it rolls on the runway they part by up to 12.6 dB, as the engine reckons the sound running along the ground in still air over the scene's runway and meadow, where Doc 29 takes off an attenuation it has from measurements (the check).

What it leaves out: one aircraft, Volume 3's hypothetical JETW, and no houses. Its sound is noise with one spectrum all along, with none of a jet's tones (its fan's, its buzz-saw noise), and the NPD curves give no directivity along the flight path but behind the take-off roll, so the source has none either.

Measurements

Measuring impulse responses

The same parameters can be measured in a real room with an exponential sine sweep, a loudspeaker, a measurement microphone and an audio interface:

```
acoustic-spaces devices                                  # pick the interface
acoustic-spaces measure --out hall --device 3 --input 1 2 --loopback 4 \
    --duration 10 --silence 4 --repeats 3 --plot
```

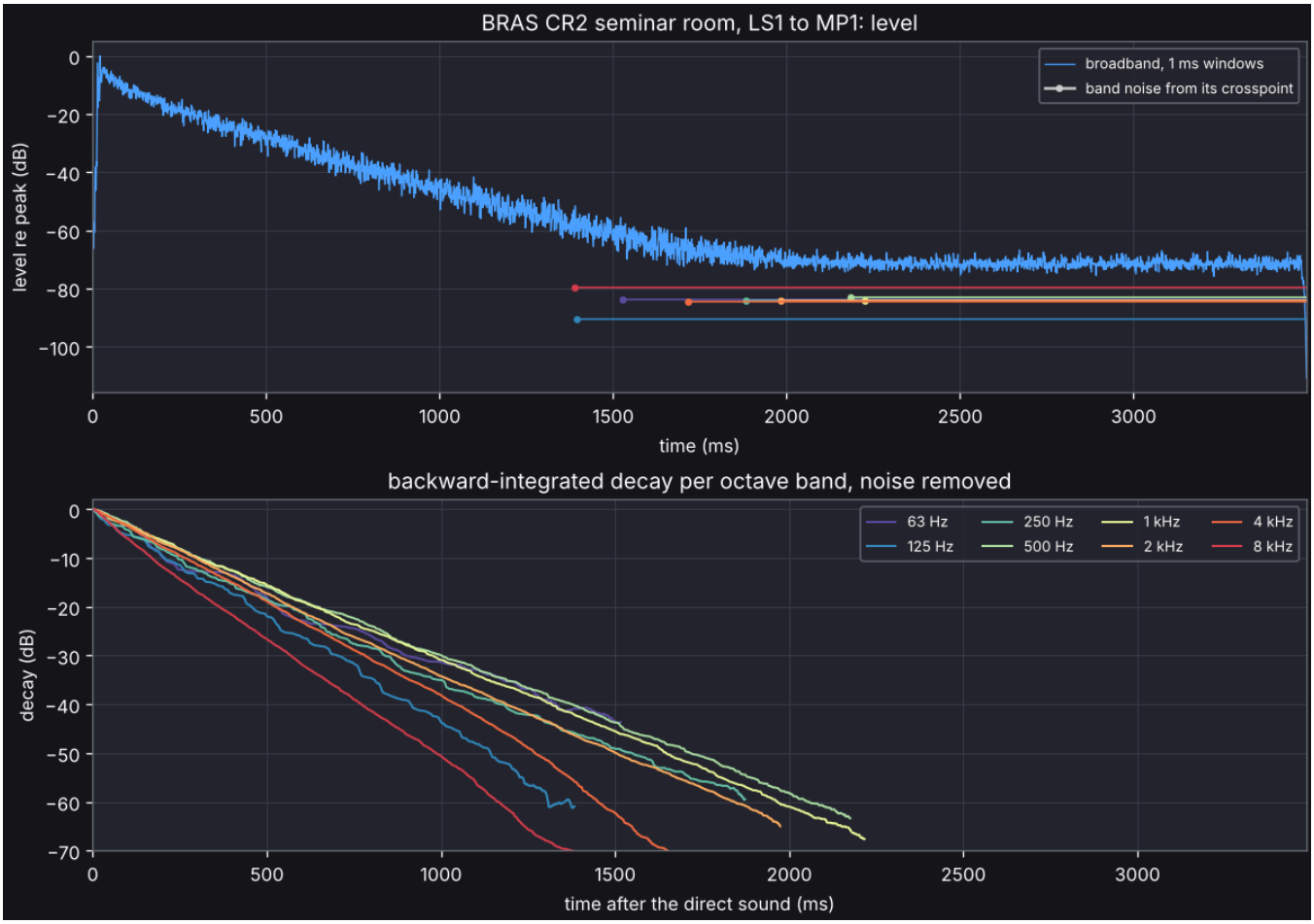
`measure` plays the sweep on `--output` (1 by default), records every `--input`, deconvolves the recording and prints the parameters of each microphone. It writes `sweep.wav` and its settings, the raw `recording.wav`, the impulse responses in `ir.wav` (one channel per input), `measurement.json` and, with `--plot`, a level and decay plot per input. A `--loopback` input wired straight to the output removes the latency of the interface, so arrival times are true times of flight.

Any other player and recorder works too: write the excitation, play it while recording, then deconvolve and analyse:

```
acoustic-spaces sweep sweep.wav --duration 10 --silence 4 --repeats 3
# ... play sweep.wav and record recording.wav, starting together ...
acoustic-spaces deconvolve recording.wav --sweep sweep.json --out ir.wav
acoustic-spaces analyze ir.wav --plot ir.png --json ir.json
```

How it works:

- The sweep is synchronized (Novak et al., 2015), so harmonic distortion of the loudspeaker lands before the linear response and is cut away. Make the silence after each sweep longer than the reverberation time: the responses are as long as the silence (`--length` can shorten them) and no longer, as the next sweep's distortion lands right after it, and a recording that stops with the excitation lacks the high frequencies of the response beyond it. Only a single sweep recorded for longer gives longer responses, up to the recording's length less the sweep's. Repeats are averaged, each doubling lowering the background noise by 3 dB.
- Deconvolution divides spectra, regularised outside the swept band.
- `analyze` filters with IEC 61260-1 octave filters (Butterworth, order 3, zero phase), finds where each band sinks into the background noise (Lundeby et al., 1995), cuts it there, subtracts the noise and adds the missing tail (ISO 3382-1, 5.3.4). Digital silence at the end of a file (zero padding, or the shift that removes latency) is not taken for noise: the noise floor is looked for before it. Decay times are left out when a band's decay does not clear the noise by enough: 20 dB for EDT, 35 dB for T20, 45 dB for T30.
- A band whose lower edge is at or above half the sample rate holds nothing of its own: all its parameters are left out, and so is the STI. At a sample rate below 11.25 kHz that is the 8 kHz band, from 5.62 kHz up. A band that reaches past half the sample rate (the 4 kHz band at 8 kHz, the 8 kHz band at 16 kHz) is analysed from its part below it, with `--reference-energy` taken over the same part. A note says so.
- Each T20 and T30 comes with its ISO 3382-2 non-linearity ξ (‰) and the decay's curvature (%), left out with the decay time they check (see Simulation): past 10 ‰ or 10 %, the decay is not straight enough for one decay time to describe it.
- G needs the source's free-field level: `--free-field ff.wav --distance 3` uses a measurement of the same source in free field (or with reflections windowed out), and `--reference-energy 0.01` suits responses made by `simulate`. `simulate --normalize` scales every file of a run by one gain, its `ir_gain` in `parameters.json`, and those take 0.01 × `ir_gain`^{2}.
- `analyze` also prints the STI and STIPA of the room alone and the echo criteria. `--lateral 2` takes channel 2 as a figure of eight with its null on the source, for LF, LFC and LJ; `--ambisonic` reads first-order AmbiX (W, Y, Z, X) facing the source, for those and the T30 of X, Y and Z; `--binaural` takes two channels as the ears and adds IACC; `--stage` adds stage support for a response measured 1 m from the source.



The response above was measured in a seminar room of the BRAS benchmark (Aspöck et al., DOI 10.14279/depositonce-6726.3, CC BY-SA 4.0, which the picture keeps): room CR2, loudspeaker 1 to microphone 1.

From Python:

```
from acoustic_spaces.measurement import Sweep, deconvolve, measure
from acoustic_spaces.parameters import from_impulse_response

sweep = Sweep(duration_s=10, silence_s=4, repeats=3)
result = measure(sweep, inputs=[1, 2], loopback=4, device=3)
for ir in result.impulse_responses.T:
    print(from_impulse_response(ir, sweep.sample_rate).table())
```

Comparing with measurements

A model is checked against the room it stands for by comparing its parameters with those measured there. `compare` analyses a simulated and a measured impulse response the same way, each at its own sample rate, and shows band by band how far apart they are in just noticeable differences (JNDs): 5 % of the measured EDT, 1 dB of C80 and G, 0.05 of D50 and 10 ms of Ts, as ISO 3382-1:2009 gives them (annex A), and by analogy 5 % of the measured T20 and T30 and 1 dB of C50.

```
acoustic-spaces simulate hall.yaml --out sim
acoustic-spaces compare sim/talker__seat-a.wav hall/ir.wav --measured-channel 2 \
    --simulated-reference-energy 0.01 --measured-free-field ff.wav --measured-distance 3 \
    --plot compare.png --json compare.json
```

Its output, for a made-up hall (the numbers show the form, not a result):

```
simulated: talker__seat-a.wav, channel 1, 48000 Hz, direct sound at 10.1 ms
measured:  ir.wav, channel 2, 48000 Hz, direct sound at 10.0 ms

                  63 Hz  125 Hz  250 Hz  500 Hz   1 kHz   2 kHz   4 kHz   8 kHz
...
T30 (s)   sim     1.533   1.372   1.267   1.111   1.030   1.004   0.899   0.761
          meas        -   1.252   1.214   1.029   1.040   0.959   0.852   0.731
          JNDs        -    +1.9*   +0.9    +1.6*   -0.2    +0.9    +1.1*   +0.8
C50 (dB)  sim      -1.6     2.0    -2.0    -1.9     1.5    -0.1     0.3     1.8
          meas     -1.9     0.7     2.2    -1.2     1.8     0.5     1.3     2.3
          JNDs     +0.2    +1.3*   -4.1*   -0.7    -0.4    -0.6    -1.1*   -0.5
...
JNDs: simulated minus measured, in just noticeable differences; * beyond one
missing:
  T20 at 63 Hz: the measured decay clears the noise by 34 dB, T20 needs 35 dB
  T30 at 63 Hz: the measured decay clears the noise by 34 dB, T30 needs 45 dB

            within 1 JND   mean |JNDs|
EDT (s)           4 of 8           2.0
...
```

- Each JNDs row is the simulated value minus the measured one; `*` marks the bands more than one JND apart, where the difference can be heard.
- The summary gives, per parameter, how many of the compared bands are within one JND, and the mean absolute difference in JNDs from 125 Hz to 4 kHz.
- A band where either side has no value shows `-` and is listed with the reason, never counted as a difference of zero: a response that decays by less than 20 dB, a decay time whose range does not clear the background noise, G without a reference, or a band above half a response's sample rate.
- Each file has its own channel (`--simulated-channel`, `--measured-channel`, 1 by default) and its own reference for G, given as for `analyze`: `--simulated-reference-energy 0.01` suits responses made by `simulate` (0.01 × `ir_gain`^{2} with `--normalize`, as above), and the measured side takes `--measured-free-field` with `--measured-distance`, or `--measured-reference-energy`. G is compared only when both sides have one.
- `--filters` and `--no-noise` apply to both files. `--json` writes every value and reason, and `--plot` overlays the two decays in each octave band (viz extra).

From Python, `compare_impulse_responses` does the same, and `compare_parameters` compares two sets of parameters already computed:

```python
from acoustic_spaces.audio import read_wav
from acoustic_spaces.compare import compare_impulse_responses

sim, sim_rate = read_wav("sim/talker__seat-a.wav")
meas, meas_rate = read_wav("hall/ir.wav")
comparison = compare_impulse_responses(
    sim[:, 0], sim_rate, meas[:, 1], meas_rate, simulated_reference=0.01
)
print(comparison.table())
t30 = comparison["t30"]
print(t30.difference_jnd, t30.within_jnd, t30.mean_abs_jnd, t30.missing)
```

`compare_parameters` also compares LF and LFC (0.05) and IACC_E and IACC_L (0.075, annex B) where both sets have them, and the directional decay times with the JND of T30. LJ and stage support have no JND in the standard and are not compared.

Fitting materials to measurements

Absorption from tables seldom matches a room as built. `calibrate` adjusts the absorption of the scene's materials, octave band by octave band, until the simulated T30, EDT and C80 at each measured source and listener pair come as close as they can to those measured, in JNDs:

```
acoustic-spaces calibrate hall.yaml \
    --measured talker seat-a hall/ir_a.wav --measured talker seat-b hall/ir_b.wav \
    -o hall_fitted.yaml --json calibration.json
```

Its output, for the same made-up hall:

```
absorption     63 Hz  125 Hz  250 Hz  500 Hz   1 kHz   2 kHz   4 kHz   8 kHz
floor           0.09    0.14    0.23    0.33    0.40    0.46    0.47    0.46
  was           0.13    0.20    0.33    0.46    0.52    0.59    0.59    0.59
  +-            0.36    0.20    0.23    0.08    0.08    0.04    0.08    0.02
...
mean |error|   63 Hz  125 Hz  250 Hz  500 Hz   1 kHz   2 kHz   4 kHz   8 kHz   (JND)
T30 before      5.86    5.84    5.36    5.78    5.32    5.14    5.77    5.06
T30 after       0.04    0.07    0.07    0.11    0.08    0.08    0.06    0.09
...
```

- Each `--measured SOURCE LISTENER FILE` is an impulse response measured where the scene has that source and listener: a WAV (its first channel, or `--channel`), or a report written by `analyze --json`. `--measured-json` takes parameters already measured, as `{"pairs": {"SOURCE LISTENER": {"t30": [...], "edt": [...]}}}` with eight octave values each (null where missing), the format of `validation/bras/*/measured.json`. A band with no measured value (null, or above half a WAV's sample rate) is not fitted to, and one that no pair has keeps its absorption.
- Every material on a surface is fitted (at most 16), or those `--materials` names; `--parameters` picks others to match (T20, C50, D50, Ts, G).
- The table gives each material's fitted absorption, what it started from (`was`), and `+-`: its standard error, how far it could move, the other materials following it, before the sum of the squared misses (in JNDs) grows by 1. Where that is a tenth or more, the measurements hardly decide it. `-o` writes the scene with the fitted absorption (a material given in third octaves keeps its shape within each octave), and `--json` every number.
- The materials keep their proportions unless the measurements ask otherwise. Measurements fix the total absorption best. Telling two materials apart needs responses that hear them differently, and surfaces that the sound meets alike (the floor and ceiling of a flat room) trade absorption freely. So the fit may scale every material by a common factor, but moving one by a factor of 2 more than the others costs as much as missing a parameter by one JND. With measurements that cannot tell the materials apart, it is the fit of one factor per band, like fitting the reverberation time with Eyring's formula, but with the rays of the model itself.
- It does not trace the room for every trial. Each round traces it once, counting each ray's reflections off each material, and the responses for other absorption come from re-weighting those rays (`acoustic_spaces.engine.reweight`); the next round traces again at the absorption found, until it moves by less than 0.01 or `--rounds` (4) run out. Within a round the early reflections keep the phases they were traced with where they add in pressure (up to 125 Hz indoors by default), which the absorption also sets; the next round's trace has them at the absorption found, and the parameters reported after the fit come from a last trace. A round costs about one simulation of the measured sources at the scene's rays (`--rays` overrides them).
- It needs stochastic ray tracing in octave bands, without the wave solver.
- On the measured rooms of the validation report, fitted to one loudspeaker position from tabulated absorption, the responses at the other position come within about one JND in the seminar room, as with the absorption BRAS fitted with Eyring's formula, and closer than with it on the decay in the hall and the auditorium. Coupled rooms need responses in each room: fitted to the loudspeaker in one of BRAS's two coupled rooms, the responses to the one in the other come out 55 JNDs off in EDT.

The `fitting` example goes through it with made-up measurements of a seminar room whose back wall was built without the absorbing panels of its design.

From Python:

```
from acoustic_spaces.calibration import calibrate
from acoustic_spaces.parameters import from_impulse_response

measured = {("talker", "seat-a"): from_impulse_response(ir_a, 48000)}
found = calibrate("hall.yaml", measured)
print(found.table())
found.fitted_scene().to_yaml("hall_fitted.yaml")
```

Running on another computer

Long runs (many rays, third-octave bands, the wave solver up to 250 or 500 Hz, big coverage maps, outdoor scenes over kilometres) can run on a computer with more cores or a GPU: a server you reach over SSH, a rented GPU machine, or a job on a cloud service such as Hugging Face Jobs. The scene goes there as a *pack*, one zip file; a container image runs the command line on the GPU there; the results come back as the same files `simulate` and `map` write here.

Nothing in acoustic_spaces uploads anything: you choose where a pack goes, and it holds everything needed to rebuild the scene (its model, materials and positions). Send it only where you would send the drawings themselves.

Packs

```
acoustic-spaces pack hall/scene.yaml hall.zip
```

writes the scene and every file it names: its model, terrain, buildings map, material libraries and coefficient files, directivity files, the WAV files sources play, the scene's recording file and HRTFs, with the files those name in turn (a `.gltf` model's buffers and images, a loudspeaker array's element data). It lists what went in and their sizes. Files under the scene's folder keep their place relative to it; files elsewhere go under `external/<their folder>/`. The scene file in the pack names them all relatively, so the pack runs wherever it is unpacked. The editor does the same from **File > Pack for another computer**, also for a scene not yet saved. The zip is written as `hall.zip.part` and takes the name only once it is complete, so a pack that fails leaves an earlier `hall.zip` as it was.

A scene with problems (a listener outside the room, surfaces without a material) is not packed, as `simulate` would not run it either; warnings are shown. A file that names one outside its own folder (`../shared/element.csv` in an array's YAML) cannot keep its place in a pack: move that file next to it. A file whose name holds `:` or `\`, which Windows refuses and unpacking refuses everywhere, is not packed, nor are two files, or a file and a folder, whose names macOS or Windows take for one: names that differ only in case (on macOS also `straße` and `strasse`), or in the way a letter is encoded (é as one character, or as e and an accent, which macOS alone takes for one). Rename them.

Every command that reads a scene takes a pack in its place: `simulate`, `map`, `validate`, `inspect`, `calibrate`, `walkthrough`, `export`, `view`, `animate` and `tune`, with the same options.

```
acoustic-spaces simulate hall.zip --out results --bands third_octave   # on the other
computer
```

The pack is unpacked into a temporary folder for the command and removed after it, once the folder is found to have room for it. `pack.json` in it records the version that wrote it and a SHA-256 of every file, which each file is checked against before it takes its name, so a damaged pack is refused, and a pack written by another version runs with a warning: results can differ between versions, so for numbers to compare, run the same version on both computers. Runs of a pack give the same results, bit for bit, as runs of the scene file on the same computer.

The container image

The repository's `Dockerfile` builds an image with the command line, the `gpu` and `sofa` extras, the Vulkan loader and Mesa's Vulkan drivers, on Ubuntu 24.04. It runs headless: the wave solver and the late rays reach the GPU through Vulkan (wgpu), with no display. Build it from a checkout:

```
docker build -t acoustic-spaces .
docker build -t acoustic-spaces --build-arg EXTRAS=gpu,sofa,ifc .   # packs with IFC models
```

Its entry point is `acoustic-spaces`, so the container takes a command and its options; without one it runs `info`. First check that it finds the GPU:

```
docker run --rm --gpus all acoustic-spaces info
```

`GPU:` should name the card, `(Vulkan, DiscreteGPU)`. With `llvmpipe ... (Vulkan, CPU)`, Mesa's software driver, or `none`, the GPU's Vulkan driver does not reach the container and every run takes the CPU:

- NVIDIA: the host needs the NVIDIA driver and the NVIDIA Container Toolkit, and the container the driver's *graphics* capability, which holds its Vulkan part. The image asks for it (`NVIDIA_DRIVER_CAPABILITIES=compute,utility,graphics`); a service that sets `compute` alone, as many CUDA images do, leaves Vulkan out. Older toolkits mount the library without telling the Vulkan loader where it is; the image's `acoustic-spaces` command finds it and writes that file itself.
- AMD and Intel: Mesa's drivers are in the image; pass the GPU's device nodes with `--device /dev/dri` and, where they belong to the `render` group, `--group-add "$(getent group render | cut -d: -f3)"`.

On the CPU, runs take every core (`simulation.performance.threads: 0`), the same as here. Device `auto` times the CPU against the GPU on the machine it runs on (see Simulation); a container keeps no timings between runs, so each run times anew, unless the scene or the options name `cpu` or `gpu`, or the cache folder (`/tmp/cache`) is a volume that `acoustic-spaces tune` filled once.

Run a pack with the current folder mounted as `/work`, the container's working folder, writing as your own user:

```
docker run --rm --gpus all -v "$PWD:/work" --user "$(id -u):$(id -g)" \
    acoustic-spaces simulate hall.zip --out results
```

A server over SSH

On any machine you reach over SSH (your own, or one rented by the hour with a GPU), build the image there from a checkout, or send it from here:

```
docker save acoustic-spaces | ssh server docker load
scp hall.zip server:jobs/
ssh server 'cd jobs && docker run --rm --gpus all -v "$PWD:/work" \
    --user "$(id -u):$(id -g)" acoustic-spaces simulate hall.zip --out hall-results'
scp -r server:jobs/hall-results .
```

Without Docker, install acoustic_spaces on the server as here (with the `gpu` extra, and a Vulkan driver for its GPU) and run the pack the same way.

Hugging Face Jobs

Jobs run a container on Hugging Face's hardware, CPUs or NVIDIA GPUs from a T4 to an H200, paid by the minute from the account's credit (`hf jobs hardware` lists them and their prices). They take a public image or the image a Docker Space builds; a private Space keeps the image private. With the `hf` command (`pip install -U huggingface_hub`, then `hf auth login`), once:

```
hf repos create <user>/acoustic-spaces --type space --space-sdk docker --private
mkdir space && git archive HEAD | tar -x -C space
printf -- '---\ntitle: acoustic-spaces\nsdk: docker\n---\n\n' | cat - README.md >
space/README.md
hf upload <user>/acoustic-spaces space . --type space
```

The Space builds the image from the `Dockerfile`; it then shows a runtime error, as it serves no web page, which does not matter: Jobs only take its image. Upload again after updating the checkout, and Jobs take the new build. Then, for each run:

```
mkdir -p jobs && cp hall.zip jobs/
hf jobs run --flavor l4x1 --timeout 2h -v ./jobs:/jobs:rw \
    hf.co/spaces/<user>/acoustic-spaces \
    acoustic-spaces simulate /jobs/hall.zip --out /jobs/results
```

The folder `jobs` is copied to a storage bucket of the account and mounted in the job, which writes the results there; the command prints how to sync the bucket's folder back when the job starts. Check the GPU first with a short job running `acoustic-spaces info`; a job stops at its timeout (30 minutes unless `--timeout` says otherwise) and is billed while it starts and runs.

Validation

This report sums up how far the results of acoustic_spaces can be trusted: what the test suite checks against theory on every change, how the simulation compares with another program, pyroomacoustics, and with measurements in two real rooms, how edge diffraction, reflections fading at edges and the reflection off a porous ground compare with the exact solution, how sound outdoors compares with measurements up to 845 m away, how its turbulence compares with a parabolic equation through random eddies, how its rays through vegetation compare with a separate particle simulation, how its wave solver in wind and temperature compares with a parabolic equation, how the weather’s effect behind barriers and buildings compares with a full-wave solution, and how much of a block’s time the real-time convolver needs. It ends with the known issues and what has not been validated yet.

Checks against theory, on every change

The test suite runs on Linux, macOS (Apple silicon and Intel) and Windows for every pull request and checks:

- an 8 x 6 x 3 m box against analytic image sources up to 4th order (1, 6, 18, 38 and 66 paths), for both early methods and for symmetric placements where paths hit edges and corners: positions, delays and energies agree to rounding;
- T30 in a box with fully diffusing surfaces against Eyring with air absorption: within 10 %, and slightly longer as absorption grows, as Kuttruff's correction for the spread of free path lengths predicts; also within 10 % with the example's uneven absorption (absorbing ceiling and carpet, hard walls) once every surface diffuses;
- the decay in the same box with specular walls against its closed form, where each direction of travel meets the walls at its own fixed rate: within 10 % with the absorption spread evenly, and with the example's, where rays parallel to the floor never meet the ceiling and T30 is 5 to 20 times Eyring's estimate;
- G in the same box against Barron's revised theory: within 1 dB;
- the statistical late method against the stochastic one in the same box, with the same absorption on every surface and scattering 0.3 and 1: G within 0.7 dB (up to 0.60 dB at scattering 0.3 and 0.43 dB at 1), C80 within 2.5 dB (higher from 1 kHz up, where the rays decay up to 7 % slower than Eyring);
- the late tail of the example box (absorbing ceiling, smooth walls, random scattering) from 10k rays with splitting against 1 million rays: within 4 dB on average from 0.5 to 2.5 s, and splitting or thinning out every ray keeps the energy of plain ray tracing;
- the late rays in batches: the same late energy as at once (to rounding without splitting; with it, a tipped split moved EDT by at most 0.2 %), and the batches' mean is the late energy; standard errors (a jackknife over 10 batches, 2,000 rays, 30 seeds of a diffuse box) of EDT, T30, C80 and G against the spread of the parameters between the runs, from 125 Hz to 4 kHz: their ratio held within 0.6 to 1.6 band by band and 0.8 to 1.25 on average (over 60 seeds, the root mean square of the standard errors came within 0.90 to 1.17 of the spread); and they follow a source's feed and new absorption (re-weighted) along with the rest;
- the sums the package adds in a fixed order (`acoustic_spaces.products`): every kind of product against the same sum added term by term in Python, bit for bit, on one and on several of its own threads, and against bit patterns stored with the tests, the same on every platform; the early pressure's spectra and its coherent signal, the IACC and a decay line through 20,000 points bit for bit the same on one and on four threads of the linear-algebra library;
- a response's decay times and ratios read the same however loud it is (20 gains, to 1e-9): the decay curve starts at 0 dB exactly, so EDT's fit always starts at the onset;
- the scattered sound of the reflections straight from the source, shared between the rays and the listeners' gather (`tests/test_first_gather.py`), on ground 400 m across: against its integral within 0.2 % for an aircraft 305 m up and 300 m off (over 24 seeds the shared runs spread by 0.08 % of it, the rays' alone by 86 %, and these averaged half of it), and within the noise of 32 seeds for a source 30 m up and one 5 cm above the ground; the same as the rays' alone within the noise through plants, from a source turned to the ground, per face group and arrival direction, and in two rooms through a partition and a doorway; each batch with its share of the gather's directions; on the GPU as on the CPU;

- the late part's last slots: the same rays traced 50 ms longer give the same echogram up to the end, where a ray's time runs out as it crosses a listener's sphere: with and without the rain on the CPU, without it on the GPU, and with rays bent by the wind on both;
- scattering: the energy of an early reflection off a wall against Christensen and Rindel's scattering of its size, to rounding; the early paths and the late part adding up to what the tracer finds on its own within 5 %, with vector-based and with random scattering; a box of tiny surfaces decaying at Eyring's rate within 10 % where its smooth walls alone decay far slower; and a box that diffuses the bass and reflects the treble decaying at Eyring's rate in the bass only, each band traced with rays of its own, with vector-based and with random scattering; off a reflector in free field, vector-based scattering bringing a listener out of the specular paths' reach what random scattering brings, to rounding, and one in reach within 0.5 %;
- third-octave bands: statistical runs in third octaves give the octave runs' parameters within 1 % or 0.1 dB from 250 Hz up (within 5 % or 0.4 dB at 63 and 125 Hz, where the early paths add in pressure and three filters weigh their interference across an octave a little differently from one), and stochastic ones within the tracer's noise; a 100 Hz third-octave absorber decays at its own Eyring rate (within 3 %) between bands that do not; the filter banks sum to their input and sit at -3 dB on the band edges;
- sources: the free-field level 4 m from a source with a sound power per band, a gain and an equalizer, against 10 log10(4 π r²) and the air absorption of ISO 9613-1 (to rounding), with G unchanged by the feed; a delayed source's response 12.5 ms later (600 samples at 48 kHz) with the same parameters, its wave bands too; two alike sources together 3 dB louder and as clear as one, and less clear with one of them 150 ms late; the speech spectra of IEC 60268-16 summing to 0 dB(A); and the STI of female speech, weighted to the treble, higher where the bass is blurred;
- line and surface sources: a 40 m line's free-field G at 2 and 4 m equal to the sum of its 80 points with their air absorption (to 1e-9 dB), falling 3.31 dB per doubling of distance as the continuous line does; an 8 x 8 m Lambert surface within 0.15 dB of the integral over the continuous square at 1 and 2 m, and silent behind; in the shoebox a 3 m line as loud as a point source whose paths add in energy, as the line's uncorrelated points' do (within 0.6 dB at 2000 rays per band) and its T30 within 10 % of the point source's at 30,000 rays per band;
- coherent arrays (their far-field patterns; nearer than L² f / c the room's warnings say so unless the array adds up its elements one by one, `near_field`): a column of 8 omnidirectional elements 20 cm apart against the analytic sums of a line array, its directivity index within 0.05 dB in all 24 third-octave bands (n² over the band mean of the pairs' sin(kr)/kr) and its vertical pattern within 0.3 dB of the band-averaged |sin(Nψ/2)/sin(ψ/2)|² wherever that is above -15 dB; a steered column's beam within 1 degree of its aim, and the same as delays given element by element; two close elements in opposite polarity a dipole (4.77 dB); two cardioids splayed ±30 degrees their exact first-order sum; and in a simulation, levels ahead of and above a column that differ by its pattern exactly;
- coherent arrays element by element (`near_field`): a steered column of 8 cardioids 3 m from its centre against the sum of its elements' pressures over 20,000 frequencies in each third-octave band, within 0.001 dB in all 24; 100 km away its pattern's own weights (within 0.00001 dB); a 3.75 m column of 16 elements on axis losing 2.8 dB per doubling of the distance at 1 kHz from 1 to 32 m, inside the 41 m where its far field starts (3 dB for a line), and 6.02 dB from 128 m on (a point); and in a simulation, levels 4 m ahead of and above a 1.4 m column that differ from omni by the elements' sum there exactly, ahead 9.5 dB below its far-field pattern at 4 kHz, where the beam has not formed yet;

- walls and openings: two 5 x 4 x 3 m rooms, 20 % absorbing, with a 12 m² partition between them reach the level behind it that diffuse-field theory gives, $G = 10 \log10(1600 \pi S \tau / A^2)$ with Eyring's absorption area A, within 0.4 dB at R = 30 and 50 dB (0.1 to 0.2 dB at 3000 rays per band), and their level difference $R + 10 \log10(A / S)$ within 0.5 dB; two such rooms joined by an opening the size of the wall give the T30 (within 4 %), G (0.3 dB) and C80 (0.5 dB) of one 10 x 4 x 3 m room; and a dry room of 96 m³ opening through a 2 m² doorway into a live one of 192 m³ decays in the two slopes of the two-room diffuse model, within 1.5 dB of its decay curve down to -30 dB and its late reverberation time within 12 % of the model's slow mode (within 1 dB to -40 dB and 5 % at 10,000 rays). Particles reach the next room straight through an opening and not through a closed door, and the wave solver's air flows through the opening. The statistical method follows the two-room model within 0.3 dB of its decay curve from -20 to -40 dB in both rooms, and comes within 0.4 dB of the rays behind the partitions (0.2 dB below them: its field reaches each point from the direct sound on, as Barron has it). Coverage maps over both rooms find at their points what listeners there find (G, T30 and C80 within 0.01 with the statistical method), and the rays' maps the level behind the partitions within 0.4 dB;
- audience seating: the grazing attenuation against its fit's own figures (17 dB at grazing incidence after 11 m, 6.3 dB at 5° and 4.8 dB at 7°, none from 17.8°, linear in dB with distance); the direct sound 18 m over seats at 0.95° 21.1 dB down in the 500 Hz to 2 kHz bands only, the ceiling reflection untouched; a balcony seat shaded only by its own seats and the front stalls the sound skims, not by the stalls far below; with the statistical method, a response that loses exactly what its early paths lose (the tail does not make it up); and a coverage map less clear and weaker over the middle of the seats;
- fitting materials to measurements: a run re-weighted for other absorption against the room traced again with it (the same early paths, with the same energies and, where they add in pressure, the same phases to 10^{-12}, those of reflections on the way round an edge included; T30 within 4 %, EDT within 5 %, C80 within 0.3 dB and G within 0.1 dB; far from the traced absorption, a ceiling of 0.5 taken to 0.99, 0.9 or 0.1, T30 within 15 % and G within 0.25 dB); the late part's exponent against the exact one for reflection counts that fall by chance; the absorption a simulated room was given recovered from its own responses within 0.05 in every band, and a shoebox's walls within 0.015 from 500 Hz up, where the fit's own uncertainty is below 0.05 and above 0.1 in the bands the measurements leave open; a third-octave material that keeps its shape; and the two BRAS rooms (see Measured rooms);
- every ISO 3382-1 parameter on synthetic exponential decays, and the air absorption against the ISO 9613-1 table;
- the further parameters against responses whose values are known: LF and LFC of one reflection from the side, from 45°, from behind and from above, exact from the echogram and within 0.01 from W and Y impulse responses; a third of diffuse late energy lateral (a half for LFC); LJ as the G of the late lateral energy; the T30 of each axis when front-back sound decays in 2 s and the rest in 1 s, exact from the echogram and within 5 % from noise; IACC of 1 for equal ears and ears 0.5 ms apart, and below 0.15 from 2 kHz for unrelated ones; stage support of known energies to rounding; STIPA of exponential decays against Houtgast's closed form within 0.002, and within 0.01 of the STI; U50 against its definition; and the echo criteria of an echo as loud as the direct sound 100 ms later within 10 % of the jump of the centre time, below the 10 % audibility mark for a smooth decay, and for music an echo at 1, 1.4 or 2 kHz reaching the 50 % mark at the same level within 0.25 dB;

- spatial outputs: SN3D spherical harmonics against closed forms and orthogonality, VBAP gains of unit power that point at the panned direction and are mirrored for a mirrored direction, the direct sound's direction in first-order Ambisonics, the mono energy kept by panned layouts (direct sound within 1 %, late part within 5 %), a binaural render through flat HRTFs equal to the mono response, and the spherical head's interaural delay (0.6 ms at the side); HRTFs interpolated between measured directions give the measured responses at those directions and, round the horizon a degree at a time, an interaural delay that moves by less than 50 µs per degree where the nearest measured response jumps by more than 150 µs; mixing pulses delayed by fractions of a sample keeps them flat within 0.5 dB up to 16 kHz, and the spherical head's within 1 dB of the nearest measured response at 8 kHz; a ring of directions or half a sphere interpolates within itself and takes the nearest response outside; the rotation of a sound field of any order turns encoded plane waves exactly (to 1e-9), is orthogonal and composes;
- rendering: the compiled FFT against NumPy, the real-time convolver and the offline convolution against direct convolution (float32 and float64 precision), resampling of tones within 1e-3 of the exact signal up to 15 kHz with nothing above the new Nyquist frequency left, and playback through a simulated sound card block by block;
- edge diffraction: the UTD coefficient against Sommerfeld's exact solution for a rigid half-plane (to 1e-8, also exactly on the shadow and reflection boundaries) and against the exact eigenmode series for wedges open over 225° to 315° (within 0.25 dB); a thin screen against Maekawa's chart (within 1.5 dB), the same from either side; and the early level across the shadow boundary of an L-shaped room, in millimetre steps, continuous within 0.25 dB with first-order reflections where specular paths alone drop by more than 5 dB, and across the boundary of a reflection off one of the corner's walls within 0.05 dB; over a 1 m thick barrier, the path over both corners against the exact BTM solution (within 1 dB, and 0.2 dB from 500 Hz up, the same both ways round; with faces absorbing 0.02, in pressure or in energy, within 1 dB of it too and, straight across the top, below the rigid block's level by the Weyl-van der Pol ground wave's loss along it worked out by hand, within 0.05 dB), and the level as the source and the listener rise through the plane of its top, continuous within 1 dB (0.1 dB from 1 kHz up); a thin screen's diffracted wave in its lit zone, all four terms of the UTD within 0.2 %; and through `early_paths`, next to the shadow boundary of a thin screen, of a right-angled corner and of a wedge open over 225°, against BTM: the bands that add in energy on the boundary as low as the known issues give (within 0.02 dB), and those in pressure within 0.08 dB;
- early reflections in pressure at low frequencies: an impulse response and the echogram it came from in a 7.3 x 5.1 x 3 m box without edge diffraction, early part alone, G within 0.02 dB, C50 within 0.05 dB and Ts within 0.3 ms at 63 and 125 Hz (G 2.1 dB and C50 1.6 dB apart when the paths add in energy); forty pieces of a diffracted wave within a millisecond rendering their energy within 0.2 of it from 250 Hz up on average over 100 seeds, where forty reflections of one phase render more than 20 times theirs at 250 Hz, and the echogram's 63 and 125 Hz bands independent of how much the pieces carry above them; a source and a listener 0.3 m above a floor 10 m apart, the direct sound and the floor reflection 5.5 dB above the free-field level over a hard floor and 4 dB below it over an absorbing one at 63 and 125 Hz, the echogram within 0.3 dB of the paths' pressures added; the phase of the reflection factor, none off a hard surface and half a turn off an absorbing one at grazing incidence; and a thin screen's diffracted wave, with all four terms of the UTD added in pressure, to 1e-9;

- the wave solver, with both stencils where both apply: the modes of a rigid box (within 0.6 Hz), both stencils' modes along a tube at 12, 8 and 5 cells per wavelength as slow as their dispersion relation along an axis says (within 2e-5, the 19-point stencil's error less than half the 7-point one's), the 19-point stencil stable for 30,000 steps in a rotated room with a cube and a screen in it, both stencils' highest modes 2^{-16} inside the stability limit (kicked, within 1.1 × 256 times the kick over 20,000 steps, rounded as the update rounds), the reflection off the end of a duct for four admittances (within 0.005, and below 0.05 for a matched end, and off the inner face of a wall thinner than a cell), and off an end whose admittance resonates (within 0.012 of the grid's theory from 30 to 250 Hz); admittances fitted to four kinds of material (within 0.01 of their absorption in every band, passive); the free-field level of the 63 Hz band (within 0.5 dB of the rays 2.5 cells from the source and 0.1 dB from 5 cells on), the source and the listeners read between the cells' centres (weights that give a cubic's value exactly; next to a wall, a screen and a slot the cells on the point's own side, linearly across the surface, none further than two cells from the point; the level of the direct sound 13 cells away within 0.06 dB wherever the two lie within their cells; along a wall at 45 degrees, no step over 0.4 dB; with a wall behind a listener three cells from the source, none over 0.2 dB), the 19-point stencil's direct sound 12 cells along the cube's diagonal (within 0.05 dB of a point source's, also with a slower speed of sound, and 0.5 dB over it without the taps that undo the grid's own gain), the arrival time and direction of the direct sound and the strength of its first-order field against a point source's (within 0.06 dB 4 m away along a grid axis, a face diagonal and a body diagonal, with either stencil), the part of the source's pulse before emission kept for a listener near the source (the direct sound's band energies 0.5 and 1 m away within 0.01 dB whatever the source's low-pass), a run that starts before emission lined up with one that starts at emission and is fed a delay shorter by the lead (the same field and energy per band and slot, exactly; at a rate where the lead is not a whole number of samples, the same field, within half a sample of its time), the source flat to the top of the highest band at 5 cells per wavelength (within 0.05 dB), the gradient's second-order and one-sided differences where a point two cells or one cell from the listener is missing (exactly, along each axis), the points among the source's own cells left out of a gradient on the 19-point grid (those nearer the source than the listener, and not where a wall leaves an axis no others; the gradient's main axis within 8 degrees of the line to the source 0.6, 1 and 2.2 cells from it), the gradient at a listener just in front of a thin screen and at one 1.5 cells in front of it, taken on the listeners' side, as in front of a solid block (within 1 dB and 0.5 dB), open models and thin screens (stable, and the sound leaves), the matched layer round an open model (what comes back 40 dB down or more, against a grid too large for anything to come back; stable for 9,500 steps; at 63 Hz the sound that runs along it 60 m over the ground under a sky 3 m up, 40 m along a strip of ground 3 m wide and 60 m under a plate within 0.05 dB of the same on a taller, wider or deeper grid; the room it leaves, in whole cells off the model, none under an outdoor ground and none on a side the model's surfaces close, found by flooding the air once), one run for bands with their own absorption, and the level of the late part across the crossover to the rays (flat within 1 dB in sixth octaves);
- the ray tracer against runs stored before its tree was rebuilt (`tests/reference/tracer.npz`): the faces and distances the tree finds, the specular paths, and every kind of late run on the CPU to single precision, so a change that moves a result is seen; the same runs traced on a GPU (in CI, Mesa's software Vulkan driver) land within the spread of the CPU's own results over seeds, and the GPU's ray searches find the CPU's faces; outdoors the GPU's chords through the atmosphere are the CPU's ray for ray, and what its rays bent by the weather or scattered by plants bring to each listener agrees within the noise (see Outdoor scenes on the GPU);

- the leak finder: a closed model keeps every ray; a box without its ceiling reports one gap of 30 m² (6 x 5 m) with every ray through it, at the ceiling's height; the example classroom's two holes, the missing window pane in the east wall (3.6 m², 3.0 x 1.2 m) and the 3 cm slit along the ceiling's edge (0.27 m²), each with the rays that crossed it inside it; a ceiling that lets sound through counted as an opening, not a leak; a source outside a closed model named; and no leak in the other examples;
- walkthroughs: the places along a walk every 25 cm, facing forward and turning the shorter way round at the set pace; crossfade weights that add up to one; crossfades that are continuous, keep a tone heard a quarter or a third of a turn later at the next place to within 0.1 dB of its level, turn a tone in antiphase up by 6 dB at most, keep unlike noise's power (to 0.2 dB) and correct each octave band by its own correlation; the late parts of the places between those traced mixed in proportion to the distance; one response everywhere equal to the plain convolution (to 1e-12); a walking listener, and a walking source heard at a listener; and the gallery walk rendered through its places;
- models read from files: a room whose walls' edges run past the floor's corners (T-junctions, with or without the zero-area slivers SketchUp exports) cleaned into a closed room of the right volume and area; a wall's edge running past the corner where three faces meet split there, and a sliver 1 µm high whose own corner splits the edge it closes dropped, not split into faces with a repeated corner; two rooms touching along an edge closed; a face given twice leaving the model open, with its odd edges counted; points inside a model whichever way its faces point; an IFC office read as one room from its space boundaries (closed, 75.6 m³) or as every element, in the file's units; a Collada room in inches with its up axis as written, and instanced components placed by their nodes; and materials from CATT-Acoustic files and ODEON libraries, the latter as ODEON writes it (Windows encodings and six-band tables too);
- outdoor scenes: over a flat ground under the open sky, exactly two paths, the direct sound and the ground's reflection, at their distances and with the reflection's energy (absorption, spreading and air) to rounding, from both early methods; a barrier's top and ends as its only diffracting edges, with the shortest path over its top; no edge at the inside corner of an L-shaped building, whichever way its footprint runs; points inside buildings refused and left out of coverage maps; and responses that run `ir_length_s` past the latest direct sound;
- grounds with an impedance: Miki's and Delany and Bazley's models at their constants, the two-parameter model's closed form and a layer's impedance; the spherical wave reflected whole off a hard ground, as the plane wave far from grazing and far away, and within 0.4 dB of the exact solution over grass, a thin layer of snow and gravel; the early part over grass within 0.05 dB of the spherical wave in every octave band, however large the ground; the points of a line source added in energy; a ground zone far past a reflection's Fresnel zone changing nothing (to 1e-9), one under all of it the same as a ground of its own, and the change in between smooth; and the wave solver's fit to an impedance within 0.05 of its reflection factor, from normal to near grazing incidence, and within 0.01 of its absorption;

- vegetation: ISO 9613-2's dense foliage to its table, a rigid cylinder's scattering against the exact series' small and large limits, and the depth through belts, hedges mitred at their bends and crowns; the early paths losing exp(-Σ μ l) along their legs to 1e-9; the late rays losing what a slab absorbs as Beer-Lambert has it (within 0.3 %), and upright stems meeting sound by its horizontal share (sound going up through a layer of them barely touches them); two bands traced with one set of rays as each traced alone (within 3 %); the rain of foliage and of stems against the detector spheres; and scenes without vegetation unchanged; leaves given by their area and width against a thin disc's closed forms (the dipole of a small one to 0.2 %, twice the mean projected area of a large one, its mean cosine towards 1/2) and a rigid wall's boundary-layer absorption (to 1e-12), per band as the mean over its frequencies, wider leaves of the same area taking more where their size nears the wavelength, and the early paths through a belt of them losing what the medium says (within 5 %);
- the wave solver in wind and temperature: the grid's own speed of sound in every cell giving the run without a speed per cell, to the bit (both stencils, with walls of frequency-dependent admittance); 0.8 times it giving sound that comes 1.25 times as late and as loud, its velocity as large next to its pressure; sound bent by a speed of sound that grows or falls by 65 m/s over 60 m arriving from 5 degrees above or below, within 12 % of the eigenrays' angle; and the matched layer stable where the speed of sound halves with height and jumps across the grid;
- sweep measurements of synthetic rooms through a distorting loudspeaker, with noise and interface latency: from 250 Hz up, the room's response comes back with errors 30 dB below it and T30 within 3 %; with a noise floor 55 dB down, T30 stays within 5 % of the noise-free value.
- outdoor results: the level relative to free field over grass against the exact spherical-wave reflection (within 0.05 dB from 1 kHz up, 0.75 dB in the band-filtered dip at 500 Hz), and over an absorbing ground that of free field; a barrier's insertion loss the level relative to free field with its sign turned, with a reflection back on the source's side that of its image source; the two runs sharing their seed, and maps giving the same at their points;
- a terrain: heights read between the grid's nodes bilinearly and past its edge as the edge's, from ESRI ASCII grids (north row first) and XYZ points in any order; a flat terrain giving the flat ground's early paths to the bit and a plane slope the reflection off the image source in it; the sound over a ridge the exact wedge's (UTD against the Biot-Tolstoy-Medwin solution, within 0.1 dB over the source's and the listener's slopes); a smooth hill's crest bending the sound, the level continuous as a listener climbs out of its shadow, over two hills each crest bending it in turn, a plane slope screening nothing and a building on the string stopping the path; buildings, barriers and vegetation standing on the terrain, zones taking its triangles, the extent keeping to it, coverage maps and insertion losses following it, footprints read from GeoJSON polygons and multipolygons, their holes built over and said so, and `inspect` describing the terrain and the heights over the ground.

Cross-check against pyroomacoustics

pyroomacoustics 0.10.1 also simulates rooms with image sources for the early part and stochastic ray tracing for the rest, which makes it the open-source program closest to this one. Both simulated the same four rooms:

- **specular**: an 8 x 6 x 3 m box with painted concrete walls, a carpeted floor and an absorbing tile ceiling, every surface reflecting specularly (no scattering);
- **diffuse**: the same box with every surface fully diffusing (scattering 1);
- **example**: the same box with the example scene's scattering: walls 0.05, floor 0.1, ceiling 0.2;
- **hall**: the fan-shaped lecture hall of `examples/lecture_hall` (1008 m³) with an audience floor (scattering 0.7), plasterboard ceiling, wooden stage wall and concrete side and back walls (scattering 0.1 to 0.15).

Both programs got the same absorption and scattering per octave band, the same air absorption (ISO 9613-1 at 20 °C and 50 %, from this package), image sources up to the 3rd order and 100,000 rays, with one omnidirectional source and four listeners per room. pyroomacoustics draws its rays afresh on every run, so each room ran five times in it: the tables give the mean of the five and the spread between the highest and the lowest. Both scatter vector-based. pyroomacoustics has no scattering from the size of a surface, so this package left it out for the comparison (`from_size: false`) and ran each room once more with its default, which adds it. Both sets of impulse responses were analysed by the same code (`acoustic_spaces.parameters.from_impulse_response`), and EDT, T30 and C80 were averaged over the four listeners, in the octave bands from 125 Hz to 4 kHz.

How the two programs differ

Reading the ray tracer of pyroomacoustics 0.10.1 (`libroom_src/room.cpp`) shows two differences besides the scattering of a surface's size:

- **How the listener receives scattered energy.** Both programs send the scattered share of every reflection straight to each listener that can see the reflection point ("diffuse rain"). Here the detector sphere then counts only the rest of a ray's energy. pyroomacoustics counts all of it at the sphere, so the scattered share reaches the listener twice, which raises the reverberant level, lengthens EDT and lowers C80.
- **One direction for all bands.** pyroomacoustics turns a ray by the mean of the bands' scattering coefficients; this package traces each band with rays of its own where the coefficients differ. In these four rooms they are the same in every band, so this makes no difference here.

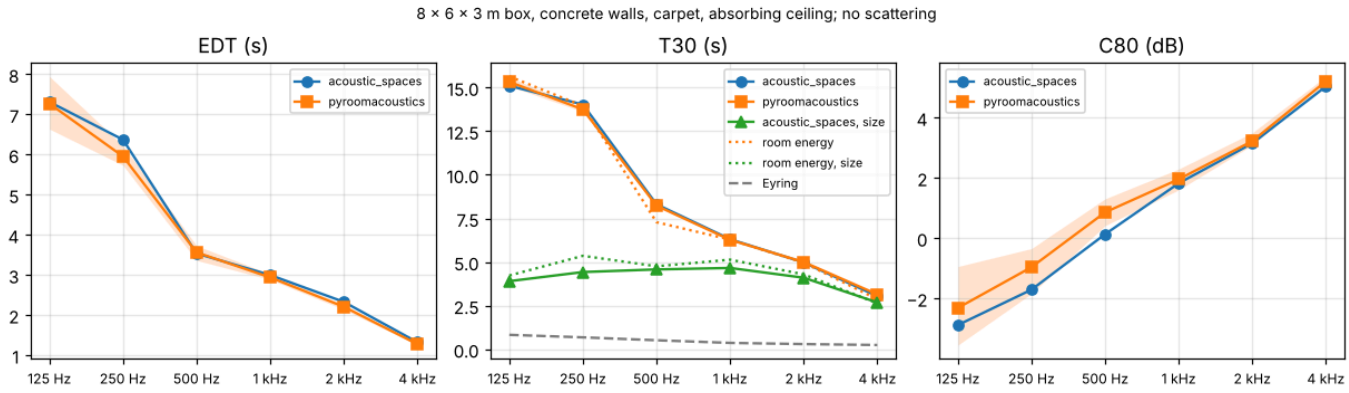
The cross-check script also traces the total sound energy in each room without any listener, with either model, and fits T30 to the same stretch of its decay as the listeners' T30 ("room energy" rows below).

Results

The figures show EDT, T30 and C80 per band for both programs, the spread of pyroomacoustics' runs shaded, this package's default with the scattering of the surfaces' size, the decay of the room's total energy under each model and, where a diffuse field makes them apply, Eyring's reverberation time and Barron's revised theory.

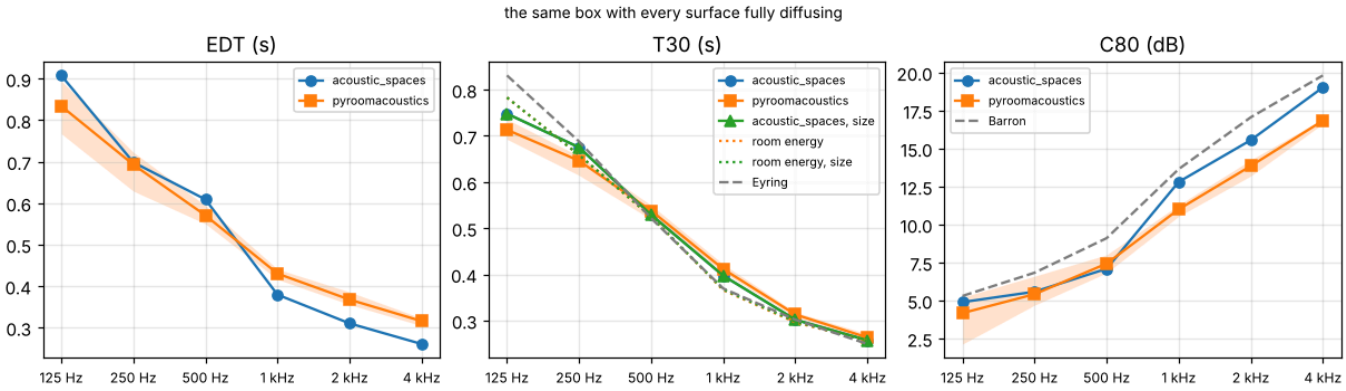
Specular box. The programs agree: T30 within 3 %, EDT within 7 % and C80 within 1 dB. Rays that travel parallel to the floor never meet the absorbing ceiling, so the decay is many times longer than Eyring's estimate in both. Even a smooth wall is only 3 m high, though, and scatters what reaches it from far away or at a grazing angle: with that, T30 comes out 2.7 to 4.7 s instead of 3 to 15 s.

	125 Hz	250 Hz	500 Hz	1 kHz	2 kHz	4 kHz
EDT (s), acoustic_spaces	7.31	6.37	3.54	3.00	2.33	1.33
EDT (s), pyroomacoustics	7.27	5.95	3.56	2.94	2.21	1.28
EDT (s), pyroomacoustics' spread over 5 runs	1.31	0.49	0.36	0.14	0.13	0.03
EDT difference	-1 %	-7 %	+1 %	-2 %	-5 %	-4 %
T30 (s), acoustic_spaces	15.10	14.03	8.31	6.31	4.98	3.05
T30 (s), pyroomacoustics	15.32	13.75	8.23	6.27	5.00	3.15
T30 (s), pyroomacoustics' spread over 5 runs	0.51	0.21	0.11	0.07	0.04	0.04
T30 difference	+1 %	-2 %	-1 %	-1 %	+1 %	+3 %
C80 (dB), acoustic_spaces	-2.9	-1.7	0.1	1.8	3.1	5.0
C80 (dB), pyroomacoustics	-2.3	-1.0	0.9	2.0	3.2	5.2
C80 (dB), pyroomacoustics' spread over 5 runs	2.6	1.5	0.9	0.7	0.4	0.2
C80 difference	+0.6 dB	+0.7 dB	+0.7 dB	+0.1 dB	+0.1 dB	+0.2 dB
T30 (s), acoustic_spaces with surface size (default)	3.91	4.43	4.58	4.66	4.10	2.69
T30 (s) of the room's energy, vector-based scattering	15.64	13.95	7.29	6.29	4.96	2.87
T30 (s) of the room's energy, with surface size	4.23	5.36	4.76	5.14	4.29	2.64
Eyring (s)	0.83	0.69	0.52	0.37	0.30	0.25



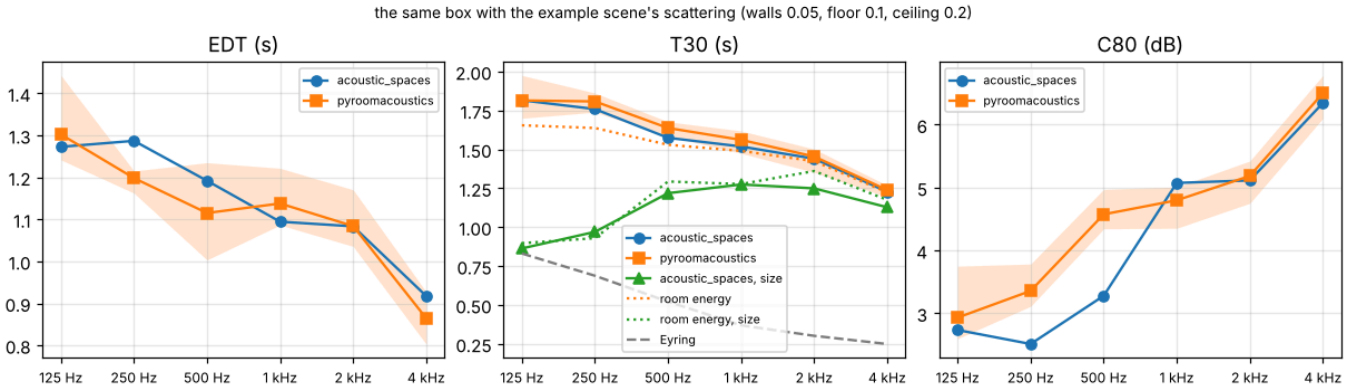
Diffuse box. T30 agrees within 5 %; this package stays within 8 % of the room's energy decay and within 10 % of Eyring's estimate, pyroomacoustics within 11 and 15 %. From 1 kHz up, pyroomacoustics gives 13 to 21 % longer EDT and 1.7 to 2.2 dB lower C80 than this package, as its double-counted scattered energy predicts. Barron's revised theory sides with this package: computed from the simulated energies, as `simulate` prints them, C80 is 0.6 to 1.0 dB below the theory in every band, where pyroomacoustics' (from its impulse responses) is 1.1 to 3.2 dB below it. From the impulse responses this package's is about 0.5 and 1.2 dB lower again at 250 and 500 Hz (2.0 dB below the theory at 500 Hz), because the direct sound and the first floor and ceiling reflections interfere within those octave bands. Surfaces that diffuse fully scatter no more for their size, so the default gives the same.

	125 Hz	250 Hz	500 Hz	1 kHz	2 kHz	4 kHz
EDT (s), acoustic_spaces	0.91	0.70	0.61	0.38	0.31	0.26
EDT (s), pyroomacoustics	0.83	0.69	0.57	0.43	0.37	0.32
EDT (s), pyroomacoustics' spread over 5 runs	0.13	0.09	0.04	0.02	0.03	0.02
EDT difference	-8 %	-1 %	-6 %	+13 %	+18 %	+21 %
T30 (s), acoustic_spaces	0.75	0.67	0.53	0.40	0.30	0.26
T30 (s), pyroomacoustics	0.71	0.65	0.54	0.41	0.31	0.26
T30 (s), pyroomacoustics' spread over 5 runs	0.04	0.04	0.03	0.02	0.01	0.01
T30 difference	-5 %	-4 %	+2 %	+4 %	+4 %	+2 %
C80 (dB), acoustic_spaces	4.9	5.6	7.1	12.8	15.6	19.0
C80 (dB), pyroomacoustics	4.2	5.4	7.4	11.0	13.9	16.8
C80 (dB), pyroomacoustics' spread over 5 runs	3.1	1.9	1.2	0.7	1.0	0.3
C80 difference	-0.7 dB	-0.2 dB	+0.3 dB	-1.8 dB	-1.7 dB	-2.2 dB
C80 (dB), Barron's revised theory	5.3	6.8	9.1	13.6	17.1	19.8
T30 (s), acoustic_spaces with surface size (default)	0.75	0.67	0.53	0.40	0.30	0.26
T30 (s) of the room's energy, vector-based scattering	0.78	0.66	0.52	0.37	0.30	0.26
T30 (s) of the room's energy, with surface size	0.78	0.66	0.52	0.37	0.30	0.26
Eyring (s)	0.83	0.69	0.52	0.37	0.30	0.25



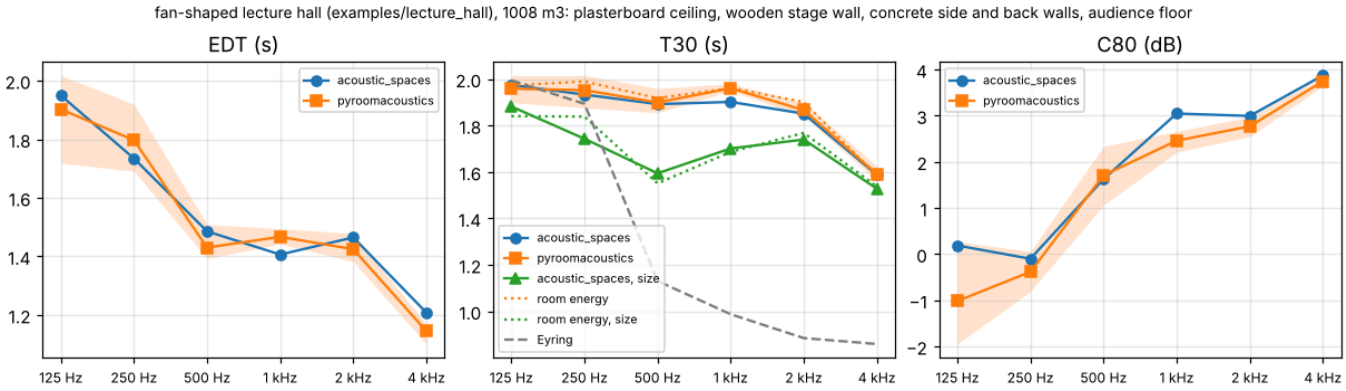
Example box. With smooth walls (scattering 0.05), T30 agrees within 4 %, EDT within 7 % and C80 within 1.3 dB. Up to version 2 this package scattered a ray with the coefficient as probability, and pyroomacoustics' T30 came out 19 to 42 % shorter. The walls' size halves T30 at 125 and 250 Hz (0.86 s against 1.82 s) and shortens it by 8 to 22 % from 500 Hz up; each run matches the room-energy decay of its model within 11 %.

	125 Hz	250 Hz	500 Hz	1 kHz	2 kHz	4 kHz
EDT (s), acoustic_spaces	1.27	1.29	1.19	1.09	1.08	0.92
EDT (s), pyroomacoustics	1.30	1.20	1.12	1.14	1.08	0.86
EDT (s), pyroomacoustics' spread over 5 runs	0.20	0.05	0.23	0.13	0.13	0.12
EDT difference	+2 %	-7 %	-6 %	+4 %	+0 %	-6 %
T30 (s), acoustic_spaces	1.82	1.76	1.57	1.52	1.44	1.23
T30 (s), pyroomacoustics	1.81	1.81	1.64	1.56	1.45	1.24
T30 (s), pyroomacoustics' spread over 5 runs	0.27	0.12	0.09	0.14	0.15	0.07
T30 difference	+0 %	+3 %	+4 %	+3 %	+1 %	+1 %
C80 (dB), acoustic_spaces	2.7	2.5	3.3	5.1	5.1	6.3
C80 (dB), pyroomacoustics	2.9	3.4	4.6	4.8	5.2	6.5
C80 (dB), pyroomacoustics' spread over 5 runs	1.2	0.7	0.6	0.7	0.7	0.7
C80 difference	+0.2 dB	+0.8 dB	+1.3 dB	-0.3 dB	+0.1 dB	+0.2 dB
T30 (s), acoustic_spaces with surface size (default)	0.86	0.97	1.22	1.27	1.25	1.13
T30 (s) of the room's energy, vector-based scattering	1.65	1.64	1.53	1.49	1.42	1.22
T30 (s) of the room's energy, with surface size	0.90	0.93	1.29	1.28	1.36	1.18
Eyring (s)	0.83	0.69	0.52	0.37	0.30	0.25



Lecture hall. T30 agrees within 3 % and EDT within 5 %; pyroomacoustics' C80 is up to 1.2 dB lower (at 125 Hz), the double-counted scattered energy again. The surfaces' size shortens T30 by 4 to 16 %, in line with the room-energy decay of that model within 6 %.

	125 Hz	250 Hz	500 Hz	1 kHz	2 kHz	4 kHz
EDT (s), acoustic_spaces	1.95	1.73	1.49	1.41	1.46	1.21
EDT (s), pyroomacoustics	1.90	1.80	1.43	1.47	1.42	1.15
EDT (s), pyroomacoustics' spread over 5 runs	0.30	0.23	0.12	0.05	0.10	0.07
EDT difference	-2 %	+4 %	-4 %	+4 %	-3 %	-5 %
T30 (s), acoustic_spaces	1.97	1.93	1.89	1.90	1.85	1.59
T30 (s), pyroomacoustics	1.96	1.95	1.90	1.96	1.87	1.59
T30 (s), pyroomacoustics' spread over 5 runs	0.12	0.14	0.10	0.03	0.06	0.06
T30 difference	-1 %	+1 %	+0 %	+3 %	+1 %	+0 %
C80 (dB), acoustic_spaces	0.2	-0.1	1.6	3.0	3.0	3.9
C80 (dB), pyroomacoustics	-1.0	-0.4	1.7	2.5	2.8	3.7
C80 (dB), pyroomacoustics' spread over 5 runs	2.2	0.9	1.3	0.4	0.4	0.2
C80 difference	-1.2 dB	-0.3 dB	+0.1 dB	-0.6 dB	-0.2 dB	-0.1 dB
T30 (s), acoustic_spaces with surface size (default)	1.88	1.74	1.59	1.70	1.74	1.53
T30 (s) of the room's energy, vector-based scattering	1.97	1.99	1.92	1.96	1.90	1.58
T30 (s) of the room's energy, with surface size	1.84	1.84	1.55	1.69	1.77	1.54
Eyring (s)	1.99	1.89	1.13	0.99	0.88	0.86



Conclusions

- With the same scattering model, the programs agree within 5 % on T30 in all four rooms, from purely specular to fully diffusing surfaces, and within 8 % on EDT where pyroomacoustics' double counting does not lengthen it. That is close to the 5 % and 1 dB the plan aimed for. With 100,000 rays, pyroomacoustics' own runs differ by up to 15 % in T30, 21 % in EDT and 3.1 dB in C80, most often at 125 Hz, which is why the tables average five of them; this package's numbers move by a few percent with the seed.
- Where the theory is clear (the diffuse box), this package agrees with Eyring (T30 within 10 %) and with Barron's revised theory (C80 from the energies within 1.0 dB); pyroomacoustics counts the scattered energy twice.
- The scattering of a surface's size, which pyroomacoustics lacks, shortens the decay of rooms with smooth walls a lot at low frequencies and less at high ones. Measurements, not another program, decide whether it helps: in the two measured rooms below it does.

Measured rooms (BRAS)

The *Benchmark for Room Acoustical Simulation* (BRAS; Aspöck, Brinkmann, Ackermann, Weinzierl and Vorländer, 2020, DOI 10.14279/depositonce-6726.3, CC BY-SA 4.0) measured real rooms and published their models, the absorption and scattering of their surfaces, and the impulse responses. Four of its rooms are in `validation/bras` as scenes (its README says how they were made):

- **CR1**, two coupled rooms, a laboratory of 102 m³ and a reverberation chamber of 121 m³, with the door between them open by 4.1° and, a scene of its own, by 30.4°, four materials. In BRAS's model with the door at 30.4° the door leaf runs 24 mm into the wall at its hinge, which leaves the model without an inside; the conversion joins the leaf to the wall and the floor;
- **CR2**, a seminar room of 146 m³ with five surface materials;
- **CR3**, a chamber music hall of 3331 m³ with upholstered seating, stage panels and structured plaster, seven materials;
- **CR4**, an auditorium of 8657 m³ with eight materials.

In CR2, CR3 and CR4 a dodecahedron loudspeaker stood at two positions and an omnidirectional microphone at five, giving ten measured responses in each. In CR1 the loudspeaker stood once in each room (LS1 in the chamber, LS2 in the laboratory) and the microphone at two positions in the laboratory, giving four. Each scene was simulated as it stands, with this package's defaults: ray tracing up to the 3rd order, first-order edge diffraction, 50,000 late rays per band and vector-based scattering with the scattering of each surface's size, in the air measured on the day. Simulated and measured parameters were compared pair by pair in just noticeable differences (`acoustic_spaces.compare`, the same JNDs as `acoustic-spaces compare`) and averaged over the pairs, in CR1 over each source's. The measured responses were analysed by this package too. Its octave values, averaged over the ten pairs, are within 3 % of the mean of the three third-octave averages BRAS publishes for T20, within 5 % for EDT and within 0.8 dB for C80, from 500 Hz to 4 kHz in CR2, CR3 and CR4 (the largest differences are EDT 4.3 % lower in the hall at 500 Hz and C80 0.8 dB higher in the auditorium at 500 Hz and 1 kHz; elsewhere within 3.8 % and 0.6 dB). BRAS publishes none for CR1. Below 500 Hz the octave holds three third-octave bands whose values differ a lot, so averaging them is not the same analysis. G is compared from seat to seat, as the level of the measured responses is uncertain by about 3 dB (see Strength below).

The tables and plots in this section were made with this version on 29 and 30 October 2026 (the coupled rooms with the door at 30.4° on 2 November), on 4 cores of a cloud machine without a GPU, where the run times below were measured with each run alone on the machine (the coupled rooms', the seminar room's and the auditorium's in a second run on 30 October, which gave the same numbers). The coupled rooms, the seminar room and the auditorium each run in 2 to 4 minutes and 0.7 to 1.1 GiB (4 to 9 minutes calibrated, 5 to 11 minutes and up to 1.2 GiB in third-octave bands); the hall takes 1.5 hours and 10.0 GiB (5.2 hours and 10.5 GiB calibrated, 3.4 hours and 11.8 GiB in third-octave bands), nearly all of it the early stage's edge diffraction at one listener (see Memory). 4.0.0 could not run the seminar room or the hall at their defaults: its early stage asked for 16.4 GiB at once in CR3 and 19.6 GiB in CR2.

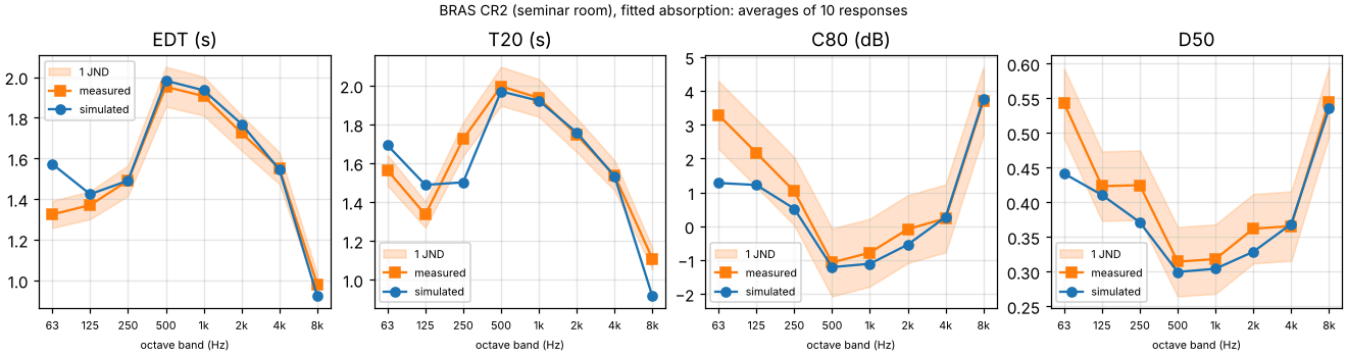
BRAS gives two sets of absorption coefficients. The **fitted** set was adjusted until Eyring's formula gave the measured reverberation time (in CR1 for each of the two rooms on its own); the scenes use it. The **initial** set is BRAS's estimate before that, from tables and measurements of the materials, which is the blind case (`--materials initial`).

Mean absolute difference from 125 Hz to 4 kHz in JNDs (for G in dB, the JND being 1 dB), and how many of the pair and band values are within one JND: of 60 in CR2, CR3 and CR4, of 24 in CR1 at each angle of the door (23 for T30 at 4.1°, which one of LS1's responses does not decay far enough for at 125 Hz). With the fitted absorption:

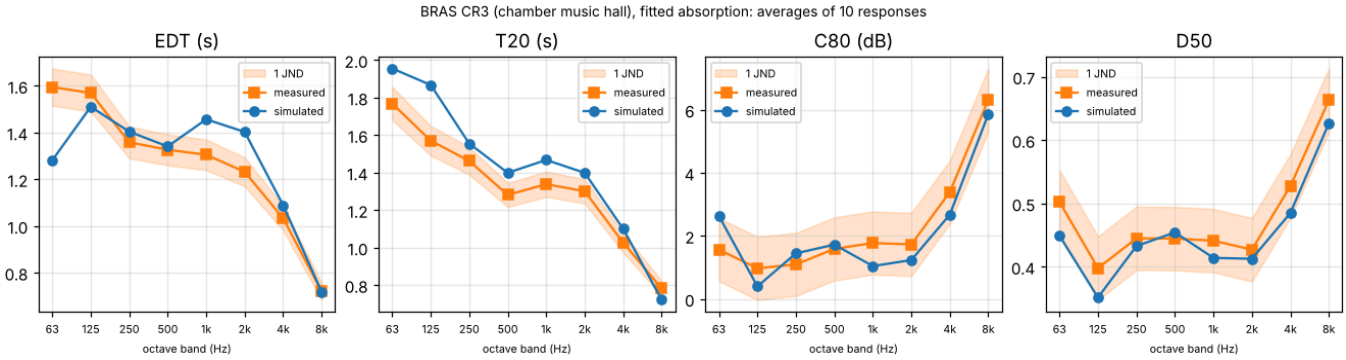
	CR1, 4.1°	CR1, 30.4°	CR2	CR3	CR4
EDT	7.8 (2)	7.6 (1)	1.0 (44)	2.0 (20)	3.7 (3)
T20	6.7 (1)	4.6 (2)	1.0 (41)	2.0 (9)	3.0 (1)
T30	7.7 (0)	3.8 (5)	1.0 (43)	1.8 (9)	3.0 (0)
C50	4.3 (1)	2.0 (6)	1.0 (41)	1.2 (34)	1.3 (29)
C80	4.0 (1)	2.5 (5)	0.7 (52)	1.0 (40)	1.3 (26)
D50	4.1 (1)	2.0 (5)	1.0 (39)	1.3 (31)	1.3 (28)
Ts	3.6 (2)	5.0 (0)	0.8 (41)	1.0 (33)	2.5 (9)
G, seat to seat	1.8 (5)	0.6 (20)	0.3 (53)	0.6 (52)	0.5 (50)

With the initial absorption:

	CR1, 4.1°	CR1, 30.4°	CR2	CR3	CR4
EDT	16.3 (0)	13.1 (1)	5.9 (6)	4.8 (8)	4.0 (8)
T20	11.0 (1)	6.3 (0)	6.5 (1)	5.8 (12)	3.7 (13)
T30	5.6 (4)	8.1 (1)	6.5 (0)	5.7 (11)	3.8 (10)
C50	6.1 (3)	2.9 (7)	1.6 (24)	1.5 (23)	1.6 (24)
C80	6.0 (4)	3.5 (5)	1.7 (28)	1.4 (26)	1.5 (27)
D50	5.9 (3)	2.9 (6)	1.7 (23)	1.6 (21)	1.7 (24)
Ts	12.5 (4)	5.7 (4)	3.3 (14)	2.2 (22)	2.8 (16)
G, seat to seat	2.9 (2)	1.2 (10)	0.3 (54)	0.6 (51)	0.5 (50)

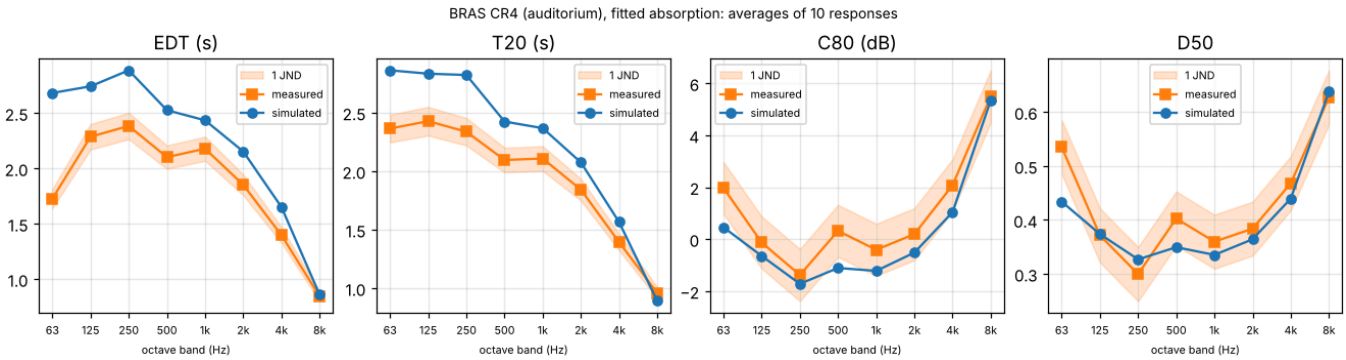


Seminar room, fitted absorption. From 500 Hz to 4 kHz the reverberation time is within 2 % of the measured one (T20 1.92 s against 1.94 s at 1 kHz), well within one JND, and C50, C80 and D50 are within one JND from 500 Hz up. At 250 Hz and 8 kHz the simulated decay is 13 and 18 % short, and so is Eyring's formula with the same absorption and air (1.48 s against 1.72 s measured at 250 Hz, 0.93 s against 1.11 s at 8 kHz), so there the fitted coefficients, averaged onto octaves, do not give the measured decay even in a diffuse field. At 8 kHz they assume less air absorption than ISO 9613-1 gives at the octave's centre frequency; a measured decay follows the octave's slowest part, its lowest third (6.3 kHz). Simulated in third-octave bands, which keep both apart, the two bands come within 4.1 % of the measurement (see the third-octave results below).



Chamber music hall, fitted absorption. The reverberation time comes out 6 to 10 % long from 250 Hz to 4 kHz (T20 1.47 s against 1.34 s at 1 kHz), 1.3 to 1.9 JNDs, and 11 and 19 % long at 63 and 125 Hz. EDT is within 3.8 % at 125, 250 and 500 Hz and 5 to 14 % long from 1 to 4 kHz, but 20 % short at 63 Hz, where the early part is stronger than measured (see the next paragraph). From 125 Hz up, C50, C80 and D50 are within about one JND, mostly slightly low, as a slightly long decay predicts. Eyring's formula with the same absorption and air is within 5.1 % of the measurement from 63 Hz to 4 kHz, as the fitting intends. That the model decays more slowly suggests that in it sound reaches the absorbing seating a little less often than Eyring's diffuse field assumes; why the real hall decays as Eyring's formula gives and the model does not is not traced (the simplified BRAS model, which smooths the hall's details away, is one candidate).

The hall's early part at low frequencies. At 125 Hz C80 is 0.6 dB low and EDT 3.8 % short; at 63 Hz the early part is too strong (C80 1.1 dB high, EDT 3.6 JNDs short).



Auditorium, fitted absorption. The decay comes out 12 to 21 % long from 125 Hz to 4 kHz (T20 2.37 s against 2.11 s at 1 kHz), 2.5 to 4.2 JNDs, and EDT 12 to 21 % long too. Eyring's formula with the same absorption and air is within 5 % of the measured T20 from 63 Hz to 4 kHz, so here too the fitting did what it intends; that the model decays more slowly still suggests that its sound reaches the absorbing surfaces less often than Eyring's diffuse field assumes, more so than in the hall (see "Calibrated absorption" below). C50, C80 and D50 are within about one JND from 125 Hz up (C80 1.4 JNDs low at 500 Hz), and like the hall's mostly low. At 8 kHz the decay is 7 % short, as in the seminar room: Eyring's formula with the same absorption and air gives 0.82 s against 0.96 s measured.

What scattering does. Up to version 2 this package scattered a ray with the coefficient as probability, and from the surface's own coefficient alone. The hall then came out 20 to 32 % long, EDT 11 to 24 %, and the seminar room 0 to 10 % long from 250 Hz to 4 kHz. Tracing the total sound energy in each room without listeners, as the cross-check above does, and fitting T20 to its decay shows what each part of the new model does (`validation/bras_room_energy.py`: 20,000 rays from LS1, seed 0). The two vector-based models trace each band with rays of its own; random scattering, as version 2 did, traces one ray for all bands and re-weights each band by its own coefficient, which makes it noisier: with another seed its row moves by up to 8 % (the hall at 63 Hz), the others by up to 1.3 %.

T20 (s)	63 Hz	125 Hz	250 Hz	500 Hz	1 kHz	2 kHz	4 kHz	8 kHz
CR2 measured	1.56	1.34	1.72	2.00	1.94	1.75	1.54	1.11
CR2 room energy, random scattering (version 2)	1.51	1.49	1.70	2.17	2.06	1.83	1.60	0.97
CR2 room energy, vector-based scattering	1.57	1.41	1.51	1.82	1.79	1.66	1.44	0.90
CR2 room energy, vector-based with surface size (default)	1.69	1.49	1.50	1.97	1.92	1.76	1.53	0.91
CR2 Eyring	1.45	1.34	1.48	2.00	1.95	1.76	1.59	0.93
CR3 measured	1.77	1.57	1.46	1.28	1.34	1.30	1.03	0.78
CR3 room energy, random scattering (version 2)	2.11	2.01	1.77	1.60	1.63	1.56	1.18	0.74
CR3 room energy, vector-based scattering	2.11	1.95	1.69	1.51	1.52	1.43	1.11	0.74
CR3 room energy, vector-based with surface size (default)	2.03	1.89	1.63	1.40	1.47	1.40	1.12	0.74
CR3 Eyring	1.79	1.65	1.49	1.28	1.33	1.33	1.06	0.71

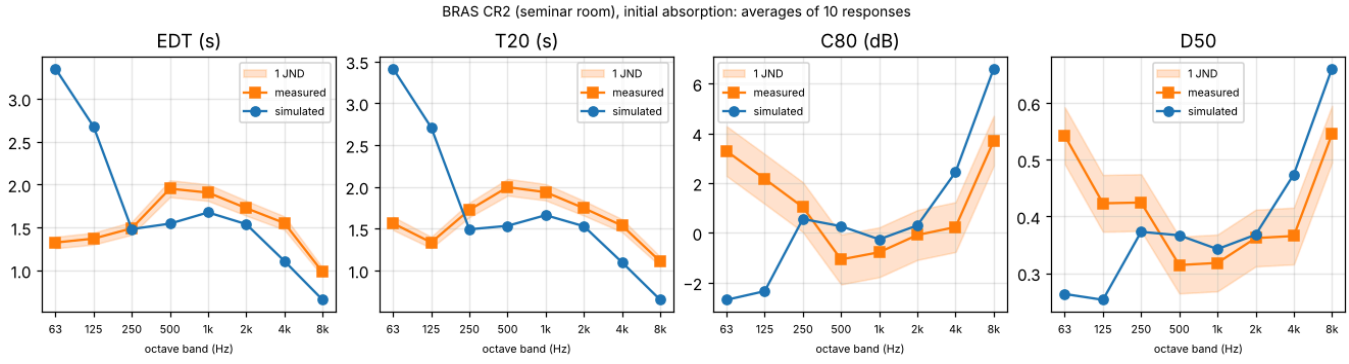
From 500 Hz to 4 kHz, vector-based scattering alone takes the hall from 15 to 25 % long down to 8 to 18 %, but makes the seminar room 5 to 9 % short. Adding the scattering of each surface's size brings the seminar room within 2 % of the measurement (and within 4

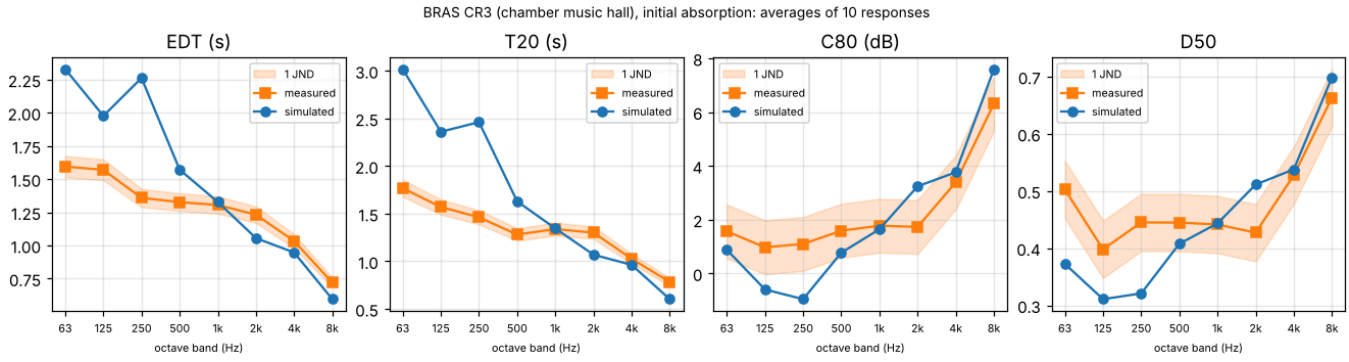
rooms better than either did. The simulated responses decay like these room-energy curves, within 0.2 % in the seminar room and 4.8 % in the hall in every band. The rest of the hall's gap is not traced further (see the hall's fitted absorption above). Fitting the absorption with this tracer instead of Eyring's formula takes up that gap, in the hall and in the auditorium (see "Calibrated absorption" below).

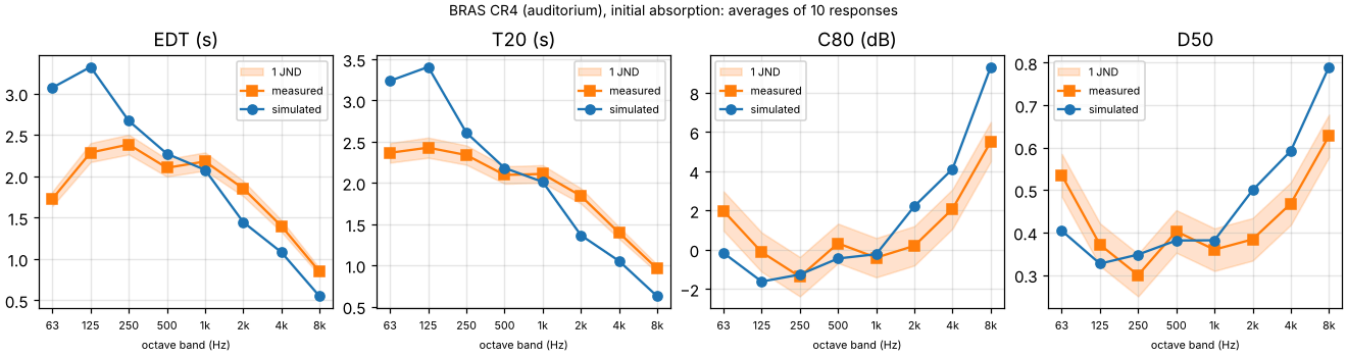
Third-octave bands. BRAS gives its coefficients in third octaves. Averaged onto octaves, an octave's three thirds decay as one, at their mean absorption, while three decays at different rates decay together more slowly than that, as the measured octave does. Simulated with `simulation.bands: third_octave`, which keeps the thirds apart (`bras_validation.py --bands third_octave`), T20 in each octave band:

T20 (s)	63 Hz	125 Hz	250 Hz	500 Hz	1 kHz	2 kHz	4 kHz	8 kHz
CR2 measured	1.56	1.34	1.72	2.00	1.94	1.75	1.54	1.11
CR2 octave bands (default)	1.69	1.49	1.50	1.97	1.92	1.76	1.53	0.91
CR2 third-octave bands	1.77	1.51	1.65	1.98	1.92	1.78	1.51	1.07
CR3 measured	1.77	1.57	1.46	1.28	1.34	1.30	1.03	0.78
CR3 octave bands (default)	1.96	1.87	1.55	1.40	1.47	1.40	1.10	0.72
CR3 third-octave bands	2.00	1.90	1.57	1.40	1.46	1.40	1.13	0.80
CR4 measured	2.36	2.43	2.34	2.10	2.11	1.84	1.39	0.96
CR4 octave bands (default)	2.86	2.83	2.82	2.42	2.37	2.07	1.57	0.89
CR4 third-octave bands	2.96	2.92	2.87	2.45	2.38	2.07	1.63	1.03

The seminar room's decay at 250 Hz and 8 kHz, 13 and 18 % short in octave bands, comes within 4.1 % of the measurement: there its thirds' absorption, and at 8 kHz the air's, differ most. The hall's 8 kHz goes from 8 % short to 2 % long, and the auditorium's from 7 % short to 7 % long. From 500 Hz to 2 kHz T20 changes by 0.03 s at most. At 63 and 125 Hz the rooms come out 1 to 5 % longer than in octave bands, and the hall at 4 kHz 10 % long instead of 7 %. Over 125 Hz to 4 kHz the seminar room's mean T30 difference goes from 1.0 to 0.9 JNDs; the hall's goes from 1.8 to 2.1 and the auditorium's from 3.0 to 3.5, since their decays were long already and third octaves lengthen them. In the coupled rooms with the door at 4.1° they lengthen the decay of LS1's sound further (T20 13.8 s at 125 Hz against 12.1 s in octave bands and 6.7 s measured). A third-octave run takes 10 and 11 minutes for the seminar room and the auditorium instead of 4, and 3.4 hours for the hall instead of 1.5.



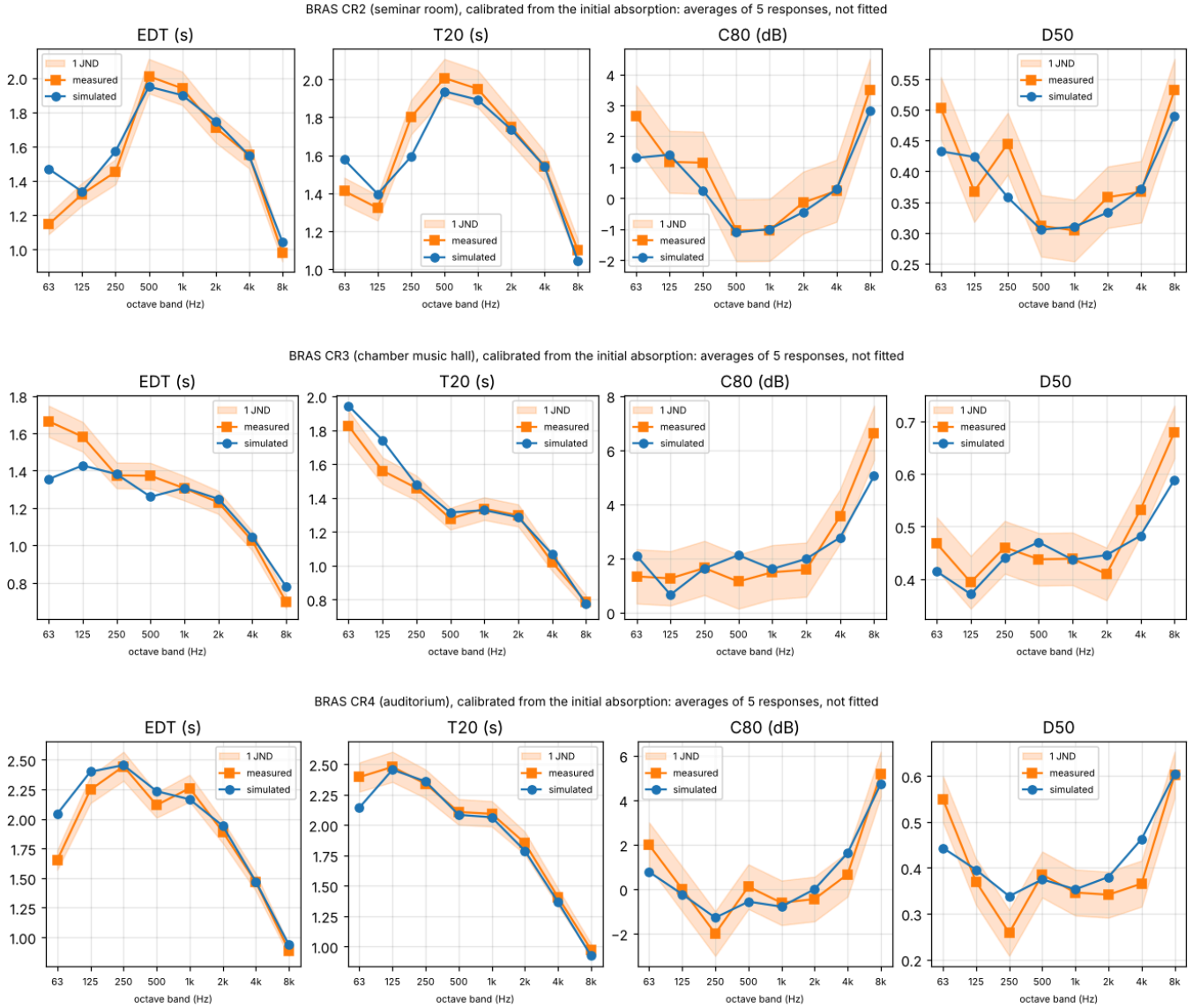




Initial absorption. With the absorption estimated before fitting, T20 at 63 and 125 Hz comes out 1.5 to 2.2 times as long as measured in the seminar room and the hall, and at 250 Hz in the hall too (2.71 s against 1.34 s in CR2 at 125 Hz), and 1.4 times in the auditorium. The initial coefficients have far less absorption there than the fitted ones. In real rooms much of it comes from structures that flex, such as panels, windows and doors, which tables of materials rarely capture. From 500 Hz to 4 kHz the seminar room's decay is 12 to 29 % short, and at 2 and 4 kHz the hall's is 6 to 18 % short and the auditorium's 26 and 25 %. The simulation follows Eyring's formula with the same coefficients within 15 % in every band of the seminar room and the hall (within 3 % from 250 Hz up in the seminar room), and is 15 to 22 % longer in every band of the auditorium, as with the fitted set, so the error is in the coefficients. This is the error to expect from tabulated coefficients alone, before a room has been measured.

Calibrated absorption. `--calibrate LS1` first fits the absorption of every surface (`acoustic-spaces calibrate`, see Measurements). It fits to the T30, EDT and C80 of the five responses of the first loudspeaker position, starting from the initial set, with 10,000 rays per band while fitting. Then it simulates the room with what the fit found, as above, and compares the five responses of the second position, which the fit did not see. The table gives the mean absolute difference from 125 Hz to 4 kHz in JNDs over those five responses, with how many of their 30 pair and band values are within one JND. The fitted and initial columns repeat the tables above, which average all ten responses:

	CR2 calibrated	CR2 fitted	CR2 initial	CR3 calibrated	CR3 fitted	CR3 initial	CR4 calibrated	CR4 fitted	CR4 initial
EDT	1.1 (17)	1.0	5.9	1.4 (14)	2.0	4.8	1.5 (12)	3.7	4.0
T20	0.9 (24)	1.0	6.5	1.0 (18)	2.0	5.8	0.7 (24)	3.0	3.7
T30	1.0 (22)	1.0	6.5	0.5 (26)	1.8	5.7	0.6 (25)	3.0	3.8
C50	1.1 (20)	1.0	1.6	1.2 (14)	1.2	1.5	1.4 (13)	1.3	1.6
C80	0.6 (26)	0.7	1.7	0.8 (19)	1.0	1.4	1.0 (16)	1.3	1.5
D50	1.1 (20)	1.0	1.7	1.3 (12)	1.3	1.6	1.4 (14)	1.3	1.7
Ts	0.7 (26)	0.8	3.3	0.8 (21)	1.0	2.2	1.2 (14)	2.5	2.8



The initial estimates made the decay up to 2.2 times as long. From them, calibration brings the seminar room to about one JND on responses it did not fit, as BRAS's fitted set does. In the hall and the auditorium it does better than the fitted set on the decay. In the hall T30 is within 0.5 JNDs on average, with 26 of the 30 values within one JND, against 1.8 with the fitted set, T20 within 1.0 against 2.0 and EDT 1.4 against 2.0. In the auditorium T20 and T30 are within 0.7 and 0.6 JNDs, with 24 and 25 of the 30 values within one JND, against 3.0 with the fitted set, and EDT 1.5 against 3.7. The fit uses the model's own rays rather than Eyring's formula, so it takes up what the model's sound misses of the absorbing surfaces: from 63 Hz to 4 kHz it found 3 to 23 % more absorption in all than BRAS's fitted set in the hall, and 18 to 30 % more in the auditorium. The clarity stays within 0.8 to 1.4 JNDs on average, as with the fitted set (1.0 to 1.3).

In the seminar room the fit found nearly BRAS's fitted set, though it started from the initial one. The windows absorb 0.24 and 0.31 at 63 and 125 Hz (BRAS fitted 0.25 and 0.19, initial 0.13 and 0.10), and every other value is within 0.022 of BRAS's. In the hall it found more absorption than BRAS in the plaster at 125 and 250 Hz (0.22 and 0.20 against 0.11 and 0.15) and in the seating at 63 Hz and 2 kHz (0.29 and 0.58 against 0.17 and 0.41), and less in the seating at 125 and 250 Hz (0.13 and 0.23 against 0.21 and 0.37) and in the structured plaster at 2 kHz (0.10 against 0.14). In the auditorium the linoleum absorbs 0.23 and 0.28 at 63 and 125 Hz (BRAS fitted 0.15 and 0.10), the brick wall 0.31 at 500 Hz (0.20) and the wood panels 0.27 at 4 kHz (0.12).

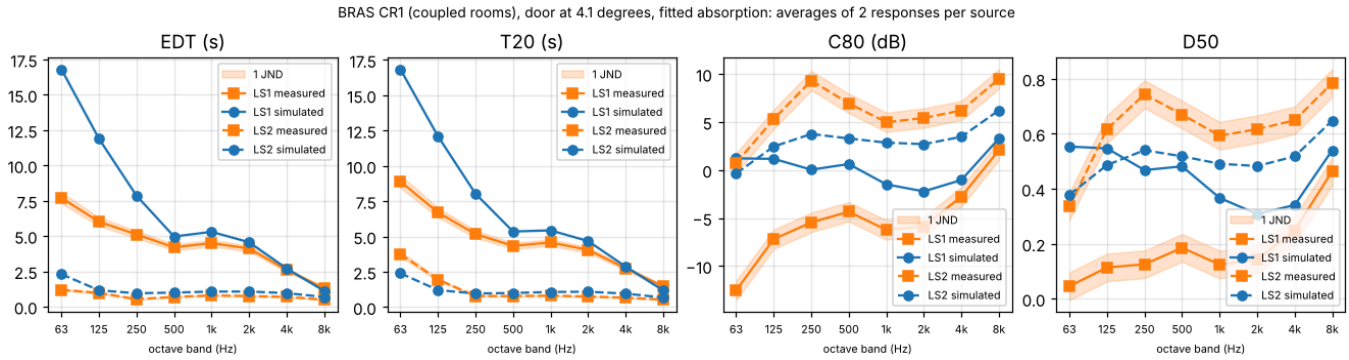
These are what a model needs to match the room, not what each surface absorbs. With one loudspeaker position and five microphones, the uncertainty the fit reports is above 0.2 for most materials and bands in the hall and the auditorium, and 0.49 or more for all of them in the seminar room. The measurements fix each band's total absorption, and the proportions between materials come from the initial estimates. Telling surfaces apart takes responses that hear them differently, such as sources and microphones near different surfaces.

Some differences remain:

- **250 Hz in the seminar room.** T30 is 3.1 JNDs short on the second loudspeaker's responses, though 1.2 off on average on the first's. The two positions measured 1.70 and 1.90 s on average. This is near the room's Schroeder frequency, about 230 Hz, where a decay depends on where it is measured.
- **8 kHz in the seminar room.** The decay stays 2 JNDs short even with surfaces that absorb next to nothing, because in octave bands the air absorbs at the octave's centre frequency (see "Third-octave bands" above).
- **63 Hz in the seminar room.** EDT stays 5.9 JNDs long. This is below the room's Schroeder frequency, where a geometric model gives an average rather than the room's modes: a likely cause, not traced.
- **63 and 125 Hz in the hall.** EDT is 3.7 and 1.6 JNDs short and T20 1.7 and 2.4 JNDs long. At 63 Hz the early part is stronger than measured, as with the fitted set (see "The hall's early part at low frequencies" above). The hall's Schroeder frequency is about 45 Hz.
- **63 Hz in the auditorium.** EDT is 5.1 JNDs long and T30 3.4 JNDs short.

Fitting took 5.3 minutes for the seminar room (three rounds of tracing at the first position's 10,000 rays per band), 4.4 minutes for the auditorium (four rounds) and 3.7 hours for the hall (four rounds), where nearly all of it is the early stage's edge diffraction, which fewer rays do not shorten. Simulating again took about 4 minutes in the seminar room and the auditorium and 89 minutes in the hall.

Coupled rooms

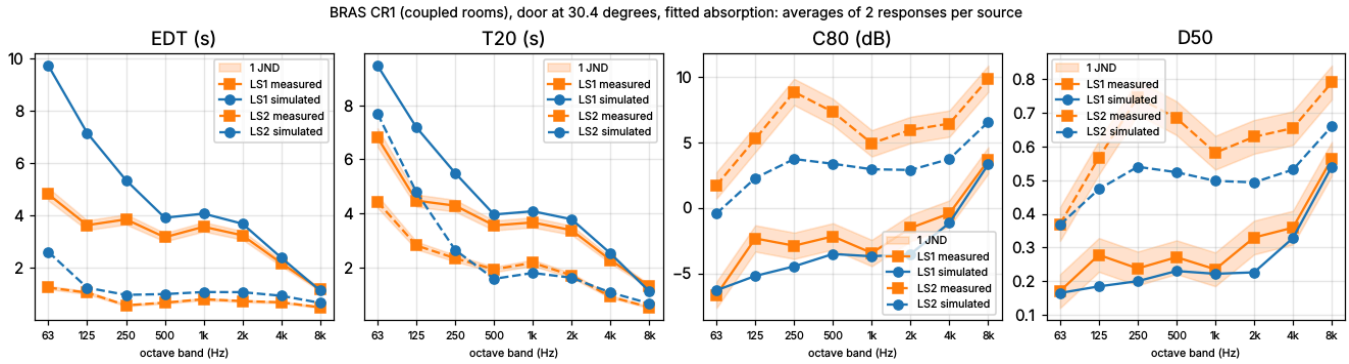


In CR1 the door, open by 4.1°, leaves the rooms joined by a slit: about 4 cm between the leaf's free end and the jamb, over the door's height, and a wedge above the leaf, about 0.1 m² in all. In the model, nothing else joins them. The scene's absorption is BRAS's fitted set, fitted to each room on its own, so the chamber alone decays in 12.4 s at 125 Hz and 5.5 s at 500 Hz by Eyring's formula, and the laboratory in 1.0 and 0.7 s (with each room's volume and surface as BRAS's material file gives them).

LS1's sound reaches the microphones in the laboratory from the chamber. Simulated, it decays in 12.1 s at 125 Hz and 5.4 s at 500 Hz (T20), within 5.3 % of what the diffuse theory of two coupled rooms gives from 125 Hz to 2 kHz with those Eyring times and an opening of 0.1 m² (11.7 and 5.4 s; `validation/coupled_rooms.py`). Near that opening the decay changes little with it, so the simulated decays correspond to openings from 0.04 to 0.33 m² in the same theory. The measured sound decays in 6.7 and 4.3 s: in the theory the rooms would need to be joined by 1.5 and 1.6 m² at 125 and 250 Hz and 1.0 m² from 500 Hz to 2 kHz, 10 to 16 times the model's opening. The model's door is a rigid slab, without the transmission BRAS gives no data for, and its rooms meet nowhere else, so what joins the real rooms that strongly is not in the model. LS1's EDT and T20 are 2 to 25 JNDs long from 63 Hz to 2 kHz, and C80 4 to 14 dB high.

LS2's sound decays in the laboratory, where the measured decays bend: their EDT is 0.80 s at 1 kHz and their T30 1.41 s, at 125 Hz 0.97 and 4.38 s, the chamber's sound coming back through the door. The simulated decays are single slopes, so T30 is 5 to 14 JNDs short from 125 Hz to 1 kHz. Their EDT is 1.1 s at 1 kHz, a third longer than the 0.8 s Eyring's formula gives for the laboratory with the fitted set, which is what was measured: as in the hall and the auditorium, the model decays more slowly than Eyring's diffuse field with the same absorption, here more so.

With the door at 4.1° and calibrated on LS1's responses from the initial set (`--calibrate LS1`), the fit raised the absorption of the chamber's painted concrete to 4.5 times BRAS's fitted value at 125 Hz and 1.2 to 1.7 times from 250 Hz to 4 kHz, which brings LS1's EDT within 16 % and its T20 within 11 % from 125 Hz to 4 kHz, and took the laboratory's absorber down from 0.49 to 0.20 at 1 kHz, as LS1's responses do not pin it (the fit reports an uncertainty of 0.33 to 6.7 for it). LS2's responses, which the fit did not see, then decay in 2.2 s at 1 kHz against 0.8 s measured, 55 JNDs off in EDT on average. A calibration fits the responses it is given; in coupled rooms each room needs responses of its own.



With the door open by 30.4°, the leaf stands in the chamber beside the doorway. LS1's sound crosses the doorway (2.9 m² beside the box on the leaf) and then goes round the leaf (about 2.0 m², between the box and the doorway's far side and over the leaf), and in the same theory two openings crossed in turn, with no absorption between them, act as one of 1.2 m². Simulated, the sound decays in 7.2 s at 125 Hz and 4.0 s at 500 Hz, 1.5 to 7.3 % faster than the theory with that opening from 125 Hz to 2 kHz (7.3 and 4.2 s): the simulated decays correspond to openings of 1.2 to 1.6 m². The measured sound decays in 4.5 and 3.6 s, so the decay comes much closer than with the door at 4.1°: T20 is 11 to 12 % long from 500 Hz to 4 kHz (2.1 to 2.4 JNDs), against 5 to 24 %, and 60 and 28 % long at 125 and 250 Hz, against 81 and 55 %; C80 is 0.3 to 2.9 dB low from 125 Hz to 4 kHz. In the theory the measured decays need openings of 2.2 to 2.5 m² from 250 Hz to 2 kHz, 0.7 to 1.2 m² more than the simulated ones, and with the door at 4.1° 0.7 to 1.5 m² more: what the model lacks changes little with the door's angle. At 125 Hz they need 3.6 m², more than the whole doorway (3.2 m²).

LS2’s measured decays bend more than with the door at 4.1° (EDT 0.80 s and T30 3.17 s at 1 kHz), and the simulated ones bend too: T30 is within 1.6 JNDs from 500 Hz to 4 kHz, though 31 and 78 % long at 250 and 125 Hz. EDT is 1.08 s at 1 kHz, as with the door at 4.1°. G from seat to seat, which holds the level of LS1’s sound in the laboratory against LS2’s, is within 1 dB in 20 of the 24 values (5 with the door at 4.1°).

The CR1 runs take 2 minutes and 0.7 GiB with the door at either angle (5 minutes and 0.9 GiB in third-octave bands, 4 minutes calibrated, with the door at 4.1°).

Strength (G)

The measured responses are in pascals. BRAS calibrated the measuring chain to 80 dB at 2 m in front of the dodecahedron at 1 kHz, and corrected each room’s level by 6 dB by hand to meet the expected direct sound. Read with the dodecahedron’s directivity (BRAS’s data, averaged over all directions) and 80 dB as the level of the 1 kHz sine, the measured G is 2.3 to 3.3 dB above classic diffuse theory ($10 \log_{10}(100/r^2 + 31{,}200\ T30/V)$, with the measured T30) at 500 Hz, 1 kHz and 4 kHz in the seminar room and the hall, and 1.7 to 2.4 dB in the auditorium, and the direct sound is about 2 to 3 dB above the calibration even in the dodecahedron’s loudest direction, in the hall’s pairs where it stands apart. At 2 kHz it is within 0.6 dB of the theory, about 3 dB below the neighbouring octaves in all three rooms, which the dodecahedron’s directivity data do not explain. The simulated G is within 1.3 dB of the same theory there (0.0 in the seminar room). Read with 80 dB as the sine’s peak instead, which lowers the measured G by 3 dB, it comes within 1.3 dB of the theory at 500 Hz, 1 and 4 kHz in all three rooms, and 2.4 to 3.5 dB below it at 2 kHz. Which reading BRAS meant is not stated, so an absolute G against these responses would test the reading, not the simulation. These absolute levels come from one-off checks with BRAS’s dodecahedron data and measured responses, and the simulated ones from the fitted runs, which the fixes at 125 Hz leave as they were from 250 Hz up; no script in the repository makes them yet.

Simulated minus measured G with the first reading, at 500 Hz, 1, 2 and 4 kHz (means over the pairs, standard deviations 0.2 to 0.9 dB):

	500 Hz	1 kHz	2 kHz	4 kHz
CR2	-2.3	-3.1	+0.2	-2.9
CR3	-1.3	-2.1	+0.1	-3.2
CR4	-1.7	-2.1	+0.5	-2.2

These are left out of the JND counts. What is compared is G from seat to seat: each pair’s simulated G less the mean over the room’s pairs, against each pair’s measured level less its mean, which the level’s reference does not touch. From 125 Hz to 4 kHz it is within one JND (1 dB) in 53 of the 60 values in the seminar room, 52 in the hall and 50 in the auditorium, 0.3 to 0.6 dB off on average. In the coupled rooms, where it holds the level of the chamber’s sound against the laboratory’s own, it is 1.8 dB off with the door at 4.1°, 5 of 24 within 1 dB: the coupling again; with the door at 30.4°, 0.6 dB off, 20 of 24.

Interaural cross-correlation

BRAS also measured binaural responses in the seminar room, the hall and the auditorium: five QSC K8 loudspeakers (LS3 to LS7), one at a time, aimed at FABIAN, a head and torso simulator 4 to 5 m away in the seminar room and 10 to 11 m in the hall and the auditorium, facing LS7 with its head straight. `validation/bras_binaural.py` simulates each loudspeaker with the K8's measured directivity (`validation/bras/QSC_K8.csv`) and the fitted absorption, renders the response through FABIAN's head-related impulse responses (every 2°), and compares IACC_E (the first 80 ms) and IACC_L (from 80 ms) with those of the measured responses, analysed by this package, in JNDs of 0.075 (ISO 3382-1). The late part of a rendered response is noise drawn anew with each seed, and so are the phases of paths over edges that arrive together, so the simulated values are means over 10 seeds. From 125 Hz to 4 kHz their standard deviation, averaged over a room's loudspeakers, is 0.10 at most, and 0.14 for one loudspeaker. Means over the five loudspeakers, measured / simulated, and how many of the 30 loudspeaker and band values from 125 Hz to 4 kHz are within one JND:

IACC_E	125 Hz	250 Hz	500 Hz	1 kHz	2 kHz	4 kHz	within 1 JND	mean (JNDs)
CR2	0.90 / 0.91	0.66 / 0.74	0.38 / 0.32	0.37 / 0.32	0.42 / 0.37	0.36 / 0.31	18	1.1
CR3	0.94 / 0.93	0.79 / 0.71	0.48 / 0.31	0.48 / 0.40	0.65 / 0.53	0.55 / 0.52	19	1.2
CR4	0.96 / 0.96	0.87 / 0.85	0.86 / 0.81	0.81 / 0.83	0.87 / 0.87	0.87 / 0.87	25	0.5
IACC_L	**125 Hz**	**250 Hz**	**500 Hz**	**1 kHz**	**2 kHz**	**4 kHz**	**within 1 JND**	**mean (JNDs)**
CR2	0.90 / 0.89	0.54 / 0.67	0.14 / 0.18	0.11 / 0.09	0.07 / 0.08	0.06 / 0.07	25	0.6
CR3	0.95 / 0.91	0.73 / 0.71	0.20 / 0.17	0.14 / 0.12	0.11 / 0.09	0.10 / 0.10	26	0.6
CR4	0.92 / 0.92	0.72 / 0.74	0.33 / 0.27	0.34 / 0.20	0.29 / 0.25	0.26 / 0.15	19	1.2

IACC_L is within one JND in 19 to 26 of the 30 values; in the auditorium it is 2.0 and 1.4 JNDs low at 1 and 4 kHz, which is not traced. IACC_E is within one JND in 25 in the auditorium, and in the seminar room within about one JND on average in every band, but 1.0 to 2.2 JNDs low in the hall from 500 Hz to 2 kHz. Split into the direct sound (its first 5 ms) and the rest of the first 80 ms, and the simulated responses also into their direct sound, specular reflections, diffracted paths and late part, rendered one at a time (one seed of the rendering, `validation/bras_iacc_parts.py`):

- **The hall from 500 Hz to 2 kHz.** From LS7 the simulated sound after the first 5 ms is about as strong against the direct sound as measured (3.0 dB above it at 1 kHz in both, 4.3 and 0.6 dB against 5.8 and 1.3 dB at 500 Hz and 2 kHz), but far less alike at the two ears: its IACC is 0.18 at 1 kHz against 0.63 measured, and 0.35 against 0.73 at 500 Hz. At 1 kHz the direct sound, the specular reflections, the diffracted paths and the late part each carry a quarter to a third of the early sound, and the diffracted paths and the late part are the least alike at the ears (0.25 and 0.21). Why the simulated early sound is less alike at the ears than the measured is not traced.
- **The auditorium at 500 Hz.** The simulated sound in the 75 ms after the direct sound is 4.4 and 6.7 dB stronger against the direct sound than measured (LS4 and LS5), and within 1.2 dB from 1 to 4 kHz. At 500 Hz the specular reflections alone, over the first 80 ms, are 1.3 dB below the direct sound from both loudspeakers, where all of the measured sound after the direct sound is 5.9 and 6.3 dB below it. IACC_E there is within one JND on average all the same (0.81 against 0.86); those reflections reach the two ears nearly alike (0.89 and 0.90).

How much a run varies

The late rays are traced in 10 batches by default, which gives each simulated parameter a standard error: one standard deviation of how much it would change between runs with other seeds (see Simulation, late reverberation). `bras_validation.py` prints the largest over each room's pairs from 125 Hz to 4 kHz, in JNDs. With the fitted absorption, at 50,000 rays per band:

largest standard error (JNDs)	CR1, 4.1°	CR1, 30.4°	CR2	CR3	CR4
EDT	1.08	0.24	0.12	0.56	0.48
T20	0.42	0.28	0.07	0.34	0.36
T30	0.22	0.23	0.05	0.25	0.20
C50	0.46	0.15	0.06	0.23	0.15
C80	0.57	0.14	0.05	0.22	0.20
D50	0.52	0.11	0.06	0.25	0.17
Ts	0.88	0.28	0.06	0.14	0.20

In the seminar room no standard error is above an eighth of a JND. In the hall and the auditorium EDT's standard error is a third of a JND for a typical pair and band (the median, 0.33 and 0.28) and half a JND at most, so there a difference from the measurement below one JND is partly the run's own; decays 2 to 4 JNDs long are not. In the coupled rooms with the door at 4.1° the largest are those of LS1's sound, which reaches the laboratory through the slit by the door: EDT 1.08 JNDs at 4 kHz; with the door at 30.4° none is above 0.28 JNDs. With the initial absorption the hall's largest is T30's, 0.92 JNDs at one pair at 4 kHz.

Edge diffraction against the exact solution

`validation/edge_diffraction.py` compares the engine's edge diffraction with the Biot-Tolstoy-Medwin (BTM) solution for rigid wedges (Svensson, Fred and Vanderkooy, 1999), which writes the diffracted sound as an integral along the edge: first order over one edge, second order over two in turn, with half the integral when the two edges share a face. The script integrates both along edges 80 m long, densest where the shortest path touches them, and checks itself against the UTD for a half-plane, which is exact far from the edge (within 0.03 dB). Levels are in dB re the direct sound in free field over the same distance, the energy averaged over the engine's eight frequencies in each octave band; sources and listeners are in metres from the edge.

Over a thin screen (first order), with the listener 28° to 33° into its shadow, the engine's UTD is within 0.9 dB of BTM at 63 Hz and 0.06 dB from 1 kHz up; Maekawa's chart reads up to 1.2 dB high:

thin screen, engine - BTM (dB)	63	125	250	500	1000	2000	4000	8000
(-5, 0, -2) to (5, 0, -1), BTM -8.4 to -27.2 dB	-0.50	-0.27	-0.13	-0.05	-0.02	-0.00	-0.00	+0.00
(-2, 0, -0.2) to (6, 0, -3), BTM -6.9 to -24.6 dB	-0.77	-0.46	-0.24	-0.11	-0.04	-0.01	-0.00	-0.00
(-3, -1, -1) to (3, 2, -0.5), BTM -6.3 to -23.2 dB	-0.89	-0.56	-0.32	-0.16	-0.06	-0.02	-0.01	-0.00

Over two edges (`second_order`): a block of width w with its top at z = 0, from x = 0 to w, and two thin screens in a row. Each pair of positions is in line across the edges and obliquely (the second from (-6, -2, -0.3) to (w + 2, 3, -4)):

over two edges, engine - BTM (dB)	63	125	250	500	1000	2000	4000	8000
block 0.3 m, (-3, 0, -1) to (3.3, 0, -1.5)	-1.72	-1.26	-0.67	-0.15	+0.19	+0.29	+0.23	+0.13
block 0.3 m, oblique	-0.65	-0.31	-0.19	-0.16	-0.14	-0.10	-0.07	-0.08
block 1 m, (-3, 0, -1) to (4, 0, -1.5)	-0.56	-0.28	-0.04	+0.12	+0.16	+0.12	+0.06	+0.03
block 1 m, oblique	-0.38	-0.28	-0.19	-0.13	-0.08	-0.06	-0.05	-0.03
block 3 m, (-3, 0, -1) to (6, 0, -1.5)	-0.21	-0.07	+0.01	+0.05	+0.04	+0.02	+0.01	+0.00
block 3 m, oblique	-0.28	-0.20	-0.13	-0.09	-0.06	-0.04	-0.02	-0.03
two screens 1 m apart	-1.86	-1.23	-0.74	-0.37	-0.12	-0.00	+0.01	+0.01
two screens 2 m apart, the second 0.4 m higher	-1.30	-0.92	-0.65	-0.44	-0.30	-0.18	-0.11	-0.05

With the block's faces absorbing 0.02 the engine's level is below its own for the rigid block by the loss of the Weyl-van der Pol ground wave along the top, the magnitude of F(w) worked out by hand (0.01 dB at 63 Hz over 0.3 m to 0.43 dB at 8 kHz over 3 m), within 0.03 dB in every band, in energy and in pressure. Against BTM for the rigid block it is then within 1.74 dB, and within 0.64 dB at 63 and 125 Hz in pressure.

The levels themselves run from -9 dB at 63 Hz to -42 dB at 8 kHz. Adding its waves in energy, the engine is low: up to 1.9 dB at 63 Hz where the edges are close together, as over a thin screen, and near a shadow boundary in every band. The waves of an edge add in energy near their boundaries, where the exact solution adds them with their phases. At a shadow boundary the source is at (-5, 0, -2) and the listener at (5, 0, 1.999), 1 mm into the shadow of a thin screen, of a right-angled corner (the block above, 300 m wide) or of a wedge open over 225° (the same block with its front face sloping down at 45° towards the source), or at (5, 0, 1), 10.5° into it; the whole early stage (`early_paths` with the default settings but `coherent_max_band_hz: null`, so that every band adds in energy, and the air's absorption over the direct distance taken off) against BTM averaged over 96 frequencies in each band:

at a shadow boundary, engine in energy - BTM (dB)	63	125	250	500	1000	2000	4000	8000
thin screen, on the boundary, BTM -4.5 to -5.9 dB	-1.29	-0.95	-0.69	-0.50	-0.36	-0.25	-0.18	-0.13
thin screen, 10.5° into the shadow, BTM -5.7 to -18.5 dB	-0.99	-0.68	-0.45	-0.28	-0.15	-0.08	-0.03	-0.02
right-angled corner, on the boundary, BTM -1.8 to -5.6 dB	-2.54	-2.15	-1.73	-1.32	-0.98	-0.72	-0.52	-0.38
right-angled corner, 10.5° into the shadow, BTM -2.1 to -16.3 dB	-1.96	-1.50	-1.05	-0.66	-0.37	-0.18	-0.07	-0.03
wedge open over 225°, on the boundary, BTM 0.3 to -5.3 dB	-3.02	-2.65	-2.22	-1.79	-1.38	-1.04	-0.77	-0.56
wedge open over 225°, 10.5° into the shadow, BTM 0.0 to -15.2 dB	-2.13	-1.61	-1.12	-0.71	-0.40	-0.19	-0.08	-0.03

On the boundary the term of the incident wave continues the direct sound almost wholly, and the terms of the reflections off the edge's faces, which arrive with it, are part of the edge's diffracted wave: the two waves add in energy, and their cross term is lost. Indoors the paths add in energy from 250 Hz up by default, so there the level on the shadow boundary of a right-angled corner is 1.73 dB low at 250 Hz, 0.98 dB at 1 kHz and 0.38 dB at 8 kHz; behind the blunter wedge open over 225° it is 2.22, 1.38 and 0.56 dB low, and behind the thin screen 0.69, 0.36 and 0.13 dB (see Known issues). In the 63 and 125 Hz bands the engine adds them in pressure by default (`simulation.early.coherent_max_band_hz`, see Simulation), the four terms of each coefficient and the paths' waves with their phases, and there it is within 0.05 dB of BTM over the thin screen, 0.08 dB at the shadow boundaries and 0.25 dB over two edges, but for the 0.3 m block in line (0.6 dB):

engine in pressure - BTM (dB)	63	125
thin screen, (-5, 0, -2) to (5, 0, -1)	+0.02	+0.01
thin screen, (-2, 0, -0.2) to (6, 0, -3)	+0.03	+0.01
thin screen, (-3, -1, -1) to (3, 2, -0.5)	+0.04	+0.02
thin screen, on the shadow boundary (`early_paths`)	-0.01	-0.01
thin screen, 10.5° into the shadow (`early_paths`)	+0.02	+0.01
right-angled corner, on the shadow boundary (`early_paths`)	-0.00	-0.01
right-angled corner, 10.5° into the shadow (`early_paths`)	+0.06	+0.03
wedge open over 225°, on the shadow boundary (`early_paths`)	+0.02	-0.01
wedge open over 225°, 10.5° into the shadow (`early_paths`)	+0.08	+0.03
block 0.3 m, in line	-0.61	-0.50
block 0.3 m, oblique	+0.12	+0.24
block 1 m, in line	+0.11	+0.11
block 1 m, oblique	+0.13	+0.06
block 3 m, in line	+0.22	+0.16
block 3 m, oblique	+0.08	+0.04
two screens 1 m apart	-0.11	-0.13
two screens 2 m apart, the second 0.4 m higher	+0.03	-0.01

The textbook UTD for two edges, with the second edge's distance parameter for a wave from a line, was up to 6 dB low there; the engine takes it from the curvature of the first edge's wave where it reaches the second.

Along a top absorbing 0.2 to 0.8, the ground wave's loss is too large against the full wave. The wave solver ran a block 24 m long and 12 m deep whose top alone absorbs (5 cm cells, the first 50 ms, and a probe 3 m above the top that widened the grid, as with the grid ending a few cells above the top its matched layer had put the rigid level 0.5 to 0.6 dB high at 125 Hz), the source and the listener 1 m below the plane of the top and 3 m off its edges, and over the 1 m top absorbing 0.5 also 2 and 4 m off them with the listener 0.5 m below. Over tops 1 and 3 m wide absorbing 0.2, 0.5 and 0.8 it puts the level at 63 and 125 Hz 0.04 to 2.0 dB below the rigid block's, and the engine is 0.22 to 3.1 dB below that, more as the top absorbs more and is wider (0.22 to 0.38 dB at 0.2, 1.8 to 3.1 dB at 0.8); before these paths took the ground wave it was 9.3 to 15.7 dB below. The wave solver there is within 0.09 dB of BTM over the rigid blocks and within 0.07 dB of the exact solution over a plane absorbing 0.2 to 0.8 (with the engine's fitted admittance, source and listener 0.1 m high and 7 m apart), and with 6.2 cm cells the level over the 3 m top absorbing 0.8 moves by 0.03 and 0.04 dB (all of these with the wave solver's trilinear reading; its cubic one moves the rigid 3 m block's level by 0.01 and 0.03 dB).

As the source rises through the plane of the 1 m block's top, 3 m in front of it, it comes into sight of the far corner, and the path over that corner alone takes over from the path over both; likewise as the listener rises behind the block. The largest step between positions 1 mm apart, over ±1 cm and exactly in the plane, is 0.62 dB for the source and 0.72 dB for the listener at 63 Hz with the waves added in energy, 0.01 dB with them in pressure (the default in that band), and at most 0.07 dB from 1 kHz up. With `first_order` the level drops from the path over one corner to nothing.

Just over the top, with the source and the listener both a little above its plane and in sight of each other, the source's wave passes both corners. Each corner's terms take their part off the direct sound and off its reflection off the top, and the path over both gives back what they take twice: at grazing incidence the direct sound, the reflection and the two corners' terms come to nothing, and the path over both is all there is, half the direct sound at high frequencies. The whole early stage (`early_paths`, with the paths in pressure up to 500 Hz and the air's absorption over the direct distance taken off) against BTM with the geometric waves, over a block 10 m wide with the source 10 m before it and the listener 5 m past it, and over the 1 m block with both 3 m off, by their heights above the plane of the top:

just over a block's top, engine - BTM (dB)	63	125	250	500
10 m, both 1 cm above, BTM -5.2 to -5.6 dB: `second_order`	-0.02	-0.03	-0.04	-0.05
`first_order`	-12.63	-15.33	-18.10	-20.66
10 m, both 0.5 m above, BTM -3.6 to -1.1 dB: `second_order`	-0.02	+0.01	+0.03	+0.04
`first_order`	-6.45	-4.57	-2.58	-0.90
10 m, 0.5 and 0.05 m above, BTM -4.5 to -3.7 dB: `second_order`	-0.06	-0.04	-0.02	-0.00
`first_order`	-10.33	-9.54	-7.59	-5.31
10 m, both 1 m above, BTM -2.1 to 2.5 dB: `second_order`	+0.02	+0.04	+0.03	+0.01
`first_order`	-2.65	-1.02	+0.18	+0.72
10 m, both 2 m above, BTM -1.8 to 3.5 dB: `second_order`	+0.05	+0.03	+0.01	+0.02
`first_order`	+0.06	+0.65	+0.52	-0.08
1 m, both 5 cm above, BTM -4.4 to -4.9 dB: `second_order`	-1.17	-0.72	-0.75	-0.62
`first_order`	-9.01	-11.05	-12.59	-12.59
1 m, both 0.3 m above, BTM -3.4 to -1.9 dB: `second_order`	-0.67	-0.13	-0.03	+0.10
`first_order`	-6.51	-5.55	-3.76	-1.89

With `second_order` the engine is within 0.06 dB of BTM over the 10 m top, and within 1.2 dB over the 1 m top, whose corners are 0.18 wavelengths apart at 63 Hz. A centimetre above the 10 m top it is at -5.84 to -5.91 dB from 1 kHz up, 0.22 to 0.28 dB below BTM, where half the direct sound is -6.02 dB. `first_order` has no path over two edges and is low by what that path carries (see Known issues). From 1 kHz up the scene adds the direct sound and its reflection off the top in energy, which BTM adds with their phases: half a metre and more above the top the two then differ by up to 4.5 dB in a band, with either setting. Swapping the source and the listener leaves every level of the table as it is. Within a millimetre of the plane, above it or below on either side of the 10 m block, the level is the same within 0.11 dB in every band; with the source or the listener exactly in the plane it differs by up to 0.8 dB at 63 Hz and 0.29 dB at 500 Hz, and no more from 1 kHz up (at a gable roof's slope by 1.5 dB at 63 Hz, in a one-off check). Over the 1 m top it differs by up to 1.7 dB at 63 Hz and 0.8 dB from 1 kHz up between those positions. With a top absorbing 0.5, the level 1 mm and 1 cm above the plane of the 1 m top is below the rigid top's by the ground wave's loss over the top (0.9 dB at 63 Hz to 9.3 dB at 8 kHz) within 0.5 dB, as it is just below the plane within 0.12 dB (`tests/test_diffraction.py`).

Behind the block, with the source still above the plane of its top, the source's wave passes the near corner on its way to the far one, round which the listener hears it: the near corner's terms take their part off it, and the path over both carries that. Over the 10 m block, the source 10 m before it and the listener 5 m past it:

behind a 10 m block, engine - BTM (dB)	63	125	250	500
source 1 cm above, listener 0.5 m below, BTM -6.2 to -8.6 dB: `second_order`	+0.06	+0.09	+0.08	+0.04
`first_order`	+5.67	+5.81	+5.87	+5.88
1 cm above, 5 m below, BTM -12.2 to -21.5 dB: `second_order`	-0.43	+0.03	+0.25	+0.19
`first_order`	+5.17	+5.75	+6.05	+6.03
0.5 m above, 2 m below, BTM -8.2 to -13.5 dB: `second_order`	+0.02	-0.01	-0.02	-0.03
`first_order`	+5.09	+4.93	+4.65	+4.23
2 m above, 2 m below, BTM -6.4 to -9.0 dB: `second_order`	-0.01	-0.02	-0.02	-0.00
`first_order`	+3.38	+2.55	+1.43	+0.10

`first_order` has the path over the far corner as if the near one were not there, up to 6 dB too high (see Known issues).

Two thin screens in a row with their tops on one line, 10 m apart, the source 10 m before the first and the listener 5 m past the second: the Fresnel-Kirchhoff integral over both half-planes (Millington, Hewitt and Immirzi, 1962) leaves, at grazing incidence and high frequencies, 1/4 + atan(√(a c / (b (a + b + c)))) / (2π) of the free field for the distances a, b and c along the way: a third for equal ones, where one edge alone leaves a half and two far apart a quarter, and -9.98 dB here. A wave bent at the first edge and again, as a wave of its own, at the second does not hold there, as the second edge stands in the transition zone of the first: BTM's second order and the product of two UTD coefficients both leave about a quarter, -12 dB. With the source and the listener 1 mm above the line of the tops the engine (`second_order`, the whole early stage) is at -10.97 to -11.29 dB from 2 kHz up, 1.0 to 1.3 dB low; 1 mm below it, where the path over both is all there is, at -12.08 to -12.11 dB, 2.1 dB low (see Known issues). `first_order` is at -30.5 to -36.2 dB above the line and has nothing below it. With both exactly on the line no ray finds the direct sound, while the rays that look for the paths over the tops, which start just off the edges, find those, and each top's path alone would stand for all of the direct sound. The engine takes the level there to be the one just above the line: every top on the way takes its term off the direct sound, as on the lit side of its boundary, by the share its edge gives where the way passes near its end or beside it, and one top's path carries the direct sound. The level is that 1 mm above the line within 0.1 dB in every band, and within 0.15 dB where one of the two screens ends a centimetre beside the way, where both end 10 and 15 cm beside it, and where the way passes through the end of a screen at 45 degrees to it (`tests/test_diffraction.py`; higher above the end of an oblique screen it is not, see Known issues). A reflection that passes level with the tops is taken the same way where both tops stand on one leg of it. With a wall beside two screens, where the reflection off the wall lies between the tops, the terms that make reflections fade at edges take the product of the two tops' terms for it, the level just below the line: the whole level is that 1 mm above the line within 0.1 dB (it was 9.3 to 11.3 dB above that). `first_order` has no path over a top and the upright end of the next screen, which holds the terms of that end's faces on the line: where a screen ends near the way it is up to 1.2 dB below its level 1 mm above the line at 63 Hz. With one of the two on the line and the other 0.5 m above it, the level is that with it 1 mm above the line, and 0.5 dB (the source) or 1.0 dB (the listener) above that with it 1 mm below at 63 Hz, 0.1 dB (the source) or 0.3 dB (the listener) at 2 kHz.

More edges in a row, the source 10 m before the first and the listener 5 m past the last, both at one height from the plane of the tops (the whole early stage with the paths in pressure up to 500 Hz):

edges in a row, `second_order`, level (dB)	63	125	250	500	1000	2000	4000	8000
two screens 10 m apart, 1 cm above	-8.89	-9.48	-9.86	-9.89	-10.28	-10.26	-10.27	-10.46
in the plane	-9.34	-9.90	-10.32	-10.42	-10.87	-10.91	-11.00	-11.28
1 cm below	-9.91	-10.58	-11.06	-11.41	-12.17	-12.25	-12.34	-12.46
three screens 5 m apart, 1 cm above	-9.75	-7.96	-8.07	-8.00	-7.41	-7.45	-7.58	-8.08
in the plane	-9.28	-7.48	-7.55	-7.47	-6.82	-6.87	-7.02	-7.57
1 cm below	-9.91	-10.58	-11.06	-11.41	-12.17	-12.25	-12.34	-12.46
one top 10 m wide, 1 cm above	-5.25	-5.46	-5.61	-5.69	-5.91	-5.91	-5.89	-5.84
in the plane	-5.17	-5.43	-5.60	-5.73	-6.02	-6.02	-6.02	-6.02
1 cm below	-5.23	-5.50	-5.71	-5.86	-6.09	-6.17	-6.27	-6.40
two tops 5 m wide, 5 m apart, 1 cm above	-7.05	-6.21	-6.31	-6.34	-6.25	-6.25	-6.26	-6.31
in the plane	-10.94	-12.29	-13.19	-13.97	-14.38	-14.53	-14.78	-15.55
1 cm below	-5.26	-5.52	-5.71	-5.85	-6.08	-6.16	-6.27	-6.39
a 10 m top with a 2 cm slot, 1 cm above	-6.35	-6.46	-6.32	-6.27	-6.66	-6.65	-6.61	-6.57
in the plane	-7.76	-9.48	-10.54	-11.12	-11.75	-11.88	-11.96	-12.01
1 cm below	-5.23	-5.50	-5.71	-5.86	-6.09	-6.17	-6.27	-6.40
three tops 3 m wide, 2 m apart, 1 cm above	-1.89	-1.75	-1.45	-1.28	-0.94	-0.93	-0.95	-1.03
in the plane	-8.80	-4.77	-4.70	-4.47	-3.71	-3.62	-3.59	-3.70
1 cm below	-5.25	-5.51	-5.70	-5.85	-6.08	-6.16	-6.27	-6.39
a top 5 m wide, then a screen, 1 cm above	-11.00	-10.39	-10.65	-10.83	-10.85	-10.82	-10.81	-10.86
in the plane	-11.37	-12.06	-11.26	-10.86	-10.34	-10.34	-10.39	-10.61
1 cm below	-10.52	-11.01	-11.36	-11.62	-12.15	-12.23	-12.33	-12.44
a screen, then a top 5 m wide, 1 cm above	-10.83	-10.14	-10.43	-10.61	-10.64	-10.62	-10.59	-10.67
in the plane	-13.67	-14.28	-14.57	-14.89	-14.94	-15.02	-15.24	-15.99
1 cm below	-10.65	-11.09	-11.42	-11.66	-12.14	-12.22	-12.31	-12.44

Over one top the level is within 0.9 dB of half the direct sound, above its plane, in it and below. A path over two edges is all the engine has of more edges in a row, and there is no exact solution here to hold it against. Three screens leave 1.5 to 2.9 dB more than two from 125 Hz up, a centimetre above their tops, where a screen more should leave less; a slot 2 cm wide across the 10 m top, far narrower than the wavelengths up to 500 Hz, lowers the level by 0.6 to 1.1 dB; over three tops in a row the level is 3.4 to 5.0 dB above that over one. Over a top 5 m wide and then a screen level with it, 5 m past it, the level a centimetre above their plane is the same with the source and the listener swapped, within 0.15 dB. Closer to the top or to the plane two things count. A path over the screen's top reflects off the top's plane where its own line meets that plane: with the two level, at the screen's top itself, where the top is not, so that it is no path, whatever the ray that looks for it meets from its start a millionth of the model's size off the edge; and the screen's top lies beside the reflection off the top, before the point of reflection one way round and after it the other, and takes its term on that leg. In six cases from a millimetre to a metre above the plane, the screen 0.1 to 5 m past the top, the two ways round agree within 0.05 dB (`tests/test_diffraction.py`), in four of them with every sequence of reflections tested (`method: image_sources`), as the rays can find the reflection between the top's wall and the screen one way round only (see Known issues). With the screen 0.1 m past the top and its top a micrometre above the top's plane or below it, the level a centimetre above is that of a level top within 0.04 dB with `first_order` and, a micrometre above, within 0.34 dB up to 500 Hz with `second_order` (below the plane and above 500 Hz it is not: see Known issues). Where such a screen ends 1 cm to 0.5 m beside the way, the level 1 or 2 mm above the plane is, with `second_order`, that a centimetre above: between two tops within 0.23 dB (0.39 dB with the way through the screen's end), and 5 m before a top or 5 m past it within 0.64 dB, both ways round. A centimetre above, with the screen first and the way a centimetre inside its end, it is up to 500 Hz 0.5 to 1.2 dB above half the sum of the fields without the screen and with an endless one (Kirchhoff's approximation for the way through the screen's end, -6.2 to -6.8 dB), each from the two-dimensional wave calculation of the Known issues; `first_order` is 7.4 to 9.3 dB below that. Exactly in the plane of several flat tops the level is off by more: 3.9 to 9.2 dB below that a centimetre above the plane of two tops, 2.5 to 6.9 dB below it over three and 2.8 to 5.3 dB below it over a screen and then a top, where a top and then a screen is within 1.7 dB of it. It is off within two millionths of the model's size below the plane too (1.2 mm here), where the rays that look for paths over edges, which start a millionth of the model's size off the edges, find paths that are not there: 0.01 mm below the plane of two tops the level is at +3.3 to +4.1 dB, and below that of three at +9.1 to +10.0 dB. In the bands that add in energy the levels up to 500 Hz are those of the table within 1.7 dB, but at 63 Hz for three screens above their tops and level with them (2.5 and 2.6 dB higher), exactly in the plane of a slotted top (2.1 dB lower) and of three tops (4.9 dB higher). `first_order` has no path over two edges: a centimetre above two tops it is at -0.4 to -1.5 dB, and above three at +4.9 to +5.8 dB (see Known issues).

At the corner of a thin screen, the quarter of the plane x = 0 below y = 0 and z = 0, the source at (-5, -3, -1) and the listener walking from (5, 2.5, -2) to (5, 4, -2) 1 cm at a time with `direct` and every band in pressure, the direct sound comes round the screen's side edge at y = 3. The exact level, BTM over both edges, each from the corner on (the sound from one edge to the other left out), changes smoothly there; the engine's steps:

a screen's corner, `direct` (dB)	**63**	**125**	**250**	**500**	**1000**	**2000**	**4000**	**8000**
largest step, engine	0.01	0.01	0.03	0.06	0.05	0.07	0.10	0.15
largest step, BTM	0.00	0.01	0.02	0.06	0.04	0.07	0.10	0.15
largest difference, engine - BTM	0.06	0.01	0.05	0.01	0.00	0.01	0.00	0.10

The path over the top edge still touches it 8 cm from the corner there as the direct sound appears round the side edge. Each edge's term takes the part of its Fresnel zone that the edge covers, the two edges sharing it at the corner, so both change smoothly there. With the terms stopping at the ends, as the UTD has them, the level steps by 2.82 dB at 63 Hz (1.37 dB at 250 Hz) and is up to 1.83 dB off.

The two halves add up so only where the edges are square (within 1°). At any other angle, and where a screen's top bends, each of an edge's four terms takes the edge's own part of it as BTM's integrand weights the edge near the point the path touches: by $1 / (t^2 + v^2)$ with the Fresnel integral's phase, v being how far the term is from its shadow or reflection boundary in the Fresnel integral's argument t. On its boundary a term then comes from that point alone, so it keeps its whole step while the point lies on the edge, and the other edge's term takes it over past the end. A screen in x = 0 with a corner of 45° to 170° at the origin (at 45° also with one edge upright), the source at (-5, -3, -1) and the listener walking from (5, -1, -2) to (5, 6, -2) 5 cm at a time across both edges' boundaries, BTM over both edges at 96 frequencies in a band:

corners of other angles, `direct` (dB)	**63**	**125**	**250**	**500**	**1000**	**2000**	**4000**	**8000**
45° and 60°, largest difference	0.09	0.07	0.06	0.07	0.03	0.03	0.08	0.10
120° to 170°, largest difference	0.05	0.08	0.07	0.04	0.05	0.12	0.10	0.17

The steps are BTM's within 0.07 dB. With each edge taking the Fresnel integral's share (halved at 45° to 120°), the level stepped by up to 3.0 dB at 125 Hz at the corner of 45° with an upright edge and was up to 2.7 dB off; at the other corners it was 0.3 to 1.1 dB off.

A screen whose top bends: two screens below z = 0 that meet along the z axis, one in x = 0 for y < 0, the other turned about the z axis by 2, 5, 10 or 14°, away from the source or towards it. The source is at (-5, -0.5, -1); the listener rises 1 cm at a time through the top's shadow boundary from z = 0.9 to 1.1 m at (5, -0.5, z) and (5, 0.3, z), where the line to the source passes the top 0.5 and 0.1 m from the bend, and walks deep in the shadow from (5, -1, -2) to (5, 3, -2) 5 cm at a time; BTM is taken over both tops from the bend on:

a bent screen, `direct` (dB)	**63**	**125**	**250**	**500**	**1000**	**2000**	**4000**	**8000**
across the boundary, largest difference	0.03	0.01	0.01	0.01	0.00	0.00	0.00	0.00
deep in the shadow, largest difference	0.04	0.02	0.01	0.01	0.02	0.01	0.01	0.01

Both ways round the steps are BTM's within 0.03 dB. With each top taking the Fresnel integral's share of its terms, the level 0.1 m from the bend was up to 5.3 dB too high across the boundary with the bend away from the source, stepping by 5.4 dB, and up to 16.4 dB too low with it towards the source, stepping by 16.6 dB.

Only a thin screen's edges fade so. At the corners of a solid block, where three right-angled edges meet, the terms still stop at the ends. The block runs from x = 0 to 2 m and y = -10 to 0 m, with its top at z = 0, open underneath and reaching far down. The source is at (1, 3, 1), and the listener walks from (4, -8, h) to (4, 2, h), 5 cm at a time, at h = 0.3 and 1.8 m, past where the paths over the block's edges reach its corners. BTM is taken over each of its edges, from end to end, with the geometric paths:

a block's corners, `direct` (dB)	63	125	250	500	1000	2000	4000	8000
0.3 m up: largest step, engine	2.25	1.83	0.99	0.29	1.11	0.48	0.07	0.08
largest step, BTM	0.11	0.03	0.03	0.04	0.04	0.05	0.04	0.03
largest difference, engine - BTM	2.29	1.50	1.62	0.46	1.44	0.59	0.14	0.14
1.8 m up: largest step, engine	2.41	2.06	0.47	1.97	0.96	0.99	0.62	0.44
largest step, BTM	0.04	0.03	0.04	0.04	0.04	0.04	0.05	0.11
largest difference, engine - BTM	1.96	1.45	0.92	1.89	0.97	0.89	0.58	0.50

Fading the three edges' terms at each corner as at a screen's corner, in a one-off check, halved the largest steps 0.3 m up (1.17 dB at 63 Hz). But it put the level up to 1.25 dB above BTM at 63 and 125 Hz from y = -4 to 0 m, where stopping is within 0.36 dB. A model of such corners is an open item on the roadmap.

The early stage of a hall 30 x 20 x 10 m with four alcoves along each long wall (16 vertical edges), with the source and the listener in alcoves out of each other's sight, takes 44 ms per source and listener on four cores without diffraction (23 paths), 525 ms with `first_order` (979 paths) and 600 ms with `second_order` (993 paths): the median of four timings, each the mean of three runs after a first one, with nothing else running on the machine (10 November 2026, the middle of three such medians). The terms that make the other specular paths fade at the edges (next section) give most of the paths, as each term is a path over its edge, and so is each product of terms. In a one-off check when they were written, on a run where the early stage took 650 ms with `first_order` and 710 ms with `second_order`, they took about 580 ms of the first; without them it took 70 ms (71 paths) and 135 ms (86 paths), and 4.0.0 took about 1.0 and 1.1 s on the same computer.

Reflections that fade at edges against the exact solution

`validation/edge_fade.py` checks the terms that make every specular path fade in and out at edges (see Simulation) where the exact solution is known: a vertical edge standing on a rigid ground. The ground mirrors the edge into itself, so the sound is that of an edge infinite in height from the source and from its image in the ground: the geometric paths of each that reach the listener, plus each one's BTM term (as above). The source stands 0.5 m up, and a listener 1.5 m up walks across the edges' boundaries 1 cm at a time:

- the end of a thin screen, with `diffraction: direct`: the reflection off the ground crosses the shadow boundary with the direct sound, and only the fade bends it;
- the corner of a building (a right-angled wedge), with `direct` and with `first_order`: the listener passes the boundary of the reflection off one wall and then the corner's shadow boundary, each with its copy off the ground. With `first_order` the edge diffraction bends all of those paths itself, and the fade must add nothing to it;
- a strip 2 m wide (a thin screen on the ground), with `direct` and with `first_order`: the listener, on the source's side, walks through the reflection off the strip, which a face that narrow gives only in part at low frequencies;
- a gap of 4 m between two screens in line, with `direct`: behind one screen's end, through the gap and behind the other's.

For the strip and the gap the exact level adds each end's term to the geometric paths and leaves out the sound diffracted from one end to the other (at the strip it changes the level by at most 0.1 dB, at 63 to 250 Hz). Everything adds in pressure, as outdoors, and the exact level is averaged over 96 frequencies in each octave band. The largest difference from it along each line, in dB:

largest difference from the exact level (dB)	**63**	**125**	**250**	**500**	**1000**	**2000**	**4000**	**8000**
screen end, `direct`, with the fade	0.04	0.02	0.01	0.01	0.01	0.00	0.01	0.01
without	5.90	5.79	5.30	3.77	5.26	4.43	3.72	3.72
building corner, `direct`, with the fade	0.08	0.05	0.02	0.01	0.01	0.01	0.02	0.04
without	5.93	5.85	5.52	4.11	5.22	4.48	3.64	3.52
building corner, `first_order`, with the fade	0.08	0.05	0.02	0.01	0.01	0.01	0.02	0.04
without	0.08	0.05	0.02	0.01	0.01	0.01	0.02	0.04
2 m strip, `direct`, with the fade	0.02	0.01	0.01	0.00	0.00	0.00	0.01	0.19
without	0.90	0.54	0.97	1.71	0.18	0.64	0.41	0.45
2 m strip, `first_order`, with the fade	0.02	0.01	0.01	0.00	0.00	0.00	0.01	0.19
without	0.42	0.27	0.29	0.14	0.01	0.01	0.01	0.19
4 m gap, `direct`, with the fade	0.03	0.02	0.01	0.00	0.01	0.00	0.00	0.01
without	7.01	5.57	5.25	3.67	6.05	4.45	3.92	3.71

With the fade the level changes from one listener to the next as the exact level does: the largest step along each line is the same as the exact level's to 0.01 dB in every band, at most 0.20 dB at 8 kHz past the screen's end, but for 8 kHz on the strip, where it is 0.06 dB and the exact level's 0.13 dB. There the exact level is the one off: its average over 96 frequencies swings round one over 768 by up to 0.19 dB near y = -1 m, which makes the strip's 0.19 dB at 8 kHz in the table, and steps by 0.03 dB where it stepped by 0.13 dB; at every fourth listener along the line the engine is within 0.001 dB of the average over 768. Without the fade the level stepped by up to 8.6 dB past the screen's end, 7.6 dB past the corner and 9.2 dB past the gap's ends. At the corner with `first_order` the fade changes the level by less than 0.001 dB: where the edge diffraction already bends a path, the fade leaves out the terms it adds. On the strip with `first_order` the edge diffraction bends the reflection off the strip, and the fade adds its copy off the ground (0.4 dB at 63 Hz).

Behind a barrier 3 m high on a rigid ground (a thin screen across x = 0), with the source at (-4, 0, 0.5) and a listener at x = 4 m rising from 1 to 8 m 1 cm at a time, the exact level is that of the geometric paths and of the four paths over the top, from the source and its image in the ground to the listener and its image, each by BTM. With `first_order` every path over the top is the edge diffraction's own, and the fade changes nothing (less than 0.001 dB):

barrier on a rigid ground, `first_order` (dB)	**63**	**125**	**250**	**500**	**1000**	**2000**	**4000**	**8000**
largest difference, `one_side`	2.20	2.98	2.60	0.89	0.80	0.67	0.76	0.73
largest difference, `both_sides`	0.03	0.02	0.03	0.01	0.03	0.05	0.04	0.05
largest step, `both_sides`	0.06	0.14	0.12	0.05	0.09	0.15	0.20	0.16
largest step, exact	0.06	0.14	0.12	0.05	0.09	0.15	0.21	0.16

`one_side` leaves out the fourth path, reflected off the ground before the top and again after it (see Outdoor results). With `both_sides` the level is within 0.05 dB of the exact level in every band, and steps as it does. Up to 4.0.1 it stepped by 0.35 dB at 63 Hz (`one_side` by 0.33 dB) at z = 6.5 m, where the listener rises into sight of the reflection off the ground before the barrier, and was 0.33 dB low on average above (0.38 dB at most): the path over the top with the reflection off the ground behind it was left out there. Both reflections come from the source's image in the ground, and the edge diffraction took the one it found for the geometric wave of both.

Behind a barrier 1 m thick on the same ground (two right-angled edges 1 m apart, 3 m up), with the source at (-5, 0, 0.5), `second_order` and listeners 5 m behind it, the exact level is BTM's sound over both edges from the source and its image to the listener and its image (up to about 4 minutes a listener). The engine minus the exact level:

thick barrier, `second_order` (dB)	**63**	**125**	**250**	**500**	**1000**	**2000**	**4000**	**8000**
listener 0.5 m up	-9.85	-6.96	+3.01	-5.66	-5.54	-5.18	-5.08	-5.40
1.5 m up	-5.06	-0.27	-0.49	-3.20	-2.44	-2.14	-2.17	-2.41
2.5 m up	-1.09	-5.48	+3.83	-2.53	-2.11	-1.89	-1.75	-1.90

A path over two edges takes no reflection off the ground, so of the four paths over the barrier the engine has one: 1.7 to 5.7 dB low from 500 Hz up, and up to 9.9 dB low and 3.8 dB high below, where the four interfere. `both_sides`, which pairs reflections round one edge, and the fade change nothing here (see Known issues).

Round a corner indoors

Indoors the paths that pass an edge reflect off the floor, the ceiling and the walls several times, and their sounds add in energy above 125 Hz (each band's level is the sum of its paths'). No exact solution covers a whole room, so the check is that the level no longer jumps where a path appears or goes: the `gallery` example's visitor walks away from the guide and round the corner at (8, 8) m, 1 cm at a time from y = 6 to 18 m along x = 3, 4 and 5.5 m (3,603 positions), with reflections up to third order and `first_order`, as the example has them. The largest step between neighbours, in dB:

largest step (dB)	**63**	**125**	**250**	**500**	**1000**	**2000**	**4000**	**8000**
with the fade	0.06	0.10	0.02	0.03	0.04	0.05	0.07	0.13
without	3.03	3.72	2.38	2.52	2.64	2.73	2.78	2.83

The largest steps left stand out from their neighbours' by at most 0.08 dB, all at 63 and 125 Hz; from 250 Hz up by at most 0.01 dB. The largest, 0.08 dB at 125 Hz between y = 13.33 and 13.34 m on x = 5.5 m, and a smaller one at 63 Hz near y = 9.81 m on x = 4 m come from a path that the corner stops on one leg while another of its reflection points slides past the end of its face into a corner of the room: the path is left out there, though its path over the corner still reflects on the face. The largest on x = 3 m at 125 Hz, 0.02 dB between y = 12.99 and 13.00 m, is a leg that takes the corner's term on the lit side on one side of the step and not on the other; the others, of up to 0.035 dB (63 Hz, y = 12.00 m on x = 4 m), are not traced. `validation/walks.py` prints how far the steps stand out on each line, and where.

Where the edge diffraction already bends every path the early stage follows (`max_order: 0` with `direct`, `max_order: 1` with `first_order`), the fade changes the level along x = 4 m by less than 0.05 dB (a one-off check). It had taken away the terms of reflections off faces the listener was behind, which the early stage follows nowhere at that order: 2.9 dB less at once at 63 Hz behind the plane of the wall round the corner.

Along a street

The `street` example's car (see Vehicles passing by) passes the pavement, a balcony of the office block, the point 2 m in front of the offices' wall (`balcony_2m`) and the garden behind the noise barrier. No exact solution covers a street either, so the check is again that the level does not jump: the early level, A-weighted with the car's sound power, every 10 cm of the car's 125 m, and again 1 cm apart round each step of 10 cm that is among the four largest for its listener in some band or in dB(A) (`validation/walks.py`). Outdoors paths add in pressure in every band, so the level also swings, smoothly, where they go in and out of step: at 8 kHz, where a centimetre is a quarter of a wavelength, the garden's level changes by up to 0.75 dB from one centimetre to the next. The late part is left out, as its rays are drawn at random (the early level repeats to the last bit): with the scene's rays few of them reach the garden, and in a one-off check two runs of the same pass without a `seed` (the car's levels of 4.1.0, every metre, as `passby_demo.py walk` simulates it) put the garden's level up to 2.1 dB(A) apart at one place (x = -35 m; 0.37 dB(A) rms over the pass), the pavement's and the balcony's up to 0.6 and 0.8 dB(A); their L_Amax came within 0.04 dB on the pavement and in the garden and 0.17 dB apart on the balcony.

On the pavement, the balcony and `balcony_2m` the early level changes by at most 0.17, 0.16 and 0.13 dB(A) from one 10 cm to the next, and in the garden by at most 1.39 dB(A), where 1 kHz swings by 2.5 dB at x = 28.8 m and does so smoothly 1 cm apart; without the fade the garden's level steps by up to 12.1 dB(A) from one 10 cm to the next. The steps left that stand out by more than 0.1 dB in some band 1 cm apart, but at 8 kHz, where a centimetre is a quarter of a wavelength and the level swings by up to 0.75 dB from one to the next (five more at 4 kHz in the garden, at x = 19.40, 19.51, 28.69, 28.72 and 28.88 m, are swings too: in a one-off check 1 mm apart the level there changed by at most 0.08 dB a millimetre at 4 kHz), each the step from 1 cm before x to x, and how far it stands out in dB(A) and in the same band without the fade (`validation/walks.py`):

car at x (m)	listener	stands out (dB)	in	dB(A)	without the fade (dB)
-41.39	garden	1.11	1 kHz	0.55	0.00
-39.92	balcony	0.16	63 Hz	0.00	0.00
-14.07	garden	0.80	125 Hz	0.31	0.00
9.99	garden	0.83	63 Hz	0.04	0.97
10.01	balcony_2m	0.10	125 Hz	0.00	0.08
10.02	garden	1.05	63 Hz	0.12	0.96
14.28	balcony_2m	0.17	125 Hz	0.00	0.04
15.98	balcony_2m	0.16	125 Hz	0.00	0.03
16.01	balcony_2m	0.11	125 Hz	0.00	0.03
18.93	balcony_2m	0.17	125 Hz	0.03	0.00
25.31	balcony	0.14	63 Hz	0.00	0.00
37.20	garden	0.52	500 Hz	0.02	0.00
53.96	pavement	0.13	250 Hz	0.00	0.04
60.03	balcony	0.30	63 Hz	0.05	0.00
60.19	balcony_2m	0.33	63 Hz	0.05	0.00
60.70	pavement	0.29	250 Hz	0.00	0.00
63.10	pavement	0.21	63 Hz	0.00	0.00
65.34	balcony_2m	0.22	500 Hz	0.09	0.00
66.62	balcony	0.15	63 Hz	0.06	0.00

The paths over the barrier's top and round its end at x = 44 m, the edge diffraction's own and the fade's terms of the same paths, fade out past the ends, as the barrier is a thin screen. The steps left are terms that still stop where the point they diffract at leaves the end of its edge (see Known issues), where traced (all but those at 9.99 to 10.02, 14.28, 53.96 and 63.10 m): mostly the fade's terms of second-order reflections that the barrier's top (x = 14 to 44 m), its end or a corner of the offices or the houses cuts off, which the edge diffraction (`first_order`) has no path for, and at 15.98 m the path over the houses' eaves, a solid wedge, at their end. The largest in dB(A), at -41.39 m, is the term of a reflection off the houses' front and the offices' façade that reaches the top's end at x = 14 m. The dips at 9.99 and 10.02 m are the next paragraph's. `tests/test_walks.py` walks the garden round three of the largest steps left and the one at 60.80 m that the fade past the barrier's end took away in CI, and holds them to 1.5 dB in a band and 0.6 dB(A).

At x = 10.0 m, where the car crosses the plane of the houses' end wall, the sound over the houses' corner dips by up to 10 dB at 63 Hz within 2 cm (1.0 dB in the garden's level over 1 cm). It is the edge diffraction's own term, with or without the fade: it gives the wall's reflection the factor of a plane wave, which for the brick's impedance turns to -1 at grazing incidence, so that the reflection cancels the incident sound there. With a rigid wall the dip is gone. A face with an absorption coefficient does the same in the bands that add in pressure, which take only the phase of that factor: it is turned over while the incidence cosine is below the fitted admittance, so for a source (not a listener: the path is then not the same both ways round) that close to the face's plane. A source 0 to 1 cm above the plane of the top of the 1 m block of `tests/test_diffraction.py` (absorbing 0.02, admittance 0.0026), 3 m in front of it, hears the far corner 14.9 to 18.7 dB below the reversed path, and the paths over both corners (`second_order`), which continue that path there, 2.9 to 6.1 dB below; from about 1 cm up the two agree again. Each of these reflections turns over at its own cosine: the specular one off the top at its reflection point, those of the paths over each corner at the source's cosine to that corner. With the source and a listener rising together 3 m either side of that block, the level at 63 and 125 Hz is therefore above the rigid block's from the height where the near corner's turns to where the far corner's does: by up to 16 dB with only the top absorbing 0.2 (9.2 to 12.3 cm, the most at 10.8 cm) and 9.6 dB at 0.5 (31.2 to 41.6 cm), with `second_order` (14.9 and 9.2 dB with `first_order`). The reflection off the ground takes the factor of a spherical wave instead (see Ground reflection), as does the wave along a face from one edge to the next on a path over both; whether that factor would also be the better one here is not checked.

Where the source or the listener rises through the plane of an absorbing top, the level steps. Below the plane only the paths over both corners reach, and they take the ground wave's loss along the top; above it the path over the far corner alone takes none, as its reflection off the top takes the plane wave's factor (in the bands that add in energy, its specular share alone). Over the 1 m block, with the source 3 m in front and the listener 3 m behind and 1.5 m below the top or the other way round, from 1 mm below the plane to on it, the level steps by up to 2.2 dB at absorption 0.1 and 12.9 dB at 0.5 (at 8 kHz), and over a block 10 m wide by 7.1 and 28.7 dB; before the paths over both corners took the ground wave these steps were at most 0.47 and 2.58 dB. A listener steps as much in pressure; a source is then in the dip above. The paths over both corners, weaker by the ground wave, also take less of the far corner's wave away above the plane: with absorption 0.5 and the source in it, the level at 8 kHz is 3.5 dB above the rigid block's in energy. Giving that reflection the spherical wave's factor as well would change every path over one edge with an absorbing face, and is not done.

A reflection off a face with an absorption coefficient takes, in the bands that add in pressure, the square root of (1 - the absorption) with the phase of the plane wave's factor, whatever the size of the face, and near grazing incidence that is wrong for a large face as for a small one. With a source and a listener 0.1 m above a rigid floor 600 m wide and 10 m apart (incidence cosine 0.020), and a square absorbing 0.5 (fitted admittance 0.10) centred on the specular point, the phase is turned over and the engine gives -10.7 dB against the free field at 63 and 125 Hz for a square 1 m, 80 m or 400 m wide, as by hand (the direct sound and a reflection of the square root of 0.5 turned over, -10.66 dB). The exact solution over an unbounded plane of that material gives +3.3 and +2.2 dB, and first-order (Born) theory, the square's own scattering added to the rigid floor's field, +6.0 dB over the 1 m square (and 3.5 and 2.8 dB over the 80 m one). Moved sideways until the reflection point leaves the 1 m square (from 0.49 to 0.51 m off the path), the engine's level steps by 16.7 dB, to the rigid floor's +6.0, where Born theory gives +6.0 dB on both sides.

Known issues of the edge fade

- **Paths that several events stop.** A path that is not found is followed where one edge explains each run of the events that stop it, or two edges explain them between them, in at most two runs. Elsewhere it is left out with its terms, as where one of its reflection points slides into a corner of the room while the path over the edge still reflects on the face, which makes the steps of up to 0.08 dB left round the gallery's corner (see above). The edge diffraction's own paths over an edge with a reflection before or after it stop where that reflection leaves its face, which the fade makes up for where it follows the path. A leg that two obstacles block, or two sides of one, is one event, which no one edge explains: behind two barriers 4 m apart, a U-shaped or a Z-shaped one, on a rigid ground, the level steps by up to 2.6 dB in a band from 63 to 250 Hz, along the one line of listeners measured, where the reflection off the ground crosses the first one's foot, and behind the U by 2.0 dB (`second_order`; 9.4 dB with `first_order`) where it lies just behind the first side's foot and the path round that side's end leaves its foot.

- **Corners of edges.** Over the edge of a thin screen (a barrier's top and ends, a face's free edge) a path's term takes the part of its Fresnel zone that the edge covers, as BTM's integral over a finite edge has it (at a free end, a bend or a corner that is not square, each of its terms weighted for the term's boundary), so it fades out past a free end (see Edge diffraction against the exact solution). Elsewhere a term stops where the point it diffracts at leaves the end of its edge, as the UTD has no terms for the corners of edges, unless that end lies in a plane the path reflects off next to the event, where the edge goes on as its image in the plane (for a leg that stops a path, the image's term is found on the leg on the other side of the reflection): at the edges of solid wedges, on paths over two edges (`second_order`) and for the fade's terms of paths the edge diffraction does not have, such as second-order reflections that an edge cuts off. Near a solid block's corners the level steps by up to 2.4 dB at 63 Hz and is up to 2.3 dB from BTM (see above); along the street such terms step by up to 1.11 dB in one band and 0.55 dB(A) (see above). A path over two edges stops likewise where it leaves the end of one of them, while the paths over each edge alone fade out past the end: just over the tops of two barriers in a row, the source and the listener both within a centimetre of the line of the tops, the level steps by 3.5 dB at 63 Hz and 5.5 to 6.2 dB from 1 kHz up where the way passes the end of one of the two (by 3.5 to 4.3 dB up to 1 kHz with both 0.1 m above the line, and by 2.8 dB at 63 Hz and 0.5 dB or less from 1 kHz up with both 0.5 m above it; a one-off check). At the end the level is, by symmetry, midway in pressure between those over one barrier and over both: from 2 kHz up the engine is 4.1 dB below that just outside the end and 2.0 dB above it just inside. Over the end of a barrier that stands oblique to the way, the point where the path over both tops touches that top moves along it with the height above the tops: with the way through the end of a barrier at 40 degrees to it, 10 m behind a long one, the level is at -6.2 dB (63 Hz) and -8.5 dB (8 kHz) up to 2.5 mm above the tops, as over a square end, and from 3 mm up at -4.8 and -6.0 dB with the barrier turning one way and at -7.7 and -11.8 dB with it turning the other (at 70 degrees from 1 mm up). The fade's terms of a reflection that an edge cuts off belong to the reflections the source's rays find, so those of the reflection off a second screen and back off the first are there with the source on one side and not with the two swapped: level with the tops of two screens 10 m apart, a centimetre beside where the first one's top drops by a metre, swapping them changes the level by 0.4 to 0.6 dB with `second_order` and, with `first_order`, whose level there is 17 to 39 dB down, by up to 8.2 dB. Likewise just above a flat top and a screen level with it close past it, where the reflection between the top's wall and the screen counts: swapping the source and the listener changes the level up to 500 Hz by up to 0.4 dB a centimetre above their plane with the screen 0.5 to 2 m past the top and a tenth of a metre above it with the screen 0.1 m or less past it, and by up to 0.7 dB a metre above it. With `method: image_sources`, which tests every sequence of reflections, the two ways round agree within 0.05 dB there (0.25 dB a tenth of a metre above with the screen 1 cm past the top). They do not past the end of a screen 5 m before a top 5 m wide and level with it, a centimetre above their plane and 1 or 10 cm inside the screen's end: one way round the fade follows the reflections off the top's wall with their copies in the top's plane, the other way without them, and the levels are 0.7 to 1.3 dB apart up to 500 Hz (2.3 to 4.0 dB with `first_order`). A model of those corners and terms is an open item on the roadmap. `tests/test_walks.py` holds the garden's four largest steps to 1.5 dB in a band and 0.6 dB(A).

- **Terms that cancel, in the bands that add in energy.** Indoors paths add in energy from 250 Hz up (`coherent_max_band_hz`). Where both edges of a narrow face border its reflection, their terms nearly cancel there at low frequencies; added in energy they would add up instead. The terms are therefore kept as waves (`engine.clusters` in the API reference): on the lit side of its boundary a term continues its geometric wave as far as it is coherent with it over the band, as it arrives later by its detour, and on the dark side, where that wave is gone, by the UTD's share, so that deep in a shadow all the terms of an edge are one wave; the rest of the term is part of its event's diffracted wave, with the terms of the event's other edges and the edge diffraction's paths there. The products of a path's terms split term by term, the events the path gets past included. A wave's energy is that of its pieces added up in pressure, averaged over each band's frequencies and the slices they stand for, never below nothing; the wave of a found path is the path's, and where its pieces' cross terms take more than it has, the rest comes off them in proportion. Walking the `archway`'s listener behind the wall 1 cm at a time (x = 7.6 m, y = 5.3 to 7.8 m), where the reflection off the arch's jamb, 40 cm deep, appears, the level steps by at most 0.08 dB from 250 Hz to 1 kHz, between y = 5.39 and 5.40 m (see the next item), and by at most 0.16 dB from 2 kHz up. The edge diffraction's own paths, without the fade, step there by up to 0.73 dB at 63 Hz, 1.46 dB at 125 Hz and 1.88 dB from 250 Hz up. Averaged over the walk, in a one-off check, the level was 0.15 dB below that with every band in pressure (`coherent_max_band_hz: 8000`) at 250 Hz, 0.67 and 0.58 dB above it at 500 and 1000 Hz and 0.11 to 0.43 dB below it from 2 kHz up. Before the paths over two edges along an absorbing face took the ground wave, which in pressure they had all but lost, it was 0.39, 1.12 and 0.73 dB above it at 250, 500 and 1000 Hz; the rest of the difference is not traced.
- **Cut off by a thick end.** Behind the `archway`'s wall the level steps by 0.56 dB at 63 Hz and 0.27 dB at 125 Hz between y = 5.39 and 5.40 m (at 63 Hz by 0.28 dB more to 5.41 m), and by 0.08 dB or less from 250 Hz up, as it did with every band in pressure in a one-off check. Reflections that the end of the arch's jamb cuts off are followed while the leg they are cut off on hits the end's face, and left out once it hits the jamb's face on the far side, 40 cm on, which is a face of the end's other corner and not of the edge whose shadow they are in. Letting that other corner explain them removed the step but made one of 1.75 dB at y = 5.90 m, where a reflection point of one of those paths also leaves its face and the path, stopped at two events, is left out at once; so it is not done. The other steps along the walk, of up to 0.20 dB at 63 and 125 Hz and 0.07 dB from 250 Hz up, were there in pressure too (a one-off check); at y = 6.83 to 6.84 m and 7.00 to 7.01 m, terms of cut-off second- and third-order reflections come or go, and why is not traced. The fix would take a term over both corners of the end in turn, as `second_order` has for the direct sound, which the fade does not have (an open item on the roadmap).
  `tests/test_walks.py` walks the whole line in CI and holds the steps to 0.65 dB at 63 Hz, 0.5 dB at 125 Hz, 0.1 dB from 250 Hz to 2 kHz and 0.2 dB above.
- **Products of terms.** The terms of a path's events multiply, as the path must get past each in turn, an approximation for terms at different edges. Each product is a path that arrives later by the detours of all its terms, an approximation too for terms bent far from their boundaries; products below -60 dB of their path are left out in the bands where paths add in pressure. Where two events meet the same edge, the product is a wave bent twice by one edge, which the exact solution does not have. It is not checked on its own; behind the corner of the tests' L-shaped room such products come 23 dB below the paths over the corner.
- **Order of reflections.** Where the edge diffraction does not bend it, a path of the highest order the early stage follows takes no terms for the reflections off the edge's faces, behind their planes or in front, as the early stage follows no path with one more reflection: near an edge the level is that of the order chosen, as for the specular paths.

- **What is checked.** Against exact solutions: vertical edges standing on a rigid ground (a screen's end, a building's corner, a 2 m strip and a 4 m gap) with `direct`, and the corner and the strip with `first_order` too; a barrier's top on a rigid ground, with a reflection off the ground before or after it, with `first_order` and with `both_sides`; and a thick barrier with `second_order`, whose paths over two edges take no terms of the fade. Only for steps, with no exact level: paths of two or more reflections indoors (the gallery, and with `second_order` behind the archway) and in a street, walked in CI (`tests/test_walks.py`) and in full by `validation/walks.py`. Not checked: surfaces with an impedance against an exact solution. Nothing is checked against measurements.

Ground reflection against the exact solution

`validation/ground_reflection.py` compares the reflection off a porous ground (see Outdoor scenes) with the exact solution for a point source over a locally reacting plane, Sommerfeld's integral as Chien and Soroka (1975) write it, which the script integrates numerically. The engine reflects with the Weyl-van der Pol factor Q, the integral's asymptotic form. Levels are those of the direct sound and its reflection together, in dB over the direct sound in free field. The grounds follow Miki's model with Nord2000's flow resistivities; the source and the listener are 2 and 1.2 m high 5 m apart, and 1 and 1.5 m high 20 and 100 m apart.

Q against the exact solution, the largest difference in level at the third-octave centres from 50 Hz to 5 kHz (in brackets, in dips more than 20 dB deep):

ground (flow resistivity, kPa s/m²)	5 m	20 m	100 m
snow (12.5)	0.16	0.19	0.12
snow 10 cm deep on ice (12.5)	0.25	0.36 (2.2)	0.41 (0.8)
grass (200)	0.02	0.04	0.04
gravel (2000)	0.00	0.01	0.01
asphalt (20000)	0.00	0.00	0.00

Q is furthest off over the thin layer of snow, the most reactive of these grounds, and by more than half a decibel only in its deepest dips.

Extended reaction. `--extended` checks grounds whose impedance changes with the angle of incidence (`impedance.reaction: extended`, see Ground impedance): sound also travels sideways inside them, and a plane wave meeting them at the angle θ from the normal meets the impedance Zc n / sqrt(n² - sin² θ), n the ground's complex index, and for a layer on a hard backing that times coth(j k d sqrt(n² - sin² θ)). The exact field of a point source over such a ground is the Sommerfeld integral with that reflection at every horizontal wavenumber, which the fast field program (`validation/ffp.py`, exact in still air, see Outdoor sound in wind against a fast field program) integrates. The engine reflects with Q taking the impedance at the reflection's own angle; the table gives the largest difference in level from the exact field at the third-octave centres from 50 Hz to 2 kHz, outside dips more than 20 dB deep, for that and for Q with the locally reacting impedance, and how much the extended reaction tells (the exact fields of the two reactions apart), the source 1 m and the listener 1.5 m high:

ground (flow resistivity, kPa s/m²)	20 m: Q at the angle	Q local	reaction tells	100 m: Q at the angle	Q local	reaction tells
fresh snow (5)	0.44	0.94	0.96	0.11	0.65	0.66
snow (12.5)	0.22	0.59	0.59	0.13	0.39	0.38
snow 10 cm deep on ice (12.5)	1.46	6.36	3.74	0.37	2.05	2.45
forest floor (31.5)	0.14	0.35	0.35	0.09	0.29	0.29
grass (200)	0.06	0.09	0.10	0.04	0.09	0.11

The extended reaction tells by a tenth of a decibel over grass, by 0.3 to 1 dB over snow and forest floors and by 2.4 to 3.7 dB over the thin layer of snow, and Q with the impedance at the angle follows it to within half a decibel but over the layer, where it still misses the exact field by 1.5 dB at 20 m: Q is the locally reacting plane's spherical wave with the impedance swapped in, and the ground wave over a layer of extended reaction is not quite that (Li, Attenborough and Heap 1998). The parabolic equation, which takes the impedance at grazing incidence, against the same exact field at 63 Hz to 2 kHz:

ground	100 m	400 m
fresh snow (5)	0.06	0.05
snow (12.5)	0.35	0.05
snow 10 cm deep on ice (12.5)	0.84	1.05
forest floor (31.5)	0.37	0.27
grass (200)	0.12	0.16

The engine's early part over grass (image sources to first order, the direct sound and its reflection added in pressure, as outdoors by default) against the exact solution: the paths' energy in each octave band, their interference averaged over the band (`coherent.total_energy`), against the exact solution's energy averaged over the same band:

over grass, exact (dB)	63	125	250	500	1000	2000	4000	8000
5 m, heights 2 and 1.2 m	3.46	-2.07	1.21	1.10	0.89	0.80	0.24	0.25
20 m, heights 1 and 1.5 m	5.46	4.00	-0.49	-5.28	2.46	0.19	1.65	1.93
100 m, heights 1 and 1.5 m	5.49	2.75	-6.86	-16.69	-7.22	-0.15	4.61	2.93

over grass, engine - exact (dB)	63	125	250	500	1000	2000	4000	8000
5 m, heights 2 and 1.2 m	-0.01	+0.00	-0.00	-0.01	-0.01	-0.01	+0.01	-0.03
20 m, heights 1 and 1.5 m	-0.01	-0.02	-0.03	+0.01	+0.00	-0.00	-0.00	-0.01
100 m, heights 1 and 1.5 m	-0.01	-0.01	-0.07	+0.01	+0.00	+0.00	-0.00	-0.00

The ground dip near 500 Hz deepens with distance, as the reflection comes in nearer grazing. These are band averages. The level a run reports reads the bands up to 500 Hz through analysis filters instead, whose skirts reach into the neighbouring bands, and there a sharp dip fills in by up to about 0.9 dB more (the grass table in Outdoor results against exact solutions).

The parabolic equation that gives the sound over the open ground in wind and temperature gradients (see Outdoor scenes), here in still air against the exact solution (`--parabolic`): the largest difference in level at the third-octave centres from 63 Hz to 2 kHz, from a source 1 m high to a listener 1.5 m high, and from a source 119 m high (a wind turbine's hub) to one 1.7 m high:

ground	25 m	100 m	400 m	178 m, from 119 m	535 m	845 m
snow (12.5)	0.64	0.38	0.39	0.26	0.02	0.04
snow 10 cm deep on ice (12.5)	1.81	0.97	1.05	0.41	0.06	0.08
grass (200)	0.13	0.14	0.16	0.28	0.02	0.05
gravel (2000)	0.12	0.15	0.19	0.45	0.02	0.10
asphalt (20000)	0.17	0.05	0.09	0.26	0.04	0.02

It is furthest off over the softest grounds near a low source at low frequencies (the 1.81 dB in a dip 36 dB deep at 160 Hz; the script prints where each row's largest difference falls), where the field it starts from, which stands for the source and its image, matters most. From the hub, 178 m away, the direct sound comes in 33 degrees from the horizontal and the reflection 34, a little steeper than the engine uses the equation for: it takes it from 1.5 times the source's and the listener's heights together (181 m here), where the reflection comes in at 33.7 degrees or flatter, and rays nearer. So the 178 m column shows the equation just outside its range, not what a run gives there; from 535 m it is within 0.1 dB.

The wave solver fits the ground's admittance with branches without mass (see Simulation). Over grass, against a rigid ground, with the source and the listener 1 and 1.5 m high 10 m apart, from the first 25 ms after the reflection, before anything off the grid's edges arrives (the solver up to its 250 Hz band):

Hz	40	63	80	100	125	160	200	250	315
exact (dB)	-0.17	-0.43	-0.66	-0.98	-1.44	-2.20	-3.21	-4.73	-7.11
wave - exact (dB)	+0.02	-0.02	+0.01	+0.01	+0.02	+0.02	+0.05	+0.10	+0.12

Outdoor propagation against measurements (Hannover)

The Hannover wind turbine propagation data (Könecke, Hörmeyer, Bohne and Rolfes, 2021, DOI 10.25835/0012136, CC BY-SA 3.0) measured a wind turbine 178, 535 and 845 m away, with microphones 1.7 m above a field on one line, in ten spells of five minutes in different weather, and gives the loss from the nearest microphone to the two others in the third-octave bands from 80 Hz to 2 kHz, with the inputs and the result of the authors' parabolic equation model. `validation/outdoor_propagation.py` downloads the ten cases and builds each as the authors modelled it: three point sources of equal power 71, 119 and 167 m up across the rotor, a flat ground of Miki's impedance (200 kPa s/m²), the air measured 53 m up, and their effective speed of sound c(z) = a0 + alog ln(1 + z / z0) + alin z as a measured atmosphere, the log term as the wind and the linear one as the temperature. The scene runs in third-octave bands with this package's defaults outdoors, the three sources' energies added at each microphone.

Overall is the energy sum of the fifteen band losses, as the data gives it (so it reads about 15 dB above any one band); then the mean absolute difference from the measured loss over the bands. Each pair is at 535 m / 845 m, in dB:

case	weather	overall, measured	published model	this package	mean difference, published model	this package
1	light downwind	22.6 / 27.5	21.6 / 26.6	23.4 / 28.9	1.8 / 3.2	1.8 / 2.3
2	light downwind	22.2 / 26.7	21.6 / 26.8	23.4 / 29.0	1.6 / 2.8	1.9 / 2.7
3	cross/downwind	20.8 / 26.5	21.4 / 26.8	23.1 / 28.7	1.8 / 2.7	2.2 / 2.7
4	cross/downwind	20.6 / 26.3	21.3 / 26.9	23.1 / 28.6	2.0 / 2.7	2.4 / 2.7
5	cross/upwind	21.0 / 26.6	21.0 / 25.9	23.0 / 28.2	1.9 / 2.5	2.0 / 2.3
6	cross/upwind	20.6 / 25.7	20.9 / 26.0	22.9 / 28.2	1.7 / 2.8	2.3 / 2.7
7	light upwind	21.9 / 26.2	20.7 / 24.9	22.7 / 27.7	2.0 / 2.2	2.2 / 2.2
8	light upwind	22.2 / 27.0	20.5 / 25.1	22.7 / 27.9	2.2 / 2.2	2.3 / 2.1
9	strong upwind	23.1 / 27.4	19.9 / 52.3	22.1 / 27.8	3.2 / 8.3	2.0 / 2.6
10	strong upwind	23.1 / 27.2	19.9 / 50.0	22.1 / 27.9	3.1 / 7.7	2.0 / 2.5

In the eight cases other than strong upwind (light winds, and cross winds in cases 3 to 6), this package is about as close to the measurements as the authors' model: 1.8 to 2.4 dB per band on average at 535 m, 2.1 to 2.7 dB at 845 m, against their 1.6 to 2.2 and 2.2 to 3.2 dB; the spread of the measurements over the five minutes is 1 to 3 dB. From 80 to 160 Hz this package loses more than was measured at 845 m, 2.6 to 6.0 dB on average over those bands (at 535 m from 0.1 dB less to 4.3 dB more), which puts its overall loss at 845 m 0.9 to 2.5 dB above the measured one. With the same inputs the authors' model loses 3.4 to 5.1 dB less than this package in those bands at the two distances, from 4.3 dB less to 1.3 dB more than was measured. This package's parabolic equation matches the exact solution in still air from the hub within 0.1 dB from 535 m (see Ground reflection); what parts the two models there is not traced. These winds bend the sound little over these distances: in still air (`--still`) the overall losses move by 1.1 dB at most and the mean differences by 0.4 dB at most, except downwind at 845 m in cases 1 and 2, which still air matches better (1.4 and 1.8 dB): there the wind takes 1 to 3 dB more than still air from 100 to 160 Hz and up to 2 dB less at 250 Hz, as the dip of the ground moves down in frequency when the sound comes in steeper. The field is flat and smooth, as the authors modelled it, so its material scatters nothing: with the few late rays that get this far, random scattering would only add noise.

Cases 9 and 10 are strong upwind. The published model puts 845 m deep in a shadow, 50 dB overall where 27 dB was measured, and the authors put the gap down to turbulence, which scatters sound into a shadow and which neither model has. Here the engine's rays in the same profile still reach the microphones' height at 845 m from the two upper sources: in case 9 they reach it up to 690 m from the source 71 m up, but up to 980 and 1230 m from those 119 and 167 m up (the script prints this for every case with a shadow), so only the lowest one is in shadow there, and the loss stays within about 1 dB of the measured one overall.

The data gives no measure of the eddies. With `--turbulence moderate` the overall losses drop by up to 0.4 dB at 845 m and 0.2 dB at 535 m, as the eddies fill the ground dip a little; the mean differences move by 0.3 dB at most, better at 845 m in the nine other cases and 0.2 dB worse in case 8. In cases 9 and 10 the overall loss at 845 m comes to 27.4 and 27.5 dB, against 27.4 and 27.2 measured.

Turbulence against a parabolic equation through random eddies

`validation/turbulence.py` marches the engine's own parabolic equation (checked against the exact solution, see Ground reflection) through random eddies and averages its sound's energy over 16 of them (Gilbert, Raspet and Di 1990): each is a field of the index of refraction drawn with the von Kármán spectrum, outer scale 10 m, in the vertical plane through the source and the listener. Both checks fitted constants of the engine, so they show how well those fits hold over the cases rather than test it independently.

The ground dip. In still air with the `strong` eddies, the level of the direct sound and its reflection off the ground, 50, 100, 200 and 400 m away at 13 frequencies from 125 Hz to 2 kHz, against the parabolic equation's: the engine's coherence factor, and the Markov approximation's alone (Clifford and Lataitis 1983), which fills the dip over soft grounds far more than the parabolic equation does. Mean (largest) absolute difference, in dB:

ground	source / listener (m)	engine	Markov alone
grass	2 / 1.5	0.75 (3.2)	2.57 (13.6)
grass	5 / 1.5	0.64 (3.8)	1.61 (7.1)
loose soil	1 / 1	1.29 (4.6)	3.55 (14.1)
gravel	2 / 1.5	1.27 (7.0)	2.06 (11.7)
grass	10 / 2	0.58 (2.1)	0.93 (5.8)
asphalt	10 / 2	0.92 (3.5)	0.86 (3.7)
asphalt	5 / 1.5	0.93 (7.2)	0.91 (7.7)
compacted soil	3 / 3	0.75 (3.5)	1.49 (7.8)

Over the four soft grounds (grass and loose soil) the engine is 0.8 dB off on average, against 2.2 for the Markov approximation alone; over the four harder ones (compacted soil, gravel and asphalt) 1.0 against 1.3.

Into a shadow. Over grass, the source 2 m and the listener 1.5 m up, on a sunny day and upwind in the `windy` preset, deep in the shadow (2.5 to 9 times as far as it starts), from 250 Hz to 2 kHz: the parabolic equation's level through the eddies against the engine's, which is the parabolic equation's in still air plus what the eddies scatter into the shadow. Mean (largest) absolute difference:

atmosphere	shadow from	`moderate` eddies	`strong` eddies
sunny day	56 m	0.8 dB (2.9)	1.2 dB (3.5)
windy, upwind	31 m	0.7 dB (2.8)	0.9 dB (2.5)

The farthest points with `moderate` eddies, in dB over the free field:

Hz	sunny day 503 m, still air	through eddies	engine	windy 281 m, still air	through eddies	engine
250	-33.3	-27.1	-25.9	-25.8	-25.7	-24.8
500	-49.2	-20.4	-21.8	-61.1	-26.3	-24.8
1000	-68.2	-13.4	-14.1	-64.9	-24.3	-21.5
2000	-61.9	-11.0	-11.6	-58.4	-19.7	-21.1

At these points, without the eddies the shadow sinks 49 to 68 dB below the free field from 500 Hz; through them it stays 11 to 26 dB below, and the engine follows within 3 dB. Only these heights and these two profiles were checked.

Eddies weaker with height. `--profile` runs both checks with the structure parameters given 2 m up (`height_m`, see Turbulence) and falling as z^(-2/3): each field is scaled in height by the square root of that, rising right down to the roughness length 5 cm above the ground, where it is 12 times as strong as at 2 m, while the engine takes Cn² no nearer the ground than the eddies' outer scale, 10 m (0.34 of the value at 2 m). The ground dip, mean (largest) absolute difference in dB, with a third column for the engine taking Cn² at each pair's mean height right down to the ground instead:

ground	source / listener (m)	engine	Markov alone	engine, Cn² down to the ground
grass	2 / 1.5	1.26 (8.0)	2.56 (12.2)	2.34 (9.9)
grass	5 / 1.5	0.89 (5.9)	1.38 (6.4)	1.15 (6.1)
loose soil	1 / 1	0.90 (4.9)	3.18 (14.3)	2.50 (9.1)
gravel	2 / 1.5	1.79 (8.3)	2.16 (9.4)	2.49 (10.5)
grass	10 / 2	0.71 (4.2)	0.81 (3.8)	0.62 (2.9)
asphalt	10 / 2	1.58 (11.3)	1.38 (8.7)	1.17 (9.2)
asphalt	5 / 1.5	1.60 (10.3)	1.52 (8.6)	1.45 (7.0)
compacted soil	3 / 3	1.10 (4.0)	1.48 (8.0)	1.37 (6.7)

Over the four soft grounds the engine is 0.9 dB off on average (0.8 with eddies the same at every height, 1.7 taking Cn² down to the ground), over the four harder ones 1.5 (1.0 and 1.6). Taking Cn² right down to the ground puts the engine further off over the soft grounds and no nearer over the harder ones, so the engine keeps the outer scale, one rule for the coherence, the scattering and the shaking alike. Into the shadows, with the same profile:

atmosphere	shadow from	`moderate` eddies	`strong` eddies
sunny day	56 m	1.3 dB (4.2)	1.7 dB (5.8)
windy, upwind	31 m	1.9 dB (8.7)	2.9 dB (12.2)

Deep in the shadow the eddies that scatter and shake sound in are those over the limiting rays, tens of metres up, where the profile leaves a third to a seventh of the strength at 2 m: the parabolic equation's level through these eddies is 6 to 10 dB (sunny day) and 4 to 7 dB (upwind) below that through eddies the same at every height at the farthest points from 500 Hz, and the engine follows within 1.3 to 2.9 dB on average (2.8 to 3.3 with Cn² at the geometric mean of the source's and the listener's heights right down to the ground). Its largest misses are at the nearest points upwind at 2 kHz, where the field's strong layer at the ground brings the parabolic equation back within 4 dB of the free field with the `strong` eddies (11 with eddies the same at every height) while the engine, stopping at the outer scale, stays 16 below: near the shadow's edge the shaking does feel the eddies right at the ground, and the engine is low there.

Vegetation against a particle simulation

`validation/vegetation.py` checks the late rays through vegetation against a particle simulation written apart from the engine, in numpy: a source in the middle of a box 200 m across whose walls absorb everything, a slab of plants from x = 10 to 40 m across it, and three receivers, spheres 10 m in radius 50 m from the source, through the slab (+x), beside it (+y) and behind the source (-x). The particles go straight to their next scattering, drawn at the medium's rate, and leave in a direction drawn from its phase function. Both count the energy in the spheres by the length of track through them, and as the engine's rain does: next-event estimates from every scattering in foliage, and from every scattering off stems the track the next leg would have through each sphere, averaged over the turns. Each run also sends the same particles through empty air, and the table gives its energy less that run's plus the exact free field's, so the direct sound's own randomness drops out. Energy over the free field's at 50 m, mean ± standard error of 16 runs of a million particles:

medium	receiver	reference, spheres	engine, spheres	reference, rain	engine, rain
none (exact 1.008)	through	1.008 ± 0.002	1.010 ± 0.002	-	-
none (exact 1.008)	beside	1.003 ± 0.003	1.011 ± 0.003	-	-
none (exact 1.008)	behind	1.009 ± 0.002	1.006 ± 0.002	-	-
isotropic	through	0.997 ± 0.002	0.996 ± 0.003	0.990 ± 0.002	0.988 ± 0.002
isotropic	beside	1.418 ± 0.002	1.420 ± 0.002	1.418 ± 0.000	1.418 ± 0.000
isotropic	behind	1.204 ± 0.002	1.203 ± 0.001	1.202 ± 0.000	1.202 ± 0.000
forward	through	0.737 ± 0.002	0.734 ± 0.003	0.730 ± 0.001	0.732 ± 0.002
forward	beside	1.103 ± 0.001	1.102 ± 0.001	1.098 ± 0.000	1.098 ± 0.000
forward	behind	1.030 ± 0.000	1.030 ± 0.000	1.030 ± 0.000	1.030 ± 0.000
stems	through	0.685 ± 0.003	0.682 ± 0.002	0.685 ± 0.002	0.682 ± 0.001
stems	beside	1.254 ± 0.001	1.253 ± 0.001	1.254 ± 0.001	1.254 ± 0.001
stems	behind	1.187 ± 0.001	1.185 ± 0.001	1.186 ± 0.000	1.185 ± 0.000
leaves	through	0.947 ± 0.003	0.950 ± 0.002	0.939 ± 0.002	0.943 ± 0.002
leaves	beside	1.529 ± 0.002	1.526 ± 0.002	1.528 ± 0.001	1.527 ± 0.001
leaves	behind	1.252 ± 0.001	1.253 ± 0.001	1.251 ± 0.000	1.252 ± 0.000
two bands, band 1	through	0.994 ± 0.003	1.005 ± 0.004	0.988 ± 0.002	0.991 ± 0.003
two bands, band 1	beside	1.418 ± 0.002	1.421 ± 0.002	1.417 ± 0.000	1.417 ± 0.000
two bands, band 1	behind	1.203 ± 0.001	1.204 ± 0.002	1.202 ± 0.000	1.202 ± 0.000
two bands, band 2	through	0.736 ± 0.002	0.743 ± 0.003	0.732 ± 0.001	0.734 ± 0.003
two bands, band 2	beside	1.101 ± 0.001	1.103 ± 0.001	1.098 ± 0.000	1.098 ± 0.000
two bands, band 2	behind	1.030 ± 0.000	1.030 ± 0.000	1.030 ± 0.000	1.030 ± 0.000

The five media: foliage that scatters every way alike, 0.03 per metre; foliage that scatters forwards (mean cosine 0.6), 0.02 per metre, and absorbs 0.01; stems, 0.03 per metre of horizontal travel, turning sound by a mean cosine of -0.3; leaves of 0.25 m^2 per m^3, 5 cm wide, as the engine works them out in the 4 kHz band (Leaves: 0.0535 per metre scattered with a mean cosine of 0.14, 0.00075 absorbed); and the first two in two bands at once, which the engine traces with one set of rays. The engine agrees with the particles within 1.1 % through the slab (0.4 % with one medium alone) and 0.3 % beside it and behind the source, with the spheres and with the rain: at most two to three of their standard errors, and the particles' own runs differ as much from one set of seeds to another (0.990 and 0.988 through the slab, rain, the same foliage alone and in two bands). In empty air the two differ by 0.7 % beside the slab (1.003 and 1.011), each within 0.5 % of the exact 1.008: the runs' own noise. Next to the slab the rain gives a little less than the spheres in both programs alike (the same with stems), as it takes the scattered sound at the sphere's centre and the spheres its mean over the sphere. Beside and behind the slab the rain is steadier than the spheres. Only a slab was checked, and against a simulation of the same model, not against measurements through trees.

Leaves against closed forms and ISO 9613-2

`validation/leaves.py` checks foliage given as leaves (`foliage: {leaf_area_per_m3, leaf_width}`, see Vegetation), which the engine takes as thin rigid discs turned every way alike. The disc's scattering cross-section over its area against its closed forms: small against the wavelength, the dipole of the air the disc holds back, 16 $(kr)^4$ / (81 π^2) (from a disc's added mass, 8 ρ r^3 / 3, a third of it turned every way alike); large, twice its mean projected area. In between the engine bridges the two, as the exact cross-section of a disc about a wavelength across is not in closed form:

kr	0.1	0.3	1	2	3	5	10	30
dipole	2.0e-6	1.6e-4	0.020	0.32	1.6	13	200	1.6e4
engine	2.0e-6	1.6e-4	0.020	0.24	0.62	0.93	0.995	0.9999
mean cosine	0	0	0.01	0.12	0.31	0.46	0.50	0.50

The leaves' absorption, the viscous and thermal boundary layers on their two faces, matches a rigid wall's in closed form (Morse and Ingard 1968, 6.4) to 1e-12: 3.6e-4 of a leaf's area at 63 Hz, 1.5e-3 at 1 kHz, 4.1e-3 at 8 kHz, as the square root of the frequency.

Then a wood of leaves 50 m deep over a ground that absorbs everything, the source and listeners 1.5 m up, the listeners 5, 20 and 50 m behind it: the wood's insertion loss (the level without it less the level with it, from two runs of the early paths and 10 million late rays per band, the mean of three seeds; the last row of each case is the largest spread between the seeds behind the wood), compared with ISO 9613-2's dense-foliage attenuation for 50 m of foliage (Annex A, its rates per metre for 20 to 200 m), and with what the early paths alone lose (the leaves' extinction):

leaves	63	125	250	500	1k	2k	4k	8k	
ISO 9613-2, dense foliage	1.00	1.50	2.00	2.50	3.00	4.00	4.50	6.00	dB
0.1 m²/m³, 5 cm wide	0.01	0.01	0.02	0.02	0.06	0.45	4.66	16.49	early paths' loss
	0.01	0.01	0.02	0.02	0.05	0.32	2.95	8.58	5 m behind
	0.01	0.01	0.02	0.02	0.05	0.36	3.46	10.23	50 m behind
	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.08	spread over seeds
0.25 m²/m³, 5 cm wide	0.02	0.03	0.04	0.06	0.14	1.11	11.65	41.22	early paths' loss
	0.02	0.03	0.04	0.06	0.12	0.79	6.96	17.37	5 m behind
	0.02	0.03	0.04	0.06	0.13	0.89	8.20	20.41	50 m behind
	0.00	0.00	0.00	0.00	0.00	0.00	0.07	0.17	spread over seeds
0.5 m²/m³, 5 cm wide	0.04	0.06	0.08	0.12	0.29	2.23	23.30	82.45	early paths' loss
	0.04	0.06	0.08	0.12	0.25	1.57	12.60	26.83	5 m behind
	0.04	0.06	0.08	0.12	0.26	1.78	14.75	30.69	50 m behind
	0.00	0.00	0.00	0.00	0.00	0.01	0.13	0.95	spread over seeds
1 m²/m³, 5 cm wide	0.08	0.11	0.16	0.24	0.57	4.45	46.61	164.90	early paths' loss
	0.08	0.11	0.16	0.23	0.49	3.13	20.44	40.50	5 m behind
	0.08	0.11	0.16	0.24	0.52	3.54	23.47	44.39	50 m behind
	0.00	0.00	0.00	0.00	0.00	0.04	0.48	1.65	spread over seeds
0.25 m²/m³, 2 cm wide	0.02	0.03	0.04	0.06	0.08	0.14	0.58	5.92	early paths' loss
	0.02	0.03	0.04	0.06	0.08	0.13	0.45	4.24	5 m behind
	0.02	0.03	0.04	0.06	0.08	0.13	0.49	4.79	50 m behind
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	spread over seeds
0.25 m²/m³, 10 cm wide	0.02	0.03	0.04	0.12	1.08	11.61	41.16	53.29	early paths' loss
	0.02	0.03	0.04	0.10	0.75	6.72	15.00	18.64	5 m behind
	0.02	0.03	0.04	0.11	0.86	7.97	18.03	21.81	50 m behind
	0.00	0.00	0.00	0.00	0.00	0.03	0.27	0.59	spread over seeds

The standard's table is empirical, for the foliage of whole woods, so this compares magnitudes rather than checking against an exact answer. Leaves 5 cm wide at 0.1 m² per m³, about a wood's trunk space with some foliage in it, take about what the table gives at 4 kHz (3.0 to 3.5 dB against 4.5) and more at 8 kHz (8.6 to 10.2 against 6); below 1 kHz they take next to nothing, where the table's 1 to 2.5 dB are the standard's allowance for the whole wood, not what leaves scatter. The loss rises far more steeply with frequency than the table, and with the leaves' area and width, as measurements in dense foliage do (Aylor 1972 measured the loss through reeds and corn rising by several decibels per octave above 500 Hz): a leaf scatters once its size nears the wavelength, so wider leaves take hold at lower frequencies, and leaves 2 cm wide hardly matter below 8 kHz. What the leaves scatter comes back as late sound, so a listener loses far less than the early paths do (the extinction row): with 1 m² per m³ the early paths lose 165 dB at 8 kHz and the listener 40 to 44, the rest arriving scattered. Behind dense leaves at 4 and 8 kHz what arrives is the rare ray that slips through, so the level takes many rays to settle: with 100,000 rays (`--rays 100000`) it swings by up to 22 dB from one seed to another at 8 kHz, with 10 million the seeds agree within 1.7 dB. No measurement of a known leaf area and width was available to check against; the bridge between the disc's two limits, the discs standing in for real leaves, which flex and hang in clumps, and the single phase function per band are the model's assumptions.

The wave solver in wind and temperature against a parabolic equation

`validation/wave_outdoor.py` checks the wave solver's speed of sound per cell against the engine's parabolic equation (itself within 0.2 dB of the exact solution over grass in still air from a source 1 m up, see above), marched in the vertical plane through the source and the listener with the effective speed of sound along it. Both take the same profiles; the wave solver steps the scene in three dimensions. Over grass (Miki's impedance, 200 kPa s/m²), the source 2 m up, the listeners 1.5 and 5 m up, 50 to 200 m away: the 63 Hz band from a grid for that band (8 points per wavelength, 28.6 to 30.2 million cells), the 125 Hz band from one for it (38.1 to 44.0 million cells), up to 100 m. The ground reaches three times the half-width of the reflection's first Fresnel zone at the farthest listener to either side, 20 m behind the source, and 63 m (63 Hz) or 60 m (125 Hz) past the farthest listener: far enough that the sound off its far end, which runs there and back, comes after the end of the run at every listener, even at 350 m/s (the run lasts the farthest distance over 330 m/s, plus 0.3 s). Levels in dB over the free field, the mean square of the pressure over the band's slices, wave solver / parabolic equation:

63 Hz	50 m, 1.5 m up	50 m, 5 m up	100 m, 1.5 m up	100 m, 5 m up	200 m, 1.5 m up	200 m, 5 m up
still	+5.3 / +5.2	+4.4 / +4.4	+5.2 / +5.2	+4.4 / +4.4	+5.0 / +5.0	+4.0 / +4.1
downwind	+5.7 / +5.7	+4.3 / +4.4	+6.3 / +6.3	+4.7 / +4.9	+7.1 / +7.2	+5.1 / +5.4
upwind	+4.8 / +4.6	+4.5 / +4.3	+3.9 / +3.8	+3.8 / +3.4	+2.0 / +1.8	+2.0 / +1.6
sunny day	+5.1 / +5.1	+4.4 / +4.4	+4.8 / +4.8	+4.2 / +4.1	+3.9 / +4.0	+3.3 / +3.3
clear night	+5.5 / +5.4	+4.5 / +4.4	+5.6 / +5.6	+4.6 / +4.6	+5.8 / +5.8	+4.6 / +4.6

125 Hz	50 m, 1.5 m up	50 m, 5 m up	100 m, 1.5 m up	100 m, 5 m up
still	+3.0 / +2.8	+1.0 / +0.8	+2.2 / +1.9	+0.5 / +0.3
downwind	+3.1 / +3.3	-0.3 / -0.2	+2.8 / +2.9	-0.9 / -0.9
upwind	+2.6 / +2.1	+1.8 / +1.4	+0.7 / +0.1	+0.9 / +0.4
sunny day	+2.8 / +2.6	+1.2 / +1.0	+1.7 / +1.4	+0.7 / +0.4
clear night	+3.2 / +3.0	+0.8 / +0.5	+2.5 / +2.3	+0.3 / +0.0

The weathers are the presets: downwind and upwind in the `windy` one (5 m/s 10 m up), `sunny_day` and `clear_night`. The wave solver is within 0.60 dB of the parabolic equation everywhere, 0.10 dB on average at 63 Hz and 0.25 dB at 125 Hz, and follows the weather as it does: 2 dB louder downwind than in still air at 200 m and 1.5 m up, 3 dB quieter upwind (1.1 and 2.1 dB 5 m up). It reads above the parabolic equation more often than below. In still air, against the exact solution over grass, it is −0.01 to +0.10 dB from it at 63 Hz and 0.16 to 0.20 dB over it at 125 Hz, the parabolic equation within 0.05 dB. Part of that is sound off the sides of the scene's ground and off its end behind the source, which reaches the listeners within the run and which neither the exact solution nor the parabolic equation has. In one-off checks with every edge too far for its sound to come within the response (63 Hz: 160 m to either side, 60 m behind, 150 m past, the response cut 300 ms after the direct sound; 125 Hz: 60 m to either side, 50 m behind, the response cut 130 ms after it), the wave solver is 0.03 to 0.07 dB over the exact solution at 63 Hz and 0.09 to 0.13 dB at 125 Hz. Upwind the wave solver reads 0.14 to 0.39 dB above the parabolic equation at 63 Hz and 0.41 to 0.60 dB at 125 Hz, which was not traced further. Neither the deep shadow upwind (it starts a few hundred metres away at these heights) nor reflections off buildings, where the effective speed of sound along the horizontal from the source stands in for the flow, were checked (moving air is, below).

Moving air in the wave solver

`validation/wave_wind.py` checks the wave solver's moving air (`simulation.wave.wind: moving`, see How the simulation bends sound) against exact solutions, and the effective speed of sound beside it, in the 63 Hz band (40 to 90 Hz, its edges a third of an octave wide) from a grid for that band; and the wind it makes flow round the model's solids against an ideal fluid's.

Free field in a uniform wind. A wind of 15 m/s over a rigid ground, the source 2 m up, listeners 1.5 m up 30 m away downwind, upwind, across the wind and at 45 degrees to it. In a uniform wind each path (the direct sound and the ground's image) is the convected Green's function, an impulse of height 1 / R* at (R* - M x) / (c (1 - M²)), R* = √(x² + (1 - M²)(y² + z²)), x along the wind. The grid's own dispersion puts the still air's sound 0.61 ms late along the grid's axes at 30 m (0.15 ms at 45 degrees), its level within 0.01 dB of the exact one along them and 0.25 dB over it at 45 degrees (the 7-point stencil's own gain along a face's diagonal, see Free field), and is the same with and without wind, so each windy run is compared with the still air's run on the same grid, moved by the exact change of its arrival and scaled by R / R*. Level and arrival over the arrival (30 ms before to 50 ms after), and the waveform's error:

15 m/s, 30 m	downwind	upwind	across	45 degrees
level, effective / moving (dB)	-0.78 / +0.00	+0.84 / -0.00	-0.14 / +0.00	-0.63 / -0.01
arrival, effective / moving (ms)	-0.03 / -0.07	+0.03 / +0.09	-0.15 / +0.00	-0.03 / +0.00
waveform's error, effective / moving (dB)	-20.9 / -30.1	-19.6 / -28.0	-23.6 / -65.5	-22.9 / -45.1

Moving air is within 0.02 dB and 0.1 ms of the exact solution in every direction. The effective speed of sound gets the arrival about right but not the level, up to 0.84 dB off: a speed of sound that changes with the bearing is not a moving medium.

The wind round a dome. Where the atmosphere's wind meets a surface of the model, moving air first makes it flow round the solids (see How the simulation bends sound). A rigid half sphere of 4 m radius on rigid ground in a uniform wind of 15 m/s: the wind of the run's own solver, on the faces between its cells over the ground, against the flow of an ideal fluid past a sphere, the gradient of U $(x + a^3 x / (2 r^3))$, which the ground, a plane of symmetry, leaves as it is. The difference across each face in per cent of the wind, its root mean square and (in brackets) its largest:

distance from the dome's centre	63 Hz band's grid (cells of 38 cm)	125 Hz band's grid (19 cm)
within two cells of the dome	8.8 % (58.6 %)	9.2 % (150 %)
from there to 1.5 radii	1.5 % (8.3 %)	0.88 % (9.7 %)
1.5 to 2 radii	0.59 % (2.9 %)	0.37 % (1.6 %)
2 to 3 radii	0.35 % (1.6 %)	0.32 % (1.0 %)
beyond 3 radii	0.28 % (2.1 %)	0.29 % (2.2 %)

Next to the dome the grid's cells make a staircase of it, and the wind is fast round each of its corners: the fastest wind on the grid is 1.55 and 1.64 times the given one, where the ideal fluid's is 1.5 times, over the top. From two cells off the surface the wind is the ideal fluid's to 1.5 % and 0.9 %, and less farther out. Beyond 3 radii the difference no longer falls with the cell: the grid ends 4 radii from the centre along the ground and 3.5 radii up, and the wind's correction with it (a one-off run with the ground 7 radii away and the sky 6.5 gives 0.06 % there and 0.17 % from 2 to 3 radii).

A façade downwind. A rigid building 40 m downwind of the source, its façade across the wind, 30 m high and 50 m wide, and a listener 10 m upwind of the source, both 2 m up: the façade sends the sound back against the wind. In a uniform wind of 15 m/s ray theory gives its arrival (a ray goes at $c\,n + v$ for its wave normal n; off the façade the wave keeps its frequency and its wavenumber along the façade), 1.78 ms later than in still air. Moving air's wind is not uniform there: it slows towards the façade and flows round the building, 13.9 m/s at the listener, 13.2 m/s at the source, 2.8 m/s 5 m before the façade and up to 49.5 m/s past the building's edges. Its reference is the time along the still air's paths through the wind the run's solver made, each piece of a path at the speed of a ray along it, $v \cdot s + \sqrt{(c^2 - |v|^2 + (v \cdot s)^2)}$ (bending the paths changes the time only to the second order; the same sum through the uniform wind gives its exact rays' arrival to within 0.001 ms). Against the still air's run on the same grid:

Façade echo	later than in still air	off by
ray theory, the uniform wind	+1.78 ms	
effective speed of sound	-8.81 ms	-10.59 ms
ray theory, the wind round the building	+1.38 ms	
moving air	+1.39 ms	+0.01 ms

The effective speed of sound sends the echo back at the speed of sound going away from the source: over its 40 m back to the source that makes it 80 v / (c² − v²) = 10.2 ms too soon, and the run gives 10.6 ms. The 0.4 ms between the two is not fully traced: the effective run's grid steps for the fastest sound, c + v, and the still air's for c, so the two disperse differently, which accounts for part of it. Moving air gets the echo 0.01 ms off the rays through its own wind, 0.39 ms sooner than a uniform wind would bring it: before the façade the air is nearly still. That wind is an ideal fluid's, without the wake and the eddies of a real building's, and at the building's edges it is 3.3 times the given wind on this grid, where a real wind separates (see What is not modelled yet).

A wind that grows with height. The `windy` preset (5 m/s 10 m up, logarithmic) over grass, downwind and upwind, the scene of the check above, against the parabolic equation with the effective speed of sound, which holds at these grazing angles. Levels in dB over the free field, wave solver / parabolic equation:

63 Hz	50 m, 1.5 m up	50 m, 5 m up	100 m, 1.5 m up	100 m, 5 m up	200 m, 1.5 m up	200 m, 5 m up
downwind, parabolic equation	+5.7	+4.4	+6.3	+4.9	+7.2	+5.4
downwind, effective	+5.7	+4.3	+6.3	+4.7	+7.1	+5.1
downwind, moving	+5.9	+4.5	+6.5	+4.9	+7.4	+5.4
upwind, parabolic equation	+4.6	+4.3	+3.8	+3.4	+1.8	+1.6
upwind, effective	+4.8	+4.5	+3.9	+3.8	+2.0	+2.0
upwind, moving	+4.6	+4.3	+3.7	+3.6	+1.9	+1.6

Moving air is within 0.26 dB of the parabolic equation, 0.09 dB on average; the effective speed of sound within 0.39 dB, 0.19 dB on average. Downwind the effective speed of sound is 0.02 to 0.29 dB below it and moving air −0.01 to +0.26 dB from it (most at 200 m, 1.5 m up); upwind moving air is within 0.13 dB and the effective speed of sound 0.14 to 0.39 dB above.

Moving air takes 3.9 to 4.9 times as long as the effective speed of sound where its wind is carried as given: 15 s against 3.7 s in the free field (3.2 million cells) and, with the preset (29.2 against 30.2 million cells), 324 and 312 s against 66 and 69 s downwind and upwind. Round the façade's building it takes 12 times as long for the façade's response of 0.45 s, 92 s against 7.6 s (6 million cells): 37 s to make the wind flow round the building (962 sweeps of its solve), which a run does once whatever its length, and its steps 8.3 times the effective speed of sound's. On 4 cores with nothing else running (11 November 2026); run twice more, the façade's case took 94 and 89 s with moving air and 7.9 and 6.9 s with the effective speed of sound.

A room with an opening. A wind blown through a room's walls makes the solver's sound grow there without bound (the scheme keeps the sound's energy only where no wind crosses a surface); made to flow round them, it leaves the room sheltered. A rigid room on rigid ground, 4 × 4 × 3 m inside, its walls and roof 0.8 m thick, with an opening of 0.8 × 0.8 m in its wall downwind, the source and a listener in it, a response of 3 s in the 63 Hz band:

wind (m/s)	fastest wind round the room (m/s)	fastest wind in the room (m/s)	moving air: listener's level, last quarter second over the second (dB)	the same with the effective speed of sound (dB)
5	7.8	0.01	-0.5	-0.4
15	23.6	0.03	-0.8	-0.2
30	46.9	0.05	-1.1	-0.1

Nothing takes the room's sound in but the opening, with either way of taking the wind. The solver still watches each run in moving air: from the end of the source's pulse on, the pressure squared over the grid's air, averaged over pieces of at least a quarter second, may not rise to twice its lowest level so far, or the run stops with an error (a rise that stays under a millionth of the run's largest piece is let be). None of these runs is stopped.

One-off checks of the solver alone, on the 63 Hz band's grid with rigid surfaces over 10 s at 15 m/s, the levels those of the grid's last quarter second under its first: the same room loses 1.1 dB with its opening downwind (1.1 and 1.2 dB at 5 and 30 m/s) and 1.1 dB with it upwind. In the open, a yard closed on four sides (4 × 4 m, walls 4 m high), a street 3 m wide between two buildings 5 m high, a façade downwind of the source and one upwind of it die away steadily, by 68, 73, 131 and 131 dB.

The border of the moving air. The air of the grid's absorbing layer is still, and the wind crosses the border between the two. Through each face of that border the cell's own sound goes out with the wind and silence comes in, which takes energy from the sound and adds none (see the notes of `fdtd.hpp`); carried with the layer's own values instead, the sound grows there in fast winds: in one-off runs over open ground on a grid 4 m tall, the slowest wind in which it did was 53 m/s, at 45 degrees to the grid. The tests hold a wind of 0.3 cells per time step along both axes over open ground (178 m/s along each) and one of 0.3 along the grid past a block, round which it gets to 0.45 (266 m/s): from 1,500 steps on, the sound over the grid stays more than 60 dB under its level over the first 500. In one-off runs of the solver alone on cells of 25 cm, none of 39 runs (28 set-ups, some at several wind speeds) grows over 20,000 to 40,000 steps in given winds of up to 0.45 cells per step (267 m/s): open ground with the wind along the grid, at 15, 30, 45 and 60 degrees to it and rising with height, on grids 2.5 and 4 m tall; a block, one next to the downwind border, stairs, a barrier, walls with a slit and with a hole, a pit, streets along and across the wind, and rooms with an opening downwind, upwind and to the side. Those in the open fall by 106 to 189 dB within their runs: 18 of the 30 until they stop falling, 181 to 189 dB down, and end up to 9 dB above their lowest, 173 to 183 dB down; a run in almost still air on the 4 m grid (1e-7 cells per step) ends 171 dB down, 3 dB above its lowest, which makes that single precision's rounding and not the wind. The other 12 were still falling at their end. The rooms lose 8 to 30 dB through their openings. More than 200 further one-off runs of 3,000 to 16,000 steps (other directions and shapes of wind, other proportions of the grid, solids one to three cells from the border) grow in none either.

A wave that leaves the moving air for the layer's still air comes back in part, by about half the wind's Mach number, and so it does carried with the layer's own values (a one-off run of this check then gives 34 to 42 dB at 10 m/s). The plate of the open model's check in a uniform wind, 25 to 150 Hz on cells of 30 cm, with the default layer against a grid 45 cells larger all round, whose border sends nothing back in the 70 ms of the runs (their last 10 ms faded out): what differs at the three listeners, in dB relative to the sound itself,

wind (m/s)	along the grid	at 45 degrees to it	half the Mach number
0	−74.0, −78.0, −75.0		
5	−39.7, −45.6, −44.2	−37.8, −45.5, −40.0	−42.7
10	−34.0, −39.8, −38.4	−32.0, −39.7, −34.2	−36.7
20	−28.4, −34.2, −32.9	−26.3, −34.0, −28.6	−30.7
40	−23.1, −28.8, −27.8	−21.0, −28.6, −23.2	−24.7

In still air the layer returns 74 to 78 dB less than the sound; in a uniform wind of 5 m/s, the `windy` preset's 10 m up, 38 to 46 dB less. Taking that away needs a layer made for moving air, which the solver does not have (see What is not modelled yet).

Outdoor results against exact solutions

`validation/insertion_loss.py` runs whole outdoor scenes through the results an outdoor run reports, the level relative to free field and the insertion loss (see Outdoor scenes), and compares them with exact solutions. The barrier's scenes take the paths over its edge (`simulation.early.rows: edges`), which those solutions are of; the solver for whole rows has its own check (Rows of roofs and barriers against a wave calculation).

A thin barrier 3 m high, 10 m from a source 1 m up, with listeners 1.5 m up, 2 to 90 m behind it, over a ground that absorbs everything: its insertion loss, from the two runs, against the Biot-Tolstoy-Medwin solution for a half-plane (Edge diffraction), per octave band, and Maekawa's chart for reference:

listener	63	125	250	500	1k	2k	4k	8k	
2 m behind	8.84	11.08	13.67	16.49	19.43	22.42	25.43	28.44	exact, dB
	-0.01	+0.00	+0.01	+0.01	+0.01	+0.01	+0.02	+0.07	engine - exact
	+0.22	+0.06	-0.05	-0.15	-0.23	-0.29	-0.33	-0.35	Maekawa - exact
10 m behind	8.32	10.06	12.16	14.63	17.38	20.29	23.28	26.28	exact, dB
	-0.01	-0.00	+0.00	+0.00	+0.01	+0.01	+0.01	+0.03	engine - exact
	-1.14	-1.35	-1.43	-1.47	-1.54	-1.62	-1.68	-1.71	Maekawa - exact
40 m behind	8.16	9.69	11.58	13.87	16.49	19.34	22.30	25.30	exact, dB
	-0.01	-0.00	+0.00	+0.00	+0.00	+0.01	+0.01	+0.03	engine - exact
	-1.49	-1.72	-1.80	-1.82	-1.86	-1.93	-2.00	-2.05	Maekawa - exact
90 m behind	8.12	9.61	11.46	13.70	16.30	19.13	22.08	25.08	exact, dB
	-0.00	-0.00	+0.00	+0.00	+0.00	+0.01	+0.01	+0.03	engine - exact
	-1.55	-1.79	-1.87	-1.88	-1.92	-1.99	-2.06	-2.11	Maekawa - exact

The engine agrees within 0.07 dB. Maekawa's chart is within 2.1 dB, giving less loss than the exact solution from 10 m behind the barrier.

The same barrier on a rigid ground, against the four paths over its top (the source and its image in the ground, to the listener and its image), each by the exact solution, and without the barrier the direct sound and its reflection, averaged over each band. In the bands up to 500 Hz the engine reads the early paths through band filters, whose skirts let in a little of the neighbouring bands where four paths interfere, so the table also compares it with the exact solution read through the same filters (*through the filters*; from 1 kHz up the engine averages each pair of paths' interference over the band exactly, and the row repeats the one above it). First with `simulation.early.diffraction_reflections: both_sides`, which takes all four paths:

listener	63	125	250	500	1k	2k	4k	8k	
2 m behind	14.20	11.15	23.58	10.23	19.56	22.62	24.84	28.16	exact, dB
	-0.57	+0.23	-2.09	+0.37	+0.03	+0.01	+0.05	+0.14	engine - exact
	-0.02	-0.06	-0.05	-0.03	+0.03	+0.01	+0.05	+0.14	through the filters
	-0.00	+0.00	+0.00	+0.00	-0.00	-0.00	-0.00	-0.01	without the barrier
10 m behind	6.30	13.89	25.20	14.79	9.06	22.30	22.66	25.94	exact, dB
	+0.05	-0.04	-2.24	-0.06	+0.01	+0.00	+0.04	+0.07	engine - exact
	-0.04	-0.07	-0.04	-0.04	+0.01	+0.00	+0.04	+0.07	through the filters
	-0.00	+0.00	+0.00	+0.00	+0.00	-0.00	-0.00	-0.01	without the barrier
40 m behind	4.41	8.41	18.05	18.54	18.19	15.47	21.17	24.46	exact, dB
	+0.03	+0.08	-0.50	+0.19	-0.00	+0.00	+0.02	+0.06	engine - exact
	-0.04	-0.06	-0.06	-0.06	-0.00	+0.00	+0.02	+0.06	through the filters
	-0.00	+0.00	+0.00	+0.00	-0.00	-0.00	-0.00	-0.00	without the barrier
90 m behind	4.16	7.93	16.65	12.67	20.65	24.89	16.26	23.52	exact, dB
	+0.03	+0.08	-0.60	+0.25	+0.02	-0.01	+0.02	+0.07	engine - exact
	-0.03	-0.06	-0.06	-0.04	+0.02	-0.01	+0.02	+0.07	through the filters
	-0.00	+0.00	+0.00	+0.00	-0.00	-0.00	-0.00	-0.00	without the barrier

Read alike, the engine is within 0.15 dB of the exact solution at every listener and band (0.07 dB up to 500 Hz, 0.14 dB at 8 kHz, 0.05 dB elsewhere), and without the barrier (the direct sound and its reflection, added in pressure) within 0.01 dB. The 2 dB at 250 Hz near the barrier is what the band filters let in from the bands beside a deep interference dip, as a measured response read through the same filters would show, not a gap in the model.

Then with the default `one_side`, where a path reflects before or after the edge but not both, so the path reflected both before and after the barrier, a quarter of the sound over it, is left out:

listener	63	125	250	500	1k	2k	4k	8k	
2 m behind	-2.31	+1.46	-3.85	+0.88	+0.36	+0.30	+0.52	+0.53	engine - exact
	-1.76	+1.17	-1.81	+0.48	+0.36	+0.30	+0.52	+0.53	through the filters
10 m behind	+0.93	-1.65	-5.05	+0.87	+0.53	+0.43	+0.65	+0.54	engine - exact
	+0.83	-1.68	-2.85	+0.89	+0.53	+0.43	+0.65	+0.54	through the filters
40 m behind	+1.71	+0.72	-2.68	+1.78	+0.40	+0.26	+0.72	+0.62	engine - exact
	+1.64	+0.57	-2.25	+1.54	+0.40	+0.26	+0.72	+0.62	through the filters
90 m behind	+1.85	+1.06	-2.41	+1.74	-0.50	-0.75	+0.76	+0.68	engine - exact
	+1.79	+0.91	-1.88	+1.46	-0.50	-0.75	+0.76	+0.68	through the filters

Without that path the insertion loss comes out 0.3 to 0.8 dB too high from 1 kHz up, but for 1 and 2 kHz 90 m behind, where it is 0.5 and 0.8 dB too low; in the bands below, where the four paths interfere, it is up to 3 dB off either way read through the filters (5 dB against the exact band averages). Over grass and other soft grounds, whose reflections near grazing are weaker, the path left out should carry less, but no check here runs `one_side` over such a ground. The sound that goes over the top, down to the ground and over the top again is left out of both.

A source 1 m over grass and nothing else, the listener 1.5 m up: the level relative to free field against the exact solution for a locally reacting ground (Ground reflection), averaged over each band:

distance	63	125	250	500	1k	2k	4k	8k	
10 m	5.27	3.74	-1.23	-1.55	1.43	0.27	1.10	0.85	exact, dB
	-0.04	-0.11	-0.06	+0.44	-0.01	-0.00	-0.01	-0.01	engine - exact
50 m	5.52	3.58	-3.02	-11.16	-2.05	3.84	2.74	2.43	exact, dB
	-0.07	-0.15	-0.03	+0.85	+0.00	-0.00	-0.00	-0.00	engine - exact
200 m	5.31	1.10	-13.10	-22.37	-12.89	-5.56	0.52	4.91	exact, dB
	-0.15	-0.10	+0.87	+0.72	+0.00	+0.00	-0.00	-0.00	engine - exact

From 1 kHz up the engine averages each pair of paths' interference over the band exactly and agrees within 0.01 dB; in the bands up to 500 Hz it reads the early paths through band filters, whose skirts fill the ground dip by up to 0.87 dB (the ground's own check above compares the band averages, without the filters). Both results are checked against theory, not against measurements of a barrier or a ground.

Outdoor sound in wind against a fast field program

`validation/ffp.py` is a fast field program for sound over a flat ground through a layered atmosphere, the kind of model the benchmark cases for outdoor sound of Attenborough and others (J. Acoust. Soc. Am. 97, 1995, 173-191) were solved with, written apart from the engine to check its parabolic equation in wind and temperature. It writes the sound field as an integral over the horizontal wavenumber and, for each wavenumber, solves the vertical problem exactly through a stack of layers in each of which the speed of sound is constant: the atmosphere as a staircase of layers a quarter wavelength thick, 1 m at most, halving sixteen times towards the ground, the source, the listener and the top; the ground an impedance boundary; and above the top, 1000 m up, where the engine's atmosphere stops changing, homogeneous air that sends nothing back. The integral follows a contour dipped into the complex plane past the poles near the real axis, with the Bessel function evaluated exactly rather than by its far-field form, so the field is right at 100 m as at 10 km. Its Bessel functions are its own (a power series up to an argument of 17 and the Hankel expansion beyond, 2×10^{-10} apart where they meet and 2×10^{-8} off tabulated integrals), so the script needs nothing but numpy.

In still air it is checked against the exact solution for a source over a locally reacting ground (Ground reflection), the source 5 m and the listener 1 m up, the level over the free field's:

	100 m	300 m	1 km	3 km	10 km	
10 Hz, rigid	6.02	6.02	6.02	6.02	6.02	exact, dB
	0.000	0.000	0.000	0.000	0.000	ffp - exact
10 Hz, grass	6.04	6.12	6.26	6.43	6.51	exact, dB
	0.000	0.000	0.000	0.000	0.000	ffp - exact
100 Hz, rigid	5.98	6.02	6.02	6.02	6.02	exact, dB
	0.000	0.000	0.000	0.000	0.000	ffp - exact
100 Hz, grass	3.12	0.72	-7.95	-23.91	-35.29	exact, dB
	0.000	0.000	0.000	0.000	-0.001	ffp - exact
1000 Hz, rigid	1.72	5.61	5.98	6.02	6.02	exact, dB
	0.000	0.000	0.000	0.000	0.000	ffp - exact
1000 Hz, grass	1.30	-6.58	-16.61	-26.04	-36.46	exact, dB
	0.000	0.000	0.000	0.000	-0.001	ffp - exact

It agrees to 0.001 dB from 10 Hz to 1 kHz and from 100 m to 10 km, over a rigid ground and over grass. With the speed of sound rising 0.1 m/s per metre of height, halving its layers changes the level by 0.006 dB at most (100 Hz and 1 kHz, 1 and 10 km out), so the staircase is converged. In a deep shadow its layers scatter a little sound: the field has a floor near -60 dB re free field, lower with thinner layers, so nothing is compared below -60 dB, where nothing is heard anyway. The tables below still print the program's level there, down to -125 dB: those values are its floor, not the field.

`validation/benchmark_1995.py` runs the benchmark's cases 1 to 3 through the program and the engine: a source 5 m and a listener 1 m over a ground of flow resistivity 200 kPa s/m² (Delany and Bazley's impedance), 100 m to 10 km apart, in still air (case 1), with the speed of sound rising 0.1 m/s per metre of height (case 2, downward refraction: downwind or under an inversion) and falling as much (case 3, upward refraction: upwind or over sun-warmed ground). The benchmark's gradients go on without end; the engine's atmosphere stops changing 1000 m up, 100 m/s off the ground's speed, and the program follows it there. Such a gradient over a kilometre of height is beyond any weather, so two more cases stop it 100 m up, 10 m/s more or less than at the ground, as a strong wind blows: the weather the engine is meant for. Case 4, case 3 with turbulence, is not run: the benchmark's eddies have a Gaussian spectrum, the engine's a von Kármán one, checked in Turbulence.

First the engine's parabolic equations alone, at the benchmark's pure tones: the narrow-angle one (`acoustic_spaces.engine.parabolic`, the Green's function parabolic equation of Gilbert and Di, the default, which gives the engine's sound over the open ground in the bands up to 2 kHz) and the wide-angle one (`acoustic_spaces.engine.wide_angle`, see The wide-angle parabolic equation); the program's level re free field and each equation's difference from it (`-`: deeper than -60 dB):

	100 m	200 m	500 m	1 km	2 km	5 km	10 km	
case 1, still air, 10 Hz	6.00	6.02	6.04	6.06	6.08	6.12	6.12	ffp, dB
	+0.04	+0.01	-0.03	-0.03	-0.04	-0.04	-0.01	narrow - ffp
	-0.02	-0.00	+0.00	+0.00	+0.00	-0.00	+0.03	wide - ffp
case 1, still air, 100 Hz	4.26	3.72	2.27	-0.11	-4.88	-16.72	-24.77	ffp, dB
	-0.03	-0.02	-0.00	+0.02	+0.02	-0.05	-0.10	narrow - ffp
	+0.00	+0.00	+0.00	+0.00	+0.00	-0.00	-0.04	wide - ffp
case 1, still air, 1000 Hz	0.72	-3.88	-11.14	-16.96	-22.88	-30.78	-36.78	ffp, dB
	-0.03	-0.04	-0.05	-0.05	-0.05	-0.05	-0.05	narrow - ffp
	-0.00	-0.00	-0.00	+0.00	-0.00	+0.00	+0.01	wide - ffp
case 2, downward refraction 0.1/s, 10 Hz	6.23	6.67	8.62	13.02	13.84	20.29	24.28	ffp, dB
	-0.20	-0.27	-0.38	-0.45	+0.14	-6.47	-2.10	narrow - ffp
	-0.02	-0.00	+0.00	+0.00	-0.01	-0.01	+0.03	wide - ffp
case 2, downward refraction 0.1/s, 100 Hz	4.60	4.84	5.51	2.81	8.72	3.92	11.89	ffp, dB
	-0.04	-0.04	-0.01	-0.03	-0.02	+5.52	-19.77	narrow - ffp
	+0.00	+0.00	-0.00	-0.00	-0.01	-0.02	-0.01	wide - ffp
case 2, downward refraction 0.1/s, 1000 Hz	2.13	1.71	8.30	9.25	5.31	-2.51	-10.13	ffp, dB
	-0.04	-0.04	-0.04	+0.32	-10.92	+3.40	-0.58	narrow - ffp
	-0.00	-0.00	-0.00	-0.00	-0.01	-0.05	-0.11	wide - ffp
case 3, upward refraction 0.1/s, 10 Hz	5.77	5.35	3.40	-1.29	-13.32	-41.23	-77.99	ffp, dB
	+0.28	+0.21	+0.21	+0.47	-0.06	-9.08	-	narrow - ffp
	-0.02	-0.00	-0.00	-0.01	-0.01	+0.28	-	wide - ffp
case 3, upward refraction 0.1/s, 100 Hz	3.88	2.40	-3.54	-16.07	-44.04	-64.00	-98.16	ffp, dB
	-0.02	-0.01	+0.01	+0.04	-0.12	-	-	narrow - ffp
	-0.00	-0.00	-0.01	-0.01	-0.05	-	-	wide - ffp
case 3, upward refraction 0.1/s, 1000 Hz	-1.08	-12.11	-52.50	-92.86	-98.39	-94.54	-121.03	ffp, dB
	-0.05	-0.04	+7.79	-	-	-	-	narrow - ffp
	-0.00	-0.00	-0.02	-	-	-	-	wide - ffp
0.1/s up to 100 m, 10 Hz	6.23	6.68	8.55	13.37	15.64	18.42	18.27	ffp, dB

	100 m	200 m	500 m	1 km	2 km	5 km	10 km	
	-0.20	-0.24	-0.33	-0.49	-0.34	-0.38	-0.49	narrow - ffp
	-0.02	-0.00	+0.00	+0.00	-0.00	-0.00	-0.01	wide - ffp
0.1/s up to 100 m, 100 Hz	4.60	4.84	5.51	2.80	-13.83	-2.11	-10.57	ffp, dB
	-0.04	-0.04	-0.02	+0.04	+3.22	-0.19	-3.12	narrow - ffp
	+0.00	+0.00	-0.00	-0.00	+0.02	-0.00	+0.00	wide - ffp
0.1/s up to 100 m, 1000 Hz	2.13	1.71	8.30	9.25	0.48	-21.83	-47.12	ffp, dB
	-0.04	-0.04	-0.04	+0.33	+1.23	-1.93	+11.32	narrow - ffp
	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	+0.02	wide - ffp
-0.1/s up to 100 m, 10 Hz	5.78	5.35	3.28	-0.47	-9.92	-44.43	-69.88	ffp, dB
	+0.28	+0.24	+0.32	+0.28	+0.22	+1.42	-	narrow - ffp
	-0.02	-0.00	-0.00	-0.01	-0.01	-0.41	-	wide - ffp
-0.1/s up to 100 m, 100 Hz	3.88	2.40	-3.53	-16.14	-45.75	-102.87	-98.15	ffp, dB
	-0.02	-0.01	-0.02	+0.10	-5.79	-	-	narrow - ffp
	-0.00	-0.00	-0.01	-0.02	+0.07	-	-	wide - ffp
-0.1/s up to 100 m, 1000 Hz	-1.08	-12.11	-52.51	-64.37	-98.31	-125.03	-121.03	ffp, dB
	-0.04	-0.05	+0.33	-	-	-	-	narrow - ffp
	-0.00	-0.00	-0.01	-	-	-	-	wide - ffp

In still air the narrow-angle equation and the program agree within 0.05 dB to 5 km and 0.1 dB at 10 km. At 10 Hz, below the engine's lowest band, the narrow-angle equation is up to 0.5 dB off in the refracting cases out to 2 km, and further out 0.5 dB downwind and 1.4 dB upwind with the gradient stopping at 100 m, 2 to 9 dB with the benchmark's gradients. In wind the agreement holds as far as the sound reaches the listener at the flat angles a parabolic equation carries. With the gradient stopping 100 m up, which turns back only the sound within 14 degrees of the horizontal, the parabolic equation is within 0.05 dB at 100 Hz and 0.35 dB at 1 kHz up to 1 km downwind, 1 to 3 dB at 2 km, where the bent paths interfere and the two place a dip a little apart, within 2 dB at 5 km, and 3 dB at 10 km at 100 Hz (11 dB at 1 kHz, 47 dB down; the engine's 1 kHz band comes from this equation there too, not compared as a band beyond 2 km). Upwind, in the shadow, it is within 0.1 dB at 100 Hz and 1 kHz until the sound is 20 dB down (0.33 dB at 1 kHz 500 m, 52 dB down) and strays by up to 6 dB deeper in (100 Hz, 2 km, 46 dB down; 1.4 dB at 10 Hz, 5 km, 44 dB down), below where the eddies keep a shadow in practice (see Turbulence). With the benchmark's gradient going on to 1000 m, sound that has risen hundreds of metres comes back down at 17 degrees and more, and upwind the speed of sound a kilometre up is 100 m/s below the ground's, an index of refraction of 1.4; the parabolic equation, a narrow-angle one that marches the field out along the ground and takes the refraction as a phase shift at each step, carries neither: downwind the two part beyond 1 km at 1 kHz and 2 km at 100 Hz, by 3 to 20 dB at a pure tone where several bent paths interfere, and upwind it comes out 8 dB too loud at 1 kHz 500 m into the shadow (0.3 dB with the gradient stopping at 100 m). Such profiles are beyond any weather; the engine's atmosphere is meant for the gradients near the ground that make the shadows and the focusing in practice, and within a few kilometres of the source. The wide-angle equation carries both: within 0.05 dB of the program in every case, at every tone and to 10 km, wherever the sound is within 40 dB of free field, but for 0.11 dB at 1 kHz 10 km downwind; deeper in the shadow within 0.07 dB at 100 Hz and 1 kHz, and 0.3 to 0.4 dB at 10 Hz 5 km upwind, 41 to 44 dB down.

Then whole scenes, each case as an outdoor scene: the gradient as a measured atmosphere (`atmosphere.table`) with the wind along the line of the listeners, the air at 19.6 °C (343 m/s at the ground) and 50 % humidity, and the engine's defaults otherwise (the early paths as image sources up to one reflection, 50,000 late rays). The level relative to free field the engine reports per octave band from 63 Hz to 1 kHz (Outdoor scenes) against the program averaged over the band as the engine reads it (eight frequencies per band; 24 at 1 kHz, as the parabolic equation takes there, and in still air the rays average the paths' interference exactly), and in the bands up to 500 Hz also against the program read through the engine's band filters, as the barrier check above is (*through the filters*). The engine's rays carry the air's absorption along their own paths and the program runs without it, a difference of the path lengths only, under 0.01 dB here:

case 1, still air

distance	63	125	250	500	1k	
100 m	5.17	3.15	-2.31	-6.43	1.37	ffp, dB
	-0.05	-0.11	-0.03	+0.59	-0.00	engine - ffp
	+0.00	-0.01	-0.01	-0.00	-0.00	through the filters
200 m	5.01	2.07	-6.18	-11.08	-2.96	ffp, dB
	-0.08	-0.14	+0.17	+0.54	-0.00	engine - ffp
	-0.02	-0.01	+0.03	-0.03	-0.00	through the filters
500 m	4.58	-0.53	-14.60	-18.32	-10.16	ffp, dB
	-0.14	-0.04	+0.93	+0.44	-0.00	engine - ffp
	-0.04	+0.05	+0.16	-0.11	-0.00	through the filters
1 km	3.85	-3.98	-22.32	-24.12	-15.97	ffp, dB
	-0.15	+0.20	+1.47	+0.31	-0.00	engine - ffp
	-0.02	+0.12	+0.32	-0.23	-0.00	through the filters
2 km	2.41	-9.47	-28.84	-30.03	-21.90	ffp, dB
	-0.10	+0.81	+1.30	+0.11	-0.00	engine - ffp
	+0.04	+0.27	+0.53	-0.42	-0.00	through the filters

case 2, downward refraction 0.1/s

distance	63	125	250	500	1k	
100 m	5.55	3.42	-2.27	-5.22	2.55	ffp, dB
	-0.05	-0.05	-0.04	-0.07	-0.04	engine - ffp
	+0.00	+0.06	-0.04	-0.68	-0.04	through the filters
200 m	6.21	2.97	-5.67	-6.00	2.28	ffp, dB
	-0.04	-0.04	-0.05	-0.07	-0.05	engine - ffp
	+0.03	+0.09	-0.27	-0.69	-0.05	through the filters
500 m	9.03	1.29	-2.98	-2.74	5.07	ffp, dB
	-0.03	-0.04	-0.06	-0.05	-0.06	engine - ffp
	+0.12	-0.19	-0.12	-0.82	-0.06	through the filters
1 km	6.79	1.37	1.43	1.55	6.63	ffp, dB
	+0.03	-0.05	-0.04	-0.11	-0.05	engine - ffp
	-0.22	-0.15	+0.12	-0.19	-0.05	through the filters
2 km	5.93	5.97	-0.51	4.14	4.18	ffp, dB
	-0.02	+0.07	-0.53	-0.75	-0.90	engine - ffp
	-0.33	+0.34	-1.32	-0.45	-0.90	through the filters

case 3, upward refraction 0.1/s

distance	63	125	250	500	1k	
100 m	4.78	2.82	-2.44	-7.68	-0.30	ffp, dB
	-0.02	-0.01	-0.03	-0.07	-0.04	engine - ffp
	+0.03	+0.10	+0.01	-0.60	-0.04	through the filters
200 m	3.72	0.85	-7.14	-16.62	-11.46	ffp, dB
	-0.02	-0.02	-0.03	-0.08	-0.05	engine - ffp
	+0.05	+0.11	-0.12	-0.53	-0.05	through the filters
500 m	-0.84	-6.20	-20.67	-44.90	-52.70	ffp, dB
	-0.02	-0.00	-0.02	-0.10	-0.07	engine - ffp
	+0.07	+0.08	-0.88	-2.33	-0.07	through the filters
1 km	-10.83	-19.87	-43.44	-90.90	-100.23	ffp, dB
	-0.01	+0.01	+0.02	-	-	engine - ffp
	+0.06	-0.13	-3.40	-	-	through the filters
2 km	-33.35	-48.82	-78.33	-88.90	-94.67	ffp, dB
	-0.01	+0.01	-	-	-	engine - ffp
	-0.13	-0.93	-	-	-	through the filters

0.1/s up to 100 m

distance	63	125	250	500	1k	
100 m	5.55	3.42	-2.27	-5.22	2.55	ffp, dB
	-0.05	-0.05	-0.04	-0.07	-0.04	engine - ffp
	+0.00	+0.06	-0.04	-0.68	-0.04	through the filters
200 m	6.21	2.97	-5.67	-6.00	2.28	ffp, dB
	-0.04	-0.04	-0.05	-0.07	-0.05	engine - ffp
	+0.03	+0.09	-0.27	-0.69	-0.05	through the filters
500 m	9.03	1.29	-2.98	-2.74	5.07	ffp, dB
	-0.03	-0.03	-0.06	-0.05	-0.06	engine - ffp
	+0.12	-0.18	-0.12	-0.81	-0.06	through the filters
1 km	6.74	1.38	1.43	1.55	6.63	ffp, dB
	+0.04	-0.04	-0.04	-0.11	-0.05	engine - ffp
	-0.21	-0.14	+0.13	-0.19	-0.05	through the filters
2 km	3.80	-1.16	-2.85	-5.93	1.15	ffp, dB
	-0.08	+0.01	-0.09	+0.44	+0.38	engine - ffp
	-0.43	-0.16	-0.01	-0.46	+0.38	through the filters

-0.1/s up to 100 m

distance	63	125	250	500	1k	
100 m	4.78	2.82	-2.44	-7.68	-0.30	ffp, dB
	-0.01	-0.02	-0.03	-0.07	-0.04	engine - ffp
	+0.03	+0.08	+0.01	-0.60	-0.04	through the filters
200 m	3.72	0.85	-7.14	-16.62	-11.46	ffp, dB
	-0.02	-0.02	-0.03	-0.08	-0.05	engine - ffp
	+0.04	+0.11	-0.12	-0.53	-0.05	through the filters
500 m	-0.84	-6.20	-20.66	-44.81	-52.68	ffp, dB
	-0.01	-0.00	-0.04	-0.20	-0.08	engine - ffp
	+0.08	+0.08	-0.89	-2.39	-0.08	through the filters
1 km	-10.87	-19.88	-43.62	-66.43	-66.91	ffp, dB
	-0.03	-0.09	+1.48	-	-	engine - ffp
	+0.01	-0.23	-1.95	-	-	through the filters
2 km	-31.61	-43.55	-69.69	-91.23	-98.25	ffp, dB
	+0.06	+0.06	-	-	-	engine - ffp
	+0.19	+0.18	-	-	-	through the filters

In still air the early paths are image sources read through the band filters, and the row *through the filters* is the one to compare: the engine is within 0.04 dB of the program up to 200 m and within 0.3 dB at 1 km, 0.5 dB at 2 km, where the ground dip at 250 Hz is 29 dB deep and the engine's reflection factor is an approximation (see Ground reflection); from 1 kHz up, where the rays average the paths' interference over the band exactly, within 0.005 dB. In wind the bands up to 2 kHz come from the parabolic equation, whose band levels are the averages over its frequencies (eight an octave, 24 from 1 kHz up), and the engine reports them as computed, so there the plain average is the row to read: from 63 to 500 Hz within 0.11 dB to 1 km in every case downwind, and upwind with the benchmark's gradient (upwind with the gradient stopping at 100 m, see below); at 2 km downwind 0.5 to 0.8 dB with the gradient going on to 1000 m, where the pure tones above show the parabolic equation itself parting from the program, and 0.1 (0.4 at 500 Hz) with it stopping at 100 m. Upwind the engine follows the program into the shadow within 0.1 dB to 500 m and 45 dB below free field (0.20 dB with the gradient stopping at 100 m, at 500 Hz), and within 1.5 dB at 1 km with the gradient stopping at 100 m (250 Hz, 44 dB down, where the pure tones above show the parabolic equation straying deep in the shadow); the comparison stops 60 dB down. The 1 kHz band comes from the parabolic equation too: within 0.09 dB to 1 km, downwind and upwind, and at 2 km downwind 0.9 dB low with the gradient going on to 1000 m, as the bands below it there, and 0.38 dB high with it stopping at 100 m.

At the join between the parabolic equation's bands and the rays' each ray band keeps its own spectrum, since a blend of spectra that stand for different sound (the whole field over the ground on one path, each ray's own on the others) would put a notch in it, and a path that carries nothing in a band is left out of that band's reading. The equation's bands carry its field at every frequency its runs took, and out of the shadow the top one on through the crossover above it (`engine/fine.py`), so both bands of a crossover give the same field there: in case 2 the dip in the crossover above the 2 kHz band, 12.8 dB deep at 2.85 kHz 100 m away, comes out within 0.19 dB of the program from 2 to 3.45 kHz at 100 m, 0.05 and 0.28 dB at 150 and 200 m and 0.45 dB at 300 m (on the rays' side, 3.1 kHz), alike with 8 frequencies an octave and with the wide-angle equation, the join of the equation's 500 Hz and 1 kHz bands within 0.07 dB, and third-octave runs within 0.12 and 0.58 dB at 100 and 300 m (`tests/test_ffp.py`). In the shadow the bands keep their own samples (see Known issues).

Up to 500 Hz the echogram reads the bands through band filters from that field, the 63 Hz band's run going on down to an octave below it, where its filter still reads it. 100 m downwind in case 2 and 200 m downwind in `windy`, C50 and C80 at 63 Hz are within 0.01 and 0.09 dB and within 0.06 and 0.07 dB of a run of the equation 27 times as dense going on down to 11 Hz, read through the same filters (27 and 81 times, to 11 or 5.5 Hz, agree within 0.06 dB), where the band's own samples were 0.30 and 0.29 dB and 0.91 and 0.19 dB off, and EDT within a step of the echogram (1 ms), where `windy`'s was 1.8 ms off (`tests/test_ffp.py`); a coverage map's point keeps the same samples in these bands as a listener there (`tests/test_refraction.py`). That run's lowest frequency, 25 Hz, thickens its absorbing layer, which moves the 63 Hz band's level by 0.002 and 0.003 dB, to within 0.0002 dB of a thicker one still. Above 63 Hz, where C50 and C80 are 50 to 88 dB, see Known issues.

The wide-angle parabolic equation

With `simulation.early.parabolic_equation: wide_angle` (`--parabolic-equation wide_angle`, **Parabolic equation** in the editor) the bands the parabolic equation gives outdoors in wind or temperature gradients (see Wind and temperature) come from `acoustic_spaces.engine.wide_angle` instead of the narrow-angle default: the split-step Padé parabolic equation of Collins (J. Acoust. Soc. Am. 93, 1993, 1736, as in his program RAM). It marches the field with the one-way operator of the Helmholtz equation, exact at every angle in air that does not change along the way, written as a rational function of ten terms that never amplifies the waves that do not propagate (RAM's stability constraint), in steps of three wavelengths; the heights are finite differences of fourth order, twenty points per wavelength, with the ground's impedance as the lower boundary, also to fourth order, and the field starts as the operator has it (Collins' self-starter, J. Acoust. Soc. Am. 92, 1992, 2069). The ten tridiagonal solves of each step run in native code, the frequencies of a band on the run's threads.

`tests/test_wide_angle.py` checks each part on every change: one step against the exact exponential of the operator, within 10^{-7} for sound up to 45 degrees from the horizontal and 10^{-13} within 18 degrees of it; that it damps the waves that do not propagate; the starting field's rational function within 10^{-8} of its own; the native solves against dense ones, and on any number of threads bit for bit; in still air the complex pressure over grass within 0.5 % (0.04 dB, 0.3 degrees) of the exact solution at 125 and 500 Hz; the field 45 degrees up within 0.15 dB of the exact one, where the narrow-angle equation's starting field falls 1 dB short; and the benchmark's case 2 at 100 Hz within 0.05 dB of the fast field program to 5 km, where the narrow-angle equation is 6 dB off.

The benchmark. At the 1995 benchmark's pure tones (the table in Outdoor sound in wind) the wide-angle equation is within 0.05 dB of the fast field program in every case, from 10 Hz to 1 kHz and from 100 m to 10 km, wherever the sound is within 40 dB of free field, but for 0.11 dB at 1 kHz 10 km downwind; deeper in the shadow it is within 0.07 dB at 100 Hz and 1 kHz, and 0.3 to 0.4 dB at 10 Hz 5 km upwind, 41 to 44 dB down. With the benchmark's gradients going on to a kilometre up, where the narrow-angle equation strays by 3 to 20 dB beyond 1 to 2 km downwind and by 8 dB 500 m into the shadow upwind at 1 kHz, the wide-angle one stays within 0.11 dB wherever the sound is within 40 dB of free field (0.28 dB at 10 Hz 5 km upwind, 41 dB down).

The weather presets. The gradients the engine is meant for change fastest near the ground. Both equations against the fast field program in the presets: a source 2 m and a listener 1.5 m up over the benchmark's ground, at 20 °C, 50 to 300 m apart, the level re free field (dB) and each equation's difference from it at the octave bands' centre tones:

	50 m	100 m	200 m	300 m	
`windy`, downwind, 63 Hz	6.14	6.91	8.24	9.32	ffp, dB
	-0.19	-0.19	-0.16	-0.18	narrow - ffp
	+0.02	+0.04	+0.05	+0.06	wide - ffp
`windy`, downwind, 125 Hz	4.76	5.33	5.66	4.67	ffp, dB
	+0.10	+0.10	-0.02	-0.11	narrow - ffp
	+0.00	+0.00	-0.02	-0.07	wide - ffp
`windy`, downwind, 250 Hz	-2.27	-6.77	-26.18	-22.74	ffp, dB
	-0.16	-0.27	-0.04	-0.42	narrow - ffp
	-0.07	-0.15	-0.05	-0.18	wide - ffp
`windy`, downwind, 500 Hz	-6.43	-6.84	-3.53	0.20	ffp, dB
	-0.07	-0.05	+0.06	+0.06	narrow - ffp
	+0.03	+0.03	+0.04	+0.04	wide - ffp
`windy`, downwind, 1 kHz	3.78	5.02	3.98	3.25	ffp, dB
	+0.00	-0.02	+0.01	+0.01	narrow - ffp
	+0.01	+0.01	+0.03	+0.03	wide - ffp
`windy`, downwind, 2 kHz	-0.02	4.78	-2.11	3.92	ffp, dB
	+0.13	-0.00	+0.17	-0.06	narrow - ffp
	-0.00	+0.00	+0.02	+0.01	wide - ffp
`windy`, upwind, 63 Hz	4.86	3.93	1.91	-0.24	ffp, dB
	+0.19	+0.10	-0.07	-0.17	narrow - ffp
	-0.03	-0.05	-0.09	-0.12	wide - ffp
`windy`, upwind, 125 Hz	3.41	1.58	-2.42	-6.73	ffp, dB
	-0.23	-0.31	-0.37	-0.48	narrow - ffp
	-0.01	-0.03	-0.07	-0.12	wide - ffp
`windy`, upwind, 250 Hz	-0.29	-4.13	-12.53	-21.73	ffp, dB
	-0.05	-0.10	-0.20	-0.30	narrow - ffp
	+0.02	+0.01	-0.01	-0.05	wide - ffp
`windy`, upwind, 500 Hz	-6.16	-14.45	-33.95	-56.97	ffp, dB
	-0.16	-0.16	-0.14	-0.12	narrow - ffp
	+0.05	+0.06	+0.05	+0.02	wide - ffp
`windy`, upwind, 1 kHz	-2.40	-21.33	-75.77	-89.89	ffp, dB
	-0.15	-0.21	-	-	narrow - ffp
	+0.01	+0.02	-	-	wide - ffp
`windy`, upwind, 2 kHz	1.76	-39.93	-105.59	-100.79	ffp, dB
	-0.15	-0.36	-	-	narrow - ffp
	+0.00	+0.04	-	-	wide - ffp

	50 m	100 m	200 m	300 m	
`sunny_day`, 63 Hz	5.32	5.02	4.36	3.66	ffp, dB
	+0.07	+0.05	+0.02	+0.00	narrow - ffp
	-0.01	-0.01	-0.02	-0.03	wide - ffp
`sunny_day`, 125 Hz	3.96	3.18	1.57	-0.03	ffp, dB
	-0.08	-0.09	-0.07	-0.08	narrow - ffp
	-0.00	-0.01	-0.01	-0.02	wide - ffp
`sunny_day`, 250 Hz	-0.47	-3.15	-8.20	-12.70	ffp, dB
	-0.02	-0.02	+0.01	+0.02	narrow - ffp
	+0.01	+0.02	+0.02	+0.03	wide - ffp
`sunny_day`, 500 Hz	-7.51	-13.17	-22.26	-30.27	ffp, dB
	-0.09	-0.10	-0.10	-0.09	narrow - ffp
	+0.01	+0.02	+0.02	+0.03	wide - ffp
`sunny_day`, 1 kHz	0.67	-7.25	-22.60	-38.38	ffp, dB
	-0.07	-0.08	-0.10	-0.11	narrow - ffp
	-0.00	+0.00	+0.00	-0.00	wide - ffp
`sunny_day`, 2 kHz	5.04	-3.35	-31.56	-68.48	ffp, dB
	-0.03	-0.08	-0.11	-	narrow - ffp
	-0.00	+0.00	+0.00	-	wide - ffp
`clear_night`, 63 Hz	5.73	5.97	6.41	6.78	ffp, dB
	-0.05	-0.04	-0.01	-0.00	narrow - ffp
	+0.01	+0.01	+0.02	+0.02	wide - ffp
`clear_night`, 125 Hz	4.39	4.37	4.18	3.78	ffp, dB
	+0.02	+0.03	+0.03	+0.03	narrow - ffp
	+0.00	+0.00	+0.01	+0.00	wide - ffp
`clear_night`, 250 Hz	-1.06	-3.81	-10.28	-17.94	ffp, dB
	-0.07	-0.07	-0.13	-0.28	narrow - ffp
	-0.02	-0.03	-0.07	-0.13	wide - ffp
`clear_night`, 500 Hz	-8.15	-11.48	-13.57	-13.55	ffp, dB
	-0.08	-0.07	-0.04	-0.04	narrow - ffp
	-0.00	-0.00	-0.00	+0.00	wide - ffp
`clear_night`, 1 kHz	2.67	1.04	1.68	3.96	ffp, dB
	-0.01	-0.01	-0.02	-0.02	narrow - ffp
	+0.00	+0.00	+0.00	+0.00	wide - ffp
`clear_night`, 2 kHz	-0.83	4.32	5.06	3.97	ffp, dB
	-0.11	-0.05	-0.04	-0.04	narrow - ffp
	+0.00	+0.00	+0.00	+0.00	wide - ffp

The narrow-angle equation is within 0.10 dB of the program on average and 0.48 dB at most (125 Hz, 300 m upwind in `windy`), the wide-angle one within 0.03 dB on average and 0.18 dB at most, where the program is no deeper than 60 dB below free field (-). Over each band as the engine reads it, the mean energy over its frequencies (eight an octave, 24 at 1 and 2 kHz), the narrow-angle equation is within 0.07 dB on average and 0.38 dB at most, the wide-angle one within 0.02 dB and 0.12 dB. In the weather near the ground both hold, so the narrow-angle equation stays the default, being the faster of the two.

Whole scenes. The benchmark's cases 2 and 3 as whole scenes again (see Outdoor sound in wind), with `parabolic_equation: wide_angle`:

case 2, downward refraction 0.1/s

distance	63	125	250	500	1k	
100 m	5.55	3.42	-2.27	-5.22	2.55	ffp, dB
	+0.00	+0.00	-0.00	-0.00	-0.00	engine - ffp
	+0.05	+0.11	+0.00	-0.61	-0.00	through the filters
200 m	6.21	2.97	-5.67	-6.00	2.28	ffp, dB
	+0.00	+0.00	-0.00	+0.00	-0.00	engine - ffp
	+0.07	+0.13	-0.22	-0.62	-0.00	through the filters
500 m	9.03	1.29	-2.98	-2.74	5.07	ffp, dB
	+0.00	-0.00	-0.00	+0.00	-0.00	engine - ffp
	+0.15	-0.15	-0.06	-0.76	-0.00	through the filters
1 km	6.79	1.37	1.43	1.55	6.63	ffp, dB
	-0.00	+0.00	-0.00	-0.00	-0.00	engine - ffp
	-0.25	-0.10	+0.16	-0.08	-0.00	through the filters
2 km	5.93	5.97	-0.51	4.14	4.18	ffp, dB
	-0.00	-0.00	-0.00	-0.00	-0.00	engine - ffp
	-0.30	+0.27	-0.79	+0.29	-0.00	through the filters

case 3, upward refraction 0.1/s

distance	63	125	250	500	1k	
100 m	4.78	2.82	-2.44	-7.68	-0.30	ffp, dB
	-0.00	-0.00	+0.00	+0.00	-0.00	engine - ffp
	+0.04	+0.10	+0.04	-0.54	-0.00	through the filters
200 m	3.72	0.85	-7.14	-16.62	-11.46	ffp, dB
	-0.00	-0.00	+0.00	+0.00	-0.00	engine - ffp
	+0.06	+0.12	-0.08	-0.45	-0.00	through the filters
500 m	-0.84	-6.20	-20.67	-44.90	-52.70	ffp, dB
	-0.01	-0.00	+0.00	+0.00	+0.01	engine - ffp
	+0.08	+0.08	-0.86	-2.23	+0.01	through the filters
1 km	-10.83	-19.87	-43.44	-90.90	-100.23	ffp, dB
	-0.01	-0.01	+0.01	-	-	engine - ffp
	+0.06	-0.14	-3.41	-	-	through the filters
2 km	-33.35	-48.82	-78.33	-88.90	-94.67	ffp, dB
	-0.04	-0.03	-	-	-	engine - ffp
	-0.15	-0.96	-	-	-	through the filters

From 63 Hz to 1 kHz the engine's levels are within 0.04 dB of the program's band averages to 2 km, downwind and upwind (at 1 kHz upwind to 500 m, 53 dB down), where with the narrow-angle equation they are 0.5 to 0.9 dB off at 2 km downwind.

Each band of the wide-angle equation marches a grid of twenty points per wavelength of its highest frequency in steps of three wavelengths, ten tridiagonal solves a step, where the narrow-angle one takes a pair of Fourier transforms a step on a grid half as fine, in steps of five wavelengths: five to eight times the time (5.4 to 5.7 on one core for a listener 100 m out, 7.6 at 500 m and 6.3 to 6.5 at 2 km, over grass in the `windy` preset and over asphalt in `clear_night`), all of it in the bands the parabolic equation gives. Neither equation is checked against measurements in strong refraction, and neither is combined with a terrain yet (see Outdoor scenes).

Steep sound. The narrow-angle equation's phase shift at each step strays for sound that comes in steeply, and its step is not converged: each step also adds an error at the ground that shrinks with the height step, not the range step, so on its grid of heights shorter steps add up more of it. From 1 m up to 1.5 m up, 200 m downwind in the `windy` preset over grass, the 63 Hz band is 0.26 dB under the fast field program in steps of five wavelengths, 0.14 dB over it in steps of one and 0.59 dB over it in steps a 32nd as long. So the listeners that the reflection off the ground reaches more than 15 degrees up get the wide-angle equation's level, with the narrow-angle equation too. They get it in whatever air the parabolic equation runs, in the still air of a weather zone as well, where the narrow-angle one is exact but at the ground: which equation a listener takes then goes neither by its band nor by the run's other listeners, as it does when it is found from the air on the run's grid (in a one-off check at 84 steep points in still air, from sources 1.5 to 30 m up to listeners 1.5 to 30 m up, 8 to 100 m away, over grass and over a ground of 20 000 kPa s/m², in the octave bands from 63 Hz to 2 kHz, it was 0.03 dB from the program on average and 0.16 dB at most, and moved by up to 0.03 dB with its absorbing layer four times as high; the wide-angle one 0.01 and 0.15 dB, the same to 0.0001 dB under the higher layer; per third octave from 50 Hz to 2.5 kHz, at three frequencies each, the narrow-angle one was 0.04 and 0.30 dB from the program and the wide-angle one 0.02 and 0.34 dB, the largest of each at 63 Hz 8 m from the source 2 m up, a wavelength and a half away). Against the program at the frequencies the engine takes in each band, eight an octave band and three a third octave (`validation/steep_sound.py`), in the `windy` preset downwind and upwind over grass, from sources 5, 10, 30 and 120 m up to a listener 1.5 m up and from 1.5 m up to 30 m up, at 16 ranges from 1.5 times their heights added (10 m at least) to 1 km and at two more a hundredth of the range either side of 15 degrees (each case's listeners in a line in one scene), where the level is above -15 dB re free field, the mean (largest) difference in dB:

bands	15° or flatter, narrow-angle	steeper, narrow-angle	steeper, narrow-angle, half its step	steeper, wide-angle
octaves, 63 to 500 Hz	0.10 (0.48)	0.10 (0.41)	0.09 (0.37)	0.01 (0.05)
third octaves, 50 to 250 Hz, sources 10 to 120 m up	0.17 (0.87)	0.24 (1.61)	0.17 (0.73)	0.03 (0.09)

A quarter of the narrow-angle step leaves the steep points 0.10 (0.42) and 0.20 (0.69) dB off. In 18 further cases (the `windy` preset and 10 m/s of wind down- and upwind, `clear_night`, and `sunny_day` with 3 m/s of wind downwind; from 20 m up to 1.5 m up and from 40 m up to 4 m up over grass, and from 60 m up to 4 m up over a ground of 2000 kPa s/m²; the 63 and 125 Hz octaves and the third octaves from 80 to 160 Hz) the steep points' octave bands are 0.16 (0.96) dB off with the narrow-angle equation, 0.14 (0.93) dB at half its step and 0.02 (0.09) dB with the wide-angle one, their third octaves 0.35 (2.20), 0.31 (1.96) and 0.04 (0.29) dB. In those cases the narrow-angle equation is about as far off just under 15 degrees: from 10 to 15 degrees up it is 0.14 (1.05) dB off per octave band and 0.38 (5.74) dB per third octave, the largest where the level is 14 dB below free field (80 Hz, 287 m from the source 60 m up, downwind in 10 m/s of wind over the hard ground; the wide-angle equation is 0.28 dB off there in a one-off check). So the level jumps at 15 degrees by the narrow-angle equation's error there: in the 28 cases of both sets, in the bands compared in each, per octave band by 0.12 dB on average and up to 0.72 dB (125 Hz, from 60 m up to 4 m up, upwind in 10 m/s of wind over the hard ground), per third octave by 0.27 dB and up to 2.3 dB (125 Hz in the same case), where the level is above -15 dB re free field; at the five largest jumps in each kind of band the wide-angle side is within 0.29 dB of the program and the narrow-angle side up to 2.4 dB off.

By the angle at which the reflection off the ground reaches the listener, with each equation run at every point of each case's line and at up to nine more between its ends, where the reflection comes in at 5.5 to 14 degrees, the mean (largest) difference in dB of the narrow-angle equation and, after it, of the wide-angle one:

reflection's angle	first set, octaves	first set, third octaves	further set, octaves	further set, third octaves
0 to 5°	0.13 (0.48), 0.03 (0.14)	0.22 (0.87), 0.07 (0.32)	0.20 (1.10), 0.05 (0.20)	0.24 (1.68), 0.07 (0.32)
5 to 7°	0.08 (0.42), 0.02 (0.06)	0.13 (0.48), 0.03 (0.09)	0.18 (1.21), 0.04 (0.14)	0.32 (4.55), 0.07 (0.41)
7 to 10°	0.06 (0.25), 0.02 (0.05)	0.12 (0.44), 0.03 (0.08)	0.11 (0.63), 0.03 (0.13)	0.25 (3.03), 0.07 (0.43)
10 to 15°	0.07 (0.23), 0.02 (0.04)	0.15 (0.64), 0.03 (0.09)	0.14 (1.21), 0.03 (0.14)	0.35 (5.74), 0.07 (0.36)
over 15°	0.10 (0.41), 0.01 (0.05)	0.24 (1.61), 0.03 (0.09)	0.16 (0.96), 0.02 (0.09)	0.35 (2.20), 0.04 (0.29)

At every angle the narrow-angle equation is 0.06 to 0.35 dB off on average and the wide-angle one 0.01 to 0.07 dB: its error does not begin at 15 degrees. A scene that needs the wide-angle equation's level at the flatter listeners too sets `parabolic_equation: wide_angle`, at the run time given above.

Over the open ground other listeners in the same direction do not change a level. Each listener is reached by a last step of its own off the march, and a run's absorbing layer starts where the layer of a run for each of its listeners alone would. A listener's own layer starts 1.5 times as high as the highest of the source, the listener and the bent paths between them, at least 20 m above the two and 0.15 times the range up; listeners share a run where those starts need the same grid of heights, a power of two of them, and the layer starts as high as that grid holds it, for one listener as for all. The listeners that take the wide-angle equation's level, whose grid is as high as its layer needs, share a run where their starts round up to the same of the heights 2^(k/2) m, where the layer then starts. In the lowest band a listener out of the shadow, whose run goes on down to an octave below the band and has a thicker layer for it, runs apart from those in the shadow. So a listener alone and the same one among others have the same early energy in every band, within 10^{-12} of it (`tests/test_refraction.py`: with others farther away and higher up, downwind and upwind, on two threads, with the wide-angle equation for all, under others in a weather zone, and from two sources whose runs are shared). The level does go by where the listener's own layer starts, more where the narrow-angle equation is near its limit or the level is low. For a listener 45 m from a source 10 m up over grass, which the reflection reaches 14 degrees up, 8.8 dB below free field in its dip over the ground in the 200 Hz third octave, in still air with a weather zone of the `windy` preset beginning 100 m away, the narrow-angle equation is 0.05 to 0.14 dB under the program under seven layers that begin 30 to 450 m up, and the wide-angle one 0.001 dB over it under each (a one-off check). So the level steps where a listener's own start needs the next grid of heights: in a one-off check of 37 such places 150 to 900 m from a source 2 m up, with listeners 1.5 m up half a percent of the range either side, over grass in the `windy` preset, by 0.001 dB or less per octave band and 0.011 dB or less per third octave where the level was less than 35 dB below free field (23 places, down- and upwind), and by up to 0.57 dB per octave band and 3.2 dB per third octave upwind where it was more than 60 dB below it (14 places). Behind barriers and buildings the others still change a level (see Known issues). Where the wide-angle equation runs behind a barrier its height step is shared too: it fits the roof of a building further along the way once one of its points lies beyond the building's front (in a one-off check of the march, from a source 2 m up over a ground of 200 kPa s/m², a barrier 3 m high 15 m away and a building 60 to 75 m away with its roof 4 m up, down- and upwind in 5 m/s of wind, a point 90 m away moved the 63 Hz band 20 m away by 0.018 dB, and the bands to 500 Hz 20 to 59 m away by 0.002 dB or less otherwise). At 1 and 2 kHz, in a one-off check at the steep points of the first set's cases from sources 5, 10 and 30 m up to a listener 1.5 m up and from 1.5 m up to 30 m up (42 points), the level is within 0.07 dB of the program at 1 kHz; at 2 kHz it is 0.11 to 0.49 dB off at each case's nearest point, where the sound comes in at 34 degrees, and within 0.27 dB at the others. The steep listeners' bands take the wide-angle equation's time: about 6 times the narrow-angle one's on one core for three listeners 10 to 20 m up, 30 to 60 m from a source 1 m up over a lawn in the `windy` preset.

Terrain against closed forms and the exact wedge

`validation/terrain.py` checks sound over a terrain (`geometry.ground.terrain`, see Outdoor scenes) against closed forms and the exact solution for a wedge, and times it on a large one. The early sound reflects off the mean plane of the ground and bends over its crests as the uniform theory of diffraction has it, wedge after wedge, with the ground's reflections off the profile's segments on both sides of each crest among its terms; the late rays bounce off the terrain's triangles. Levels are in dB re the direct sound in free field over the same distance, the early paths added in pressure in every band and the air's attenuation taken off each path over its own length.

A flat terrain (every height alike, 101 × 101 points 10 m apart) against the flat ground: the direct sound and the one reflection have the same distances to the bit and energies within 2.2e-16, and whole runs (20,000 rays, the same seed) the same level relative to free field in every band, the two models differing only in how the ground is cut into triangles.

A plane sloping up 20 % against the image source in the tilted plane: the reflected path's distance is the image's to 2e-9 m and its energy within 1.1e-15 straight up the slope, and within 2 mm and 7e-5 with the listener 160 m aside, where the ground also tilts across the profile, which the mean plane, level across it as the standards have it, leaves out.

The mean plane of an uneven profile is CNOSSOS-EU's, the least-squares line of the whole profile: `tests/test_terrain.py` checks it against ISO/TR 17534-4's test cases TC05 and TC06 (the plane's slope and height and the equivalent heights of the source and the receivers over it, to the tables' two decimals), and that the same ground on a 10 m and on a 5 m grid gives the same reflection.

A ridge, two slopes of 0.6 meeting at 241.9 degrees, with the source on one slope and the listener in its shadow on the other, against the Biot-Tolstoy-Medwin solution for the rigid wedge (Edge diffraction), integrated along 640 m of the edge, per octave band; the engine's path is the wedge of the string pulled taut over the profile, with the reflections off both slopes in its four terms:

source, listener (m over the slopes)	63	125	250	500	1k	2k	4k	8k	
2.0 at 30 m before, 1.5 at 30 m behind	-11.27	-14.21	-17.21	-20.21	-23.22	-26.23	-29.24	-32.25	exact, dB
	+0.00	+0.00	+0.00	+0.00	-0.00	+0.00	+0.00	+0.00	engine - exact
1.0 at 40 m before, 4.0 at 60 m behind	-13.32	-16.28	-19.28	-22.29	-25.30	-28.31	-31.32	-34.33	exact, dB
	+0.00	+0.00	+0.00	-0.00	-0.00	+0.00	-0.00	+0.00	engine - exact
3.0 at 20 m before, 1.5 at 40 m behind, 30 m aside	-9.81	-12.72	-15.71	-18.72	-21.73	-24.74	-27.75	-30.76	exact, dB
	+0.09	+0.09	+0.09	+0.09	+0.09	+0.09	+0.09	+0.09	engine - exact

The engine agrees within 0.005 dB for the first two pairs and is 0.09 dB high in every band for the third. A ridge sampled on a grid is the exact wedge, which sits at the crest with no width and has the slopes for its faces; the uniform theory's four terms, a high-frequency form of the exact solution, come that close to it here. The third pair is oblique, 30 m aside of the plane through the source and the crest.

A smooth hill, 12 m high with a Gaussian profile 25 m wide at half its height, the source and the listener 1.5 m up 40 m before and behind its crest, 2.71 m of path difference over it. A rounded crest has no exact solution, so the sound over it is compared with the engineering formulas the standards use for the same path difference: over a soft ground (absorption 0.999), Maekawa's thin screen without ground reflections (10 lg(3 + 20 N), ISO 9613-2 with $C_2 = 40$), and over a rigid ground, ISO 9613-2's figure with $C_2 = 20$ (10 lg(3 + 10 N)), which stands for the reflections off a typical ground:

	63	125	250	500	1k	2k	4k	8k	
soft ground	-11.94	-14.49	-17.37	-20.57	-23.59	-26.55	-29.57	-32.57	engine, dB
	-13.60	-16.28	-19.14	-22.07	-25.04	-28.03	-31.03	-34.03	thin screen, 10 lg(3 + 20 N)
	+1.66	+1.79	+1.76	+1.49	+1.45	+1.48	+1.46	+1.46	engine - thin screen
rigid ground	-6.81	-14.34	-16.72	-14.98	-19.67	-23.28	-25.45	-28.60	engine, dB
	-11.12	-13.57	-16.28	-19.14	-22.07	-25.04	-28.03	-31.03	ISO 9613-2, 10 lg(3 + 10 N)
	+4.32	-0.77	-0.43	+4.16	+2.40	+1.76	+2.57	+2.43	engine - ISO

Over the soft ground the engine lets 1.4 to 1.8 dB more over the hill than the thin screen, evenly across the bands. The engine bends the sound over the wedge of the hill's chords at the crest, open wider than a half-plane, where Maekawa's chart is for a thin screen; this check does not separate how much of the difference that makes. Over the rigid ground the reflections off both flanks join the direct wave, each by way of the ground near its foot, 0.86 m longer than the straight line to the crest (the source's image in the flat ground before the hill is that much farther from the crest than the source): in phase at 63 Hz (5.1 dB more than over the soft ground, 4.3 above the standard's typical ground), against it near 200 Hz, where that extra path is half a wavelength (the ground dip: 0.15 and 0.65 dB more than over the soft ground at 125 and 250 Hz, a little below the standard's figure), and from 500 Hz up 3.3 to 5.6 dB more than over the soft ground (1.8 to 4.2 above the standard), where the waves no longer interfere alike across a band and add in energy: 6 dB for four of a size, less because the reflected ones come from images deeper in the shadow. The standard's figure stands for a ground that reflects less than a rigid one. Both are engineering comparisons, not exact solutions.

Climbing out of the shadow behind the same hill over the rigid ground, from 1.5 to 29.5 m up in steps of 0.5 m, the level changes by at most 0.6 dB a step at 63 Hz and 3.0 dB at 8 kHz, where the edge of the shadow is sharpest (the line of sight clears the hill from 23.2 m up), and never falls by more than 1.8 dB a step, in the ground's interference and the lit zone's ripples:

up (m)	63	125	250	500	1k	2k	4k	8k
1.5	-6.81	-14.34	-16.72	-14.98	-19.67	-23.28	-25.45	-28.60
4.0	-9.64	-14.02	-16.65	-16.40	-19.95	-23.27	-25.98	-29.13
6.5	-9.86	-11.74	-15.29	-15.95	-19.10	-22.40	-25.13	-28.20
9.0	-8.00	-11.19	-14.31	-15.17	-18.31	-21.72	-24.51	-27.59
11.5	-6.44	-10.57	-13.44	-14.17	-17.38	-20.80	-23.59	-26.66
14.0	-5.21	-9.91	-12.21	-12.93	-16.04	-19.42	-22.22	-25.29
16.5	-4.36	-8.72	-10.61	-11.22	-14.06	-17.34	-20.11	-23.17
19.0	-3.66	-7.49	-8.57	-8.92	-11.12	-13.96	-16.55	-19.52
21.5	-2.85	-6.24	-6.14	-6.06	-6.95	-8.40	-9.65	-11.44
24.0	-2.06	-5.01	-3.52	-2.95	-2.20	-1.46	+0.32	+1.17
26.5	-1.24	-3.85	-1.18	-0.33	+0.86	+0.96	+0.93	+0.55
29.0	-0.51	-3.09	+0.64	+0.91	+0.26	+0.98	+0.45	+0.91

At 63 Hz the level first falls as the listener rises, from -6.8 dB at 1.5 m to -9.9 at 6.5 m: the reflection off the ground behind the hill, in phase with the wave over the crest at the ground, lags it more the higher the listener. From there every band rises through the edge of the shadow, 12.6 dB in the 2.5 m round the line of sight at 8 kHz, and every band but 125 Hz comes to within 1 dB of free field above it, where the direct sound, the ground's reflection and the crest's wave ripple together; 125 Hz stays 3 to 4 dB down (-3.9 and -3.1 dB at 26.5 and 29 m).

Cost. Rolling hills 2 km across on a 20 m grid (10,000 cells, 21,233 triangles with a building): the model builds in 0.3 s, its 8 diffracting edges (the building's) are found in 0.1 s, the early path over the hills in 0.4 s and a whole run with 20,000 rays takes 0.8 s, on four cores of a cloud machine with nothing else running. The terrain costs the late rays nothing beyond the triangles it adds, and the early sound the profile and the string over it, once per listener.

Outdoor scenes on the GPU

The GPU tracer (`simulation.late.device: gpu`, `both` or `auto`) follows the rays bent by wind and temperature, takes them through vegetation and lets them out through the open sky, as the CPU's does; the wave solver on the GPU steps its grid with the speed of sound per cell. The tests check each against the CPU, and `validation/outdoor_gpu.py` compares whole runs of the outdoor examples.

Chords, ray for ray. The GPU works in single precision, and its sines, cosines and arc tangents are only guaranteed to about 11 bits, which moved a few percent of the chords' ends by metres in a first port: the chords are therefore stepped without trigonometry (the ray's direction kept as a cosine and sine pair, the turn over a step summed as a series). 4,000 rays from random places in random directions, half of them leaving a plane they must not meet again, in four atmospheres (sound speed rising or falling 0.3/s to 100 m up, a wind of 6 m/s, and a wind of 4 m/s with the speed falling 0.1/s): the GPU's chords turn back where the CPU's do (every one), point the same way at their ends to 3e-5, and stretch as far to a part in 5,000 for 99 % of them (the rest end where the chord's tolerance is met a little sooner or later as the turn gathered along it rounds, by up to 1.2 % where the wind shears hard just above the ground). Cut to a common length, the two chords are the same curve: ends within 0.32 mm and times within 4e-6 s, over 400 m.

Energy and time. In a field 400 m across under a 200 m sky, 200,000 rays per run, listeners 100 m off downwind, upwind, across the wind and 60 m up: the energy per listener and band and the mean time it arrives at, the rays bent up out of the field and the energy over time the weight window reads, agree with the CPU's within the noise of the ray count, in a warmer sky (sound bent down), a cooler one (bent up) and a wind with a cooler sky. The energy over time is compared as what is left of it from each 10 ms window on: in single precision a ray bent over many chords and bounces arrives a little sooner or later now and then, and its energy lands in the next window (on every GPU, and more on Direct3D 12's software GPU in Windows CI). With only mirror reflections in the warmer sky, one window held 7 % less than the CPU's and the next 4 % more (15 % and 8 % on Direct3D 12), while what is left from each window on agreed to 0.1 % (0.2 % on Direct3D 12) wherever it was at least a hundredth of the whole: a shift the weight window, rounded to quarter octaves, cannot tell. Through a crown of foliage and a stand of stems, the energy each band loses, the energy scattered on and where it goes, with and without an atmosphere, agree the same way; a tree belt of 17 media is declined with its reason and traced on the CPU.

Whole runs. `validation/outdoor_gpu.py` simulates the outdoor examples with their own settings, but for grounds that scatter 0.1 where the examples' scatter nothing (over those no late ray reaches the stage's listeners or the hillside's crest), once per device and once more on the CPU with the next seed, and compares the late sound's level per band and its centre time at every listener (the mean absolute difference over listeners and bands); the next seed shows how much the ray count's noise alone moves them:

Scene	Faces	Rays	Device	Late (s)	Level diff (dB)	Centre time diff (ms)
open_air_stage	12	50,000	cpu, next seed	0.59	4.21	6.0
open_air_stage	12	50,000	gpu	20.21	0.02	0.2
tree_belt	30	200,000	cpu, next seed	1.12	1.20	22.3
tree_belt	30	200,000	gpu	33.30	1.29	13.0
hillside	3,730	50,000	cpu, next seed	0.34	2.04	1.9
hillside	3,730	50,000	gpu	11.49	1.88	1.6

On SwiftShader, a software Vulkan driver on the container's 4 cores (the CPU runs took 0.3 to 1.1 s), which shows that the two agree, not how fast a GPU is; `python validation/outdoor_gpu.py` times your own. The GPU rows are from that run, when the port was added; the CPU's next-seed rows come out the same with this version's code (the container that checked them has no GPU driver). The tree belt's rows are from when it traced its image sources to the first order; at the second, as it does now, the house's reflection keeps its ground partners, the late sound changes with it, and the next seed differs by 1.32 dB and 20.6 ms (seeds 1 and 2; its GPU row needs a run on a GPU). The tree belt's rays through its foliage and stems come out as far from the CPU's as the next seed's do. The stage in the wind agrees to 0.02 dB with the same seed, whose rays start where the CPU's do; the next seed starts them elsewhere and moves the late sound at the two listeners downwind by 4.2 dB on average (upwind, in the wind's shadow, no late ray arrives in either run). The stage's eddies play no part in this: they act on the early paths only, and the script compares the late sound (without them the next seed differs by the same 4.21 dB). On the hillside the GPU was 1.9 dB from the CPU in that run, against the next seed's 2.0 dB now: the listener in the shadow gets no late rays, the one on the crest about the same level from every run, and the one on the slope a level that moves by several decibels with the seed, so one run of each, as here, cannot tell the GPU's difference from the rays' noise there.

The wave solver in wind. The GPU steps the tests' grids with a speed of sound that grows with height, bending the sound as the rays do, to within 3e-5 of the largest pressure of the CPU's (single-precision rounding; 2e-3 on Intel Macs, whose GPU drifts further), its matched layer within 1e-5, and the 63 and 125 Hz bands of a scene whose sound speed rises from 337 m/s at the ground to 402 m/s at the top of the grid within 1e-5, the two naming the same slowest and fastest speed of sound.

Weather zones against a Crank-Nicholson parabolic equation

`validation/weather_zones.py` checks the engine where the weather changes on the way (see Weather zones). The reference is a Crank-Nicholson parabolic equation written for the script (Salomons, *Computational atmospheric acoustics*, appendix G): a narrow-angle equation marched in steps of a quarter wavelength on a grid of a tenth of a wavelength, with the ground's impedance as its lower boundary, a Gaussian starter and an absorbing layer at the top, which takes the speed of sound of whichever zone each step lies in. The script first checks the reference itself: against the exact solution for the reflection off grass in still air it is within 0.17 dB at 63 to 500 Hz out to 800 m, and against the engine's own parabolic equation in the `windy` preset over the whole way it is within 0.35 dB to 400 m, 0.5 dB at 63 Hz 800 m out and 1.4 dB at 500 Hz 800 m out, where the two equations' starters and absorbing layers part.

The source is 2 m high and the listener 1.5 m, over grass (flow resistivity 200 kPa s/m²), and the weather changes 150 m from the source:

case	from the source	past 150 m
shelter downwind	`windy` (5 m/s 10 m up, towards the listener)	0.5 m/s
gust downwind	0.5 m/s	5 m/s
shelter upwind	5 m/s towards the source	0.5 m/s
inversion past a lake shore	`neutral` (still air)	`clear_night` (air warming with height)
cool water downwind	`windy`	from 100 to 300 m the ground at 12 °C instead of 20

The parabolic equation through the zones. The engine's parabolic equation marches through the zones, changing its speed of sound where its path crosses a border. The level re free field (dB) in pure tones, and in brackets the difference from the reference of the engine's narrow-angle parabolic equation, its default, and then of its wide-angle one (see The wide-angle parabolic equation):

case, tone	100 m	200 m	400 m	800 m
shelter downwind, 63 Hz	6.67 (-0.22, -0.07)	7.14 (-0.10, -0.08)	6.65 (-0.08, -0.06)	5.49 (-0.08, -0.04)
shelter downwind, 125 Hz	3.16 (+0.09, +0.03)	0.97 (+0.14, +0.03)	-4.63 (+0.04, +0.06)	-15.71 (-0.22, +0.07)
shelter downwind, 250 Hz	-21.10 (-0.34, -0.02)	-22.24 (-0.17, +0.00)	-25.09 (-0.24, +0.02)	-28.48 (-0.38, -0.03)
shelter downwind, 500 Hz	-4.54 (+0.01, -0.01)	-4.37 (+0.13, +0.03)	-7.94 (+0.10, +0.08)	-9.24 (+0.08, +0.10)
gust downwind, 63 Hz	5.50 (-0.02, -0.04)	6.09 (-0.14, -0.04)	7.35 (-0.14, -0.08)	4.00 (-0.35, -0.11)
gust downwind, 125 Hz	2.41 (+0.02, +0.02)	1.11 (-0.04, +0.02)	-7.70 (+0.17, -0.09)	-12.62 (+0.03, +0.03)
gust downwind, 250 Hz	-10.56 (-0.04, +0.03)	-25.63 (-0.53, -0.09)	-23.87 (+0.39, +0.07)	-17.53 (+0.18, +0.02)
gust downwind, 500 Hz	-10.77 (-0.07, -0.01)	-11.59 (-0.12, +0.05)	-2.05 (-0.02, +0.05)	1.88 (-0.36, -0.36)
shelter upwind, 63 Hz	3.77 (+0.16, +0.05)	2.59 (+0.01, +0.08)	1.97 (+0.02, +0.05)	0.45 (+0.02, +0.02)
shelter upwind, 125 Hz	0.53 (-0.16, +0.07)	-2.60 (-0.26, +0.12)	-5.98 (-0.23, +0.09)	-12.88 (-0.20, +0.06)
shelter upwind, 250 Hz	-8.22 (-0.01, +0.07)	-17.69 (+0.08, +0.14)	-29.33 (+0.07, +0.12)	-39.75 (-0.12, -0.07)
shelter upwind, 500 Hz	-17.84 (+0.04, +0.20)	-35.34 (+0.02, +0.45)	-47.37 (-0.09, +0.48)	-61.17 (-0.14, +0.42)
inversion past a lake shore, 63 Hz	5.36 (+0.00, -0.04)	5.43 (-0.03, -0.04)	5.71 (-0.01, -0.06)	5.08 (-0.02, -0.07)
inversion past a lake shore, 125 Hz	2.29 (+0.01, +0.02)	0.69 (-0.00, +0.02)	-3.31 (+0.05, +0.01)	-16.52 (-0.12, -0.01)
inversion past a lake shore, 250 Hz	-10.04 (-0.02, +0.03)	-20.50 (-0.19, -0.00)	-27.08 (-0.01, +0.00)	-26.87 (-0.02, +0.05)
inversion past a lake shore, 500 Hz	-11.44 (-0.07, -0.00)	-15.67 (-0.11, +0.01)	-14.45 (-0.11, +0.01)	-8.31 (-0.07, +0.03)
cool water downwind, 63 Hz	6.67 (-0.22, -0.07)	7.70 (-0.23, -0.05)	8.43 (-0.21, -0.05)	4.36 (-0.42, -0.03)
cool water downwind, 125 Hz	3.16 (+0.09, +0.03)	1.03 (+0.18, +0.10)	-14.52 (+0.73, +0.32)	-15.33 (+0.22, +0.18)
cool water downwind, 250 Hz	-21.10 (-0.34, -0.02)	-24.13 (-0.18, +0.06)	-21.07 (+0.17, +0.07)	-15.33 (-0.01, -0.04)
cool water downwind, 500 Hz	-4.54 (+0.01, -0.01)	-0.95 (-0.07, +0.04)	3.66 (+0.01, -0.03)	0.89 (-1.19, -1.20)

The engine is within 0.15 dB of the reference on average and 1.19 dB at most, at 500 Hz 800 m out over the cool water, where the two equations part by 1.4 dB in the uniform wind too. The wide-angle equation is within 0.09 dB on average and 1.20 dB at most, at the same point, where the engine's two equations agree within 0.01 dB and the reference, a narrow-angle equation with a Gaussian starter, is the one that parts; elsewhere it is within 0.5 dB, the most 47 dB down in the upwind shadow at 500 Hz. The shelter downwind bends the sound down over the first 150 m and then lets it spread, 13 dB less than the uniform wind at 250 Hz 800 m out and 10 dB less at 500 Hz, where the wind would bring the sound back to the ground; the gust turns still air's ground dip into the wind's pattern past the border; the inversion past the shore carries 20 dB more than still air at 500 Hz 800 m out; and the cool water deepens the 125 Hz dip 400 m out by about 1.5 dB.

The mean profile along the way. The early paths bend through one profile, the mean along their way, each zone's weighted by the length of the way over it. The parabolic equation through that mean profile against the parabolic equation through the zones, at 500 Hz, 1 kHz and 2 kHz (averaged over the band's frequencies; the profile the rays' bands, 4 and 8 kHz, take), from 200 m on (nearer, the way lies under one weather); the level re free field through the zones, and in brackets the mean profile's difference from it:

case, band	200 m	400 m	800 m
shelter downwind, 500 Hz	-1.35 (+0.51)	-5.21 (+3.33)	-6.13 (+4.22)
shelter downwind, 1 kHz	3.11 (+1.38)	3.87 (-0.72)	4.06 (+1.75)
shelter downwind, 2 kHz	3.45 (+1.84)	1.96 (+3.60)	0.33 (+6.57)
gust downwind, 500 Hz	-9.12 (+0.72)	-0.49 (+3.32)	-0.93 (+1.97)
gust downwind, 1 kHz	3.22 (+1.28)	2.24 (+1.25)	-0.88 (-2.71)
gust downwind, 2 kHz	1.75 (+3.78)	-0.30 (+3.55)	-0.33 (-1.23)
shelter upwind, 500 Hz	-34.56 (+1.97)	-46.14	-58.52
shelter upwind, 1 kHz	-62.73	-77.33	-89.45
shelter upwind, 2 kHz	-58.80	-79.26	-96.83
inversion past a lake shore, 500 Hz	-13.92 (+0.29)	-11.21 (-0.02)	-4.08 (+1.21)
inversion past a lake shore, 1 kHz	1.91 (+2.02)	5.98 (+2.05)	3.63 (+3.32)
inversion past a lake shore, 2 kHz	3.67 (-0.13)	5.80 (+1.65)	4.04 (+2.48)
cool water downwind, 500 Hz	1.99 (+0.08)	0.97 (+0.29)	1.21 (-0.02)
cool water downwind, 1 kHz	3.47 (-0.06)	-0.66 (-0.01)	-2.39 (-0.10)
cool water downwind, 2 kHz	3.11 (+0.18)	1.89 (-0.34)	-1.10 (-0.11)

Over the 37 points above -40 dB re free field the mean profile is within 1.6 dB on average and 6.6 dB at most; the 8 points deeper in the upwind shadow are left out, as sound there is inaudible next to any other and the parabolic equation itself is at the floor of its absorbing layer. Over the cool water, which changes the air near the ground for 200 m, the mean profile is within 0.35 dB. Where the wind changes, or an inversion sets in past the shore, it gives more sound at most points, mostly 1 to 4 dB and 6.6 dB at most (2 kHz, 800 m downwind of the shelter), but up to 2.7 dB less at 800 m past the gust and 0.7 dB less at 1 kHz 400 m past the shelter: one profile cannot both bend the sound down over the first 150 m and let it go straight after, and bends it a little the whole way instead. That is the cost of the equivalent profile the standards take (Nord2000, Harmonoise), and a reason the bands up to 2 kHz come from the parabolic equation, which goes through the zones.

Barriers and buildings in wind against a full-wave solution

`validation/barrier_wind.py` checks the weather's effect behind barriers and buildings (see How the simulation bends sound): the level in the weather less the level in still air, which the engine takes from its parabolic equation with the barriers and buildings inside (a barrier lets nothing through below its top, a roof is a rigid floor) and puts on the paths it finds in still air. Two references are written for the script, and the wind is a logarithmic profile over a roughness of 5 cm, 5 or 10 m/s 10 m up, towards the listener or the source:

- **A full-wave solution**: the Helmholtz equation round a source on an axis, in air whose speed of sound changes with height, solved by coupled modes stepping in range (Evans 1983): exact in range, a twentieth of the shortest wavelength in height, a perfectly matched layer on top of a domain 30 m high (60 m where given). Barriers are thin rigid rings round the source and a building a rigid ring block; the wind blows through them; 63 to 500 Hz, over rigid ground and over grass (Miki's impedance, 200 kPa s/m²). Where given it is also solved along a plane, from a line source with straight barriers (the two dimensions of the engine's equation, where nothing focuses the sound going back between two barriers as a ring does). It checks itself: over open rigid ground it is within 0.002 dB of the exact field of the source and its image, round the axis and along the plane; a barrier in still air is -0.10 to +0.03 dB from a straight barrier's four paths (the uniform theory of diffraction) at 125 and 250 Hz as a ring and +0.04 to +0.09 dB along the plane; over open ground in 5 m/s of wind it is within 0.25 dB of the engine's parabolic equation at each frequency of the 125 Hz band; and deep in the upwind shadow of a barrier 4 m high 15 m from a source 1 m up, at 552 Hz, the domain 30 m high is within 0.00 dB of one 90 m high 120 m away in 5 m/s of wind (1.2 dB in 10 m/s) but 1.0 and 10.5 dB off 150 and 200 m away (15 and 45 dB in 10 m/s), and 60 m high within 0.04 dB in 5 m/s and 1.24 dB in 10 m/s, which is why the cases upwind beyond 120 m take it.
- **A parabolic equation started behind the barrier** from the exact field of a rigid straight barrier over rigid ground (the four paths), 1 m behind it (within 0.18 dB of those paths in still air), and marched in a wind that blows behind the barrier only (the engine's equation is given the same), 125 Hz to 8 kHz (from 4 kHz up to 100 m only): the `windy` preset downwind and upwind, `clear_night` and `sunny_day`, barriers 3 and 6 m high 20 and 50 m from a source 1 m up, listeners 50, 100 and 200 m away, 1.5 and 4 m up.

The full-wave cases (source 1 m up unless given; listeners 1.5 m up unless given):

case	on the way	listeners (m away)
near	barrier 15 m away, 4 m high; also along a plane	30, 60, 120; 60 at 4 m up
far	barrier 100 m away, 4 m high	110, 130; 130 at 4 m up
rail	source 0.5 m up, barrier 4 m away, 3 m high	25, 50, 100
steep 45 and 60	source 0.5 m up, barrier 4 m away, 4.5 and 7.43 m high (its top 45 and 60 degrees up from the source)	15, 30, 60
roof	building 40 to 52 m away, 6 m high	80, 150
two	barriers 45 and 55 m away, 8 m high; also along a plane	100, 150
two 40	barriers 20 and 60 m away, 3 and 5 m high; 125 and 250 Hz; also along a plane	100, 160
near upwind	as near, upwind only, 5 and 10 m/s, on a domain 60 m high	60, 120, 200; 120 at 4 m up
rooftop	source 7 m up on a building from the axis to 5 m, 6 m high	27.5, 42.5, 100
mast	source 10 m up, building 15 to 25 m away, 6 m high	40, 45, 50, all 10 m up

Against the full-wave solution. The engine’s weather effect less the solution’s, the largest difference (dB) over the listeners and the bands 63 to 500 Hz, at the engine’s own frequencies (the solution’s tops on its cells), and the solution’s effects:

case	reference	ground	5 m/s downwind	5 m/s upwind	8 m/s downwind	10 m/s downwind	10 m/s upwind	the effect
near	ring	rigid	0.32	0.62		0.63		-6.8 to +10.3
near	ring	grass	0.29	0.57		0.46		-7.3 to +8.3
near	plane	rigid	0.27	0.61		0.53		-6.9 to +10.4
near	plane	grass	0.24	0.53		0.45		-7.4 to +8.4
far	ring	rigid	0.38	0.42		0.67		-11.6 to +2.8
far	ring	grass	0.47	0.41		0.74		-7.8 to +10.4
rail	ring	rigid	0.25	0.38		0.27		-6.4 to +3.6
rail	ring	grass	0.21	0.35		0.25		-6.1 to +9.5
steep 45	ring	rigid	0.43	0.45		0.98		-6.8 to +4.1
steep 45	ring	grass	0.17	0.41		0.50		-1.1 to +3.5
steep 60	ring	rigid	0.54	0.53		0.41		-7.4 to +6.5
steep 60	ring	grass	0.10	0.40		0.46		-1.3 to +3.1
roof	ring	rigid	0.15	0.24		0.32		-6.1 to +9.7
roof	ring	grass	0.40	0.21		0.51		-6.8 to +8.1
rooftop	ring	rigid	0.09	0.18		0.31		-8.4 to +3.7
rooftop	ring	grass	0.13	0.17		0.22		-3.4 to +2.4
mast	ring	rigid	0.03	0.03		0.05		-0.5 to +0.3
mast	ring	grass	0.02	0.02		0.04		-0.6 to +0.3
two	ring	rigid	1.00	1.68		1.49		-6.7 to +17.3
two	ring	grass	0.87	0.75		1.63		-4.4 to +5.2
two	plane	rigid	1.00	1.63		1.61		-6.6 to +17.4

case	reference	ground	5 m/s downwind	5 m/s upwind	8 m/s downwind	10 m/s downwind	10 m/s upwind	the effect
two	plane	grass	0.82	0.69		1.73		-2.2 to +5.3
two 40	ring	rigid	2.23	0.69	4.95	0.73		-5.9 to +2.0
two 40	ring	grass	1.01	0.40	1.28	1.15		-1.3 to +0.6
two 40	plane	rigid	2.19	0.75	4.91	0.77		-5.9 to +2.1
two 40	plane	grass	0.96	0.36	1.21	1.06		-1.3 to +0.7
near upwind	ring	rigid		0.71			1.64	-14.5 to +9.3
near upwind	ring	grass		0.61			1.37	-18.8 to +2.9

Behind one barrier or building the engine is within 0.71 dB in 5 m/s of wind (at 500 Hz 120 m behind the near barrier 4 m up, upwind over rigid ground) and 0.74 dB downwind in 10 m/s. Upwind in 10 m/s it comes out low, by 0.2 to 1.64 dB, the more the higher the band and the farther the listener (1.64 dB at 500 Hz 120 m away 4 m up, 1.0 to 1.4 dB at 250 Hz 120 and 200 m away): the narrow-angle equation's error, as the sound comes down steeply into the shadow. Behind a top 45 degrees up from the source it is within 0.98 dB in 10 m/s at 250 Hz over rigid ground, 60 degrees up 0.54 dB. From a source 1 m over the roof of a building 6 m high it is within 0.31 dB (250 Hz 100 m away in 10 m/s downwind over rigid ground), and with the source and the listeners 10 m up, higher than the building (mast), within 0.05 dB. Behind two barriers 10 m apart it is off by up to 1.73 dB, and 40 m apart by up to 4.95 dB at 250 Hz in 8 m/s downwind over rigid ground (2.23 dB in 5 m/s, 1.28 dB over grass): the equation marches away from the source only, so nothing goes back and forth between the barriers in it. Along a plane the solution's effect is within 0.14 dB of the ring's, so the ring's focusing between the barriers does not explain it. The limits of the script (`LIMITS`) are these largest errors rounded up to 0.05 dB, so that a change that makes any of them worse fails. Deeper than 25 dB below still air (listed, not judged) the engine is off by up to 4.2 dB 39 dB below (500 Hz 200 m away upwind in 10 m/s over grass) and 20 dB 52 dB below.

Against the started parabolic equation. The engine's weather effect less the reference's, the largest difference (dB) over the listeners where the reference is within 25 dB of still air, at the engine's own frequencies (16 an octave at 4 kHz, 32 at 8 kHz), and the reference's still air against the four paths it starts from:

case	125 Hz to 2 kHz	4 and 8 kHz	starting field
`windy` downwind, barrier 3 m high 20 m away	0.27	0.27	0.06
`windy` downwind, 6 m high 20 m away	0.42	0.17	0.18
`windy` downwind, 3 m high 50 m away	0.27	0.30	0.02
`windy` upwind, 3 m high 20 m away	0.42	0.25	0.06
`windy` upwind, 3 m high 50 m away	0.47	0.26	0.02
`clear_night`, 3 m high 20 m away	0.15	0.18	0.06
`sunny_day`, 3 m high 20 m away	0.24	0.11	0.06

The engine is within 0.47 dB everywhere (0.5 allowed). Deeper in the upwind shadow, where the effect is listed but not bounded, it is within 0.81 dB down to 57 dB below still air, but 6.5 and 17.4 dB off at 2 kHz 72 and 74 dB below.

The engine end to end. `simulate`'s early paths (their coherent sum; reflections off both sides of an edge) in the wind less in still air, against the full-wave solution's effect, over rigid ground with the barriers and the building 1 km long: the largest difference (dB) over the listeners and the bands 63 to 500 Hz where the solution is within 25 dB of still air:

case	reference	5 m/s downwind	5 m/s upwind	8 m/s downwind	10 m/s downwind	10 m/s upwind
near	ring	0.33	0.66		0.64	
far	ring	0.41	0.43		0.66	
rail	ring	0.26	0.38		0.29	
steep 45	ring	0.44	0.45		1.01	
steep 60	ring	0.53	0.51		0.43	
roof	ring	0.15	0.24		0.32	
two	ring	1.00	1.69		1.53	
two 40	ring	2.23	0.69	4.95	0.73	
two 40	plane	2.19	0.76	4.91	0.77	
near upwind	ring		0.71			1.70

End to end the engine's difference from the solution is within 0.08 dB of its parabolic equation's alone, 0.18 dB behind the top 60 degrees up (0.15 dB allowed); behind the top 45 degrees up it is 1.01 dB off in 10 m/s and behind two barriers 1.69 dB, as its equation is.

Convergence. The engine's runs behind barriers take a quarter wavelength's height step from 1 kHz up (a tenth below) and 8 frequencies a band up to 2 kHz, 16 at 4 kHz and 32 at 8 kHz. The weather's effect from its parabolic equation alone against finer runs (`--only convergence`), the largest difference (dB) over the listeners and both winds of 5 m/s: a tenth's step against the quarter's, and 32 frequencies an octave against 8 and 16, 64 against 32. `high` is a barrier 3 m high 10 m from a source 1 m up, with listeners 25 m away 10 and 14 m up and 40 m away 1.5 m up:

case	ground	step, 1 and 2 kHz	step, 4 and 8 kHz	frequencies, 63 Hz to 2 kHz	4 kHz	8 kHz
near	rigid	0.09	0.01	0.22	0.04	0.04
near	grass	0.07	0.05	0.08	0.07	0.03
rail	rigid	0.16	0.08	0.13	0.06	0.13
rail	grass	0.18	0.09	0.08	0.10	0.18
roof	rigid	0.06	0.05	0.22	0.04	0.03
roof	grass	0.08	0.09	0.12	0.03	0.02
steep 60	rigid	0.28	0.97	0.53	0.18	0.51
steep 60	grass	0.23	2.19	0.40	0.21	0.13
high	rigid	0.07	0.01	0.11	0.02	0.05
high	grass	0.04	0.01	0.09	0.02	0.02

The quarter wavelength's step costs 2 to 2.5 times less than the tenth's and holds within 0.2 dB but 11 m behind the top 60 degrees up, where it is 0.3 dB off at 1 and 2 kHz and 1.0 and 2.2 dB at 4 kHz over rigid ground and grass. The ground's interference fringes behind a barrier are dense in frequency: at 8 kHz 16 frequencies an octave are off by up to 1.8 dB against 32 over rigid ground and 6.4 dB at `high`'s listener 14 m up, where the fringes fall in step with them (a one-off check), which is why the 8 kHz band takes 32, within 0.18 dB of 64 but 0.51 dB 11 m behind the steepest top.

Steep points behind a building. Where the sound reaches a point more steeply than 15 degrees the engine's runs take the wide-angle equation, which holds a building's roof as a rigid floor on its height steps, as the narrow-angle one does between them (`--only steep`). 60 m from a source 1 m up, behind a building 20 to 30 m away with its roof 8 m up, in 10 m/s downwind over rigid ground, the weather's effect steps by 0.30, 0.05, 0.02, 0.01 and 0.35 dB at 125 Hz to 2 kHz between listeners 15 and 15.5 m up, either side of that angle: the narrow-angle equation's own error below it. From 10 to 24 m up it is within 0.31 dB of the full-wave solution's at 125 Hz and 0.09 dB at 250 Hz; the wide-angle equation alone within 0.02 and 0.08 dB, the narrow-angle one within 0.45 and 0.09 dB. A source or a listener on a roof starts or ends the march over it: from a source 1 m over a long rigid roof in still air both equations are within 0.11 dB of the source and its image in the roof 10 to 60 m away (`tests/test_parabolic_obstacles.py`).

Continuity. How much the level steps as one thing changes a little, end to end (`simulate`, the coherent sum of the early paths), the most in any band, in the `windy` preset down- and upwind and in still air (a one-off check). The scene: a source 1 m up over hard ground and a barrier 4 m high 15 m away across the way:

what changes	in the wind (dB)	in still air (dB)	aim (dB)
the wind, from none to 0.01 m/s (0.1 m/s)	0.024 (0.24)		0.2 at 0.1 m/s
the barrier's top, 1 mm higher (listeners 40 and 60 m away)	0.020	0.006	0.01
a listener 40 m away, 1 cm higher through the line of sight over the top	0.077	0.070	0.05
a building's roof 25 to 37 m away, 1 cm higher, from 6 to 6.3 m (listeners 60 and 120 m away, 5 m off its middle)	0.14	0.021	0.1 a cell
a listener 60 and 120 m away behind a barrier slanted at 63 degrees to the way, 0.01 degrees round, from 0.40 to 0.60 degrees	0.017	0.018	
a listener 2 cm further, where the parabolic equation starts (a 1.5 m barrier 2 m from a source 0.5 m up)	0.24	0.16	
a listener 1 mm either side of the shadow boundary of a barrier's end, 60 and 120 m away	0.04 and 0.22	0.04 and 0.17	1
a listener 10 cm across a roof edge's shadow boundaries, of the direct sound and of the ground reflection, 28 and 43 m from a source 1 m over the roof (and the source and the listener swapped)	0.11 and 0.09	0.09 and 0.04	
a listener 2 cm further, where the parabolic equation starts, 12.75 m from a source 1 m over a roof 6 m high (and the two swapped)	0.13	0.024	
a listener 1.5 m over a roof 45 m away, 1 mm either side of the shadow boundary of a barrier's end (and the two swapped)	2.83	2.79	
a listener 1 mm either side of the line of an end below the line of sight seen from above (a barrier 1 m high 30 m away), 100 and 200 m away	12.8 and 6.0	0.01	

The wind's effect grows in proportion to the wind (500 Hz, 100 m away and 4 m up), with no step as it starts. The roof stands at its own height between the parabolic equation's cells (the field over it mirrored about that height), so as it rises the weather's effect behind it changes smoothly with it: by up to 0.13 dB a centimetre at 1 kHz downwind and 0.14 dB at 4 to 8 kHz upwind, 0.03 dB a millimetre, and from 63 to 500 Hz by 0.03 dB at most, about as in still air; on the grid it stepped from one cell to the next by up to 0.50 dB at 1 kHz in the same places. Between two whole degrees the listener takes both runs' effects, so the level turns smoothly round. The steps at a barrier's end come from the plane's two dimensions (see What is not modelled yet): the paths over the top take the barrier's effect, past the end too as they fade out there, and the direct sound the open ground's, and the paths round the end change as the open ground's field along their way does. Across the end's shadow boundary the level steps by at most 0.04 dB a millimetre 60 m away, 0.07 dB 120 m downwind and 0.22 dB 120 m upwind, where still air steps by 0.17 dB (more with eddies and at 4 and 8 kHz in weather zones); across the line of an end below the line of sight, where the direct sound goes from the barrier's effect to the open ground's, by up to 0.94 dB a millimetre from 63 to 250 Hz, 4.7 dB from 500 Hz to 2 kHz 100 m away and 6.0 dB 200 m away, and 12.8 dB at 8 kHz upwind (see Known issues). From a source or to a listener on a roof the equation holds the building, and across the roof edge's shadow boundaries the weather's effect jumps by at most 0.04 dB in any band, octave or third octave, against the steps either side (`tests/test_barriers_in_wind.py`); where the equation starts, the effect steps by up to 0.12 dB (see Known issues).

Rows of roofs and barriers against a wave calculation

`validation/rows.py` checks the solver for sound that passes close over a whole row of edges (`acoustic_spaces.engine.rows`): the roofs along a street, a roof and the parapet behind it, screen after screen. Near grazing incidence each edge of such a row stands in the others' transition zones, and paths over one edge or two do not add up to the field (see Known issues). The solver marches the engine's parabolic equation over the row in still air, in the vertical plane through the source and the listener, from both ends, with a barrier or a wall as a screen that lets nothing through below its top and a roof as a rigid floor, and adds for each rigid edge what the uniform theory of diffraction gives it beyond such a screen, lit by the marched field at the edge. That theory takes a barrier for a half-plane, lit by the end and by its image in the surface it stands on. A barrier alone between the ends on a rigid ground, or on a rigid roof under both, is with its image in that surface a rigid strip in free space, whose field has an exact solution: there the solver adds what the strip sheds beyond the march's screen instead (`acoustic_spaces.engine.strip`, up to a height of 32 over the wavenumber, five wavelengths), which holds the sound that goes on between the barrier's top and the surface too. Each march begins at its first obstacle, from the field that its end and the end's image in the ground have there (for an end over a roof: at the first top past it, with the end's image in the roof), and its field becomes a point source's by the length of the path pulled taut over the tops, so that the way from the street up to a roof may be steep. Outdoor simulations take a row's sound from it (see Sound over rows of barriers and buildings): the check is of the solver by itself and, for the cases on a ground, of whole simulations (further down).

The reference is a wave calculation written apart from the engine: finite differences in time in two dimensions (cells of 2 cm, rigid obstacles as whole cells, matched layers on every side, 0.5 s), turned into a point source's field by the integral over the wavenumber along the edges. With one rigid screen it is within 0.14 dB of the exact solution, the direct sound and the Biot-Tolstoy-Medwin integral over the edge, which the script checks on every run. Its levels for the 149 cases, and for 66 of them with the ends offset along the edges (see the ways at a slant below), are stored beside the script (`rows_wave.json`); `--references` computes them again, 3 to 6 minutes for a listener's place on one core.

The rows stand between x = 0 and 10 m, the source 10 m before x = 0 and the listener 5 m past x = 10 m: one screen; one roof 4, 9 or 10 m wide; a roof 9 m wide and a screen 1 m past it, with its top level with the roof or up to 0.5 m below or above it; two roofs 4 m wide and 2 m apart; two roofs side by side, the second 0.2 or 1 m lower or higher; two and three screens; three roofs 2 m wide. In free space the source is a centimetre to 2 m above the roofs' plane and the listener a centimetre above it or 0.5 or 2 m below (the screen whose top passes that plane rises up to half a metre above both). On a rigid ground 6 m below the tops, the source is a centimetre or half a metre above the tops and the listener a centimetre above them or half a metre below, or either stands 1.5 m above the ground. The table gives the level in the octave bands from 63 to 500 Hz, which a scene adds in pressure, less the wave calculation's (dB), by the angle the sound bends by at the row's first or last edge. The simulation's columns are the early stage of the same scenes with the paths over the edges alone (the roofs and screens given as a model file's faces), either way round, with every sequence of reflections tested (`method: image_sources`); `first_order` has no path over two edges and so no sound where neither end sees over the row. In the last five lines the source stands over a roof instead, 0.5 or 3 m before x = 10 m, a centimetre or 1.5 m above it: the roof ends there, with no parapet or one 0.1, 0.3 or 1.1 m high on its edge, and the listener stands 5 m past the edge, 2 m below the roof or 1 m above it; or the roof goes on under the listener, who is 1.5 m above it, with a screen of those heights at x = 10 m:

rows	cases	march alone	the solver	simulation, `second_order`	simulation, `first_order`
free space, up to 6°	52	-1.96 to -0.03	-0.38 to +0.18	-20.60 to +6.64	-21.72 to +14.16, or no sound
free space, 22° down to the listener	8	-2.26 to -0.88	-0.16 to +0.14	-20.57 to +1.09	-1.22 to +8.42
roofs and one screen on a ground, up to 6°	16	-1.66 to -0.28	-0.50 to +0.65	-10.63 to +1.74	-11.79 to +7.99
the same, 13 to 24° up from the source	8	-2.70 to -1.43	-0.74 to +0.41	-6.22 to +7.86	-0.93 to +9.05, or no sound
the same, 42° down to the listener	12	-5.29 to -2.19	-0.71 to +1.24	-4.69 to +7.35	-1.94 to +8.21, or no sound
three screens on a ground, up to 6°	4	-3.53 to -1.10	-1.26 to +0.07	-11.27 to +2.21	-3.35 to +5.66
three screens on a ground, 24°	2	-6.19 to -2.90	-2.55 to -0.30	-1.97 to +4.59	+3.53 to +6.51, or no sound
three screens on a ground, 42°	3	-9.20 to -4.06	-3.42 to -0.09	-2.65 to +4.57	+2.01 to +7.61, or no sound
a roof under the source, to its edge	8	-2.16 to +0.81	-0.18 to +0.05	-0.10 to +0.23	-0.10 to +0.23
the same, a parapet 1.1 m high on the edge	8	-9.90 to +0.33	-0.82 to +0.48	-1.10 to +1.13	-1.12 to +1.17
the same, the parapet 0.1 or 0.3 m high	16	-5.75 to +1.02	-1.13 to +1.13	-1.06 to +1.95	-0.98 to +1.56
a screen 1.1 m high on a roof under both ends	4	-8.75 to +0.88	-0.05 to +0.42	-3.43 to +2.68	-3.43 to +2.68
the same, the screen 0.1 or 0.3 m high	8	-4.85 to +0.93	-0.13 to +0.50	-2.91 to +1.38	-2.91 to +1.38

The march alone is low: a screen that reflects nothing lets through less than a rigid edge. With what the edges add the solver is within 0.38 dB of the wave calculation near grazing incidence in free space (0.24 dB but for two roofs side by side with a step of 1 m between them, in the 63 Hz band), the roofs of different heights and the screen whose top passes the roof’s plane included (within 0.18 dB as that top rises from 0.5 m below the plane to 0.5 m above), and within 0.65 dB over roofs and one screen on a rigid ground. Where the sound bends widely at the row’s ends: within 0.16 dB down to a listener 2 m below the last top and 5 m past it in free space (22°); on the ground within 0.74 dB up from a source 4.5 m below the first top (13 to 24°); and down to a listener at street level right behind the row (42°) within 0.15 dB behind one roof, 0.47 dB behind one screen, 0.71 dB behind two roofs and 1.24 dB behind a roof with a screen 1 m past it. With the source over a roof the solver is within 0.18 dB up to the roof’s edge, where the march alone is up to 2.16 dB low; with a parapet 1.1 m high on the edge within 0.82 dB (in the 63 Hz band, where the parapet is a fifth of a wavelength high; 0.48 dB in the others), and with one 0.1 or 0.3 m high within 1.13 dB. A screen on a roof that goes on under the listener, which the solver takes as a strip, is within 0.42 dB at 1.1 m and 0.50 dB at 0.1 and 0.3 m, where the simulation’s paths over its top are up to 3.43 dB off: see the barriers lower than a wavelength below.

In an outdoor scene the early stage takes the solver's sound for the row, in still air and in wind (`simulation.early.rows: solver`, the default). With `--engine` the 45 cases on a ground are simulated as outdoor scenes too, their roofs as buildings and their screens as barriers 400 m long (`second_order`, reflections on both sides of an edge, every sequence of reflections tested, a response of 0.5 s, which ends before what comes back from the far ends of those buildings and barriers): with the solver the simulation's level is the solver's own within 0.02 dB in every band (the air takes its energy along the path pulled taut over the tops in the one and along the straight line in the other). Against the wave calculation (dB), and with `rows: edges`, the paths over the edges alone:

row	cases	`rows: solver`	`rows: edges`
one screen	9	-0.15 to +0.47	-0.87 to +0.54
one roof 10 m wide	9	-0.16 to +0.14	-3.81 to +3.37
a roof 9 m wide and a screen 1 m past it	9	-0.60 to +1.23	-6.22 to +5.41
two roofs 4 m wide, 2 m apart	9	-0.74 to +0.64	-10.63 to +7.85
three screens	9	-3.42 to +0.07	-11.30 to +4.52

The simulation gives the solver the early paths' eight frequencies in each octave band, as the solver's own check does. Against 72 in a band, at nine places behind one screen 3 or 4 m high the band levels with eight are within 0.10 dB at seven, and 0.24 and 0.38 dB off in the 500 Hz band at the other two (10 m behind the screen 4 m high). Near the end of a top the solver's sound gives way to the paths over the edges (see the link above): with a listener 10 m behind a barrier 30 m long, 10 m from the source, moved along it and past its end in steps of 2 m, the level stays within 0.26 dB of the paths' in every band, and is the paths' own from where the way crosses the barrier 5 m before its end. In wind behind a barrier 300 m long the weather changes the level by as much with the solver as with the paths over the edges (within 0.001 dB, 25 and 85 m behind it).

One rigid screen has an exact solution, the direct sound and the Biot-Tolstoy-Medwin integral over the edge, which holds the solver closer to a wall than the wave calculation's cases come. In free space, over a grid of 1670 places of the two ends (the source 5 cm to 30 m before the screen and from 20 m below its top to 5 m above, the listener 0.3 to 40 m past it and from 22 m below to 5 m above), the solver is within 0.36 dB of it in the 63 Hz band, 0.33 dB in the 125 Hz band and 0.24 dB in the 250 and 500 Hz bands, wherever it takes the two (1362 to 1485 of the places, by the band); ends within decimetres of the top and the top's place within the march's cells are looked at closer below. In the tests' seven places with the source or the listener 39 to 86° below the screen's top (a source half a metre before a screen 8 m above it), the march alone is 3 to 11 dB low.

On a rigid ground the same integral gives the four paths over the top, from the source and its image in the ground to the listener and its image, but not the sound that goes on between the top and the ground: the exact level there is that of the rigid strip which the screen makes with its image in the ground (`validation/barrier_strip.py`), and the solver takes that strip up to a height of 32 over the wavenumber. A screen 3 to 20 m high, with the source 0.5 to 30 m before it and the listener 0.5 to 40 m past it, each 0.5 to 4 m above the ground (672 places): where the solver takes the strip it is within 0.18 dB of the exact level in the 63 Hz band (605 places), 0.22 dB in the 125 Hz band (495), 0.24 dB in the 250 Hz band (235) and 0.31 dB in the 500 Hz band (95), where the four paths are up to 0.71, 0.30, 0.31 and 0.23 dB from it; in the other bands of the 563 places that it takes in every band, above that height or with both ends close to a high screen, it is within 0.31 dB of the four paths. In the tests' seven places on a rigid ground the march alone is 1.7 to 28 dB low. Where the first or the last stretch of the path pulled taut over the tops is steeper than 30°, the marches run on a grid of heights eight times as high: deep in the shadow of a high wall the level depends on the height of that grid (behind a wall 20 m high, with both ends 5 m from its foot, 46 dB below the free field, by up to 4.6 dB on the usual grid and 0.72 dB on one four times as high).

A roof's corner has an exact solution too, with the reflection off the roof. Over a grid of 591 places (the source on the roof or up to 3 m above it, 5 cm to 10 m before the roof's far corner; the listener 0.5 to 40 m past the corner, from 8 m below the roof to 2 m above it), the solver is within 0.30 dB of it in the 63 Hz band, 0.25 dB in the 125 Hz band, 0.35 dB in the 250 Hz band and 0.53 dB in the 500 Hz band, wherever it takes the two (494 to 538 of the places, by the band); in the tests' thirteen places the march alone is between 5.8 dB below it and 0.7 dB above. With a barrier 0.5 to 2 m high on a roof that both ends stand over, the source 5 cm to 10 m before the barrier and up to 3 m above the roof, the listener 0.5 to 15 m past it and up to 4 m above the roof, the solver is within 0.50 dB of the exact level of the strip that the barrier makes with its image in the roof in the 63 Hz band, 0.29 dB in the 125 Hz band, 0.38 dB in the 250 Hz band and 0.37 dB in the 500 Hz band, in the 456 to 513 of 630 places it takes, where the four paths over a screen on a rigid ground are up to 2.92, 3.05, 1.88 and 0.95 dB from it and the march alone between 23.7 dB below it and 2.3 dB above. The 0.50 dB is the march's: with a barrier 1.1 m high, the source 3 m before it and 3 m above the roof and the listener half a metre past it and 1.5 m up, the sound comes down to the top at 32° and goes on up at 39°, and the march is 0.50 dB below the open field with Kirchhoff's screen, for which the strip's part takes it.

Ways that cross the row at a slant. Seen from above, a way seldom crosses a row's edges at right angles. The solver takes the angle by which each obstacle is turned from across the way (`slants`), and takes the row in one of two ways. In the first the marches run in the vertical plane along the way as before, and each edge's terms are those of the edge at its slant, on the shortest way from end to end over it. One edge between the ends has its exact solutions at a slant too. One rigid screen is within 0.10 dB of the direct sound and the edge integral at 30, 60 and 75°, at four places of the source and the listener; taken as an edge across the way, as the vertical section along the way shows it, the solver is 0.09 to 0.62, 0.54 to 2.84 and 1.48 to 5.33 dB low there. A roof's corner is within 0.11, 0.36 and 0.61 dB at those slants, the most with an end a metre before the corner (within 0.29 dB otherwise). A barrier 0.1 to 3 m high on a rigid ground, which the solver takes as the strip summed for ends apart along it, is within 0.08 dB of the strip's exact level at 30 to 75° in eight places, where it is 0.21 to 7.61 dB off when not turned.

In a row a march lights the edges, and the march's way over an edge, in the vertical plane along the way, is longer than the sound's own where an end's way to the row is steep: at 60°, from a source level with two roofs and 10 m before them to a listener 4.5 m below them and 5 m past them (both across the edges), by 8 cm, and by 38 cm for the way to the listener's image in the ground. The edges' terms have the phase of the sound's own way and the marched field that of the section, and the two do not add up: given that row all the same, the first way would be up to 6.4 dB above the wave calculation. It does not take a row where the section's phase on those terms would change a band's sound by more than a tenth of it, nor an edge in a row that is turned more than 60° (one edge that both ends light themselves: 75°).

Such a row is taken the second way where it stands on a rigid ground with all its edges in one direction, turned up to 75°: solved across its edges and moved to the slant. A rigid row that is the same all along its edges answers a point source's pulse, between ends y apart along the edges, as it does straight across them at the time sqrt(t² - y²/c²), so that every way between the ends and their images has its own path at the slant. At a frequency that is a sum of the row's sound across its edges over a lattice of wavenumbers, whose weights hold the row's response up to 8 m past the longest of those ways. The sum moves a free field and its reflection off the ground to slants of 3 to 80° within 0.002 of the direct sound. For a row it is as good as the solver is across the row at the lattice's wavenumbers, for the part of the row's response that the weights hold, as long as the response has ended a period of the lattice less the weights' fade past the shortest way (what arrived later would come into the sum out of place); the solver takes a row down to ends half a wavelength apart over the tops. Where the sound crosses the row at a lower wavenumber, or weights of more than a tenth of the first arrival's lie below it, the row is out of reach this way too (`OutOfReach`): one screen with the ends 6.8 m apart across it in the 63 and 125 Hz bands at 75°. One rigid screen moved so is within 0.15 dB of the exact solution at 2 to 75° (the first way within 0.06 dB at those places), and a barrier 0.1 to 8 m high on a rigid ground within 0.15 dB of the strip's exact level, or of the four paths over its top, in eight of ten cases at 30 to 75°, and 0.23 and 0.29 dB above it in the two others (the first way within 0.08 dB in all ten). The solver takes the first way where it can: on the 66 cases below at 30 and 60° the second takes 7.5 times as long (by set and slant 1.6 to 54 times), and where both take a band their levels are within 0.23 dB of each other (0.37 dB over three screens on a rigid ground).

The wave calculation's scenes are the same all along their edges, so each run also gives the sound between ends offset along them; with one screen that is within 0.14 dB of the exact solution at 30, 60 and 75° as well. Its levels for 66 of the cases are stored at those three slants, and the solver takes every band of every one. The table gives its level less the wave calculation's (dB), and its level with the row as the vertical section shows it, not turned:

rows	band levels	30°: the solver	not turned	60°: the solver	not turned	75°: the solver	not turned
free space, up to 6°	80	-0.19 to +0.17	-0.35 to +0.07	-0.54 to +0.63	-1.81 to -0.12	-0.41 to +0.66	-2.60 to -0.50
free space, 22° down to the listener	32	-0.18 to +0.15	-0.30 to -0.06	-0.22 to +0.17	-1.37 to -0.52	-0.19 to +0.37	-2.50 to -0.80
a roof under the source, to its edge	32	-0.15 to +0.03	-0.26 to +0.13	-0.08 to +0.19	-0.82 to +0.85	-0.14 to +0.32	-1.38 to +0.88
a screen 1.1 m high on a roof under both ends	16	-0.02 to +0.31	-0.63 to +0.41	-0.02 to +0.30	-3.41 to +0.89	-0.02 to +0.30	-5.73 to +1.33
the same, the screen 0.1 or 0.3 m high	32	-0.11 to +0.53	-0.29 to +0.91	-0.15 to +0.53	-2.53 to +2.16	-0.19 to +0.21	-4.14 to +3.39
two roofs on a ground	36	-0.75 to +0.85	-1.46 to +0.36	-1.26 to +0.59	-4.79 to +1.90	-1.16 to +1.21	-5.90 to +1.32
three screens on a ground	36	-3.21 to +0.65	-4.15 to +0.07	-5.78 to +0.49	-8.76 to +0.06	-8.18 to +2.39	-9.73 to -1.87

The first way takes the rows in free space at 30 and 60°, and the roof under the source and the screens on a roof at every slant. The second takes the rows on a ground but for the 63 Hz band at 30° at 15 of 18 places (57 of 72 band levels at 30°, all at 60 and 75°), and at 75° the rows in free space (104 of 112 band levels).

In free space and over a roof the solver is as close at 30 and 60° as across the row, but in the 63 Hz band at 60°: over three screens it is 0.44 to 0.54 dB low there and over three roofs 0.60 to 0.63 dB high, and within 0.22 dB in every other case and band. At that slant the row is, in wavelengths across its edges, what it is across the way an octave lower, and across the way in the 31.5 Hz band the solver is 0.30 to 0.33 dB below a wave calculation of the three screens (5 m apart, a third to two thirds of a wavelength there) at the two places where it is 0.44 and 0.47 dB low at the slant. That calculation is not made for the three roofs. At 75°, nearly two octaves lower across the edges, the solver is 0.37 to 0.52 dB high over the three roofs in the 63 Hz band and 0.61 to 0.66 dB high in the 125 Hz band, and within 0.41 dB in every other case and band in free space; with a roof under the source within 0.32 dB, and with a screen on a roof within 0.31 dB.

On the rigid ground, with ends from half a metre below the tops to half a metre above them or 1.5 m above the ground, the two roofs are within 0.85 dB of the wave calculation at 30°, 1.26 dB at 60° and 1.21 dB at 75° (across the row: 0.74 dB). The three screens are up to 3.21 dB low at 30°, 5.78 dB at 60° and 8.18 dB at 75°, and up to 0.65, 0.49 and 2.39 dB high (across the row up to 3.42 dB low); what lacks between rigid screens is described below.

What it does not hold:

- **Rigid screens on a rigid ground.** Over three screens 6 m high and 5 m apart the solver is up to 1.26 dB low near grazing incidence, 2.55 dB with the source 4.5 m below their tops and 3.42 dB with the listener at street level (two roofs with a gap 2 m wide between them near grazing incidence: within 0.50 dB; one screen, one roof and a roof with a screen 1 m past it: within 0.65 dB), and at a slant up to 3.21 dB low at 30°, 5.78 dB at 60° and 8.18 dB at 75°. Between rigid screens on a rigid ground sound goes to and fro, and a march one way has none coming back. That is part of what lacks: with both ends at street level the wave calculation's response across the row holds 22 to 29 % of its energy 9 to 60 m after the shortest way and the solver's 0.1 to 0.2 %, and without that late sound the wave calculation's own levels are up to 1.74 dB lower across the row, 2.14 dB at 30° and 2.62 dB at 60° (at 75° that is not traced). It is not all: in the 63 Hz band across the row the solver is 3.4 dB low there, where the wave calculation's level does not fall without its late sound, and the cause of that is not traced.
- **A row at a slant that neither way takes.** The second way is for a rigid ground and edges in one direction. On a ground with an impedance, or with edges at different slants, a row has the first way alone and is refused where an end's way to it, or its image's in the ground, is steep, or an edge in it that a march lights is turned more than 60° (one that both ends light themselves: 75°). Edges turned more than 75° are refused either way: neither is measured beyond. Edges at different slants are each taken as if the row's other edges were turned as it is, which gives the same sound from either end but is not measured against a reference. The second way does not reach a row that the sound crosses at wavenumbers below the solver's lowest, where the ends are few wavelengths apart across the edges: with a way of 25 m at 75°, 6.8 m across the edges, the 63 and 125 Hz bands. And it takes time: 7.5 times as long as the first way over the 66 cases at 30 and 60°. A barrier's strip at a slant takes more terms where the ends are close to it: with both ends halfway up the barrier it is solved from 0.14 of its height from it across it, from 0.41 along the way at 60° and from 0.83 at 75°, and closer the edges' terms stand in its place. The strip's sum for ends apart along the barrier takes 1.7 to 4.9 times as long as across it up to 45°, 2.5 to 9.9 times at 60° and 5 to 37 times at 75° (eight places, 63 to 500 Hz), the most with the ends 4 and 6 m from a barrier 3 m high: 1.8 to 5.4 s a frequency at 75°.
- **What stands behind an end.** It does nothing: a wall behind a source reflects nothing, and a roof under an end goes on behind it without end. Against the exact solution for the corner, with a source on a roof and a listener 1.5 m above it 20 m on: with the roof's edge 5 cm behind the source the 63 and 125 Hz bands are 2.6 and 2.0 dB too high (3.5 dB at the edge itself), with the edge 20 cm to 1 m behind it up to 1.2 dB too high or 0.9 dB too low, from 1.5 m on within 0.35 dB, 5 m behind it within 0.03 dB.

- **An end the march does not reach.** A march begins from its end's field on the plane of its first obstacle, from the top there up to the upper end of the first Fresnel zone of the way on, and the solver refuses (`OutOfReach`) an end for which that needs more than eight times the march's usual absorbing layer: an end level with its first top and closer to it than 3.7 cm (63 Hz band) to 3.9 cm (500 Hz band); one whose way to its first top, or whose image's in the ground or in the roof under it, rises too steeply, in the 500 Hz band beyond 88.2° from an end a metre before a wall and 87.4° from one 10 cm before it, in the 250 Hz band beyond 89.1° and 88.7°, in the 125 Hz band 89.6° and 89.4°, in the 63 Hz band 89.8° and 89.7°; and one close before an obstacle and high above its top (10 cm before it: more than 2.6 m above in the 500 Hz band, 5.2 m in the 250 Hz band, 10.5 m in the 125 Hz band). On a rigid ground, an end half a metre from the foot of a wall is refused in the 500 Hz band where the wall is 15 m high or more (12 m high: taken), and in the 250 Hz band too where it is 60 m high; 2 m from the foot of a wall 60 m high it is taken in every band. A source a metre above a roof is refused 10 cm before a barrier 3 m high on the roof in the 500 Hz band (30 cm before it: taken) and 5 cm before it in every band. Ends closer to each other over the tops than half a wavelength of the band's lowest frequency (3.6 m in the 63 Hz band, 0.46 m in the 500 Hz band) are refused too: the march's field becomes a point source's by the length of the path, which holds from there on (a tenth of a wavelength apart the level would be up to 0.5 dB off). The solver refuses, with a `ValueError`, an end below the top of a building it stands in or of a wall or a barrier it stands against.
- **A ground that changes before the first obstacle, a roof under both ends with nothing on it between them, and nothing between the ends.** That end's march begins at the end itself, with the parabolic equation's own starting field, which is the end's within some 35° of the horizontal and from about a wavelength on. Beyond that the end is refused (`OutOfReach`): where its way on, or its image's in the ground or the roof, is steeper than 35°, and where its first top, or the other end with no top between them, is closer than a wavelength of the band's lowest frequency (7.3 m in the 63 Hz band, 0.9 m in the 500 Hz band). Within those limits the solver is within 0.49 dB of the paths over a screen behind a change of the ground (at 35° a wavelength away; 0.25 dB up to 20°) and 0.31 dB of the direct sound and the reflection with nothing between the ends (0.35 dB over a roof under both). Without the limits such a march is 1.0 dB low at 50° and 5 dB at 75° behind one screen, and up to 1.0 dB low a quarter of a wavelength before a barrier on a rigid ground. A strip of another ground narrower than 25 cm before the first obstacle can go unseen.
- **An end within decimetres of a top.** Just past a top the field changes over less than a cell of the march's grid of heights, so the march that arrives at an end runs on cells of half the end's distance from the nearest top at most (down to an eighth of the usual cell). With an end 5 cm to 0.8 m before a screen in free space and up to 0.4 m below or above its top, the level is then between 0.49 dB below the exact solution and 0.34 dB above it. In the shadow it is low throughout: in the 63 Hz band by up to 0.48 dB at 5 and 10 cm, 0.40 dB at 20 cm and 0.25 dB at 0.8 m, in the 500 Hz band by up to 0.32, 0.28, 0.22 and 0.13 dB; finer cells still do not change that, and the cause is not traced.
- **The tops' place in the march's cells.** The level moves with a top's place within the march's cells, a tenth of the band's shortest wavelength high, and is lowest with the top in the middle of a cell: by up to 0.08 dB where the sound turns by 40° at the top, and by up to 0.19 dB where it turns by 80 to 115° or an end stands within decimetres of the top.
- **A wall given as a building.** A building's wall is a screen to the march and its two corners shed as a roof's, however narrow the roof: a wall given as a building a millimetre thick comes out between 0.41 dB below and 0.09 dB above the same wall as a barrier (below it, by 0.14 to 0.41 dB, in five of six places in its shadow).

- **Barriers lower than a wavelength on a rigid roof or ground.** The edges' terms take a barrier for a half-plane, lit by the end and by its image in the surface it stands on. The sound that goes on between its top and that surface is not in them, nor in the four paths of the edge integral. With a barrier 0.1, 0.3, 0.5 or 1.1 m high on a roof under both ends (the source 0.5 to 8 m before it, a centimetre, 0.5 and 1.5 m above the roof, twelve places; the listener 5 m past it and 1.5 m up), the four paths are up to 3.20 dB from the exact level of the strip that the barrier makes with its image in the roof (`validation/barrier_strip.py`) at 0.1 m, 2.41 dB at 0.3 m, 2.10 dB at 0.5 m and 1.79 dB at 1.1 m. The solver takes that strip for a barrier alone between the ends on a rigid ground, or on a rigid roof under both, and is within 0.20 dB of the exact level in those places (0.13 dB from 0.5 m on), where the march alone is between 8.74 dB below the wave calculation and 1.34 dB above. It is within 0.59 dB of the wave calculation, whose barrier is a cell of 2 cm thick: the thin barrier's exact level is itself up to 0.58 dB from the wave calculation's (0.20 dB in the 63 Hz band). A barrier a millimetre high, half a metre past a source half a metre above the roof, leaves the level 0.46 dB below that without it in the 63 Hz band and within 0.18 dB in the others, where the edges' terms would leave it 2.15 dB above in the 63 Hz band: the strip sheds less than 0.02 dB there, and the rest is the march's, which begins at the barrier (with the source 3 m before it, 0.12 dB). The edges' terms stand, with what they lack, for a parapet on a roof's edge, for a barrier with anything else between the ends or on a ground with an impedance, for a wall given as a building, above a height of 32 over the wavenumber, where the strip's sound and theirs are within 0.13 dB of each other (nine places), and with the two ends closer to each other than about a quarter of the barrier's height, both within about a twentieth of its height from it at whatever heights, or between the two (about where six times the height over the distance between the ends and the height over the farther end's distance from the barrier come to more than 30 together), where the strip's equation would take more terms than the solver gives it: half a metre above the ground and 39 cm either side of a screen 3 m high they are 0.25 dB from the strip's level in the 63 and 125 Hz bands (0.83 dB at 44.5 Hz alone), up to 0.26 dB in a band at fourteen such places beside screens 3 to 20 m high, at four of which one end is two to three and a half times as far from the screen as the other, and up to 0.27 dB at twenty-three places with the ends at different heights, both 2 to 4.5 % of the screen's height from it. A parapet 0.1 to 1.1 m high on the edge of the roof under the source, from the same twelve places, with the listener 5 m past the edge, 3 m below the roof or 1 m above it, is within 0.73 dB of the wave calculation where it is a quarter of a wavelength high or more and 1.28 dB below that (0.15 dB with no parapet); a parapet a centimetre high on the edge, 8 m from the source, leaves the level up to 0.66 dB above the exact corner's (63 Hz band). With a screen 5 cm high on grass the level is up to 0.50 dB from the open ground's with the edges' terms (a source on the grass; 0.45 dB with it 1.5 m up), and 0.23 dB without them.
- **Steps.** Over a step of 1 m between two roofs the 63 Hz band is 0.38 dB low (over a step of 0.2 m within 0.02 dB there, and up to 0.22 dB low in the 250 Hz band); the cause is not traced. A roof that rises by a centimetre where the roof under the source ends comes out 0.27 dB above the flat roof in the 63 Hz band with the source 8 m before the step and 0.38 dB with it 3 m before.
- Absorbing roofs and walls, which the solver takes as rigid, and the bands above 500 Hz are not in the check.

`tests/test_rows.py` holds, on every change: thirty-one of the cases against their stored wave levels, the thirteen in free space also at least twice as far from them without the edges' terms, and two more, with a barrier 10 cm high on a roof, within 0.25 dB of the strip's exact levels and, with the edges' terms in the strip's place, as far off as those are; one rigid screen in free space and on a rigid ground against the exact solutions as above, with ends far below its top, close beside it and within centimetres of it, and within 0.2 dB for a source and a listener up to 3 m above its top and 2 m below; the higher grid behind a high wall and the finer cells past a top, each against the exact solution with it and without; one, two and three screens level with both ends, which let through a half, a third and a quarter of the free field without a reflection at their tops, within 2 %; a march from a point source's exact field against that field further on, and the direct sound and the reflection off the ground each by its own distance; the same sound with the source and the listener swapped, over a ground that changes along the way and close under walls too; an end over a roof before the roof's corner, down to 10 cm from it, and before a barrier on the roof, down to 20 cm, a barrier on a roof far from both ends and a roof's edge behind an end, against the exact solutions as above; the march over a barrier on a rigid plane against the sound without the barrier plus what Kirchhoff's screen sheds, the strip's sound against the edges' terms where the one hands over to the other, within 0.15 dB, and where the edges' terms stand for a barrier (a parapet on a roof's edge, two barriers, a wall given as a building, a ground that is rigid at some frequencies only, the ends close to each other beside a high screen); what a corner sheds of the wave of an end over its roof and of that end's image against closed forms; the errors for what the solver refuses, and marches from an end at the limits of what it takes; and a level that changes by less than 0.05 dB a centimetre as a barrier's top passes the plane of the roof before it. At a slant: one screen, a roof's corner, a high screen and seven barriers on a rigid ground against the exact solutions as above, each further off when not turned; four rows in free space against their stored wave levels at 60°, two of them at 30° too, each further from them when not turned; the same sound with the ends swapped, with edges at different slants too; the row from roof level to street level at 60° refused by the first way, 6.4 dB off without that refusal, and taken by the second; what the slants have to be; and the geometry of an edge at a slant against the shortest way over it, found by a search. Across the row and moved: the weights against an arrival's place and size at the slant, before the shortest way and past the kernel's end; the weights below the lowest wavenumber solved, and the refusal; a free field and its reflection moved to slants of 3 to 80° with a solver that gives exact arrivals; the wavenumbers and the row that the solver is asked for, and those it refuses; which way the solver takes, and both refusals; one screen and a screen and a barrier on a rigid ground against the exact solutions; rows on a ground against their stored wave levels at 30 and 60°, three screens among them; and the same sound with the ends swapped. `tests/test_strip.py` holds the strip's solution by itself: a barrier far lower than a wavelength against the plate across a uniform flow that it is, the kernel's parts against each other and their limits, the rule for the logarithm against closed forms, the equation at the far end of the sum over the wavenumbers, where a factor of its kernel would grow beyond the digits of a number and is faded, how far the density's number of terms holds what the barrier adds (to 6e-5 of it, 4e-5 dB, at a height of 31 over the wavenumber with one end at the barrier's foot and the other far off, the least of the places tried), the sums against the field of an end millimetres from the barrier, the sum over the wavenumbers along the barrier against a point source's field, the same sound with the ends swapped, the sound shed against a reference written apart (`validation/barrier_strip.py`: every term of the density, the kernel as it stands, the sum over the real wavenumbers), and, for ends apart along the barrier, the sum against a point source's field, the sound shed against that reference at slants of 15 to 75°, the sum's points behind a barrier 31 over the wavenumber high against three times as many for waves that cross the strip twice as often, a slant that tends to none against the sound across the barrier, and where the strip's equation is not solved at a slant.

Levels over a long term

`validation/long_term.py` checks how the engine combines weather classes and the periods of the day (see Levels over a long term). The scene is a source 1 m over grass (Miki's impedance, 200 kPa s/m²) with the same sound power in every octave band, and listeners 1.5 m up from 50 to 800 m away along the wind, in four weather classes with shares of the day, the evening and the night as a weather statistic might give them:

class	weather	day	evening	night
downwind	`windy` (5 m/s 10 m up, towards the listeners)	30 %	25 %	15 %
upwind	`windy`, towards the source	30 %	25 %	15 %
still	`neutral` (still air)	40 %	30 %	20 %
inversion	`clear_night` (air warming with height)	0 %	20 %	50 %

The downwind class is the scene's own weather, so the four classes take four runs. The level in each class, each period (12, 4 and 8 hours), over the long term and L_den (5 dB in the evening, 10 at night), in dB(A):

	downwind	upwind	still	inversion	day	evening	night	long term	L_den
50 m	47.1	36.6	47.6	47.7	46.0	46.3	46.9	46.3	53.1
100 m	41.8	21.6	39.1	41.0	38.9	39.4	40.1	39.4	46.3
200 m	34.5	10.5	26.5	35.8	30.1	32.0	33.8	32.0	39.7
400 m	24.1	-2.8	11.8	29.5	19.2	23.9	26.8	23.9	32.5
800 m	12.9	-20.2	-2.1	20.0	7.9	14.0	17.2	13.9	22.8

The arithmetic. The script works the period levels, the long-term levels and L_den out again from each class's own run: the energy mean over the classes by their shares (ISO 1996-2, CNOSSOS-EU), over the periods by their hours, and L_den by Directive 2002/49/EC. The engine agrees with it to rounding (0.0 dB at most).

The meteorological correction. ISO 9613-2 predicts the level in conditions favourable to propagation, downwind or under a moderate ground-based inversion such as clear nights bring, and takes the long-term level from it as L_LT = L_DW - C_met, with C_met = C_0 (1 - 10 (h_s + h_r) / d_p) beyond d_p = 10 (h_s + h_r) and 0 nearer, where C_0 lies between 0 and 5 dB by the local weather statistics. The engine's C_met is the A-weighted level in the favourable classes (downwind and the inversion, combined by their shares of the whole day) less the long-term level. With C_0 fitted to it at 3.36 dB, within the standard's range:

distance	engine	ISO 9613-2	difference
50 m	1.00	1.68	-0.68
100 m	2.01	2.52	-0.51
200 m	3.14	2.94	+0.20
400 m	3.46	3.15	+0.31
800 m	3.51	3.25	+0.26

The engine's correction follows the standard's shape within 0.68 dB, the most at 50 m, where the standard's formula is only a step past d_p = 25 m and the ground's own pattern still differs from one weather to the next. This is a check of plausibility, not of accuracy: the standard's correction is an empirical fit, and the engine's comes from each class's physics, which the sections above check against a fast field program and parabolic equations.

Moving sources against the exact field

`validation/doppler.py` checks walks with Doppler (see Moving sources and Doppler). A source passes a listener 10 m from its line, from 60 m before it to 60 m past, 1 m up with the listener 1.5 m up, over a rigid ground, at 30 m/s (a car) and 80 m/s (a train), and a listener passes a source the same way at 30 m/s; the places are 1 m apart. Here the field is known exactly: the direct sound and the image source's, each heard at its retarded time (for a moving source the time t_e it left at, t = t_e + R(t_e) / c; for a moving listener t - R(t) / c), over the distance then, with the air's absorption, for a source that keeps its sound as it moves (quasi-static, as the engine's). Pure tones through the engine are compared with it sample by sample (the waveform's error, the energy of the difference relative to the field's) and in 20 ms windows (the level's error, mean and most), away from the walk's ends:

case	tone	waveform	level	without Doppler: waveform	level
car, 30 m/s	250 Hz	-71.3 dB	0.000 / 0.000 dB	2.9 dB	0.73 / 3.42 dB
	1 kHz	-58.0 dB	0.000 / 0.012 dB	2.9 dB	0.76 / 3.54 dB
	4 kHz	-46.0 dB	0.001 / 0.019 dB	3.0 dB	0.96 / 4.13 dB
train, 80 m/s	250 Hz	-68.7 dB	0.002 / 0.009 dB	2.9 dB	1.62 / 2.75 dB
	1 kHz	-56.6 dB	0.006 / 0.034 dB	2.9 dB	1.63 / 3.54 dB
	4 kHz	-39.5 dB	0.077 / 0.419 dB	3.1 dB	2.39 / 12.21 dB
passing listener, 30 m/s	250 Hz	-71.4 dB	0.000 / 0.000 dB	-0.5 dB	0.22 / 2.87 dB
	1 kHz	-58.1 dB	0.000 / 0.001 dB	1.7 dB	0.24 / 3.45 dB
	4 kHz	-46.1 dB	0.000 / 0.005 dB	2.9 dB	0.35 / 3.78 dB

With Doppler the sound follows the field to within 39 dB and its level to within 0.42 dB, both the most for the train at 4 kHz, and there mostly the air's doing, not the delays': ahead of the train its tone is heard at 5.2 kHz, which the engine's band filters share between the 4 and the 8 kHz band, each with its own absorption in the air, where the field here takes the 4 kHz band's all along. The level differs most where the comparison starts, the train 47 m away. With the field's absorption taken at the pitch heard instead, through the same band filters (a one-off check, not repeated by the script), the train at 4 kHz is within 44.9 dB and its level within 0.025 dB. The same walks crossfaded from place to place, as without Doppler, keep each place's pitch, so their phase jumps from place to place: the waveform's error is about as large as the field itself (-0.5 dB or more). Their crossfades keep each band's power however alike the places sound, so at 30 m/s the level misses by 0.2 to 1.0 dB on average and by 2.9 to 4.1 dB at most. The train, its places 12.5 ms apart and so always crossfading, misses by 1.6 to 2.4 dB on average and 2.7 to 3.5 dB at most at 250 Hz and 1 kHz; at 4 kHz by 12 dB 0.72 s in, where the field's two paths cancel for a moment (10 dB below its median level) and the places' do not.

The sound sent each way. The same walks with the source sending a tone each way (an `Emission`, `acoustic_spaces.emission`): 1 + 0.8 cos θ in amplitude, θ from straight ahead, and turned in phase by 90° times the sine of the elevation, on a grid 2° apart. In the exact field each path carries what is sent the way it leaves the source, at the time it leaves:

case	tone	waveform	level
car, 30 m/s	1 kHz	-57.7 dB	0.004 / 0.025 dB
	4 kHz	-47.1 dB	0.005 / 0.060 dB
train, 80 m/s	1 kHz	-56.1 dB	0.010 / 0.038 dB
	4 kHz	-35.7 dB	0.080 / 0.423 dB
passing listener, 30 m/s	1 kHz	-57.5 dB	0.004 / 0.032 dB
	4 kHz	-46.8 dB	0.005 / 0.086 dB

With a cardioid directivity on the source, which the emission takes the place of in the early paths, every number is the same to the digits shown. The level stays within 0.09 dB at 30 m/s and 0.42 dB for the train at 4 kHz, as one signal's does there; the train's waveform at 4 kHz is 3.8 dB further than one signal's, as the tone sent ahead weighs the approach more, where the air's absorption at the pitch heard differs most from the field's (with the field's taken at the pitch heard, as above, it is within 46.6 dB and its level within 0.012 dB). In one-off checks, not repeated by the script, the tone sent louder behind instead (1 - 0.8 cos θ) came within 44.9 dB, and with the amplitude alike every way (only the phase turning) within 39.5 dB, as one signal; a grid 1° apart changed the table by 0.4 dB at most, and places 0.5 m apart brought the cases at 30 m/s within 63 dB and the train at 1 kHz within 60 dB, the train at 4 kHz staying within 36 dB.

The places. The car at 1 kHz with its places further apart:

places apart	waveform	level
0.25 m	-93.8 dB	0.000 / 0.000 dB
1 m	-58.0 dB	0.000 / 0.012 dB
2 m	-38.7 dB	0.002 / 0.033 dB
4 m	-18.3 dB	0.015 / 0.235 dB

The delays are curves through the places, so the further apart the places, the more they stray between them; up to 1 m they hardly do.

A helicopter's places. A source flying over a listener 150 m up at 63 m/s, from 300 m before it to 300 m past, over the rigid ground, the listener 1.2 m up right under its track, as in a helicopter's flyover, with its places further apart (the level's windows from 0.3 s after the first sound arrives):

places apart	tone	waveform	level
1 m	250 Hz	-110.8 dB	0.000 / 0.000 dB
	1 kHz	-102.3 dB	0.000 / 0.001 dB
	4 kHz	-88.5 dB	0.000 / 0.002 dB
4 m	250 Hz	-81.6 dB	0.001 / 0.015 dB
	1 kHz	-68.5 dB	0.002 / 0.047 dB
	4 kHz	-55.3 dB	0.004 / 0.088 dB
8 m	250 Hz	-64.0 dB	0.003 / 0.094 dB
	1 kHz	-50.4 dB	0.006 / 0.202 dB
	4 kHz	-37.2 dB	0.018 / 0.308 dB
16 m	250 Hz	-45.9 dB	0.013 / 0.819 dB
	1 kHz	-32.2 dB	0.025 / 0.490 dB
	4 kHz	-19.4 dB	0.072 / 0.994 dB

From 150 m up the delays curve far less than from a road 10 m away, so places 4 m apart follow the field to within 55 dB at 4 kHz, more closely than the car's places 1 m apart (46.0 dB at 4 kHz); the flyover demo simulates its helicopter every 4 m.

Passing close. The road's walks again with the walker passing 2 m or 1 m from the other's line (2.06 and 1.12 m away at the nearest, with their heights), its places 0.25 m apart, heard in mono; at 1 kHz through a cardioid microphone aimed level at the walker's line; and at the ears of a listener facing along that line, whom the walker passes on one side (the spherical head; the worse ear's waveform, the level's error over both). At 70 m/s the walker goes 1.4 m in 20 ms, so here the level's error is in 5 ms windows:

walker	from the line	tone	waveform	level
source, 70 m/s	2 m	1 kHz	-67.8 dB	0.002 / 0.010 dB
		4 kHz	-52.9 dB	0.022 / 0.128 dB
		1 kHz, cardioid	-65.2 dB	0.002 / 0.010 dB
		1 kHz, ears	-55.5 dB	0.002 / 0.058 dB
		4 kHz, ears	-47.6 dB	0.023 / 0.129 dB
source, 70 m/s	1 m	1 kHz	-56.1 dB	0.002 / 0.010 dB
		4 kHz	-44.2 dB	0.023 / 0.130 dB
		1 kHz, cardioid	-53.3 dB	0.002 / 0.023 dB
		1 kHz, ears	-48.9 dB	0.002 / 0.020 dB
		4 kHz, ears	-41.1 dB	0.024 / 0.130 dB
source, 30 m/s	1 m	1 kHz	-56.6 dB	0.000 / 0.011 dB
		4 kHz	-45.1 dB	0.001 / 0.034 dB
		1 kHz, cardioid	-53.6 dB	0.000 / 0.032 dB
		1 kHz, ears	-48.4 dB	0.001 / 0.033 dB
		4 kHz, ears	-41.8 dB	0.001 / 0.105 dB
listener, 30 m/s	1 m	1 kHz	-56.8 dB	0.000 / 0.008 dB
		4 kHz	-45.3 dB	0.001 / 0.026 dB
		1 kHz, cardioid	-53.8 dB	0.000 / 0.029 dB
		1 kHz, ears	-50.1 dB	0.000 / 0.038 dB
		4 kHz, ears	-41.4 dB	0.001 / 0.045 dB

A path’s spreading is not in its filter: each sample is divided by the path’s delay then, so the level follows one over the distance through the pass. In mono at 70 m/s the most is not at the pass but where the comparison starts, 33 m before it, and is the air’s again: the 4 kHz tone is heard at 5.0 kHz there. With the field’s absorption taken at the pitch heard (a one-off check), the level at 70 m/s is within 0.028 dB at 4 kHz and 0.009 dB at 1 kHz, the most at the pass.

What depends on the direction a path arrives from (a microphone’s pattern, the ears, a loudspeaker’s panning, the source’s directivity) stays in the filter, which is known at the places and goes in a straight line from each to the next, taken when each place’s sound is heard however many of them go by in 20 ms. Through the cardioid microphone (each path times 0.5 + 0.5 cos of the angle it arrives at) the level stays within 0.032 dB of the field’s in 5 ms windows. A straight line between two places 0.25 m apart strays from the pattern by 0.026 dB in the direct sound 1.12 m from the capsule (by hand: an eighth of the step squared times the pattern’s curvature over its value).

At the ears a walking source's field is exact too, for ears whose responses depend on the direction alone, as the engine's do: the head stands still, so each sample of the pressure at its centre goes through the ears' responses to the direction it arrives from, and the responses add up where they land. The engine keeps where each ear's response starts as a delay and takes what passed the head's centre that much earlier (see Moving sources and Doppler). The level stays within 0.06 dB at 1 kHz, the most 1.9 m after the pass at 70 m/s, 2 m away, where the ground's reflection takes the field 15 dB down. At 4 kHz it is within 0.13 dB: at 70 m/s the most is where the comparison starts, as in mono (0.13 dB there too), at 30 m/s (0.105 dB) the far ear's 0.3 m before the pass, the field 12 dB down. With the places 0.125 m apart the 0.06 dB at 1 kHz and this 0.105 dB are 0.015 and 0.034 dB (a one-off check, not repeated by the script), so they come mostly from the straight line between the places. A walking listener's head moves through the field, which has no exact solution of this kind: its rows compare the engine with its own model worked out sample by sample, each ear a point its onset further along the path at the moment it hears, and what arrives there through the rest of the ear's response.

Formats of many channels turn faster with the direction than a cardioid does. In a one-off check, not repeated by the script (a source passing 1 m from the listener at 70 m/s, the direct sound alone, 1 kHz, against the field through the format's gains for the direction at each instant), third-order Ambisonics differed from the field by 1.3 % of it over the pass with the places 0.25 m apart (14° apart at the pass, as the listener sees them), by 0.33 % with 0.125 m and 0.08 % with 0.0625 m; panning to 5.0 by 1.5 %, 0.26 % and 0.14 %.

The levels. White noise through the car's walk over grass (Miki's impedance, 200 kPa s/m², scattering 0.1, which gives the walk a late part over the open ground), 16 noises up to 18 kHz: the energy of the sound in each octave band over the pass-by, relative to the source's, against the noise through each place's own impulse response for the time its sound is heard (Doppler moves the sound in time and pitch but keeps its energy), and against the sound exposure level `pass_by` gives from the places' band levels:

band	rendered	responses	pass-by	early part alone: rendered	responses
63 Hz	-25.89	-25.91	-26.13	-25.90	-25.92
125 Hz	-27.53	-27.54	-27.69	-27.62	-27.64
250 Hz	-32.11	-32.07	-32.29	-32.31	-32.31
500 Hz	-34.87	-34.92	-34.04	-34.97	-34.91
1 kHz	-29.52	-29.53	-29.63	-29.56	-29.59
2 kHz	-30.18	-30.19	-30.28	-30.21	-30.23
4 kHz	-30.75	-30.77	-30.68	-30.78	-30.79
8 kHz	-31.81	-31.81	-31.85	-31.82	-31.82

The sound keeps the energy of the responses to within 0.07 dB in every band, with its late part and without. The late part is crossfaded by weights that add up to one, as the car sends its sound, drawn from one random sequence for every place; the responses here are drawn from the same one, as otherwise their late parts' chance correlation with the early sound would differ by up to 0.25 dB in a band (a one-off check). The pass-by's band levels are within 0.24 dB of the sound, 0.8 dB in the 500 Hz band, where the ground's dip falls: the pass-by takes the places' band levels, averaged over each band, where the sound is read through band filters, whose skirts fill a dip (a cause not checked separately). For a car at 30 m/s over this grass with 94 dB of sound power, the listener has L_Amax 61.4 dB(A) 2.03 s into the walk, SEL 61.2 dB(A) and L_Aeq 55.2 dB(A) over 4 s.

What it leaves out. The engine's source is quasi-static, as the field it is compared with: a moving source's convective amplification is left out (its size depends on the kind of source: a point monopole's far-field pressure goes as $(1 - M \cos\theta)^{-2}$, $M \cos\theta$ its Mach number towards the listener as the sound leaves, so its level as $40 \lg 1/(1 - M \cos\theta)$, up to 1.6 dB louder straight ahead of it at 30 m/s and 4.0 dB at 70 m/s, 1.5 and 3.2 dB quieter straight behind), and so is the Doppler shift of the late part, which is crossfaded. At the ears what is left out is the amplification at the head's centre: each ear keeps its own Doppler factor over the centre's, as the exact field above has it (up to 0.15 dB at an ear at 70 m/s, 1 m away, a one-off check). The ears' responses depend on the direction a sound arrives from, not on how far its source is: one passing 1 m away is heard through the responses of a far one. A walking source is checked against the exact field above; a walking listener's moving head has none, and is checked against the engine's model worked out sample by sample, as above; and both against the convolution with a response when the walker stands still (to within 45 dB at each ear in the tests). A test drives a source past a listener whose ears' responses start 4 samples apart from one direction to the next, as a measured set's may, and holds its treble (6 to 16 kHz, frame by frame) within 0.3 dB of the same walk with ears that start together.

Helicopter rotors against theory and a certified level

`validation/rotorcraft.py` checks the sound a helicopter's rotors send (see Helicopters): its tones against Gutin's formula and against the far field summed over the blades in two other ways, the store that keeps them, the airfoil self-noise against the sample outputs of its authors' code, the blades' pitch against its closed form, Pegg's broadband noise against his worked example, and the whole, with either broadband noise, against the certified noise level of the helicopter it describes, the AS350 B3, flying at 63 m/s (25 °C).

The tones. Each harmonic of the blade passing frequency, against:

- Gutin's (1936) formula for a hovering rotor whose thrust and torque act on one radius, in 18 directions from the axis;
- the far field's integral over a turn, summed numerically with 4,096 instants a turn over the blades' points where they truly are, each a moving force (the lift and the drag) or a source (the thickness) on its straight chord, in 24 directions;
- Farassat's formulation 1A in time, the same points each heard at the time it left (solved by Newton's method) at an observer 20,000 km away in eight directions, its harmonics taken from 1,024 instants a turn. This check shares nothing with the closed form but the blade's loads.

check	rotor	harmonics	most off
Gutin	3 blades, 41.3 rad/s	6	3×10^{-15}
the integral round the turn	main	12	6.5×10^{-7}
	tail	12	1.7×10^{-4}
in time (Farassat 1A)	main	12	7.5×10^{-6}
	tail	12	1.6×10^{-5}

“Most off” is the largest difference of a harmonic’s complex amplitude, over that harmonic’s loudest in any direction (over the loudest harmonic in the direction, in time). The tail rotor’s 1.7 x 10⁻⁴ comes from the chord’s points’ radius, taken to second order: its chord is a fifth of its radius. In time the observer stands so far away that the blade’s radius squared over twice the distance, which the far field leaves out of the phase, does not count: in a one-off check at 200 km it put the main rotor 7.5 x 10⁻⁴ off, at 2,000 km 7.4 x 10⁻⁵, falling as one over the distance.

The store. A rotor’s tones are kept as one period of the sound in each direction on a grid (`Periodic`); what it sends over 4,000 samples 8.3 s in, each sample a random direction, against the harmonics summed there:

rotor	grid	harmonics	rms error	size	made in
main	5°	33	-44 dB	11 MB	2.6 s
	2.5°	33	-56 dB	44 MB	8.4 s
tail	5°	31	-48 dB	11 MB	1.8 s
	2.5°	31	-59 dB	42 MB	5.6 s

The harmonics kept reach 90 dB under the loudest (`QUIET_DB`). The default grid is 5° apart.

Pegg’s example. Pegg’s example I (NASA TM-80200, 1979), a hovering transport helicopter heard 61.6 m from its main rotor, 5° under its disc, and 62.2 m from its tail rotor, 10° off its axis: the broadband levels come out at 84.50 + S and 82.49 + S dB (his 84.5 and 82.5), and the peak frequencies at 289 and 545 Hz (his 289 and 544).

Airfoil self-noise. Brooks, Pope and Marcolini’s report (NASA RP-1218, 1989) gives two sample outputs of its code: one untripped segment of a 30.48 cm chord at 1.516° (table D2), and ten tripped segments of 15.24 cm at 5.4° with a rounded tip (D3), both in a 71.3 m/s flow heard 1.22 m away. Each mechanism’s band levels from 100 Hz to 40 kHz, against the tables:

table	mechanism	most off
D2	turbulent boundary layer: pressure side, suction side, angle of attack	0.006, 0.006, 0.008 dB
	laminar boundary layer’s vortex shedding	0.003 dB
	total	0.006 dB
D3	turbulent boundary layer: pressure side, suction side, angle of attack	0.006, 0.006, 0.006 dB
	tip vortex at 7.7°	0.188 dB
	tip vortex at 7.668°	0.006 dB

The tables are rounded to 0.001 dB. The tip vortex column is the code’s equations at a tip angle of 7.668°, not at the 7.7° of the table’s input; the code as the report prints it gives 7.7°’s levels, 0.19 dB over the table at 100 Hz and 0.04 dB under it at 40 kHz, so the table may come from another version of it. Neither table has the trailing edge’s bluntness or a stalled section. In one-off checks for the AS350 B3 at 63 m/s, straight down, the bluntness adds under 0.05 dB (its 0.5 mm trailing edges send theirs around 20 kHz) and the 1.5 % of the places round the turn where a section stalls nothing to 0.001 dB; the turbulent boundary layers give 106.8 of the main rotor’s 107.5 dB 1 m away, and the laminar boundary layer’s vortex shedding 3 dB of the tail rotor’s 89.4. The directivity’s angles are those of the report’s figure B3: theta from the chord to the observer, phi from the span to the observer’s projection across the chord.

The blades' pitch. The collective and cyclic pitch that carry the trim's thrust with no flapping, for the main rotor's rigid blades taken from the hub to the tip (no root cut-out and no tip loss, so that it has a closed form), twisted by -10°, in an even inflow, against Johnson's closed form (*Helicopter Theory*, 5.6):

speed	theta_0	theta_1s	most off
0 m/s	15.5675°	0°	9 x 10^{-12} °
30 m/s	13.2553°	-1.7615°	7 x 10^{-12} °
63 m/s	14.5351°	-3.9524°	1.5 x 10^{-11} °

`tests/test_rotorcraft.py` checks the blade sections' sum (`bpm_bands`) against one done place by place, each section's frame and motion through the air taken from the disc's own vectors, each mechanism's level with its own directivity and its bands moved one by one, hovering and at 63 m/s, to 10^{-9}. In one-off checks against 2,304 places round the turn, the default 144 hold the main rotor's level, on the 5° grid, to 0.01 dB everywhere below the horizon and to 0.7 dB above it, close to its disc's plane. The tail rotor's disc is upright, so listeners below the helicopter are close to its plane: there its level holds to 1.3 dB (0.6 dB with 288 places, 0.3 dB with 576), because BPM's directivity changes abruptly where a section's chord points at the listener, an angle where the report itself finds it inaccurate. The tail rotor's broadband noise of this kind is 36 dB under the helicopter's sound in the certification's overflight (below). 48 sections instead of 24 changed the levels by under 0.06 dB.

The certified level. The AS350 B3 at its maximum mass of 2,250 kg flying the overflight of ICAO Annex 16, chapter 11 (14 CFR part 36, appendix J): level, 150 m over a microphone 1.2 m above grass, at 0.9 V_H, in still air at 25 °C and 70 % humidity at sea level. The main rotor turns at 394 rpm, the most in powered flight of the AS350 B3 with the Arriel 2B engine (its type certificate data sheet, EASA.R.008, 9.1): the rotor speed appendix J flies (J36.3(c), J36.105(c)(2)) for that variant's noise record, D73, which certifies 84.6 dB(A) at 2,250 kg. The type certificate data sheet for noise (EASA.R.008) gives no airspeed; V_H is taken as Airbus's fast cruise speed, 136 kt, so 63.0 m/s. The direct sound and the ground's reflection (Weyl and van der Pol's, with Miki's impedance of 200 kPa s/m²), each heard at the time it left the helicopter with the air's absorption (ISO 9613-1), A-weighted; the slow time weighting finds the points 10 dB under the most, and the sound exposure level is the energy between them:

part	SEL	L_ASmax	10 dB down for
all	88.0 dB(A)	81.7 dB(A)	9.4 s
main rotor's tones	63.3 dB(A)	55.6 dB(A)	8.9 s
tail rotor's tones	79.8 dB(A)	72.4 dB(A)	10.8 s
main rotor's broadband noise	87.3 dB(A)	81.5 dB(A)	7.9 s
tail rotor's broadband noise	64.9 dB(A)	57.8 dB(A)	11.0 s
certified	84.6 dB(A)		

That is with Pegg's broadband noise, the default. With each blade section's own (`broadband="bpm"`):

part	SEL	L_ASmax	10 dB down for
all	80.1 dB(A)	72.4 dB(A)	12.0 s
main rotor's broadband noise	68.6 dB(A)	62.2 dB(A)	8.4 s
tail rotor's broadband noise	44.3 dB(A)	37.4 dB(A)	10.2 s

With Pegg's noise the model is 3.4 dB louder than the certified level, nearly all of it Pegg's broadband noise of the main rotor; with the blade sections' own it is 4.5 dB quieter, nearly all of it the tail rotor's tones, the main rotor's broadband noise 19 dB under Pegg's. Self-noise is the noise of a blade in a smooth flow: it leaves out the blades cutting through the wake of those ahead and through turbulence, which a full-size rotor in flight makes and which Pegg's fit to measured rotors holds. Pegg's method is a correlation of the 1970s' measurements: in his own example the band of the broadband peak (315 Hz) comes out 5.3 dB over the level measured there, though a tone of the tail rotor in that band is over it alone; the bands where only the broadband noise counts have no measured level in the report. The tones leave out the loads that change round the turn, which make a rotor's lower harmonics in forward flight louder than steady loads do. Some of the AS350 B3's inputs are assumed (the main blades' chord and twist, the tail rotor's chord and thickness, V_H); `AS350_B3`'s source code says which.

Sirens and road vehicles against their rules

`validation/urban_sources.py` holds the pass-by demos' police car's siren, the fire engine's siren and air horn, the car's horn and the train's horn (`acoustic_spaces.demos.siren`, `acoustic_spaces.demos.horn`, see Vehicles passing by) against what the rules for sirens and horns ask, with the siren's directivity table in the engine, and compares the car's sound power by the two versions of CNOSSOS-EU's table. The siren's level and directivity are those the National Bureau of Standards measured on a commercial 100 W electronic siren (SP 480-28, 1978): 118 dB(A) 3 m away on its axis, its yelp averaged over 20 s, 112 dB(A) 50 degrees off it and 107 at 90.

California's rule (California Code of Regulations, title 13, section 1028): the least A-weighted levels of its two classes of siren 3.0 m away from 0 to 50 degrees off the axis (table I), the wail and the yelp as the engine gives them there over a ground that takes all the sound:

degrees off the axis	0	10	20	30	40	50
class A, least (dB(A))	120	119	118	117	115	113
class B, least (dB(A))	115	114	113	112	110	108
wail and yelp (dB(A))	118.0	116.8	115.6	114.4	113.2	112.0

The siren is above class B at every angle and 1 to 2 dB under class A: a class B siren by the rule. The rule takes the average of five peaks of the sweep, NBS the energy over 20 s, so the rule's reading of the same siren would be at least as high. Its level on the axis is highest in the 1 kHz octave band, as the rule asks (1 or 2 kHz), and the wail's level in 125 ms falls 2.2 dB(A) within its cycle, where the rule allows 10 dB(A) under the table. That fall comes from the A-weighting of a sweep from 500 to 1500 Hz through a loudspeaker taken to be flat: the loudspeaker's response is not in NBS's report.

The Swiss rule for priority vehicles' two-tone horns (SR 741.41, annex 11, the values of DIN 14610): the two tones, 440 and 586.7 Hz, lie within the base frequencies of 360 to 630 Hz, a fourth apart (the rule: 3:4, -3 to +7 %), with a cycle of 3 s (2.5 to 3.5 s) and no pause. On the axis 2 m away in a free field they give 119.4 dB(A), where the rule asks 116 to 129 dB(A) on the test bench; 7.00 m in front of the police car's siren, taken at the car's front, 1.5 m up, over the street's asphalt, the engine gives 108.1, 108.7 and 110.8 dB(A) 0.5, 1.0 and 1.5 m up, where the rule asks 100 to 115 dB(A) on the vehicle. The level of the two tones follows from the yelp's sound power, the same in all, as the loudspeaker is taken to send each pattern with one drive: 128.6 dB(A) against the yelp's 130.6.

The fire engine. Its siren plays the same two tones 6.0 dB louder, 3.0 m up on its cab's roof, which the demo takes as loud as the Swiss rule lets two tones be on a vehicle. The engine gives them 112.3 to 114.9 dB(A) 7.00 m in front of the siren, which the demo takes to be at the vehicle's front, over the street's asphalt, every 0.1 m from 0.5 to 1.5 m up, the loudest 0.8 m up (the rule: 100 to 115 dB(A) 7.00 m in front of the vehicle; a siren set 0.5 to 2 m back on the roof gives 0.4 to 1.4 dB less there), and 125.4 dB(A) 2 m away on the axis in a free field (the bench: 116 to 129). Its air horn, 1.2 m up, is held against the rule for vehicles' horns (Council Directive 70/388/EEC, Annex I): on the rule's bench, 2 m in front of it on its axis in a free field, a blast gives 118.0 dB(A), the most the rule allows (105 to 118, point 1.2.1.6); between 1800 and 3550 Hz it gives 110.2 dB(A) there, where the rule asks 105 at least, and its loudest component in that band is 6.0 dB over the loudest above 3550 Hz, which the rule asks it to be over (point 1.2.3.1), from the spectrum of the made blast; each blast is at its level 0.05 s after it starts (the rule: 0.2 s at most, point 1.2.1.7); 7 m in front of the vehicle over asphalt, 0.5 to 1.5 m up, it gives 107.9 to 108.6 dB(A), where the rule asks 93 at least (point 2.1.4). The horn's pitch, its spectrum, its blasts, its height and its directivity are assumptions, as no measured air horn of a fire engine was found; the rule bounds them only as above.

The car's horn. The street car's pair of disc horns (`--car car_and_horn`), at 360 and 430 Hz, 0.5 m up, against the same rule. On its bench, 2 m in front of it on its axis in a free field, a blast gives 109.9 dB(A) (the rule: 105 to 118; Hella gives its S77 110 +/- 3, the air over the 2 m taking the last 0.1 dB of the treble); between 1800 and 3550 Hz it gives 107.6 dB(A) there (the rule: 105 at least), its loudest component in that band 1.9 dB over the loudest above 3550 Hz (the rule: over); each blast is at its level 0.02 s after it starts (the rule: 0.2 s at most); 7 m in front of the car over asphalt, 0.5 to 1.5 m up, it gives 100.4 to 101.3 dB(A), the most 0.5 m up (the rule: 93 at least). Its spectrum is a disc horn's, its harmonics struck alike and shaped by its tone disc's resonance at 3 kHz, where a single-disc horn's main components are (US patent 8,689,723), with a Q of 2 (assumed); made as the air horn is, a sawtooth at each fundamental, it would give only 102.8 dB(A) between 1800 and 3550 Hz, under the rule, which a horn sold as approved meets. Its blasts, its height and its directivity are assumed.

The train's horn. The railway train's warning horn (`--horn blasts`) sounds the two notes of the EU's rule for locomotives and passenger trains (TSI LOC&PAS, Regulation (EU) No 1302/2014, point 4.2.7.2.1), 660 and 370 Hz (the rule: 660 +/- 30 and 370 +/- 20), one after the other. EN 15153-2:2020, which the rule names for the level, asks each note sounded alone for 101 to 109 dB(C) (L_pCeq) on the vehicle, 25 m ahead of the train's front on its centre line over an open, flat site. Each note is made to 109 there with the horn 2.2 m and the microphone 1.5 m over the railway example's ballast (assumed: the sample of the standard seen leaves out where the microphone stands, and M. Toward et al., "Sound propagation and audibility of train horns", ISVR and RSSB, measured train horns 1.5 m up), each harmonic's reflection off the ballast added in pressure and taken out of its sound power. The engine, which averages a reflection's interference over each band, gives the high note 108.8 dB(C) there and the low one 109.0, and 109.6 and 109.1 in a free field. Sounded together as a chord (`--horn chord`), which EN 15153-2 allows, the two notes are made to 109 dB(C) together, its level for a group of horns; the engine gives 108.9 dB(C) there and 109.3 in a free field. Toward et al. measured a train horn of 368 Hz over ballast with the horn from 0.5 to 3.0 m up and found virtually no difference at 25 m; the low note's A-weighted level there changes by +1.5 to 0 dB over those heights by its harmonics and by +0.1 to -0.5 dB in the engine. A-weighted, the low note is 2.4 dB under its C-weighted level harmonic by harmonic (Toward et al. take 2 dB off the rules' C-weighted levels for their A-weighted ones) and 1.4 dB under it by octave bands, as the engine weights it; the high note 0.5 and 1.0 dB. Its blasts, its height on the train and its directivity, the measured siren's, are assumptions: Toward et al. measured a train horn 1 dB down 30 degrees off its axis, where the siren is 3.6 dB down, so beside the track the horn may be a few decibels louder than here.

The car's table. CNOSSOS-EU's Table F-1 as amended in 2021, which the road vehicles take (`acoustic_spaces.demos.road`), against Directive (EU) 2015/996's, which the car had before acoustic-spaces 4.1.0, at 20 °C:

speed (km/h)	30	50	70	90	110	130
2021: sound power (dB(A))	92.5	98.4	103.0	106.6	109.6	112.1
2015/996 (dB(A))	90.4	96.0	100.4	104.0	106.9	109.4
more by 2021's	+2.1	+2.5	+2.6	+2.7	+2.7	+2.7

The amendment has the vehicle's point radiate alike every way, where 2015/996 had it radiate into the half space above the road, and raises the coefficients A. The table is used as published, in the two copies of it checked against each other (the consolidated directive and Germany's BUB-D 2021): no measured pass-by is compared with it here. Measured pass-bys of 75 medium heavy vehicles in France found the 2015 table's levels 3 to 6 dB(A) low (Euronoise 2018, paper 210, which gives its measured levels in figures only).

Trains against Schall 03 and the EU's limits

`validation/rail_sources.py` holds the pass-by demos' trains (`acoustic_spaces.demos.rail`, see Trains passing by) against Schall 03's own way of reckoning their level, and beside the limits the EU sets for new trains. The engine's pass is the demo's: each source's front point walked along the track every 1 m with Schall 03's directivity, its other points that much later and louder or quieter, the three heights' energies added; the air at Schall 03's 10 °C and 70 %. Schall 03's sum takes each unit's sound power per metre of track (equation 1) along the same 200 m of track in pieces of 0.1 m, each a point with its directivity (equation 8) heard over a sphere (equation 11) through the air of its Tabelle 17 (equation 12), one train an hour, the SEL 10 lg 3600 dB over its hourly level. It needs none of what the demo assumes: not where a unit's sources are along it, nor how its points share them. Both read the data sheets through the same code (equation 1, checked against the gazette by `tests/test_rail.py`); the directivity Schall 03's sum writes out itself.

In a free field (a ground that takes all the sound), where EN ISO 3095 measures a pass-by:

train	speed	7.5 m, 1.2 m up: engine	Schall 03	25 m, 3.5 m up: engine	Schall 03
S-Bahn	80 km/h	88.3	88.3	82.9	82.8
regional train	160 km/h	98.2	98.2	92.6	92.6

(SEL, dB(A)); every octave band within 0.01 dB. The pass in the engine holds the sound power Schall 03 gives, wherever along the train the demo puts it.

Over the railway example's ground (its ballast bed 4 m wide, `gravel`, and the meadow's `grass` beyond, without its barrier), against Schall 03 with its ground: the solid angle of the reflection near the source (D_Ω, equation 9, about +3 dB) and the ground's attenuation (A_gr,B, equation 14, 0 dB this near):

train	speed	7.5 m: engine	Schall 03	25 m: engine	Schall 03
S-Bahn	80 km/h	87.0	91.1	80.5	84.7
regional train	160 km/h	97.0	100.9	91.5	94.5

The engine gives 3.0 to 4.1 dB(A) less; by octave band, 7.5 m from the S-Bahn, +2.9, +2.2, +0.8, -2.1, -6.1, -4.3, -1.0 and -2.2 dB from 63 Hz to 8 kHz. The two reckon the ground in different ways. Schall 03 adds the reflection's energy, about as much again as the direct sound's at every frequency. The engine adds the reflection with its phase (+6 dB where the two arrive together, at 63 and 125 Hz, a dip where they cancel), off a ground whose impedance it weighs over the reflection's Fresnel zone. Traced at 7.5 m from the S-Bahn: of the 4.0 dB, 0.9 dB is the ballast taking sound where a hard bed would not (87.9 dB(A) with the bed hard), 1.2 dB is the dip of a point at the rail top and its reflection in step with each other at 1 to 2 kHz (89.1 dB(A) with the bed hard and the reflection's energy added, `coherent_max_band_hz: 63`), and the last 2.0 dB is the meadow: up to 2 kHz most of the reflection's Fresnel zone lies on the grass beyond the 4 m bed. Schall 03 puts a train's rolling noise at one point at the rail top; it comes from its wheels, rails and sleepers, at heights apart, where one point's dip would not be as deep.

The TSI's limits. Each unit of the demo's trains alone at 80 km/h over the example's ground, 7.5 m from the track 1.2 m above the rail top: the sound exposure of its whole pass, each of its points walked from 100 m before the listener to 100 m past it (Schall 03's sum over the same 200 m of track), over Tp, the time the unit takes to pass, its length over its speed (what the script prints as L_pAeq,Tp), beside the limits of Commission Regulation (EU) No 1304/2014 (the TSI Noise, table 4) for new units there:

unit	Tp	engine	Schall 03	limit
electric multiple unit (the S-Bahn, 67.4 m)	3.03 s	82.2	86.2	80
electric locomotive (19.0 m)	0.85 s	87.7	90.4	84
coach (26.4 m)	1.19 s	83.9	88.0	79
freight wagon with composite blocks, not a tank wagon (17.8 m)	0.80 s	85.7	89.9	83

(dB(A)). Schall 03's units are 6.2 to 9.0 dB over the limits by its own reckoning, and 2.2 to 4.9 dB over them in the engine. The limits are for new units on a track of low roughness (EN ISO 3095:2013, 6.2); Schall 03's levels are for units of the German fleet on rails in average condition, rougher than the TSI's track. That Schall 03 is over the limits says that, not that either is wrong; no measured pass-by is compared here.

EN ISO 3095:2013 measures a trailer unit, a coach or a wagon, over two identical units one behind the other, from the middle of the first passing to the middle of the second (over Tp), or, as an exception, one unit that is acoustically symmetric, at the end of a train behind a quieter vehicle, from its middle passing until its level is 10 dB under its maximum (over Tp / 2), as S. Radulović et al., "Pass-by noise measurements using one versus two identical trailer units within available standard methods" (RAILCON '24, Niš), quote its figures: for a lone symmetric unit, the table's quantity but for the tail past those 10 dB. In the engine, two of each unit windowed the first way, or one windowed the second, give at most 0.2 dB less than the table; one unit from its front passing to its rear passing, 0.1 dB less for the S-Bahn and 1.1 to 1.7 dB less for the others.

Road works against Defra's rows and the EU's limits

`validation/plant_sources.py` holds the road works demo's machines and reversing alarm (`acoustic_spaces.demos.plant`, see Road works in the street) against the data they come from and the EU's limits for outdoor machines. Each machine is a row of Defra's noise database for construction sites (2005, the data of BS 5228-1 Annex C), its sound power the row's level 10 m away plus 28 dB:

machine	Defra's row, 10 m away	its octaves, A-weighted and added	the engine, 10 m away	sound power
breaker (table 1, row 9)	90	90.3	90.1	118.3
generator (table 4, row 87)	65	64.9	64.8	92.9
dumper (table 4, row 4)	76	75.5	75.3	103.5

(dB(A)). Defra rounds its levels to whole decibels. The engine's is each machine at its height over a hard ground, its reflection added by its energy, heard 10 m away 1.5 m up (Defra gives no height): within 0.2 dB of the octaves' sum, as a point over hard ground radiating into a half sphere gives back.

The EU's limits. Stage II of Directive 2000/14/EC (Article 12) lets a new 6 kVA generator (its electric power taken as 6 kW) send 97 dB(A); Defra's sends 92.9. A wheeled dumper of 75 kW may send 103 dB(A); Defra's sends 103.5, but Defra's is the loudest moment of a drive-by on a site and the directive's a test cycle of its own. The directive's 103 dB(A) for an excavator of 121 kW is for the excavator without its breaker. Breaking, Defra's sends 118.3 dB(A); Epiroc guarantees 121 dB(A) for its MB 1650 (by 2000/14/EC, measured to EN ISO 3744), a 1650 kg breaker as Defra's.

The reversing alarm. In a free field, 1.2 m up facing along its axis, the engine gives 107.0 dB(A) 1.2 m in front of it while it sounds, the 107 of SAE J994 Type B, and 96.0 dB(A) 1.2 m behind it, 11 dB under, as the siren's directivity it is given has it. Its sound power in the scene is its average over its beeps and pauses, 3.1 dB under its sound power while it sounds (108.6 and 111.7 dB(A)).

No measured site is compared here: Defra's levels are averages over the machines it measured, and the made sounds' time structure (the blows, the firing, the beeps' rate) is assumed.

Aircraft noise against Doc 29 Volume 3

`validation/doc29_reference.py` holds `acoustic_spaces.doc29` (see Aircraft by ECAC Doc 29) against the reference cases of Doc 29 Volume 3 Part 1 (5th edition, 2026), which ECAC publishes for checking a new implementation of Volume 2's method: three hypothetical aircraft (a jet with its engines on the rear fuselage, one with them under the wings, a turboprop), each arriving and departing on a straight and a curved route, twelve cases. Volume 3 gives their levels at 18 receptors (its table B-1, which the package holds), every term of every segment at those receptors and the levels on a grid (the worksheets B-2 and B-3 of its Part 1 workbook, which `--workbook` reads).

The levels at the receptors. All 99 within 0.0055 dB of table B-1, which gives them to 0.01 dB; the largest difference of a case is 0.0036 to 0.0055 dB.

The grids. The rms difference δRMS that Volume 3 aims at 0.01 dB or less:

case	points	δRMS	largest	case	points	δRMS	largest
JETFAC	2732	0.0029	0.0053	JETWDC	3868	0.0029	0.0059
JETFAS	2750	0.0028	0.0053	JETWDS	3876	0.0029	0.0056
JETFDC	3269	0.0029	0.0063	PROPAC	3042	0.0028	0.0058
JETFDS	3273	0.0029	0.0058	PROPAS	3042	0.0028	0.0056
JETWAC	2929	0.0030	0.0099	PROPDC	2875	0.0030	0.0061
JETWAS	2932	0.0030	0.0100	PROPDS	2870	0.0029	0.0061

(dB.) The grid's levels are given to 0.01 dB, whose rounding alone is 0.0029 dB rms.

Every segment. Each case has as many segments as Volume 3's, and each term of its 3516 rows matches to within 0.0014 dB for the baseline level, 0.0008 dB for the speed, 0.0003 dB for the engines' installation, 0.0013 dB for the lateral attenuation and 0.0014 dB for the start-of-roll directivity; the finite segment correction differs by up to 0.052 dB on a few segments, and their segment levels with it. Two things of Volume 3's own make those differences, both traced:

- Its ground tracks are 0.0019 % short: the routes in metres were converted to feet through nautical miles of 6076 ft (a nautical mile is 6076.1 ft), so a track's end 100 km out is 1.9 m nearer, and the nodes of a short segment by a turn move by a few tenths of a metre.
- It lays the lengths into which eq. 3-3i cuts a climbing or descending segment by speed, which are lengths along the segment, along the ground track instead: its cuts lie up to 7.3 m farther on than the cuts here.

Cut its way, the segments' places agree within 1.9 m, the finite segment correction within 0.021 dB, and the levels at the receptors change by at most 0.0005 dB.

Two choices follow Volume 3 where Volume 2 leaves room: the heights that cut the initial climb and the final approach are Volume 2's in feet (its heights in metres are rounded: 304.8 m scaled by them gives 37.72 m where Volume 2's example says 37.8 m), and an arrival's landing threshold is its approach's lowest node above the runway (290.2 m before touchdown for the jets, 328.1 m for the turboprop). The azimuth of the start-of-roll directivity is taken in the plane of the runway, as Volume 3 takes it (straight behind a source 1 m up it is 180 degrees).

What is not checked: the maximum levels (Volume 3 gives none; the tests check them beside, behind and ahead of segments against Volume 2's equations) and banked turns (the cases fly them level).

Flight profiles against Doc 29 Volume 3

The same script holds `acoustic_spaces.doc29.performance`, the flight performance of Volume 2 Appendix B (see Aircraft by ECAC Doc 29), against the reference profiles of Volume 3 Part 2: JETW flying departure and approach procedures and PROP departure procedures, nine arrivals and seventeen departures (its tables C-7 and C-8; Volume 3 gives no arrivals of PROP), and the fixed-point profiles Volume 3 gives for them (D-1 and D-2, which the package holds, to 0.1 ft, kt and lb). The cases fly from sea level and from 2000 and 5000 ft, at 15 and 40 °C, at sea-level pressures of 29.92 and 30.71 inHg, into headwinds of -5, 0 and 8 kt; the departures climb at constant speed, accelerate at a rate of climb or with a share of the thrust, cut back to climb thrust or deeply to the minimum reduced thrust, and fly level at constant speed or accelerating; the arrivals descend at constant speed or decelerating, fly level at constant speed, decelerating or at idle, and descend at idle.

The profiles. Every case has as many points as Volume 3's, 314 in all, and every point's height, true airspeed and corrected net thrust per engine agree to Volume 3's rounding (within 0.05 ft, kt and lb). The distances along the track agree within 0.15 ft; Volume 3's own distances for the same step differ by as much between cases (ICAO_A's acceleration from 3787.5 ft is 4678.3 ft long in case 6, 4678.5 ft in case 74, with the same start).

Volume 3's profiles differ from Volume 2's printed equations in three places, and the package follows Volume 3's:

- A propeller aircraft's thrust at the middle of a climb (eq. B-12 in B6.1.2) takes the true airspeed there of the climb's calibrated airspeed, as the jets' thrust takes the middle's altitude, not the rms of the two ends' true airspeeds that Volume 2 prints: that puts PROP's climbs up to 545 ft farther along at 10,000 ft.
- The deceleration of a Descend-Decel step (eq. B-41) takes the speeds along the path over the ground, the true airspeeds less w/cos γ, where Volume 2 prints the true airspeeds less w, divided by cos γ: that moves the thrust by up to 9.3 lb.
- An arrival flies at 90 % of the greatest landing weight, as Volume 2 B7 says (143,299.8 lb), not at the 143,300 lb of the default weights table (C-5): 0.12 lb.

Volume 3's approach procedures are taken as its table C-6.2 prints them, to 0.1 ft, kt and degree, which its profiles are computed from: its workbook's steps have more digits (49.868766 ft where the table prints 49.9), which move the profiles by up to 0.62 ft and 0.86 lb. Its table gives the lengths of the Level-Idle steps in the column of the descent angle; they are the steps' distances.

What is not checked: the idle steps of an approach away from sea level in the standard atmosphere, which Appendix B7.1.2 and B7.1.4 adjust so that the aircraft slows as it does there (all of Volume 3's arrivals are at sea level in the standard atmosphere; the tests check that the deceleration holds), a runway's gradient (all of its runways are level; the tests check eq. B-18), a propeller aircraft's approach (Volume 3 gives none), the noise of these profiles (Volume 3 gives none) and turns (the calculation flies them straight).

An aircraft in the engine against Doc 29

`validation/aircraft_noise.py` holds the airport example's flight demo (`acoustic_spaces.demos.aircraft`, see Aircraft taking off and landing) against Doc 29's own levels: JETW's source (`doc29.source`), simulated in the engine, against its NPD curves in the scene's air, and the example's flights against Doc 29's sum over the same flight path's segments. Over the meadow and the runway, which scatter nothing, the late rays bring the microphones no sound.

The source against its NPD curves. JETW flies straight and level at 160 kt, the NPD curves' speed, over a microphone 1.2 m above grass, at 15,000 lb of thrust an engine departing and 5,000 lb arriving:

distance	departure: L_ASmax	SEL	arrival: L_ASmax	SEL
200 ft (61 m)	99.7 (102.5)	104.0 (104.0)	95.7 (98.4)	99.9 (101.8)
400 ft (122 m)	94.2 (95.4)	100.1 (100.2)	90.1 (91.2)	96.0 (98.0)
630 ft (192 m)	90.1 (90.6)	97.4 (97.4)	86.0 (86.5)	93.3 (95.2)
1000 ft (305 m)	85.6 (85.8)	94.5 (94.4)	81.5 (81.6)	90.3 (92.1)
2000 ft (610 m)	78.1 (78.2)	89.5 (89.3)	74.1 (74.0)	85.5 (87.0)
4000 ft (1219 m)	69.7 (70.0)	83.7 (83.6)	66.0 (65.9)	79.9 (81.4)

(dB(A); the NPD curves' in brackets, with Doc 29's impedance adjustment.) The engine's pass is measured as Doc 29 defines its levels (Volume 2's glossary, section 4.1 and its footnote 15): L_ASmax is the highest of its A-weighted level through a sound level meter's slow time weighting, and the SEL is taken over the time that level is within 10 dB of it. The source is made to the NPD curves' L_Amax 1000 ft away, raised by the 0.29 dB the slow time weighting takes off a point source's loudest moment there, and the engine gives it back within 0.2 dB. Nearer, the NPD curves' L_Amax grows faster than one point source's with one spectrum: at 200 ft the engine reads 2.8 dB (departure) and 2.7 dB (arrival) under them. With no time weighting, the highest level of the places is 2.0 dB over L_ASmax at 200 ft, 1.0 dB at 400 ft, 0.6 dB at 630 ft and less farther out (0.3 dB at 1000 ft, under 0.1 dB at 4000 ft), and the SEL of the whole walk is 0.34 to 0.46 dB over the 10 dB-down one. The departure's SEL is within 0.2 dB of its NPD curves, the arrival's 1.5 to 2.0 dB under: JETW's arrival curves have SEL - L_Amax = 10 lg(πd/V) within 0.22 dB out to 1000 ft, the SEL of a point source's whole pass over its level as it passes, with no time weighting and in air that absorbs nothing. Measured as Doc 29 defines its levels, that source's SEL - L_ASmax is 1.0 dB more at 200 ft and 0.7 dB less at 1000 ft, and the air takes more off; the departure's curves are 1.9 to 2.2 dB under 10 lg(πd/V). Volume 3's aircraft are hypothetical, and its arrival curves seem not to follow those definitions.

The airport's flights. The demo's departure and arrival, simulated every 20 m at each listener, their level following their thrust:

listener	departure: L_ASmax	SEL	arrival: L_ASmax	SEL
departure_4km	87.1 (87.2)	95.9 (96.0)	50.2 (43.1)	64.4 (60.7)
lateral_450m	85.0 (85.1)	95.5 (94.5)	41.8 (54.4)	56.4 (68.8)
approach_2000m	52.0 (54.8)	67.3 (68.6)	90.5 (91.5)	96.8 (98.8)

(dB(A); Doc 29's in brackets.) Where the arrival is heard rolling, the walk starts or ends less than 10 dB under its L_ASmax, so its SEL there is over less than the 10 dB-down time. Where the jet is heard in the air, they agree within 1.0 dB but for the arrival's SEL, 2.0 dB under as its NPD curves have it. Where it is heard rolling on the runway they part, as the two reckon sound running along the ground differently. Doc 29 takes off its lateral attenuation, 10.9 dB at grazing for a listener ahead of a landing roll or behind a take-off roll, from measurements and so averaged over their weather, and behind a take-off roll adds its start-of-roll directivity (5.1 dB down at approach_2000m). The engine reckons the scene's ground in still air: the hard runway reflects the sound back up, so the arrival's roll-out is 7.1 dB over Doc 29 at departure_4km, 2.4 km ahead of it along the runway, while the meadow takes much of it away from 250 Hz up, so it is 12.6 dB under at lateral_450m, 1 km away across the grass; the departure's take-off roll is 2.8 dB under at approach_2000m.

Ground-borne vibration against the FTA's manual

`validation/fta_vibration.py` holds `acoustic_spaces.vibration` (see Vibration through the ground) against the manual it comes from, the FTA's *Transit Noise and Vibration Impact Assessment Manual* (FTA Report No. 0123, 2018).

The base curves against figure 6-4. The manual gives its curves twice: as equations (table 6-10) and drawn (figure 6-4). The equations, against the figure read by eye to about half a decibel:

curve	10 ft	100 ft	300 ft
locomotive-powered trains	94.6 (94.5)	78.4 (78.5)	67.1 (67.0)
rapid transit or light rail	81.6 (81.5)	67.5 (67.5)	55.8 (56.0)
rubber-tyred vehicles	75.5 (75.0)	56.8 (57.0)	50.6 (50.5) at 200 ft

(VdB; the figure in brackets. The bus's curve leaves the figure's 50 VdB at about 200 ft.) The figure draws the curves from 10 to 300 ft; fitted as cubics in log10 of the distance, they turn back up farther out (the bus's past about 1090 ft). `Curve.level` and `ground_level` refuse distances outside that range (`Curve.level` takes them with `extend=True`); `Curve.levels`, which the pass-by demos' rumble and shaking take, holds the 10 ft value nearer and goes on along the curve's slope at 300 ft farther, straight on in log10 of the distance, as `extend` does.

The worked example. Example 6-1 has light rail at 40 mph 125 ft from wood-frame houses on soil that propagates efficiently, reads the base curve there as 65 VdB, and adds: 65 - 1.9 for the speed = 63.1; + 10 for the soil - 5 for the coupling = 68.1; + 6 for the floor's resonances - 2 to the first floor = 72.1. From the same reading the code gives 63.1, 68.1 and 72.1; from the curve itself, 65.43 VdB at 125 ft, 72.5.

The speed rows. Table 6-11's adjustments for 20 to 60 mph against 20 log10 of the speed (eq. 6-4), for rail against 50 mph and buses against 30 mph: all ten within the table's 0.1 dB.

The plant's and the damage criteria's levels. Tables 7-4 and 7-5 give each row's rms level beside its peak particle velocity, "calculated from the PPV limits using a crest factor of 4", 12 dB. All 14 rows of plant and the four damage criteria are within the tables' rounding, 0.5 dB, but for a hydromill in rock: its 0.017 in/s gives 72.6 VdB, and the table prints 75. The code keeps both as printed.

The manual's words. Its text has light rail "70 VdB or more near their tracks" (73.1 VdB 50 ft out on its curve) and buses and trucks rarely over 70 VdB (69.6 VdB 25 ft out). It also puts heavy diesel locomotives "approximately 5 to 10 dB higher" than rail transit vehicles; its two curves are 12.3, 11.5 and 10.9 dB apart at 25, 50 and 100 ft.

No measurement is compared here: the curves are the upper range of measurements of North American systems, and the method gives no spectra.

Edge diffraction in whole cities

Edge diffraction looks, for the source and for each listener, at every pair of an edge and a plane they face, for a path that reflects off the plane before or after bending over the edge. In a city the edges and the planes both grow with the buildings, so the pairs grow with their square: 900 buildings make 20 million pairs per listener. Most pairs cannot carry a path. The ray tests start just off the edge and head for the mirror image of the source or listener in the plane, so a ray from an edge behind the plane never reaches it, and one from the rest of the edge meets the plane where the image casts the edge's shadow on it, to within a margin that the ray's slant sets. Where that shadow misses the box round the plane's faces, no reflection can happen, and the engine leaves the pair out before tracing a ray. For a listener's images it also leaves out the paths that would touch the edge's line beyond its ends or arrive too late. Pairs whose rays graze the plane, or whose image lies too near it for the margin to hold, are all kept. The exact tests then run on what is left, as before, so the results do not change.

`validation/city_speed.py` builds cities of n by n blocks 32 m apart, each a building of random size (12 to 20 m a side) and height (6 to 30 m), with grass in the streets. A source stands in a street crossing in the middle, 1 m up, and two listeners stand 3 and 5 blocks off. The simulation is the outdoor default: early reflections to the third order, the direct sound and first-order reflections bending over the edges, and 50,000 rays per band for the late part. Times are from one run (`--sizes 10 20 30 40 60`) on 4 cores of a cloud machine with nothing else running:

Buildings	Faces	Edges	Build (s)	Simulation (s)	Of it diffraction (s)	Pairs tested	Peak memory
100	1,012	800	0.1	3.1	2.1	8.5 %	0.3 GB
400	4,012	3,200	0.4	7.0	6.4	4.3 %	0.3 GB
900	9,012	7,200	1.0	12.9	12.1	2.9 %	0.4 GB
1,600	16,012	12,800	2.1	17.9	16.9	2.7 %	0.5 GB
3,600	36,012	28,800	5.3	48.1	46.7	3.3 %	1.8 GB

Diffraction includes finding the model's edges and where they end, and the terms that make the other specular paths fade at the edges (above): those terms took three quarters of the simulation of 900 buildings and half of that of 3,600, and finding the edges about a fifth of the latter. In a one-off check when the culling was written, on a run where 900 buildings simulated in 16.6 s (15.6 s of it diffraction), 1,600 in 21.4 s (20.1 s) and 3,600 in 56.4 s (54.5 s), they did so without those terms in 2.6 s (1.6 s), 4.8 s (3.6 s) and 24.4 s (22.4 s). The tested share stays near 3 %: the shadow of a wall facing the source covers a wedge of the city that widens away from it, and only the ray tests tell whether other buildings hide the wall. The work therefore still grows with the square of the buildings, and a few thousand buildings take tens of seconds per source. Tens of thousands need the map cut down to the area round the sources and listeners, or `diffraction: direct`, whose cost grows with the edges alone.

Exactness. The script's check runs a city of 4 by 4 blocks with the buildings turned at random angles and paths that reflect on both sides of an edge. It runs two sources, one in a street and one over the roofs, to two listeners, once as the engine runs, on one thread and on every core, and once with every pair tested. Every path, its energy and its spectrum come out the same bit for bit; the early paths took 0.9 s instead of 3.0 s. When the culling was written, the same comparison over every example and three test cities, with paths reflecting on one side and on both sides of an edge and over two edges, found no difference either (a one-off check; the script repeats the 4 by 4 city). The tests check the culling against rays traced from random edges to a tilted rectangle of faces, at grazing angles, for faces that stray from their plane and for planes with only openings.

Wave-based solver

`validation/wave_solver.py` checks the wave solver against theory: closed forms where there are any, and for a whole room the sum of its modes. It has two stencils, each at its stability limit: the standard one (7 points, the six axial neighbours) and the interpolated isotropic one (19 points, the twelve edge neighbours too), which closed rooms take. Each check runs with both where both apply. (Each takes its Laplacian times $1 - 2^{-18}$, which slows sound by 2^{-19}: at the limit itself the highest modes, the 7-point stencil's checkerboard and the 19-point one's modes alternating along two axes, sit on the edge of stability, where a kick grows linearly and single precision's rounding can take them over it.)

Speed of sound on the grid. Waves on the grid travel a little slower the shorter they are. From the peaks of a rigid 2.4 x 2.4 x 0.2 m box (10 cm cells) ringing for 8 s, against the exact frequencies of its modes and against each stencil's dispersion relation, $2 - 2\cos \omega T = \Sigma\, g_k\,(1 - \cos k \cdot d_k)$ over its links:

direction	mode	cells per wavelength	7-point, relation	wave solver	19-point, relation	wave solver
along an axis	(6, 0, 0)	8.00	−1.740 %	−1.740 %	−0.672 %	−0.672 %
along a diagonal	(4, 4, 0)	8.49	−0.387 %	−0.388 %	−0.573 %	−0.573 %

Both follow their relations within 2e-5. At 8 cells per wavelength the 7-point stencil is 1.74 % slow along the axes, 0.44 % along the face diagonals and exact along the body diagonals; the 19-point one is 0.65 to 0.67 % slow in every direction. It needs 5.2 cells per wavelength to be as close as the 7-point one is at 8.

Modes and their decay. A 7 x 5 x 3 m box rings for 8 s (10 cm cells). Its walls' admittance changes with frequency as a material's does on the wave solver's grid: a constant 0.002 and a branch resonating at 60 Hz (Q 1.5, admittance 0.008 at resonance), so below 60 Hz they give like a spring and above it weigh like a mass. For each mode that stands apart from the others, the frequency of its peak and the decay rate of its envelope match first-order perturbation theory: the decay rate $\delta = c\,\mathrm{Re}(\beta)/2\,\Sigma$ (4/L along each axis the mode varies, 2/L along the others), β the walls' admittance at the mode, and the rigid box's frequency moved by $-c\,\mathrm{Im}(\beta)/(4\pi)$ times the same sum. With either stencil the frequencies come out within 0.25 % of theory (the walls move them by up to 0.5 %), the decay rates, which differ more than fourfold, within 1.8 %:

mode	rigid box (Hz)	theory (Hz)	7-point (Hz)	19-point (Hz)	Re(β)	decay, theory (1/s)	7-point (1/s)	19-point (1/s)
(1, 0, 0)	24.50	24.39	24.39	24.39	0.0028	0.779	0.770	0.770
(0, 1, 0)	34.30	34.14	34.14	34.15	0.0039	1.185	1.167	1.168
(1, 1, 0)	42.15	41.93	41.91	41.91	0.0057	1.990	1.955	1.958
(2, 0, 0)	49.00	48.84	48.87	48.88	0.0078	2.200	2.191	2.195
(0, 0, 1)	57.17	57.10	56.97	56.98	0.0098	3.407	3.442	3.445
(0, 2, 1)	89.30	89.56	89.55	89.54	0.0052	2.152	2.124	2.117

Each decay rate is the slope of the mode's envelope from 1 s on, over 50 dB at most: the fastest modes reach the noise floor of single precision before the run ends. The theory's rates move by up to 0.35 % on the 19-point grid, whose time step is longer.

Axial modes lose energy at walls they meet head-on and at walls they graze, so this checks absorption at all angles.

Walls. A plane wave down a duct (10 cm cells) reflects off its end as (1 − β)/(1 + β) predicts at normal incidence within 0.004 for β up to 0.5, but returns 0.044 where it predicts nothing (β = 1); grazing its absorbing walls it decays as exp(−βP/(2A)) per metre (perimeter P, cross-section A), whatever the cell size:

end admittance β	0	0.1	0.25	0.5	1
reflection, theory	1.000	0.818	0.600	0.333	0
7-point stencil (60 to 150 Hz)	0.998	0.817	0.601	0.336	0.044
19-point stencil	0.999	0.818	0.602	0.337	0.044

walls β = 0.05, 0.6 x 0.6 m duct, amplitude after 10 m	theory	10 cm cells	5 cm cells
7-point stencil	0.1889	0.1900	0.1906
19-point stencil	0.1889	0.1905	0.1907

Walls that absorb as their materials do. The wave solver gives each material an admittance that changes with frequency: a constant and resonant mass-spring-damper branches, one at each wave band's centre and one beyond each end, fitted so that the random-incidence absorption, counted through the analysis's band filters, matches the material's in every band. At the grid of the 500 Hz band:

material, 63 to 500 Hz	absorption	fitted	branches
porous absorber	0.05, 0.15, 0.40, 0.70	0.050, 0.150, 0.400, 0.700	6
panel absorber	0.35, 0.30, 0.15, 0.08	0.350, 0.300, 0.150, 0.080	5
audience	0.50, 0.60, 0.75, 0.85	0.500, 0.600, 0.750, 0.850	6
plaster on concrete	0.01, 0.01, 0.015, 0.02	0.010, 0.010, 0.015, 0.020	6
a steep resonance	0.10, 0.60, 0.25, 0.10	0.136, 0.579, 0.253, 0.100	4

The octave filters overlap, so a band next to a much more absorbing one cannot absorb much less than it lets through: the steep resonance is met within 0.04. Simulations report the largest miss as `wave_absorption_miss`. With the porous absorber's fitted admittance at the end of a duct (5 cm cells, 0.2 m across), against a rigid end's reflection, which the grid returns whole at every frequency:

frequency (Hz)	31.5	44	63	88	125	177	250	354	500	707
7-point: normal-incidence absorption, fitted admittance	0.030	0.022	0.022	0.037	0.075	0.138	0.259	0.366	0.567	0.597
7-point: the grid's theory	0.030	0.022	0.022	0.037	0.074	0.137	0.253	0.355	0.542	0.571
7-point: wave solver	0.030	0.022	0.022	0.037	0.074	0.137	0.253	0.355	0.542	0.571
19-point: normal-incidence absorption, fitted admittance	0.030	0.022	0.022	0.037	0.072	0.143	0.268	0.346	0.532	0.659
19-point: the grid's theory	0.030	0.022	0.022	0.036	0.072	0.142	0.262	0.338	0.509	0.651
19-point: wave solver	0.030	0.022	0.022	0.036	0.072	0.142	0.262	0.338	0.509	0.651

(Each stencil has its own time step, so its own fit.) The wave solver matches the grid's theory (a plane wave on the grid, the wall half a cell past the last cell centres) within 0.0002 with either stencil, from 30 to 720 Hz, and the rigid end, which the grid returns whole, measures within 0.0002 of 1. (The source also excites the 20 cm duct's first symmetric cross mode, which rings on just above its cut-on, 1.59 kHz on the 7-point grid, for as long as a run lasts, and an abrupt cut spreads it over every frequency. So the runs' end fades over 20 ms, which brings the absorption that close, and so does the window that cuts out the incident wave, which brings the rigid end that close.) The half cell between the last cell centres and the wall takes a little of the absorption away as the wavelength shortens: within 0.013 (7-point) and 0.009 (19-point) of the fitted admittance's below 360 Hz, 0.025 and 0.023 less at 500 Hz with 14 cells per wavelength. The fit does not correct for it.

Open models. Round a model whose air reaches the edge of the grid, the grid grows by a perfectly matched layer (split-field, its damping growing as the square of the depth). A rigid 8 x 6 m plate in the open, the source 1.5 m above it, three listeners round it, 25 to 250 Hz (20 cm cells, the 7-point stencil, the only one with the matched layer), against a grid 16 m larger all round, from whose edge nothing comes back in the 90 ms of the runs: what differs, relative to the sound itself (the runs' last 10 ms fade out: between 500 Hz and 1 kHz, where a wavelength has fewer than four of these cells, the 8-cell layer returns sound only 15 to 30 dB below the sound there, in a one-off check, and an abrupt end spreads some of it over the bands compared),

boundary of the grid	listener 1 (dB)	listener 2	listener 3	air cells
the grid's edge alone (admittance 1, as in 2.0.0)	−6.6	−13.5	−11.0	28,380
matched layer, 6 cells	−61.7	−64.9	−60.0	79,200
matched layer, 8 cells (the default)	−72.2	−74.1	−74.4	104,076
matched layer, 12 cells	−80.7	−81.7	−82.2	168,036

The edge alone absorbs only sound that meets it head-on; what a layer returns falls with its depth at every listener. Runs take 8 cells, which is not a setting. In the layer the sound keeps falling over 59,409 steps (20 s): 78 dB below the first second's level after one second, at least 134 dB below it through the second half.

Sound that runs along the layer meets it near grazing, where it sends much more back. In one-off checks over a rigid plate (open below), against the exact solution over a rigid plane, a layer only a few cells above the source and the listener, both 1.5 m above the plate, puts the level 0.26 dB low at 125 Hz 20 m away, 0.50 and 1.71 dB low at 63 and 125 Hz 40 m away (20 cm cells), and at 63 Hz 2.84 dB low 80 m away and 9.63 dB low 160 m away (40 cm cells); with 5 cm cells, the two 0.1 m up and 7 m apart, 0.63 dB high at 125 Hz. So the grid reaches past the source and the listeners, on each side the model leaves open to them, by 15 % of the farthest a listener lies from the source along that side (none under an outdoor ground, and in whole cells, so that the grid keeps its place on the model). With that room the same runs are within 0.03 dB of the exact solution 20 to 160 m away, where three times the room moves them by 0.02 dB at most (to 80 m away; 160 m away that grid has more cells than a run takes), and within 0.04 dB with the 5 cm cells. One case stays 0.07 and 0.15 dB off at 63 and 125 Hz: the source and the listener 0.1 m up and 20 m apart with 10 cm cells, whose layer is 0.8 m deep; 25 % of room brings it within 0.01 dB. (In two more one-off checks, with the trilinear reading: beside the plate's edge the room brought the level at 125 Hz within 0.01 dB of a wider grid's, and over the rigid 3 m block of the edge diffraction checks, at 6.2 cm cells, to −9.00 and −11.21 dB against BTM's −9.04 and −11.25. Set up again, that block reads −9.07 and −11.35 dB trilinearly and −9.06 and −11.31 dB with the cubic reading.) It costs cells on each open side where the model reaches less far than that room: in these checks 1.5 and 2.0 times as many 20 and 40 m away at 20 cm cells and 2.2 and 3.5 times 80 and 160 m away at 40 cm cells. A side that the model's own surfaces close round the source and the listeners gets none: the air round them, flooded over the grid's cells within that room, reaches no layer there (so the plate's grids grow above it and to its sides, not below it). In a dry layout of a hall 40 x 20 x 4 m open at one end, its listeners 8 to 33 m from the source, the grid stays on the hall, where room on every side would take 1.8 and 2.0 times the cells at 125 and 250 Hz, under its floor and over its roof; its levels are within 0.001 dB of those on a grid with three times the room. Outdoors, where nothing is added under the ground, only where the sky (by default 30 m above the highest building) or the ground's edge lies nearer than that room: 13 % more on the 63 Hz grids of the wave solver in wind and temperature (a sky 30 m up, listeners up to 200 m away), with or without moving air, and none on its 125 Hz grids, on the free field and façade grids of moving air or on the ground reflection's wave check.

One run for all bands. A 7 x 5 x 3 m box whose absorption differs in every band (0.10 to 0.25 from 63 to 500 Hz), the wave solver up to the 500 Hz band: one run of 868,701 cells and 5,617 steps of the 19-point stencil (the 0.5 s response and 0.2 s past its end), 5.0 s on 4 cores with the fit and the grid's set-up, where 2.0.0 ran one per band. Timed on its own over as many steps, the same grid runs in 4.2 s with the walls' six branches and in 3.3 s with constant walls: the branches cost about a quarter more than a run with constant walls, and save three runs.

The end of a response. The bands' filters spread each arrival both ways in time, so they read a response's last slots from the run after them too, and a simulation steps the solver 0.2 s further than its responses last (0.4 s with third-octave bands). In a 5 x 4 x 3 m box with 2 % absorption, the wave bands' energy per 10 ms of a 1 s response against the same slots of a 2 s response, the largest difference over two listeners and the bands up to 125 Hz:

bands	the run	last five slots (dB)	earlier slots (dB)
octave	stops at the response's end	7.54	0.19
octave	goes on for 0.2 s	0.0001	0.0000
third-octave	stops at the response's end	15.72	1.67
third-octave	goes on for 0.4 s	0.0016	0.0009

The run that goes on also keeps a slow drift out of the bands. In single precision the 19-point stencil's rounding lets a closed room's mean pressure creep. That is far below the bands, and only the end of a run brings it into them. In a one-off check of a rigid room against the same scheme in double precision, the mean was as large as the sound after 100 s: the bands' last five slots were then 5.9 dB off with a run that stopped at the response's end, and every slot within 0.012 dB with the run going on.

Free field. In a 30 x 24 x 24 m room with 95 % absorption, the direct sound's energy in the 63 and 125 Hz bands along x from the source, in dB over the free-field level the rays give:

distance	19-point, 63 Hz	19-point, 125 Hz	7-point, 63 Hz	7-point, 125 Hz
1 m	−0.03	+0.01	+0.06	+0.08
2 m	−0.02	−0.00	+0.00	+0.01
4 m	−0.01	−0.01	−0.00	−0.01
8 m	−0.04	−0.02	−0.04	−0.02

That is for one place of the source within its cell and one direction. The source and each point a listener samples lie between the cells' centres and take four cells a side, weighted as a cubic through their centres (see Simulation). Over 18 places of the source within a cell, with listeners in 50 directions, the grid's 26 axes and diagonals and 24 others (in a rigid cube large enough for nothing to come back within the direct sound's first 50 ms), against the exact field of a point source with the same pulse: the pressure's energy, and the energy of the pressure's gradient (the first-order field's X, Y and Z) over the pressure's, each in dB over the exact one, from the least to the most; and how far the gradient's main axis lies off the line to the source at most, in the worse of the two bands:

cells from the source	19-point: pressure, 63 Hz	125 Hz	gradient over pressure, 63 Hz	125 Hz	off the line	7-point: pressure, 63 Hz	125 Hz	gradient over pressure, 63 Hz	125 Hz	off the line
1	−1.76 to +2.86	−1.79 to +2.64	−7.30 to −0.01	−6.49 to +0.16	11.4°	−1.62 to +2.38	−1.49 to +2.35	−5.87 to +3.86	−5.53 to +3.57	14.9°
1.5	−1.36 to +1.53	−1.19 to +1.17	−6.29 to +0.69	−5.12 to +0.88	15.6°	−0.88 to +1.68	−0.70 to +1.60	−0.96 to +6.18	−0.88 to +5.53	21.2°
2	−0.59 to +0.61	−0.57 to +0.37	−4.48 to +0.61	−3.64 to +0.45	13.4°	−0.31 to +0.67	−0.21 to +0.64	−1.49 to +5.41	−1.20 to +4.58	14.6°
2.5	−0.44 to +0.42	−0.38 to +0.10	−0.80 to +3.66	−0.66 to +2.64	9.8°	−0.20 to +0.83	−0.11 to +0.58	−3.41 to +2.23	−2.10 to +1.78	5.3°
3	−0.13 to +0.31	−0.03 to +0.05	−0.75 to +1.20	−0.71 to +0.59	7.2°	−0.07 to +0.32	+0.18 to +0.26	−1.68 to +0.74	−0.99 to +0.42	4.4°
4	−0.04 to +0.04	−0.04 to +0.02	−0.43 to +0.27	−0.15 to +0.18	2.3°	−0.02 to +0.16	+0.12 to +0.25	−0.33 to +0.24	−0.10 to +0.24	1.5°
8	−0.00 to +0.00	−0.02 to +0.01	−0.01 to +0.01	+0.01 to +0.02	0.1°	+0.03 to +0.04	+0.02 to +0.32	−0.04 to +0.05	−0.03 to +0.09	0.4°
12	−0.00 to +0.00	−0.02 to +0.00	+0.00 to +0.00	+0.01 to +0.02	0.1°	+0.01 to +0.05	−0.01 to +0.34	−0.02 to +0.03	−0.02 to +0.06	0.4°

From four cells on the 19-point stencil’s level is within 0.04 dB wherever the source and the listener lie within their cells and in every direction. A grid has a gain of its own on a point source’s sound, more the higher the frequency. The 19-point stencil’s is the same in every direction within 0.01 dB over a band, 9 / (5 + 4 cos ωT) in pressure (0.06 and 0.35 dB over these two bands), and three taps in the source’s signal undo it. The 7-point stencil’s changes with the direction, so no signal can: twelve cells away in the 125 Hz band it reads −0.01 to +0.01 dB along an axis, +0.25 dB along a face’s diagonal and +0.32 to +0.34 dB along the cube’s (the 19-point one −0.02 to +0.00 dB in all three).

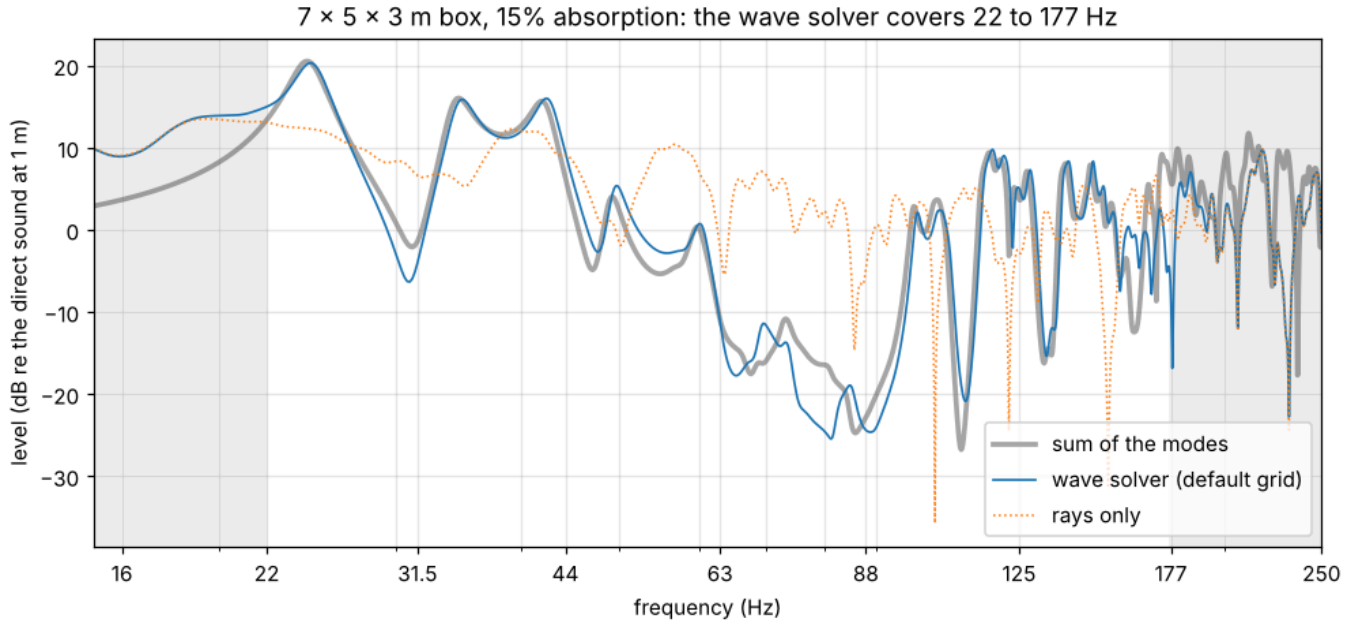
Within three cells of the source its sound on the grid is not a point's. There a listener's points take the eight cells round them, weighted trilinearly, which read the 19-point grid's pressure and the 7-point grid's gradient closer than the cubic does (not the 7-point grid's pressure: 2.5 cells away it is up to 0.83 dB off, against 0.35 dB with the cubic). Where the reading changes, three cells from the source, the level steps by up to 0.37 dB in the octave up to 8 cells per wavelength and 0.15 dB in the one below (40 directions, in a one-off check). A listener's gradient comes from points up to two cells from it and settles a cell further out. On the 19-point grid it goes without a point that lies nearer the source than 1.5 cells and nearer it than the listener: with those points its main axis is up to 26 and 21 degrees off the line 1.5 and 2 cells away and its energy up to 13.2 dB off 2.5 cells away (16 and 13 degrees and 3.7 dB without them). A listener nearer the source than 1.5 cells so keeps the points on its far side, and from 0.9 cells out the gradient's energy changes by 2.4 dB at most from one listener to the next 0.1 cells further out (a one-off scan every 0.1 cells from 0.5 to 2). On the 7-point grid leaving the points out gains nothing clear, and it keeps them. At some distances the gradient's energy over the pressure's is further off than with the source and every point read trilinearly and no point left out: 3.0 to 3.9 and 4.1 to 4.8 cells away on the 19-point grid (1.6 dB against 0.6 dB at 3.5 cells, 0.41 against 0.29 at 4.4), and 1.2 to 3.9 cells away on the 7-point one (5.4 dB against 3.8 dB at 2 cells, 3.3 against 1.6 at 2.6, 0.62 against 0.45 at 3.8); elsewhere it is closer. Its main axis lies up to 4.9 degrees further off the line (12.3 against 7.4 degrees, 2.3 cells away on the 19-point grid). From 2 to 3.4 cells away that comes mostly from the source: spread over four cells a side, its sound is not a point's within about two cells of it, where some of a listener's gradient points lie (two cells away on the 7-point grid, 4.0 dB off with the source over its eight cells and everything else the same), and on the 19-point grid an axis that goes without a point near the source is a second-order difference. From 3.4 to 3.8 cells away on the 7-point grid, and 3.8 and 4.4 to 4.8 cells away on the 19-point one, it comes from the listener's reading, and part of it from the measure: read trilinearly, the pressure loses about as much as the gradient, which cancels in their ratio (up to 0.16 dB), while the pressure read as here is the closer of the two (3.6 cells away on the 7-point grid, 0.24 dB off against 0.36 dB). (One-off scans every 0.1 cells from 1 to 6, over 18 places of the source and 50 directions, the worse of the two bands.)

Next to a surface along the grid (a floor, a wall, flat ground) a point leaves out the cells behind it: across the surface it reads the two cells round it, weighted linearly (within half a cell of the surface the one layer on its side), and along the surface still the cubic. A source and a listener 0.3 to 2.5 cells above a rigid plate and 20 m apart, with 20 cm cells, read the 125 Hz band within 0.02 dB of the exact solution wherever they lie within their cells (0.26 to 0.47 dB low read trilinearly, in one-off checks). Sound that arrives across such a surface is read linearly along its way: within 1.5 cells of a wall along the grid, the source 8.3 cells from the wall and 7 to 11.6 cells from the listeners, the level in the octave up to 8 cells per wavelength is 0.14 to 0.39 dB low on the 19-point grid, and between 0.42 dB low and 0.25 dB high on the 7-point one.

Next to a surface at an angle to the grid, an edge or an opening, where the cells round a point are not all its own, it reads the cells it sees, weighted trilinearly (up to 0.7 dB low at 8 cells per wavelength, halfway between two centres along an axis). Along such a wall, listeners 0.05 cells apart differ from one to the next, in the octave up to 8 cells per wavelength, by up to 0.51 dB 0.3 cells from it (0.24 dB at 45 degrees) and 1.2 dB 0.1 cells from it, as they do read trilinearly everywhere. Where the reading changes to the cubic, 1.5 to 2 cells from the wall, the level steps by up to 0.41 dB (0.12 dB read trilinearly everywhere, which reads low instead; walls at 3 to 45 degrees), and 1 to 2 cells above the top of a screen along the grid by up to 0.37 dB (0.09 dB; all in one-off checks).

A whole room. A 7 x 5 x 3 m box with 15 % absorption on every surface, simulated with the wave solver switched on at its default settings (the 63 and 125 Hz bands, 20 cm cells, the 19-point stencil), against the sum of the box's modes up to 1.5 kHz, each with its decay rate, all analysed alike (ISO 3382-1, octave filters):

model	G 63 Hz (dB)	G 125 Hz	EDT 63 Hz (s)	EDT 125 Hz	T20 63 Hz (s)	T20 125 Hz
sum of the modes	20.3	23.5	1.02	0.96	0.94	0.90
wave solver	21.1	22.8	1.04	0.95	0.94	0.87
wave solver, 7-point stencil	21.1	23.1	1.04	0.94	0.94	0.87
wave solver, 16 cells per wavelength	20.8	23.7	1.01	0.84	0.90	0.90
rays only	25.5	21.5	0.69	0.77	0.92	0.78



The wave solver follows the modes peak for peak from 22 Hz up. What is left comes mostly from where the grid puts the walls: within half a cell of the model's, here 6.93 x 4.95 x 2.97 m with 20 cm cells, which moves each mode by up to 1 %. In a room this sparse in modes, that moves G by up to 1 dB, EDT by up to 12 % and T20 by up to 5 %; more cells per wavelength move the walls but do not make every figure converge. The two stencils' times agree within 0.02 s: in this band their dispersion moves the modes less than the walls' placement does. The 7-point one's G is 0.3 dB higher at 125 Hz, where its own gain on the source's sound is not undone (see Free field): without its taps the 19-point one's is within 0.01 dB of it. Rays give the average over the room instead: 5.2 dB more at this seat in the 63 Hz band, where the listener sits near the nodes of most modes.

Where the grid puts the walls. `validation/wave_walls.py` measures what walls on cell faces cost (the leftover under A whole room, above), with both stencils on the grids the solver takes when its highest band is the 63, 125 or 250 Hz one, at 8 points per wavelength (a run up to 125 Hz, the default, takes the 19.8 cm grid for every band). A rigid 4.13 x 3.37 x 2.71 m box, whose sides are no whole number of cells, is placed as the solver places a room whose source and listeners lie more than two cells from its walls, moved from there by parts of a cell along the axes (0, 0, 0; 0.25, 0.5, 0.75; 0.5, 0.25, 0.4), turned by 3, 8 and 14 degrees about the vertical, and turned by 8 and tilted by 5 degrees. Its modes are the eigenvalues of the grid's own update, each analytic mode taken at the eigenvector its shape correlates with best. With every side rounded to whole cells and the walls on cell faces, the grid's modes follow the stencil's dispersion relation within 2e-6 (the 2^{-19} by which the solver slows sound, above): what is off there is the dispersion alone. In each grid's own band (sharp edges at the synthesis's crossovers, 89.1, 176.8 and 353.6 Hz), the worst mode (and the largest root-mean-square) over the placements:

stencil	grid	sides in whole cells, walls on cell faces	moved within a cell	turned or tilted
19-point	63 Hz band's (40.5 cm)	−0.4 % (0.3 %)	+11.3 % (7.0 %)	−6.7 % (4.4 %)
19-point	125 Hz band's (19.8 cm)	−0.4 % (0.3 %)	+4.9 % (2.0 %)	−6.4 % (3.2 %)
19-point	250 Hz band's (9.9 cm)	−0.4 % (0.3 %)	−2.6 % (1.2 %)	−2.8 % (1.3 %)
7-point	63 Hz band's (39.6 cm)	−1.1 % (0.6 %)	−6.1 % (4.0 %)	−10.3 % (5.7 %)
7-point	125 Hz band's (19.8 cm)	−1.0 % (0.5 %)	+4.5 % (2.0 %)	−7.0 % (3.6 %)
7-point	250 Hz band's (9.9 cm)	−1.0 % (0.4 %)	−3.2 % (1.2 %)	−3.7 % (1.6 %)

A box turned against the grid couples modes that lie close together, and a mode's shape then spreads over several eigenvectors: on their own grids, 1 to 11 of the 31 modes in the 125 Hz band and 22 to 120 of the 195 in the 250 Hz band correlate less than 0.8 with the one they match best (19-point; 0 to 11 and 4 to 117 with the 7-point one; none on the 63 Hz band's grids or in a box moved within a cell). Of the modes that keep their shape, the worst turned or tilted on those two grids is −4.5 % and −2.5 % off (19-point; −6.8 % and −3.7 % 7-point); the furthest eigenvector that holds a quarter of a mode's shape (a squared correlation of 0.25), −6.4 % and −3.6 % (−7.4 % and −4.2 %). These are samples, not bounds: which placement is worst depends on how each side rounds to whole cells (on the 9.9 cm grid the first two placements round every side alike), and cells 0.06 % smaller (at 343.0 m/s instead of 343.2) move the worst modes by up to 0.25 percentage points; which modes keep their shape hangs on the 0.8: in the tilted box on the 7-point 125 Hz band's grid, those smaller cells take the worst of them from −6.8 % to −4.6 % (one-off checks).

The band levels at five listeners 0.74 to 1.19 m from the nearest wall, the source 0.41 to 0.53 m from three (on the 63 Hz band's grids that is within two cells, where the solver would start its grid less than a cell further out: the placements are samples either way), against the box's image sources with the solver's own pulse (more images and a longer FFT move them by less than 0.001 dB), in the same sharp-edged bands (the 63 Hz one from 44.5 Hz), are off by up to 2.7 / 2.8 dB (first 100 / 300 ms) in the 63 Hz band on its grid, 2.5 / 4.3 dB in the 125 Hz band and 4.4 / 2.8 dB in the 250 Hz band with the 19-point stencil (3.3 / 2.9, 2.4 / 4.6 and 4.0 / 3.7 dB with the 7-point one), the most over the listeners and the seven placements; with the walls on cell faces, 0.56 / 0.30, 0.24 / 0.12 and 0.15 / 0.15 dB (19-point). On the 19-point grid an octave finer than a band's own, the 63 Hz band is off by up to 1.2 / 1.9 dB and the 125 Hz band by up to 1.4 / 0.8 dB.

A single rigid wall, against the same grid's free field driven by the source and by its image in the model's wall (so that the grid's dispersion and the source are the same on both sides): the root-mean-square error over the cell centres 1.5 to 6.5 cells in front of it, by octave of cells per wavelength, with the 19-point stencil (the 7-point one within 0.08 dB of it), and the place of the wall that fits the error best in the first octave, along its normal in steps of 0.01 cell (negative: behind the model's wall, away from the air). The walls along the grid lie that share of a cell into the air from a cell face:

wall	8–16 cells per wavelength	16–32	32–64	place fitted (cells)
along the grid, 0.1 cell off a cell face	0.35 dB	0.21 dB	0.10 dB	−0.10
along the grid, 0.25 cell off	0.91 dB	0.53 dB	0.24 dB	−0.25
along the grid, 0.4 cell off	1.50 dB	0.85 dB	0.38 dB	−0.40
at 3 degrees to the grid	1.17 dB	0.39 dB	0.22 dB	−0.04
at 8 degrees	0.93 dB	0.07 dB	0.03 dB	−0.04
at 14 degrees	0.68 dB	0.20 dB	0.05 dB	−0.06
at 25 degrees	0.54 dB	0.15 dB	0.07 dB	−0.04

Along the grid the place is the whole error: against the source's image in the cell face the grid puts the wall on, what is left is below 0.001 dB, as it is with the model's wall on a cell face. At an angle, the wall's place that fits the error best lies within 0.08 cells of the model's wall with either stencil, and the error is in the steps.

Fitted walls. The same script runs its parts with fitted walls (`simulation.wave.walls: fitted`) on the 7-point stencil, the only one they take, beside walls on cell faces, and four parts of their own; each figure the simulation page states for them is one of its checks, to the precision it is stated in (`--walls` picks the kinds of walls). In the boxes with every side a whole number of cells the grid's modes follow the stencil's relation within 2e-6, as with walls on cell faces (the update, $D^{-1}K$ with K symmetric and D the cells' volumes, is symmetric within 3e-8). In each grid's own band, at every placement moved or turned, fitted walls' modes are off by the stencil's dispersion alone, as with every side in whole cells: −1.1, −1.0 and −1.0 % at most on the 63, 125 and 250 Hz bands' grids; 2 of the 195 modes of the 250 Hz band mix at most, none on the other grids. The box's 16 lowest modes against the exact ones with the dispersion taken out (least … most, %; on the 125 Hz band's grid also the worst over 24 more placements, the grid's origin drawn within a cell):

grid	turned (degrees)	walls on cell faces	fitted walls	fitted, 24 more placements
125 Hz band’s (19.8 cm)	0 / 0	−2.31 … +0.04	−0.03 … −0.00	0.05
	3 / 0	−2.54 … +0.08	−0.03 … +0.01	0.05
	8 / 0	−4.19 … −0.95	−0.03 … +0.10	0.12
	14 / 0	−2.92 … −0.62	−0.03 … +0.37	0.37
	22.5 / 11	−5.72 … −1.19	+0.12 … +0.95	1.00
	30 / 20	−6.03 … −1.59	+0.43 … +1.77	1.83
	45 / 0	−7.12 … −0.61	−0.03 … +3.48	4.30
	40 / 35	−6.19 … −2.17	+1.12 … +3.26	3.56
250 Hz band’s (9.9 cm)	0 / 0	−0.75 … +1.30	−0.00 … −0.00	
	3 / 0	−0.91 … +1.30	−0.00 … +0.01	
	8 / 0	−0.95 … +1.30	−0.00 … +0.05	
	14 / 0	−1.42 … +1.30	−0.00 … +0.18	
	22.5 / 11	−2.03 … −0.52	+0.06 … +0.49	
	30 / 20	−2.61 … −0.81	+0.20 … +0.92	
	45 / 0	−3.46 … +1.30	−0.00 … +1.26	
	40 / 35	−3.36 … −1.26	+0.58 … +1.77	

Moved along the grid (the three placements above), fitted walls’ 16 lowest modes are within 0.03 % on the 125 Hz band’s grid and 0.004 % on the 250 Hz band’s, walls on cell faces up to 5.2 % and 2.3 % off. Every mode keeps its shape with fitted walls (correlation 0.987 or more; on cell faces down to 0.71). Turned further than 14 degrees the modes come out high, more the steeper. At 45 degrees the worst over the placements falls in two groups, 3.2 and 4.3 %, by where the two walls lie against the grid’s diagonal planes through the cells’ centres; on the 250 Hz band’s grid eight more placements gave 1.9 % there (one-off checks).

The box’s band levels (the same listeners, windows and placements) are off by up to 0.3 / 0.5 dB in the 63 Hz band on its grid, 0.2 / 0.4 dB in the 125 Hz band and 1.1 / 1.2 dB in the 250 Hz band on theirs (first 100 / 300 ms), where walls on cell faces are 3.3 / 2.9, 2.4 / 4.6 and 4.0 / 3.7 dB off on the same stencil and the box with every side in whole cells, which is another box, 0.30 / 0.29, 0.10 / 0.26 and 1.02 / 0.49 dB.

The single wall, as above, on the 7-point stencil (dB rms in front of it, by cells per wavelength):

wall	on cell faces: 8–16	16–32	32–64	fitted: 8–16	16–32	32–64	fitted, place fitted (cells)
along the grid, 0.1 cell off a cell face	0.34	0.21	0.10	0.02	0.00	0.00	+0.00
along the grid, 0.25 cell off	0.89	0.53	0.24	0.03	0.00	0.00	−0.01
along the grid, 0.4 cell off	1.47	0.85	0.38	0.01	0.00	0.00	+0.00
along the grid, 0.48 cell behind one	1.81	1.01	0.44	0.04	0.00	0.00	+0.01
at 3 degrees to the grid	1.17	0.41	0.23	0.04	0.01	0.00	+0.00
at 8 degrees	1.01	0.09	0.04	0.12	0.02	0.00	+0.01
at 14 degrees	0.69	0.20	0.06	0.18	0.03	0.01	+0.01
at 25 degrees	0.55	0.18	0.09	0.25	0.04	0.02	+0.02
at 45 degrees	0.32	0.12	0.09	0.26	0.11	0.08	+0.05
at 45 degrees, 0.2 cell behind a cell face	0.59	0.43	0.27	0.40	0.16	0.12	+0.08

Along the grid fitted walls lie where the model has them, and against the source's image in the cell face nearest the wall they are about as far off as walls on cell faces are against the model's wall; at an angle the error grows with it.

Fitted walls: an L-shaped room. Its plan 2.4 x 2.0 m with a 1.0 x 0.9 m notch cut from a corner, 1.5 m high, every length rounded to whole cells, on the 125 and 250 Hz bands' 7-point grids: its lowest mode (the plan's) against the plan's own problem written in the script (no flow through its walls, on squares), on squares of the grid's cell (which walls on cell faces give within 2e-6) and in the continuum (from squares of 2.5, 1.2 and 0.6 cm extrapolated, carried through the grid's time step: 63.35 and 64.41 Hz on the two grids' rooms), % off the continuum's, the room moved off the cell faces by parts of a cell along each axis:

grid	moved (cells)	fitted walls	walls on cell faces
125 Hz band's (19.8 cm)	0.311, 0.173, 0.457	+0.83	−1.42
	0.05 along each axis	−0.22	−1.42
	−0.48 along each axis	+0.83	−1.42
	0.48 along each axis	−1.20	−1.42
	0 to 0.975 in steps of 0.025, and 0.499 and 0.501	−1.42 (at 0) … +1.01 (at 0.5)	−1.42
250 Hz band's (9.9 cm)	0.311, 0.173, 0.457	+0.33	−0.50
	0.05 along each axis	−0.06	−0.50
	−0.48 along each axis	+0.35	−0.50
	0.48 along each axis	−0.44	−0.50
	0 to 0.975 in steps of 0.025, and 0.499 and 0.501	−0.51 (at 0.499) … +0.41 (at 0.5)	−0.50

Walls on cell faces put the room's concave edge on whole cells, and lie below the continuum wherever the room lies. Fitted walls lie above or below it by where the edge falls within its cell, and jump by 2.31 % on the 125 Hz band's grid and 0.89 % on the 250 Hz band's as the edge crosses the planes of the cells' centres (from 0.499 to 0.501 of a cell); just below those planes on the 250 Hz band's grid they lie 0.01 % further below than walls on cell faces. At 0.05 the cube round the concave edge keeps a thin L of air, whose link the fit scales to the L's volume; taken whole, it put the mode 3.3 % above the continuum on the 125 Hz band's grid (a one-off check).

Fitted walls that absorb. A duct 0.3 m square on 10 cm cells, its far end 0, 0.25, −0.48 and 0.48 of a cell past a cell face, reflects off an end of admittance 0.25 and 0.5 within 0.010 of (1 − β)/(1 + β) with fitted walls (0.003 on cell faces); off an end of admittance 1, which reflects nothing in the continuum, 0.044 on a cell face and 0.061, 0.002 and 0.077 at the others (0.044 at every place on cell faces; the grid's own reflection off a wall half a cell past the last cell centre is 0.02 to 0.09 at these frequencies, 40 to 200 Hz). An end of a material with a branch resonant at 100 Hz reflects within 0.003 of the grid's theory for its admittance on a cell face, and 0.012 and 0.025 off it 0.25 and 0.48 of a cell from one (0.003 on cell faces), as the fitted update's own analysis in one dimension gives (a one-off check). In the box turned by 8 degrees, its floor and ceiling of admittance 0.05 and its walls 0.02, the nine lowest modes decay at 0.992 to 0.999 of modal theory's rate (from the update's own terms, to first order), where walls on cell faces give 1.002 to 1.100; and with walls of the branch's material the branch states weigh what the walls' area says within 6e-9 (on cell faces, whose walls are the staircase's squares, 11 to 16 % more).

Fitted walls over long runs. 200,000 steps of the L-shaped room on 10 cm cells with four plates in it (one turned 31 and 57 degrees, one level, two upright with a 3 cm slit between them), all turned 22.5 and 11 degrees, and of the box turned 22.5 degrees on the 19.8 cm grid: rigid, each keeps its oscillating part within 0.999 to 1.001 of the second block of 10,000 steps, its update's coefficients are not below 0 and each cell's sum to at most 2, and its largest eigenvalue is 3.95 and 3.96, under 4; the uniform pressure's Rayleigh quotient is 2.3e-9 and 1.0e-9, so that it rings once in about 130,000 and 200,000 steps. With walls of admittance 1, or of 0.05 with a branch resonant at 60 Hz, the room falls below 4e-11 of its first block by the half and stays there. Open (its ceiling left out, in a grid with the matched layer), it falls to 9e-10 by the eighth block and then rises by about 1e-10 a block, to 2.1e-9 at the end: single precision's rounding round the matched layer, which walls on cell faces show too (a one-off check).

Fitted walls: cost. Through the solver with two threads, the least of three runs (the script's own machine, 4 cores shared):

model	grid (cell)	cells of air	the fit (s)	per step over the 7-point stencil on cell faces	the 19-point stencil's
the box, 5 s of sound	250 Hz band's (9.9 cm)	38,556	0.07	1.57	1.90
the church, closed, 0.25 s	125 Hz band's (19.8 cm)	1,228,682	0.92	1.35	1.89
the leaky classroom, open, 0.25 s	125 Hz band's (19.8 cm)	117,504	0.16	1.03	

An open model is fitted twice: on the closed grid the solver tries first and on the grid with the matched layer. In one-off checks at 125 Hz with the examples' own lengths, a run took 1.1 to 1.4 times as long as with walls on cell faces and the default stencil in seven examples, and 2.4 to 2.6 times in a small room holding a sphere of 5,120 faces, where the fit took most of the extra time (0.03 to 1.1 s on the examples, 13 to 18 s on a grid of 37 million cells). The set-up's memory peaks while the model is fitted, at 2.7 (the church) to 4.1 times (a box of 37 million cells) that with walls on cell faces.

Fitted walls: one-off checks. Beyond the script:

- Turned rooms with absorbing materials, 63 and 125 Hz bands, against the same room on cells half as large: the lecture hall turned 17 degrees within 0.43 dB (G) and 2.4 % (T20) with fitted walls; the studio turned 17 degrees within 0.40 dB, its 125 Hz T20 6.0 % and 4.7 % above at its two sources, 3 % unturned (the default grid, 8 points per wavelength at the band's top, is coarse for its T20; finer grids approach the half-cell result smoothly). Walls on cell faces on the default grid were 0.05 to 1.75 dB (G) and 4.8 to 22.6 % (T20, always shorter) off the same reference.
- A thin rigid screen behind which the level is read against the exact diffraction (Biot-Tolstoy-Medwin), 8 to 16 cells per wavelength: with its edge on a cell face, fitted walls are 0.15 dB rms off (walls on cell faces 0.73); with the edge inside a cell, 0.11 to 1.2 dB, more the higher the edge lies in its cell (on cell faces 0.26 to 0.70 dB): in the cell the edge ends in the fit leaves the screen out, so it acts as if it ended up to a cell lower and lets too much sound past.
- A listener 0.2 to 1.5 cells from a rigid wall, against the exact field: along the grid and at 8 degrees fitted walls give the level within 0.40 dB and its gradient along the wall within 0.89 dB (walls on cell faces up to 2.29 and 7.96 dB); the gradient's component normal to the wall, nearly 0 there, is up to 11.4 dB off with fitted walls and 13.8 dB on cell faces. Fitted walls' errors hardly depend on where the wall lies in its cell, and walls on cell faces can be closer where their errors cancel: by up to 0.35 dB with the wall a quarter of a cell off a cell face, and by up to 2.8 dB in the normal component at 8 degrees.
- Every indoor example checked (17 at 63, 125 and 250 Hz, every source; not the BIM office, which needs the `ifc` extra) fits without a refusal; where its mesh is closed, its air volume is the mesh's within 3e-16. Outdoor scenes are refused by design.
- Next to the matched layer a cell keeps its link to the layer whole where a grid without the layer would scale it: one more cell between the model and the layer changed the open classroom's responses by up to 7e-4 of their peak (rms), at most 1.9 times as much as with walls on cell faces.
- The fit gives the same arrays to the bit on every platform CI tests (Linux, Windows, macOS on Intel and on Apple silicon): a test compares their hashes.

On a GPU. `validation/wave_gpu.py` computes the wave bands (up to 125 Hz, 0.5 s) of the first source of three example rooms to every listener on the CPU and on the GPU, and compares the two. The tests do the same with resonant walls and with the matched layer, in runs split so that the fields swap both ways between them, with both stencils. On Mesa's lavapipe, a Vulkan driver that runs on the CPU, with 4 cores (the 19-point stencil):

room	cells	steps	CPU (s)	GPU (s)	largest difference
studio	10,350	1,016	0.2	1.5	2.8e-7
concert hall	1,223,312	1,016	2.9	25.4	6.8e-7
church	1,228,682	1,016	3.5	30.1	1.5e-6

The difference is relative to the largest pressure: the GPU's single-precision rounding, which stays at that level. Apple's GPU, in CI, rounds further from the CPU: in the test's nearly closed room with resonant walls the two differ by up to 1.1e-5 after 1,601 steps, in the slow drift such a room builds up. These times are from an earlier version of the script and include the grid's set-up on the CPU (the same on both) and, in the first room, the GPU's start-up; the script now prints the set-up on its own, the CPU's and the GPU's time steps without it, and the speed-up of the steps. A software driver is no faster than the CPU solver; the script is there to time a hardware GPU. With fitted walls (`--walls fitted`, the 7-point stencil, 1,516 steps) on SwiftShader, another Vulkan driver that runs on the CPU, the three rooms differ by 7.4e-7, 8.4e-7 and 5.9e-7, and the tests' closed and open rooms by 2.4e-7 and 1.5e-7 after 400 steps (one-off checks; the tests' bar on a software driver is 1e-5).

Real-time convolver

The plan's target: a 3 s response with 16 output channels (third-order Ambisonics), fed by one input in blocks of 128 frames at 48 kHz, without dropouts for 10 minutes. The benchmark sends 10 minutes of noise (225,000 blocks) through the convolver from Python, as the playback callback does, on one core of a 4-core x86-64 cloud machine with nothing else running, and times every block:

per block	mean	median	99 %	99.9 %	max	over the 2.667 ms budget
thread CPU time	0.127 ms	0.096 ms	0.248 ms	0.295 ms	1.175 ms	0 of 225,000
wall-clock time	0.128 ms	0.097 ms	0.250 ms	0.305 ms	3.663 ms	1 of 225,000

The response is split into 16 partitions of 128 samples, 14 of 1024 and 16 of 8192, and the work of the long partitions is spread over the blocks in between. The convolver's own work (CPU time) never took longer than a block lasts: on average 4.8 % of it, at most 44 %. In wall-clock time, one block was late (3.7 ms), while no block's own work took more than 1.2 ms: the thread was kept off the core for a moment. On an earlier run, on a machine shared with other work, 48 blocks of 225,000 (0.02 %) were late that way. On a desktop with a real-time audio thread and the usual two or more buffers at the sound card, such single late blocks are absorbed. Whether a given sound card plays 10 minutes without a dropout can only be checked on that machine: `acoustic-spaces play ir.wav --live --block 128 --seconds 600` reports the longest time a block took and every block the sound card dropped. The output matched offline convolution to 2.6×10^{-7} of its peak (single precision).

Known issues

Reviews of the code found these, but for the one about barriers, which the checks against the exact solution found, and those about barriers in wind and a leg's shadow, which their checks found. Each one was checked against the code; the numbers come from the tests or scripts named, and a number from a one-off check says so. They are open items on the roadmap. The edge fade's own are listed in Known issues of the edge fade.

Results:

- **The first milliseconds of a rendered response.** An impulse response is put together from its bands with zero-phase filters (`synthesis.combine_bands`), which spread each arrival both ways in time, and what would fall before the response's first sample is lost. For an arrival whose level differs from band to band, that changes its level in the low bands when it comes in the first milliseconds: a sound 1.5 dB lower at 125 Hz than at 63 Hz, 4.7 dB at 250 Hz, 10 dB at 500 Hz, 15 dB at 1 kHz and 20 dB from 2 kHz up gives a G 0.14 dB lower at 63 Hz 2.9 ms in than the same sound 200 ms later, and within 0.05 dB in the other bands; a sound of the same level in every band does not change (`tests/test_coherent.py`). The echograms, and the parameters `simulate` prints and writes from them, are not affected; the parameters `analyze` takes from the WAV files are.
- **The early order with vector-based scattering.** Vector-based scattering (the default) sends each reflection along the sum of its specular direction and a Lambert direction, which leans towards the wall's normal. The detector spheres count each reflection's specular share along the exact specular direction, but from where that ray met the face: the first order's share is the exact paths', and the later orders' leave the walls from other places than the exact paths and bring less energy to the listener. The early part replaces orders 1 to 3 with the exact paths, so the total depends a little on how many it takes: in a 4 x 3.5 x 3 m box (absorption 0.15, scattering 0.3), with the default `early.max_order: 3` it is 0.02 to 0.05 dB lower than taking orders 1 to 9, and with `max_order: 0` 0.04 to 0.13 dB lower, while taking the first order early or not moves it by under 0.03 dB (from 250 Hz up, four seeds). Random scattering keeps the specular share on the specular paths: within 0.02 dB either way (the room of `tests/test_scattering.py` with its seed; the test's bounds are looser, to allow for other seeds). A lobe centred on the specular direction would remove this, but changes the late part's decay, which the default was chosen to match in the BRAS rooms.
- **Pairs of paths from 1 kHz up.** In the bands above 500 Hz that add in pressure (outdoors, 1 to 8 kHz), the echogram shares each pair of paths' cross term between their two time slots, and a slot that would go below zero is set to zero (`coherent.pair_energy`). That adds energy where paths cancel; a one-off check over the nine outdoor examples, when this was found, put it at hundredths of a decibel at most, in a band's response and in the lateral figures of eight behind LF, LFC and LG. Impulse responses add the paths in pressure without it.
- **Scattered sound near a listener in bent air.** Where the rays bend in wind or a temperature gradient, scatter in plants before they reflect, or reach the listeners as secondary sources, the scattered sound of a reflection near a listener comes from the few rays that land there, not from the listener's own gather: a run with a far source over a ground that scatters has bursts of it that change with the seed.
- **Paths over two edges take no reflections.** With `second_order` a path over two edges reflects nowhere before, between or after them, so behind a thick barrier on a hard ground the engine has one of the four paths over it: 1.7 to 5.7 dB low from 500 Hz up, and off by up to 9.9 dB below (see Reflections that fade at edges). In sight just over a building's roof the path over both of its edges has no copies off the ground either. In a one-off check against the package's wave solver (8 points per wavelength), with the source and the listener 0.3 to 1.5 m above the roof of a building 4 m wide and 2 m high on a rigid ground, the early stage was within 0.9 dB at 125 Hz for a building 4 m long (three positions) and 1.9 dB high for one 40 m long (one position); at 63 Hz, where the building is also less than half a wavelength high, it was 0.5 to 6.6 dB low for the short building and 6.4 dB high for the long one (`first_order`: 3.8 dB low to 7.8 dB high at 63 Hz, 0.9 to 2.4 dB high at 125 Hz). The difference at 63 Hz is not traced further; the wave solver (`simulation.wave`) is the tool for those bands.

- **Sound that just clears two edges in a row, without paths over two edges.** Each edge the direct sound passes takes its terms off it, and with `first_order` (the default) or `direct` nothing gives back what two edges in a row take twice: where the source and the listener see each other just over both edges of a flat roof, a thick wall or two barriers, or along a wall in line with both, the level is too low. Over a top 10 m wide, the source 10 m before it and the listener 5 m past it, it is 12.6 to 20.7 dB low from 63 to 500 Hz a centimetre above the top's plane, 6.4 to 0.9 dB half a metre above it, 2.6 and 1.0 dB at 63 and 125 Hz a metre above it, and within 0.7 dB two metres above it. Behind the top with the source above its plane the level is too high, the path over the far corner taking nothing off for the near one: by 5.7 to 5.9 dB with the source a centimetre above the plane and the listener half a metre below it, 5.1 to 4.2 dB with them 0.5 m above and 2 m below, and 3.4 to 0.1 dB with both 2 m off it. `second_order` has the path over both and is within 0.09 dB in all of these (see Edge diffraction against the exact solution); its reflected copies, off a ground, are missing as in the item above.
- **More than two edges in a row near grazing incidence.** A path bends over two edges at most (`second_order`), so where sound just clears three edges or more in a row (three barriers, two flat roofs of one height, a roof with a slot across it), what their terms take off it more than twice is not given back. Just over three screens 5 m apart the level is 1.5 to 2.9 dB above that over two from 125 Hz up; a 2 cm slot across a 10 m top lowers the level by 0.6 to 1.1 dB; over three tops in a row it is 3.4 to 5.0 dB above that over one. Behind three screens with the source on the line of their tops and the listener 0.5 m below it, the path over the last top and the paths over each of the others and the last cancel: the level is at -20.8 to -54.3 dB where two screens leave -10.7 to -21.7 dB, and with the source 0.5 m above the line it is 9.1 dB (63 Hz) to 1.4 dB (8 kHz) below the two screens'. Behind two flat roofs 5 m wide and 5 m apart, the listener 0.5 m below their plane, the level up to 500 Hz is at -21.1 to -32.0 dB with the source a centimetre above the plane, at -18.4 to -15.7 dB with it 0.5 m above and at -9.9 to -4.1 dB with it 2 m above, where one roof 15 m wide, with no gap, leaves -6.2 to -8.6, -5.6 to -6.8 and -3.7 to -2.1 dB; with the source below the plane the two are the same. With `first_order` the level a centimetre above two tops in a row is at -0.4 to -1.5 dB and above three at +4.9 to +5.8 dB, where one top leaves -5.3 to -5.9 dB. There is no exact solution for these in the report. One check against a two-dimensional wave calculation turned into a point source's (on a 2 cm grid, within 0.05 dB of the exact solution over the 10 m top), up to 500 Hz, a centimetre above the plane of a flat top and a screen level with it: with the top 9 m wide and the screen 1 m past it the level is -6.3 to -6.9 dB, and `second_order` is 3.2 to 5.8 dB below that (`first_order` 0.3 to 2.5 dB below it); with the top 5 m wide and the screen 5 m past it, -7.6 to -8.4 dB, `second_order` 2.3 to 3.4 dB below (`first_order` within 1.3 dB), either way round with the rays' own search for reflections. The level also steps with the height of the screen's top at that plane, by 13 to 16 dB over 4 mm: past the 9 m top with `second_order` and every sequence of reflections tested (`method: image_sources`), it is -13.7 to -19.0 dB with the screen's top from 2 mm to a micrometre below the plane, -9.6 to -12.6 dB with it level or up to 0.5 mm above, and -0.3 to -1.9 dB from 0.55 mm above, from where the path over the screen's top reflects off the top (0.53 mm above, it does so at the top's far edge). Above 500 Hz, where the bands add in energy, the level is -10.0 to -10.2 dB with the screen's top level and -5.6 dB with it a micrometre above. With it 10 µm to 0.5 mm above the plane and the rays' own search, the two ways round are also up to 0.4 dB apart. A solver for whole rows is in the package, within 0.38 dB of wave calculations near grazing incidence in free space (see Rows of roofs and barriers against a wave calculation). An outdoor scene takes the sound over the tops of its barriers and buildings from it in the bands up to 500 Hz; the levels here are those of the paths over the edges, which stand with `simulation.early.rows: edges`, in coverage maps and walks, near the ends of a top and over a model file's objects.

- **Exactly in the plane of several tops in a row.** With the source and the listener both exactly in the plane of two flat roofs of one height, or of a roof and a barrier's top, the paths over one edge and over two do not add up as they do just above it. The level is 3.9 to 9.2 dB below that a centimetre above the plane of two tops 5 m apart, 2.5 to 6.9 dB below it over three tops, 1.4 to 5.4 dB below it with a 2 cm slot across a 10 m top and 2.8 to 5.3 dB below it over a screen and then a top; over a top and then a screen it is within 1.7 dB of it (in pressure up to 500 Hz and in the bands that add in energy). Within two millionths of the model's size below that plane, the rays that look for paths over edges, which start a millionth of it off the edges, find paths that the tops hide: the level is then 9.4 to 9.7 dB above that a centimetre below the plane of two tops, 15 dB above it for three tops, and 8.9 to 9.9 dB behind a top and a screen in either order. Within that distance above the plane they miss paths that are there: a millimetre above the tops of three screens 5 m apart, the last of which ends a centimetre beside the way, the path round that end is not found and the level at 63 Hz is 1.5 dB below that 3 mm above them. Likewise above a roof 5 m wide and a screen level with it, 5 m before the roof or past it, that ends 1 cm to 0.1 m beside the way: the term of the screen's upright end for the direct sound is missing up to 1.2 or 1.8 mm above the plane, where the level up to 500 Hz is 0.2 to 0.6 dB below that a centimetre above (0.5 to 1.1 dB with the way 1 cm outside the end).
- **Reflections level with several tops.** With the source and the listener exactly level with the tops of two screens, the level is that just above the line for the direct sound and for a first reflection that passes level with the tops too (see Edge diffraction against the exact solution). The terms that make reflections fade at edges, which have the reflections of second order and up, count each top for the whole of one that passes level with both: with a wall 2 m behind the source, which sends such reflections off the first screen and the wall, the level on the line is 4.6 to 5.7 dB above that 1 mm above it (11.5 to 13.4 dB above it before, when each top stood for all of the direct sound and of the first reflection as well).
- **A way through a corner of a flat top.** Where the straight way between the source and the listener, both in the plane of a flat top, passes exactly through a corner of the top (a box turned so that only its corner touches the way), no ray finds the direct sound and no path over the top's edges stands for it: over the box alone there is no sound at all, where a millimetre above the plane the level is at +0.6 to -3.0 dB. With a screen on the way too, its paths are all there is: where a screen ends on the way 7 m before the corner, the level is 8 to 28 dB below that a millimetre above the plane (the screen's paths stood for the direct sound there before, within 2.2 dB of it).
- **A slight bend of a barrier.** Where a barrier's top goes on in another across a bend of less than a degree in plan, neither edge fades at the joint, and the point a path touches can fall past the end of both (no path over the top) or on both (two paths). Behind a barrier with a bend of 0.95 degrees, the source 10 m before it and the listener 15 m behind it, the level on a strip beside the plane through the bend is nothing at all or 6.0 dB too high, as the bend turns: the strip is 1 mm wide with both 2 m below the top, and 4.6 cm wide with the listener 20 m below it. Bends of a degree or more fade (see Edge diffraction against the exact solution).
- **Two separate edges in each other's transition zones.** A path over two edges takes the product of the two edges' coefficients, the second's with the curvature of the first edge's wave. Where the second edge stands in the transition zone of the first and no face joins them (two screens in a row, near grazing incidence over both), that leaves too little at high frequencies: a quarter of the free field just below the line of two tops 10 m apart, where two knife edges leave 0.317 of it, 2.1 dB low from 2 kHz up, and 1.0 to 1.3 dB low just above it. As the source or the listener alone crosses the line of the tops, the level steps by 0.5 and 1.0 dB at 63 Hz. The corners of one block or wall, joined by a face, are not affected.

- **A barrier’s end in wind.** Behind a barrier the paths in the plane take the barrier’s effect, and the paths over the top past its end, as they fade out there, take it too; the paths round the end change as the open ground’s field along their way unfolded does from the straight direct sound and ground reflection in still air to the weather’s, so that at the end’s shadow boundary they jump by the open ground’s field, as the direct sound does. 1 mm either side of the boundary in the `windy` preset (a barrier 4 m high 15 m from a source 1 m up, listeners 1.5 m up) the level steps by at most 0.04 dB in octave bands 60 m down- and upwind and 0.07 dB 120 m downwind, but 120 m upwind by 0.22 and 0.20 dB at 250 and 500 Hz, where still air steps by 0.17 and 0.08 dB; in third octaves by 0.28 dB 120 m downwind (in the 50 Hz one, where still air steps by 0.30 dB). With eddies it steps more at 4 and 8 kHz, and in still air the same eddies step across the boundary too (largest step, dB, 1 mm either side; in wind at 4 and 8 kHz, the lower bands as without eddies):

eddies	wind, 60 m	wind, 120 m	still air, 60 m	still air, 120 m
weak	0.02 and 0.01	0.03 and 0.10	0.19 at 4 kHz	0.27 at 8 kHz
moderate	0.08 and 0.02	0.13 and 0.34	0.60 at 4 kHz	0.29 at 4 kHz
strong	0.11 and 0.01	0.49 and 1.08	0.95 at 4 kHz	0.81 at 2 kHz

The eddies ease the decorrelation only of two paths with the same walls and edge (`Eddies.wander`), and a path round the end shares neither with the open ground's path it stands for nor, in still air, with the direct sound: with that easing taken for every pair the step is at most 0.04 dB in still air (strong eddies, 60 m) and 0.01 dB at 4 and 8 kHz in wind. With weather zones (a zone from 30 m on with 1 m/s of wind, 60 m downwind) it steps by at most 0.04 dB up to 2 kHz but by 0.20 and 0.76 dB at 4 and 8 kHz, where the eigenrays take the mean of the weathers along their way: those shares, sampled every metre of each leg, come out 0.495 for the way round the end and 0.49 along the listener's line (0.5 exactly); sampled every millimetre, the step is 0.00 dB at 4 and 8 kHz. At a building's corner the paths round its vertical edge stop where they would touch it above the roof, the edges of a box's corner stopping as an endless edge would: past a building 6 m high (x 20 to 30 m, the source at the origin 1 m up) a listener at (60, 28) rising past that point (15.808 m up) steps 1 mm by 1.3 to 3.0 dB in still air and by 0.7 to 2.6 dB in the `windy` preset downwind, where the copies of those paths stop with them. 60 m downwind the level on the boundary is the open ground's less 6.12 to 5.83 dB from 250 Hz to 8 kHz (half the field, 6.02 dB, for a half-plane; 6.48 to 5.74 dB in still air) and 1 m into the lit side within 0.32 dB of the half-plane's Fresnel factor from 1 to 8 kHz. 60 m upwind at 4 and 8 kHz it is the open ground's less 1.78 dB and plus 4.30 dB with the default frequencies, but less 1.56 and 4.13 dB with 8 an octave in the runs: it depends on the frequencies they take. Where the end stands below the line of sight (a barrier 1 m high 30 m from the source), a listener 100 m away crossing the end's line seen from above, from the barrier's effect to the open ground's sound, steps 1 mm across it by up to 0.94 dB from 63 to 250 Hz, 4.7 dB from 500 Hz to 2 kHz and 7.7 and 12.8 dB at 4 and 8 kHz upwind; 200 m away by up to 0.57 dB from 63 to 250 Hz, 6.0 dB from 500 Hz to 2 kHz (downwind) and 1.1 dB at 4 and 8 kHz. Both are two estimates of the same field that disagree, not a missing blend. In a deep upwind shadow the barrier's own runs at 4 and 8 kHz and the open ground's shadow there (deepened from the 2 kHz band's) are 12 and 40 dB apart (a barrier 5 cm high across the whole way 100 m upwind raises those bands by as much). From 500 Hz to 2 kHz it is the barrier's side: 200 m downwind, 0.5 mm behind the end's line, the wind's effect is the open ground's own there (the barrier moved aside) less 5.8 dB at 500 Hz and plus 6.3 dB at 2 kHz, from the barrier's runs on the paths in the plane, while 0.5 mm on the open side it is within 0.26 and 0.82 dB of it and the paths round the end follow it within 0.14 dB on both sides. Where the end's top comes onto the line of sight (200 m downwind, 0.5 m either side of the end's line, the listener rising past 1 m) the paths round it of the direct sound's kind stop, and the level steps 1 mm by up to 4.3 dB from 500 Hz to 2 kHz and 3.9 dB at 8 kHz (0.16 dB in still air): the paths round the end step there as in still air (6.0 and 5.9 dB at 1 and 2 kHz) and the other paths against them, which cancels in still air; in wind the two weigh differently, by the disagreement above, and 1.8 and 4.3 dB are left at 1 and 2 kHz. In a breath of wind (0.05 m/s) the paths round the end share the open ground's own departure from still air, its equation's ground fitted from the absorption near grazing against the image sources' reflection: 60 m downwind on the boundary by -0.07 to -0.86 dB from 63 Hz to 2 kHz and +0.62 dB at 4 kHz (10 m into the lit side as much, 10 m behind up to 0.27 dB), over grass, whose impedance both share, by at most 0.21 dB.

- **A model file's object in wind.** The parabolic equation's runs behind barriers and buildings cannot hold the model file's objects, so where one stands in the way the paths over its edges keep their straight legs, and the bent direct sound and its ground reflection are there where the straight ones are: the edges' terms, made for the straight paths, do not follow the bent rays' times and levels. Walked in 10 cm steps across the shadow boundaries of a wall 4 m high 15 m from a source 8 m up in the `windy` preset, down- and upwind, the weather's effect steps across the direct sound's boundary by at most 0.13 dB in octave bands and 0.2 dB in third octaves, and across the ground reflection's by up to 0.59 dB at 250 Hz upwind and 0.8 dB at 2 kHz downwind in octave bands and 2.5 dB at 250 Hz in third octaves (a one-off check; `tests/test_barriers_in_wind.py` bounds octave bands from 63 to 500 Hz). A source or a listener on a building's roof takes the equation with the building inside, as behind barriers and buildings, where those steps are gone (see Continuity); there the narrow-angle march from the roof and the one to it differ by up to 0.22 dB at 8 kHz 42.5 m away (0.7 dB in third octaves), the wide-angle one by 0.03 dB.
- **Where the runs from a roof take over.** Nearer than 1.5 times the source's and the listener's heights above the ground added, the paths over a roof's edge keep their straight legs, which leave out the weather's effect close behind it: 12.75 m from a source 1 m over a roof 6 m high, listeners 1.5 m up (and the two swapped), the effect steps by up to 0.12 dB in 2 cm (0.10 dB at 125 Hz, where the full-wave solution's is 0.08 dB), down- and upwind, in either setting of the equation. Measured from the roof, that distance would put the runs close behind the wall, where the wide-angle equation is up to 0.26 dB off the full-wave solution at 250 Hz 6 m from the source and the runs' frequencies fall in step with the ground's fringes 11.5 to 12.4 m away (up to 2.9 dB at 8 kHz, 0.11 dB with 64 frequencies an octave).
- **A barrier's end seen from a roof.** On the lit side of the shadow boundary of a barrier's end the roof's near edge diffracts the direct sound and its ground reflection, but no path goes round the end and then over that edge, so that 1.5 m over a roof 6 m high 45 m from a source 1 m up the level steps 1 mm across the boundary by up to 2.79 dB in still air (at 250 Hz; by up to 0.05 dB with no building). In wind the paths round the end take there the lit side's runs with the building inside, the paths over the barrier's top those with the barrier too, and with the roof edge's paths stepping the weather's effect steps by up to 0.37 dB downwind and 0.57 dB upwind at 500 Hz, and by up to 1.28 dB in the 400 Hz third octave; without the roof edge's paths it steps by at most 0.02 dB (0.07 dB in third octaves; `tests/test_barriers_in_wind.py`). Diffraction to the second order leaves the still-air step (3.53 dB at 250 Hz). A path round the end and then over the roof's edge would keep its straight legs in wind as the paths are routed now (it goes round a vertical and then a horizontal edge), while the lit side's paths over that edge take the building's runs, so supplying it in still air alone would not be known to remove the step of the weather's effect: that is not measured. In other cases (one-off checks): 0.8 m over the same roof the weather's effect steps by 0.82 dB at 4 kHz (still air by 5.76 dB at 125 Hz); with a barrier 8 m high and the source on the roof by 0.69 and 1.15 dB at 250 and 500 Hz; and where a second building 4 m high stands with its corner's top 0.17 m under the line of sight from a source on a roof, by 3.3 and 4.5 dB at 1 and 2 kHz (up to 6.5 dB with the two swapped): there the direct sound's own runs step as the way to the listener takes in that building at its footprint's corner (their effect goes from 0.97 to -2.29 dB at 250 Hz), and the paths over its roof's edge past the end keep straight legs.

- **A caustic in the rays' bands.** Over the open ground in wind or a temperature gradient the bands above the parabolic equation's (4 and 8 kHz) come from the eigenrays, and where two of them merge (a caustic) each is capped at 10 dB over a straight path's on one side and both are gone on the other, with no transition (Airy) zone between: the level steps by 11.1 and 12.9 dB at 4 and 8 kHz between listeners 1 cm apart 119.86 and 119.87 m downwind in the `windy` preset (a source 2 m and a listener 1.5 m up over grass), and by 10.5 dB at both between 2.04 and 2.05 m up 250 m away in `clear_night` (a source 0.5 m up), where up to 2 kHz the level changes by at most 0.05 dB a centimetre (`tests/test_refraction.py`, a one-off check at 4 and 8 kHz). With `simulation.early.open_ground_frequencies: rays` the rays give 1 and 2 kHz too short of half the way to the shadow, and those step at the same caustics by 10.5 and 12.0 dB in the wind and by 14.0 and 7.2 dB on the clear night (a one-off check). In one-off checks against `validation/ffp.py`'s fast field program on twelve walks over the open ground (a ground of 200 kPa s/m² by Delany and Bazley's model with a scattering coefficient of 0, image sources to the first order, no diffraction; in `windy` 20 to 400 m from the source, downwind from a source 1 m up to listeners 1.5 and 3 m up, from one 3.5 m up to 1.5 and 3 m, from one 7 m up to 1.5 and 7 m and from one 2.5 m up to 2.5 m, 60° off downwind from a source 2 m up to listeners 1.5 m up, and downwind over a ground of 20000 kPa s/m² from a source 1 m up to listeners 1.5 m up; in `clear_night` listeners 0.2 to 6 m up 150 and 300 m from a source 1.5 m up and 200 m from one 0.8 m up), the level steps on eleven of them where paths appear: between listeners a centimetre apart, placed where the level changes most between listeners a metre apart (all the way up on the clear night), the largest change on each is 7.2 to 15.5 dB at 4 kHz, where the program's level changes by 0.04 dB at most (6.5 to 15.1 dB at 8 kHz). On ten of them two paths appear there together; on the walk from 1 m up to 3 m up the engine finds one of the two from 200.99 m and the other from 208.40 m, where the level steps again, by 2.5 and 3.1 dB at 4 and 8 kHz. At listeners a metre apart (a centimetre apart in height on the clear night) the 4 kHz band of the eleven is rms 0.9 to 2.1 dB from the program and at its worst 4.0 to 15.2 dB above it, within a metre of such a step on the side of the paths it adds. On the twelfth (the source 7 m and the listeners 1.5 m up), where the 4 kHz band changes by 0.49 dB at most between listeners a centimetre apart all along the walk, it is rms 0.14 dB from the program and at its worst 0.54 dB below it. On a thirteenth with `rays` (a source 3.5 m and listeners 1.5 m up in `windy`, 15 to 400 m downwind), the 1 and 2 kHz bands are rms 1.0 and 1.1 dB from the program at listeners a metre apart and step by 8.2 and 9.1 dB.
- **A wall's reflection in wind and temperature gradients.** There the paths off a façade or a barrier's face are eigenrays in every band, where the open ground's own sound is the parabolic equation's up to 2 kHz by default: such a reflection steps at its rays' caustics in every band, and up to 2 kHz its ground effect is the rays'. In one-off checks in `clear_night` with the default early settings, a source 1 m and listeners 1.5 m up over a ground of 200 kPa s/m² by Delany and Bazley's model, 5 m beside a rigid wall 40 m high along their way (from 40 m behind the source to 340 m ahead of it; scattering coefficients of 0), against `validation/ffp.py`'s fast field program, its direct sound and that of the source's image in the wall added in pressure (an endless wall): from 50 to 150 m, at listeners a metre apart, the 63 and 125 Hz bands are rms 0.67 and 0.51 dB from it and the 500 Hz band rms 4.74 dB, 10.4 dB above it at 150 m; at the caustic of the wall's rays 166.6 m away every band steps within a centimetre, by 6.1 dB at 2 kHz to 19.4 dB at 500 Hz; and from 168 to 300 m the 500 Hz band is 12.5 dB above the program on average. Without the wall the same listeners, 50 to 300 m away, are within rms 0.11 dB of the program's direct sound in every band up to 2 kHz.

- **Other listeners in a listener's direction behind barriers, in wind and temperature gradients.** Behind a barrier or a building the listeners and map points that a source's sound goes the same way to share a run of the parabolic equation under one absorbing layer, which begins above the highest of them, and part of the sound that the obstacle's top sends steeply up comes back down, by where the layer begins. In one-off checks a listener 40 m from a source 1 m up, behind a barrier 4 m high 15 m from the source in the `windy` preset, moved by 0.0013 dB with two more 80 m away, one of them 10 m up. One 15 m from a source 0.5 m up, behind a barrier 7.43 m high 4 m from the source over a rigid ground, downwind in a wind of 10 m/s at 10 m, moved by 2.0 dB at 2 kHz with another 290 m away, and by 2.1 and 2.4 dB at 2 and 4 kHz with another 60 m away, 23 m up; alone, its 2 kHz band stayed within 0.25 dB under ten of twelve layers set to begin 21 to 240 m up and was 0.4 and 2.0 dB under their mean under those beginning 36 and 43 m up, and 0.02 dB under it under a layer twice as thick beginning 43 m up. Over the open ground a listener reads the same alone as among others (see The wide-angle parabolic equation).
- **What the equation's field through the crossovers does not mend.** Between its samples (59 Hz apart in the 2 kHz band) the field is drawn linearly in the complex pressure, within h^2 / 8 times the sum of |a_i| (2 pi dtau_i)^2 for arrivals a_i dtau_i away from the field's own delay (`engine/fine.py`'s `interpolation_bound` reads it from the samples); a field that turns by half a period or more from one sample to the next is not resolved. 300 m downwind in `windy` the equation's dip to -10.1 dB at 2225 Hz reads -9.9 dB; 1 km downwind the estimate is 0.3 to 0.55 of the band's level in its bands from 500 Hz up, and the equation's dip to -27 dB at 2090 Hz is drawn at -50 dB 10 Hz below it (a one-off check). In the weather presets the rays' band above the top crossover (2.4 to 3.4 kHz) is 5 to 22 dB off the equation inside it, 100 and 300 m downwind in `windy` and on the clear night, while the equation's bands follow it within 0.8 dB. Against the bands' own samples, whose two errors there partly cancel, the response in the crossover comes closer to the equation in `windy` (rms 7.2 to 3.3 dB 100 m away, 4.3 to 3.1 dB 300 m away, though 3.6 dB further at 3010 Hz) and goes further from it on the clear night (rms 1.4 to 1.8 and 1.2 to 1.7 dB, up to 2.4 dB further at 3075 Hz 100 m away; a one-off check). So it does in case 2 with the source 1.5 m and the listener 1 m up, 300 m away, where the rays' band is 2.8 dB low at 3.45 kHz: up to 0.42 dB further from the program at 2.8 to 2.95 kHz, though closer over 2 to 3.45 kHz (worst 4.0 to 3.0 dB, rms 1.8 to 1.6; a one-off check). In the shadow the bands keep their own samples, since the bands above the equation, deepened from the octave below, are not continuous with its field: 18 dB (upwind in `windy` 100 m away) to 36 dB (`sunny_day` 300 m away) louder than it at 3.4 kHz. In case 3, 300 m away, where they are within 0.3 dB of the equation at 3.4 and 3.9 kHz, the response in the crossover is 8.6 dB below the program at 2.95 kHz. The paths round a barrier's end or a building's corner take their way's runs at every frequency too, with each path in still air drawn between its own frequencies (within 0.18 dB and 0.9 degrees of the same path at 24 frequencies an octave from 0.7 to 4.5 kHz), and 60 and 120 m away, downwind in `windy` and in case 2 over a Delany-Bazley ground of 200 kPa s/m², their rendered third octaves through the crossovers are within 0.03 dB of runs 27 times as dense (over an absorption of 0.02, 0.34 dB at 800 Hz 2.6 m up, as far as the way's own field is off its dense run there; with `open_ground_frequencies: rays` 0.18 dB at 400 Hz, where the band energies over 8 frequencies an octave leave the listener's own field 0.10 dB off). Below an end's line of sight, where the straight paths along the way all but cancel, they keep the weather's effect at the early paths' frequencies (100 m downwind in `windy`, within 0.01 dB of the dense runs in the 2 and 2.5 kHz third octaves but up to 1.1 dB off them in the crossovers); the paths over a roof's edge and behind barriers still take one ratio of the equation's band energies a band through them (a one-off check).

- **A ground given by absorption coefficients in wind.** The parabolic equation (the bands up to 2 kHz in wind and temperature gradients) takes such a ground as a locally reacting surface of the real admittance fitted to its absorption (`engine/impedance.py`'s `fit_admittance`: 0.003 for 0.02, 0.051 for 0.3), whose reflection goes to -1 at grazing incidence and adds a ground wave; still air's image sources reflect off it with the square root of (1 - its absorption) at every angle and that admittance's plane-wave phase, as off a floor. So a breath of wind (0.05 m/s) moves the level, in `tests/test_barriers_in_wind.py`'s scene without its barrier (scattering 0, a listener 1.5 m up 60 and 120 m away), from 63 Hz to 2 kHz by -0.15 to -1.08 dB from a source 1 m up over an absorption of 0.02 (-0.11 to -0.81 dB from 5 m up), and over 0.3 by up to +18 dB at 63 Hz from 1 m up (-4.8 to +1.8 dB from 5 m up). Each against the fast field program over a locally reacting ground of that admittance in its own air (still, and the scene's wind), the equation is within 0.05 dB in every band of these cases, and the image sources are 0.10 to 0.91 dB above it over 0.02 and up to 18 dB off it over 0.3 (63 Hz, 60 m, from 1 m up); the bending itself, the program's breath against its still air, is at most 0.11 dB of these moves but for 0.31 dB at 2 kHz 60 m from the source 1 m up and 0.38 dB at 1 kHz 120 m from 5 m up. The rays' bands, whose reflections take the image sources' rule, move with the bending alone: +0.62 dB at 4 kHz 60 m from the source 1 m up, where the ground reflection comes 4.0 µs later against the direct sound than in still air and 0.16 dB louder, by the first interference dip (3.4 kHz), and +2.2 dB at 8 kHz 120 m away (a one-off check). The paths round a barrier's end follow the equation through the crossover too: on an end's shadow boundary 60 m away (the source 1 m up, absorption 0.02), the 3.15 kHz third octave of those paths moves by -0.56 dB at 0.01 m/s and -1.92 dB at 0.05 m/s against still air, the open ground's by -0.57 and -1.93 dB, and the fast field program's, in the scene's own wind and over that admittance, by -0.36 and -1.78 dB (its still air 0.27 dB below the image sources'). Outdoors give the ground an impedance (The ground and the sky).
- **The eddies downwind.** The eddies' coherence factor was fitted in still air and in the shadow (see Turbulence); downwind it is checked only here. In `windy` with the `strong` eddies, over grass with a scattering coefficient of 0, a source 5 m and a listener 1.5 m up, the eddies raise the 1 kHz octave band by 0.33, 0.60 and -0.34 dB 100, 200 and 300 m away, where the engine's narrow-angle parabolic equation marched at the band's 24 frequencies through 224 fields of these eddies (drawn as by `validation/turbulence.py`, C_n^2 the same at every height) raises it by 0.62 ± 0.24, 2.39 ± 0.34 and 0.67 ± 0.34 dB (95 % confidence): 0.3 to 1.8 dB apart (a one-off check). Since the eddies add the rays' energy into the equation's bands, a step of the rays' at a caustic comes through: in the same case, between 335.9 and 336.0 m away, the 125 Hz to 2 kHz bands step up by 1.8 to 8.1 dB with the `strong` eddies and 0.5 to 6.2 dB with the `moderate`, where without eddies they move by at most 0.008 dB.
- **The eddies in a band's echogram.** Up to 500 Hz, where the early paths' energy over time is read through each band's filter, each two paths' product still takes their coherence factor as its mean over the band's frequencies, where the band's level takes it at each frequency. In still air with the `strong` eddies, a listener 1.5 m up over grass with a scattering coefficient of 0, the 500 Hz octave band's echogram so sums to 0.02 dB more 50 m from a source 5 m up, and 0.06 and 0.12 dB more 100 and 200 m from one 10 m up, than with the coherence taken at each frequency (0.005 dB over asphalt at 100 m; within 0.05 dB from 63 to 250 Hz, and within 0.01 dB in third octaves at 100 m; a one-off check).

- **A ground's scattering coefficient in wind.** The parabolic equation's bands over the open ground (63 Hz to 2 kHz in wind or a temperature gradient) keep a ground's whole reflection, while the rays and still air's image sources keep sqrt(1 - s) of it at each bounce, s the ground's scattering coefficient. Over a ground with a scattering coefficient of 0.1 this steps the level by 0.075 and 0.122 dB more at 1 and 2 kHz over grass (0.196 and 0.151 dB over asphalt) where the rays take over with `open_ground_frequencies: rays` (1 m/s of wind upwind, a source 10 m and a listener 1.5 m up, between 90.8 and 90.9 m away), takes 0.11 to 0.14 dB from the 4 and 8 kHz bands but nothing from the 2 kHz band beside them (the same weather, 90.4 m away), and changes by up to 0.23 dB how far a breath of wind is from still air (0.05 m/s, a source 2 m and a listener 1.5 m up, 50 m away, 63 Hz to 2 kHz). With a scattering coefficient of 0 only the models' own differences remain: up to 0.08 dB at the hand-over, and between still air and the breath up to 0.06 dB from 63 to 500 Hz and 0.13 dB over grass and 0.19 dB over asphalt at 1 and 2 kHz (a one-off check; `tests/test_refraction.py` holds the equation's bands the same with 0 and 0.1).
- **The late part of a band read through its filter outdoors.** Between the equation's samples h apart (11, 22 and 44 Hz in the 125, 250 and 500 Hz octave bands) a band's spectrum is drawn linearly, which weighs the response about the field's delay by sinc^2(t h) and repeats it faintly every 1/h: what comes 1/(2h) later is 7.8 dB down, later still more, and images come in. Where C50 and C80 are 50 to 88 dB, the late part that far below the early sound, they so read up to 8.1 dB off runs of the equation 27 and 81 times as dense, read the same way, 100 m downwind in case 2 and 200 m downwind in `windy` (the bands' own samples up to 13.6 dB off); closer than those in every case but C80 at 500 Hz, 6.5 to 7.5 and 2.3 to 4.8 dB off against 4.6 to 5.5 and 0.3 to 2.2 dB, where the two dense runs differ by 0.9 and 2.6 dB. The dense run's own field, drawn on a 0.12 Hz grid and read through the same octave filters outside the response's synthesis, gives its C50 and C80 at 250 and 500 Hz within 0.1 dB in case 2 and 0.3 dB in `windy`. What cuts the late part differs by the case: in case 2 mostly the 500 Hz band's own samples (those alone at their own spacing move its C80 from 82.0 to 91.3 dB, the bands from 1 to 3.4 kHz alone to 83.8 dB), in `windy` the bands from 1 to 3.4 kHz, which the 500 Hz filter reads through its skirts (alone they move it from 73.1 to 84.6 dB). Denser samples close it only slowly: C80 at 500 Hz in case 2 reads 88.5, 95.2, 87.3, 81.9 and 81.0 dB with 1, 3, 9, 27 and 81 times as many from 63 Hz to 3.4 kHz (a one-off check).
- **Coverage maps behind barriers** read the high bands higher than a listener at the same point: 130 m behind two barriers over grass by 0.6, 1.3, 3.8 and 13.5 dB at 1, 2, 4 and 8 kHz, in still air as in 8 m/s of wind, where the weather's effect is the same in the map and at the point (a one-off check).

- **Near a shadow boundary, in the bands that add in energy.** There the term of the edge's incident wave is part of the direct sound's wave, and the terms of the reflections off the edge's faces, which arrive with it, are part of the edge's diffracted wave, which adds to it in energy: their cross term is lost. On the shadow boundary of a right-angled corner the level is 1.73, 1.32 and 0.98 dB low at 250, 500 and 1000 Hz and 0.72, 0.52 and 0.38 dB at 2, 4 and 8 kHz; behind a blunter edge more, 2.22, 1.79 and 1.38 dB at 250, 500 and 1000 Hz and 0.56 dB at 8 kHz behind a wedge open over 225°; behind a thin screen less, 0.69, 0.50 and 0.36 dB at 250, 500 and 1000 Hz; 10.5° into the shadow, 1.05 dB at 250 Hz behind the corner, 1.12 dB behind the wedge and 0.45 dB behind the screen (see Edge diffraction). These bands add in energy indoors from 250 Hz up by default; in pressure, as the 63 and 125 Hz bands and outdoors every band do by default, the level is within 0.08 dB. Keeping the cross term, weighted by how coherent the terms are over the band, would remove it. Likewise where the source or the listener crosses the plane of a top that the other sees from above: the path over both of its edges continues the path over one below the plane and gives back what both edges take above it, and its pieces add up differently in energy on the two sides. Half a metre above a 10 m top the level steps by up to 1.1 dB as the other of the two crosses the top's plane (by 1.5 dB at a gable roof's slope in a one-off check), and by 0.01 dB in pressure. Over a top 1 m wide absorbing 0.5, the level 0.3 m above its plane was 0.56 dB above the rigid top's at 4 kHz (a one-off check).
- **Surfaces a hair off the wave solver's cell centres.** The wave solver takes a surface within 1e-10 cells of a plane of its cell centres to lie on it (see Simulation). Close to that distance, but not on it, cells and links can still be placed wrong (one-off checks on prisms, a tetrahedron, a wedge, an octahedron, a box and a screen in a 4 m room, 0.1 m cells, both stencils). A face 1e-10 cells off a plane of centres, to within the rounding of its offsets (about 1e-14 cells in that room; the rounding grows in proportion to the distance from the grid's origin), can put the cells of that plane under it on the wrong side: with one slanted face of a tetrahedron so placed, the 174 centres of that plane inside it were air, and listeners 1.3 and 3.4 cells from the face read 0.47 and 0.50 of the response off. Where every face of a closed object lies so, the object can be read as air throughout (those runs checked, 4000 steps, stayed bounded). Where a flat side made of two triangles ends on a plane of centres, a centre on the side's end edge from 1e-10 to 1e-10 / (2 sin a) cells past that plane, a being the angle between the side's diagonal and its edges that leave the plane (tan a is the side's width over its length), can be held by one triangle and not by the other (seen on prisms along x and along y, on none along z): for a long, narrow side about length / (2 width) x 1e-10 cells, 1.8e-9 cells for a side 1 cell wide and 36 long. A few boundary cells and links then move, no air cell, and a listener beside it read up to 0.16 of the response off. Where a face at 45 degrees to the axes lies 7e-11 to 1e-10 cells off a plane of centres, a link that meets it across its diagonal is cut (1 boundary cell, 2 to 6 links). A box 6e-11 to 3e-9 cells past planes halfway between centres has 42 of its 46662 19-point links cut by its edges (0.026 of the response). A screen whose plane and rims round to different sides of 1e-10 cells gives up to 0.11 of the response on the 19-point grid. Past 1.3 million cells from the grid's origin, rounding moves some ties out of reach (16 of 32 tied links of one scene). With fitted walls (`simulation.wave.walls: fitted`) none of these arise: the fit decides every side exactly. Its own sudden changes are the next item.

- **Sudden changes with fitted walls.** With `simulation.wave.walls: fitted`, a cell is stepped where its centre lies in air, and a piece of air that holds no cell centre is stepped with the neighbouring cell it shares the most open area with. A face that crosses a plane of the cells' centres takes centres from air to solid or back, and the response changes suddenly there: the L-shaped room's lowest mode by 2.3 % on the 125 Hz band's grid and 0.9 % on the 250 Hz band's (`validation/wave_walls.py`), and the response of solids whose faces pass through cell centres, moved by 1e-7 of a cell, by up to 0.34 of its peak (one-off checks, the solver's own pulse, which reaches the top of the grid's band). Where two cells share a centreless piece alike to within 1e-12 of a cell face, the one with the lower index takes it, and a model moved through that choice changes the response suddenly too: by up to 0.25 of its peak in a one-off check, a solid with faces at 45 degrees moved by less than 1e-12 of a cell from where they lie alike against the grid. Any rule that gives each piece of air to one cell changes suddenly where its choice changes; sharing such pieces among their neighbours would remove it. Where a solid's convex edge whose faces lie a hair inside the grid's planes meets a face off them (a corner of a box), the response changes by up to 0.05 of its peak (0.07 dB in a band) as those faces leave the planes, where along the rest of the edge it changes continuously (a one-off check): a rule for the depth of air behind each such link would remove it. A closed chamber thinner than a cell, joined to a cell through a hole much smaller than a cell face, nearly closes that cell's link on the other side, as if the chamber were a sliver of the cell's own; and the rule that scales that link starts where a sliver's face is larger than 1e-9 of a cell face, with a step there (8e-7 of the link's weight in a test's room).

Models:

- **Collada polygons with holes** (`<ph>`) are read with their holes filled.

Files and runs:

- **No limit on time.** The numbers of rays have no upper bound: a mistyped count runs as long as it asks for. (The memory of a run's late part is estimated before it starts, and a run whose late part would need more than the computer has is refused. The early stage's is not estimated, and in a room of many facets with edge diffraction it is most of a run's: BRAS CR3 peaks at 10.0 GiB in its edge fade (11.8 GiB in third-octave bands), which in third-octave bands starts each listener from less than 0.9 GiB; see Memory.)

Not validated yet

- **More measured rooms.** Four rooms have been compared with measurements (see Measured rooms): their ISO 3382-1 parameters and G from seat to seat, and in three of them IACC with a dummy head. G's absolute level has not, as BRAS's calibration leaves the measured level uncertain by about 3 dB, nor the lateral parameters (LF, LFC, LG), stage support and directional decay, which BRAS did not measure, nor coupled rooms whose model joins them as the real ones are joined: in two-room diffuse theory BRAS's two rooms are joined by about 1 m² more than their model from 250 Hz to 2 kHz, with the door ajar (10 to 16 times the model's slit) and open alike.
- **Wave effects.** Above the wave solver's bands the simulation is geometric. Edge diffraction is modelled (UTD, for the direct sound and first-order reflections, and the direct sound over two edges; the fade of every other specular path at edges, which leaves some steps, see Known issues of the edge fade) and room modes by the wave solver, both checked against theory but not against measurements in rooms. Without the wave solver, results below the Schroeder frequency of a room (about 2000 √(T/V) Hz) are a statistical average, not the response of one point.
- **Source directivity** was not part of the cross-check (both programs ran omnidirectional sources). The tests check that every pattern keeps the sound power, that SOFA and CSV data come out in the source's frame, and that the direct sound, early paths, late rays and particles all follow the same weights.
- **Sound through walls** is checked against diffuse-field theory, not against measurements. A wall is one sound reduction index per band for sound from all directions: sound passes straight on in its direction, with no flanking paths and no dependence on the angle it arrives at, and the wave solver does not model such walls: with `simulation.wave.enabled`, a model with one stops with an error (openings are fine).

- **Outdoor propagation** is checked against geometry, closed forms, the exact solution for the reflection off a ground and one set of measurements, a wind turbine up to 845 m away in light to strong winds. A terrain is checked against the exact solution for a wedge, the flat ground, the image source in a slope and the thin screen's closed forms, not against measurements over hills (see Terrain), and a rounded crest, which has no exact solution, is an engineering model; wind and temperature gradients cannot be combined with a terrain yet (see Outdoor scenes), and strong ones, where they matter most, are compared with a fast field program on the 1995 benchmark cases (see Outdoor sound in wind), not with measurements; the benchmark's gradients that go on to a kilometre up bend sound through angles the engine's default, narrow-angle parabolic equation does not carry, so beyond 1 to 2 km its pure tones stray there, though the bands it reports hold to 2 km, and the wide-angle one carries them (see The wide-angle parabolic equation). Weather that changes over the scene is checked against a parabolic equation marched through the zones, not against measurements, and a zone's border is a step (see Weather zones). Barriers and buildings in wind are checked against a full-wave solution and a parabolic equation started from the exact field, both in the vertical plane with the wind blowing through the barrier, not against measurements, wind-tunnel data or Nord2000's screen cases (see Barriers and buildings in wind). Levels over a long term are checked against the formulas they come from and against ISO 9613-2's meteorological correction, not against long-term measurements, and are only as good as the weather classes and shares given to them (see Levels over a long term). Moving sources are checked against the exact field of a quasi-static source over a rigid ground, not against measured pass-bys, and leave out the convective amplification of a moving source (see Moving sources). The road vehicles' sound power is CNOSSOS-EU's table as published, and the siren's level and directivity one measured siren's, checked against the rules for sirens, not against measured pass-bys (see Sirens and road vehicles). The trains' sound power is Schall 03's as published: a pass in the engine is checked against Schall 03's own sum, exact in a free field and 3.0 to 4.1 dB(A) under it over the railway example's ground, which the two reckon otherwise, not against measured pass-bys (see Trains). The road works' machines are rows of Defra's database for construction sites as published, checked against the rows' own levels and the EU's limits, not against a measured site; the reversing alarm's level is its rule's (see Road works). A helicopter's tones are checked against theory, its broadband noise against Pegg's worked example, and the whole against one certified level, which it is 3.4 dB over (see Helicopter rotors); not against measured spectra or directivity. Vegetation is checked against a particle simulation of the same model, not against measurements through trees; its foliage takes ISO 9613-2's rates, which the standard gives for depths of 20 to 200 m, or leaves as thin rigid discs, whose cross-section between its two closed-form limits is bridged, not exact, and which are compared with the standard's table, not with measurements of leaves. Turbulence is checked against a parabolic equation through random eddies in two dimensions, with the source and the listener near the ground, not against measurements; how much the sound flickers over time is not checked against either. Between two moments of the eddies, a tone played through them, standing and along a walk with Doppler, is checked to have the response whose paths are delayed half-way (in the tests, to within 1 % of the tone's amplitude). The wave solver in wind and temperature is checked against the parabolic equation over open ground up to 200 m, not against measurements, nor round buildings; its moving air against the exact field in a uniform wind, its wind round a dome against an ideal fluid's and a façade's echo against ray theory through that wind, not against a measured wind round a building, whose wake it leaves out (see Moving air). The level relative to free field and a barrier's insertion loss are checked against exact solutions, not against measurements; the path reflected both before and after a barrier is checked over a rigid ground only, and the sound that goes over a barrier, down to the ground and over it again is left out (see Outdoor results).

- **Spatial outputs** are checked against closed forms and energy balances, not by listening tests.
- **Speed on a hardware GPU.** CI checks the wave solver's and the late stage's GPU paths on software and virtual GPUs (Mesa's software Vulkan driver on Linux, Direct3D 12's WARP on Windows and the virtual machines' Metal GPUs on macOS), which shows that they agree with the CPU, not how fast they are; `validation/wave_gpu.py`, `validation/late_gpu.py` and `validation/outdoor_gpu.py` time them on your own GPU.

Reproducing

```
python -m pip install pyroomacoustics matplotlib
python validation/pyroomacoustics_crosscheck.py --json results.json --plots docs/validation
python validation/convolver_benchmark.py
python validation/wave_solver.py --plot docs/images/wave.png
python validation/wave_walls.py --json wave_walls.json   # 41 to 51 min; --walls picks the
kinds
python validation/edge_diffraction.py --json diffraction.json
python validation/edge_fade.py --json edge_fade.json
python validation/walks.py --json walks.json   # --walks gallery archway street
python validation/ground_reflection.py --wave --parabolic --extended --json ground.json
python validation/outdoor_propagation.py --json outdoor.json --plots docs/validation
python validation/outdoor_propagation.py --turbulence moderate
python validation/outdoor_propagation.py --still   # the same cases in still air
python validation/turbulence.py --json turbulence.json
python validation/turbulence.py --profile --json turbulence_profile.json
python validation/vegetation.py --json vegetation.json
python validation/leaves.py --json leaves.json
python validation/wave_outdoor.py --json wave_outdoor.json
python validation/wave_wind.py --json wave_wind.json   # --skip-profiles: no parabolic
equation
python validation/insertion_loss.py --json insertion_loss.json
python validation/terrain.py --json terrain.json
python validation/ffp.py --json ffp.json
python validation/curved_rays.py --json curved_rays.json   # about 30 min
python validation/benchmark_1995.py --json benchmark.json   # --far: whole scenes to 10 km
python validation/open_ground_frequencies.py --json open_ground.json   # --cache DIR --part
K/N to share it
python validation/steep_sound.py --json steep_sound.json   # about 2.3 h; --only steps: the
range step's check
python validation/weather_zones.py --json weather_zones.json   # --only pe or mean
python validation/barrier_wind.py --json barrier_wind.json   # about 4 h; --quick: fewer
bands
python validation/rows.py --engine --json rows.json   # --references: the wave calculation
again
python validation/long_term.py --json long_term.json
python validation/doppler.py --json doppler.json
python validation/rotorcraft.py --json rotorcraft.json
python validation/urban_sources.py --json urban_sources.json
python validation/rail_sources.py --json rail_sources.json
python validation/plant_sources.py --json plant_sources.json
python validation/doc29_reference.py --workbook doc29-volume3-part1-workbook.xlsx
python validation/aircraft_noise.py --json aircraft_noise.json
python validation/fta_vibration.py
python validation/city_speed.py --json city_speed.json   # --sizes 10 20 30 40 60
python validation/wave_gpu.py   # needs the gpu extra and a GPU driver
python validation/late_gpu.py   # the late stage on each device, and its parameters
python validation/outdoor_gpu.py   # the outdoor examples on each device: levels and centre
times
python validation/rays_gpu.py   # the ray searches alone, and whether both find the same
faces
python validation/late_memory.py   # the late part's memory against its estimate (Linux)
python validation/bras_validation.py --json bras.json --plots docs/validation
python validation/bras_validation.py --materials initial --plots docs/validation
python validation/bras_validation.py --bands third_octave --json bras_third.json
python validation/bras_validation.py --materials initial --calibrate LS1 --plots
docs/validation
python validation/bras_binaural.py path/to/BRAS --json binaural.json   # FABIAN's responses
from BRAS
python validation/bras_iacc_parts.py path/to/BRAS CR3 LS7   # one loudspeaker's response in
parts
python validation/coupled_rooms.py --results bras.json   # CR1 against two-room diffuse
theory
python validation/coupled_rooms.py --scene CR1_DoorAngle3 --results bras.json
```

```
python validation/bras_room_energy.py --results bras.json   # what scattering does, rooms'
own decay
```

On 4 cores of a cloud machine without a GPU, each run alone on the machine (30 and 31 October 2026), the cross-check takes about 10 minutes, the convolver benchmark half a minute, the wave solver's checks 3.0 minutes (both stencils, 11 November), the edge diffraction's 40 seconds, the fade at edges' 53 minutes, the walks' 14, the ground's 5.5 (51 s of them the wave solver's, 100 s the parabolic equation's), the outdoor measurements' 3.1 (3.0 with turbulence, 20 seconds in still air), the turbulence's 3.9 (3.8 with `--profile`), the vegetation's 5.2, the leaves' 2.6, the wave solver's outdoors 12.6 and the moving air's 12.3 (both 10 November), the insertion loss's 1.8, the terrain's 41 seconds (2 November), the fast field program's 5.7, the benchmark's 55 (much longer with `--far`), the weather zones' 7.2 (46 s of them the parabolic equation's, 6.4 minutes the mean profile's), the long term's 16 seconds, the moving sources' 5.6 minutes, the rotors' 3.3, the sirens' 4 seconds, the cities' 22 (1.3 minutes with `--sizes 10 20 30 40 60`), each BRAS run of the coupled rooms, the seminar room or the auditorium 2 to 4 minutes (4 to 9 calibrated, 5 to 11 in third-octave bands) and of the hall about 1.5 hours (5.2 calibrated, 3.4 in third-octave bands), the binaural comparison 10 minutes in the seminar room, 6 in the auditorium and an hour in the hall, a loudspeaker of the hall in parts about 10 minutes, the coupled rooms against theory under a second, and the rooms' own decay 5.8 minutes. Since then the open ground's parabolic equation in wind runs up to 2 kHz, its bands carry its field through the crossovers and over the open ground each listener's run marches under the absorbing layer of a run for it alone: on 16 November, on another machine of the kind, the outdoor measurements took 28 minutes (the cases already downloaded, without the plots), the long term 1.6 and the benchmark 93 (the sum of its parts, each timed alone); the others listed here do not reach that part of the engine. Since 11 November the ray tracer and the wave solver do less work on the CPU for the same results: the seminar room's BRAS run takes 10 to 12 % less, and of the others only the wave solver's checks were timed again. The cross-check's numbers come from 4.0.1 with seed 1; this version's own are within 0.012 % of them (EDT, whose line fit now starts at the onset in every response). pyroomacoustics draws its rays afresh on every run, so the script averages five runs of it (`--pra-runs`); the means still move by a few percent from one call to the next, and the differences in the tables with them. Other seeds move this package's numbers by a few percent.

API reference

The names most scripts need are also importable from the package itself: `acoustic_spaces.Room`, `Scene`, `Source`, `Listener`, `Material` and `simulate`.

Scenes and rooms

Scene model: what a simulation needs to know about a room.

```
class acoustic_spaces.scene.Air(*, temperature_c: Annotated[float, Ge(ge=-20.0),
    Le(le=50.0)] = 20.0, humidity_percent: Annotated[float, Ge(ge=0.0), Le(le=100.0)] =
    50.0, pressure_kpa: Annotated[float, Gt(gt=0)] = 101.325)
```

temperature_c: float

humidity_percent: float

pressure_kpa: float

```
class acoustic_spaces.scene.ArrayElement(*, position: tuple[float, float, float],
    orientation: ~acoustic_spaces.scene.model.Orientation = <factory>, gain_db:
    ~typing.Annotated[float, ~annotated_types.Ge(ge=-100.0),
    ~annotated_types.Le(le=60.0)] = 0.0, delay_ms: ~typing.Annotated[float,
    ~annotated_types.Ge(ge=0.0), ~annotated_types.Le(le=1000.0)] = 0.0, invert: bool =
    False)
```

One loudspeaker or driver of a coherent array, in the array's (the source's) frame: x forward, y left, z up.

position: Vector3

orientation: Orientation

gain_db: float

delay_ms: float

invert: bool

```
class acoustic_spaces.scene.Atmosphere(*, preset: str | None = None, wind: Wind | None =
    None, temperature_gradient_k_m: Annotated[float, Ge(ge=-0.2), Le(le=0.2)] = 0.0,
    temperature_log_k: Annotated[float, Ge(ge=-5.0), Le(le=5.0)] = 0.0, roughness_m:
    Annotated[float, Le(le=2.0), Gt(gt=0)] = 0.05, table: list[tuple[float, float,
    float]] | None = None, turbulence: Turbulence | None = None, zones:
    list[WeatherZone] | None = None)
```

How the speed of sound changes with height outdoors, which bends sound: the temperature (warmer air carries sound faster) and the wind (sound goes faster with it). Both vary with height only. The scene's air temperature is the temperature at the ground. Heights are in metres above the ground. Eddies of both (`turbulence`) make the sound flicker.

preset: str | None

wind: Wind | None

temperature_gradient_k_m: float

temperature_log_k: float

roughness_m: PositiveFloat

table: list[tuple[float, float, float]] | None

turbulence: Turbulence | None

zones: list[WeatherZone] | None

zone_names() → list[str]

The weather zones' names, `weather_<n>` for those without one.

weathers() → list[Atmosphere]

The atmosphere's own weather (first) and each zone's, as an atmosphere each, without zones or eddies: a zone's preset gives it that weather whole, and its other fields replace the atmosphere's (see `WeatherZone`).

under(given: WeatherClass) → Atmosphere

This atmosphere in a weather class: the class's weather over its own (see `_WeatherOverride`), the class's eddies or its own, and its zones.

ground_temperatures(ground_c: float) → list[float]

The temperature at the ground (°C) under the atmosphere's own weather (`ground_c`, the scene's) and in each zone, in `weathers()` order.

to_wind() → ndarray

(2,) the unit vector the wind blows along.

temperature(z, ground_c: float) → ndarray

The temperature (°C) at heights `z` (m) above the ground, the ground's being `ground_c` (a table gives its own).

wind_speed(z) → ndarray

The wind's speed (m/s) at heights `z` (m) above the ground.

sound_speed(z, ground_c: float, towards=None) → ndarray

The speed of sound (m/s) at heights `z` (m) above the ground: in still air, or the effective speed along the horizontal direction `towards` (2,), the wind's component along it added.

property still: bool

Whether sound goes in straight lines: no wind and one temperature, over the whole scene (the zones too).

```
class acoustic_spaces.scene.ArraySpec(*, element: Literal['omni', 'subcardioid',
    'cardioid', 'supercardioid', 'hypercardioid', 'figure8'] | Literal['lambert'] |
    DirectivitySpec = 'omni', elements: list[ArrayElement] | None = None, count:
    Annotated[int | None, Ge(ge=2), Le(le=256)] = None, spacing_m: Annotated[float,
    Gt(gt=0)] | None = None, steer_deg: Annotated[float, Ge(ge=-90.0), Le(le=90.0)] =
    0.0)
```

A coherent array: elements fed the same signal, each with its own gain, delay and polarity, whose sound adds up with its phase. Either `elements` one by one, or a straight column of `count` elements `spacing_m` apart along the up axis, centred on the source's position.

element: PatternName | Literal['lambert'] | DirectivitySpec

elements: list[ArrayElement] | None

count: int | None

spacing_m: PositiveFloat | None

steer_deg: float

layout() → list[ArrayElement]

The elements, a column's laid out.

```
class acoustic_spaces.scene.Barrier(*, name: str | None = None, path:
    Annotated[list[tuple[float, float]], MinLen(min_length=2)], height: Annotated[float,
    Gt(gt=0)])
```

A thin screen standing on the ground along a path (surfaces `<name>` and `barriers`).

name: str | None

path: list[tuple[float, float]]

height: PositiveFloat

```
class acoustic_spaces.scene.BlumleinPair(*, pair: Literal['blumlein'])
```

Two figure-eights in one place, aimed 45 degrees to each side.

pair: Literal['blumlein']

```
class acoustic_spaces.scene.Building(*, name: str | None = None, footprint:
    Annotated[list[tuple[float, float]], MinLen(min_length=3)], height: Annotated[float,
    Gt(gt=0)])
```

A building standing on the ground: its footprint raised to a height (surfaces `<name>_wall_1`…, `<name>_walls`, `<name>_roof`, and `walls`, `roofs` and `buildings` for all of them).

name: str | None

footprint: list[tuple[float, float]]

height: PositiveFloat

```
class acoustic_spaces.scene.BuildingFile(*, file: Path, height: str | Annotated[float,
    Gt(gt=0)] = 'height', default_height: Annotated[float, Gt(gt=0)] | None = None,
    name: str = 'name', without: list[str] | None = None)
```

Buildings from a map: a GeoJSON file whose `Polygon` and `MultiPolygon` features are footprints (see `acoustic_spaces.geometry.terrain`), each raised to the height a property gives. Its coordinates are in the geometry's units on a plane (not longitude and latitude), and `ground.map_origin` is subtracted from them.

file: `Path`

height: `str | PositiveFloat`

default_height: `PositiveFloat | None`

name: `str`

without: `list[str] | None`

class acoustic_spaces.scene.**Coverage**(*, surfaces: list[str] | None = None, height: Annotated[float, Gt(gt=0)] = 1.2, spacing: Annotated[float, Gt(gt=0)] = 1.0, rays: Annotated[int | None, Ge(ge=1)] = None, noise_db: list[float] | float | None = None, spread: bool = False)

A coverage map: room parameters on a grid over the audience area (see `acoustic_spaces.engine.coverage`).

surfaces: `list[str] | None`

height: `float`

spacing: `float`

rays: `int | None`

noise_db: `list[float] | float | None`

spread: `bool`

class acoustic_spaces.scene.**DirectivitySpec**(*, pattern: list[Literal['omni', 'subcardioid', 'cardioid', 'supercardioid', 'hypercardioid', 'figure8'] | Annotated[float, FieldInfo(annotation=NoneType, required=True, metadata=[Ge(ge=0.0), Le(le=1.0)])]] | None = None, file: Path | None = None, array: **ArraySpec** | None = None, measurement: Annotated[int | None, Ge(ge=0)] = None, near_field: bool | None = None)

A source's directivity per octave band: first-order patterns (a name, or the `a` of `a + (1 - a) cos(theta)`, for each band), a measured balloon from a SOFA file or a CSV table, or the far-field pattern of a coherent array, given here or in a YAML file (see `acoustic_spaces.directivity`).

pattern: `list[PatternName | Coefficient] | None`

file: `Path | None`

array: **ArraySpec** | None

measurement: `int | None`

near_field: `bool | None`

class acoustic_spaces.scene.**Foliage**(*, preset: Literal['dense', 'none'] | None = None, attenuation: list[Annotated[float, FieldInfo(annotation=NoneType, required=True, metadata=[Ge(ge=0.0), Le(le=10.0)])]] | None = None, leaf_area_per_m3: Annotated[Annotated[float, Gt(gt=0)] | None, Le(le=50.0)] = None, leaf_width: Annotated[Annotated[float, Gt(gt=0)] | None, Le(le=1.0)] = None, albedo:

```
Annotated[float, Ge(ge=0.0), Le(le=1.0)] = 0.0, asymmetry: Annotated[float,
Ge(ge=-0.9), Le(le=0.9)] = 0.0)
```

What the leaves and twigs of a vegetation volume take from sound passing through it, per metre: a preset, an attenuation per band, or the leaves themselves, their area per cubic metre and their width, from which the engine works out what they scatter and absorb in each band. `albedo` is the share of an attenuation that they scatter on, the rest they absorb; `asymmetry` how far forward they scatter it (the mean cosine of the angle, as in a Henyey-Greenstein phase function); leaves given by their size and density work both out for themselves.

preset: Literal['dense', 'none'] | None

attenuation: list[Annotated[float, Field(ge=0.0, le=10.0)]] | None

leaf_area_per_m3: PositiveFloat | None

leaf_width: PositiveFloat | None

albedo: Coefficient

asymmetry: float

property bare: bool

Whether there is no foliage.

attenuation_db(resolution: Literal['octave', 'third_octave'] = 'octave', speed_of_sound: float = 343.0) → ndarray

dB per metre in each band at `resolution`: what the foliage takes in all, scattered on or absorbed; leaves given by their area and width take it for sound at `speed_of_sound` (m/s).

describe() → str

class acoustic_spaces.scene.**Geometry**(*, file: Path | None = None, box: tuple[Annotated[float, Gt(gt=0)], Annotated[float, Gt(gt=0)], Annotated[float, Gt(gt=0)]] | None = None, floor_plan: Annotated[list[tuple[float, float]] | None, MinLen(min_length=3)] = None, height: Annotated[float, Gt(gt=0)] | None = None, units: Literal['m', 'cm', 'mm', 'in', 'ft'] | None = None, up_axis: Literal['y', 'z'] | None = None, space: str | None = None, ground: Ground | None = None, buildings: list[Building] | None = None, buildings_file: BuildingFile | None = None, barriers: list[Barrier] | None = None, vegetation: list[Vegetation] | None = None)

The room's surfaces: a CAD file, or a box or extruded floor plan built from dimensions (surfaces `floor`, `ceiling`, `walls` and `wall_1`, `wall_2`…).

Outdoors, `ground` lays flat ground under an open sky, with `buildings` and `barriers` built on it and, from `file`, any other objects standing on it. `vegetation` adds trees, hedges and belts of them, which are no surfaces: sound passes through them, losing some of itself.

`file`: `Path | None`

`box`: `tuple[PositiveFloat, PositiveFloat, PositiveFloat] | None`

`floor_plan`: `list[tuple[float, float]] | None`

`height`: `PositiveFloat | None`

`units`: `Literal['m', 'cm', 'mm', 'in', 'ft'] | None`

`up_axis`: `Literal['y', 'z'] | None`

`space`: `str | None`

`ground`: `Ground | None`

`buildings`: `list[Building] | None`

`buildings_file`: `BuildingFile | None`

`barriers`: `list[Barrier] | None`

`vegetation`: `list[Vegetation] | None`

`building_names``() → list[str]`

The buildings' names, `building_<n>` for those without one: the scene's own buildings, then the buildings file's.

`all_buildings``() → list[Building]`

The scene's own `buildings`, then those of `buildings_file`, every one named (`building_<n>` for those without a name).

`file_buildings``(first: int = 0, all: bool = False) → list[Building]`

The buildings of `buildings_file` as `Building` objects, in the geometry's units with `ground.map_origin` subtracted, named `building_<n>` from `first + 1` where the file gives no name, and those in `without` left out (kept with `all`).

`footprint_notes``() → list[str]`

What is worth knowing about the buildings file: footprints with holes, which are built over.

`barrier_names``() → list[str]`

The barriers' names, `barrier_<n>` for those without one.

`vegetation_names``() → list[str]`

The vegetation's names, `vegetation_<n>` for those without one.

`part_names``() → list[str]`

The names of an outdoor scene's buildings, barriers and vegetation, in turn: what an insertion loss can be given for (`simulation.insertion_loss_of`).

```
    describe() → str
```

```
class acoustic_spaces.scene.Ground(*, height: float = 0.0, terrain: Terrain | None =
    None, map_origin: tuple[float, float] | None = None, extent: tuple[tuple[float,
    float], tuple[float, float]] | None = None, margin: Annotated[float, Gt(gt=0)] =
    30.0, sky: Annotated[float, Gt(gt=0)] | None = None, zones: list[GroundZone] | None
    = None)
```

Flat ground under an open sky, or a terrain, which makes the scene an outdoor one (see `acoustic_spaces.geometry.outdoor`). Lengths are in the geometry's units.

```
    height: float
    terrain: Terrain | None
    map_origin: tuple[float, float] | None
    extent: tuple[tuple[float, float], tuple[float, float]] | None
    margin: PositiveFloat
    sky: PositiveFloat | None
    zones: list[GroundZone] | None
    zone_names() → list[str]
```

The zones' names, `zone_<n>` for those without one.

```
    height_grid()
```

The terrain as a `HeightGrid` in the geometry's units, `map_origin` subtracted and `height` not yet added; None for flat ground.

```
class acoustic_spaces.scene.GroundZone(*, name: str | None = None, polygon:
    Annotated[list[tuple[float, float]], MinLen(min_length=3)])
```

A part of the ground with a material of its own (a road through a field, a lawn, a pond): a polygon on the ground, its surfaces `<name>`, `zones` and `ground` (an assignment to the zone's name wins over one to `ground`). The part of it past the ground's edge is left out.

```
    name: str | None
    polygon: list[tuple[float, float]]
```

```
class acoustic_spaces.scene.Impedance(*, preset: str | None = None, model:
    Literal['miki', 'delany_bazley', 'two_parameter'] = 'miki', flow_resistivity:
    Annotated[float, Gt(gt=0)] | None = None, porosity_rate: Annotated[float | None,
    Ge(ge=0.0)] = None, thickness: Annotated[float, Gt(gt=0)] | None = None, reaction:
    Literal['local', 'extended'] | None = None)
```

The acoustic impedance of a porous ground (grass, soil, snow, gravel, asphalt), from its flow resistivity by an empirical model (see `acoustic_spaces.materials.ground`): a `preset`, one of the ground classes of Nord2000, or a `flow_resistivity`; with a `thickness`, a layer of it on a hard backing.

A material with an impedance takes its absorption from it, and reflects sound as a ground does: the spherical wave's reflection (Weyl-van der Pol), with its phase and the ground wave near grazing incidence, in place of the absorption.

preset: str | None

model: Literal['miki', 'delany_bazley', 'two_parameter']

flow_resistivity: PositiveFloat | None

porosity_rate: float | None

thickness: PositiveFloat | None

reaction: Literal['local', 'extended'] | None

property resistivity: float

The flow resistivity (kPa s/m²), the preset's when a preset is given.

property extended: bool

Whether the impedance changes with the angle of incidence.

admittance(f, speed_of_sound: float = 343.0, cosine=None) → ndarray

The normalised admittance (1 / impedance) at frequencies `f` (Hz), in the engine's sign convention, e^(jωt): with the extended reaction, the one met at the incidence `cosine` (broadcasting with `f`; 1 by default, normal incidence); with the local, the same at every angle.

describe() → str

In a few words: the preset or the flow resistivity, and the rest.

class acoustic_spaces.scene.**Keyframe**(*, time_s: Annotated[float, Gt(gt=0)], position: tuple[float, float, float], orientation: Orientation | None = None)

Where a listener or a source is at one moment of its movement, and how it faces (`Walk`, `keyframes`).

time_s: PositiveFloat

position: Vector3

orientation: Orientation | None

class acoustic_spaces.scene.**Listener**(*, name: str, group: str | None = None, enabled: bool | None = None, position: tuple[float, float, float], orientation: ~acoustic_spaces.scene.model.Orientation = <factory>, walk: ~acoustic_spaces.scene.model.Walk | None = None, pattern: ~typing.Literal['omni', 'subcardioid', 'cardioid', 'supercardioid', 'hypercardioid', 'figure8'] | list[~typing.Literal['omni', 'subcardioid', 'cardioid', 'supercardioid', 'hypercardioid', 'figure8'] | ~typing.Annotated[float, FieldInfo(annotation=NoneType, required=True, metadata=[Ge(ge=0.0), Le(le=1.0)])]] | ~typing.Annotated[~acoustic_spaces.scene.model.XYPair |

```
~acoustic_spaces.scene.model.BlumleinPair | ~acoustic_spaces.scene.model.MSPair,
FieldInfo(annotation=NoneType, required=True, discriminator='pair')] | None = None)
```

name: str

group: str | None

enabled: bool | None

position: Vector3

orientation: Orientation

walk: Walk | None

pattern: CapsulePattern | MicrophonePair | None

property in_use: bool

Whether simulations and maps take it: not left out (`enabled: false`).

```
class acoustic_spaces.scene.MSPair(*, pair: Literal['ms'], mid: Literal['omni',
'subcardioid', 'cardioid', 'supercardioid', 'hypercardioid', 'figure8'] |
list[Literal['omni', 'subcardioid', 'cardioid', 'supercardioid', 'hypercardioid',
'figure8'] | Annotated[float, FieldInfo(annotation=NoneType, required=True,
metadata=[Ge(ge=0.0), Le(le=1.0)])]] = 'cardioid', side_db: Annotated[float,
Ge(ge=-40.0), Le(le=20.0)] = 0.0)
```

A mid capsule aimed forward and a figure-eight aimed left, decoded to left and right.

pair: Literal['ms']

mid: CapsulePattern

side_db: float

```
class acoustic_spaces.scene.Material(*, preset: str | None = None, file: Path | None =
None, entry: str | int | None = None, absorption: list[Annotated[float,
FieldInfo(annotation=NoneType, required=True, metadata=[Ge(ge=0.0), Le(le=1.0)])]] |
None = None, scattering: list[Annotated[float, FieldInfo(annotation=NoneType,
required=True, metadata=[Ge(ge=0.0), Le(le=1.0)])]] | Annotated[float,
FieldInfo(annotation=NoneType, required=True, metadata=[Ge(ge=0.0), Le(le=1.0)])] |
str | None = None, transmission: list[Annotated[float,
FieldInfo(annotation=NoneType, required=True, metadata=[Ge(ge=0.0), Le(le=1.0)])]] |
Annotated[float, FieldInfo(annotation=NoneType, required=True, metadata=[Ge(ge=0.0),
Le(le=1.0)])] | None = None, transmission_loss: list[float] | float | None = None,
audience: bool = False, impedance: Impedance | None = None, source: str | None =
None, uncertainty: AbsorptionUncertainty | None = None)
```

Random-incidence absorption and scattering per octave or third-octave band, and the share of sound that passes through.

Coefficients come from, in order of precedence: values given here (8 per octave band or 24 per third-octave band), a coefficient file (`file`), or a built-in database entry (`preset`). The file may also be a material library of CATT-Acoustic or ODEON, with `entry` naming the material in it (see `acoustic_spaces.materials.libraries`). `scattering` may also name a scattering entry of the built-in database; a material that gives none scatters `DEFAULT_SCATTERING` in every band, and `GROUND_SCATTERING`, nothing, where it has an `impedance`. Data at other frequencies is mapped onto third-octave bands when it has more than one value in some octave band, else onto octave bands; bands without data take the nearest band's value, and `source` says which ones. `coefficients()` gives both lists in the bands a simulation runs in.

`transmission` (a share from 0 to 1) or `transmission_loss` (the sound reduction index R in dB, the transmission being 10^(-R/10)) makes a surface let sound through: a partition between two rooms, a door, or with a transmission of 1 an opening. What passes through counts as absorbed on the side it comes from, so the absorption must be at least the transmission in every band; without absorption values, a preset or a file, the absorption is the transmission (an opening absorbs 1, a heavy wall no more than it lets through). `transmission_coefficients()` gives it in the bands a simulation runs in.

```
preset: str | None
file: Path | None
entry: str | int | None
absorption: list[Coefficient] | None
scattering: list[Coefficient] | Coefficient | str | None
transmission: list[Coefficient] | Coefficient | None
transmission_loss: list[float] | float | None
audience: bool
impedance: Impedance | None
source: str | None
uncertainty: AbsorptionUncertainty | None
property clipped: list[str]
```

What of the coefficients taken from a preset or file lay outside 0 to 1 and was taken as 0 or 1: one note each for the absorption and the scattering.

```
property builtin: bool
```

Whether the absorption is a built-in database entry's (`preset`, with no other absorption values given): the database does not name where its values come from, so a spread takes `simulation.uncertainty.presets` for it. By value: a copy with other absorption is not.

```
coefficients(bands: Literal['octave', 'third_octave'] = 'octave') →
    tuple[list[float], list[float]]
```

Absorption and scattering in octave or third-octave `bands`: an octave value holds for its three third octaves, and an octave takes the mean of its three.

```
transmission_coefficients(bands: Literal['octave', 'third_octave'] = 'octave') →
    list[float] | None
```

The share of sound that passes through, in octave or third-octave `bands` (converted as `coefficients()` does), or None for a surface that lets nothing through.

to_data() → dict

The material as scene data, as `Scene.to_data()` writes it: its absorption, scattering and source only where they are not what the rest of the entry gives (a preset, a file and entry, an impedance, a transmission, a scattering entry, written by its name, or the default scattering), so that the entry read again has the same values, and with another preset, file or entry the values of that.

model_post_init(context: Any, /) → None

This function is meant to behave like a BaseModel method to initialize private attributes.

It takes context as an argument since that's what pydantic-core passes when calling it.

Args:

self: The BaseModel instance. context: The context.

class acoustic_spaces.scene.**Orientation**(*, yaw: float = 0.0, pitch: float = 0.0, roll: float = 0.0)

Yaw, pitch and roll in degrees; all zero looks down +x with +z up.

Yaw turns left (from +x towards +y), pitch looks up, and roll tilts the head to the right (the left ear goes up), applied in the order roll, pitch, yaw.

yaw: float

pitch: float

roll: float

forward() → tuple[float, float, float]

Unit vector of the look direction: yaw turns from +x towards +y, pitch up.

rotation() → list[list[float]]

3 x 3 matrix whose columns are the forward, left and up axes in room coordinates; it turns head coordinates (x forward, y left, z up) into room ones.

class acoustic_spaces.scene.**Output**(*, format: Literal['mono', 'binaural', 'ambisonics', 'loudspeakers'], order: Annotated[int, Ge(ge=0), Le(le=7)] = 1, hrtf: Path | None = None, layout: str | None = None, speakers: list[Speaker] | None = None)

A format every response is rendered in: `mono`, `binaural`, `ambisonics` (AmbiX) or `loudspeakers`. A bare name stands for the format with its defaults.

format: Literal['mono', 'binaural', 'ambisonics', 'loudspeakers']

order: int

hrtf: Path | None

layout: str | None

speakers: list[Speaker] | None

class acoustic_spaces.scene.**Scene**(*, name: str = 'untitled', geometry: ~acoustic_spaces.scene.model.Geometry, materials: ~typing.Annotated[dict[str,

```
~acoustic_spaces.scene.model.Material], ~annotated_types.MinLen(min_length=1)],
assignments: dict[str, str] = <factory>, default_material: str | None = None,
recording: ~pathlib.Path | str | None = None, sources:
~typing.Annotated[list[~acoustic_spaces.scene.model.Source],
~annotated_types.MinLen(min_length=1)], listeners:
~typing.Annotated[list[~acoustic_spaces.scene.model.Listener],
~annotated_types.MinLen(min_length=1)], air: ~acoustic_spaces.scene.model.Air =
<factory>, atmosphere: ~acoustic_spaces.scene.model.Atmosphere | None = None,
simulation: ~acoustic_spaces.scene.model.Simulation = <factory>, outputs:
list[~acoustic_spaces.scene.model.Output] = <factory>, coverage:
~acoustic_spaces.scene.model.Coverage | None = None, long_term:
~acoustic_spaces.scene.model.LongTerm | None = None)
```

name: str

geometry: Geometry

materials: dict[str, Material]

assignments: dict[str, str]

default_material: str | None

recording: Path | str | None

sources: list[Source]

listeners: list[Listener]

air: Air

atmosphere: Atmosphere | None

simulation: Simulation

outputs: list[Output]

coverage: Coverage | None

long_term: LongTerm | None

property recording_file: Path | None

The scene's recording when it is a file of its own (not a free one), or None.

level_inputs(source: str) → tuple[ndarray | float, list[float] | float | None, Literal['male', 'female']]

(sound power, noise, talker) that the levels and the speech results of the source named `source` are found with at a listener: its sound power per octave band (dB re 1 pW, `Source.power_db()`) and its talker, 94 dB in all and a male talker for a source the scene does not have; and the background noise of `coverage` (dB SPL, in all bands or per octave band), None without.

property still: bool

Whether sound goes in straight lines in every run of the scene: in its own weather (`Atmosphere.still`, the zones too) and in each weather class of `long_term` (`Atmosphere.under()`).

under(given: WeatherClass) → Scene

This outdoor scene in a weather class: its atmosphere under the class's weather (`Atmosphere.under()`), the air at the class's ground temperature when it gives one, and without `long_term`.

without(names: Sequence[str]) → Scene

This outdoor scene without the buildings, barriers and vegetation of these names (see `Geometry.part_names()`), and without `insertion_loss_of`.

classmethod from_yaml(path: str | Path) → Scene

Load a scene file; relative paths in it are resolved against its folder.

to_data() → dict

The scene as plain YAML data, dicts and lists. Settings left unset are left out, apart from those whose null is not their default, and so are a material's coefficients and source where its preset, file or other settings give them (`Material.to_data()`).

yaml_text(file_name: Callable[[str], str], *, readable: bool = False) → str

The scene file's text, with each file it names written as `file_name(file)`. `readable` writes each list of numbers or names on one line in brackets and letters outside ASCII as they are, as the example scenes do; the data are the same.

file_text(folder: str | Path, *, readable: bool = False) → str

The text of the scene's file in `folder`: paths to files under it relative to it, so the folder can move, others absolute (`readable` as in `yaml_text()`).

to_yaml(path: str | Path) → None

Write the scene file (`file_text()` for its folder).

files() → list[Path]

The files the scene uses, each once: its model, terrain and map of buildings, material tables, source directivities (with the files a coherent array names for its element, in the scene or in its YAML file), the sources' signals, the scene's recording when it is a file, and HRTFs.

class acoustic_spaces.scene.**Source**(*, name: str, group: str | None = None, enabled: bool | None = None, position: tuple[float, float, float], orientation: ~acoustic_spaces.scene.model.Orientation = <factory>, directivity: ~typing.Literal['omni', 'subcardioid', 'cardioid', 'supercardioid', 'hypercardioid', 'figure8'] | ~typing.Literal['lambert'] | ~acoustic_spaces.scene.model.DirectivitySpec = 'omni', line_m: ~typing.Annotated[float, ~annotated_types.Gt(gt=0)] | None = None, surface_m: tuple[~typing.Annotated[float, ~annotated_types.Gt(gt=0)], ~typing.Annotated[float, ~annotated_types.Gt(gt=0)]] | None = None, spacing_m: ~typing.Annotated[float, ~annotated_types.Gt(gt=0)] = 0.5, level_db: float | list[float] = 94.0, spectrum: ~typing.Literal['flat', 'speech_male', 'speech_female'] = 'flat', gain_db: ~typing.Annotated[float, ~annotated_types.Ge(ge=-100.0), ~annotated_types.Le(le=60.0)] = 0.0, eq_db: list[~typing.Annotated[float, FieldInfo(annotation=NoneType, required=True, metadata=[Ge(ge=-100.0), Le(le=60.0)])]] | None = None, delay_ms: ~typing.Annotated[float, ~annotated_types.Ge(ge=0.0), ~annotated_types.Le(le=10000.0)] = 0.0, walk:

```
~acoustic_spaces.scene.model.Walk | None = None, signal: ~pathlib.Path | None =
None, channel: ~typing.Literal['left', 'right', 'mono'] | None = None)
```

```
name: str
```

```
group: str | None
```

```
enabled: bool | None
```

```
position: Vector3
```

```
orientation: Orientation
```

```
directivity: PatternName | Literal['lambert'] | DirectivitySpec
```

```
line_m: PositiveFloat | None
```

```
surface_m: tuple[PositiveFloat, PositiveFloat] | None
```

```
spacing_m: PositiveFloat
```

```
level_db: float | list[float]
```

```
spectrum: Literal['flat', 'speech_male', 'speech_female']
```

```
gain_db: float
```

```
eq_db: list[Annotated[float, Field(ge=-100.0, le=60.0)]] | None
```

```
delay_ms: float
```

```
walk: Walk | None
```

```
signal: Path | None
```

```
channel: Literal['left', 'right', 'mono'] | None
```

```
property in_use: bool
```

Whether simulations and maps take it: not left out (`enabled: false`).

```
property extended: bool
```

Whether the source is a line or a surface rather than a point.

```
elements() → ndarray
```

Positions (n, 3) of the point sources that make up the source: its position alone for a point source; along a line or over a surface evenly spaced, each in the middle of an equal share of it.

```
outline() → ndarray
```

Corners (n, 3) of the source's extent: a line's two ends, a surface's four corners in turn, or a point source's position alone.

```
property talker: Literal['male', 'female']
```

The speech the STI weights for: female for `speech_female`, else male.

```
feed_db(bands: Literal['octave', 'third_octave'] = 'octave') → ndarray
```

The gain of the feed per band (dB): `gain_db` and `eq_db` together.

```
property sound_power_db: float
```

Sound power level in all (dB re 1 pW) that `level_db` gives, summed over its bands when given per band, without the feed's gain.

power_db(bands: Literal['octave', 'third_octave'] = 'octave') → ndarray

Sound power level per band (dB re 1 pW), the feed's gain included: an octave's power is split evenly between its third octaves.

class acoustic_spaces.scene.**Speaker**(*, name: str, azimuth: float, elevation: Annotated[float, Ge(ge=-90.0), Le(le=90.0)] = 0.0, lfe: bool = False)

name: str

azimuth: float

elevation: float

lfe: bool

class acoustic_spaces.scene.**Stems**(*, per_m2: Annotated[float, Le(le=100.0), Gt(gt=0)], diameter: Annotated[float, Le(le=5.0), Gt(gt=0)])

The trunks and stems of a vegetation volume: upright rigid cylinders, which scatter sound sideways, all of it (Bullen and Fricke 1982).

per_m2: PositiveFloat

diameter: PositiveFloat

class acoustic_spaces.scene.**Terrain**(*, file: Path | None = None, heights: list[list[float]] | None = None, origin: tuple[float, float] = (0.0, 0.0), spacing: Annotated[float, Gt(gt=0)] | None = None)

Uneven ground: its heights on a square grid, from a file or given here (see `acoustic_spaces.geometry.terrain`). Between the grid's nodes the ground is bilinear, past its edge it goes on at the height of the edge, and the scene's ground is the grid sampled over the ground's extent, so the extent must lie inside the grid. Lengths are in the geometry's units.

file: Path | None

heights: list[list[float]] | None

origin: tuple[float, float]

spacing: PositiveFloat | None

class acoustic_spaces.scene.**Turbulence**(*, preset: str | None = None, temperature_structure: Annotated[float, Ge(ge=0.0), Le(le=10.0)] = 0.0, wind_structure: Annotated[float, Ge(ge=0.0), Le(le=10.0)] = 0.0, outer_scale_m: Annotated[float, Le(le=1000.0), Gt(gt=0)] = 10.0, height_m: Annotated[Annotated[float, Gt(gt=0)] | None, Le(le=100.0)] = None, falloff: Annotated[float, Ge(ge=0.0), Le(le=2.0)] = 0.6666666666666666)

The eddies of the wind and of the temperature, which make sound outdoors flicker: the paths of one sound that take different ways stop adding in step (the ground dip fills in), and sound is scattered into shadow zones. Their strength is given by the structure parameters of the temperature and of the wind, C_T² and C_v², with a von Kármán spectrum whose largest eddies are `outer_scale_m` across.

preset: str | None

temperature_structure: float

wind_structure: float

outer_scale_m: PositiveFloat

height_m: PositiveFloat | None

falloff: float

property calm: bool

Whether there are no eddies at all.

```
class acoustic_spaces.scene.Vegetation(*, name: str | None = None, footprint:
    ~typing.Annotated[list[tuple[float, float]] | None,
    ~annotated_types.MinLen(min_length=3)] = None, path:
    ~typing.Annotated[list[tuple[float, float]] | None,
    ~annotated_types.MinLen(min_length=2)] = None, width: ~typing.Annotated[float,
    ~annotated_types.Gt(gt=0)] | None = None, tree: tuple[float, float] | None = None,
    crown_diameter: ~typing.Annotated[float, ~annotated_types.Gt(gt=0)] | None = None,
    height: ~typing.Annotated[float, ~annotated_types.Gt(gt=0)], base:
    ~typing.Annotated[float, ~annotated_types.Ge(ge=0)] = 0.0, foliage:
    ~acoustic_spaces.scene.model.Foliage = <factory>, stems:
    ~acoustic_spaces.scene.model.Stems | None = None)
```

Plants standing on the ground outdoors: a volume through which sound loses some of itself, scattered on or absorbed. Its shape is one of a `footprint` raised from `base` to `height` (a tree belt, a wood), a `path` of a `width` (a hedge), or a lone tree's crown over `tree`: a spheroid `crown_diameter` across from `base` to `height`. Lengths and heights are in the geometry's units, over the ground; `foliage` and `stems` say what fills it.

name: str | None

footprint: list[tuple[float, float]] | None

path: list[tuple[float, float]] | None

width: PositiveFloat | None

tree: tuple[float, float] | None

crown_diameter: PositiveFloat | None

height: PositiveFloat

base: float

foliage: Foliage

stems: Stems | None

property kind: Literal['belt', 'hedge', 'tree']

belt (a footprint), hedge (a path) or tree.

describe() → str

What fills it, as text: its foliage and stems.

class acoustic_spaces.scene.**Walk**(*, enabled: bool = True, to: list[tuple[float, float, float]]=<factory>, keyframes: Keyframe] | None, ~annotated_types.MinLen(min_length=1)] = None, speed_m_s: Annotated[float, ~annotated_types.Gt(gt=0)] = 1.2, face_forward: bool = True, turn_deg_s: Annotated[float, ~annotated_types.Gt(gt=0)] = 90.0, doppler: bool = False)

A path a listener or a point source walks along, for a walkthrough (`acoustic_spaces.walkthrough`): from its position through `to` at `speed_m_s`, or through `keyframes`, each timed.

enabled: bool

to: list[Vector3]

keyframes: list[Keyframe] | None

speed_m_s: PositiveFloat

face_forward: bool

turn_deg_s: PositiveFloat

doppler: bool

class acoustic_spaces.scene.**LongTerm**(*, periods: Annotated[list[Period] | None, MinLen(min_length=1)] = None, classes: Annotated[list[WeatherClass], MinLen(min_length=1)])

The levels over a long term, outdoors: the weather changes from day to day, and the level over a year is the energy mean of the levels in each weather, weighted by the share of the time it holds (ISO 1996-2, CNOSSOS-EU, Nord2000). The scene is run once in each weather class, and the results give the level in each class, in each period of the day (the classes' levels, each by its share in that period) and the rating level over the day (the periods' levels, each by its hours, with its penalty: L_den with the default periods).

periods: list[Period] | None

classes: list[WeatherClass]

class_names() → list[str]

The classes' names, `class_<n>` for those without one.

period_list() → list[Period]

The periods of the day, L_den's when none are given.

period_names() → list[str]

shares() → ndarray

(classes, periods) the share of each period's time each class holds; a class that gives no share for a period, in a mapping, holds none of it.

hours() → ndarray

(periods,) how many hours a day each period lasts.

penalties() → ndarray

(periods,) the penalty added to each period's level in the rating (dB).

class acoustic_spaces.scene.**Period**(*, name: Annotated[str, MinLen(min_length=1)], hours: Annotated[float, Le(le=24.0), Gt(gt=0)], penalty_db: Annotated[float, Ge(ge=0.0), Le(le=20.0)] = 0.0)

A part of the day the rating level over a long term is taken over: its hours and the penalty added to its level in the rating (5 dB in the evening and 10 dB at night for L_den).

name: str

hours: PositiveFloat

penalty_db: float

class acoustic_spaces.scene.**WeatherClass**(*, name: str | None = None, preset: str | None = None, wind: ZoneWind | None = None, temperature_gradient_k_m: Annotated[float | None, Ge(ge=-0.2), Le(le=0.2)] = None, temperature_log_k: Annotated[float | None, Ge(ge=-5.0), Le(le=5.0)] = None, roughness_m: Annotated[Annotated[float, Gt(gt=0)] | None, Le(le=2.0)] = None, table: list[tuple[float, float, float]] | None = None, temperature_c: Annotated[float | None, Ge(ge=-20.0), Le(le=50.0)] = None, share: float | dict[str, float], turbulence: Turbulence | None = None)

A weather that holds for a share of the time, for the levels over a long term (`LongTerm`): downwind, upwind, a clear night's inversion… Its weather is given over the scene's atmosphere's (see `_WeatherOverride`), the eddies too, and the scene's weather zones hold in every class.

name: `str | None`

share: `float | dict[str, float]`

turbulence: `Turbulence | None`

class acoustic_spaces.scene.**WeatherZone**(*, name: str | None = None, preset: str | None = None, wind: ZoneWind | None = None, temperature_gradient_k_m: Annotated[float | None, Ge(ge=-0.2), Le(le=0.2)] = None, temperature_log_k: Annotated[float | None, Ge(ge=-5.0), Le(le=5.0)] = None, roughness_m: Annotated[Annotated[float, Gt(gt=0)] | None, Le(le=2.0)] = None, table: list[tuple[float, float, float]] | None = None, temperature_c: Annotated[float | None, Ge(ge=-20.0), Le(le=50.0)] = None, polygon: Annotated[list[tuple[float, float]], MinLen(min_length=3)])

A part of the scene over which the weather differs: the wind slowed in the lee of buildings or over a wood, a sea breeze over the water, an inversion over a lake. A polygon on the ground, in the geometry's units; where zones overlap the first wins, and outside every zone the atmosphere's own weather holds. Its weather is given over the atmosphere's (see `_WeatherOverride`); the eddies are the atmosphere's.

name: `str | None`

polygon: `list[tuple[float, float]]`

class acoustic_spaces.scene.**Wind**(*, speed_m_s: Annotated[float, Ge(ge=0.0), Le(le=40.0)] = 0.0, direction_deg: float = 0.0, height_m: Annotated[float, Gt(gt=0)] = 10.0, profile: Literal['log', 'linear', 'constant'] = 'log')

The wind over the ground: its speed at a height, the way it blows, and how the speed grows with height.

speed_m_s: `float`

direction_deg: `float`

height_m: `PositiveFloat`

profile: `Literal['log', 'linear', 'constant']`

class acoustic_spaces.scene.**XYPair**(*, pair: Literal['xy'], angle_deg: Annotated[float, Gt(gt=0.0), Le(le=180.0)] = 90.0, capsule: Literal['omni', 'subcardioid', 'cardioid', 'supercardioid', 'hypercardioid', 'figure8'] | list[Literal['omni', 'subcardioid', 'cardioid', 'supercardioid', 'hypercardioid', 'figure8'] | Annotated[float, FieldInfo(annotation=NoneType, required=True, metadata=[Ge(ge=0.0), Le(le=1.0)])]] = 'cardioid')

Two capsules of one pattern in one place, aimed apart in the horizontal plane.

pair: `Literal['xy']`

angle_deg: `float`

capsule: `CapsulePattern`

```
class acoustic_spaces.scene.ZoneWind(*, speed_m_s: Annotated[float | None, Ge(ge=0.0),
    Le(le=40.0)] = None, direction_deg: float | None = None, height_m: Annotated[float,
    Gt(gt=0)] | None = None, profile: Literal['log', 'linear', 'constant'] | None =
    None)
```

The wind over a weather zone: what it gives replaces the atmosphere's wind, and what it leaves out (None) stays the atmosphere's.

```
speed_m_s: float | None
```

```
direction_deg: float | None
```

```
height_m: PositiveFloat | None
```

```
profile: Literal['log', 'linear', 'constant'] | None
```

A scene with its geometry loaded: a material on every surface, and checks.

`Room` is what the simulation stages consume. It also gives the classic statistical estimates (Sabine and Eyring reverberation times) so a model can be sanity-checked before any ray is traced.

```
acoustic_spaces.room.MIN_SOURCE_DISTANCE_M = 0.01
```

A listener closer than this (metres) to a source has no response: the direct sound's level grows without bound as the distance goes to zero.

```
acoustic_spaces.room.ON_SURFACE_M = 1e-06
```

A source or listener closer than this (metres) to a surface is on it: neither in the room nor out of it (the distance at which two corners count as one).

```
acoustic_spaces.room.ON_SURFACE_MARGIN = 2.0
```

In a large model the distance is this many times the least length of a ray traced on the CPU, where that is more (`Room.on_surface_m()`).

```
acoustic_spaces.room.NEAR_SURFACE_SHARE = 0.1
```

The late part's detector sphere, kept clear of the surfaces, smaller than this share of the smallest one sized for the room (`late.RADIUS_RANGE`) catches a hundredth of the rays or fewer: a listener that close to a surface gets a noisy late part.

```
acoustic_spaces.room.every_left_out(kind: str) → str
```

The problem of a scene whose every `kind` (source, listener) is left out with `enabled: false`, for a run that takes those in use.

```
acoustic_spaces.room.matching_name(label: tuple[str, ...], names: Iterable[str]) → str |
    None
```

The first of `names` that a surface's label holds: as it is written, or else in any case (`brick_rough` names SketchUp's `Brick_Rough`); None for none.

```
class acoustic_spaces.room.Room(scene: 'Scene', mesh: 'RoomMesh', report: 'MeshReport',
    material_names: 'list[str]', face_materials: 'np.ndarray', sources_inside:
    'np.ndarray | None', listeners_inside: 'np.ndarray | None', unused_assignments:
    'list[str]', directivities: 'list[Directivity]', air_cells: 'np.ndarray', outdoor:
```

```
'Layout | None' = None, sources_in_solid: 'np.ndarray | None' = None,
listeners_in_solid: 'np.ndarray | None' = None)
```

material_names: list[str]

Material names in `scene.materials` order; `face_materials` indexes this.

face_materials: ndarray

(F,) index into `material_names` per face, `UNASSIGNED` (-1) when none matched.

sources_inside: ndarray | None

Whether each source / listener is inside; None when the mesh is not closed.

unused_assignments: list[str]

Assignment keys that match no name in the geometry.

directivities: list[Directivity]

Each source's directivity, in `scene.sources` order.

air_cells: ndarray

(R,) the cells of `mesh.cells()` that hold air: the rooms. Usually the model's rooms (see `acoustic_spaces.geometry.cells`); when a source or a listener lies in another enclosed cell (a model with thick walls, whose rooms are cells inside the walls' shell), the cells they lie in, with those that surfaces letting sound through join to them. Empty when the mesh is not closed.

outdoor: Layout | None = None

Where the ground and the sky lie in an outdoor scene (`geometry.ground`); None for a room.

sources_in_solid: ndarray | None = None

Whether each source (any of its point sources) / listener stands inside a closed solid: in a cell the faces enclose that holds no air (see `air_cells`); None when the mesh is not closed.

classmethod from_scene(scene: Scene, *, reuse: Room | None = None) → Room

Load the scene's geometry and give every surface its material.

`reuse` is an earlier room: when its geometry settings equal the scene's, its mesh is kept instead of being loaded and checked again (an editor changing materials or positions), and so are directivities read from the same settings.

directivity(source: str, bands: Literal['octave', 'third_octave'] = 'octave') → Directivity

The directivity of the source with this name, in octave or third-octave `bands`.

property bands: Literal['octave', 'third_octave']

The bands the scene's simulation runs in: `octave` or `third_octave`.

property band_centres: tuple[int, ...]

Centre frequencies (Hz) of the bands the scene's simulation runs in.

sky_faces() → ndarray

(F,) the faces of an outdoor scene's open sky (all False for a room).

building_faces() → ndarray

(F,) the faces of an outdoor scene's buildings (`geometry.buildings`).

shown_faces() → ndarray

(F,) the faces worth drawing: all but an outdoor scene's sky, which would hide the rest.

unassigned_labels() → dict[str, float]

Surface area (m2) without a material, by the names the faces carry.

problems(in_use: Collection[str] = ('sources', 'listeners'), named: Mapping[str, Collection[str]] | None = None) → list[str]

Issues that would make a simulation wrong; fix these first. Sources and listeners left out (`enabled: false`) are not checked, unless `named` names them (`{"sources": names, "listeners": names}`: those a run takes by name, even when left out); but every one of them left out is a problem for the kinds in `in_use`: those a run takes as the scene has them in use, rather than by name (a map takes no listeners).

on_surface_m() → float

The distance (m) under which a source or a listener is on a surface: `ON_SURFACE_M`, or `ON_SURFACE_MARGIN` times the least length of a ray traced on the CPU in the model (`ray_start()`) where that is more, which it is in a model more than 500 m from corner to corner.

warnings() → list[str]

Things worth a look that do not stop a simulation; about the sources and listeners in use.

property volume_m3: float | None

The volume of air (m3): of all the rooms together; None when the model is not closed.

property room_volumes: ndarray

(R,) volume of each room (m3).

property face_room: ndarray

(F, 2) the room (an index into `room_volumes`) on the front (the side the face's normal points to) and on the back of each face, or -1 where a side faces no room: the outside, or the inside of a solid.

room_of(points) → ndarray

The room each point lies in (an index into `room_volumes`), or -1.

volume_at(points) → ndarray | None

(P,) the volume (m3) of the room each point lies in, where partitions or openings divide the model into rooms, and that of all the rooms together for a point in none; None when the model is not closed.

face_transmission(bands: Literal['octave', 'third_octave'] = 'octave') → ndarray | None

(F, bands) share of sound each face lets through, or None when none does.

area_by_material() → dict[str, float]

Surface area (m2) per material, in `material_names` order.

absorption_area(bands: Literal['octave', 'third_octave'] = 'octave') → ndarray

Equivalent absorption area per octave band, or per third-octave band with `bands="third_octave"` (m2): the sum of S * alpha over the sides of the faces that face a room. With partitions or openings between rooms, it is that of all the rooms as one: what passes from one room to another is not lost.

air_attenuation(bands: Literal['octave', 'third_octave'] = 'octave') → ndarray

Energy attenuation of the scene's air per octave band, or per third-octave band with `bands="third_octave"` (1/m), ISO 9613-1.

reverberation_time(formula: Literal['sabine', 'eyring', 'arau_puchades'] = 'sabine', air_absorption: bool = False, bands: Literal['octave', 'third_octave'] = 'octave') → ndarray

Statistical reverberation time per octave band, or per third-octave band with `bands="third_octave"` (s).

Sabine and Eyring assume a diffuse field; they are a sanity check, not a prediction. `arau_puchades` (Arau-Puchades, Acustica 65, 1988) allows for absorption spread unevenly between the room's directions: an Eyring time for each of the x, y and z axes from the surfaces facing it, combined as a geometric mean weighted by their area. Each face's area is shared between the axes by the squares of its normal's components, so in a box aligned with the axes every wall counts for the axis it faces. `air_absorption` adds the `4 m V` term of ISO 9613-1 air.

acoustic_spaces.room.**outline_edges(room: Room) → tuple[ndarray, ndarray]**

The model's edges seen from above, for a plan of the room: (E, 2, 2) end points (m) of the edges where the surface folds or ends, and (E,) the label of a face beside each. Vertical edges, points from above, are left out.

Source directivity

Source directivity: how a source shares its sound power between directions, per octave band (or third-octave band).

A directivity gives energy weights that average 1 over the sphere in every band, so a source radiates its sound power (`level_db`) whatever its pattern; the pattern only says where that power goes. Directions are in the source's own frame: x forward, y left, z up, as `acoustic_spaces.scene.Orientation` turns them; azimuth turns from the front towards the left and elevation is positive upwards, in degrees.

- `FirstOrder`: the patterns `a + (1 - a) cos(theta)` from omni (a = 1) to figure-8 (a = 0), one per band, so a loudspeaker can be omni in the bass and narrow in the treble.
- `Balloon`: measured levels at any set of directions, read from a SOFA file (AES69, the `FreeFieldDirectivityTF` convention) or a CSV table, interpolated over the sphere; or the far-field pattern of a coherent array (`array_directivity()`).

`load_directivity` turns a source's `directivity` entry of a scene file into one:

```
directivity: cardioid                       # one pattern for every band
directivity: {pattern: [omni, omni, subcardioid, cardioid, cardioid, supercardioid,
                        hypercardioid, hypercardioid]}
directivity: {file: speaker.sofa}           # or speaker.csv
directivity: {array: {count: 8, spacing_m: 0.15, steer_deg: -8}}
directivity: {file: line_array.yaml}        # an array's fields in a YAML file
```

The CSV table has a header row `azimuth, elevation` followed by band centre frequencies in Hz (octaves, thirds or any other layout), then one row per direction with the level in dB in each band (any reference; only differences matter). Lines starting with `#` are comments.

Measured data is averaged into octave bands. A simulation in third-octave bands takes each octave band's directivity for its three third octaves (`Directivity.in_bands()`).

acoustic_spaces.directivity.**FAR_FIELD_HZ** = 1000.0

The frequency (Hz) at which `far_field_distance()` is given in warnings.

acoustic_spaces.directivity.**GRID_STEP_DEG** = 5.0

Angular step of the grid measured data is interpolated on (degrees).

acoustic_spaces.directivity.**PATTERNS**: dict[str, float] = {'cardioid': 0.5, 'figure8': 0.0, 'hypercardioid': 0.25, 'omni': 1.0, 'subcardioid': 0.7, 'supercardioid': 0.37}

The first-order patterns by name: the pressure is a + (1 - a) cos(theta).

acoustic_spaces.directivity.**SPEED_OF_SOUND** = 343.21462268256795

Speed of sound (m/s) an array's phases take when no room gives one: air at 20 C.

class acoustic_spaces.directivity.**Balloon**(table: ndarray, name: str = '', extrapolated: tuple[int, ...] = (), near: **NearField** | None = None)

Measured energy at some directions, interpolated over the sphere.

The measurements are first spread onto a regular grid of elevations and azimuths (`GRID_STEP_DEG` apart), each of its points from the measured directions nearest to it (`from_measurements`), then read off it bilinearly, which the compiled tracer does the same way. Each band is scaled to average 1 over the sphere.

`table``: ndarray`

Interpolation grid: energy weights (elevations, azimuths, bands).

`name``: str = ''`

Short description, for listings and reports.

`extrapolated``: tuple[int, ...] = ()`

Octave bands (Hz) without data of their own, copied from the nearest band.

`near``: NearField | None = None`

A coherent array's elements, for its sound added up element by element near it (`near_field: true`); None reads every weight off the pattern.

`classmethod` **`from_measurements`**`(directions, energy, name: str = '', extrapolated: tuple[int, ...] = (), step: float = 5.0) → Balloon`

Interpolate energies (n, bands) measured in the directions of vectors (n, 3).

Directions given more than once (a table's -180 and 180 degrees of azimuth, its rows at a pole) are one direction, the mean of them, with the mean of their energies. Each grid point takes the `NEIGHBOURS` directions nearest to it, weighted by the inverse squares of their angles from it. Where several are as near as the last of those (within `SAME_DIRECTION`) and not all of them are among the `NEIGHBOURS`, none of them is taken, only the nearer ones; or all of them, if none is nearer. So directions equally near a point are taken alike, whatever their order in the data and the last digits of their distances.

`classmethod` **`from_table`**`(table, name: str = '', extrapolated: tuple[int, ...] = ()) → Balloon`

Energies on the grid of elevations (-90 to 90 degrees) and azimuths (0 up to 360) that `table` (elevations, azimuths, bands) spans, each band scaled to average 1 over the sphere; silent bands become omnidirectional.

`energy``(directions) → ndarray`

Energy weights (n, bands) for unit vectors (n, 3) in the source's frame; they average 1 over the sphere in each band.

`near_energy``(points) → ndarray`

Energy weights (n, bands) at points (n, 3) in the source's frame: for an array with `near`, its elements added up from where each is; otherwise, and far from the array alike, the pattern's weights towards the points.

`native``() → tuple[ndarray, ndarray]`

What the compiled tracer takes: the first-order `a` per band, and a table of weights on the interpolation grid (elevations x azimuths x bands, empty for first-order patterns).

in_bands(bands: Literal['octave', 'third_octave']) → Balloon

This directivity in octave or third-octave `bands`: an octave band's weights for each of its three third octaves, or the mean of three third octaves' for their octave.

class acoustic_spaces.directivity.**Directivity**

Energy weights per direction and octave band; see the module docstring.

name: str = ''

Short description, for listings and reports.

extrapolated: tuple[int, ...] = ()

Octave bands (Hz) without data of their own, copied from the nearest band.

energy(directions) → ndarray

Energy weights (n, bands) for unit vectors (n, 3) in the source's frame; they average 1 over the sphere in each band.

index() → ndarray

Directivity index per band (dB): the on-axis level above the average.

native() → tuple[ndarray, ndarray]

What the compiled tracer takes: the first-order `a` per band, and a table of weights on the interpolation grid (elevations x azimuths x bands, empty for first-order patterns).

in_bands(bands: Literal['octave', 'third_octave']) → Directivity

This directivity in octave or third-octave `bands`: an octave band's weights for each of its three third octaves, or the mean of three third octaves' for their octave.

class acoustic_spaces.directivity.**FirstOrder**(a: tuple[float, ...], name: str = '', extrapolated: tuple[int, ...] = ())

`a + (1 - a) cos(theta)` patterns, one `a` per octave band.

name: str = ''

Short description, for listings and reports.

extrapolated: tuple[int, ...] = ()

Octave bands (Hz) without data of their own, copied from the nearest band.

classmethod **pattern**(name: str) → FirstOrder

The same named pattern in every band.

energy(directions) → ndarray

Energy weights (n, bands) for unit vectors (n, 3) in the source's frame; they average 1 over the sphere in each band.

native() → tuple[ndarray, ndarray]

What the compiled tracer takes: the first-order `a` per band, and a table of weights on the interpolation grid (elevations x azimuths x bands, empty for first-order patterns).

in_bands(bands: Literal['octave', 'third_octave']) → FirstOrder

This directivity in octave or third-octave `bands`: an octave band's weights for each of its three third octaves, or the mean of three third octaves' for their octave.

class acoustic_spaces.directivity.**NearField**(positions: ndarray, gains: ndarray, delays: ndarray, turns: ndarray, element: Directivity, speed_of_sound: float, edges: ndarray, scale: ndarray)

A coherent array's elements (see `array_directivity()`), to add up their pressures at points near it, each from its own position with its own distance and direction, rather than read the far-field pattern off one point.

positions: ndarray

(elements, 3) positions in the source's frame (m).

gains: ndarray

(elements,) pressure gains, negative where the polarity is inverted.

delays: ndarray

(elements,) the feed's delays (s), steering included.

turns: ndarray

(elements, 3, 3) each element's forward, left and up axes as columns.

element: Directivity

The elements' own pattern, in third-octave bands.

edges: ndarray

(25,) the third-octave bands' edges (Hz).

scale: ndarray

(24,) one over the pattern's mean energy over the sphere in each third-octave band, which scales it to the weights' average of 1; 0 where the band is silent.

weights(points) → ndarray

Energy weights (n, 24) at points (n, 3) in the source's frame, per third-octave band, as the pattern's weights are: the elements' energy there, averaged over each band (of a flat spectrum) exactly, times the squared distance from the source's position. Far away they become the pattern's.

acoustic_spaces.directivity.**array_directivity**(spec, speed_of_sound: float = 343.21462268256795, near_field: bool = False) → Balloon

The far-field pattern of a coherent array (an `ArraySpec`), per third-octave band; with `near_field`, it also keeps the elements (`NearField`), whose sound the early paths add up element by element.

The elements' pressures add up with their phases: the path difference towards each direction, the feed's delays and the steering, the gains and polarities, and each element's own pattern along its aim (first-order patterns with their sign, measured ones by magnitude). Energies are exact averages over each band (of a flat spectrum), on a grid fine enough for the beam (1 to 5 degrees). The pattern holds in the far field, beyond about the array's length squared over the wavelength.

acoustic_spaces.directivity.**array_files**(path: str | Path, _seen: frozenset[Path] = frozenset({})) → list[Path]

The files a coherent array's YAML file names for its element, and those they name in turn; none for other files, or a missing one.

`acoustic_spaces.directivity.array_length(spec) → float | None`

The largest distance between two elements (m) of the coherent array a source's directivity entry describes, given in the scene or in a YAML file; None for other directivities.

`acoustic_spaces.directivity.directivity_files(spec, _seen: frozenset[Path] = frozenset({})) → list[Path]`

The files a source's directivity entry uses: its own file and, for a coherent array, its element's, whether the array is in the entry or in a YAML file (which names its element's file relative to itself), and so on down.

`acoustic_spaces.directivity.far_field_distance(length_m: float, frequency_hz: float = 1000.0, speed_of_sound: float = 343.21462268256795) → float`

From how far (m) an array `length_m` long acts as one point with its far-field pattern at `frequency_hz`: its length squared over the wavelength. Nearer, its sound spreads as a line's and its beam has not formed yet.

`acoustic_spaces.directivity.lambert() → Balloon`

A vibrating surface's radiation: energy in proportion to the cosine of the angle to the forward axis, nothing behind (Lambert's law), in every band.

`acoustic_spaces.directivity.load_directivity(spec, speed_of_sound: float = 343.21462268256795) → Directivity`

The directivity a scene's source entry describes: a pattern name (or `lambert`), or an object with `pattern` (a name or `a` value per band), `file` or `array`; an array's phases take `speed_of_sound` (m/s).

`acoustic_spaces.directivity.on_grid(table: ndarray, directions) → ndarray`

(n, …) the values `table` (elevations, azimuths, …) holds on a grid of elevations from -90 to 90 degrees and azimuths from 0 up to 360, each evenly spaced, read bilinearly at unit vectors (n, 3) in the source's frame.

`acoustic_spaces.directivity.read_array(path: str | Path)`

A coherent array (`ArraySpec`) from a YAML file holding its fields; files named in it are found next to it.

`acoustic_spaces.directivity.read_directivity(path: str | Path, measurement: int | None = None, speed_of_sound: float = 343.21462268256795, near_field: bool = False) → Balloon`

Measured directivity from a `.sofa` file or a CSV table, or an array's from a YAML file (`.yaml` or `.yml`; see `read_array()`), with its elements for `near_field`.

`acoustic_spaces.directivity.read_directivity_csv(path: str | Path) → Balloon`

A `Balloon` from a CSV table of levels (see the module docstring).

`acoustic_spaces.directivity.read_directivity_sofa(path: str | Path, measurement: int | None = None) → Balloon`

A `Balloon` from a SOFA file of source directivities (the `FreeFieldDirectivityTF` convention, or impulse responses in `Data.IR`).

Every receiver of every measurement gives one direction: where it sits relative to the source, in the source's frame (`SourceView` and `SourceUp`). `measurement` picks one measurement (an instrument's notes, for example); by default their energies are averaged. Spectra are averaged in energy over each octave band.

```
acoustic_spaces.directivity.sphere_mean(table: ndarray) → ndarray
```

(…) the mean over the sphere of what `table` holds on the grid of `on_grid()`, read bilinearly as it reads them.

```
acoustic_spaces.directivity.write_directivity_csv(path: str | Path, balloon: Balloon,
    comment: str = '') → Path
```

Write `balloon` (in octave bands) as the CSV table `read_directivity_csv()` reads back: a row for each direction of its grid with its level in each band (dB over the sphere's mean), under `comment`'s lines.

Geometry

Room geometry: CAD import, cleanup and checks.

Every importer returns a `RoomMesh`: triangles in metres with +z up, and for each triangle the names it carried in the source file (material, group, object or layer) so the scene's `assignments` can map them to acoustic materials. Simple rooms can also be built from dimensions (`box_mesh()`, `extruded_mesh()`).

class acoustic_spaces.geometry.**MeshCells**(side: ndarray, volume: ndarray, rooms: ndarray)

The cells of a mesh (see the module notes).

side: ndarray

(F, 2) cell of the front (the side the face's normal points to) and of the back of each face.

volume: ndarray

(C,) volume of each cell (m3): positive when enclosed, negative for the space around the model, about 0 for slivers and the inside of an open model.

rooms: ndarray

(R,) the cells that are rooms, in the order of their first face.

property outside: ndarray

(C,) True for the cells of the space around the model.

property enclosed: ndarray

(C,) True for cells with a volume: rooms and solids.

property face_room: ndarray

(F, 2) room (an index into `rooms`) of the front and back of each face, or -1 where a side faces no room.

property room_volumes: ndarray

(R,) volume of each room (m3).

property open_faces: ndarray

(F,) True for faces with neither side in an enclosed cell: a hole joins the room behind them to the outside, or they stand outside the model.

property closed: bool

At least one room, and every face has a side in an enclosed cell.

class acoustic_spaces.geometry.**MeshReport**(vertices: 'int', faces: 'int', merged_vertices: 'int', removed_degenerate_faces: 'int', boundary_edges: 'int', non_manifold_edges: 'int', inconsistent_edges: 'int', surface_area_m2: 'float', volume_m3: 'float | None', bounds_min: 'tuple[float, float, float]', bounds_max: 'tuple[float, float, float]', notes: 'list[str]' = <factory>, normals_inward: 'bool | None' = None,

odd_edges: 'int' = 0, room_volumes_m3: 'tuple[float, ...]'=(), open_faces: 'int' = 0)

removed_degenerate_faces: int

Faces cleaning removed: degenerate ones, and copies of another face.

normals_inward: bool | None = None

True when face normals point into the room, False outward, None when unknown (open model or inconsistent orientation).

odd_edges: int = 0

Edges shared by three or another odd number of faces: where a partition or an opening between two rooms meets the walls, or a crack.

room_volumes_m3: tuple[float, ...] = ()

Volume of each room (m3): the separate volumes of air the faces enclose, two or more where partitions or openings divide the model (see `acoustic_spaces.geometry.cells`).

open_faces: int = 0

Faces with no room or solid on either side: a hole joins the room behind them to the outside, or they stand outside the model.

property closed: bool

True when the faces enclose at least one room and every face has a room or a solid (the inside of a closed pillar) on one side at least. Free-standing panels inside a room leave it closed.

class acoustic_spaces.geometry.**RoomMesh**(vertices: ndarray, faces: ndarray, face_labels: ndarray, labels: list[tuple[str, ...]])

Triangles of a room model.

`labels` holds, for each distinct combination of names found in the source file, the candidate names in priority order (for OBJ: material, group, object). Each face points at one entry through `face_labels`.

property triangles: ndarray

(F, 3, 3) corner coordinates of every face.

face_normals() → ndarray

Unit normals following the right-hand rule on the vertex order.

cells()

The rooms and other cells the faces enclose (`MeshCells`), worked out once.

label_names() → list[str]

Every distinct name that appears in any label, in first-seen order.

acoustic_spaces.geometry.**box_mesh**(size: tuple[float, float, float]) → RoomMesh

A rectangular room from (0, 0, 0) to `size` = (length, width, height).

acoustic_spaces.geometry.**check_mesh**(mesh: RoomMesh, merged: int = 0, removed: int = 0) →

MeshReport

Topology, size and orientation checks on a cleaned mesh.

`acoustic_spaces.geometry.clean_mesh(mesh: RoomMesh) → tuple[RoomMesh, int, int]`

Merge duplicate vertices, drop degenerate faces, copies of a face and the vertices no face uses, split faces at T-junctions and orient the faces alike.

A copy is a face with the same corners as an earlier one, turning the same way: the same face given twice, which would leave every edge of it shared by three faces and the model with no inside. Two faces with the same corners turning opposite ways both stay: that is where two closed models touch, each with its own face.

A face whose edge runs past a corner of the faces on its other side (a T-junction, which CAD exporters leave, or close with a zero-area sliver that is dropped here) is split at that corner, so the faces share the edge and the model can close. Faces that meet along an edge are turned to agree with each other, so a closed model with a few faces flipped by its exporter still has a volume and an inside test that works; each connected part keeps the orientation most of its area had. Returns the cleaned mesh, the number of merged vertices and of removed faces.

`acoustic_spaces.geometry.distance_to_surface(mesh: RoomMesh, points: ndarray) → ndarray`

Distance from each point to the nearest triangle of the mesh (metres).

`acoustic_spaces.geometry.extruded_mesh(plan: ndarray, height: float) → RoomMesh`

A room whose floor is the polygon `plan` (N, 2) at z = 0 and whose ceiling is the same polygon at z = `height`. The plan must be a simple polygon (see `plan_problem()`).

`acoustic_spaces.geometry.ifc_spaces(path: str | Path) → list[tuple[str, str]]`

The rooms (IfcSpace) of an IFC file: for each, what `space` names it by (its name, or its GlobalId when it has none) and how it reads (name and long name).

`acoustic_spaces.geometry.load_mesh(path: str | Path, units: str | None = None, up_axis: str | None = None, *, space: str | None = None) → RoomMesh`

Load a room model and convert it to metres with +z up.

`units` are those of the file's coordinates; None takes the units a DXF, IFC or Collada file records, and metres for other files. `up_axis` overrides the format's convention ("y" for glTF, the axis a Collada file records, "z" otherwise); Blender's OBJ exporter, for example, writes +y up. `space` names a room of an IFC file (an IfcSpace) to take on its own, from its space boundaries; None reads all building elements.

`acoustic_spaces.geometry.mesh_cells(vertices: ndarray, faces: ndarray) → MeshCells`

The cells of the mesh with `vertices` (V, 3) and `faces` (F, 3).

`acoustic_spaces.geometry.missing_extra(path: str | Path) → str | None`

The extra that reading `path` needs (see `READER_EXTRAS`) when it is not installed; None when none is needed or it is installed.

`acoustic_spaces.geometry.plan_problem(plan: ndarray) → str | None`

Why the polygon `plan` (N, 2) cannot be a floor plan, or None: too few corners, two corners in one place, no area, or walls that cross or touch.

`acoustic_spaces.geometry.`**`points_inside`**`(mesh: RoomMesh, points: ndarray) → ndarray`

Whether each point lies inside a room of a closed mesh (any of its rooms, when partitions or openings divide it; see `acoustic_spaces.geometry.cells`).

`acoustic_spaces.geometry.`**`room_of`**`(mesh: RoomMesh, points: ndarray) → ndarray`

The room each point lies in (an index into the rooms of `mesh.cells()`), or -1 outside every room. Generalised winding numbers: the signed solid angles that a room's faces subtend at the point add up to 4 pi inside it and 0 outside.

`acoustic_spaces.geometry.`**`triangle_gaps`**`(one: ndarray, other: ndarray) → ndarray`

The least distance (metres) between each of the triangles `one` (n, 3, 3: the three corners of each) and the one of `other` at its place: nothing where the two touch or cross.

`acoustic_spaces.geometry.`**`triangles_near`**`(triangles: ndarray, points: ndarray, reach: float) → tuple[ndarray, ndarray, ndarray]`

The pairs of one of `points` and one of `triangles` (n, 3, 3: the three corners of each) no farther apart than `reach` metres: the points' indices, the triangles' and the distances (metres), in no order.

Only the points in a triangle's box, grown by the reach, are measured against it: along the axis that leaves the fewest, each box takes the run of the sorted points that its span holds.

Materials

Acoustic material data: the built-in database, coefficient files and the material libraries of CATT-Acoustic and ODEON.

```
class acoustic_spaces.materials.LibraryMaterial(name: str, number: int | None,
    frequencies: tuple[float, ...], absorption: tuple[float, ...], scattering:
    tuple[float, ...] | None = None, category: str | None = None)
```

One material of a library.

name: str

Its name: CATT's material name, or ODEON's description.

number: int | None

ODEON's material number; None in CATT libraries.

frequencies: tuple[float, ...]

Centre frequencies (Hz) of the values.

absorption: tuple[float, ...]

Random-incidence absorption coefficients, from 0 to 1.

scattering: tuple[float, ...] | None = None

Scattering coefficients at the same frequencies, or None.

category: str | None = None

ODEON's category of the material (`{Category=...}`), or None.

property label: str

How the material is named in messages: with its number, if it has one.

```
class acoustic_spaces.materials.MaterialEntry(id: 'str', kind: 'Kind', category: 'str',
    description: 'str', frequencies: 'tuple[float, ...]', coefficients: 'tuple[float,
    ...]', source: 'str')
```

octave_bands() → tuple[list[float], list[int]]

Coefficients on `OCTAVE_BANDS_HZ` and the octaves that had to be extrapolated.

```
acoustic_spaces.materials.band_centres(resolution: Literal['octave', 'third_octave'] =
    'octave') → tuple[int, ...]
```

Nominal centre frequencies (Hz) of the bands at `resolution`.

```
acoustic_spaces.materials.find_material(path: str | Path, entry: str | int) →
    LibraryMaterial
```

The material of a library named `entry`: a name or description (any case), or an ODEON material's number.

```
acoustic_spaces.materials.read_coefficients_csv(path: str | Path) → tuple[list[float],
    list[float], list[float] | None]
```

Return (frequencies, absorption, scattering or None) as read from `path`.

```
acoustic_spaces.materials.read_library(path: str | Path) → list[LibraryMaterial]
```

Every material of a CATT-Acoustic or ODEON library, in file order.

The format is found from the contents: `ABS` definitions make a CATT file, and numbered descriptions followed by a line of coefficients an ODEON one.

```
acoustic_spaces.materials.to_bands(frequencies: Sequence[float], values: Sequence[float],
    resolution: Literal['octave', 'third_octave'] = 'octave') → tuple[list[float],
    list[int]]
```

Map coefficients given at arbitrary centre frequencies onto the bands at `resolution`.

Values whose centre falls inside a band (third-octave data in an octave band, for example) are averaged. A band with no data takes the value of the nearest band that has some; those bands are returned as the second element so callers can say so.

```
acoustic_spaces.materials.to_octave_bands(frequencies: Sequence[float], values:
    Sequence[float]) → tuple[list[float], list[int]]
```

Map coefficients given at arbitrary centre frequencies onto `OCTAVE_BANDS_HZ` (`to_bands()` at octave resolution).

Material libraries of other room acoustics programs: CATT-Acoustic and ODEON.

A scene's material can take its coefficients from an entry of such a library, with `file` naming the library and `entry` the material in it, and `acoustic-spaces materials --file` lists a library's entries.

CATT-Acoustic writes materials in its geometry and library files as `ABS name = <a125 a250 a500 a1k a2k a4k> L <s125 s250 s500 s1k s2k s4k> {r g b}`: absorption and scattering in percent from 125 Hz to 4 kHz, with values at 8 and 16 kHz optional (after a colon, `<a125 ... a4k : a8k a16k>`, or without one). `ABS1` gives them as fractions from 0 to 1 instead. `L <...>` and the colour may be left out, `;` starts a comment, and `ABS name = other` gives a material the coefficients of another in the same file. Scattering that CATT estimates itself (`L <estimate(...)>`) is not read.

ODEON's library (`Material.Li8`) has each material on two lines: its number and description, then eight absorption coefficients from 63 Hz to 8 kHz. ODEON's own library writes the description as `{Product name*=...}, {Category=...}`, and the materials its Material calculator made carry the calculator's settings after the description and, after the eight values, `/Area` and `/A` with the sample area and third-octave values, which are not read: ODEON simulates with the octave values. Libraries of six values (125 Hz to 4 kHz) are read too. ODEON gives scattering to surfaces, not to materials, so these entries have none.

Simulation

Propagation: early specular paths, the late tail, and the responses they make.

The usual entry point is `simulate()`:

```python
from acoustic_spaces.engine import simulate

result = simulate("scene.yaml")
response = result["talker", "seat-a"]
print(response.parameters().table())
ir = response.impulse_response(48_000)
```

`coverage_map()` gives the same parameters on a grid over the audience area.

class acoustic_spaces.engine.**CoverageMap**(source: str, grid: MapGrid, values: dict[str, ndarray], spread: MapSpread | None = None)

Parameters at every map point for one source, or for all of them together.

values: dict[str, ndarray]

Parameter name (see `MAP_PARAMETERS`) -> (points, bands) or (points,).

spread: MapSpread | None = None

Their uncertainty from the materials' absorption, where the map was run with its spread (`coverage_map()`).

parameters() → list[str]

The parameters the map has, in the order of `MAP_PARAMETERS`: those of `OUTDOOR_MAP_PARAMETERS` outdoors only, those of `INSERTION_LOSS_MAP_PARAMETERS` when the map was also run without some of the scene's buildings, barriers or vegetation, those of `LONG_TERM_MAP_PARAMETERS` when it was run in weather classes, and those of `LONG_TERM_INSERTION_LOSS_MAP_PARAMETERS` when it was run both ways.

value(name: str, band: int | None = None) → ndarray

One value per point: a parameter in one band (index into `OCTAVE_BANDS_HZ`), or a parameter that has no bands.

spread_of(name: str, part: str = 'standard', band: int | None = None) → ndarray

One value per point of the spread of parameter `name` (as `value()` takes it): its `standard` uncertainty; the `low` or the `high` end of the interval that holds it with the spread's coverage probability; or the value it is `at_least` or `at_most` with the spread's one-sided probability. The interval and the bounds take the result as normal about the map's value (GUM, JCGM 100:2008, 6.3 and G.2): the value less and plus the normal quantile times the standard uncertainty (1.96 for 95 %; 1.645 for a one-sided 95 %), held to what the parameter can be (`LIMITS`).

as_dict() → dict

Points and values as plain lists, None where undefined (JSON friendly); with its spread, `uncertainty` holds the probabilities, each parameter's `standard` uncertainty and each material's part of it (`materials`).

class acoustic_spaces.engine.**CoverageResult**(room: ~acoustic_spaces.room.Room, settings: ~acoustic_spaces.scene.model.Coverage, grid: ~acoustic_spaces.engine.coverage.MapGrid, maps: dict[str, ~acoustic_spaces.engine.coverage.CoverageMap], combined:

```
~acoustic_spaces.engine.coverage.CoverageMap | None = None, details: dict[str,
dict[str, float]] = <factory>, removed: tuple[str, ...] = ())
```

Coverage maps of every source, and of all of them together.

combined: `CoverageMap | None = None`

All sources playing together at their levels; None with one source.

removed: `tuple[str, ...] = ()`

The buildings, barriers and vegetation the map was also run without, for the insertion loss (`coverage_map()`).

parameters() → `list[str]`

The parameters of its maps (see `CoverageMap.parameters()`).

get(source: str | None = None) → `CoverageMap`

The map of one source; None gives all sources together (or the only one).

as_dict() → `dict`

The settings, the points and every map as plain lists (JSON friendly).

```
class acoustic_spaces.engine.EarlyReflections(order: ndarray, delay_s: ndarray,
    distance_m: ndarray, direction: ndarray, energy: ndarray, points: list[ndarray] =
    <factory>, faces: list[ndarray] = <factory>, edge: ndarray | None = None, crossings:
    ndarray | None = None, spectrum: ndarray | None = None, point: ndarray | None =
    None, wander: Wander | None = None, lag: ndarray | None = None, cosines:
    list[ndarray] = <factory>, own_phase: ndarray | None = None, fine: ndarray | None =
    None, fine_delay: ndarray | None = None)
```

Paths from one source to one listener: specular (order 0 is the direct sound), and diffracted at an edge when edge diffraction is on.

`edge``: ndarray | None = None`

(n,) the edge a path bends over (an index of the model's diffracting edges, see `find_edges()`), -1 for specular paths, -2 for a path over the crests of the terrain (`crests`), or -3 for the path that carries the sound of the solver for whole rows (`row_paths`).

`crossings``: ndarray | None = None`

(n,) how many surfaces that let sound through a path passes, openings aside.

`spectrum``: ndarray | None = None`

(n, C, M) in the lowest C bands, where the paths add in pressure (see `coherent`): each path's pressure at M frequencies in each band (`sample_frequencies()`), relative to its delay and to its level in the band, which its energy gives (the mean of its squared magnitude over a band's frequencies is 1). None when the paths add in energy in every band.

`point``: ndarray | None = None`

(n,) which of the uncorrelated points of a line or surface source each path comes from (outdoors, where those paths carry a spectrum): in the coherent bands the paths of one point add in pressure, those of different points in energy.

`wander``: Wander | None = None`

Outdoors in turbulence, how the eddies delay the paths at random (`Wander`): on average two paths' pressures add with their coherence factor. None without turbulence, and at a moment of it (`frozen()`).

`lag``: ndarray | None = None`

(n, bands) at a moment of turbulence, each path's delay (s) in each band beyond `delay_s`, its delay in the highest band (the eddies part paths near the ground less at low frequencies); None otherwise.

`cosines``: list[ndarray]`

Where the paths carry a `spectrum`: for each path, the cosine of the angle at which it meets the face of each of its reflections (as `faces`), whose phase its spectrum takes from that face's absorption (`reflection_phases()`), so that re-weighting for other absorption can turn it (`turned_paths()`); NaN for a reflection whose phase comes from elsewhere (a ground's impedance). Empty without a spectrum.

`own_phase``: ndarray | None = None`

(n,) in the bands above the coherent ones, the phase factor a rendered response gives each path beyond the one held from those bands (`with_own_phases()`); None outside one.

`fine``: ndarray | None = None`

(n, C, K) where a path carries the field the parabolic equation gives over the open ground (outdoors in wind or temperature gradients, out of the shadow): in each coherent band its pressure, unnormalised, relative to its delay, at the frequencies of the layout (`fine_frequencies()`) the equation ran at there (in the lowest band down to an octave below it, in its top band through the upper crossover too where a response is rendered or the band filters read it); NaN where a path has none. Its band's shape is drawn from these and its neighbours' (`shaped()`); its level stays the one its energy gives. None when no path carries one.

`fine_delay: ndarray | None = None`

(n,) the delay (s) past `delay_s` whose phase `fine`'s samples follow on average (`reference_delay()`), relative to which they are interpolated; 0 for paths without them. None with `fine`.

`property coherent_bands: int`

How many of the lowest bands the paths add in pressure in.

`frozen(seed: int) → EarlyReflections`

The paths at a moment of the turbulent air drawn with `seed`: each delayed in each band as the eddies have it then, adding in pressure (the same paths without turbulence).

`wander_roots() → list`

How the eddies delay the paths, band by band, as `roots()` (empty without turbulence).

`at_moment(roots: list, draws: list[ndarray]) → EarlyReflections`

The paths delayed as `delays()` has it for the normal variables `draws` (one array per group of `roots`, `wander_roots()`).

`property diffracted: ndarray`

(n,) True for paths that bend over an edge or the terrain's crests, and for the one that carries the sound of the solver for whole rows.

`sorted() → EarlyReflections`

The same paths in order of arrival.

```
class acoustic_spaces.engine.LateEnergy(energy: ndarray, time_step: float, escaped_rays:
    int = 0, rays: int = 0, traced_rays: int = 0, directions: ndarray = <factory>,
    directional: ndarray = <factory>, direction_step: float = 0.01, moments: ndarray =
    <factory>, moment_step: float = 0.0, plan: Plan | None = None, batches: ndarray =
    <factory>, batch_step: float = 0.01)
```

Energy per listener, band and time slot (time-integrated intensity).

traced_rays: int = 0

Rays traced, counting those split off (stochastic ray tracing only).

directions: ndarray

Arrival directions (room coordinates, towards where the energy comes from) and the same energy split between them (listeners, directions, bands, slots of `direction_step`); empty without directions.

moments: ndarray

With face groups (`stochastic_energy(groups=...)`): per listener, the energy in slots of `moment_step` times 1, the reflections the rays had off each group n_i, and n_i n_j (i <= j) (listeners, terms, bands, slots), for re-weighting the response for new absorption (`acoustic_spaces.engine.reweight`); else empty.

plan: Plan | None = None

Where the rays were traced (stochastic ray tracing only).

batches: ndarray

With `stochastic_energy(batches=...)` of 2 or more: `energy` as each batch of rays gives it (listeners, batches, bands, slots of `batch_step`), each an estimate of its own and their mean `energy`; else without batches.

class acoustic_spaces.engine.**MapGrid**(origin: tuple[float, float], cell: float, index: ndarray, points: ndarray, columns: ndarray, surface: ndarray, low: ndarray, high: ndarray, volume: ndarray, footprint: ndarray, cover: ndarray, room: ndarray, level: ndarray)

Map points over the audience area: columns `cell` metres wide from the corner `origin` (x, y), each holding up to `index.shape[2]` points.

`index``: ndarray`

(ny, nx, slots) the points of each column, by level and then by room, -1 for none.

`points``: ndarray`

(n, 3) the points, at listening height above the column's centre, or above the middle of their part of the column when the centre is not in it.

`columns``: ndarray`

(n, 2) column (x, y) of each point.

`surface``: ndarray`

(n,) height of the surface under each point (m).

`low``: ndarray`

(n,) bottom and top of each point's slab of air (m), and the volume of the slab in the air of the point's room, outside buildings (m^3).

`footprint``: ndarray`

(m, 3, 2) the surfaces under the map seen from above: x, y of their triangles.

`cover``: ndarray`

(n, 25) which fifths of the column, by x then y, lie over the surfaces (in the point's room, where the model holds several).

`room``: ndarray`

(n,) the room each point lies in (an index into `Room.room_volumes`) where the model holds several, else -1. A point's slab counts the rays in its room alone.

`level``: ndarray`

(n,) the level of each point in its column, from 0 at the lowest surface under it (the stalls; a balcony above them is 1). Where a wall between rooms crosses a column, it holds a point on each side that holds at least a fifth of it, on the same level.

`squares()` `→ ndarray`

(n, 4, 3) corners of each point's column, at the point's height.

`tiles()` `→ tuple[ndarray, ndarray]`

Squares at the points' height over the part of each column above the surfaces (in the point's room): (m, 4, 3) corners and the point each belongs to. A column only partly over them, or split between rooms, is cut into fifths along each side.

`class` `acoustic_spaces.engine.`**`Particles`**`(times: ndarray, positions: ndarray, energy: ndarray, order: ndarray, band_hz: int, source: str = '')`

Positions of sound particles at a series of times.

energy: ndarray

Energy relative to a particle’s share at emission (frames, particles).

order: ndarray

Reflections so far (frames, particles).

source: str = ''

Name of the source the particles left.

```
class acoustic_spaces.engine.Response(source: str, listener: str, early:
    EarlyReflections, late: ndarray, time_step: float, length_s: float, direct_delay_s:
    float, transition_s: float, volume_m3: float | None, speed_of_sound: float, seed:
    int = 0, orientation: ndarray = <factory>, late_directions: ndarray = <factory>,
    late_directional: ndarray = <factory>, direction_step: float = 0.01, wave:
    WaveResponse | None = None, source_direction: ndarray | None = None, feed: ndarray |
    None = None, feed_delay_s: float = 0.0, late_moments: ndarray = <factory>,
    moment_step: float = 0.0, late_batches: ndarray = <factory>, batch_step: float =
    0.01, free_field: ndarray | None = None)
```

Early specular paths plus late energy per band and time slot, and the lowest bands from the wave solver when it ran (`wave`), which then replace the rays'. The bands are the 8 octave bands or the 24 third-octave bands the simulation ran in; room parameters are given in octave bands either way.

Energies are time-integrated intensity for a source of unit energy (the direct sound at r metres carries 1 / (4 pi r^2)); times are measured from emission.

The late part's arrays (`late`, `late_directional`, `late_moments` and `late_batches`) may each be given as a `Deferred`: the response then keeps that, and reading the field makes the array, which a read of the field again gets until that of another response is made.

`direct_delay_s`: `float`

Arrival time of the direct sound (s), whether or not the direct path is blocked.

`transition_s`: `float`

Early paths end here (s, from emission); the late part covers everything else.

`volume_m3`: `float | None`

Volume of the listener's room (m3; None in a model that is not closed): how densely the synthesized late part's reflections come.

`seed`: `int = 0`

Seed for the noise of the synthesized late part.

`orientation`: `ndarray`

The listener's head axes (forward, left, up) as columns, in room coordinates.

`late_directions`: `ndarray`

Arrival directions of the late energy (room coordinates) and the late energy split between them (directions, bands, slots of `direction_step`). Without them the late part arrives evenly from all directions.

`wave`: `WaveResponse | None = None`

The lowest octave bands from the wave solver, or None.

`source_direction`: `ndarray | None = None`

Unit vector from the listener towards the source (room coordinates); None takes the direct path's, or the listener's forward axis without one.

`feed`: `ndarray | None = None`

The energy gain per band of the source's feed (its gain and equalizer), which the energies include, or None for none. G is taken against the same source in free field, so the feed does not change it.

`feed_delay_s`: `float = 0.0`

How much later the feed makes every arrival (s), which the times include.

`late_moments`: `ndarray`

With face groups (`simulate(groups=...)`): the moments of the late energy's reflection counts off each group (terms, bands, slots of `moment_step`), for `reweighted()`; else empty.

`late_batches``: ndarray`

With the late rays traced in batches (`simulation.late.batches` of 2 or more): `late` as each batch gives it (batches, bands, slots of `batch_step`, a whole number of `time_step`), each an estimate of its own and their mean `late`, for `standard_errors()`; else empty.

`free_field``: ndarray | None = None`

Outdoors: the energy per band of the same source at the listener in free field (its direct sound over the straight line, with the air's attenuation and the source's directivity, without the feed), for `relative_to_free_field()`; else None.

`property coherent_bands``: int`

How many of the lowest bands the early paths add in pressure in (`coherent`); the wave solver's take the place of those it covers.

`property band_hz``: tuple[int, ...]`

Centre frequencies (Hz) of the bands: octave or third-octave bands.

`energy``(time_step: float | None = None) → ndarray`

Echogram: energy per band and time slot (bands, slots), early paths included.

`octave_energy``(time_step: float | None = None) → ndarray`

Echogram in octave bands (8, slots): each octave the mean of its three third-octave bands, when the simulation ran in those.

`weighted_energy``(axis: ndarray, exponent: int = 2, time_step: float | None = None) → ndarray`

Echogram of a figure of eight facing `axis` (a unit vector, room coordinates): each arrival's energy times the cosine of its angle to the axis, taken without its sign, to the power `exponent`. With 2 that is the energy the figure of eight picks up; with 1 the magnitude of its product with the pressure, as LFC takes it. Late energy without directions counts as arriving evenly from all of them.

`lateral_axis``() → ndarray`

The axis of a figure of eight with its null on the source (ISO 3382-1): level, across the direction of the source, towards the listener's left side of it.

`fed``(gain_db: Sequence[float] | ndarray, delay_s: float = 0.0) → Response`

This response with the source's feed: `gain_db` per band (dB, the bands of the response) on every energy, and every arrival `delay_s` later.

`reweighted``(face_group: np.ndarray, ratio: np.ndarray, absorption: tuple[np.ndarray, np.ndarray] | None = None, admittance_of=None, redraws: Redraws | None = None) → Response`

This response for other absorption, without tracing again (`acoustic_spaces.engine.reweight`): `face_group` the group of each face (-1 for none) as traced, `ratio` the change per reflection off each group, ln((1 - new) / (1 - old)) per group and band (`log_ratio()`), and `absorption` each group's absorption (groups, bands) as traced and as `ratio` changes it, (old, new). The early paths re-weight exactly (but for reflections off a ground with an impedance or with zones, whose factor is not 1 - absorption: see the module), the late part from its moments, which the response needs (`simulate(groups=...)`). Where the early paths add in pressure (`coherent`), the phase of their reflections depends on the absorption too, and turns with it (`turned_paths()`, with `admittance_of` if given): that needs `absorption`, unless no path reflects off a group whose absorption changes. `redraws` (a `Redraws` of this response's early paths), for one of many draws of the absorption, holds what does not depend on the draw; the response is the same with it as without it.

`parameters``(energy: ndarray | None = None)` → **`RoomParameters`**

ISO 3382-1 parameters per octave band, straight from the energies.

Those of a figure of eight with its null on the source (LF, LFC, LJ) and the decay times of figures of eight along the listener's axes come from the directions the energy arrives from; stage support only for a listener 0.9 to 1.1 m from the source (`STAGE_DISTANCE_M` and `STAGE_TOLERANCE_M`), as the standard measures it 1 m away. IACC needs a binaural response: `interaural_cross_correlation()`. Outdoors (with `free_field`), G of a band that does not decay as a room's does is that of the energy the response holds. `energy`: its `octave_energy()`, when already at hand.

`standard_errors``()` → **`RoomParameters`** | `None`

The standard error of each of `parameters()` from the late rays' batches (`simulation.late.batches`): one standard deviation of how much it would change between runs with other random seeds and as many rays. A jackknife: the parameters again without each of the n batches in turn, whose spread, times (n - 1) / n under the root, is the standard error. The early paths and the wave solver's bands do not depend on the seed, and add none. Those from the directions the late energy arrives from (LF, LFC, LJ and the decay times along the listener's axes) are NaN, as IACC is; so is a parameter some batches leave undefined or infinite (a clarity without late energy). None without 2 batches or more.

The batches hold the energy in slots of `batch_step`; without a batch, each slot of `time_step` keeps its share of its slot of `batch_step`, so a strong arrival in one of them (a ray through the listener long after the rest, in a room whose walls keep a few going) leaves with the batch it came in.

`relative_to_free_field``()` → `ndarray`

The level relative to free field per octave band (dB): how much louder (or, negative, quieter) the source is here than the same source the same distance away in free field, which the air between them attenuates as here. The ground, the weather, buildings, barriers and vegetation make the difference; with the sign turned, it is the excess attenuation of outdoor acoustics. It compares the energy the response holds (nothing is extrapolated past its end, as G does for a room's decay) with the free field's; -inf where no sound arrives, NaN in every band of a response without `free_field`.

levels(level_db: float | Sequence[float] = 94.0, parameters: RoomParameters | None = None) → ndarray

Sound pressure level per octave band (dB SPL) of a source of sound power `level_db` (dB re 1 pW, the feed's gain included): one value spread evenly over the octave bands, or one per octave band (`power_db()`). It is the band's sound power plus its G, less 31 dB, the level of a source of 0 dB at 10 m in free field.

levels_or_held(level_db: float | Sequence[float] = 94.0, parameters: RoomParameters | None = None) → ndarray

`levels()`, but in a band whose G is undefined (the few rays that reach a far place may carry no energy in it, or too little of its decay to read), the level of the energy the response holds from the direct sound's arrival, as outdoors: -inf where none arrives. The level a listener is given in `simulate`'s report, the editor's results, the demos' captions and each place of `pass_by()`; the coverage maps and `speech()` leave such a band undefined.

speech(level_db: float | Sequence[float] = 94.0, noise_db: float | Sequence[float] | None = None, parameters: RoomParameters | None = None, talker: Literal['male', 'female'] = 'male') → SpeechParameters

STI, STIPA and U50 for speech from a source of sound power `level_db` (dB re 1 pW: in all, spread evenly over the octave bands, or per octave band), with background noise `noise_db` (dB SPL, one value or one per band), as the coverage maps take them, weighted for a male or female `talker`; the STI and STIPA are NaN when a band from 125 Hz up has no level (its G is undefined). `parameters` are this response's, when already at hand.

levels_and_speech(level_db: float | Sequence[float] = 94.0, noise_db: float | Sequence[float] | None = None, talker: Literal['male', 'female'] = 'male', parameters: RoomParameters | None = None, energy: ndarray | None = None) → Levels

The levels and speech results at the listener for a source of sound power `level_db` (dB re 1 pW, in all or per octave band), as a run's report gives them: `levels_or_held()` and its A-weighted sum; the STI, STIPA and U50 of `speech()` with background noise `noise_db` for `talker`; and `relative_to_free_field()`, with its A-weighted value (the A-weighted level less that of the same source in free field). `parameters` are this response's, and `energy` its `octave_energy()`, when already at hand.

level_errors(level_db: float | Sequence[float] = 94.0, noise_db: float | Sequence[float] | None = None, talker: Literal['male', 'female'] = 'male') → Levels | None

The standard error of each of `levels_and_speech()` from the late rays' batches, by the jackknife of `standard_errors()`: NaN where some batch leaves the value undefined or infinite, None without 2 batches or more. Where G is defined, a band's level is its G and the source's power, so its error is G's; where it is not, the level is that of the energy held (`levels_or_held()`) and has an error of its own.

impulse_response(sample_rate: int = 48000, seed: int | None = None) → ndarray

Mono impulse response; the direct sound at r metres is an impulse of height 1 / r.

Early paths become fractional-delay impulses filtered per band, in the lowest bands with the phase of their reflections (`coherent`), in turbulence as at one moment (`moment()`); the late energy drives a random-sign Dirac sequence with the density of reflections in the room. From 50 ms after the direct sound the late energy is smoothed over 10 ms first, so a single strong ray late in the tail does not become a click. The wave solver's bands, when it ran, take the place of the lowest ones: through the crossover the early paths make way for them in pressure and the noise, which does not interfere with them, in energy (`WaveResponse.high_pass`).

moment(seed: int | None = None) → EarlyReflections

The early paths at a moment of the turbulent air, drawn with `seed` (default the response's): outdoors in turbulence the eddies delay each path at random, and an impulse response is one moment of them, whose levels scatter about the echogram's average. Without turbulence, the early paths.

spatial_impulse_response(fmt, sample_rate: int = 48000, seed: int | None = None, *, threads: int = 1) → ndarray

Impulse responses (samples, channels) in a format of `acoustic_spaces.spatial`: `Ambisonics(order)`, `Loudspeakers.layout(name)`, `Binaural(hrtf)` or `Mono()`, its channels rendered on `threads` threads (the same result with any number; at most the computer's cores).

class acoustic_spaces.engine.**SimulationResult**(room: Room, settings: SimulationSettings, responses: dict[tuple[str, str], Response], details: dict[str, dict[str, float]] = <factory>, removed: tuple[str, ...] = (), insertion_loss: dict[tuple[str, str], np.ndarray] = <factory>, long_term: LongTermLevels | None = None, spreads: dict = <factory>)

Responses for every (source, listener) pair, plus run details.

details: dict[str, dict[str, float]]

Per source: rays, escaped rays, receiver radius, seed and timings.

removed: tuple[str, ...] = ()

The buildings, barriers and vegetation of an outdoor scene that it was also run without (`simulate()`), and their insertion loss per (source, listener) in each octave band (dB): the level without them less the level with them. Empty otherwise.

long_term: LongTermLevels | None = None

The levels over a long term, when the scene has weather classes (`long_term`; `simulate()`): the responses are the scene's own weather's, this holds each class's run and the levels combined. None otherwise.

spreads: dict

The spread of each response's parameters from its materials' absorption uncertainty, by (source, listener) (`material_spread()`), where it was drawn. Empty otherwise.

parameters() → dict[tuple[str, str], RoomParameters]

ISO 3382-1 parameters of every response.

class acoustic_spaces.engine.**TraceModel**(room: 'Room', bvh: 'object', face_plane:

```
'np.ndarray', plane_normal: 'np.ndarray', plane_offset: 'np.ndarray', oriented:
'bool', absorption: 'np.ndarray', scattering: 'np.ndarray', plane_two_sided:
'np.ndarray | None' = None, transmission: 'np.ndarray | None' = None, face_room:
'np.ndarray | None' = None, room_volumes: 'np.ndarray | None' = None, surface_size:
'np.ndarray | None' = None, vector_scattering: 'bool' = True, by_band: 'bool' =
True, bands: 'Resolution' = 'octave', size_from_order: 'int' = 1, ray_tree: 'str' =
'sah', sky: 'np.ndarray | None' = None, ground: 'float | None' = None,
face_impedance: 'np.ndarray | None' = None, admittance: 'np.ndarray | None' = None,
grazing_admittance: 'np.ndarray | None' = None, impedance_extended: 'np.ndarray |
None' = None, impedances: 'tuple' = (), ground_part: 'np.ndarray | None' = None,
part_face: 'np.ndarray | None' = None, plants: 'object | None' = None, deciding:
'np.ndarray | None' = None)
```

face_plane: ndarray

(F,) plane index of every face.

plane_normal: ndarray

(P, 3) unit plane normals, pointing into the room when `oriented` (and the plane is not two-sided).

plane_offset: ndarray

(P,) plane offsets: points x on plane p satisfy normal[p] . x == offset[p].

oriented: bool

True when the rooms are known (a closed model): each face's normal then points into the room on its side.

absorption: ndarray

(F, bands) random-incidence absorption and scattering coefficients.

plane_two_sided: ndarray | None = None

(P,) planes with a room on both sides of some face: a partition between two rooms or a free-standing panel; every plane when the model is not `oriented`.

transmission: ndarray | None = None

(F, bands) share of sound each face lets through (see `acoustic_spaces.scene.Material.transmission_coefficients()`), or None.

face_room: ndarray | None = None

(F, 2) the room on the front (the side the mesh's face normal points to) and the back of each face (an index into `room_volumes`), or -1.

room_volumes: ndarray | None = None

(R,) volume of each room (m3).

surface_size: ndarray | None = None

(F, 2) width and length (m) of the surface each face belongs to (see `surface_sizes()`), when the scattering of a surface's size is added; else None.

vector_scattering: bool = True

Vector-based scattering (the scene's `simulation.scattering.method`).

by_band: bool = True

Each band traced with rays of its own where the bands scatter differently (`simulation.scattering.by_band`).

bands: Literal['octave', 'third_octave'] = 'octave'

The bands the coefficients (and so every energy traced) are in.

size_from_order: int = 1

Reflections of this order and higher add the scattering of the surface's size. Below it, the early stage's edge diffraction accounts for the size of surfaces: its terms add coherently to the direct sound and the reflections off the faces of an edge (with `direct` diffraction), and to the reflections of first-order reflections off them too (`first_order`), so those waves keep their full specular share.

ray_tree: str = 'sah'

How `bvh` was split (`simulation.performance.ray_tree`); trees built from this model's mesh later take the same.

sky: ndarray | None = None

(F,) the faces of an outdoor scene's open sky, whose edges diffract nothing; None in a room.

ground: float | None = None

Height (z) of an outdoor scene's ground, whose edges on it diffract nothing; None in a room.

face_impedance: ndarray | None = None

(F,) for faces whose material has an impedance (a ground's: see `Impedance`), its row of `admittance`; -1 for the others. None when no material has one.

admittance: ndarray | None = None

(K, bands, M) the normalised admittance of each material with an impedance at the frequencies the early paths are evaluated at (`band_frequencies()`), at normal incidence (a locally reacting ground's at every angle).

grazing_admittance: ndarray | None = None

(K, bands, M) the same at grazing incidence, which differs for grounds of extended reaction (`impedance.reaction`), for the parabolic equation.

impedance_extended: ndarray | None = None

(K,) whether each material with an impedance reacts extendedly: its admittance changes with the angle of incidence, and `impedances[k]` gives it.

impedances: tuple = ()

The materials' impedances (`Impedance`), row by row.

ground_part: ndarray | None = None

(F,) for an outdoor scene whose ground has zones (`geometry.ground.zones`), the part of the ground each face of it lies in (0 outside the zones, then each zone), whose reflections share their Fresnel zone with the parts round them; -1 for the faces that are not the ground. None without zones.

`part_face``: ndarray | None = None`

(parts,) a face of each part of the ground, whose material it has; -1 for a part with no faces (the zones may cover all the ground).

`plants``: object | None = None`

An outdoor scene's vegetation in `bands` (`Plants`); None without any.

`deciding``: ndarray | None = None`

(F, bands) the absorption that decides for a map's rays in place of `absorption`, wherever the energy a ray carries decides what happens to it (the splitting and thinning out, whether it passes a partition): the run then follows the rays of the run with this absorption (`stochastic_map_energy()`). None: its own.

`classmethod` **`from_room`**`(room: Room, settings=None) → TraceModel`

The model of `room` with the simulation `settings` (default: the scene's).

`property` **`transparent`**`: ndarray`

(F,) faces that let all sound through in every band: openings (worked out once, as the early stage asks for every reflection point it places).

`property` **`height_error`**`: float`

The rounding (m) allowed for in a height over a plane worked out from the model's coordinates: 16 units in the last place of the largest of them.

`property` **`face_spread`**`: ndarray`

(F,) the largest distance (m) of a vertex of each face from its plane (`face_plane`); 0 where all lie within `height_error` of it. Faces whose normals and offsets round to the same multiples of `PLANE_TOLERANCE` make one plane, the plane of the first of them, without lying in it exactly (worked out once).

`property` **`plane_spread`**`: ndarray`

(P,) the largest `face_spread` of each plane's faces (worked out once).

`property` **`face_two_sided`**`: ndarray`

(F,) the faces with a room on both sides: a partition between two rooms or a free-standing panel; every face when the model is not `oriented`. A wall with a room on one side has one side, also where its plane holds such a face (`plane_two_sided`).

`property` **`geometric_plane`**`: ndarray`

(P,) for each plane the first of the planes that lie where it does, whichever way they face: two walls with a room on one side in line with each other that face opposite ways (the wall between two rooms drawn once for each, the two sides of a step in a room's plan) are two planes in one geometric plane, and a point has one image in both (worked out once); a partition or a panel in line with them lies in the plane of one of them. Two planes lie in one place where their normals and offsets, in multiples of `PLANE_TOLERANCE` and with the signs of one plane's turned where it faces the other way, round to numbers at most one apart: always where they differ by less than that tolerance in each, on whichever side of a rounding step either falls. So do two planes that each lie where a third does.

property **face_planes**: tuple[ndarray, ndarray]

(F, 3) unit normals and (F,) offsets of the plane each face itself lies in, the normal facing the way of its plane's; its plane's own where its `face_spread` is 0 (worked out once).

property **face_grids**: dict

The faces of planes whose faces do not all lie in them, binned in squares by `_face_grid` of `acoustic_spaces.engine.early`, by plane and distance allowed (each worked out when first asked for).

property **partition**: ndarray

(F,) partitions: faces that let some sound through and reflect some, so a path that meets one goes on both ways. Openings reflect nothing, and neither does a face that absorbs all the sound it does not let through.

in_front(distance: ndarray, tolerance: float) → ndarray

Whether points at signed `distance` (…, P) from every plane, farther than `tolerance`, can see it: in front of it, or on either side of a two-sided plane.

passes(origins: ndarray, directions: ndarray, lengths: ndarray, ignore: ndarray, transparent_only: bool = False, counted: ndarray | None = None) → tuple[ndarray, ndarray]

(n, bands) share of sound that travels from each origin `lengths` along its direction (unit): the product of the transmission of the faces it crosses, 0 where a face lets nothing through; and (n,) how many of those faces it crosses that are not openings, or (n, m) how many of each column of `counted` (F, m). With `transparent_only`, only openings let sound through.

property **band_hz**: tuple[int, ...]

Centre frequencies (Hz) of the model's bands.

mirror(points: ndarray, planes: ndarray) → ndarray

Mirror each point across the plane with the same row index.

impedance_faces(faces: ndarray) → ndarray

(n,) whether each of `faces` has a material with an impedance.

factor_faces(faces: ndarray) → ndarray

(n,) whether reflections off each of `faces` take a reflection factor that depends on the path (see `ground_factors()`): faces with an impedance, and the ground of a scene with ground zones.

reflection(faces: ndarray, incoming: ndarray, order: int, impedance: bool = False) → ndarray

(n, bands) specular energy share of reflections off `faces` (n,) of sound that travelled `incoming` (n, 3) from the source or the previous reflection, the `order`-th reflection of its path: (1 - absorption) (1 - scattering), with the scattering of the surface's size when that applies, as the stochastic tracer reflects its rays. With `impedance`, the faces of `factor_faces()` give (1 - scattering) alone: their reflection factor, which depends on the path, takes the place of the absorption.

acoustic_spaces.engine.**coverage_map**(room: Room | Scene | str | Path, *, settings: Coverage | None = None, simulation: Simulation | None = None, sources: Sequence[str] | None = None, threads: int | None = None, progress: Callable[[str], None] | None = None, grid: MapGrid | None = None, rays_of: Scene | None = None) → CoverageResult

Coverage maps of the scene's sources (all by default) over its audience area.

`settings` default to `scene.coverage` (surfaces, height, spacing, rays and background noise); `simulation` (default: `scene.simulation`) gives the early order, late method, response length, seed and threads (`threads` overrides them) as for `simulate()`. Each source radiates its sound power per band (`power_db()`: its level, spectrum, gain and equalizer), which the speech transmission index takes as the speech level, weighted for female speech when the source's spectrum is `speech_female`. In the map of all sources together, each source's sound arrives `delay_ms` later. Outdoors, every point also has its level relative to free field (`re_free_field`).

Maps do not run the wave solver: whatever `simulation.wave` says, the lowest bands come from the early paths, in pressure up to `early.coherent_max_band_hz`, and the rays.

`grid` takes the place of the one the settings give, so that two runs of a scene share their points (`coverage_map()`); the rooms its points lie in and the air in their slabs are those of `room`.

`rays_of` is the scene as it was before its materials' absorption was moved: with the same seed, the map then follows the rays of that scene's map, whose materials' absorption decides for them (`deciding`), and counts on them the energy this scene's absorption leaves. The two maps differ by what the absorption does, and not by their rays (`sensitivity`).

acoustic_spaces.engine.**early_paths**(room: Room | Scene | str | Path, source: str | None = None, listener: str | None = None, *, max_order: int | None = None, method: str | None = None, diffraction: str | None = None, threads: int | None = None) → EarlyReflections

Specular and diffracted paths from one source to one listener, without the late part: those a simulation keeps as early paths, up to `scene.simulation`'s `transition_ms` after the direct sound (the mixing time with the statistical late method) and within `ir_length_s`.

Names default to the first source and listener; `max_order`, `method` and `diffraction` default to `scene.simulation.early`, `threads` to `scene.simulation.performance.threads`. Useful for drawing paths. The room's checks, with which a simulation refuses to run, are not made (`problems()`): a listener nearer a surface than a ray can meet it, say, may lack the reflection off it here.

acoustic_spaces.engine.**map_grid**(room: Room, surfaces: Sequence[str] | None = None, *, height: float = 1.2, spacing: float = 1.0) → MapGrid

Map points `spacing` metres apart, `height` metres above the named surfaces (default: `audience_surfaces()`). Only faces that can be walked on count, not walls, and only points inside the room are kept (outdoors, in the open air: not inside a building). Where the model holds several rooms, a column that a wall between them crosses gets a point in each that holds at least a fifth of it.

`acoustic_spaces.engine.`**`mixing_time`**`(volume_m3: float | ndarray | None) → float | ndarray`

Time after the direct sound when the late field is diffuse (s): sqrt(V) ms, V the volume of the room the listener is in (one per listener in an array).

`acoustic_spaces.engine.`**`simulate`**`(room: Room | Scene | str | Path, *, settings: Simulation | None = None, sources: Sequence[str] | None = None, listeners: Sequence[str] | None = None, threads: int | None = None, progress: Callable[[str], None] | None = None, groups: ndarray | None = None, late_from: Mapping[str, Mapping[str, float]] | None = None, share_waves: Collection[str] | None = None) → SimulationResult`

Simulate the room's impulse responses as set in `scene.simulation`.

`room` may be a `Room`, a `Scene` or a scene file. `sources` and `listeners` pick names from the scene, simulated even when left out (`enabled: false`); by default, those in use. `threads` overrides `simulation.performance.threads` (0 uses every core). Raises `ValueError` if `Room.problems()` reports anything (it checks the sources and listeners in use and those named; every source, or every listener, left out is a problem when they are not named), if a name is not the scene's, or if there is no listener to simulate.

`groups` (the group of each face, or -1; `face_groups()`) keeps in each response the moments of the late part's reflections off each group, so that `Response.reweighted()` gives it for other absorption (stochastic ray tracing only).

`late_from` gives listeners or sources the late part of others, mixed: `{name: {other: weight}}`, the others being listeners (for a listener) or sources (for a source) that keep their own. Only those are traced by the stochastic ray tracer, by far the costliest stage; the statistical method gives everyone their own. Walks use it (`acoustic_spaces.walkthrough.simulate_walk()`): the late field changes slowly from place to place, the early sound does not. The traced late parts are kept once, and a response makes its mix of them when it reads its late part (`Deferred`); only a line's or a surface's is mixed for every listener as it is traced.

`share_waves` names sources whose runs of the parabolic equation (outdoors in wind or temperature gradients) may serve one another: those at one height over one ground, the sound going the same way to their listeners, where the weather and the grounds have no zones (`solve()`), each listener getting what its own source's run would give it. Walks give it the places of a walking source, which would otherwise each run the equation anew.

`acoustic_spaces.engine.`**`trace_particles`**`(room: Room, source: str | None = None, *, count: int = 4000, duration: float = 0.1, frames: int = 100, band: int = 4, seed: int = 0) → Particles`

Trace `count` particles from `source` (default: the first) for `duration` seconds and sample them at `frames` evenly spaced times from 0 to `duration`.

`band` picks the octave band whose absorption and scattering apply (4 = 1 kHz).

Coverage maps

`acoustic_spaces.engine.coverage_map()` and the classes it returns are listed under Simulation above; the module also has the parameters a map shows and the A-weighting.

Coverage maps: room acoustic parameters on a grid over the audience area.

The audience surfaces, picked by name, are covered with square columns `spacing` metres wide. Each column holds a map point at listening `height` above the surface (two or more where surfaces lie above each other, such as a balcony over the stalls, or where a wall between rooms crosses it), with a slab of air around it. For every source:

- the direct sound and the specular reflections up to `simulation.early.max_order` are found exactly at each point, as at a listener;
- everything else comes from one stochastic ray tracing run in which every leg of every ray adds its track length through each slab it crosses (`stochastic_map_energy()`), so a map costs about as much as one listener; with the statistical late method, the exponential decay of `statistical_energy()`, as at a listener.

Each point's parameters follow from its energy per band and time slot as at a listener (ISO 3382-1), together with the speech transmission index (IEC 60268-16) and sound pressure levels from the sources' sound power.

```
acoustic_spaces.engine.coverage.MAP_PARAMETERS: dict[str, MapParameter] = {'c50': ('C50
    (dB)', 'clarity, 50 ms', True, 1.0, '{:.1f}'), 'c80': ('C80 (dB)', 'clarity, 80 ms',
    True, 1.0, '{:.1f}'), 'd50': ('D50', 'definition', True, 1.0, '{:.2f}'), 'edt':
    ('EDT (s)', 'early decay time', True, 1.0, '{:.2f}'), 'g': ('G (dB)', 'strength',
    True, 1.0, '{:.1f}'), 'insertion_loss': ('Insertion loss (dB)', 'insertion loss of
    the buildings, barriers or vegetation the map was also run without', True, 1.0,
    '{:.1f}'), 'insertion_loss_a': ('Insertion loss (dB(A))', 'A-weighted insertion loss
    of the buildings, barriers or vegetation', False, 1.0, '{:.1f}'), 'long_term':
    ('Long-term level (dB)', 'level over a long term, outdoors: the mean over the
    weather classes and the day', True, 1.0, '{:.1f}'), 'long_term_insertion_loss':
    ('Long-term insertion loss (dB)', 'insertion loss over a long term: the long-term
    level without the buildings, barriers or vegetation less that with them', True, 1.0,
    '{:.1f}'), 'long_term_insertion_loss_a': ('Long-term insertion loss (dB(A))', 'A-
    weighted insertion loss over a long term of the buildings, barriers or vegetation',
    False, 1.0, '{:.1f}'), 'rating_level': ('Rating level (dB(A))', "level over a long
    term with each period's penalty, A-weighted (L_den)", False, 1.0, '{:.1f}'),
    're_free_field': ('Re free field (dB)', 'level relative to free field, outdoors: the
    excess attenuation with its sign turned', True, 1.0, '{:.1f}'), 'spl': ('SPL (dB)',
    'sound pressure level', True, 1.0, '{:.1f}'), 'spl_a': ('SPL (dB(A))', 'A-weighted
    sound pressure level', False, 1.0, '{:.1f}'), 'sti': ('STI', 'speech transmission
    index', False, 1.0, '{:.2f}'), 't20': ('T20 (s)', 'reverberation time from a 20 dB
    decay', True, 1.0, '{:.2f}'), 't30': ('T30 (s)', 'reverberation time from a 30 dB
    decay', True, 1.0, '{:.2f}'), 'ts': ('Ts (ms)', 'centre time', True, 1000.0,
    '{:.0f}')}
```

The parameters of every map point.

```
acoustic_spaces.engine.coverage.OUTDOOR_MAP_PARAMETERS = ('re_free_field',)
```

The map parameters of an outdoor scene alone, those of a map also run without some of its buildings, barriers or vegetation (`simulation.insertion_loss_of`), those of a map run in weather classes (`long_term`), and those of a map run both ways.

acoustic_spaces.engine.coverage.**ROOM_MAP_PARAMETERS** = ('spl', 'spl_a', 'g', 'c50', 'c80', 'd50', 'edt', 't20', 't30', 'ts', 'sti')

The parameters of every map; an outdoor map, one also run without some of its buildings, barriers or vegetation, and one run in weather classes have the others as well.

acoustic_spaces.engine.coverage.**SPREAD_PARTS** = ('standard', 'low', 'high', 'at_least', 'at_most')

What a map's spread gives of a parameter: its standard uncertainty, the ends of its interval, and the value it is at least and at most (`CoverageMap.spread_of()`).

acoustic_spaces.engine.coverage.**TIME_STEP** = 0.001

Time resolution of each point's energy response (s).

class acoustic_spaces.engine.coverage.**MapParameter**(label: str, description: str, per_band: bool, scale: float, format: str)

How a map parameter is shown: `label` with its unit, what it is, whether it has a value per octave band, and the `scale` and number `format` of the shown value (the stored value is in SI units: Ts in seconds shows in milliseconds). The label and description are marked for translation; show them through `_()`.

label: str

Alias for field number 0

description: str

Alias for field number 1

per_band: bool

Alias for field number 2

scale: float

Alias for field number 3

format: str

Alias for field number 4

class acoustic_spaces.engine.coverage.**MapSpread**(standard: dict[str, ~numpy.ndarray], materials: dict[str, dict[str, ~numpy.ndarray]], coverage: float = 0.95, one_sided: float = 0.95, assumed: dict[str, str] = <factory>, held: dict[str, list[int]] = <factory>)

The uncertainty of a map's parameters that the uncertainty of the materials' absorption leads to, by the law of propagation of uncertainty (`coverage_map()`).

`standard``: dict[str, ndarray]`

Parameter -> its combined standard uncertainty, shaped as the map's values: the root of the sum of the squares of the materials' parts. NaN where the value is undefined, in the map or in a run of the spread.

`materials``: dict[str, dict[str, ndarray]]`

Material -> parameter -> its part: half of the map with its absorption one standard uncertainty up less the map with it one down (negative where more absorption lowers the value).

`coverage``: float = 0.95`

The probability of the interval, and that of the one-sided bounds.

`assumed``: dict[str, str]`

As a listener's spread has them (`Spread`): the built-in materials that took the scene's default uncertainty, and the bands in which a material absorbs everything, which no run moved.

`acoustic_spaces.engine.coverage.`**`audience_surfaces`**`(room: Room) → list[str]`

The surfaces a map covers when none are named: those whose names contain "audience", else "floor" (outdoors, "ground").

`acoustic_spaces.engine.coverage.`**`uncertainty_text`**`(value: float) → str`

A standard uncertainty as a map shows it: to two significant digits, which are as many as it has (GUM, JCGM 100:2008, 7.2.6), and to four decimals at most.

The uncertainty of a coverage map from its materials' uncertain absorption.

A listener's spread is a Monte Carlo over the absorption that re-weights its traced response for each draw (`uncertainty`). A map keeps no response at its points to re-weight (no count of the reflections off each material, no batches of rays), so its spread comes from the law of propagation of uncertainty instead (GUM, JCGM 100:2008, 5.1.2), with the sensitivities found numerically (5.1.3, note 2). The inputs are those of the listeners' draws: one deviate per material, which its bands share, the materials independent of one another. For each material the map is traced again twice, with its absorption where a deviate of plus and of minus one standard deviation puts it (`absorption_nodes()`: its own plus and minus its standard uncertainty, held to a draw's limits; in a run in third octaves, each third octave of a material given in them), on the same points and with the same seed. Half the difference of the two maps is that material's part of the uncertainty of each value at each point, and the root of the sum of the parts' squares the value's combined standard uncertainty (`MapSpread`). A map with M such materials is run 2 M + 1 times.

The two runs of a material have to follow the same rays, or their difference is the rays' as much as the absorption's. A seed alone does not see to that, because the tracer lets the energy a ray carries decide for it: the late part splits and thins out its rays by it (`simulation.late.splitting`), a ray passes a surface that lets sound through or reflects off it at random in proportion to the two shares, plants scatter it at the rate of the bands it still carries, and it ends where too little is left. Each of these comes out otherwise once the absorption is moved. So a material's two runs take these choices from the scene's own absorption (`rays_of` of `coverage_map()`): they follow the rays of the map itself, and count on them the energy that the moved absorption leaves. In a box of 8 by 6 by 3.5 m (2 000 rays) the walls' part of T30 at 1 kHz, for an uncertainty of 0.001, is that uncertainty times a slope of -2.4 to -2.6 s at each of the 12 points; with rays of their own the two runs gave -1.6 to -4.1 s.

The runs are whole map runs, in the weather classes and without the parts an insertion loss names where the map has those (`coverage_map()`), so each kind of value a map holds gets its uncertainty. A value that one of a material's two runs leaves undefined (a decay too short for its T30, say) has none.

It is first order: where a value bends over a material's uncertainty, as a decay time does in a room with little absorption for a tolerance of 0.1, the standard uncertainty comes out low. A material whose absorption lies within its uncertainty of 0, or of the most a draw takes, has its draws held there, which the two runs follow in part only.

Not counted: the rays' random error, and what a listener's spread does not count either (the scattering, the method, a ground's absorption uncertainty beside its impedance). The uncertainty has that error itself: where a value changes from one seed to the next, so does its uncertainty, and a material's small part of it more. How far both go is measured in docs/coverage-maps.md ("Uncertainty"): the law against the Monte Carlo of a listener at a point of the map, and the uncertainty between two seeds of a map.

The interval and the one-sided bounds take the result as normal (`spread_of()`), where a listener's are its draws' quantiles.

```
acoustic_spaces.engine.sensitivity.coverage_map(room: Room | Scene | str | Path, *,
    spread: bool | None = None, long_term: LongTerm | None = None, without:
    Sequence[str] | None = None, settings: Coverage | None = None, simulation:
    Simulation | None = None, sources: Sequence[str] | None = None, threads: int | None
    = None, progress: Callable[[str], None] | None = None) → CoverageResult
```

`coverage_map()`, and with `spread` (default: the map settings' `spread`) the uncertainty of the values of every map from the materials' absorption uncertainty (`spread`): the map is run twice more for each material of `spread_materials()`, on the same points and along the rays of the map itself. The one-sided bounds' probability is `simulation.uncertainty.one_sided`. A map none of whose materials has an uncertainty is run once and has no spread.

```
acoustic_spaces.engine.sensitivity.spread_materials(room: Room) → list[str]
```

The materials whose absorption a map's spread moves: those on `room`'s surfaces that have an uncertainty a spread takes (`uncertainty_of()`) and absorb less than everything in some band.

Edge diffraction

Edge diffraction: sound that bends around the edges of the room, after the uniform theory of diffraction (UTD) for a rigid wedge (Kouyoumjian and Pathak, 1974).

The edges that diffract are those where two faces meet at an angle that opens more than 180 degrees towards the room (the corner of a pillar, a balcony front, the end of a wall) and the free edges of open surfaces (panels and screens). Corners of the room itself open less than 180 degrees towards the room, cast no shadow and are left out, as are the slight bends between the facets of a curved surface.

Sound from a source reaches a listener over an edge along the shortest path that touches it, at the point where the path makes equal angles with the edge on both sides. The diffracted wave splits into terms that each belong to one geometric wave: the direct sound, or its reflection off either face of the edge. Where that wave arrives, its terms are added to it coherently, frequency by frequency, which changes its energy per band; on the far side of its boundary, where the edge cuts the wave off, the terms make a new path. At the boundary itself both sides carry a quarter of the wave's energy, so levels stay continuous where the direct sound or a reflection disappears behind an edge.

In the bands where the paths add in energy, each specular path interferes with the terms of its own wave only, as it adds to the other paths in energy: each term counts in two waves (`clusters`), the geometric wave it continues, by c, and the edge's diffracted wave, by sqrt(1 - c^2). On the lit side of its boundary c is the term's coherence with the geometric wave over the band; on the dark side, where the term stands in for the wave, c = `sqrt(1 - |F|^2)` with F its transition function, 0 well off the boundary. So in the diffracted path the terms of all the waves add up coherently, as they must deep in the shadow, except near each wave's boundary, where they add in energy too. Coherent all the way, levels would step by up to 4 dB where one wave's boundary is crossed while another's terms are there; in energy all the way, the shadow of a screen would be 3 dB deeper than Maekawa's chart. As it is, deep behind the corner of an L-shaped room the 63 Hz band is 0.7 dB below the coherent sum, the 125 Hz band 0.4 dB, and from 500 Hz up the two agree within 0.1 dB.

With `first_order` diffraction the same is done for the first-order reflections: sound that reflects off a plane and then bends over an edge, and sound that bends over an edge and then reflects. Where the path touches the edge closer to that plane than the rays that test it start from the edge, it counts by where the path itself reflects: at that very point where it lies in the plane, off a face that lies there (the ground an upright edge stands on), not off a face of the plane elsewhere (a roof level with a barrier's top).

With `second_order` the direct sound also bends over two edges in turn: over a thick wall or a pillar, round both of its corners. Its shortest path touches each edge where the path makes equal angles with it on both sides. The wave from the first edge reaches the second as a wave of its own, which the second bends again: where the path runs along a face between them (the top of a thick wall), that wave already holds the face's reflection, so the second edge's coefficient is halved; as it is where the path only leaves the first edge along a face, which ends before the second (a roof's corner, then a barrier level with the roof), or only reaches the second along one, so that the path is the same both ways round. Along a face that absorbs, it is a point source's wave along a locally reacting surface at grazing incidence (Weyl-van der Pol): both edges take the face as rigid, and the path the ground wave's loss F(w) over the way along it. Near the first edge's boundaries its wave still curves like the source's, not like a wave from the edge, and the second edge's distance parameter takes that curvature where the wave reaches it; seen from the listener, the same holds for the second edge's wave at the first. Each edge takes the other's curvature by how much nearer the other's boundary is, so the path is the same both ways round. In the bands where the paths add in energy, each term splits at each edge as at one: near the first edge's boundary it is part of the wave of the path over the second edge alone, near the second's of the path over the first alone, and well off both the waves of the two edges add coherently. Where the source's wave reaches the listener past both edges, the paths over each edge alone both take their terms off it, and the path over both gives back what they take twice: over a block's top that the source and the listener both just clear, the direct sound and its reflection off the top less the two corners' terms come to nothing, and the path over both is all there is, half the direct sound. Only with the source or the listener exactly in the plane of a face that the path runs along at the edge next to them (the first edge's, or the second's) is it left out, where the paths over one edge count once already (`_in_front()`).

Openings are not there for diffraction: the edges of a doorway are the free edges of the wall round it, and diffracted paths pass through openings as through air. Outdoors, the open sky is an opening too, but the edges it meets diffract nothing (the ground goes on past them), nor do edges that lie on the ground, which closes the corner of a wall or a barrier standing on it (see `acoustic_spaces.geometry.outdoor`). Any other surface stops them, even one that lets some sound through: the sound diffracted through a wall is left out, as it is far weaker than the sound that passes it straight.

`acoustic_spaces.engine.diffraction.`**`DIFFRACTION_MODES`** `= ('none', 'direct', 'first_order', 'second_order')`

Diffraction settings: none, of the direct sound only, of the direct sound and first-order reflections, or also of the direct sound over two edges in turn.

`acoustic_spaces.engine.diffraction.`**`FREQUENCIES_PER_BAND`** `= 8`

Frequencies per octave band at which the diffracted waves are evaluated, one in the middle of each equal slice of the band; energies are averaged over the band. A third-octave band has a third as many, rounded up.

`acoustic_spaces.engine.diffraction.`**`MIN_BEND_DEG`** `= 15.0`

Faces that bend away from each other by less than this (degrees) make no diffracting edge: the facets of a curved surface, for example.

`acoustic_spaces.engine.diffraction.`**`MIN_RELATIVE_ENERGY`** `= 0.0001`

A diffracted path is dropped when its energy is this far below that of the direct sound in free field, in every band.

class acoustic_spaces.engine.diffraction.**EdgeDiffraction**(model: TraceModel, source: ndarray, *, speed_of_sound: float, air: ndarray, directivity: Directivity | None = None, orientation: ndarray | None = None, mode: str = 'first_order', edges: Edges | None = None, coherent: int = 0, both_sides: bool = False, threads: int = 0)

Diffraction of one source's sound at the edges of a model, for any number of listeners: `apply()` adds it to the specular paths of one listener.

`mode` is `"direct"` (the direct sound only), `"first_order"` (also the first-order reflections, before or after the edge) or `"second_order"` (also the direct sound over two edges in turn); with `both_sides` the paths over one edge may also reflect both before and after it (the four paths over a barrier on the ground). `air` is the energy attenuation per metre per band; `directivity` and `orientation` are the source's, as for `find_paths()`. In the lowest `coherent` bands the terms of every wave add coherently, the faces of the edges and the planes on the way reflect with the phase of their reflection factors, and the diffracted paths carry their pressure (`EarlyReflections.spectrum`) to add to the specular paths in pressure (`coherent`) instead of through cross terms. Only the pairs of an edge and a plane that a reflection between them can take are tested (`_reflecting_pairs()`, on `threads` threads, 0 for every core).

ledger: tuple[EarlyReflections, Clusters] | None

The paths `apply()` returned last and the waves of their pieces.

unheard: tuple[EarlyReflections, EarlyReflections] | None

The paths `apply()` returned last and its paths over edges in a row (`_in_line()`) that carry too little to be among them, or None.

apply(listener: ndarray, paths: EarlyReflections, max_delay_s: float) → EarlyReflections

`paths` (the specular paths from the source to `listener`) with the diffracted sound up to `max_delay_s`: specular paths near the edges' shadow and reflection boundaries change their energy, and paths over the edges into the shadows are added. In the bands where the paths add in energy, the waves of their terms (`clusters`) are kept in `ledger` with the paths returned, for the edge fade to add to.

clusters(listener: ndarray, paths: EarlyReflections) → Clusters | None

The waves of the specular paths `paths` at `listener` in the bands where the paths add in energy, each path its own; None without such bands.

settled(paths: EarlyReflections, energy: ndarray, has: ndarray)

`paths` with the `energy` (n, bands where the paths add in energy) the waves give the paths that `has` says are in them.

class acoustic_spaces.engine.diffraction.**Edges**(start: ndarray, direction: ndarray, length: ndarray, along: ndarray, normal: ndarray, wedge: ndarray, face: ndarray, plane: ndarray, reflection: ndarray, fade: ndarray | None = None, corner: ndarray | None = None, closed: ndarray | None = None)

The diffracting edges of a model, as straight segments.

Each edge is a wedge between face 0 and face n that is open over `wedge` times pi radians (between 1 and 2; exactly 2 for the free edge of a surface, where face 0 and face n are the two sides of the same face). Angles around an edge are measured from face 0 towards its `normal`, through the open side.

fade: ndarray | None = None

(E, 2) how far the paths over each edge fade at its start and end, (E, 2, 2) the edge it meets at a corner there in the plane of face 0 and of face n, or -1, and (E, 2) whether a face across it closes the end (end_fades)

bisector() → ndarray

(E, 3) unit vectors from each edge into the middle of its open side.

distances(index: ndarray, points: ndarray)

Position along edge `index` of each point and distance from it, as `frame()` has them.

frame(index: ndarray, points: ndarray)

Position along edge `index` of each point, distance from it and angle around it from face 0, and whether the point lies in the open side of the wedge.

acoustic_spaces.engine.diffraction.**band_frequencies**(per_band: int | None = None, bands: Literal['octave', 'third_octave'] = 'octave') → tuple[ndarray, ndarray]

(bands, per_band) frequencies at the middle of equal slices of each octave band (from its centre over the square root of 2 to its centre times it) or third-octave band (over and times the sixth root of 2), and the width of the slices per band (bands,). `per_band` defaults to `FREQUENCIES_PER_BAND` per octave.

acoustic_spaces.engine.diffraction.**diffract**(model: TraceModel, source: ndarray, listener: ndarray, paths: EarlyReflections, *, speed_of_sound: float, air: ndarray, max_delay_s: float, directivity: Directivity | None = None, orientation: ndarray | None = None, mode: str = 'first_order', edges: Edges | None = None, coherent: int = 0) → EarlyReflections

`paths` from one source to one listener with the sound diffracted at the model's edges added; see `EdgeDiffraction`.

acoustic_spaces.engine.diffraction.**find_edges**(model: TraceModel, min_bend_deg: float = 15.0) → Edges

The edges of `model` that diffract (see the module notes); collinear pieces of one edge between the same two planes are joined.

acoustic_spaces.engine.diffraction.**fresnel_tail**(z: ndarray) → tuple[ndarray, ndarray]

`1/2 - C(z)` and `1/2 - S(z)` for z >= 0, where C and S are the Fresnel integrals of cos(pi t^2 / 2) and sin(pi t^2 / 2) from 0 to z.

Power series below z = 3.5 and the asymptotic expansion above, good to about 1e-8 where they meet and better elsewhere; computing the tails directly keeps their accuracy for large z.

acoustic_spaces.engine.diffraction.**transition**(x: ndarray) → ndarray

The UTD transition function F(X) = 2j sqrt(X) e^(jX) times the integral of e^(-j t^2) from sqrt(X) to infinity, for X >= 0. It rises from 0 at X = 0 to 1 for large X, where the diffracted wave no longer depends on the shadow boundary.

```
acoustic_spaces.engine.diffraction.wedge_prefactor(n: ndarray, k: ndarray, sin_beta:
    ndarray) → ndarray
```

-e^(-j pi/4) / (2n sqrt(2 pi k) sin beta0), the factor in front of the terms.

```
acoustic_spaces.engine.diffraction.wedge_terms(n: ndarray, phi: ndarray, phi_source:
    ndarray, kl: ndarray, lit: ndarray | None = None) → ndarray
```

The four cotangent terms of the UTD wedge coefficient, without its prefactor.

`n` is the open angle of the wedge over pi; `phi` and `phi_source` are the angles of the listener and source around the edge, from face 0; `kl` is the wave number times the distance parameter L. Arrays broadcast; the result has a last axis of 4: the two terms of the incident wave, then those of the reflections off face n and off face 0. Each term is singular on its shadow or reflection boundary, where it is replaced by its limit; exactly on a boundary, `lit` (a last axis of 3: the incident wave, the reflections off face 0 and off face n) says which side's limit to take (default: the lit side).

The terms at the edges of every specular path, so that each path fades in and out at the edges that cut it off instead of stepping.

The edge diffraction (`diffraction`) bends the direct sound round each edge and, with `first_order`, the first-order reflections, one event at a time: each term it adds stands alone. A path with more reflections, such as the copy of a first-order reflection off the ground, or a second-order reflection, would come or go at full strength where an edge cuts it off: where a leg of it passes an edge, or where a point it reflects at passes the edge of the face. Here every specular path, found or just missed, takes the terms of the uniform theory of diffraction (Kouyoumjian and Pathak, 1974) at each of its events.

An event is a leg of the path, where it passes edges, or a reflection, near the edges of the face it reflects off. Its term at an edge is the UTD's term of the boundary that belongs to it: the incident wave's at a leg, the reflection's off the face at a reflection. It runs from the path's image on the source side (the source mirrored in the reflections before) to its image on the listener side (the listener mirrored in the reflections after), so that the path over the edge has the path's own length near the boundary, relative to the path.

The terms of one event add: every edge counts on its own side of its boundary, as the edges round an opening do in the boundary diffraction wave (Rubinowicz, 1917), so that both edges of a gap count where the path goes through it, and both where one of them cuts it off. The events of a path multiply, as the path must get past each in turn: with b = 1 where the path gets past an event and 0 where it does not, and G the sum of the event's terms, the path carries the product of b + G over its events. Each product of terms in it is a path of its own, which arrives later by the detours of all its terms. Off the terms comes what the edge diffraction already adds where it has the path over the edge (the term on the lit side where the path is found, on the dark side where it is not), so that both together make the product.

A path that is not found is followed where one event stops it and an edge of what stops it explains that, on the dark side of its boundary: a leg blocked by a face of the edge (where the path crosses the face right at a point of reflection, as at the foot of a barrier standing on the ground it reflects off, the leg that crosses the face's plane), or a point of reflection past the edge of its face. Where a point of reflection lies past the end of its plane's faces at an edge that stands on one of them (a corner of the room on the floor; not a wall as high as a roof elsewhere in the plane, nor a building's corner under its roof, where the roof's plane does not go on round it), the face of the edge that blocks a leg next to it stops the path, and the reflection takes the face nearest to that point. Where several events stop it, in at most two runs (the events with no reflection on its face between them), the path is followed where one edge explains each run, with its images in the planes it has a face in or stands on (its blocked legs blocked by the edge's faces, its reflections past the end of their faces in such a plane, in front of the edge's face where it has one there: not under a roof), or two edges explain the stops between them, each a part in a row; the path then takes the terms of both stops. Its reflection points are those of the image method; of the orders of reflections off perpendicular planes, which have one image, the one that can be (each is tried, as the early stage may have found only one). Paths that pass through a surface that lets sound through take no terms, as they are not the geometric wave the terms continue.

Reflections off perpendicular planes commute: in either order they give one image, and the path takes the order its points come in along the way. The path over an edge may take another, such as off the floor after a corner that stands on it rather than before, so each path also takes the terms of the events of every other order of its reflections that gives its image, those that differ from its own once each (all orders up to four reflections, beyond that those that swap two next to each other).

At a leg, the terms of the edge's reflections whose paths cannot be, with the source or the listener side behind the face's plane, stand in for those reflections, which the early stage follows in front of the plane only for paths of a lower order than the highest; they take what the face gives the reflection where it meets the plane, the scattering of its surface's size included where the early stage weakens the reflection for it (`size_scattering()`).

A leg that gets past an edge on the dark side of its boundary, as beside the end of a strip, takes the edge's term where the edge's face crosses the leg's line on the leg: where the line crosses the plane of the edge's face (of a solid wedge, the plane through the middle of its solid side), it must lie on the leg's side of the planes the leg reflects off at its ends. Beyond one of them the edge belongs to the next leg (a barrier's top past the reflection off the ground that ends a leg, which blocks the leg after it), save for an edge along the plane's normal, its own image there. A point of another edge in such a plane (on a barrier's top level with the roof a path reflects off) belongs to the leg it stands beside: the one before the point of reflection or the one after it.

Each term makes a path over its edge, which must be clear from the edge to where it reflects first before and after it (where the line from the edge to the image on either side first meets a plane it may reflect off there), save where two events stop the path and it reflects next to the other's blocked leg, which blocks the way there too. Where it stands in for a missed path (at an event that stops it, and where several do, at the legs next to a stopped reflection) it must be clear on from there too, leg after leg off the path's planes to the source and the listener, as it reflects at other points than the path does (past a barrier's foot, behind another barrier), up to another event that stops the path, whose terms go on from there: the terms of one event add, each one way past all that stops the path there, so each must get past all of it on its own. At an event the path gets past, the term goes on with the path's wave, and an obstacle on its further legs is mostly another event of the path, whose terms multiply it already: checked there too, that obstacle would count twice (in validation/edge_diffraction.py's hall with alcoves the level then steps by 4.2 dB at 63 Hz over 2 cm, against 1.7 dB). In the bands where paths add in pressure it carries its phase. In the others it is a piece of two waves (`clusters`): it continues the geometric wave of its boundary, the path's, by c, its coherence with that wave over the band on the lit side of the boundary and the UTD's `sqrt(1 - |F(X)|^2)` on the dark side, where the wave is gone, and is part of its event's diffracted wave by sqrt(1 - c^2), with the terms the edge diffraction adds there (round a narrow face the two edges' terms nearly cancel). A product of terms is split alike, term by term, and so are the events the path gets past before a term: the part of their terms in an event's diffracted wave, times what continues the path at its other events, is part of that wave. A term stops where the point it diffracts at leaves the end of its edge (the UTD has no terms for the corners of edges), so the level can still step slightly there; where that end lies in a plane the path reflects off next to the event, the edge goes on as its image in the plane, and the leg on one side of the reflection or the other takes the term. Where such a leg stops a path and the edge belongs to the obstacle that stops it (its own face, or one joined to it by no inside corner, as the far part of a bent or curved barrier or another wall of a building), the term of the edge's image, found on the leg on the other side, is the stopping leg's: in the plane's image the two legs are one line, and the edge is one more way round that one obstacle.

```
class acoustic_spaces.engine.edge_fade.EdgeFade(diffraction: EdgeDiffraction, candidates:
    list[ndarray])
```

The terms at the edges of `diffraction`'s model of one source's specular paths (see the module notes), for any number of listeners. `candidates` are the plane sequences the early stage tests at each listener (as `find_paths()` takes them); those not found there may still be found past an edge.

`covered(before: int, after: int) → bool`

Whether the edge diffraction bends the waves from a source side mirrored in `before` planes to a listener side mirrored in `after` planes.

`terms(listener: ndarray, paths: EarlyReflections, max_delay_s: float, bent: EarlyReflections | None = None) → dict`

The terms at the edges of `paths` (the specular paths at `listener`) and of the paths just missed there, up to `max_delay_s`: for `join()`. With `bent`, the paths the edge diffraction finds at `listener`, the terms leave out exactly those of its paths over an edge (also those it keeps `unheard`); without, those it adds where it bends an event, as it does where nothing else stops its paths.

`join(paths: EarlyReflections, terms: dict) → EarlyReflections`

`paths` (the specular paths of `terms()` with the edge diffraction's) with the `terms` of `terms()`: in the bands where the paths add in energy, the terms' pieces join the waves of `paths` (those the edge diffraction keeps for them in its `ledger`, or the specular paths' own), which give every path its energy; and the paths over the edges that carry at least the edge diffraction's least energy (`MIN_RELATIVE_ENERGY` of the direct sound in free field) in some band.

The waves that are one wave, in the bands where the early paths add in energy.

Indoors, above `simulation.early.coherent_max_band_hz`, the early paths add in energy, as if their phases were random. The sound bent round an edge does not: near its boundary each term of the uniform theory of diffraction continues the geometric wave it belongs to, and its pressure adds to that wave's, so that the two together change smoothly where the geometric wave stops (`diffraction`, `edge_fade`). Away from it the term is a wave of its own, bent round the edge, whose terms (one for each boundary of the edge) add up as one wave by how far each lies from its boundary.

Each term therefore counts in two waves. It continues its geometric wave by c and is part of its diffracted wave by s = sqrt(1 - c^2): its own energy stays `|t|^2 = (c^2 + s^2) |t|^2`, and two terms cohere by the products of their shares, through every wave both are part of. On the lit side of its boundary, where the geometric wave arrives, c is the term's coherence with it over the band, as the term goes X / k farther (X the argument of its transition function); on the dark side, where the wave is gone, c = `sqrt(1 - |F(X)|^2)`, so that the term continues it near the boundary only, and deep in the shadow all the terms of an edge are one wave (`transition()`). All the edges of an event are one diffracted wave, as the edges round an opening are in the boundary diffraction wave (Rubinowicz, 1917): the terms whose paths over an edge run from the same image of the source to the same image of the listener, whatever the edge, so that round a narrow face the two edges' terms nearly cancel as they do in pressure, and the many edges of a curved surface's facets add up to the surface's own reflection.

A wave's energy is that of the pressures of its pieces added up, averaged over each band's frequencies and over the slice of the band each frequency stands for, where two pieces that arrive t seconds apart interfere by sinc(t times the slice's width) of what they would at its frequency: it is never negative, as it is for the pressures of any set of waves. A wave of a few pieces takes them pair by pair; in one of more, the slice is averaged over with a Gauss-Legendre rule of as many points as the pieces' spread in arrival needs (within 1e-10 of sinc), so that the work grows with the pieces, not with their pairs: the facets of a curved surface give events of hundreds of thousands of terms.

The wave of a specular path, found by the early stage, is that path's: the cross terms of the pieces that continue it go to its energy, and each piece keeps its own; where the cross terms take more than the path has, the rest comes off the pieces in proportion to their energy, which is always enough. In every other wave each piece takes its own energy and half its cross term with each other piece; where that comes to less than nothing it takes none, and the others give it in proportion, so that the wave's energy stays what it is.

class acoustic_spaces.engine.clusters.**Pages**

The pressure of pieces, kept in pages of `_PAGE` bytes. The C library maps an array that large on its own and hands it back to the system once it is let go, where it keeps smaller ones in its heap: the millions of pieces of a room of many facets (BRAS CR3), let go at one listener, left gigabytes there that the next could not use.

keep(values: ndarray) → ndarray

`values` (n, …) in single precision, copied into a page.

acoustic_spaces.engine.clusters.**hand_back**() → None

Hands back to the system the memory the C library keeps free (glibc's `malloc_trim`; elsewhere nothing). The pieces of a room of many facets come and go by gigabytes at each listener, many of them in blocks too small for the library to map on their own, and it kept what they freed: at the start of BRAS CR3's second listener in octave bands, 1.3 GiB free of a heap of 1.5 GiB, and in third-octave bands more at each listener, until the third ran out of memory.

class acoustic_spaces.engine.clusters.**Clusters**(slice_hz: ndarray, speed_of_sound: float, size: float)

The pieces of the waves that are one wave (see the module notes), in the bands where the early paths add in energy, whose slices are `slice_hz` (Hz) wide, for sound at `speed_of_sound`; images closer than `_STEP` times `size` (m) are the same.

`geometric(image: ndarray, order: ndarray) → ndarray`

The waves of the specular paths of `order` (n,) reflections whose image of the source is `image` (n, 3).

`diffracted(source_side: ndarray, listener_side: ndarray) → ndarray`

The diffracted waves whose paths over an edge run from the image of the source `source_side` (n, 3) to that of the listener `listener_side` (n, 3).

`private(count: int) → ndarray`

`count` waves of their own.

`add(rows: ndarray, keys: ndarray, pressure: ndarray, distance: ndarray, anchor: bool = False, own: ndarray | None = None) → None`

Pieces of the rows `rows` (-1 for a row left out) in the waves `keys`: their pressure (n, bands, M) with the phase of their whole way, which is `distance` (n,) metres long. `anchor`: the rows are specular paths, whose waves they are, with energy `own` (n, bands). Pressure in single precision with something in every piece is kept as given (the edge fade's comes so, in its own `Pages`); any other is copied into a page. The energy of pieces in single precision without `own` is not kept but worked out again from their pressure when it is needed.

`alone(rows: ndarray, energy: ndarray) → None`

Pieces of the rows `rows` that are waves of their own, with `energy` (n, bands).

`renumber(mapping: ndarray) → None`

Rows `i` become `mapping[i]` (-1: left out).

`resolve(count: int) → tuple[ndarray, ndarray]`

(count, bands) the energy of each of `count` rows from its pieces, and (count,) which rows have any.

Insertion loss

The insertion loss of an outdoor scene's buildings, barriers and vegetation.

The scene is run as it is and again without them (`without()`), with the same settings and seed, so that the two runs' rays start alike; the insertion loss is the level without them less the level with them, per octave band (and A-weighted, for the sound power of each source). A barrier that takes 10 dB off has an insertion loss of 10 dB; a building that sends sound back towards the listener can have a negative one.

`simulation.insertion_loss_of` names what to take away; `simulate()` and `coverage_map()` read it unless given `without`, and are `simulate()` and `coverage_map()` when it names nothing.

`acoustic_spaces.engine.insertion.a_weighted_loss(spl: ndarray, loss: ndarray) → float`

The A-weighted insertion loss (dB) from the levels with the parts per octave band (dB SPL) and their insertion loss per band: the A-weighted level without them less that with them.

`acoustic_spaces.engine.insertion.bare_room(room: Room, names: Sequence[str]) → Room`

The room without the buildings, barriers and vegetation of these names.

`acoustic_spaces.engine.insertion.coverage_map(room: Room | Scene | str | Path, *, without: Sequence[str] | None = None, settings: Coverage | None = None, simulation: Simulation | None = None, sources: Sequence[str] | None = None, threads: int | None = None, progress: Callable[[str], None] | None = None, grid: MapGrid | None = None, rays_of: Scene | None = None) → CoverageResult`

`coverage_map()`, and the insertion loss of the buildings, barriers and vegetation named in `without` (default: `simulation.insertion_loss_of`) at every map point: each map gains `insertion_loss` (dB per octave band) and `insertion_loss_a` (dB(A), for the sources' sound power), the level without them less the level with them. Both runs share the map's points, those of the scene as it is (or `grid`, as `coverage_map()` takes it, as it does `rays_of`, for both runs).

`acoustic_spaces.engine.insertion.simulate(room: Room | Scene | str | Path, *, without: Sequence[str] | None = None, settings: Simulation | None = None, sources: Sequence[str] | None = None, listeners: Sequence[str] | None = None, threads: int | None = None, progress: Callable[[str], None] | None = None, groups: ndarray | None = None) → SimulationResult`

`simulate()`, and the insertion loss of the buildings, barriers and vegetation named in `without` (default: `simulation.insertion_loss_of`) at every listener: the result is the scene's as it is, with `removed` and `insertion_loss` (dB per octave band, the level without them less the level with them; NaN where either level is undefined). `groups` are for the run of the scene as it is.

Early reflections in pressure

Early paths that add in pressure at low frequencies.

The echogram adds the early paths in energy, band by band, as if their phases were random. At low frequencies they are not: a band's filter rings longer than the early paths lie apart, and their pressures add. The impulse response has always added them that way (its paths are impulses that add sample by sample), so the parameters of the 63 and 125 Hz bands read from it differed from the echogram's by up to 2 dB.

In the lowest bands, up to `simulation.early.coherent_max_band_hz`, every early path therefore carries its pressure (`EarlyReflections.spectrum`) at the frequencies the edge diffraction is evaluated at: the phase of each reflection factor on the way, and for paths over an edge the whole complex UTD coefficient, whose terms then add to the specular paths in pressure, as the theory has them, rather than through the cross terms of the higher bands. Its level in each band stays the energy that band gives it. A surface reflects like a locally reacting one whose admittance has the material's absorption in every band (`fit_admittance()`), so the phases in a band do not depend on how many bands add in pressure: R = (cos theta - Y) / (cos theta + Y) at the angle of incidence theta. A hard surface reflects in phase at every angle; an absorbing one turns the phase of sound that grazes it, by half a turn at grazing incidence. Above those bands each path keeps the phase it has at their top (`held_phase()`).

But for the paths over an edge: in those bands the energy of a wave bent round edges is that of its pieces added up in pressure already (`clusters`), and each piece carries its share of it. Added in pressure once more, with the phases held from the bands below, the pieces of one wave counted their interference twice: the facets of BRAS's hall, thousands of pieces within a period, rendered 16 times their energy at 250 Hz at the dummy head. A rendered response therefore turns each path over an edge by a phase of its own (`own_phases()`), drawn from the path and the response's seed, as far as other paths' sound reaches it where it arrives, so that it adds to them in energy on average, as in the echogram. One that arrives alone keeps its held phase, which its bands need to join the coherent ones. Reflections keep theirs: two that arrive together add in pressure, as at a point of a real room, where the echogram gives their average over the band.

The impulse response builds those bands' early part from the pressures, and the echogram reads each band's early energy from the same signal through the band filters of the analysis of measured responses (`band_filter_gains()`), as for the wave solver's bands: an impulse response and the echogram it came from now give the same parameters in those bands. The filters let in a little of the bands above, where the echogram gives each path over an edge the mean over seeds of the phase factor a rendered response turns it by (`with_mean_phases()`): with the phases held, the pieces' interference counted twice reached it through them, twice the early energy at 125 Hz at BRAS's hall's dummy head. The late part still adds in energy, and so do the uncorrelated points of a line or surface source.

```
acoustic_spaces.engine.coherent.FILTERED_MAX_BAND_HZ = 500
```

The coherent bands up to this octave band are read through the analysis's band filters (`EarlyPressure`); outdoors, the bands above it from the interference of each pair of paths averaged over the band (`pair_energy()`), which costs the same however far the sound travels.

```
class acoustic_spaces.engine.coherent.EarlyPressure(paths: EarlyReflections, weights:
    np.ndarray | None = None, kept: SameDelays | None = None)
```

The early part of an impulse response at a low sample rate, as the echogram reads the coherent bands from it: as spectra from 0.2 s before emission, the pressure and the three components of the paths' arrival directions times it, or with `weights` (paths, K) the sums of the paths weighted by each column.

Each band is scaled so that every path alone carries its energy in it: the analysis filters also let in a little of the neighbouring bands, whose levels differ, and the filters of the impulse response's bands overlap with them. What is left is the paths' interference, which in turbulence takes each two paths' coherence factor: there each path's spectrum is kept, and `energy()` adds the filtered paths' products.

With `kept` (a `SameDelays`), the paths' delay phases are taken from it and kept in it, and each band's filtered pressure is kept for the next reading.

band(b: int, time_step: float, slots: int, origin: float = 0.0, start: float | None = None) → tuple[ndarray, ndarray, float]

The spectra through band b's analysis filter (K, T) at the samples that fall into the `slots` of `time_step`, slot 0 at `origin` (s from emission); the slot of each sample (T,); and the factor that makes a squared sample energy.

The filters ring before an arrival as much as after it: what they spread to before `start` (s from emission; default `origin`), the arrival of the direct sound, counts as arriving with it, as in the analysis of a measured response. Kept for the next reading of the same arguments (read-only) when the pressure was made with a `SameDelays`.

energy(time_step: float, slots: int, axis: ndarray | None = None, exponent: int = 2, origin: float = 0.0, start: float | None = None) → ndarray

(count, slots) energy per coherent band and time slot, as `energy()` reads the wave solver's bands, through each band's analysis filter (`origin` and `start` as for `band()`). With `axis`, that of a figure of eight facing it (`exponent` as there).

class acoustic_spaces.engine.coherent.**SameDelays**(limit: int = 134217728)

What `EarlyPressure` and `arriving_with()` take from the early paths' delays alone (and which paths carry a band), kept for paths that differ in nothing else but their energies and spectra: those of one traced response re-weighted for draw after draw of the absorption (`material_spread()`). Each value is kept under the bytes of the arrays it is made from, as every echogram slices each point's paths anew, and is read-only. An `EarlyPressure` given one also keeps each band's filtered pressure (`EarlyPressure.band()`) for the echograms that read it.

The delay phases are the most of it: a complex value for each path and frequency of the echogram's spectra, about 6 MB for a hall's response of 200 paths, up to 110 MB for a line source of 10 points outdoors (examples/street, in wind and turbulence too). What is kept stops at `limit` bytes in all (128 MiB by default); past it, each value is worked out again each time, as without one. The filtered pressures and what `coherent_energy()` keeps for a draw's echograms are not counted in it: they go with the draw's response, about 25 MB for the street's.

delay_phases(shift: ndarray, n_fft: int, rate: float) → ndarray

exp(-2j pi outer(`shift`, f)): the phase of paths `shift` s late at the frequencies f of an rFFT of `n_fft` points at `rate`.

band_grid(delay: ndarray, b: int, bands: int, live: ndarray) → tuple[ndarray, ndarray, int, ndarray, ndarray]

`_band_grid()` of these arguments.

acoustic_spaces.engine.coherent.**absorption_phases**(absorption: ndarray, cosines: ndarray, f: ndarray, admittance_of=None) → ndarray

(n, count, M) e^(j arg R) at the frequencies `f` (count, M) of reflections at incidence cosines `cosines` (n,) off locally reacting surfaces whose admittance gives the absorption `absorption` (n, bands) in every band, as `reflection_phases()` has them. `admittance_of(row)` gives the admittance (a callable of the frequency) of an absorption row; by default the one fitted to it (`fit_admittance()`).

```
acoustic_spaces.engine.coherent.amplitude_spectra(paths: EarlyReflections, f: np.ndarray,
    gains: np.ndarray, only_coherent: bool = False) → np.ndarray
```

(n, len(f)) each path's pressure at frequencies `f` relative to its delay, as the impulse response has it: its amplitude in each band through the synthesis's band `gains` (bands, len(f)) at those frequencies, in the coherent bands times its spectrum's shape in that band (`shaped()`) and above them times its `held_phase()`; with `only_coherent`, the coherent bands alone.

```
acoustic_spaces.engine.coherent.arriving_with(paths: EarlyReflections, kept: SameDelays |
    None = None) → np.ndarray
```

(n,) for each path, the largest over the bands above the coherent ones of how loud the other paths' sound is where it arrives against its own, each path with the phase held from the coherent bands: the magnitude of the sum over the others of their pressure times the band filter's autocorrelation at the difference of their delays, over its pressure. Zero for a path that arrives alone; about one or more for one that others arrive with. `kept` (a `SameDelays`) keeps what each band's reading takes from the delays and which paths carry the band alone, for the same paths with other energies and phases.

```
acoustic_spaces.engine.coherent.band_count(max_band_hz: int | None, bands: int,
    wave_max_band_hz: int | None = None) → int
```

How many of the lowest of `bands` (8 octave or 24 third-octave bands) add in pressure: those of the octave bands up to `max_band_hz` (None: none), and none when the wave solver runs up to that band or higher (`wave_max_band_hz`), as its bands take their place.

```
acoustic_spaces.engine.coherent.band_impulses(paths: EarlyReflections, n: int,
    sample_rate: float, gains: np.ndarray | None = None) → tuple[np.ndarray, np.ndarray
    | None]
```

The band signals (bands, n) of the early paths above the coherent bands, each path times `gains` (paths,) (default ones), as `synthesis.place_impulses()` has them, and those of the quarter turn of their `held_phase()` for `synthesis.combine_bands()` (None when no path has one).

```
acoustic_spaces.engine.coherent.coherent_energy(paths: EarlyReflections, time_step:
    float, slots: int, axis: np.ndarray | None = None, exponent: int = 2, origin: float
    = 0.0, start: float | None = None, cache: dict | None = None, kept: SameDelays |
    None = None) → np.ndarray
```

(coherent bands, slots) the coherent bands' energy per time slot (arguments as for `EarlyPressure.energy()`): through the band filters up to `FILTERED_MAX_BAND_HZ` and by `pair_energy()` above. The paths of each of a line's or a surface's points add in pressure, those of different points in energy. The band filters let in a little of the bands above, where each path over an edge has the mean of the phase a rendered response turns it by (`with_mean_phases()`). `cache` (a dict) keeps the `EarlyPressure` of each point between calls on the same paths; `kept` (a `SameDelays`) what they take from the paths' delays alone, between calls on paths that differ in their energies and spectra only.

acoustic_spaces.engine.coherent.**filtered_bands**(paths: EarlyReflections) → int

How many of the paths' coherent bands are read through the band filters: those up to `FILTERED_MAX_BAND_HZ`.

acoustic_spaces.engine.coherent.**ground_factors**(model: TraceModel, faces: np.ndarray, cosines: np.ndarray, distance: np.ndarray | None, spherical: bool = True, ends: tuple[np.ndarray, np.ndarray, np.ndarray] | None = None) → np.ndarray

(n, bands, M) the reflection factors off `faces` (n,) of sound that meets them at incidence cosines `cosines` (n,), at the frequencies of every band's samples (`band_frequencies()`): for surfaces with an impedance, that of the spherical wave (Weyl-van der Pol, see `spherical_reflection()`) along paths `distance` (n,) m long from the image source, or with `spherical` False the plane wave's; 1 for the others.

On the ground of a scene with ground zones, a reflection is the sum of the factors of the grounds its Fresnel zone covers, each weighted by the share it covers (see `fresnel_shares()`), from the points `ends` (n, 3) before the reflection, of the reflection and after it; without `ends`, the ground's own. There, a ground without an impedance reflects with the square root of (1 - its absorption) and the phase of `reflection_phases()`.

acoustic_spaces.engine.coherent.**ground_wave_factors**(model: TraceModel, faces: np.ndarray, distance: np.ndarray) → np.ndarray

(n, bands, M) the Weyl-van der Pol factor Q of the sound that runs `distance` (n,) m along `faces` (n,) at grazing incidence, from a point on them, at the frequencies of every band's samples: the field there is (1 + Q) = 2 F(w) times the free one (see `spherical_reflection()`). Faces with an impedance of their own take `ground_factors()`' spherical wave, the others, a ground zone's among them, that of the admittance fitted to their absorption (as `reflection_phases()` has it); a hard face gives 1.

acoustic_spaces.engine.coherent.**held_phase**(paths: EarlyReflections) → np.ndarray

(n,) the phase factor of each path at the top of the coherent bands, which it keeps in the bands above them: there the paths add in energy, and a path whose phase stepped back to zero at the crossover would cancel itself in it, as two bands half a turn apart do (ones without coherent bands). Times the path's own phase (`own_phase`) where a rendered response gives it one (`with_own_phases()`).

acoustic_spaces.engine.coherent.**own_phases**(paths: EarlyReflections, seed: int | None, angles: np.ndarray | None = None) → np.ndarray

(n,) the phase factor beyond the one held from the coherent bands that a rendered response gives each path above them. A path over an edge is turned by an angle drawn uniformly from the path (its edge, order, faces, crossings and source point, and which of the paths alike it is) and `seed`, times how loud the other paths' sound is where it arrives against its own (`arriving_with()`), up to one: so it adds to the paths it arrives with in energy on average, and one that arrives alone keeps its phase. A reflection keeps its phase (one). `angles`: their `path_angles()`, when worked out already.

acoustic_spaces.engine.coherent.**pair_energy**(paths: EarlyReflections, bands: range, time_step: float, slots: int, axis: np.ndarray | None = None, exponent: int = 2, origin: float = 0.0, memo: dict | None = None) → np.ndarray

(len(bands), slots) energy per time slot of `time_step` (slot 0 at `origin`, s from emission) in `bands` of the paths' coherent ones, adding them in pressure: each path's own energy where it arrives, and the cross term of each pair of paths, averaged over the band's frequencies as in `total_energy()` (with their coherence factor in turbulence), shared between the two paths' slots as their energies are (a slot that would go below zero is left at zero). With `axis`, that of a figure of eight facing it (`exponent` as in `EarlyPressure.energy()`; with 1 the cross terms take the geometric mean of the two paths' weights). `memo` (a dict), if given, keeps what each band takes whatever the axis (`_pair_terms()`) for the next call on the same paths.

```
acoustic_spaces.engine.coherent.path_angles(paths: EarlyReflections, seed: int | None) → np.ndarray
```

(n,) an angle in [-pi, pi) for each path over an edge, zero for the others, drawn from what the path is and `seed`: wherever the source and the listener stand, the same path gets the same angle, and at every moment of the eddies.

```
acoustic_spaces.engine.coherent.reflection_phases(model: TraceModel, faces: np.ndarray, cosines: np.ndarray, count: int) → np.ndarray
```

(n, count, M) e^(j arg R) of the reflections off `faces` (n,) of sound that meets them at incidence cosines `cosines` (n,), at the frequencies of `sample_frequencies()` of the lowest `count` bands: R is that of a locally reacting surface whose admittance gives the face's absorption in every band; 1 for the faces whose reflection `ground_factors()` gives (see `factor_faces()`).

```
acoustic_spaces.engine.coherent.sample_frequencies(bands: int, count: int) → ndarray
```

(count, M) the frequencies (Hz) a spectrum holds in each of the lowest `count` of `bands` bands: those the edge diffraction is evaluated at.

```
acoustic_spaces.engine.coherent.shaped(spectrum: ndarray, bands: int, f: ndarray, amplitude: ndarray, gains: ndarray, fine: ndarray | None = None, fine_delay: ndarray | None = None) → ndarray
```

(n, len(f)) the coherent bands' amplitudes (n, count) through the synthesis's band `gains` (count, len(f)) at frequencies `f` (Hz), each band's times the shape its spectra (n, count, M, `EarlyReflections.spectrum`) give it: the magnitude and the unwrapped phase interpolated linearly in log frequency between the band's own samples and held beyond them. A band so carries its own sound into the crossovers with its neighbours, not theirs: outdoors a path's bands below the parabolic equation's top band hold the whole sound over the ground and those above it the path's own, and a shape drawn across that edge would notch the sum.

Paths with the parabolic equation's samples in a band (`fine` (n, count, K) and `fine_delay` (n,), `EarlyReflections.fine`) take its shape through those and the neighbouring bands' instead (`fine_shape()`), so that both bands of a crossover give the field it has there.

```
acoustic_spaces.engine.coherent.signal(paths: EarlyReflections, n: int, sample_rate: float, weights: np.ndarray | None = None) → np.ndarray
```

(n, K) the coherent bands' part of the early paths' impulse response, at `sample_rate`, each path weighted by `weights` (paths, K) (default: one column of ones). Zero without coherent bands.

```
acoustic_spaces.engine.coherent.total_energy(paths: EarlyReflections) → np.ndarray
```

(bands,) the paths' energy together in each band: their sum, and in the coherent bands that of their pressures added at each of the band's frequencies, averaged over them; each frequency stands for its slice of the band, over which the interference of two paths that arrive far apart averages out; in turbulence it takes their coherence factor (`coherence()`). Paths of different points of a line or a surface source add in energy.

```
acoustic_spaces.engine.coherent.with_mean_phases(paths: EarlyReflections, kept:
    SameDelays | None = None) → EarlyReflections
```

`paths` as the echogram reads the coherent bands from them (`EarlyPressure`), through filters that let in a little of the bands above: each path over an edge with the mean over seeds of the phase factor `own_phases()` gives it there, sinc of how far it is turned. One turned all the way adds nothing to the others there in pressure, as on average in a rendered response. The same paths without such bands or such paths. `kept`: as for `arriving_with()`.

```
acoustic_spaces.engine.coherent.with_own_phases(paths: EarlyReflections, seed: int |
    None, angles: np.ndarray | None = None) → EarlyReflections
```

`paths` as a rendered response has them: each path over an edge turned by its `own_phases()` in the bands above the coherent ones. The same paths without such bands or such paths. `angles`: their `path_angles()` for `seed`, when worked out once for every moment of the same paths.

Leak finder

The leak finder: where rays escape from a model that is not closed.

The mesh checks count the open edges of a model, but do not show where its holes are. `find_leaks()` traces rays from the sources, reflecting them diffusely off every surface (a Lambert direction each time, whatever the materials, so that they reach every corner) and passing them through surfaces that let sound through with the share they let through. A ray that finds no surface ahead has escaped. Each escape is put down to the gap whose open edges it passes closest to; gaps whose edges come within a short distance of each other are one gap (a ceiling that stops short of a wall leaves open edges on both). An escape right after a ray passed through a surface that lets sound through (an open window to the outside) is put down to that surface. The surfaces next to a gap say where to look, and its area is estimated from how often rays find it: in a field of diffuse reflections, each reflection meets a share of the boundary in proportion to its area, so a gap takes the share of reflections that escape through it, times the area of the surfaces.

`acoustic-spaces validate --leaks` prints the report and `acoustic-spaces view --leaks` draws the escaping rays in 3D, as does the editor's View menu.

```
acoustic_spaces.engine.leaks.MAX_REFLECTIONS = 100
```

Reflections each ray makes at most.

```
acoustic_spaces.engine.leaks.RAYS = 5000
```

Rays that the finder traces by default, shared by the sources.

```
class acoustic_spaces.engine.leaks.Gap(rays: int, centre: ndarray, surfaces: tuple[str,
    ...], open_edges_m: float, area_m2: float, size_m: tuple[float, float] = (0.0, 0.0),
    kind: str = 'gap')
```

A place where rays left the model.

`rays`: `int`

Rays that escaped through it.

`centre`: `ndarray`

(3,) the mean of the points where they left (m).

`surfaces`: `tuple[str, ...]`

Names of the surfaces next to it, the longest open edges first; for a surface that lets sound through, its own name.

`open_edges_m`: `float`

Length of the open edges round it (m); 0 for a surface that lets sound through, or where rays slipped between faces that the mesh joins.

`area_m2`: `float`

Its area (m2): the area its open edges run round, when they close up into loops; else as the rays found it, from the share of reflections that met it. 0 for a surface that lets sound through and a source outside the room.

`size_m`: `tuple[float, float] = (0.0, 0.0)`

Its length and width (m): how far its open edges reach along the two directions they spread out most in; zeros where there are no open edges.

`kind`: `str = 'gap'`

`"gap"`: open edges round a hole; `"surface"`: a surface that lets sound through (an opening the scene means to have, not a leak); `"between faces"`: an edge of a closed mesh that rays slipped through; `"outside"`: rays that left a source outside the room without meeting a surface (`surfaces` names the source).

class acoustic_spaces.engine.leaks.**Leaks**(rays: int, reflections: int, max_reflections: int, starts: ndarray, directions: ndarray, points: ndarray, gap: ndarray, gaps: list[Gap], surface_area_m2: float)

Where the rays of `find_leaks()` escaped.

`rays`: `int`

Rays traced.

`reflections`: `int`

Reflections they made in all (passes through surfaces included).

`max_reflections`: `int`

Reflections each ray made at most.

`starts`: `ndarray`

(k, 3) where each escaping ray last reflected (or left its source).

`directions`: `ndarray`

(k, 3) its direction from there.

`points`: `ndarray`

(k, 3) where it left the model: where it crosses the plane of its gap's open edges (for a gap that is not flat, its point nearest them; for an edge between joined faces, its point nearest that), or `starts` when it left through a surface or straight from its source.

`gap`: `ndarray`

(k,) the gap each escaping ray left through (an index into `gaps`).

`gaps`: `list[Gap]`

The gaps, the one the most rays escaped through first.

`surface_area_m2`: `float`

Total area of the surfaces (m2).

`property leaked`: `int`

Rays that escaped through a gap, not through a surface that lets sound through.

`summary()` → `str`

A few lines for people: how many rays escaped, and where.

`acoustic_spaces.engine.leaks.`**`find_leaks`**`(room: Room, *, rays: int = 5000, max_reflections: int = 100, seed: int = 0) → Leaks`

Trace `rays` rays from the room's sources (shared by them; from the middle of the model when it has none) for up to `max_reflections` diffuse reflections each, and find where they escape.

`acoustic_spaces.engine.leaks.`**`open_edges`**`(room: Room) → tuple[ndarray, ndarray]`

The edges of the room's mesh that only one face has: (E, 2) their corners (indices into the mesh's vertices, in the order the face runs along them) and (E,) that face.

Audience seating

Grazing incidence over the audience: sound that reaches a listener low over rows of seats loses much more than the seats' absorption says, as it passes heads and seat backs on the way.

The model is the fit of Jurkiewicz, Rummler, Green, Wulfrank and Kahle ("Impact of grazing incidence attenuation on early reflection design", Proc. IOA 47 Pt. 1, 2025) to measurements over a seated audience in a 1:20 scale model. Sound arriving at an elevation φ above the audience plane, having travelled d metres over the audience, keeps

> min(1, (3.40 sin²φ + 2.17 sin φ + 0.02) ^ ((d - 1) / 10))

of its energy: 17 dB less at grazing incidence after 11 m, 0.6 dB per metre at 5°, and nothing lost from 17.8° up. It was fitted in the 500 Hz to 2 kHz octave bands, where the attenuation hardly changed with frequency, and applies there only (in third octaves, 400 Hz to 2.5 kHz). Lower down the seat dip depends on the seats themselves, and higher up the fit has no data.

Here it applies to the last leg of each early path (the direct sound, a reflection or a diffracted path) to a listener above a surface whose material is `audience`: φ is the leg's elevation above that surface's plane, and d how far the leg runs over audience surfaces no higher than `CLEARANCE_M` above the listener's own height over the audience (so the stalls do not shade a balcony seat). The leg from the source to the first reflection is not attenuated, as in the fit, which therefore errs on the side of too little attenuation. The late part is left as it is; the statistical method reckons its diffuse field less the early paths as they would come without the audience, so what the seats take from them is lost and not made up by the tail.

`acoustic_spaces.engine.grazing.`**`CLEARANCE_M`** `= 2.0`

How much higher than the listener's ears above the audience a leg still counts as running over it (m).

`acoustic_spaces.engine.grazing.`**`FIT`** `= (3.4, 2.17, 0.02)`

Coefficients of the fit's polynomial in sin φ: a sin²φ + b sin φ + c.

`acoustic_spaces.engine.grazing.`**`FIT_RANGE_HZ`** `= (400, 2500)`

The bands the fit covers (Hz): the 500 Hz to 2 kHz octaves and their third octaves.

`acoustic_spaces.engine.grazing.`**`MAX_EAR_HEIGHT_M`** `= 2.0`

Highest a listener's ears may be above an audience surface for it to count (m).

`acoustic_spaces.engine.grazing.`**`apply_grazing`**`(model: TraceModel, listener: ndarray, paths: EarlyReflections)`

`paths` to `listener` with the grazing attenuation over the audience applied to their energies; unchanged when the model has no audience surfaces.

`acoustic_spaces.engine.grazing.`**`attenuation`**`(elevation_rad, distance_m) → ndarray`

The share of energy left after `distance_m` over the audience at `elevation_rad` above its plane (the fit above; the first metre is free).

`acoustic_spaces.engine.grazing.`**`audience_faces`**`(model: TraceModel) → ndarray`

(F,) faces whose material is audience seating.

```
acoustic_spaces.engine.grazing.fit_bands(model: TraceModel) → ndarray
```

(bands,) the model's bands the fit covers.

```
acoustic_spaces.engine.grazing.grazing_factors(model: TraceModel, listener: ndarray,
    paths: EarlyReflections, audience: ndarray) → ndarray | None
```

(paths, bands) share of each path's energy left by grazing incidence over the audience (1 outside the fit's bands), or None when the listener is not above an audience surface.

Wave-based solver

Wave-based simulation of the lowest octave bands (or their third-octave bands).

Rays and image sources treat sound as rays, which holds while the room is large next to the wavelength. At low frequencies it is not: small rooms ring at their modes, and sound bends round everything. The wave solver computes the sound field itself, with the finite-difference time-domain method (FDTD) on a cubic grid over the room, and its octave bands replace the ray-traced ones.

- One grid covers every octave band up to `max_band_hz`, with `points_per_wavelength` cells per wavelength at the top of that band's crossover. In a simulation in third-octave bands, those are the third octaves of these octave bands, and the grid is set by the top of the highest of them. Each octave higher costs about 16 times as much (cells 8 times smaller, steps twice as many). All bands share the grid, so a mode near a crossover is the same mode on both sides of it.
- Waves on the grid travel slower the shorter they are. `scheme` picks the stencil: the interpolated isotropic one (19 points, `ISOTROPIC_COURANT`) is 0.67 % slow at 8 cells per wavelength, within 0.03 % the same in every direction; the standard one (7 points, `COURANT`) up to 1.7 %, along the grid's axes. `auto` takes the isotropic one but round an open model, where only the standard one has the matched layer. Along the axes both are slowest (`slow_by()`); the run's details give how slow the top of the highest band is there (`wave_slow_percent`).
- The surfaces absorb as their materials do in each band: each material becomes a locally reacting surface whose admittance changes with frequency, fitted so that its random-incidence absorption matches the material's in every wave band (`acoustic_spaces.engine.impedance`). One run covers all the wave bands.
- Cells are air when the source reaches them without crossing a surface, so closed rooms, open models and thin screens all work. Round an open model, whose air reaches the edge of the grid, the grid grows by a perfectly matched layer of `LAYER_CELLS` cells that takes in the sound leaving it, and leaves room (`LAYER_ROOM`) for sound that runs along it, on the sides where no surface of the model lies between that sound and the layer (`_open_sides()`). Openings are not surfaces here: the air flows through them. Surfaces that let only part of the sound through are not modelled, so a model with any stops the wave solver with an error.
- `walls` says where the grid's walls lie: `"cell_faces"`, each surface on the cell faces nearest it, or `"fitted"`, where the model has it, cut exactly through the cells. Fitted walls take the standard stencil (`auto` takes it for them) and still air with one speed of sound: they are refused with the isotropic stencil, outdoors and in moving air.
- At these frequencies the source radiates the same way in all directions. The listener records the pressure and, from its gradient, the direction the sound comes from (first-order Ambisonics), so spatial outputs keep the low end's direction.
- The bands join the ray-traced ones through the zero-phase crossovers the impulse response synthesis uses, which sum to one; the lowest starts at `LOW_EDGE_HZ` (22 Hz) instead of 0 Hz. Their energies, for the room parameters, are counted through the octave filters of the analysis of measured responses.
- Outdoors, in wind and temperature that change with height (`Atmosphere`), the speed of sound changes from cell to cell: the still air's at the cell's height above the ground, plus the wind's component along the horizontal from the source, the effective speed of sound the rays and the parabolic equation take (`acoustic_spaces.engine.refraction`). The grid is then fine enough for the slowest sound and steps in time for the fastest, so it costs a little more than in still air. Sound that turns back towards the source, off a building, keeps the speed of sound of sound going away from it.

- With `wind="moving"` the air moves instead: each cell keeps the still air's speed of sound, and before each step both of the leapfrog's pressure levels are carried along with the wind, dq/dt = -v . grad q (the strong-stability-preserving Runge-Kutta method of third order); the still air's step follows, its boundaries, the matched layer and the source fixed to the ground. In the air's own frame that is the still air's step exactly, so a uniform wind gives the convected wave equation: downwind sound at c + v, upwind at c - v whichever way it turned. Where the wind changes, the term of the linearised Euler equations that the wind's shear puts on the air's velocity is left out, about `|dv/dz| / omega` of the rest. That scheme keeps the sound's energy only where no wind crosses a surface of the model and no cell gains or loses air, so the atmosphere's wind is first made to flow round the model's solids: on the faces between the cells, the wind of least change from the given one (each face's change weighed by one over the given speed there) that crosses no surface and leaves every cell as much as it enters it. A dead end, such as a room with one opening, then holds nearly still air whatever the wind's profile. It is a flow without wakes or eddies behind the solids, and at a sharp edge it speeds up, the more the finer the grid; the run says how fast it gets through a face between two cells (`wave_wind_fastest_m_s`), which must stay under half a cell a step. The carrying is of fourth order where the wind is the same along a row of cells and of second order where it changes; over open, flat ground, where the given wind already crosses nothing, it is carried as given, by fourth-order central differences. Over open ground it takes four to five times as long as the effective speed of sound's run; round solids its steps take about eight times as long, after the solve of the wind (see `validation/wave_wind.py`). It runs on the CPU (asked for the GPU, it says so and stays there). The matched layer's air is still: where the wind crosses its border, each cell's own sound is carried out and silence in, which takes energy and adds none, and a wave that leaves the moving air for the layer comes back by about half the wind's Mach number (see the notes of `fdtd.hpp`). The run is watched all the same: once the source's pulse has ended, the pressure squared over the grid's air, averaged over pieces of `WATCH_S`, may not rise to `WATCH_RISE` times its lowest so far (a `ValueError`).
- The grid is laid out on the CPU and stepped there, or with `device` on a GPU, through WebGPU (`acoustic_spaces.engine.gpu`); the two agree to rounding, in wind and temperature too (with the effective speed of sound).

The levels follow the ray tracer's convention: in free field, the direct sound at r metres is an impulse of height 1 / r.

`acoustic_spaces.engine.wave.`**`COURANT`** `= 0.5773502691896258`

Courant number of the 7-point (standard rectilinear) stencil: its stable limit.

`acoustic_spaces.engine.wave.`**`ISOTROPIC_COURANT`** `= 0.8660254037844386`

Courant number of the 19-point (interpolated isotropic) stencil: its stable limit.

`acoustic_spaces.engine.wave.`**`HIGH_PASS_S`** `= 0.12`

The source's high-pass, applied twice: once so that the air the source pushes into the room stops flowing, twice so that as much flows out again and a closed room's pressure comes back to rest. Its length in seconds; twice, it is flat within 0.2 dB above 20 Hz.

`acoustic_spaces.engine.wave.`**`LAYER_CELLS`** `= 8`

Cells of the matched layer round an open model, and its damping at the grid's edge (the damping rate times half a time step). Sound leaving the model comes back from it 72 to 74 dB down; from the grid's edge alone, 7 to 14 dB down (docs/validation.md, Open models).

acoustic_spaces.engine.wave.**LOW_EDGE_HZ** = 22.273863607376246

The wave solver's part of a response starts here (Hz), with the synthesis's crossover shape round it: an octave below the 63 Hz band's lower edge, so that band, skirts and all, comes from the wave solver. Further down a closed room's air is squeezed as a whole (the wave solver's source leaves no air in it, so this falls away), and the rays' response stays.

acoustic_spaces.engine.wave.**MAX_CELLS** = 40000000

Grids with more cells than this are refused (about 20 bytes each).

acoustic_spaces.engine.wave.**LINK_WEIGHTS** = {7: (0.3333333333333333, 0.0), 19: (0.25, 0.125)}

Each stencil's link weights, by its points: of an axial link and of an edge link (the Courant number squared shared out over the links, as the update takes them).

acoustic_spaces.engine.wave.**STENCILS** = {'isotropic': (19, 0.8660254037844386), 'standard': (7, 0.5773502691896258)}

The stencils' points, and their Courant numbers, by scheme.

acoustic_spaces.engine.wave.**WAVE_BANDS_HZ** = (63, 125, 250, 500)

The highest bands the solver may compute (`simulation.wave.max_band_hz`).

acoustic_spaces.engine.wave.**Scheme**

Which stencil steps the grid (see `wave_responses()`).

alias of `Literal`['auto', 'standard', 'isotropic']

acoustic_spaces.engine.wave.**Walls**

Where the grid's walls lie (see `wave_responses()`).

alias of `Literal`['cell_faces', 'fitted']

acoustic_spaces.engine.wave.**WindModel**

How the wave solver takes the wind (see `wave_responses()`).

alias of `Literal`['effective', 'moving']

class acoustic_spaces.engine.wave.**Admittance**(beta: float, conductance: ndarray = <factory>, quality: ndarray = <factory>, frequency_hz: ndarray = <factory>, achieved: ndarray | None = None, relaxation: ndarray = <factory>, relaxation_hz: ndarray = <factory>)

A locally reacting surface's specific admittance: `beta` plus one resonant branch per entry of `conductance` (its admittance at resonance, 1 / R), `quality` and `frequency_hz`, and one branch without mass per entry of `relaxation` (its admittance at high frequencies, 1 / R) and `relaxation_hz` (its corner frequency, S / (2 pi R)), as a porous ground's admittance needs.

achieved: ndarray | None = None

The band absorption the fit reached, or None when not fitted.

branches(rate: float) → ndarray

The branches as the wave solver takes them (branches, 3): mass over the time step, resistance, and stiffness times the time step.

class acoustic_spaces.engine.wave.**BandGrid**(band: int, cell: float, rate: float)

The grid of one band's run.

cell: float

Cell size (m).

rate: float

Sample rate of the run (Hz): the output rate over or times a whole number.

class acoustic_spaces.engine.wave.**WaveResponse**(rate: float, signals: ndarray, bands: int, cell: float, total_bands: int = 8, lead_s: float = 0.0, band_gain: ndarray | None = None, delay_s: float = 0.0)

The lowest octave bands (or third-octave bands) of one response, from the wave solver.

`signals` holds the run at `rate`: the pressure and the first-order directional parts (samples, 4), AmbiX W, X, Y and Z in room coordinates, from `lead_s` before emission on. Only the frequencies of the wave bands count.

`bands``: int`

The bands 0 to bands - 1 come from the wave solver.

`cell``: float`

Cell size of the grid (m).

`total_bands``: int = 8`

How many bands the whole response has: 8 octave or 24 third-octave bands.

`lead_s``: float = 0.0`

How long before emission `signals` start (s): the source's pulse is centred on emission, so a listener near the source hears it start before then.

`band_gain``: ndarray | None = None`

The energy gain of each wave band from the source's feed, or None for none.

`delay_s``: float = 0.0`

How long after emission the source starts playing (s).

`fed``(gain: ndarray, delay_s: float) → WaveResponse`

These bands with the energy `gain` of each and `delay_s` later.

`field``(sample_rate: float, n: int) → ndarray`

The first-order sound field (n, 4) of the wave bands together, from emission: W, X, Y, Z. The bands' filters take the run before emission too, and what they spread to before emission is left out, as it is of the rays' sound. They take the run after the n samples as well, where it goes on (`PAST_END_S`).

`high_pass``(signals: ndarray, sample_rate: float, late: bool = False) → ndarray`

`signals` (n, channels) without the frequencies the wave bands cover, which take them at a gain g: the rays' early paths at 1 - g, as they add to the wave bands' sound in pressure, or with `late` their noise at sqrt(1 - g^2), as it adds to the wave bands' tail in energy.

`energy``(time_step: float, slots: int, axis: ndarray | None = None, exponent: int = 2) → ndarray`

Energy per band and time slot (bands, slots), as the ray tracer counts it: the energy of the pressure through each band's filter (those of the analysis of measured responses, `acoustic_spaces.parameters.band_filter_gains()`) over that filter's share of the spectrum. The filters spread each arrival a little both ways in time; what they spread to before emission counts in the first slot.

With `axis` (a unit vector in room coordinates), that of a figure of eight facing it instead, from X, Y and Z: with `exponent` 1, the magnitude of its product with the pressure.

acoustic_spaces.engine.wave.**absorption_of_impedance**(impedance: ndarray) → ndarray

Random-incidence absorption of a locally reacting surface with a real normalised impedance (Paris's formula, for a diffuse field).

acoustic_spaces.engine.wave.**admittance**(absorption: ndarray) → ndarray

Real specific (normalised) admittance of a locally reacting surface whose random-incidence absorption is `absorption`: the inverse of Paris's formula, on the side of hard surfaces. Coefficients from `MAX_ABSORPTION` up get the most absorbing admittance.

acoustic_spaces.engine.wave.**fit_admittance**(absorption: Sequence[float], total: int, rate: float | None = None) → Admittance

The admittance whose random-incidence absorption, at `rate` steps a second (None: exactly), matches `absorption` in each of the lowest len(absorption) of the `total` (8 octave or 24 third-octave) bands.

acoustic_spaces.engine.wave.**band_grid**(band: int, sample_rate: float, speed_of_sound: float, points_per_wavelength: float = 8.0, total: int = 8, courant: float = 0.5773502691896258) → BandGrid

The coarsest grid with `points_per_wavelength` cells per wavelength at the top of band `band` (of `total`, 8 octave or 24 third-octave bands) whose time step is a whole number of the output's sample periods, or one of them over a whole number where the grid needs more steps per second than the output has samples. `courant` is the stencil's Courant number.

acoustic_spaces.engine.wave.**band_top**(band: int, total: int = 8) → float

Highest frequency band `band` of the `total` (8 octave or 24 third-octave) bands has (Hz): the top of its upper crossover.

acoustic_spaces.engine.wave.**grid_phase**(theta, stencil: int, ratio: float = 1.0) → ndarray

The phase w T a wave turns through in a time step on the `stencil`-point grid, from the phase it turns through across a cell along each axis, `theta` (…, 3), k h: the stencil's dispersion relation, 2 - 2 cos(w T) = ratio^2 sum over the links of g_k (1 - cos(k . d_k)). `ratio` is the speed of sound over the one the grid's time step is set by (in wind and temperature, below 1 in all but the fastest air).

acoustic_spaces.engine.wave.**slow_by**(stencil: int, per_wavelength: float, ratio: float = 1.0) → float

How much slower than it should (a fraction) a wave of `per_wavelength` cells per wavelength travels on the `stencil`-point grid along its axes, the way it is slowest with either stencil; `ratio` as in `grid_phase()` (below 1, slower still). The 19-point stencil at 8 cells per wavelength: 0.67 %, at 5: 1.85 %; the 7-point one 1.74 % and 4.56 %.

acoustic_spaces.engine.wave.**source_signal**(rate: float, cell: float, courant: float = 0.5773502691896258, speed: float = 1.0, *, level: bool = True) → tuple[ndarray, int]

What the source adds to the grid on each step, and its delay in steps.

An impulse, low-passed (so the grid carries no frequencies it cannot) and high-passed twice (so it leaves no air in the room), scaled so that in free field the pressure at r metres is that impulse over r. `courant` is the stencil's Courant number, which says which stencil it is for and sets the low-pass's cutoff (`_cutoff()`); `speed` the speed of sound at the source over the grid's (1 where it is the same everywhere).

On the 19-point grid a point source's sound comes out louder than what the source adds, the more the higher the frequency: by 9 / (5 + 4 cos wT) in pressure, w T the frequency's turn over a time step (the far field of the scheme's own response to one cell, by stationary phase: 3 / (2 + cos kh) along an axis, kh the turn over a cell there, which the scheme ties to the frequency by cos kh = 1 - 4 (1 - cos wT) / 3). That is the gain along the axes and the cube's diagonals; along a face's diagonals it is up to 0.03 dB more at eight cells per wavelength, and the same within 0.01 dB over a band: 0.06 and 0.35 dB over the 63 and 125 Hz bands of a grid up to the 125 Hz band. Three taps, 2/9, 5/9 and 2/9, have the inverse gain exactly and are part of the signal, with a step of delay (where the source's speed of sound is `speed` times the grid's, the side taps are over the square of `speed`, the middle one what is left of one); `level=False` gives the signal without them, the impulse that the signal with them stands for. The 7-point grid's gain changes with the direction (twelve cells away, over that 125 Hz band: none along an axis, 0.25 dB along a face's diagonals and 0.32 to 0.34 dB along the cube's, by the places of the source and the listener within their cells; validation/wave_solver.py), so no signal can undo it.

```
acoustic_spaces.engine.wave.wave_gains(n_fft: int, sample_rate: float, bands: int, total:
    int = 8) → ndarray
```

Zero-phase gains (bands, n_fft // 2 + 1) of the lowest `bands` of the `total` (8 octave or 24 third-octave) bands as the wave solver covers them: the synthesis's band gains, the lowest one starting at `LOW_EDGE_HZ`.

```
acoustic_spaces.engine.wave.wave_responses(model, source: ndarray, listeners: ndarray,
    names: Sequence[str], *, max_band_hz: int, length_s: float, sample_rate: float,
    speed_of_sound: float, points_per_wavelength: float = 8.0, threads: int = 0, device:
    Literal['cpu', 'gpu', 'auto'] = 'cpu', scheme: Literal['auto', 'standard',
    'isotropic'] = 'auto', say: Callable[[str], None] | None = None, label: str = '',
    wind: Literal['effective', 'moving'] = 'effective', walls: Literal['cell_faces',
    'fitted'] = 'cell_faces') → tuple[list[WaveResponse], dict[str, float]]
```

The wave bands of the responses from `source` to each of `listeners` in the room of `model` (a `TraceModel`), and run details. `device` says where the grid is stepped (see `gpu`); `scheme` which stencil steps it: `"isotropic"` (19 points), `"standard"` (7 points) or `"auto"`, the isotropic one but round an open model, where only the standard one has the matched layer. `wind` says how it takes the wind outdoors: `"effective"`, as a speed of sound changed by the wind's component along the horizontal from the source, or `"moving"`, as air that moves (see the module's notes). `walls` says where the grid's walls lie: `"cell_faces"`, on the cell faces nearest the surfaces, or `"fitted"`, where the model has them, cut through the cells; fitted walls take the standard stencil whatever the `scheme` but `"isotropic"`, which they refuse, as they refuse an outdoor model and moving air. The fit's own refusals of a model (a crack, faces laid on one another, a model reaching into the matched layer) are `ValueError` starting "fitted walls:", raised before any step.

Locally reacting surfaces for the wave solver: frequency-dependent admittances fitted to a material's absorption coefficients.

A material table gives each band's random-incidence absorption. The wave solver needs the surface's specific admittance (the normal velocity times rho c over the pressure), as a function of frequency it can step in time. `Admittance` is a constant conductance beta plus parallel mass-resistance-stiffness branches,

> Y(s) = beta + sum_j 1 / (M_j s + R_j + S_j / s),

each resonant (at f_j, with quality factor Q_j) and passive, so the scheme stays stable whatever the values. `fit_admittance()` chooses beta and the branches so that the random-incidence absorption of Y (Paris's formula for a locally reacting surface), counted through the analysis's band filters, matches the material's in every band the wave solver covers: one branch at each band's centre and one beyond each end. It fits the admittance the scheme actually has at its time step, which bends a little from Y towards the grid's Nyquist frequency.

A locally reacting surface absorbs at most `MAX_ABSORPTION` (0.951) at random incidence; higher coefficients get that. The band filters overlap, so bands whose coefficients change steeply from one to the next (0.1 to 0.6 in an octave) are matched only roughly; `Admittance.achieved` holds what the fit reached.

acoustic_spaces.engine.impedance.**MAX_ABSORPTION** = 0.951

Highest random-incidence absorption a locally reacting surface reaches (Paris's formula), and the normalised impedance that reaches it.

class acoustic_spaces.engine.impedance.**Admittance**(beta: float, conductance: ndarray = <factory>, quality: ndarray = <factory>, frequency_hz: ndarray = <factory>, achieved: ndarray | None = None, relaxation: ndarray = <factory>, relaxation_hz: ndarray = <factory>)

A locally reacting surface's specific admittance: `beta` plus one resonant branch per entry of `conductance` (its admittance at resonance, 1 / R), `quality` and `frequency_hz`, and one branch without mass per entry of `relaxation` (its admittance at high frequencies, 1 / R) and `relaxation_hz` (its corner frequency, S / (2 pi R)), as a porous ground's admittance needs.

achieved: ndarray | None = None

The band absorption the fit reached, or None when not fitted.

branches(rate: float) → ndarray

The branches as the wave solver takes them (branches, 3): mass over the time step, resistance, and stiffness times the time step.

acoustic_spaces.engine.impedance.**absorption_of_impedance**(impedance: ndarray) → ndarray

Random-incidence absorption of a locally reacting surface with a real normalised impedance (Paris's formula, for a diffuse field).

acoustic_spaces.engine.impedance.**admittance**(absorption: ndarray) → ndarray

Real specific (normalised) admittance of a locally reacting surface whose random-incidence absorption is `absorption`: the inverse of Paris's formula, on the side of hard surfaces. Coefficients from `MAX_ABSORPTION` up get the most absorbing admittance.

acoustic_spaces.engine.impedance.**band_absorption**(admittance: **Admittance**, bands: int,

`total: int, rate: float | None = None) → ndarray`

The random-incidence absorption of `admittance` in the lowest `bands` of the `total` bands, counted as `fit_admittance()` counts it.

`acoustic_spaces.engine.impedance.band_weights(f: ndarray, bands: int, total: int) → ndarray`

How much each of the frequencies `f` (on a log grid) counts in each of the lowest `bands` of the `total` (8 octave or 24 third-octave) bands (bands, len(f)): the energy of white noise through the analysis's band filters.

`acoustic_spaces.engine.impedance.fit_admittance(absorption: Sequence[float], total: int, rate: float | None = None) → Admittance`

The admittance whose random-incidence absorption, at `rate` steps a second (None: exactly), matches `absorption` in each of the lowest len(absorption) of the `total` (8 octave or 24 third-octave) bands.

`acoustic_spaces.engine.impedance.fit_impedance(admittance: Callable[[ndarray], ndarray], bands: int, total: int, rate: float | None = None) → Admittance`

The admittance whose value, at `rate` steps a second (None: exactly), matches that of a surface with an impedance, `admittance(f)` (normalised, complex, in the engine's convention, at frequencies f in Hz), over the lowest `bands` of the `total` bands: its reflection at every angle and its phase, not only its random-incidence absorption, as a ground's dip needs.

A porous ground's admittance grows with frequency at a nearly constant phase, as no resonant branch does: the fit is `beta` and branches without mass whose corner frequencies are spread evenly in log frequency, whose sum makes that shape (as a ladder of resistors and capacitors makes a fractional-order element). With the corners fixed the admittance is linear in the branches' conductances, which the fit finds by least squares, at least 0 each, with the misses relative to the admittance's size.

`acoustic_spaces.engine.impedance.random_incidence_absorption(admittance: ndarray) → ndarray`

Random-incidence absorption of a locally reacting surface with the complex specific admittance `admittance`: Paris's formula, the integral over incidence of `1 - |R|^2` weighted by `sin(2 theta)`, in closed form.

The wave solver's time steps on a GPU, through WebGPU.

`GpuWaveSolver` takes a grid that the C++ solver (`acoustic_spaces._native.WaveSolver`) has set up, with its air, boundaries, branches, matched layer, source and probes, and a speed of sound per cell where the grid has one (outdoors in wind and temperature), and steps it with one WGSL compute shader on any GPU that wgpu reaches: Vulkan (AMD, Intel, NVIDIA, and Mesa's software lavapipe), Metal or Direct3D 12. It needs the optional `gpu` extra (`pip install "acoustic-spaces[gpu]"`). Its arithmetic is the CPU's, in single precision, so the two agree to rounding.

Each time step is one dispatch: one thread for each cell of the grid (the interior air cells take the plain update and the rest return), then one for each boundary cell, each matched-layer cell and each probe. The two pressure fields swap from one step to the next, and so do the two sets of branch velocities, by the step's parity.

`acoustic_spaces.engine.gpu.`**`Device`**

Where the wave solver runs: `"auto"` takes a GPU when wgpu finds a hardware one (and `acoustic-spaces tune` did not find the CPU faster for the run's length).

alias of `Literal`['cpu', 'gpu', 'auto']

`class` `acoustic_spaces.engine.gpu.`**`GpuWaveSolver`**`(solver)`

Steps the grid of `solver` (a fresh or reset `acoustic_spaces._native.WaveSolver`) on the GPU: `run()` does what the solver's `run` does, from the GPU's own state.

`run``(signal: ndarray) → ndarray`

Advances one step per sample of `signal` (added at the source) and returns the pressure at every probe before each step (probes, steps).

`acoustic_spaces.engine.gpu.`**`gpu_name`**`() → str | None`

The GPU the wave solver would run on, as `"<device> (<backend>, <type>)"`, or None when wgpu is missing or finds none.

`acoustic_spaces.engine.gpu.`**`hardware_gpu`**`() → bool`

Whether wgpu finds a hardware GPU (not a software renderer such as lavapipe).

`acoustic_spaces.engine.gpu.`**`use_gpu`**`(device: Literal['cpu', 'gpu', 'auto'], steps: int | None = None) → bool`

Whether the wave solver runs on the GPU for the setting `device`, for a run of `steps` steps. `auto` takes a hardware GPU: when `acoustic-spaces tune` timed both on this computer, only if it steps that many times clearly faster (see `acoustic_spaces.engine.tuning`).

How big a run is: the threads it works on and the memory its late part needs.

What grows without bound with the settings is the late part's energy: one array per listener (or map point) and band, as long as the response, and at a listener one more per arrival direction. Every source's is kept to the end of the run, and while one is traced the tracer needs more: on the CPU a copy for each thread that sums rays (at most `_native.COPY_MEMORY` together, one at least), on a GPU its buffers. So a run needs memory in proportion to `simulation.ir_length_s`, the listeners and `late.directions`, the map points (`coverage.spacing`) and the sources: those traced for their late part, as a source or a listener whose late part is a mix of others' (a walk's places between the traced ones, `simulate`'s `late_from`) holds none. On the CPU the rays cost time, not memory. A run whose late part would need more memory than the computer has is refused before it starts (`check()`), and `warnings()` notes one that would need more than half of it. A map is checked again once its first source point is traced, with the early paths found there at every point (`held_bytes()`). The estimate takes the threads and the devices of the computer it runs on: with `late.device` gpu, both or auto (where there is a hardware GPU), the worst of the CPU, the GPU and the two together.

Threads above the computer's cores run as its cores: more would only take turns on them, each with its copy (`worker_threads()`).

class acoustic_spaces.engine.run_size.**Need**(bytes: float, what: str, advice: str)

Bytes of memory a run's late part needs, what for and what to change for less.

note(memory: int) → str

What the run needs against `memory` bytes (the computer's).

acoustic_spaces.engine.run_size.**check**(need: **Need**, memory: int | None = None) → None

Raises ValueError when `need` is more than `memory` bytes (by default the computer's, `machine_memory()`).

acoustic_spaces.engine.run_size.**each_on_threads**(function: Callable[[_Item], _Result], items: Iterable[_Item], threads: int) → list[_Result]

`function` of each of `items`, in their order, on at most `threads` threads at once, so that at most `threads` of the calls are in progress at any time. With one thread (or one item) the calls run one after another in the calling thread, with no pool.

acoustic_spaces.engine.run_size.**held_bytes**(value: object) → float

Bytes the arrays in `value` take (in lists, tuples and dataclasses, each array once, with its header), such as a map's early paths at every point.

acoustic_spaces.engine.run_size.**late_call**(results: float, bands: int, groups: int, *, splitting: bool, threads: int, rays: int, devices: Sequence[str] = ('cpu',), estimates: float = 0.0) → float

Bytes one run of the late part's tracer takes at its peak, whose arrays take `results` bytes in `bands` bands traced in `groups` groups of bands (`band_groups()`), and the energy as each batch of rays gives it `estimates`: the arrays the groups are joined in, the sums of the runs before while the next is traced (`splitting`: a first batch, or batches traced in runs), a group's estimates, and a group's own arrays, on the worst of `devices`: on the CPU with its threads' copies (at most `_native.COPY_MEMORY` together, one at least), on a GPU with its buffers (`run_memory()`), or on both with the sums of the parts and a part on each.

acoustic_spaces.engine.run_size.**late_devices**(setting: str | None) → tuple[str, ...]

Where the late part may trace its rays on this computer for `late.device`: always on the CPU, which traces what a GPU declines; on the GPU for gpu; and on both too for both, and for auto where there is a hardware GPU.

acoustic_spaces.engine.run_size.**machine_memory**() → int | None

Bytes of memory this computer has, or the limit of the container the run is in where that is lower; None where neither can be read.

acoustic_spaces.engine.run_size.**map_need**(room: **Room**, settings: Simulation, sources: Sequence, points: int, corners: np.ndarray, grid: float, threads: int) → **Need**

The memory a coverage map of about `points` points within `corners` (those of its extent) needs: `grid` bytes while its grid is laid out, and with stochastic ray tracing the late energy at every point from each point of every source, kept to the end, and while one is traced, the tracer's own (`late_call()`; maps are traced on the CPU).

```
acoustic_spaces.engine.run_size.simulation_need(room: Room, settings: Simulation,
    sources: Sequence, listeners: Sequence, threads: int, groups: int = 0, mixed:
    Collection[str] = ()) → Need
```

The memory the late part of a simulation of `sources` at `listeners` needs: the energy of every source, kept to the end; that of a line or surface source's points summed as each is traced; and while one is traced, the tracer's own (`late_call()`, on the devices `late_devices()` gives). Then, for a source with a feed of its own, a listener's copy at a time while its response takes it (four arrays as large at most). `groups` face groups (`simulate(groups=...)`) add their moments to each listener's arrays, and batches of rays (`late.batches`) the energy as each gives it.

`mixed` names the sources and the listeners whose late part is a mix of others' (`simulate`'s `late_from`, with the stochastic ray tracer): such a source is not traced and keeps no energy, and a point source's energy is traced and kept at the other listeners alone (a line's or a surface's is traced at those and mixed for every listener as its points are summed).

Its advice gives what the run would need with each setting it may lower changed on its own (`_lowered()`), the least first.

```
acoustic_spaces.engine.run_size.warning(need: Need, memory: int | None = None) → str |
    None
```

A note when `need` is more than half of `memory` bytes (by default the computer's), else None.

```
acoustic_spaces.engine.run_size.worker_threads(threads: int) → int
```

The threads a run works on for `threads` asked (`performance.threads`): as many as the computer has cores for 0 or more.

The late stage on a GPU

Where the late stage traces its rays: on the CPU, on a GPU, or on both at once.

`simulation.late.device` picks it:

- `cpu` (the default): the compiled core, on `simulation.performance.threads`.
- `gpu`: `acoustic_spaces.engine.gpu_tracer`, through WebGPU (the `gpu` extra).
- `both`: the two share the rays. Each takes the next share of them as soon as it is done with the last, so the faster one traces more.
- `auto`: whichever of the three is fastest on this computer for the run's response length. It takes the choice `acoustic-spaces tune` kept for that length (see `acoustic_spaces.engine.tuning`); without one, a short trial traces a few thousand rays on the scene's own model, for the run's own length, on the CPU and on the GPU and keeps the fastest for that range of lengths. It takes the CPU, the safe choice, when wgpu finds no GPU or only a software one, or when neither the GPU nor both together would be clearly faster. With the trial switched off (`set_trial()`, `--no-device-trial`), `auto` takes the choice kept for the nearest length, or the CPU when there is none.

What only the CPU traces (sound through partitions and openings, a source or a listener too near a surface or the line where two meet for the GPU's rays, vegetation of more media than the GPU keeps) runs there, whatever the setting, and so does a run the GPU declines once it has it (one whose results its buffers cannot hold, or a ray through more volumes of vegetation at once than its tables hold): the plan `trace()` returns says so. Outdoors the GPU bends the rays in the atmosphere and takes them through vegetation as the CPU does.

The two agree statistically, not ray for ray (the GPU works in single precision and draws its own random numbers). With `cpu` a run gives the same result every time, and with `gpu` too until the weight window fills a family of copies (`max_family`): which copies split then depends on the order the GPU's threads come in. With `both` which rays each one traces depends on how fast each goes. Results then vary from run to run within the noise of the ray count.

`acoustic_spaces.engine.ray_device.`**`TRIAL_FLAG`** `= '--no-device-trial'`

The command-line switch (`acoustic-spaces simulate` and the editor) for set_trial.

`acoustic_spaces.engine.ray_device.`**`Device`**

The settings of `simulation.late.device`.

alias of `Literal`['cpu', 'gpu', 'both', 'auto']

`class` `acoustic_spaces.engine.ray_device.`**`Plan`**`(device: str, cpu_rate: float = 1.0, gpu_rate: float = 1.0, reason: str = '')`

Where rays go: `device` is cpu, gpu or both; `cpu_rate` and `gpu_rate` (rays per second, from the trial; 1 for unknown) share them; `reason` says why, when it differs from the setting.

`acoustic_spaces.engine.ray_device.`**`choose`**`(setting: str | Plan, values: dict) → Plan`

Where to trace a run with the arguments `values` of `_native.trace_stochastic` for the device `setting`, or as the `Plan` given (as `acoustic-spaces tune` times each).

`acoustic_spaces.engine.ray_device.`**`release`**`() → None`

Lets go of what the GPU keeps from one of a run's calls to the next, as the run ends (nothing where the GPU's tracer was never loaded).

`acoustic_spaces.engine.ray_device.`**`set_trial`**`(enabled: bool) → None`

Switch `auto`'s trial on or off for this session (off: the saved choice).

`acoustic_spaces.engine.ray_device.`**`trace`**`(plan: Plan, values: dict, keep: bool = False) → tuple[tuple, Plan]`

`_native.trace_stochastic(**values)` where `plan` says, and the plan it followed: the CPU, with the GPU's reason, where the GPU declined the rays. With `keep`, the GPU keeps its rays in flight for the next call, until `release()`.

`acoustic_spaces.engine.ray_device.`**`trial`**`(values: dict) → tuple[float, float]`

Rays per second on the CPU and on the GPU for a run like `values` (the arguments of `_native.trace_stochastic`): a few thousand rays traced on each, past the run's own random streams.

How fast the CPU and the GPU are on this computer, and what `auto` picks from it.

`acoustic-spaces tune` (`tune()`) times the late stage on the CPU, on the GPU and on both at once, for several response lengths, and the wave solver on the CPU and on the GPU, on one scene. It keeps the fastest for each in the user's cache folder, for the computer (its GPU and CPU thread count):

- the late stage's `auto` (`ray_device`) takes the choice for the run's response length: the one for the shortest timed length at least as long, else the longest. Without `tune`, its trial keeps a choice there too, for its range of lengths (up to 0.5, 1, 2, 4 s and so on).
- the wave solver's `auto` compares the timed costs for the run's step count: per grid cell, a set-up and a cost per step on each device. Without `tune`, it takes a hardware GPU when wgpu finds one.

A software GPU (such as lavapipe or SwiftShader) is timed like any other, but `auto` never takes it.

`auto` applies to scenes set to it, and to scenes that set no device once `tune` has made it this computer's default (`set_default()`); otherwise those run on the CPU, so that a scene gives the same result on every computer.

```
acoustic_spaces.engine.tuning.LENGTHS = (0.5, 1.0, 2.0, 4.0)
```

Response lengths `tune` times the late stage at (s).

```
acoustic_spaces.engine.tuning.SCENE = 'concert_hall'
```

The example scene `tune` times when given none.

```
class acoustic_spaces.engine.tuning.Tuning(scene: str, machine: str, rays: int, late:
    list[LateTiming] = <factory>, wave: WaveTiming | None = None, wave_skipped: str =
    '', software: bool = False, file: Path = <factory>)
```

What `tune()` timed and kept.

```
keep() → None
```

Keep the choices for this computer, where `auto` reads them (`tune()` does unless told not to).

```
summary() → str
```

The timings as a table, and what `auto` will pick.

```
acoustic_spaces.engine.tuning.cache_file() → Path
```

Where the choices are kept: `acoustic-spaces/devices.json` in the user's cache folder.

```
acoustic_spaces.engine.tuning.keep_late(max_time: float, choice: dict) → None
```

Keep the late stage's `choice` for the range of `max_time`.

```
acoustic_spaces.engine.tuning.keep_wave(costs: dict) → None
```

Keep the wave solver's `costs` (see `wave_costs()`).

```
acoustic_spaces.engine.tuning.late_choice(max_time: float) → dict | None
```

The late stage's choice for responses `max_time` long (s): the one kept for the shortest range at least that long, else for the longest; None when none is kept. It has `device`, `cpu_rate` and `gpu_rate` (rays per second; only their ratio counts) and `from` (`tune` or `trial`).

`acoustic_spaces.engine.tuning.late_choices() → dict[float, dict]`

The late stage's choices kept for this computer, by range of response lengths (its upper end, s).

`acoustic_spaces.engine.tuning.late_entry(max_time: float) → dict | None`

The late stage's choice kept for exactly the range of `max_time`, or None.

`acoustic_spaces.engine.tuning.length_key(max_time: float) → str`

The range of response lengths `max_time` (s) falls in, by its upper end: a power of two seconds (0.5, 1, 2, 4, …).

`acoustic_spaces.engine.tuning.machine() → str`

The computer the choices are kept for: its GPU and CPU thread count.

`acoustic_spaces.engine.tuning.saved() → dict`

What is kept for this computer: `{"late": {length: choice}, "wave": costs, "default": device}`.

```
acoustic_spaces.engine.tuning.tune(room, *, rays: int | None = None, lengths:
    Sequence[float] = (0.5, 1.0, 2.0, 4.0), wave_lengths: Sequence[float] | None =
    (0.05, 0.2), threads: int | None = None, say: Callable[[str], None] | None = None,
    keep: bool = True) → Tuning
```

Time the late stage and the wave solver of `room` (a loaded `Room`) on the CPU and the GPU, keep the fastest for this computer and return the timings.

The late stage traces the scene's first source to its listeners with the scene's settings, `rays` per band (default: the scene's), once for each response length of `lengths` (s) on each device; the listeners too near a surface for the GPU (`gpu_tracer.listeners_too_near`) are left out. The wave solver runs the scene's wave settings (enabled or not) for the two `wave_lengths`; None leaves it out. `threads` defaults to the scene's. `keep=False` returns the timings without saving them; `Tuning.keep()` saves them later (the editor does that on OK).

`acoustic_spaces.engine.tuning.wave_costs() → dict | None`

The wave solver's timed costs, `{"cpu": [a, b], "gpu": [a, b]}`: seconds per grid cell to set up (a) and per step (b); None when not timed.

`acoustic_spaces.engine.tuning.wave_pick(costs: dict) → str`

What the wave solver's `auto` takes with `costs`, in words.

`acoustic_spaces.engine.tuning.wave_prefers_gpu(steps: int) → bool | None`

Whether the GPU steps a grid `steps` times clearly faster than the CPU did when timed; None when not timed.

The late stage's stochastic ray tracer on a GPU, through WebGPU.

`trace_stochastic()` takes the arguments of the compiled core's `acoustic_spaces._native.trace_stochastic` and returns what it does, computed by one WGSL compute shader on any GPU wgpu reaches (see `acoustic_spaces.engine.gpu`). It traces what the CPU does: detector spheres and diffuse or specular rain, vector-based or random scattering, the scattering of a surface's size, air absorption, the early stage's paths left out, arrival directions, the weight window that splits rays and thins them out, and the face groups that re-weight the absorption (each ray counts its reflections off each group, in 16 bits, and the listeners sum the moments of those counts); outdoors, the rays bent by the atmosphere, followed in chords as the CPU follows them (`cpp/include/acoustic_spaces/atmosphere.hpp`), and the plants that absorb and scatter them (`vegetation.hpp`), with the stems' and the foliage's rain. It leaves to the CPU (raising `Unsupported`) sound through surfaces into other rooms, a source or a listener nearer a surface, or the line where two meet, than its rays tell apart (`too_near()`), runs whose results its buffers cannot hold, and vegetation of more than `MAX_MEDIA` media or so many volumes along one ray that its fixed tables run out of room.

Rays start where the CPU's do (`_native.ray_starts`: the same directions and energies for the same seed), then go their own way: the GPU draws its random numbers from streams of its own (xoshiro128**) and works in single precision, so the two agree statistically, not ray for ray. Each ray adds its energy to the listeners' time slots with atomic additions of 64-bit fixed-point numbers (two 32-bit words with a carry, rounded at random to the nearest step, so that sums are exact and unbiased; WGSL has no atomic floating-point addition), scaled so that a window's sums stay under 2**40 times the reflections per ray, where the listeners' sums grow with the ray count (and the GPU gives up a run whose sums overflow).

The rays advance in rounds of a few reflections each, so that no dispatch runs long enough for the operating system to reset the GPU. Rays still going at the end of a round, and the copies split off during it, are packed into the other half of a buffer for the next round, and new rays from the source fill it up to half its size; the other half takes the copies, and a split that finds no room left is not made (which leaves every expectation as it was). A ray's copies so advance together, where the CPU follows each to its end before it takes the next: a family that reaches its limit (`max_family`) reaches it sooner here, its copies made earlier on their way. That leaves the expected energies as they were too, but where the weight window asks for more copies than a family may have, the GPU traces more reflections in the late tail than the CPU (in the shoebox example, up to twice as many), and its tail varies less. Which of a family's copies split before it is full depends on the order the threads come in, so such runs differ from one another within the noise of the ray count. Bent rays take many chords to a leg, so their rounds hold fewer rays.

```
acoustic_spaces.engine.gpu_tracer.MAX_MEDIA = 16
```

Most media of vegetation (the shader keeps a scattering rate per medium), and most volumes along one piece of a ray (more are dropped, and the run declined).

```
class acoustic_spaces.engine.gpu_tracer.GpuTracer(bvh)
```

The tree of `bvh` (an `acoustic_spaces._native.Bvh`) and its faces' normals on the GPU, to trace rays in.

box

The model's box (its lowest and highest corner) in the shader's coordinates.

trace(keep: bool = False, **arguments) → tuple

Traces rays (see `_trace()` for the arguments), one run at a time. With `keep`, keeps its buffer of rays in flight for the next call, until `release()`.

release() → None

Lets go of the rays in flight it keeps from one call to the next.

bent_chords(origins, directions, budget, away, atmosphere, wind_direction, ground, chord_sagitta) → tuple

As `_native.bent_chords`, on the GPU: one chord of each ray bent by the atmosphere (see `atmosphere.hpp`), from `origins` along `directions` (unit), at most `budget` metres of arc, stopping where a ray turns back towards the plane whose normal is its row of `away` (zeros: none). Returns (ends, directions at the ends, arc lengths, times, turned), for the tests.

exception acoustic_spaces.engine.gpu_tracer.**Unsupported**

The GPU tracer does not do what a run asks for; the CPU's does.

acoustic_spaces.engine.gpu_tracer.**release**() → None

Lets go of the rays in flight that GPU tracers keep from one call to the next (calls with `keep`), as a run's late part ends.

acoustic_spaces.engine.gpu_tracer.**run_memory**(rays: int, bands: int, results: float) → float

Bytes a run of `rays` rays in `bands` bands takes on the GPU and off it, for results (the energy and directional arrays, in 64-bit floats) of `results` bytes: the results, the GPU's sums of them and one part of those read back at a time; the rays' starts (`_native.ray_starts`), their copies for the GPU and a word per ray for its family; and the GPU's rays in flight (at most `_MOST_RAYS` of `21 + 3 * bands` words and those of the most face groups, twice). A GPU built into the processor takes its buffers from the computer's memory.

acoustic_spaces.engine.gpu_tracer.**trace_stochastic**(bvh, source, directivity_a, orientation, directivity_table, absorption, scattering, air, surface_size, band_hz, vector_scattering, size_from_order, transmission, face_room, source_room, receivers, directions, rays, max_time, time_step, dir_time_step, speed_of_sound, receiver_radius, diffuse_rain, specular_rain, covered_order, covered_time, seed, threads, window_time_step, first_ray, split_target, split_ratio, max_split, max_family, room_level, face_group, groups, moment_time_step, atmosphere=None, wind_direction=None, ground=0.0, chord_sagitta=0.02, plants=None, plant_outline=None, plant_absorption=None, plant_scattering=None, plant_asymmetry=None, plant_upright=None, gather_ratio=0.0, face_kind=None, covered_ground=0, *, total: int | None = None, keep: bool = False) → tuple

As `acoustic_spaces._native.trace_stochastic`, on the GPU. `total` (default `rays`) is the number of rays the source's energy is shared between, for tracing part of a run's rays here and the rest elsewhere. With `keep`, the GPU keeps its buffer of rays in flight for the next call, until `release()`, as a run's late part does; without, the call makes and lets go of its own. Raises `Unsupported` for what only the CPU traces: transmission into a room (see `passes_sound()`), a source or a listener too near a surface (`too_near()`) and vegetation of more media than `MAX_MEDIA`.

`acoustic_spaces.engine.gpu_tracer.`**`tracer`**`(bvh) → GpuTracer`

The GPU tracer of `bvh`, made once per model.

Ray queries against a room model on a GPU, through WebGPU.

`GpuBvh` uploads the compiled core's bounding volume hierarchy (see `acoustic_spaces._native.Bvh.layout`) and answers the CPU's batch queries, `Bvh.intersect` (the nearest face along each ray) and `Bvh.occluded` (whether any face blocks it), with one WGSL compute shader on any GPU wgpu reaches (see `acoustic_spaces.engine.gpu`). It needs the optional `gpu` extra. It is the ray search a tracer on the GPU builds on; `validation/rays_gpu.py` times it against the CPU's.

The GPU works in single precision, where the CPU's double precision needs next to no slack. So the model is moved to centre its box on the origin (which keeps coordinates small), every box of the tree is widened by `BOX_PAD` of its own size, and every triangle by `EDGE` across each edge, so that a ray through an edge or a corner two faces share hits one of them instead of slipping between. The nearest face along a ray is then the CPU's, but for rays meeting two faces at nearly the same point, and the distance agrees to single-precision rounding.

```
acoustic_spaces.engine.gpu_rays.BVH_WGSL = "\nconst NONE: u32 = 0xffffffffu;\nconst FAR:
    f32 = 3.0e38;\nconst EDGE: f32 = 0.0001;\nconst STACK: u32 = 64u;\n\nstruct Hit
    {\n    t: f32,\n    face: u32,\n}\n\n// 1 / d per axis, with axes the ray runs
    (nearly) parallel to kept finite.\nfn reciprocal(d: vec3<f32>) -> vec3<f32> {\n
    let tiny = abs(d) < vec3<f32>(1e-20);\n    let away = select(vec3<f32>(1e-20),
    vec3<f32>(-1e-20), d < vec3<f32>(0.0));\n    return 1.0 / select(d, away,
    tiny);\n}\n\n// Where the ray enters the box of `node`, if it does before t_max
    (else FAR).\nfn enter(node: u32, o: vec3<f32>, inv: vec3<f32>, t_max: f32) -> f32
    {\n    let lo = bitcast<vec3<f32>>(nodes[2u * node].xyz);\n    let hi =
    bitcast<vec3<f32>>(nodes[2u * node + 1u].xyz);\n    let ta = (lo - o) * inv;\n
    let tb = (hi - o) * inv;\n    let t_in = min(ta, tb);\n    let t_out = max(ta,
    tb);\n    let t0 = max(max(t_in.x, t_in.y), max(t_in.z, 0.0));\n    let t1 =
    min(min(t_out.x, t_out.y), min(t_out.z, t_max));\n    return select(FAR, t0, t0 <=
    t1);\n}\n\n// Moller-Trumbore, two-sided, the edges widened by EDGE: t, or a
    negative value.\nfn hit_triangle(place: u32, o: vec3<f32>, d: vec3<f32>) -> f32
    {\n    let v0 = bitcast<vec3<f32>>(triangles[3u * place].xyz);\n    let e1 =
    bitcast<vec3<f32>>(triangles[3u * place + 1u].xyz) - v0;\n    let e2 =
    bitcast<vec3<f32>>(triangles[3u * place + 2u].xyz) - v0;\n    let p = cross(d,
    e2);\n    let det = dot(e1, p);\n    let flat = abs(det) < 1e-30;\n    let inv_det =
    1.0 / select(det, 1.0, flat);\n    let s = o - v0;\n    let u = dot(s, p) *
    inv_det;\n    let q = cross(s, e1);\n    let v = dot(d, q) * inv_det;\n    let
    inside = !flat && u >= -EDGE && u <= 1.0 + EDGE && v >= -EDGE && u + v <= 1.0 +
    EDGE;\n    return select(-1.0, dot(e2, q) * inv_det, inside);\n}\n\n// The nearest
    face along o + t d (d of unit length) with t in (t_min, t_max), skipping\n// face
    `skip`, or (t_max, NONE); with `any_hit`, the first face found, which is enough\n//
    to know the way is blocked. Of two children the ray enters, the nearer goes
    first;\n// the other waits and is dropped if a hit nearer than its box turns up
    meanwhile.\n// (No early exits from the loops: with them, the late tracer's searches
    on Direct3D 12\n// missed faces and could run forever.)\nfn search_ray(o: vec3<f32>,
    d: vec3<f32>, t_min: f32, t_max: f32, skip: u32,\n              any_hit: bool) ->
    Hit {\n    var best = Hit(t_max, NONE);\n    let inv = reciprocal(d);\n    var
    waiting_node: array<u32, STACK>;\n    var waiting_entry: array<f32, STACK>;\n    var
    top = 0u;\n    // The node to look in next; NONE once there is none.\n    var node =
    select(0u, NONE, enter(0u, o, inv, t_max) > t_max);\n    while (node != NONE)
    {\n        let low = nodes[2u * node];\n        let count = nodes[2u * node +
    1u].w;\n        var next = NONE;\n        if (count > 0u) {\n            let end =
    low.w + count;\n            for (var place = low.w; place < end && !(any_hit &&
    best.face != NONE); place++) {\n                let face = triangles[3u *
    place].w;\n                if (face != skip) {\n                    let t =
    hit_triangle(place, o, d);\n                    if (t > t_min && t < best.t)
    {\n                        best = Hit(t, face);\n
    }\n                }\n            }\n            if (any_hit && best.face != NONE)
    {\n                top = 0u;\n            }\n        } else {\n            let left
    = low.w;\n            let right = left + 1u;\n            let to_left = enter(left,
```

```
o, inv, best.t);\n            let to_right = enter(right, o, inv,
best.t);\n            let in_left = to_left <= best.t;\n            let in_right =
to_right <= best.t;\n            if (in_left && in_right) {\n                let
left_first = to_left <= to_right;\n                waiting_node[top] = select(left,
right, left_first);\n                waiting_entry[top] = select(to_left, to_right,
left_first);\n                top++;\n                next = select(right, left,
left_first);\n            } else if (in_left || in_right) {\n                next =
select(right, left, in_left);\n            }\n        }\n        // Else the nearest
waiting node the ray still reaches before its best hit, if any.\n        while (next
== NONE && top > 0u) {\n            top--;\n            if (waiting_entry[top] <=
best.t) {\n                next = waiting_node[top];\n            }\n
}\n        node = next;\n    }\n    return best;\n}\n"
```

The tree and the search in WGSL, for any shader that binds `nodes` (two vec4<u32> per node: its box's low corner and first child or triangle, then its high corner and triangle count, zero for an inner node) and `triangles` (three vec4<u32> per place in leaf order: the three corners, the first with its face) as read-only storage.

class acoustic_spaces.engine.gpu_rays.**GpuBvh**(bvh)

The tree of `bvh` (an `acoustic_spaces._native.Bvh`) on the GPU, with the CPU's batch queries.

scale

Length of the model's box diagonal (m).

centre

Where the model's box is centred: the GPU's coordinates are relative to it.

property most_rays: int

The most rays one `RayBatch` holds on this GPU.

upload(origins, directions, t_max, ignore) → **RayBatch**

Puts rays on the GPU: `origins` and unit `directions` (n, 3), how far to search (n,; inf for no limit) and a face each one skips (n,; -1 for none).

search(batch: **RayBatch**, any_hit: bool = False) → tuple[ndarray, ndarray]

Searches the rays of `batch`: per ray the distance and the face of the nearest hit, or with `any_hit` of a hit, (inf, -1) for none.

intersect(origins, directions, t_max, ignore) → tuple[ndarray, ndarray]

As `Bvh.intersect`: per ray, the nearest hit's distance and face, (inf, -1) for a miss.

occluded(origins, directions, t_max, ignore) → ndarray

As `Bvh.occluded`: per ray, whether a face lies within its `t_max`.

class acoustic_spaces.engine.gpu_rays.**RayBatch**(rays, hits, count: int, t_max: ndarray)

Rays uploaded to the GPU (see `GpuBvh.upload()`), to search more than once.

acoustic_spaces.engine.gpu_rays.**pack_tree**(layout: dict[str, ndarray], scale: float)

The tree of a `Bvh.layout()` as the shader reads it: the centre of the model's box (the shader's coordinates are relative to it), the nodes (nodes, 2, 4) and the triangles (places, 3, 4), as 32-bit words (see `BVH_WGSL`).

Parameters, air and impulse responses

Room acoustic parameters of ISO 3382-1, from energy decays or impulse responses.

All parameters are per octave band and measured from the arrival of the direct sound:

- `edt`, `t20`, `t30`: decay times (s) from a line fitted to the backward-integrated (Schroeder) curve between 0 and -10 dB, -5 and -25 dB, and -5 and -35 dB.
- `c50`, `c80`: clarity (dB), energy before 50 or 80 ms over the energy after.
- `d50`: definition, energy before 50 ms over all of it (0 to 1).
- `ts`: centre time (s), the energy-weighted mean arrival time.
- `g`: strength (dB), total energy over that of the same source at 10 m in free field.

Where the response has them (see `MORE_PARAMETERS`), also:

- `lf`, `lfc`: early lateral energy fractions (annex A), the energy a figure of eight with its null on the source picks up from 5 to 80 ms over all the energy up to 80 ms; `lfc` weights each arrival by the cosine of its angle to the figure's axis, not the square.
- `lj`: late lateral sound level (dB), the figure of eight's energy from 80 ms on over the free-field energy at 10 m, as for G.
- `iacc_e`, `iacc_l`: interaural cross-correlation coefficients (annex B) of a binaural response, 0 to 80 ms and from 80 ms to the end.
- `st_early`, `st_late`: stage support (dB), measured 1 m from the source: the energy from 20 to 100 ms and from 100 to 1000 ms over that of the first 10 ms. `st_early` is Gade's ST1.
- `t30_x`, `t30_y`, `t30_z`: T30 of figure-of-eight pickups facing forward, left and up from the listener, the directional decay of a first-order Ambisonics response.

`speech_transmission_index()` gives the STI of IEC 60268-16 from the same energies, or STIPA, `useful_to_detrimental()` U50, and `echo_criterion()` the echo criteria of Dietsch and Kraak from an impulse response. `time_weighted()` gives a level over time through IEC 61672-1's exponential time weighting, and `event_levels()` a sound event's maximum time-weighted level and its sound exposure level over its 10 dB-down time.

A response that stops before it has decayed loses the energy of its missing tail and reads too short; the tail is estimated from the last part of the decay and added back (ISO 3382-1, 5.3.4). One that stops before it has decayed by 20 dB leaves too much to that estimate, and all its parameters are NaN.

```
acoustic_spaces.parameters.ALL_PARAMETERS = {'c50': ('dB', 'clarity, 50 ms'), 'c80':
    ('dB', 'clarity, 80 ms'), 'curvature': ('%', 'curvature, 100 (T30 / T20 - 1) (ISO
    3382-2)'), 'd50': ('', 'definition'), 'edt': ('s', 'early decay time'), 'g': ('dB',
    'strength'), 'iacc_e': ('', 'interaural cross-correlation, 0 to 80 ms'), 'iacc_l':
    ('', 'interaural cross-correlation, 80 ms on'), 'lf': ('', 'early lateral energy
    fraction'), 'lfc': ('', 'early lateral energy fraction, cosine weighted'), 'lj':
    ('dB', 'late lateral sound level'), 'st_early': ('dB', 'early stage support'),
    'st_late': ('dB', 'late stage support'), 't20': ('s', 'reverberation time from the
    -5 to -25 dB decay'), 't30': ('s', 'reverberation time from the -5 to -35 dB
    decay'), 't30_x': ('s', 'T30 of a figure of eight facing forward'), 't30_y': ('s',
    'T30 of a figure of eight facing left'), 't30_z': ('s', 'T30 of a figure of eight
    facing up'), 'ts': ('s', 'centre time'), 'xi_t20': ('‰', 'non-linearity of the T20
    decay, 1000 (1 - r²) of its line (ISO 3382-2)'), 'xi_t30': ('‰', 'non-linearity of
    the T30 decay, 1000 (1 - r²) of its line (ISO 3382-2)')}
```

Every parameter, in display order.

```
acoustic_spaces.parameters.A_WEIGHTING_DB: tuple[float, ...] = (-26.2, -16.1, -8.6, -3.2,
    0.0, 1.2, 1.0, -1.1)
```

A-weighting of the octave bands (dB, IEC 61672-1, rounded).

```
acoustic_spaces.parameters.C_WEIGHTING_DB: tuple[float, ...] = (-0.8, -0.2, 0.0, 0.0,
    0.0, -0.2, -0.8, -3.0)
```

C-weighting of the octave bands (dB, IEC 61672-1, rounded).

```
acoustic_spaces.parameters.DECAY_RANGES = {'edt': (0.0, -10.0), 't20': (-5.0, -25.0),
    't30': (-5.0, -35.0)}
```

Fit ranges in dB for EDT, T20 and T30.

```
acoustic_spaces.parameters.ECHO_CRITERIA: dict[str, tuple[float, float, tuple[int, ...],
    tuple[float, float]]] = {'music': (1.0, 0.014, (1000, 2000), (1.5, 1.8)), 'speech':
    (0.6666666666666666, 0.009, (1000,), (0.9, 1.0))}
```

Echo criteria (Dietsch and Kraak, 1986): the power n of the pressure's magnitude, the window (s), the octave bands the response is filtered to with one band-pass over them (one octave about 1 kHz for speech, two about 1.4 kHz for music), and the EC at which 10 % and 50 % of listeners hear an echo.

```
acoustic_spaces.parameters.MORE_PARAMETERS = {'curvature': ('%', 'curvature, 100 (T30 /
    T20 - 1) (ISO 3382-2)'), 'iacc_e': ('', 'interaural cross-correlation, 0 to 80 ms'),
    'iacc_l': ('', 'interaural cross-correlation, 80 ms on'), 'lf': ('', 'early lateral
    energy fraction'), 'lfc': ('', 'early lateral energy fraction, cosine weighted'),
    'lj': ('dB', 'late lateral sound level'), 'st_early': ('dB', 'early stage support'),
    'st_late': ('dB', 'late stage support'), 't30_x': ('s', 'T30 of a figure of eight
    facing forward'), 't30_y': ('s', 'T30 of a figure of eight facing left'), 't30_z':
    ('s', 'T30 of a figure of eight facing up'), 'xi_t20': ('‰', 'non-linearity of the
    T20 decay, 1000 (1 - r²) of its line (ISO 3382-2)'), 'xi_t30': ('‰', 'non-linearity
    of the T30 decay, 1000 (1 - r²) of its line (ISO 3382-2)')}
```

Parameters only some responses give, in display order: those of a lateral or a binaural response, those 1 m from the source and those of first-order Ambisonics. They are NaN in every band of a response that cannot give them.

```
acoustic_spaces.parameters.PARAMETERS = {'c50': ('dB', 'clarity, 50 ms'), 'c80': ('dB',
    'clarity, 80 ms'), 'd50': ('', 'definition'), 'edt': ('s', 'early decay time'), 'g':
    ('dB', 'strength'), 't20': ('s', 'reverberation time from the -5 to -25 dB decay'),
    't30': ('s', 'reverberation time from the -5 to -35 dB decay'), 'ts': ('s', 'centre
    time')}
```

Parameter names, units and short descriptions, in display order.

```
acoustic_spaces.parameters.STIPA_MODULATION_HZ: tuple[tuple[float, float], ...] = ((1.6,
    8.0), (1.0, 5.0), (0.63, 3.15), (2.0, 10.0), (1.25, 6.25), (0.8, 4.0), (2.5, 12.5))
```

STIPA (IEC 60268-16:2020, annex B): the two modulation frequencies of each band (Hz).

```
acoustic_spaces.parameters.STI_BANDS_HZ: tuple[int, ...] = (125, 250, 500, 1000, 2000,
    4000, 8000)
```

Speech transmission index (IEC 60268-16:2020, annex A): its octave bands, the modulation frequencies (Hz), the weighting factors alpha and beta of male and female speech (the female ones from the 2011 edition, which leave out the 125 Hz band), and the absolute speech reception threshold of each band (dB).

```
acoustic_spaces.parameters.SLOW_S = 1.0
```

The time constant (s) of IEC 61672-1's slow time weighting, S.

```
acoustic_spaces.parameters.SPEECH_SPECTRUM_DB: dict[str, tuple[float, ...]] = {'female':
    (-18.7, -6.7, 5.3, -1.9, -9.1, -15.8, -16.7, -18.0), 'male': (-9.1, 2.9, 2.9, -0.8,
    -6.8, -12.8, -18.8, -24.8)}
```

The long-term spectrum of speech (IEC 60268-16:2011, table A.2): octave-band levels (dB) relative to the A-weighted level, `OCTAVE_BANDS_HZ`. The standard gives none at 63 Hz, nor at 125 Hz for female speech; those are taken 12 dB below the next band up (an assumption; neither counts in the STI).

```
acoustic_spaces.parameters.TIME_WEIGHTING_STEP_S = 0.01
```

How far apart (s) `time_weighted()` works the time-weighted level out by default.

```
class acoustic_spaces.parameters.EventLevels(max_db: float, exposure_db: float, start_s:
    float, end_s: float)
```

The levels of a sound event, from `event_levels()`.

`max_db`: `float`

The highest time-weighted level (dB): L_ASmax, A-weighted and slow.

`exposure_db`: `float`

The sound exposure level (dB) over the time the time-weighted level is within the chosen drop of its highest: SEL over the 10 dB-down time.

`start_s`: `float`

When that time starts and ends (s, as the times given).

`end_s`: `float`

Alias for field number 3

class acoustic_spaces.parameters.**Levels**(spl: ndarray, spl_a: float, sti: float, stipa: float, u50: ndarray, re_free_field: ndarray, re_free_field_a: float)

What a listener gets from a source of a given sound power (`levels_and_speech()`): the sound pressure level per octave band (`spl`, dB SPL) and A-weighted (`spl_a`, dB(A)); the speech transmission index, `sti` and `stipa` (0 to 1), and `u50` per octave band (dB); outdoors the level relative to free field per octave band (`re_free_field`, dB) and A-weighted (`re_free_field_a`), NaN in a room. NaN where undefined.

`as_dict`() → dict

Plain numbers and lists under the report's names (`spl_db`, `spl_a_db`, `sti`, `stipa`, `u50`, `re_free_field_db`, `re_free_field_a_db`), with None for undefined values (JSON friendly).

class acoustic_spaces.parameters.**RoomParameters**(edt: ndarray, t20: ndarray, t30: ndarray, c50: ndarray, c80: ndarray, d50: ndarray, ts: ndarray, g: ndarray, lf: ndarray | None = None, lfc: ndarray | None = None, lj: ndarray | None = None, iacc_e: ndarray | None = None, iacc_l: ndarray | None = None, st_early: ndarray | None = None, st_late: ndarray | None = None, t30_x: ndarray | None = None, t30_y: ndarray | None = None, t30_z: ndarray | None = None, xi_t20: ndarray | None = None, xi_t30: ndarray | None = None, curvature: ndarray | None = None)

ISO 3382-1 parameters, one value per octave band (NaN where undefined).

Those of `MORE_PARAMETERS` are NaN in every band unless given.

classmethod **from_dict**(data: dict) → **RoomParameters**

The parameters `as_dict()` gave (None for undefined values); those of `MORE_PARAMETERS` it lacks are NaN.

as_dict() → dict[str, list[float | None]]

Plain lists, with None for undefined values (JSON friendly).

given() → list[str]

The parameters with a value in at least one band, and those of `PARAMETERS` always, in display order.

table() → str

A text table: one row per parameter (see `given()`), one column per band.

class acoustic_spaces.parameters.**SpeechParameters**(sti: float, stipa: float, u50: ndarray, signal_db: ndarray)

Speech intelligibility at a listener: STI and STIPA (0 to 1), U50 per octave band (dB), and the speech level (dB SPL per band) they take.

as_dict() → dict

Plain numbers and lists, with None for undefined values (JSON friendly).

acoustic_spaces.parameters.**Talker**

Speech that talks: male or female.

alias of `Literal`['male', 'female']

acoustic_spaces.parameters.**decay_curve**(energy: ndarray, time_step: float) → ndarray

Backward-integrated (Schroeder) decay in dB at the start of each slot.

`energy` is one band's energy per slot, starting at the direct sound; the curve starts at 0 dB and includes the estimated energy after the end of the response.

acoustic_spaces.parameters.**echo_criterion**(ir: ndarray, sample_rate: float, kind: str = 'speech', *, filters: str = 'iec') → tuple[float, float]

The echo criterion EC of Dietsch and Kraak (1986) of a mono impulse response, and the time (s after the direct sound) where it peaks.

The response, filtered to the bands of `ECHO_CRITERIA[kind]` (1 kHz for speech, 1 and 2 kHz for music, with one band-pass over both: two octaves about 1.4 kHz, as Dietsch and Kraak have it), gives the build-up of its centre time ts(t), the mean arrival time up to t weighted by the pressure's magnitude to the power n; EC is the steepest rise of ts over the window, in units of the window. An echo makes ts jump: listeners hear one where EC exceeds 1.0 for speech or 1.8 for music (half of them; 0.9 and 1.5 for a tenth). NaN for a response shorter than the window, or with a band at or above half `sample_rate`.

```
acoustic_spaces.parameters.event_levels(times: Sequence[float] | ndarray, levels_db:
    Sequence[float] | ndarray, time_constant_s: float = 1.0, down_db: float = 10.0,
    step_s: float = 0.01) → EventLevels
```

The maximum level and the sound exposure level of a level `levels_db` (dB) heard at `times` (s), as ECAC Doc 29 measures an aircraft's (Volume 2, its glossary and section 4.1): the highest level through the time weighting of `time_constant_s` (slow by default, `time_weighted()`), and the sound exposure (the mean square integrated over time with no time weighting, IEC 61672-1), as a level held for 1 s, from the first to the last time the time-weighted level is within `down_db` of that highest: the "10 dB-down" time of eq. 4-1, which the NPD curves' SEL is taken over (footnote 15). Doc 29's eq. 4-2 integrates its L_A(t), which its glossary has slow-weighted; over a flyover's 10 dB-down time that differs from the sound exposure by under 0.01 dB. Where a pass starts or ends within `down_db` of its highest, that time is cut there: `start_s` or `end_s` is its first or last time.

```
acoustic_spaces.parameters.from_energy(energy: ndarray, time_step: float, onset_s: float,
    reference_energy: float | Sequence[float] | ndarray | None = None, *, stage: bool =
    False, lateral: tuple[ndarray, ndarray] | None = None, directional:
    Sequence[ndarray] | None = None, open_air: bool = False) → RoomParameters
```

Parameters from energy per band and time slot (bands, slots), slot 0 at emission.

`onset_s` is the arrival time of the direct sound. `reference_energy` is the energy of the same source at 10 m in free field, one value or one per band; without it `g` and `lj` are NaN. With `stage`, `st_early` and `st_late` (for a listener 1 m from the source). `lateral` is the energy a figure of eight with its null on the source picks up and its product with the pressure, each on the slots of `energy`, for `lf`, `lfc` and `lj`; `directional` the energies of figure-of-eight pickups facing forward, left and up, for `t30_x`, `t30_y` and `t30_z`. With `open_air` (an outdoor scene's response), G is that of the energy the response holds in a band that does not decay as a room's does, where the other parameters stay undefined: there is no reverberation to extrapolate past its end.

```
acoustic_spaces.parameters.interaural_cross_correlation(binaural: ndarray, sample_rate:
    float, *, filters: str = 'iec') → dict[str, ndarray]
```

IACC_E and IACC_L of a binaural impulse response (samples, 2), per octave band.

The interaural cross-correlation coefficient (ISO 3382-1, annex B) is the largest magnitude of the normalised cross-correlation of the two ears' responses within 1 ms either way; `iacc_e` takes them from the direct sound's arrival at the nearer ear to 80 ms later, `iacc_l` from there to the end. NaN in a band at or above half `sample_rate`.

```
acoustic_spaces.parameters.modulation_transfer(energy: ndarray, time_step: float,
    frequencies: Sequence[float] = (0.63, 0.8, 1.0, 1.25, 1.6, 2.0, 2.5, 3.15, 4.0, 5.0,
    6.3, 8.0, 10.0, 12.5)) → ndarray
```

Modulation transfer function (bands, modulation frequencies) of an energy response e(t) (bands, slots): the magnitude of the sum of e(t) exp(-2 pi j F t) over the sum of e(t), at each F of `frequencies` (Schroeder's method).

```
acoustic_spaces.parameters.speech_transmission_index(energy: ndarray, time_step: float,
    *, signal_db: Sequence[float] | ndarray | None = None, noise_db: Sequence[float] |
    ndarray | float | None = None, stipa: bool = False, talker: Literal['male',
    'female'] = 'male') → float
```

STI (0 to 1) of an energy response per octave band (bands, slots), from the modulation transfer function (IEC 60268-16:2020, indirect method), weighted for male speech or, with `talker="female"`, female speech. With `stipa`, STIPA instead: two modulation frequencies per band, not fourteen (`STIPA_MODULATION_HZ`).

`energy` has the bands of `OCTAVE_BANDS_HZ` (the 63 Hz band is not used) or of `STI_BANDS_HZ`. `signal_db` is the speech level per band at the listener (dB SPL); with it, auditory masking and the reception threshold apply, and `noise_db` (dB SPL per band, or one value for all) lowers the modulation by the signal-to-noise ratio. Without either, the STI is that of the room alone.

```
acoustic_spaces.parameters.sti_from_impulse_response(ir: ndarray, sample_rate: float, *,
    filters: str = 'iec', noise: bool = True, stipa: bool = False) → float
```

STI, or with `stipa` STIPA, of the room alone from a mono impulse response: from its octave bands (125 Hz to 8 kHz) from the onset on, with `noise` cut where they sink into the background noise and the noise taken off, as `from_impulse_response()` takes them. NaN when one of those bands lies at or above half `sample_rate` (at a sample rate below 11.25 kHz, the 8 kHz band does).

```
acoustic_spaces.parameters.time_weighted(times: Sequence[float] | ndarray, levels_db:
    Sequence[float] | ndarray, time_constant_s: float = 1.0, step_s: float = 0.01) →
    tuple[ndarray, ndarray, ndarray]
```

A level `levels_db` (dB) heard at `times` (s, increasing) through IEC 61672-1's exponential time weighting of time constant `time_constant_s` (the slow time weighting S by default), as a sound level meter reads it: the mean square at each moment t, 10^(L/10), weighted by exp(-(t - ξ)/τ)/τ over every earlier moment ξ. Between the given times the mean square goes linearly in time; before the first it is taken to have stood at its first value. Returns the times of a uniform grid `step_s` apart from the first time to the last, the level there, and the time-weighted level there (dB), worked out exactly for a mean square linear from one grid time to the next.

```
acoustic_spaces.parameters.useful_to_detrimental(d50: Sequence[float] | ndarray,
    signal_db: Sequence[float] | ndarray | float, noise_db: Sequence[float] | ndarray |
    float | None = None) → ndarray
```

U50 (dB) per band (Bradley, 1986): the energy of the first 50 ms over that of the rest and the background noise, from the definition `d50` and the speech and noise levels at the listener (dB SPL, one value or one per band). Without noise it is C50.

```
class acoustic_spaces.parameters.BandDecay(energy: ndarray, sample_rate: float, onset_s:
    float, crosspoint: ndarray, noise: ndarray, decay_range_db: ndarray)
```

Squared octave-band responses of one impulse response, from its onset on.

onset_s: float

Arrival of the direct sound in the impulse response (s).

crosspoint: ndarray

Per band: sample where the response sinks into the background noise (the length when no noise floor shows), the noise power per sample, and the decay range (dB).

cleaned(band: int) → ndarray

One band's energy up to its crosspoint, with the noise taken off.

acoustic_spaces.parameters.**FILTER_ORDER** = 3

Order of the Butterworth octave filters for impulse responses.

acoustic_spaces.parameters.**OCTAVE_RATIO** = 1.9952623149688795

Ratio between adjacent octave mid-band frequencies, base 10 (IEC 61260-1).

acoustic_spaces.parameters.**a_weighted**(levels_db: Sequence[float] | ndarray) → float

The A-weighted level (dB) of levels per octave band (`OCTAVE_BANDS_HZ`).

acoustic_spaces.parameters.**band_decay**(ir: ndarray, sample_rate: float, *, filters: str = 'iec', noise: bool = True, start: int | None = None) → BandDecay

Filter `ir` into octave bands and, with `noise`, find each band's noise floor.

`ir` is (samples,) or (samples, 1), as `read_wav()` gives a mono file. The bands start at its onset, or at sample `start`. A band whose lower edge (its exact mid-band frequency over 10^0.15, where its IEC 61260-1 filter is 3 dB down: 5623 Hz for 8 kHz) is at or above half the sample rate holds nothing of its own, and has no energy here; one that reaches past it holds its part below it.

acoustic_spaces.parameters.**c_weighted**(levels_db: Sequence[float] | ndarray) → float

The C-weighted level (dB) of levels per octave band (`OCTAVE_BANDS_HZ`).

acoustic_spaces.parameters.**band_energy_share**(sample_rate: float, filters: str = 'iec') → ndarray

Energy of a unit impulse inside each band, for converting broadband references.

acoustic_spaces.parameters.**band_filter_gains**(n_fft: int, sample_rate: float, bands: int = 8, order: int = 3) → ndarray

Magnitudes (bands, n_fft // 2 + 1) of Butterworth band-pass filters of the 8 octave bands (as `octave_filter_gains()`) or the 24 third-octave bands, whose edges are fm G^(+-1/6).

acoustic_spaces.parameters.**free_field_reference**(ir: ndarray, sample_rate: float, distance_m: float, *, filters: str = 'iec') → ndarray

Per-band reference energies for G from a free-field impulse response.

`ir` is the source measured at `distance_m` with no reflections (or with them windowed out); the result is its band energies moved to 10 m (inverse square law). The bands are taken with silence on both sides, as the zero-phase filters spread the sound both ways in time: a response that starts with its direct sound keeps all of each band's energy.

acoustic_spaces.parameters.**from_impulse_response**(ir: ndarray, sample_rate: float, reference_energy: float | Sequence[float] | None = None, *, filters: str = 'iec', noise: bool = True, stage: bool = False, lateral: ndarray | None = None, directional: ndarray | None = None) → RoomParameters

Parameters from a mono impulse response, split into octave bands.

`ir` is (samples,) or (samples, 1); an empty or silent one has every parameter NaN. `reference_energy` is the energy of the same source at 10 m in free field: per band (as from `free_field_reference()`), or that of a broadband impulse, which for responses made by this package is 0.01 (0.01 * ir_gain**2 for those written with `normalize`, `ir_gain` in their `parameters.json`). `filters` picks the band filters (see `octave_bands()`). With `noise`, each band is cut where it sinks into the background noise, the noise is subtracted and the decay beyond the cut estimated (ISO 3382-1, 5.3.4); decay times whose range does not clear the noise by 10 dB (EDT 20 dB, T20 35 dB, T30 45 dB) are NaN.

Every parameter is NaN in a band whose lower edge (as `band_decay()` gives it) is at or above half `sample_rate`, as the response holds nothing of its own there (at 8 kHz, the 8 kHz band). A band that reaches past it is analysed from its part below it (at 8 kHz the 4 kHz band, at 16 kHz the 8 kHz band); a broadband `reference_energy` is taken over the same part.

With `stage`, `st_early` and `st_late` as well, for a response measured 1 m from the source. `lateral` is the response of a figure of eight with its null on the source, measured with `ir` and at the same sensitivity on its axis, for `lf`, `lfc` and `lj`; `directional` (samples, 3) those of figure-of-eight pickups facing forward, left and up, as the X, Y and Z of first-order Ambisonics (SN3D), for `t30_x`, `t30_y` and `t30_z`. Both are taken from the onset of `ir`.

acoustic_spaces.parameters.noise_crosspoint(energy: ndarray, sample_rate: float) → tuple[int, float, float]

Where a squared response sinks into its background noise (Lundeby et al., 1995).

`energy` is one band's squared response from the direct sound on. Returns the crosspoint (samples), the mean noise power per sample and the decay range (dB, from the loudest 10 ms down to the noise). A response that ends in silence (its last tenth more than 120 dB below the loudest 10 ms), or is still decaying in its last tenth, has no crosspoint (`len(energy)`), no noise and an infinite range; one with no decay above its noise has a range of 0.

acoustic_spaces.parameters.octave_bands(signal: ndarray, sample_rate: float, filters: str = 'iec') → ndarray

Split a signal into octave bands: (n,) -> (bands, n).

`filters="iec"` applies the magnitudes of `octave_filter_gains()` with zero phase, so no band is delayed against another; `"crossover"` uses the crossovers of `acoustic_spaces.synthesis`, whose bands sum back to the input.

Zero-phase filters spread a sound both ways in time, and the bands cover the signal's samples only: what a band spreads before the first sample or after the last is not in it. Of an impulse at the first sample a band keeps about half its energy (0.50 at 63 Hz to 0.71 at 8 kHz with the IEC filters); with the impulse 0.2 s in, all of it. Pad the signal with silence for its energies, as `free_field_reference()` and `from_impulse_response()` do.

acoustic_spaces.parameters.octave_filter_gains(n_fft: int, sample_rate: float, order: int = 3) → ndarray

Magnitudes (bands, n_fft // 2 + 1) of Butterworth octave band-pass filters.

A band-pass of `order` (that of its low-pass prototype; 3 meets IEC 61260-1 class 1) has 1 at mid-band, -3 dB at the band edges fm G^(+-1/2) and about -20 dB at the neighbouring mid-bands.

```
acoustic_spaces.parameters.octave_midbands() → ndarray
```

Exact mid-band frequencies (Hz) of `OCTAVE_BANDS_HZ`, base 10 (IEC 61260-1).

```
acoustic_spaces.parameters.onset(ir: ndarray, threshold_db: float = -20.0) → int
```

Sample where the response first rises to `threshold_db` below its peak (ISO 3382-1).

Sound in air: speed of sound and atmospheric absorption (ISO 9613-1:1993).

```
acoustic_spaces.air.EXACT_OCTAVE_HZ = (63.09573444801933, 125.89254117941675,
    251.188643150958, 501.18723362727224, 1000.0, 1995.2623149688795,
    3981.0717055349724, 7943.282347242814)
```

Exact octave midband frequencies (base-10 series) for the nominal `OCTAVE_BANDS_HZ`.

```
acoustic_spaces.air.STATED_F_OVER_P = (0.0004, 10.0)
```

The ratios of frequency to pressure (Hz/Pa) for which ISO 9613-1:1993, clause 7, states the accuracy of equations 3 to 5, at pressures below 200 kPa: the same in 7.1 to 7.3, which together take in every humidity and temperature a scene can have (±10 % for a molar concentration of water vapour of 0.05 % to 5 %, ±20 % from 0.005 % to 0.05 % and above 5 %, ±50 % below 0.005 %).

```
acoustic_spaces.air.absorption_db_per_m(frequencies_hz: Sequence[float] | ndarray,
    temperature_c: float = 20.0, humidity_percent: float = 50.0, pressure_kpa: float =
    101.325) → ndarray
```

Pure-tone atmospheric attenuation coefficient (dB/m), ISO 9613-1 equations 3 to 5.

```
acoustic_spaces.air.density(temperature_c: float = 15.0, pressure_kpa: float = 101.325) →
    float
```

Density of dry air (kg/m³) at the given temperature and pressure, as an ideal gas.

```
acoustic_spaces.air.energy_attenuation_per_m(temperature_c: float = 20.0,
    humidity_percent: float = 50.0, pressure_kpa: float = 101.325, bands:
    Literal['octave', 'third_octave'] = 'octave') → ndarray
```

Energy attenuation `m` (1/m) per octave band, or per third-octave band with `bands="third_octave"`, at the exact mid-band frequencies: intensity falls as exp(-m * distance).

This is the `m` of the `4 m V` air term in Sabine's and Eyring's formulas.

```
acoustic_spaces.air.speed_of_sound(temperature_c: float) → float
```

Speed of sound in air (m/s) at the given temperature.

```
acoustic_spaces.air.stated_pressures_kpa(bands: Literal['octave', 'third_octave'] =
    'octave') → tuple[float, float]
```

The pressures (kPa) at which the frequency-to-pressure ratio of every band lies in the range ISO 9613-1:1993 states its accuracy for (`STATED_F_OVER_P`).

Impulse response synthesis from octave-band (or third-octave-band) energies.

The filter bank splits the spectrum into the eight octave bands with zero-phase, half-octave raised-cosine crossovers, or into the 24 third-octave bands with crossovers a sixth of an octave wide. The band gains sum to exactly one, so a signal split into bands and summed again comes back unchanged (perfect reconstruction). Each function takes the bands from the number of band signals or energies it is given (8 or 24).

Energy convention: the simulation gives time-integrated intensity for a source of unit energy, so the direct sound at r metres carries 1 / (4 pi r^2). Impulse responses are scaled by `ENERGY_TO_IR` so that this direct sound is an impulse of height 1 / r: 1 at 1 m, and 0.01 of energy at 10 m (the reference for strength G).

A path or time slot with energy `E` in every band becomes a broadband impulse or noise burst of energy `E`. Band energies are spectral levels, so the part of the output inside band b carries `E_b` times that band's share of the spectrum.

`acoustic_spaces.synthesis.`**`ENERGY_TO_IR`** `= 12.566370614359172`

Impulse response energy per unit of simulated energy (time-integrated intensity).

`acoustic_spaces.synthesis.`**`REFERENCE_ENERGY_10M`** `= 0.01`

Energy of the impulse response of the direct sound at 10 m, the reference for G.

`acoustic_spaces.synthesis.`**`CROSSOVER_WIDTH`** `= 0.5`

Width of each crossover between octave bands, in octaves: half a band.

`acoustic_spaces.synthesis.`**`crossover_frequencies`**`(bands: int = 8) → ndarray`

Edges between adjacent bands (Hz) of the 8 octave bands or 24 third-octave bands.

`acoustic_spaces.synthesis.`**`crossover_width`**`(bands: int = 8) → float`

Width of each crossover (octaves): half a band.

`acoustic_spaces.synthesis.`**`band_gains`**`(n_fft: int, sample_rate: float, bands: int = 8) → ndarray`

Zero-phase gains (bands, n_fft // 2 + 1) on the rFFT grid of the 8 octave bands or 24 third-octave bands; they sum to one.

`acoustic_spaces.synthesis.`**`fft_size`**`(n: int, sample_rate: float) → int`

FFT length for filtering n samples with zero padding on both sides.

`acoustic_spaces.synthesis.`**`combine_bands`**`(band_signals: ndarray, sample_rate: float, quadrature: ndarray | None = None) → ndarray`

Filter each band signal with its band and sum: (bands, n) -> (n,). With `quadrature` (bands, n), signals to add a quarter turn ahead in phase (their spectra times j).

`acoustic_spaces.synthesis.`**`equalized`**`(signal: ndarray, sample_rate: float, gains) → ndarray`

`signal` (n, …) along its first axis through a zero-phase filter whose gain in each band is `gains` (8 octave bands or 24 third-octave bands), going from one to the next across the bands' crossovers (`band_gains()`); the same gain in every band is that gain at every frequency.

`acoustic_spaces.synthesis.`**`matched_gains`**`(gains, sample_rate: float, rounds: int = 30) →`

ndarray

Gains for `equalized()` (8 octave bands or 24 third-octave bands) whose filter changes the energy of white noise in each band, as `split_bands()` splits it, by `gains` squared. Across a crossover `equalized()` goes from one band's gain to the next's, so where they differ a band takes some of its neighbour's; these are `gains` nudged until each band's energy is its own, as near as `rounds` rounds get it (to 0.001 dB): for gains stepping 12 dB from band to band, the lower ones some 5 dB further down. A band of no gain stays at none.

`acoustic_spaces.synthesis.`**`split_bands`**`(signal: ndarray, sample_rate: float, bands: int = 8) → ndarray`

Split a signal into the 8 octave bands or 24 third-octave bands: (n, …) -> (bands, n, …), along its first axis; the bands sum to the input. As the bands are zero-phase, what each spreads before the first sample or after the last is not in it (see `acoustic_spaces.parameters.octave_bands()`).

`acoustic_spaces.synthesis.`**`band_energy_share`**`(sample_rate: float, n_fft: int = 65536, bands: int = 8) → ndarray`

Energy of a unit impulse inside each band: its share of the spectrum.

`acoustic_spaces.synthesis.`**`place_impulses`**`(delays_s: ndarray, amplitudes: ndarray, length: int, sample_rate: float, taps: int = 32) → ndarray`

Band signals (bands, length) with a fractional-delay impulse per path.

`amplitudes` is (paths, bands). Delays between samples use a Hann-windowed sinc.

`acoustic_spaces.synthesis.`**`placing_bytes`**`(paths: int, taps: int = 32) → int`

Bytes `place_impulses()` holds beside its output while it places `paths` impulses: six arrays (paths, taps) of eight bytes an element, one of one byte, and three (paths,).

`acoustic_spaces.synthesis.`**`dirac_sequence`**`(length: int, sample_rate: float, volume_m3: float | None, speed_of_sound: float, rng: Generator, time_step: float) → ndarray`

Random-sign impulses whose density grows like reflections in a room: 4 pi c^3 t^2 / V.

Every time slot of `shape_noise()` (`time_step` long, a whole number of samples or not) gets at least one impulse, so no slot's energy is lost.

`acoustic_spaces.synthesis.`**`smooth_energy`**`(energy: ndarray, time_step: float, start_s: float, window_s: float = 0.01) → ndarray`

Spread each slot's energy after `start_s` over a `window_s` window centred on it.

The late energy of a ray tracer is a sum of discrete hits; a single strong ray late in the tail would become a click. Smoothing moves energy by at most half a window and keeps the total. A slot within half a window of `start_s` or of the end spreads over the part of its window from `start_s` to the end, so no energy moves ahead of `start_s` (into the early window of C50, when that is 50 ms after the direct sound) or past the end.

`acoustic_spaces.synthesis.`**`shape_noise`**`(energy: ndarray, time_step: float, noise: ndarray, sample_rate: float) → ndarray`

Band signals (bands, n): `noise` scaled so each slot carries `energy[band, slot]`.

Spatial outputs

Spatial outputs: binaural, Ambisonics and loudspeaker impulse responses.

Every format turns the arrival directions of a simulated response into channels:

- `Ambisonics`: AmbiX (ACN channel order, SN3D normalisation), orders 0 to 7;
- `Loudspeakers`: vector base amplitude panning onto a named layout (stereo, 5.0, 7.0, 5.0.4, 7.0.4, and 5.1 to 7.1.4 with a silent LFE channel) or any list of speakers;
- `Binaural`: the two ears, with HRTFs from a SOFA file or a spherical head model;
- `Mono`: the omnidirectional response;
- `Microphone`: a listener's microphone, a first-order pattern or a coincident pair (XY, Blumlein, MS) that its mono output picks up through (`for_listener()`).

Early reflections keep their exact direction. The late part comes from the arrival directions the stochastic ray tracer records (or from all around, for the statistical tail). `spatial_impulse_response()` (also `Response.spatial_impulse_response`) renders a response in a format.

class acoustic_spaces.spatial.**HRTF**(directions: 'np.ndarray', hrir: 'np.ndarray', sample_rate: 'float', name: 'str' = 'hrtf')

nearest(directions: ndarray) → ndarray

Index of the nearest measured direction for each of (dirs, 3).

onsets() → ndarray

(m, 2) onset of each response (samples, fractions included): the lag at which it best matches its minimum-phase version, which starts at once (see `_onsets()`).

interpolate(directions: ndarray, *, onsets: Literal[False] = False) → ndarray

interpolate(directions: ndarray, *, onsets: Literal[True]) → tuple[ndarray, ndarray]

Responses (dirs, 2, taps) for head directions (dirs, 3), between the measured ones; with `onsets`, also the onset (dirs, 2) each starts at (samples, fractions included).

The measured directions are triangulated (the faces of their convex hull, as for VBAP). Each direction mixes the responses at the corners of the triangle it falls in, aligned at their onsets, with its barycentric weights, and the mix is delayed by the same mix of the onsets, to a fraction of a sample: that mix is the onset given. A measured direction gives its own response, and the weights, and with them the responses and the interaural time difference, change smoothly with the direction. Where no triangle covers a direction (a set that leaves part of the sphere out), the nearest response is used, and its onset.

onsets_at(directions: ndarray) → ndarray

The onsets (dirs, 2) that `interpolate()` gives for head directions (dirs, 3) (samples, fractions included), without mixing the responses.

resampled(sample_rate: float) → HRTF

The same responses at another sample rate (band-limited, by FFT).

The responses are filters: their frequency response, and so their gain, stays the same at every rate.

class acoustic_spaces.spatial.**Ambisonics**(order: int = 1)

AmbiX (ACN / SN3D) encoding of the given order: (order + 1) ** 2 channels.

pressure_gains() → ndarray

(channels,) what each channel picks up of a sound pressure that is the same all over the room (all of it in W, none in the other channels).

gains(directions: ndarray) → ndarray

Encoding gains (dirs, channels) of plane waves from head directions (dirs, 3).

class acoustic_spaces.spatial.**Binaural**(hrtf: HRTF | None = None, channels: int = 2, label: str = 'binaural')

Left and right ear, through `hrtf` (default: `spherical_head()`).

pressure_gains() → ndarray

(channels,) what each channel picks up of a sound pressure that is the same all over the room (all of it in each ear).

at(sample_rate: float) → HRTF

The HRTF at `sample_rate`.

class acoustic_spaces.spatial.**Format**(*args, **kwargs)

class acoustic_spaces.spatial.**Loudspeakers**(speakers: tuple[Speaker, ...], name: str = 'speakers')

A loudspeaker layout: a name from `LAYOUTS` or a list of speakers.

pressure_gains() → ndarray

(channels,) what each channel picks up of a sound pressure that is the same all over the room: 1/√n in each of the n main speakers, at unit power as the panning plays one direction (on a layout without height speakers, as it plays sound from straight above or below), and none in the LFE.

channel_mask() → int

WAV channel mask of a named layout; 0 (no assignment) for others.

gains(directions: ndarray) → ndarray

VBAP gains (dirs, channels) of unit power for head directions (dirs, 3).

class acoustic_spaces.spatial.**Microphone**(label: str, names: tuple[str, ...], weights: tuple[float, ...], axes: tuple[tuple[float, float, float], ...], a: tuple[float, ...] | None = None, aims: tuple[tuple[float, float, float], ...] = (), sides: tuple[tuple[float, float, float], ...] = ())

Capsules in one place: channel `k` picks up `weights[k] + axes[k] . v` of sound arriving from `v` in head coordinates.

a: tuple[float, ...] | None = None

For a pattern that changes with frequency, the `a` of its capsules in each band (8 or 24; the channels share it), each channel's capsule aim (a unit vector) and what it picks up besides that capsule (an MS pair's figure-eight); `weights` and `axes` then hold the pattern at 1 kHz (`SHOWN_BAND`). None for one pattern at every frequency.

pressure_gains() → ndarray

(channels,) what each channel picks up of a sound pressure that is the same all over the room: the capsules' omnidirectional part (in the lowest band, for a pattern that changes with frequency), none of their gradient part.

gains(directions: ndarray) → ndarray

What each channel picks up (n, channels) of sound arriving from `directions` (n, 3), unit vectors in head coordinates.

parts() → tuple[Microphone, Microphone]

A microphone whose pattern changes with frequency as two that do not: what each channel picks up whatever `a` (`d . v` and its side's) and what `a` scales (`1 - d . v`), `d` its capsule's aim. It picks up the first plus the second through a filter of gain `a` in each band (`a_in()`).

a_in(bands: int) → ndarray

`a` in `bands` bands (8 octaves or 24 third octaves), as a source's pattern goes from one to the other.

band_gains(directions: ndarray, bands: int) → ndarray

What each channel picks up (n, channels, bands) in each of `bands` bands (8 octaves or 24 third octaves) of sound arriving from `directions` (n, 3).

spectral_gains(directions: ndarray, shapes: ndarray) → ndarray

What each channel picks up (n, channels, bins) of sound arriving from `directions` (n, 3) at the frequencies of the synthesis's band filters `shapes` (bands, bins) (`acoustic_spaces.synthesis.band_gains()`), which sum to one: each band's gain across its share of the frequencies.

capsules() → list[tuple[FirstOrder, ndarray]]

Each channel as a first-order pattern and the rotation that aims it: columns forward, left and up in head coordinates (an MS pair's left channel is a first-order pattern aimed between the front and the left).

class acoustic_spaces.spatial.**Mono**(channels: int = 1, label: str = 'mono')

The omnidirectional response, as `Response.impulse_response` gives it.

pressure_gains() → ndarray

(channels,) what each channel picks up of a sound pressure that is the same all over the room (all of it).

class acoustic_spaces.spatial.**Speaker**(name: 'str', azimuth: 'float', elevation: 'float' = 0.0, lfe: 'bool' = False)

acoustic_spaces.spatial.**fibonacci_sphere**(n: int) → ndarray

`n` near-uniform unit vectors (n, 3) on a Fibonacci lattice, each standing for an equal share of the sphere.

acoustic_spaces.spatial.**for_listener**(fmt, listener)

The format `listener`'s output in `fmt` is rendered in: its microphone when `fmt` is mono and it has a pattern, else `fmt`.

acoustic_spaces.spatial.**from_angles**(azimuth_deg, elevation_deg) → ndarray

Unit vectors (…, 3) from azimuth and elevation in degrees.

acoustic_spaces.spatial.**from_output**(output) → Format

The format an `outputs` entry of a scene file (`scene.Output`) describes.

acoustic_spaces.spatial.**read_sofa**(path: str | Path) → HRTF

HRTF from a SOFA file with head-related impulse responses for two ears.

acoustic_spaces.spatial.**real_spherical_harmonics**(order: int, directions: ndarray) → ndarray

ACN / SN3D spherical harmonics (dirs, (order + 1) ** 2) of unit vectors (dirs, 3).

acoustic_spaces.spatial.**spatial_impulse_response**(response: Response, fmt: Format, sample_rate: int = 48000, seed: int | None = None, *, threads: int = 1) → np.ndarray

Impulse responses (samples, channels) of `response` in `fmt`.

Levels match the mono response: an arrival from straight ahead has the same height in W, in a speaker it is panned to, and (below the head's influence) at both ears. `threads` render the channels (binaurally the late part's bands and the ears) side by side, each as it is rendered alone, so the result is the same with any number. They are at most the computer's cores (all of them for 0 or fewer, as `performance.threads`), and each past the first holds one channel's band signals and their spectra more (some 75 MB for 3.5 s at 48 kHz in 8 bands), so a format's channels take no more of them than the computer's memory lets have a channel in progress (`channel_threads()`).

acoustic_spaces.spatial.**spherical_head**(sample_rate: float = 48000, *, radius: float = 0.0875, ear_azimuth: float = 100.0, speed_of_sound: float = 343.0, directions: int = 2048) → HRTF

Brown and Duda's spherical head model on a grid of `directions` directions.

Each ear sees a one-pole, one-zero head-shadow filter that boosts high frequencies up to 6 dB towards the source and cuts them up to 20 dB on the far side, and the path length around the sphere (Woodworth's formula). Responses start 1 ms early, so the earliest ear still has room for the filter's ringing.

acoustic_spaces.spatial.**to_angles**(directions: ndarray) → tuple[ndarray, ndarray]

Azimuth in (-180, 180] and elevation in [-90, 90] degrees of vectors (…, 3).

`acoustic_spaces.spatial.to_head(directions: ndarray, rotation: ndarray) → ndarray`

Room directions (…, 3) in the head coordinates of a listener whose `rotation` has the forward, left and up axes as columns (see `Orientation.rotation`).

Rotating sound fields, and decoding them to the ears:

Rotations: of directions (3 x 3 matrices from angles or quaternions) and of Ambisonics sound fields (one matrix per order of the AmbiX channels).

Angles follow the scene's `Orientation`: yaw turns left (from +x towards +y), pitch looks up and roll tilts the head to the right, applied in the order roll, pitch, yaw. A rotation matrix has the turned forward, left and up axes as its columns.

`class acoustic_spaces.spatial.rotation.FieldRotation(order: int)`

Rotations of an Ambisonics (ACN / SN3D) sound field of `order`.

`matrix()` gives the (channels, channels) matrix that turns the channels of any sound field into those of the same field turned by a rotation: a plane wave from `d` comes from `R @ d` afterwards. Orders do not mix, so the matrix is block diagonal. The first-order block is the rotation itself, its axes in ACN order (y, z, x), and each higher one follows from the one below by Ivanic and Ruedenberg's recursion (J. Phys. Chem. 1996, with the 1998 correction), which is exact and takes well under a millisecond at seventh order.

`matrix(rotation: ndarray) → ndarray`

(channels, channels): the field turned by `rotation` (3, 3).

`acoustic_spaces.spatial.rotation.angles_matrix(yaw: float = 0.0, pitch: float = 0.0, roll: float = 0.0) → ndarray`

The rotation (3, 3) of yaw, pitch and roll in degrees.

`acoustic_spaces.spatial.rotation.matrix_angles(rotation: ndarray) → tuple[float, float, float]`

Yaw, pitch and roll in degrees of a rotation (3, 3); the inverse of `angles_matrix()` (yaw and roll are 0 to 180 either way, pitch up to 90).

`acoustic_spaces.spatial.rotation.quaternion_matrix(w: float, x: float, y: float, z: float) → ndarray`

The rotation (3, 3) of a quaternion, normalised first (identity if it is zero).

Binaural decoding: filters that turn the AmbiX channels of a sound field into the signals at the two ears (`binaural_decoder()`), or the channels of a loudspeaker layout, each speaker heard as a virtual one (`binaural_filters()` gives the filters of either format).

The Ambisonics filters fit the HRTF of every measured direction with the spherical harmonics up to the order (least squares, `"ls"`). Low orders cannot follow the fast changes of the HRTF's phase with direction at high frequencies, and least squares then loses level and spaciousness; above a cutoff (the order times 600 Hz, after Schörkhuber, Zaunschirm and Höldrich 2018) the magnitude least squares decoder (`"magls"`, the default) fits only the magnitudes and keeps the phase that varies smoothly from the bin below.

```
acoustic_spaces.spatial.decoder.DECODERS = ('magls', 'ls')
```

Decoding methods.

```
acoustic_spaces.spatial.decoder.MAGLS_HZ_PER_ORDER = 600.0
```

Above this many hertz per order, `magls` fits magnitudes only.

```
acoustic_spaces.spatial.decoder.binaural_decoder(order: int, hrtf: HRTF, method: str =
    'magls') → ndarray
```

Filters (taps, (order + 1) ** 2, 2) from the AmbiX channels of `order` to the left and right ear, at the HRTF's sample rate. A plane wave encoded from a direction decodes to about that direction's HRTF.

```
acoustic_spaces.spatial.decoder.binaural_filters(fmt: Format, hrtf: HRTF, method: str =
    'magls') → np.ndarray
```

Filters (taps, channels, 2) from the channels of `fmt` to the left and right ear, at the HRTF's sample rate, so that a signal in `fmt` is heard as it plays: for `Ambisonics`, `binaural_decoder()` of its order; for `Loudspeakers`, each speaker as a virtual one, the HRTF of its direction (`HRTF.interpolate()`), and the LFE channel, which the renders leave silent, as nothing.

Rendering and playback

Rendering: impulse responses applied to dry signals (auralisation).

- `convolve()`: offline convolution of a recording with an impulse response (FFT overlap-add in float64), for any number of response channels.
- `resample()`: band-limited sample-rate conversion, so a 44.1 kHz recording can go through a 48 kHz response.
- `crossfaded()`: a crossfade from one sound to another whose power runs on however alike the two are, broadband or band by band: a tour's cuts from one seat to the next, a walk's from one place to the next.
- `auralize()`: the sources of a simulation each playing their own signal, heard at one listener in a spatial format (mono, binaural, Ambisonics, loudspeakers), summed. Outdoors in turbulence the early paths change as the eddies drift past: a new moment of them every `MOMENTS_PER_CORRELATION`-th of their correlation time (at most every `MOMENT_MIN_S`), each path's delay going from one moment's to the next's, so the sound flickers as it does in gusty air.
- `Convolver`: the compiled real-time convolver, block by block. It divides the response into partitions that grow along it, from the block size up to 8192 frames, and multiplies in the frequency domain, so the only latency is the block itself; nothing allocates while it runs.
- `play()`: a signal, or live input, through a response on the sound card (needs the `measure` extra: sounddevice); or through any block processor with one input, such as `HeadTrackedBinaural`.

class acoustic_spaces.render.**Convolver**(ir: ndarray, block: int = 256)

Real-time convolution through `ir`, one block of `block` frames at a time.

`ir` is (taps,), (taps, outputs) for one input, or (taps, inputs, outputs), where each output sums its inputs, each through its own response. `block` is a power of two, and the only latency. The response is split into segments whose partitions grow along it (see `segments`), and the work of the long partitions is spread over the blocks in between, so the cost per block stays even and grows slowly with the response's length: a 3 s, 16-channel response in blocks of 128 frames takes about 0.3 ms of the 2.7 ms each block lasts at 48 kHz, on one current x86-64 core.

property segments: list[tuple[int, int, int]]

(first tap, partition length, partitions) of each segment of the response.

process(block: ndarray, out: ndarray | None = None) → ndarray

One block, (block,) or (block, inputs), to (block, outputs) float32.

Given `out`, a writable C-contiguous float32 (block, outputs) array, the block is written into it and nothing new is allocated for it, as a sound card's callback needs (a float32 C-contiguous `block` is not copied either).

run(signal: ndarray) → ndarray

A whole signal, (samples,) or (samples, inputs), through the convolver: (samples + taps - 1, outputs), as `convolve()` gives it.

reset() → None

Forget all past input.

class acoustic_spaces.render.**Playback**(frames: int, xruns: int, load: float)

How a `play()` run went.

xruns: int

Blocks the sound card reported an underflow or overflow for.

load: float

Longest time spent convolving one block, as a share of the block's duration.

acoustic_spaces.render.**auralize**(result: SimulationResult, signals: Mapping[str, np.ndarray | tuple[np.ndarray, float]], listener: str | None = None, fmt: Format | None = None, *, sample_rate: int | None = None, seed: int | None = None, equal_power: bool = False) → np.ndarray

Every source in `signals` playing its signal, heard at `listener`.

`signals` maps source names to a signal at `sample_rate`, or to a (signal, rate) pair, resampled when the rates differ; multichannel signals are mixed down, as sources are points. `fmt` is a format of `acoustic_spaces.spatial` (default mono, through the listener's microphone when it has a pattern: `for_listener()`); `listener` may be left out when the result has one listener. Returns (samples, channels) at `sample_rate` (default: the simulation's), with the sources at the levels the simulation gives them: the responses carry each source's feed (gain, equalizer, delay) but are per unit of sound power, so each signal is scaled by its source's sound power (`level_db`) against the loudest source's in `signals` (`power_gains()`). `equal_power` leaves that out, as if every source had the loudest one's power.

acoustic_spaces.render.**convolve**(signal: ndarray, ir: ndarray) → ndarray

Full linear convolution: len(signal) + len(ir) - 1 samples.

`signal` and `ir` are (samples,) or (samples, channels). A mono signal goes through every channel of the response (a dry source through a binaural or Ambisonics response); a multichannel signal goes through a mono response channel by channel. The result is 1-D only when both inputs are.

acoustic_spaces.render.**crossfaded**(x0: ndarray, x1: ndarray, ramp: ndarray, sample_rate: float | None = None) → ndarray

`x0` fading out and `x1` fading in, (n,) or (n, channels): `x1`'s gain is `ramp` (n, rising from 0 to 1) and `x0`'s sqrt(1 - ramp²), equal power for two unlike sounds, both scaled, sample by sample, by how alike the two are over the crossfade (their correlation coefficient, one for all channels), so that the power runs from one's to the other's as it would for unlike ones. Alike sounds would otherwise be up to 3 dB louder in the middle, sounds in antiphase silent. The scaling is at most +6 dB: where the two cancel to less than half the amplitude unlike sounds would have (near antiphase, in the middle of the crossfade), their sum stays below it. With `sample_rate`, each octave band (`split_bands()`) by its own correlation.

acoustic_spaces.render.**play**(ir: np.ndarray | Convolver | HeadTrackedBinaural, sample_rate: int, signal: np.ndarray | None = None, *, block: int = 256, gain: float = 1.0, device: object = None, seconds: float | None = None, stop: threading.Event | None = None) → Playback

Play `signal` through `ir` (samples, outputs) on the sound card, convolving in real time; without a signal, the sound card's first input goes through it live.

`ir` may also be a processor that works block by block, as a `Convolver` or `HeadTrackedBinaural` does; its own block size then replaces `block`. Playback ends with the signal's reverberation tail; live input runs for `seconds`, until `stop` is set, or until Ctrl-C. `signal` must be at `sample_rate`.

`acoustic_spaces.render.`**`power_gains`**`(sources, names) → dict[str, float]`

Amplitude gain of each source in `names` for its sound power: 1 for the loudest of them, 10^(-ΔL/20) for one ΔL dB below it. `sources` are scene sources, or a mapping of name to sound power level (dB re 1 pW); names it lacks get no gain.

`acoustic_spaces.render.`**`resample`**`(signal: ndarray, rate_in: float, rate_out: float) → ndarray`

`signal` (samples,) or (samples, channels) at `rate_out`.

Polyphase windowed-sinc interpolation (Kaiser, 32 zero crossings each side), with the cutoff just below the lower Nyquist frequency so nothing aliases.

Walkthroughs

Walkthroughs: a listener or a source walking through the room, heard as it goes.

A listener or a point source with a `walk` in the scene goes from its position through the points of `walk.to`, at `walk.speed_m_s`, facing the way it goes unless `walk.face_forward` is false (it then keeps its orientation):

```
listeners:
  - name: visitor
    position: [2.0, 3.0, 1.6]
    walk: {to: [[12.0, 3.0, 1.6], [12.0, 8.0, 1.6]], speed_m_s: 1.2}
```

`walk_points()` places the walker every `step_m` along its path, and `simulate_walk()` simulates a response at each place. The early sound changes quickly from place to place, the late field slowly, so the late part, the costliest to trace, is traced only every `late_step_m`; the places in between mix the late parts of those on either side, in proportion to how near they are. `render_walk()` then plays signals through them: the response of the place the walker is nearest to, with a crossfade of `crossfade_s` half-way between two places, so that the sound changes smoothly as it walks. In each octave band the crossfade's power goes from one place's to the next's along a raised cosine, however alike the two sound (`crossfaded()`): a tone heard in another phase at the next place keeps its level, except near antiphase, where the two cancel in the middle of the crossfade by more than its gain, at most +6 dB, makes up. Each place's late part is drawn from the same random sequence, so the crossfades join tails that only differ in their decay. The walker stands at the end of its path once it is there, for as long as the signals last.

At walking speed there is no Doppler shift (it is under half a percent, 8 cents). A walk with `doppler: true` (a car, a train, an aircraft) plays each early path through a delay that changes as the walker goes (`acoustic_spaces.doppler`), so the pitch of each shifts as it does on its way; the late part is crossfaded by weights that add up to one (`crossfade_weights()`), which keep the power of tails this alike, a walking source's as its sound is sent and a walking listener's as it is heard; and `pass_by()` gives the level over time, its maximum and the sound exposure level, also as a sound level meter's slow time weighting reads them. Outdoors in turbulence the early sound flickers along the walk as in `auralize()`, through the same eddies from place to place.

A walk may instead give `keyframes`: where the walker is and how it faces at given times after its start, between which it moves and turns evenly, so that it can wait, turn on the spot or go at any pace (`knots()` gives either kind of walk as the knots it moves between). Several listeners and sources may move at once, each on its own walk, all on one clock: `simulate_movement()` simulates them together, every mover as a copy at each place it takes, in one simulation; `render_movement()` plays the signals through it as heard at every listener in use, still or moving; and `save_movement()` writes one WAV per listener. A walk with Doppler moves alone, there as in a walk of its own: every listener hears each early path through its changing delay. `walk.enabled: false` keeps a walk but leaves its walker standing still, and `with_moving()` picks which walks are on for one run, so that one scene serves both still and moving.

Each copy is named for its walker, its place and its time on the walk (`place_name()`, `car #12 @ 3.40 s`), and the simulation's progress messages name it so; a demo's video names the moment of each walk it shows (`moment_name()`, `car @ 3.40 s`): `place_of()` reads the walker and the time back from both. A head turned as a demo scripts it is named with its yaw (`facing_name()`, `standing facing 90.0 deg`), which `facing_of()` reads back.

`acoustic_spaces.walkthrough.`**`CROSSFADE_S`** `= 0.05`

Length (s) of the crossfade from one place's response to the next.

`acoustic_spaces.walkthrough.`**`LATE_STEP_M`** `= 2.0`

Distance (m) between the places a walk's late part is traced at.

`acoustic_spaces.walkthrough.`**`STEP_DEG`** `= 20.0`

Angle (degrees) a walker turns between the places of a walk along keyframes where it turns without moving on (on the spot, or nearly).

`acoustic_spaces.walkthrough.`**`STEP_M`** `= 0.25`

Distance (m) between the places a walk is simulated at.

```
class acoustic_spaces.walkthrough.Knots(times: ndarray, positions: ndarray, angles:
    ndarray)
```

The knots of a walk: where the walker is and how it faces at each, moving and turning evenly from one to the next (`knots()`).

times: ndarray

When it is at each knot (s from the start; the first is 0).

positions: ndarray

Where each knot is (K, 3).

angles: ndarray

How it faces at each knot (K, 3: yaw, pitch and roll in degrees), unwrapped so that it turns the shorter way round from one to the next.

property length_m: float

How far the walk goes.

property top_speed: float

The fastest it goes (m/s), over a leg.

at(times) → tuple[ndarray, ndarray]

Where it is and how it faces (yaw, pitch and roll in (-180, 180]) at `times` (s, a number or an array): at its start before, at its end after.

class acoustic_spaces.walkthrough.**Movement**(room: Room, times: ndarray, result: SimulationResult, copies: dict[tuple[str, str], tuple[str, ...]], late_places: int = 0, doppler: bool = False)

Responses with every mover at each moment of its walk, from `simulate_movement()`.

`room`: `Room`

The scene's room, as it was given (the movers with their walks).

`times`: `ndarray`

The moments the responses are at (s from the start, the first 0).

`result`: `SimulationResult`

The simulation: the still sources and listeners under their names, each mover as a copy at each place it takes (`copies`).

`copies`: `dict[tuple[str, str], tuple[str, ...]]`

Each mover (("listeners" or "sources", name)) and the name of its copy at each moment of `times`.

`late_places`: `int = 0`

How many of the movers' places have a late part of their own; the others mix theirs.

`doppler`: `bool = False`

Whether the mover, then alone, walks with Doppler (`walk.doppler`): each pair it is part of plays its early paths through delays that change as it goes.

`property places`: `int`

How many places the movers were simulated at in all.

`key``(source: str, listener: str, k: int) → tuple[str, str]`

The response of `source` at `listener` at moment `k`, in `result`.

`moves``(source: str, listener: str) → bool`

Whether either of the pair moves.

`class` `acoustic_spaces.walkthrough.`**`PassBy`**`(source: str, listener: str, times: ndarray, levels: ndarray, levels_a: ndarray)`

The level at a listener along a walk, from `pass_by()`: as a source passes it, or as it passes a source.

`times`: `ndarray`

(N,) when the sound of each place is heard (s from the start): for a walking source, when it is there plus its direct sound's delay; for a walking listener, when it is there.

`levels`: `ndarray`

(N, 8) the sound pressure level per octave band (dB) at each place, of the source's sound power, as it would be with the source standing there (-inf in a band no sound reaches).

`levels_a`: `ndarray`

(N,) the same, A-weighted (dB(A)).

property `duration_s`: `float`

How long the walk is heard for (s), from the first place's sound to the last's.

property `max_a`: `float`

The highest A-weighted level along the walk (dB(A)), L_Amax.

property `max_time`: `float`

When the highest A-weighted level is heard (s from the start).

property `exposure`: `ndarray`

(8,) the sound exposure level per octave band (dB): the level that, held for one second, carries the energy of the whole walk.

property `exposure_a`: `float`

The A-weighted sound exposure level (dB(A)), SEL or L_AE.

property `equivalent_a`: `float`

The A-weighted equivalent level over the walk (dB(A)), L_Aeq,T with T its `duration_s`.

property `slow_a`: `EventLevels`

The pass as a sound level meter reads an aircraft's for ECAC Doc 29 (`event_levels()`): the A-weighted level through IEC 61672-1's slow time weighting (1 s), its mean square going linearly in time from one place's `times` to the next; its highest, L_ASmax (`max_db`), and the sound exposure level over the time it is within 10 dB of that, the "10 dB-down" time (`exposure_db`, from `start_s` to `end_s`).

class acoustic_spaces.walkthrough.**WalkPoints**(times: ndarray, positions: ndarray, yaws: ndarray | None, angles: ndarray | None = None)

The places a walker is simulated at.

`times``: ndarray`

When it is at each place (s from the start).

`positions``: ndarray`

Where each place is (N, 3).

`yaws``: ndarray | None`

The way it faces at each place (yaw in degrees), or None when it keeps its own.

`angles``: ndarray | None = None`

Along keyframes, how it faces at each place (N, 3: yaw, pitch and roll in degrees).

`class acoustic_spaces.walkthrough.`**`WalkResponses`**`(name: str, kind: str, points: WalkPoints, names: tuple[str, ...], result: SimulationResult, late_places: tuple[int, ...] = (), doppler: bool = False)`

Responses along a walk, from `simulate_walk()`.

`name``: str`

The walker's name.

`kind``: str`

`"listeners"` or `"sources"`: what walks.

`points``: WalkPoints`

The places it was simulated at.

`names``: tuple[str, ...]`

The names the walker has at its places in `result`, in order (`"<name> #<place> @ <time> s"`).

`result``: SimulationResult`

The simulation, with the walker at every place.

`late_places``: tuple[int, ...] = ()`

The places that have their own late part (indices); the others mix theirs.

`doppler``: bool = False`

Whether the early paths go through delays that change as it goes (`walk.doppler`).

`acoustic_spaces.walkthrough.`**`crossfade_weights`**`(times, t, crossfade_s: float = 0.05) → ndarray`

Each place's weight (N, len(t)) at times `t`: 1 while the walker is nearest to it, falling to 0 over the crossfade to the next (a raised cosine centred half-way between them). They add up to 1. They are the gains of the late part of a walk with Doppler, and each place's share of the power in each band of a walk without.

`acoustic_spaces.walkthrough.`**`facing_name`**`(name: str, yaw_deg: float) → str`

`name` turned to the yaw `yaw_deg` (degrees, as a scene's orientation gives it), as a demo says what it shows while a head turns: `standing facing 90.0 deg`.

`acoustic_spaces.walkthrough.facing_of(label: str) → tuple[str, float] | None`

What a name that says which way it faces (`facing_name()`) names, and the yaw (degrees); None for any other name.

`acoustic_spaces.walkthrough.knots(item) → Knots`

The knots of `item`'s walk (a scene listener or source): along `to`, its corners at `speed_m_s` and, facing forward, where each turn through a corner starts and ends; along `keyframes`, its position at 0 s and each keyframe, facing as it says or as before.

`acoustic_spaces.walkthrough.late_mixing(distances: ndarray, late_step_m: float) → dict[int, dict[int, float]]`

The places of a walk (`distances` along it, m, rising) that take their late part from others, each with the places it mixes and their weights: places are traced every `late_step_m` or a little less, from the first to the last.

`acoustic_spaces.walkthrough.moment_name(name: str, time_s: float) → str`

The walker `name` `time_s` (s) into its walk, as a demo's video says what it shows: `car @ 3.40 s`.

`acoustic_spaces.walkthrough.movers(scene) → list[tuple[str, object]]`

The listeners and sources of `scene` that move: in use, with a walk that is enabled, as ("listeners" or "sources", item), listeners first.

`acoustic_spaces.walkthrough.pass_by(walk: WalkResponses, source: str | None = None) → PassBy`

The level along `walk` at its listener (a walking source) or of `source` (a walking listener; by default the only source): each place's level, of the source's sound power, at the time it is heard (quasi-static: as with the source standing there), with its maximum, the sound exposure level and the equivalent level. In a band where a place's G is undefined (no sound arrives in it, or its decay cannot be read), the place's level is that of the energy its response holds, as outdoors (`levels_or_held()`).

`acoustic_spaces.walkthrough.place_name(name: str, k: int, time_s: float) → str`

The name of the copy of the walker `name` at its `k`-th place, `time_s` (s) into its walk: `car #12 @ 3.40 s`. Numbered, as places less than 10 ms apart would share a time.

`acoustic_spaces.walkthrough.place_of(label: str) → tuple[str, float] | None`

The walker and the time (s) of a place's name (`place_name()`) or a moment's (`moment_name()`); None for any other name.

`acoustic_spaces.walkthrough.render_movement(movement: Movement, signals: Mapping[str, ndarray | tuple[ndarray, float] | Emission], fmt=None, *, crossfade_s: float = 0.05, sample_rate: int | None = None, seed: int | None = None, equal_power: bool = False) → dict[str, ndarray]`

`signals` (source names to a signal at `sample_rate`, or a (signal, rate) pair) played through the movement, as heard at every listener in use: each as (samples, channels) in format `fmt` (default mono; mono through each listener's microphone when it has a pattern) at `sample_rate` (default: the simulation's), all of one length and, as in `render_walk()`, each signal scaled by its source's sound power against the loudest one's unless `equal_power`. A pair that moves crossfades from each moment's response to the next's, as a walk does, or with Doppler (`Movement.doppler`) plays each early path through its changing delay and crossfades the late part, as `render_walk()` does; a still pair plays through its one response. A source that moves with Doppler, or that a listener passes with it, may send an `Emission` at `sample_rate`, the sound it sends each way, as `render_walk()` plays it.

```
acoustic_spaces.walkthrough.render_walk(walk: WalkResponses, signals: Mapping[str,
    ndarray | tuple[ndarray, float] | Emission], fmt=None, *, crossfade_s: float = 0.05,
    sample_rate: int | None = None, seed: int | None = None, equal_power: bool = False,
    followers: Sequence[tuple[float, ndarray | tuple[ndarray, float]]] = ()) → ndarray
```

`signals` (source names to a signal at `sample_rate`, or a (signal, rate) pair) played along the walk, as (samples, channels) in format `fmt` (default mono; mono through the listener's microphone when it has a pattern) at `sample_rate` (default: the simulation's). As in `auralize()`, each signal is scaled by its source's sound power against the loudest one's, unless `equal_power`. Along a walk with Doppler, the source that walks (or every source, for a walking listener) may send an `Emission` at `sample_rate`, the sound it sends each way: each early path carries what is sent the way it leaves the source, and the late part the emission's sound averaged over the directions.

`followers` are more sounds of a walking source with Doppler, each (lag_s, signal) going along its walk `lag_s` seconds after it, as the wheels and the cars of a train follow its front: each is heard as the walker's own signal would be, `lag_s` later, and scaled as it is. The result is long enough for the last follower's.

```
acoustic_spaces.walkthrough.save_movement(rendered: Mapping[str, ndarray], out: str |
    Path, rate: int, fmt=None, *, normalize: bool = False) → tuple[dict[str, Path],
    float]
```

Write what `render_movement()` gave: `out` itself for one listener, else `<out stem>_<listener>.wav` beside it for each (with `-2`, `-3` … where two listeners' names would give the same file); `normalize` brings the loudest peak of them all to -1 dBFS, one gain for all, so that the balance between the listeners holds. Gives each listener's file and the gain applied.

```
acoustic_spaces.walkthrough.simulate_movement(room: Room, *, step_m: float = 0.25,
    step_deg: float = 20.0, late_step_m: float | None = 2.0, settings: Simulation | None
    = None, threads: int | None = None, progress: Callable[[str], None] | None = None) →
    Movement
```

Simulate the scene with every mover (`movers()`) on one clock, in one simulation: the moments are every mover's places (every `step_m` along its walk, every `step_deg` of a turn on the spot, and its knots), and at each moment each mover stands as a copy where its walk has it then, a copy serving every moment it stands still at (once its walk is over, say). Every listener in use hears every source in use, still or moving. The late part is traced every `late_step_m` along each mover's walk, and barriers and buildings have the paths over their edges, as in `simulate_walk()`.

Every mover goes slower than sound. A walk with Doppler moves alone: with another mover it is refused. It goes without the wave solver, as in `simulate_walk()`.

```
acoustic_spaces.walkthrough.simulate_walk(room: Room, name: str | None = None, *,
    listener: str | None = None, step_m: float = 0.25, late_step_m: float | None = 2.0,
    settings: Simulation | None = None, threads: int | None = None, progress:
    Callable[[str], None] | None = None) → WalkResponses
```

Simulate the responses along a walk: `name` walks (by default the only listener or source with a walk), placed every `step_m`.

A walking listener hears every source in use; a walking source is heard at `listener`, even one left out (`enabled: false`), by default the only listener in use, with the other sources in use standing still. The walk must go slower than sound.

The late part is traced every `late_step_m` (see the module's description), or at every place with `late_step_m=None` or the statistical late method. `settings` (default: the scene's) and `threads` (default: their `performance.threads`) go to `simulate()`, with the paths over the edges of barriers and buildings in place of the solver for whole rows (`_over_the_edges()`).

```
acoustic_spaces.walkthrough.walk_points(item, step_m: float = 0.25, step_deg: float =
    20.0) → WalkPoints
```

The places along `item`'s walk (a scene listener or source), `step_m` or a little less apart, from its position to the walk's last point; along keyframes also every `step_deg` of a turn on the spot.

```
acoustic_spaces.walkthrough.with_moving(scene, names: Iterable[str] | None)
```

`scene` with only `names` moving: their walks turned on and every other walk off, so that its walker stands where it starts. None keeps the scene's own. Each name must be a source or listener in use with a walk.

The Doppler shift of a walk: each early path through a delay that changes as the walker goes.

A source moving towards a listener shortens the way its sound has to go while it is on it, so the listener hears its pitch raised by 1 / (1 - M cos theta), M its speed over the speed of sound and theta the angle between the way it goes and the way to the listener; going away, lowered. A listener moving past a source hears the same. Each early path has its own shift (the sound reflected off a wall behind a car falls in pitch while the direct sound still rises), so `render()` follows every path from place to place along the walk:

- **The paths.** A path at one place is the same path at the next when it meets the same planes (the ground, a facade) in the same order over the same edge, failing that when it is of the same kind, and its delay carries on from it: it changes by no more than one and a half times the walker's longest step over the speed of sound, and is nearest where the path's last two places lead. Paths that meet the same planes over the same edge are paired so that the most go on, and of those pairings the one with the delays nearest where they lead. Where a path appears or goes, it fades in or out between two places.
- **The delay.** Each path's delay is known at every place: for a walking source, the time its sound takes from there, heard that much later (the retarded time); for a walking listener, at the moment it is there. Between the places a cubic through them gives the delay at every sample, and the signal is read through it with a windowed sinc (Kaiser, 32 zero crossings each side) whose cutoff falls with the shift, so a raised pitch does not alias: 0.97 of the Nyquist frequency over the most the path raises the pitch (as a ratio), rounded down to 32 / n of the Nyquist frequency, n the taps it then needs on each side, so that the paths share a few tables of taps.
- **The spreading.** The direct sound's level, and a reflection's, falls as one over the path's length, which is its delay times the speed of sound, and a fast walker close by changes that length by much of itself within one window of the filter. So each sample read is divided by the path's delay at that sample (the cubic's, held past the path's own ends, before the eddies move it and without an ear's onset; no less than that of `MIN_SOURCE_DISTANCE_M`, the nearest a listener may stand to a source), and the level follows the distance from sample to sample, as the pitch does.
- **The filter.** What else the path does to the sound (the air, the ground's reflection, a diffracting edge, the source's directivity, the direction it arrives from in the output format, and in turbulence how the eddies delay its other bands from its top one at that moment) goes into a filter: the path's spectrum as the impulse response has it, without its delay and times its delay at the place, which the division by the delay takes back there (a path bent over an edge spreads otherwise: what of that is not one over its length stays in the filter). The filter is known at the places, and between two of them it is theirs in proportion to how near they are. It is taken every `HOP_S` and, between those times, at the time each place's sound is heard; the delayed signal goes through it in windows that overlap, each rising from the time before its own and falling to the time after (triangles, which add up to one), so the filter goes linearly from each place to the next however fast the walker passes them, and a path heard from one place alone gives exactly the convolution with its response.

- **The ears.** Where each ear's response starts (its onset, as the ears are mixed between measured directions: `interpolate()`) is a delay too: it goes into that ear's delay, read through the cubic and the sinc as the path's is, and out of its filter. Measured ears can start a few samples apart from one direction to the next; filters that start apart, mixed between two places or in the overlapping windows, would cancel the treble there. A walking listener's ear is, at each moment, that much further along the path than the head's centre: the onset adds to the path's delay then. A walking source's sound passes the head's centre and reaches the ear an onset later, so at each sample the ear hears what passed the centre an onset earlier: the path's delay, its spreading and how far its filter is between two places are all taken at that earlier time. And while the onset grows, what passed the centre in one sample takes more than one to reach the ear (less while it shrinks), so the sound is divided by one plus the onset's slope: the ear's Doppler factor over the centre's, which the two ears do not share (the centre's own, the convective amplification, is left out: see below). The ears then hear what a standing head makes of the pressure at its centre: each moment of it through the ears' responses to the direction it arrives from. Added to the delay instead, the onset would be a time of the sound sent, not of the sound heard, and the time between the two ears would shrink as the pitch rises and grow as it falls (by up to a fifth at 70 m/s, straight ahead and behind).
- **The eddies.** In turbulence the eddies move each path's delay from one moment to the next (by microseconds, as `Gusts` draws them): the top band's goes into the path's delay, taken between the places and the moments on either side as the filter is, so that two moments' filters mixed do not cancel the treble either.
- **The sound sent.** Every path carries the same signal, unless it is an `Emission`, the sound a source sends each way: then each path carries what is sent the way it leaves the source (its first leg, in the source's frame at each place, in between turning from one place's to the next's) at the time it leaves, and the source's directivity is taken out of its filter.

The late part and the wave solver's bands have no Doppler shift here: the late part of each place is crossfaded to the next's by weights that add up to one, which keep its power as the places' tails, drawn from one random sequence, are alike (a walking source's sound as it is sent, so that each place's late part follows its early paths at the retarded time; a walking listener's as it is heard); and the wave solver's bands are not taken (a walk with Doppler and the wave solver stops with an error). The source radiates as it would standing at each place, as road and rail emission data have it: the convective amplification of a moving source is left out, and emission measured in a pass-by already carries it, as an emission may. Its size depends on the kind of source. For a point monopole (a source of volume flow q) in steady motion the far-field pressure is rho q' / (4 pi R (1 - M cos theta)^2), R and theta as the sound left: two powers of the Doppler factor, 40 lg(1 / (1 - M cos theta)) dB in level, which is 1.6 dB more straight ahead at 30 m/s and 4.0 dB at 70 m/s, 1.5 and 3.2 dB less straight behind.

```
acoustic_spaces.doppler.HOP_S = 0.02
```

How often each path's filter is taken at least (s): the hop of the blocks, between whose centres it is also taken at each place's time (`_turns()`).

```
acoustic_spaces.doppler.KERNEL_S = 0.3
```

Length of a path's filter (s), as the binaural early paths have it.

```
class acoustic_spaces.doppler.Track(present: ndarray, rows: ndarray, received: ndarray,
    delays: ndarray)
```

One path followed along a walk.

present: ndarray

(N,) whether the path is heard from each place.

rows: ndarray

(N,) its row among the paths of each place (-1 where it is not heard).

received: ndarray

(N,) when the sound of each place along it is heard (s from the start).

delays: ndarray

(N,) its delay at each place (s); where it is not heard, carried on from where it is.

property span: tuple[int, int]

The first and last place it is heard from.

acoustic_spaces.doppler.**hermite**(x: ndarray, y: ndarray, at: ndarray, *, hold: bool = True) → ndarray

The cubic Hermite curve through (`x`, `y`) (`x` rising) at `at`, its slope at each point the weighted mean of the chords on either side; held at the ends' values beyond them (or carried on along the end slopes when `hold` is false).

acoustic_spaces.doppler.**plane_ids**(mesh) → ndarray

(F,) the plane each face of `mesh` lies in, numbered: faces in one plane (the triangles of a ground or a wall) share it.

acoustic_spaces.doppler.**read**(dry: ndarray, positions: ndarray, cutoff: float) → ndarray

`dry` (n,) read at the fractional sample `positions` through a windowed sinc cut off at `cutoff` (a share of Nyquist, below one where the reading runs faster than the samples, so nothing aliases); zero before and after it.

acoustic_spaces.doppler.**render**(dry: ndarray | Emission, responses: list, times: ndarray, fmt, sample_rate: int, length: int, *, source_walks: bool, planes: ndarray, step_s: float, gusts=None, seed: int | None = None, axes: ndarray | None = None, directivity=None, span: tuple[int, int] | None = None, followers: Sequence[tuple[int, ndarray]] = ()) → ndarray

(length, channels) the early paths of `responses` (the responses at the places of a walk, in order, the walker there at `times`) heard as it goes, the mono `dry` signal through each path's changing delay and filter (see the module's description). `step_s` is the most a path's delay may jump from one place to the next; `gusts` gives the moments of the eddies in turbulence, `seed` the moment of each place's eddies without them. `dry` may be the sound the source sends each way, at `sample_rate`: then `axes` (N, 3, 3) has the source's forward, left and up axes as columns at each place, and `directivity` (the source's, if it has one) is taken out of the paths' filters. With `span` (begin, end), samples begin to end alone: (end - begin, channels).

`followers` are more mono signals going along the walk after the walker, each (lag, signal) `lag` samples behind it, as the wheels and the cars of a train follow its front: each goes through the same paths, places and filters, all of it `lag` samples later (`length` counts them). They share what the paths cost to follow and filter, so each costs less than a rendering of its own.

```
acoustic_spaces.doppler.tracks(places: list, times: ndarray, *, source_walks: bool,
    planes: ndarray, step_s: float) → list[Track]
```

The paths of `places` (the early paths at each place of a walk, in order), followed from place to place. `times` (N,) are when the walker is at each place; with `source_walks` its sound is heard a path's delay later, otherwise then. `step_s` is the most a delay may jump between places and stay the same path's.

The sound a source sends each way, for a walk with Doppler.

A source's directivity (`acoustic_spaces.directivity`) weighs each path's energy per octave band, and every path carries the same signal. The sound of a rotor changes with the direction in more than its band levels: each harmonic of the blades has a pattern and a phase of its own, and the broadband noise has another. An `Emission` gives the signal itself in every direction, in the source's own frame (x forward, y left, z up, as `acoustic_spaces.scene.Orientation` turns it), and a walk with Doppler (`acoustic_spaces.doppler.render()`) plays down each early path the sound sent the way the path leaves the source, at the time it leaves.

What the early paths carry then comes from the emission alone: each path's share of the source's directivity is taken out of its filter. The directivity still gives the levels the simulation reports and the late part, which plays the emission's sound averaged over the directions (`Emission.averaged()`). For the two to agree, give the source the emission's own directivity (`Emission.directivity()`, written for a scene by `acoustic_spaces.directivity.write_directivity_csv()`) and, for an emission in pascals 1 m away, its sound power (`Emission.sound_power_db()`).

- `Mixture` is made of signals that do not correlate with one another (harmonics, as a cosine and a sine each, and noises in separate bands), each with an amplitude in every direction on the grid of `acoustic_spaces.directivity.on_grid()`.
- `Periodic` is a sound that repeats, such as a rotor's tones: one period of it in every direction, which keeps a long walk's emission small.
- `Sum` sends several emissions at once, such as a helicopter's two rotors.
- `Faded` fades one in and out, so that it does not start or stop at once.

```
class acoustic_spaces.emission.Emission
```

The signal a source sends each way (see the module's description).

`sample_rate: int`

Samples per second.

`property length: int`

How many samples it sends.

`sent(start: int, stop: int, directions: ndarray) → ndarray`

(stop - start,) the samples `start` to `stop` of the sound sent the ways `directions` ((stop - start, 3) unit vectors in the source's frame, one for each sample); nothing before 0 or from `length` on.

`averaged() → ndarray`

(length,) a signal with the emission's power spectrum averaged over the sphere: what the late part carries.

`scaled(gain: float) → Emission`

This emission `gain` times as loud in amplitude.

`directivity(bands: Literal['octave', 'third_octave'] = 'octave', name: str = '') → Balloon`

The emission's directivity: its energy in each band and direction, on the grid `GRID_STEP_DEG` apart that measured balloons use, over the sphere's mean. The bands are IEC 61260-1's, between their base-10 edges (`exact_limits()`), as in `sound_power_db()`.

`sound_power_db(density: float, sound_speed: float, bands: Literal['octave', 'third_octave'] = 'octave') → ndarray`

The sound power level (dB re 1 pW) in each band of an emission that sends the far field's pressure 1 m from the source (Pa): its mean square over the sphere times 4π m² over the air's impedance, `density` times `sound_speed`.

The bands are IEC 61260-1's octaves or third octaves, as ideal filters between their base-10 edges (`exact_limits()`): the 63 Hz octave holds what is sent from 44.7 to 89.1 Hz and the 8 kHz octave what is sent from 5.6 to 11.2 kHz. What is sent below the lowest band's lower edge (such as a rotor's blade passing at 20 Hz) or above the highest's upper edge is in no band.

`class acoustic_spaces.emission.Faded(part: Emission, fade: int = 0, length_samples: int = 0)`

An emission's first `length` samples, faded in over the first `fade` and out over the last, by a squared sine, so that it does not start or stop at once. Its bands and its directivity are the emission's own.

length_samples: int = 0

How many samples of it are sent.

property sample_rate: int

property length: int

How many samples it sends.

sent(start: int, stop: int, directions: ndarray) → ndarray

(stop - start,) the samples `start` to `stop` of the sound sent the ways `directions` ((stop - start, 3) unit vectors in the source's frame, one for each sample); nothing before 0 or from `length` on.

averaged() → ndarray

(length,) a signal with the emission's power spectrum averaged over the sphere: what the late part carries.

scaled(gain: float) → **Faded**

This emission `gain` times as loud in amplitude.

class acoustic_spaces.emission.**Mixture**(signals: ndarray, weights: ndarray, sample_rate: int = 48000)

Signals that do not correlate with one another, each sent with an amplitude of its own in every direction.

signals: ndarray

(K, samples) the signals.

weights: ndarray

(elevations, azimuths, K) each signal's amplitude on the grid of `on_grid()` (elevations from -90 to 90 degrees, azimuths from 0 up to 360, each evenly spaced).

sample_rate: int = 48000

Samples per second.

property length: int

How many samples it sends.

sent(start: int, stop: int, directions: ndarray) → ndarray

(stop - start,) the samples `start` to `stop` of the sound sent the ways `directions` ((stop - start, 3) unit vectors in the source's frame, one for each sample); nothing before 0 or from `length` on.

averaged() → ndarray

(length,) a signal with the emission's power spectrum averaged over the sphere: what the late part carries.

scaled(gain: float) → Mixture

This emission `gain` times as loud in amplitude.

class acoustic_spaces.emission.**Periodic**(table: ndarray, period_s: float = 1.0, sample_rate: int = 48000, samples: int = 0, axes: ndarray = <factory>, turn_s: float = 0.0, centre: ndarray = <factory>, speed: float = 0.0, sound_speed: float = 343.0)

A sound that repeats, sent each way: one period of it in every direction of a grid like that of `on_grid()`, in a frame of its own (`axes`), read between its samples by a cubic through four of them and between the directions bilinearly.

The sound of a rotor turns with its blades. In hover, heard from an azimuth `phi` round the axis it is the sound heard from azimuth 0 a share `phi / 2 pi` of a turn before, and its phase turns that fast between the grid's directions. The table holds each direction's sound from that share of a turn (`turn_s`) later, which makes the directions differ only as slowly as the rotor's pattern, and `sent()` puts the delay back. It also takes into account where the sound comes from (`centre`): from there, the sound sent in a direction `d` leaves `d . centre / (c - v d_x)` later than it would from the source's place, for a source moving at `v` along its x through still air (`speed`, `sound_speed`), and the table holds it as sent from the centre.

`table``: ndarray`

(elevations, azimuths, N) one period in each direction of the grid, in the frame `axes`, N samples evenly spread over it; each direction's shifted as above.

`period_s``: float = 1.0`

How long a period lasts (s).

`sample_rate``: int = 48000`

Samples per second.

`samples``: int = 0`

How many samples it sends.

`axes``: ndarray`

(3, 3) the table's frame: its x, y and z as rows, in the source's frame.

`turn_s``: float = 0.0`

How long one turn takes (s), a whole number of periods: positive for a turn from the table's x towards its y, negative the other way, 0 for a sound that does not turn.

`centre``: ndarray`

(3,) where the sound comes from, in the source's frame (m).

`speed``: float = 0.0`

How fast the source moves along its x through the air (m/s).

`sound_speed``: float = 343.0`

The speed of sound (m/s).

`property length``: int`

How many samples it sends.

`sent``(start: int, stop: int, directions: ndarray) → ndarray`

(stop - start,) the samples `start` to `stop` of the sound sent the ways `directions` ((stop - start, 3) unit vectors in the source's frame, one for each sample); nothing before 0 or from `length` on.

averaged() → ndarray

(length,) a signal with the emission's power spectrum averaged over the sphere: what the late part carries.

scaled(gain: float) → Periodic

This emission `gain` times as loud in amplitude.

class acoustic_spaces.emission.**Sum**(parts: tuple[Emission, ...] = ())

Emissions sent at once from the same source, such as a helicopter's rotors and their broadband noise: what they send adds up. Each part's sound is taken as not correlating with the others' when they are averaged and put in bands.

parts: tuple[Emission, ...] = ()

The emissions, all at one sample rate.

property sample_rate: int

property length: int

How many samples it sends.

sent(start: int, stop: int, directions: ndarray) → ndarray

(stop - start,) the samples `start` to `stop` of the sound sent the ways `directions` ((stop - start, 3) unit vectors in the source's frame, one for each sample); nothing before 0 or from `length` on.

averaged() → ndarray

(length,) a signal with the emission's power spectrum averaged over the sphere: what the late part carries.

scaled(gain: float) → Sum

This emission `gain` times as loud in amplitude.

Helicopters: the sound their rotors send each way, for a walk with Doppler.

A `Helicopter` is its mass, its fuselage's drag and two rotors (`Rotor`). Flying level at an airspeed, `trim()` balances it the way momentum theory does (Leishman, *Principles of Helicopter Aerodynamics*, 2nd ed., chapters 2 and 5): the main rotor's thrust holds the weight up and the fuselage's drag back, its disc tilting forward by the angle between them; Glauert's inflow gives the induced velocity through the disc; the shaft power is the induced power (times `induced`), the blades' profile power (Leishman's `1 + 4.65 mu^2`) and the parasite power of the drag; and the tail rotor's thrust balances the main rotor's torque. `emission()` then gives the sound the two rotors send, as an `Emission` from the main rotor's hub, in the helicopter's frame: x along its flight, y to its left, z up.

The tones. Each blade is a row of stations at Gauss points (split at 97 % of the radius, where the lift ends), each with a steady lift and drag (the lift growing with the radius, as a uniform inflow through an ideally twisted blade gives it, up to 97 % of the radius; the drag an induced part, even along the blade, and a profile part growing with the square of the radius, sized to the trimmed torque) and a NACA four-digit section's thickness. Their far field is that of the Ffowcs Williams and Hawkings equation (Farassat's formulation 1A) for a rotor moving through still air: the loading compact, the thickness a row of sources and sinks at Gauss points along each straight chord, each where it truly is (its radius a little more than the station's, taken to second order). With loads that do not change round the turn, each harmonic of the blade passing frequency comes in closed form, from Bessel functions, in every direction; its convective amplification is in it, and the walk adds the Doppler shift. They are kept as one period of the sound in each direction (`Periodic`).

The broadband noise, by the helicopter's `broadband` model:

- `"pegg"` (the default): Pegg's semi-empirical method (NASA TM-80200, 1979, equations 7 and 8), a fit to measured rotors' broadband noise: the third-octave band levels `20 log (V_T / c)^3 + 10 log (A_b / r^2) (cos^2 theta + 0.1) + S + f(C_L) + 130` round a peak at `-240 log T + 2.448 V_T + 942` Hz, with the spectrum's shape `S` of its figure 9, for each rotor (`theta` from the thrust's axis) (`pegg_broadband()`). It is a level for hovering; in flight it is sent with the convective amplification of a moving dipole, as the tones are. Its bands run from 1/32 to 128 times the peak, the range of the figure.
- `"bpm"`: each blade's own noise in the flow past it, Brooks, Pope and Marcolini's airfoil self-noise (`acoustic_spaces.airfoil_noise`), summed over 24 sections from the blade's root to its tip at 144 places round the turn (`bpm_bands()`). The blades' pitch is trimmed (`pitch()`): rigid blades whose sections' lift is linear in their angle of attack (`lift_slope`), twisted by `twist_deg`, in the trim's inflow, even over the disc, carry its thrust and do not flap. Each section's speed and angle of attack follow round the turn; its levels hold the convective amplification of its own motion through the air, its bands are shifted by its Doppler shift over the hub's (the walk adds the hub's), and each place round the turn counts as long as its sound takes to arrive. The boundary layers turn turbulent by themselves unless `tripped`, the tips are square unless `rounded_tip`, the trailing edges `trailing_edge_m` thick; sections whose flow reverses or whose angle of attack is beyond `BPM_ALPHA_DEG` are left out, and so is the bluntness noise of sections faster than `BLUNT_MACH`. It is the blades' noise in a smooth flow only: the noise of the blades cutting through the wake of those before (blade-wake interaction) and through turbulence is not in it.

What it leaves out. For the tones, loads that change round the turn (in forward flight a blade's lift changes as it goes round, and the blades flap; BPM's sections follow their angle of attack round the turn, but do not flap), the blades meeting the tip vortices of those before (blade-vortex interaction), noise from the engine and the gearbox, the fuselage's reflection and shielding, and wind: the air is still, and the airspeed is the speed over the ground.

```
acoustic_spaces.rotorcraft.AS350_B3 = Helicopter(mass_kg=2250.0, main=Rotor(blades=3,
    radius_m=5.345, chord_m=0.35, rpm=394.0, hub_m=(0.0, 0.0, 0.0), thrust=(0.0, 0.0,
    1.0), turning='clockwise', root=0.2, thickness=0.09, profile_drag=0.01,
    twist_deg=-10.0, lift_slope=5.73, trailing_edge_m=0.0005, trailing_edge_deg=14.0,
    tripped=False, rounded_tip=False), tail=Rotor(blades=2, radius_m=0.93,
    chord_m=0.185, rpm=2086.0, hub_m=(-6.665, 0.0, -1.0), thrust=(0.0, 1.0, 0.0),
    turning='counter-clockwise', root=0.2, thickness=0.12, profile_drag=0.01,
    twist_deg=0.0, lift_slope=5.73, trailing_edge_m=0.0005, trailing_edge_deg=14.0,
    tripped=False, rounded_tip=False), drag_area_m2=0.674, induced=1.15, name='AS350
    B3', broadband='pegg')
```

An Airbus Helicopters AS350 B3 (Ecureuil, sold as the H125) with the Arriel 2B engine at its maximum mass of 2,250 kg. From its type certificate (EASA.R.008): the mass, the rotors' diameters (10.69 m and 1.86 m) and blades (three and two), and the main rotor turning between 385 and 394 turns a minute in powered flight (375 and 405 with the Arriel 2B1 or 2D). From NASA's flight tests of an AS350 SD1 (Greenwood, Sim and Boyd, AHS Forum 72, 2016; Watts et al., NASA TM-2016-219354, table 1): the drag area (f = 7.26 ft², 0.674 m², which they give as f / A = 0.0071, a ratio that fits a larger disc than this one), the main rotor's speed (394 turns a minute, the Arriel 2B's most in powered flight; its tip at 220.6 m/s from their weight coefficient), the tail rotor's speed (2,086 turns a minute, its tip at 203 m/s) and the main rotor's OA209 blades (9 % thick). Assumed, from no source to hand: the main blades' chord (0.35 m) and twist (-10 degrees), the tail rotor's chord (0.185 m), thickness (12 %), untwisted blades and the way it turns, both rotors' trailing edges (a rotor's default), and its hub 6.67 m behind the main rotor's shaft (from Airbus's 12.94 m overall with the rotors turning, less both rotors' radii) and 1 m lower. The main rotor turns clockwise seen from above.

```
acoustic_spaces.rotorcraft.BLUNT_MACH = 0.45
```

Brooks, Pope and Marcolini's bluntness noise is left out of blade sections faster than this Mach number, as their appendix C recommends for rotors.

```
acoustic_spaces.rotorcraft.BPM_ALPHA_DEG = 25.0
```

Their fits hold for angles of attack up to this (degrees); sections beyond it, in the slow flow next to where it reverses, are left out.

```
acoustic_spaces.rotorcraft.BPM_SOURCE_BANDS = (7, 46)
```

The bands (`10^(k/10)` Hz) BPM's levels are found in, in a blade section's frame, and those the noise is sent in from the hub, after each section's Doppler shift.

```
acoustic_spaces.rotorcraft.BROADBAND_MODELS = ('bpm', 'pegg')
```

The ways a helicopter's broadband noise can be found (`Helicopter.broadband`).

```
acoustic_spaces.rotorcraft.GRAVITY = 9.80665
```

Standard gravity (m/s²).

acoustic_spaces.rotorcraft.**PEGG_SPECTRUM_DB** = (-29.0, -24.5, -19.5, -15.3, -11.7, -7.5, -11.5, -12.1, -16.5, -17.0, -21.8, -26.4, -30.0)

Pegg's figure 9: third-octave band levels (dB, over the overall level) at 1/32, 1/16, …, 64 and 128 times the peak frequency, one octave apart.

acoustic_spaces.rotorcraft.**QUIET_DB** = 90.0

A rotor's tones keep the harmonics of the blade passing frequency up to the last one within this many dB of the loudest harmonic in any direction.

acoustic_spaces.rotorcraft.**TIP_LOSS** = 0.97

The share of the radius out to which a blade carries lift (Leishman's tip loss).

class acoustic_spaces.rotorcraft.**Helicopter**(mass_kg: float, main: Rotor, tail: Rotor, drag_area_m2: float, induced: float = 1.15, name: str = '', broadband: str = 'pegg')

A single main rotor helicopter with a tail rotor.

drag_area_m2: float

The fuselage's drag as the area of a flat plate (m²): `D = rho V^2 f / 2`.

induced: float = 1.15

The induced power over that of momentum theory (Leishman's `kappa`).

broadband: str = 'pegg'

How its rotors' broadband noise is found: "pegg" (Pegg's semi-empirical method for the whole rotor) or "bpm" (Brooks, Pope and Marcolini's airfoil self-noise of each blade section round the turn).

property tail_arm_m: float

How far the tail rotor's thrust acts from the main rotor's shaft (m).

class acoustic_spaces.rotorcraft.**Load**(thrust_n: float, axis: ndarray, induced_m_s: float = 0.0, induced_torque_nm: float = 0.0, profile_torque_nm: float = 0.0)

What a rotor carries in a trim.

thrust_n: float

Its thrust (N).

axis: ndarray

(3,) the way the thrust points, in the helicopter's frame.

induced_m_s: float = 0.0

The induced velocity through the disc (m/s).

induced_torque_nm: float = 0.0

The torque (N m) of its induced and parasite power, and of its blades' profile drag.

class acoustic_spaces.rotorcraft.**Rotor**(blades: int, radius_m: float, chord_m: float, rpm: float, hub_m: tuple[float, float, float] = (0.0, 0.0, 0.0), thrust: tuple[float,

```
float, float] = (0.0, 0.0, 1.0), turning: str = 'counter-clockwise', root: float =
0.2, thickness: float = 0.12, profile_drag: float = 0.01, twist_deg: float = 0.0,
lift_slope: float = 5.73, trailing_edge_m: float = 0.0005, trailing_edge_deg: float
= 14.0, tripped: bool = False, rounded_tip: bool = False)
```

A rotor: its blades, its speed and where it sits on the helicopter.

`chord_m``: float`

Each blade's chord (m), the same along it.

`rpm``: float`

Turns a minute.

`hub_m``: tuple[float, float, float] = (0.0, 0.0, 0.0)`

Where its hub is (m), in the helicopter's frame (x along its flight, y to its left, z up), from the helicopter's place: the main rotor's hub is best at the origin, where the sound is sent from.

`thrust``: tuple[float, float, float] = (0.0, 0.0, 1.0)`

The way its thrust pushes the helicopter, before the trim tilts it: up for a main rotor, sideways for a tail rotor.

`turning``: str = 'counter-clockwise'`

How its blades turn, seen from the side its thrust points to: "counter-clockwise" or "clockwise".

`root``: float = 0.2`

The share of the radius where the blades begin.

`thickness``: float = 0.12`

The blades' thickness over their chord (a NACA four-digit symmetric section).

`profile_drag``: float = 0.01`

The blade sections' mean profile drag coefficient.

`twist_deg``: float = 0.0`

How much the blades' pitch changes from the hub to the tip (degrees, linearly; negative, the tip pitched down), for the blade sections' angles of attack.

`lift_slope``: float = 5.73`

The sections' lift curve slope (per radian).

`trailing_edge_m``: float = 0.0005`

The trailing edge's thickness (m; 0 for a sharp edge, which sends none) and the angle its surfaces meet at (0 to 14 degrees), for Brooks, Pope and Marcolini's bluntness noise.

`tripped``: bool = False`

Whether the blades' boundary layers are tripped (else they go turbulent by themselves), and whether their tips are rounded (else square).

`property omega``: float`

How fast it turns (rad/s).

`property tip_speed``: float`

The blade tips' speed round the hub (m/s).

```
    property blade_area_m2: float
```

All its blades' area, as Pegg's `A_b` and the solidity count it.

```
    property spin: int
```

+1 for a turn counter-clockwise seen from where the thrust points, -1 the other way.

```
class acoustic_spaces.rotorcraft.Trim(helicopter: Helicopter, airspeed_m_s: float,
    temperature_c: float, pressure_kpa: float, main: Load, tail: Load)
```

A helicopter balanced in level flight.

```
    airspeed_m_s: float
```

Its speed through the air (m/s).

```
    property power_w: float
```

The power both rotors take (W).

```
    lift_coefficient(which: str) → float
```

A rotor's mean lift coefficient, `6 C_T / sigma`: "main" or "tail".

```
acoustic_spaces.rotorcraft.bpm_bands(trim: Trim, which: str, ways: ndarray, *, segments:
    int = 24, azimuths: int = 144) → tuple[ndarray, ndarray]
```

A rotor's broadband noise in a trim by Brooks, Pope and Marcolini's airfoil self-noise (`acoustic_spaces.airfoil_noise`), sent towards `ways` (…, 3) in the helicopter's frame: the third-octave bands' centres (Hz) and their mean square pressures 1 m from the hub (Pa², `(..., bands)`), averaged over a turn.

Each blade is `segments` sections from its root to its tip, each at `azimuths` places round the turn, with the flow past it and its angle of attack from the trim's pitch (`pitch()`). A section's levels hold the convective amplification of its motion through the air; its bands are shifted by its own Doppler shift over the hub's (the walk adds the hub's), and each place round the turn counts as long as its sound takes to arrive. The bluntness noise is left out of sections faster than `BLUNT_MACH` and sections beyond `BPM_ALPHA_DEG`, or where the flow reverses, are left out; the tip's vortex is the outermost section's.

```
acoustic_spaces.rotorcraft.bpm_powers(trim: Trim, *, grid_deg: float = 5.0, rotors:
    tuple[str, ...] = ('main', 'tail')) → tuple[ndarray, ndarray]
```

The `rotors`' broadband noise in a trim by `bpm_bands()` on a grid `grid_deg` apart in the helicopter's frame: the third-octave bands' centres (Hz) and the rotors' mean square pressures 1 m from the hub, summed, `(elevations, azimuths, bands)`. They do not depend on how long the noise lasts: `broadband()` takes them as `bands` to send another length without summing the blades again.

```
acoustic_spaces.rotorcraft.broadband(trim: Trim, *, samples: int, sample_rate: int =
    48000, grid_deg: float = 5.0, seed: int = 0, rotors: tuple[str, ...] = ('main',
    'tail'), model: str | None = None, bands: tuple[ndarray, ndarray] | None = None) →
    Mixture
```

The rotors' broadband noise in a trim, `samples` of it, its pattern on a grid `grid_deg` apart in the helicopter's frame, by the helicopter's `broadband` model or `model`: "bpm" (`bpm_bands()`, a noise in each third-octave band with its level in every direction) or "pegg" (`pegg_broadband()`). For "bpm", `bands` are the bands' centres and powers that `bpm_powers()` gives for this trim, grid and `rotors`, if already found (their shape is checked, not their trim or rotors); by default they are found here.

```
acoustic_spaces.rotorcraft.emission(trim: Trim, *, seconds: float, sample_rate: int =
    48000, grid_deg: float = 5.0, seed: int = 0) → Sum
```

All the helicopter sends in a trim for `seconds`: both rotors' tones and their broadband noise, from the main rotor's hub.

```
acoustic_spaces.rotorcraft.pegg_bands(peak_hz: float) → tuple[ndarray, ndarray]
```

The third-octave bands of Pegg's spectrum round a peak: their centres (Hz, base-10 exact) and their levels `S` (dB) from his figure 9, the peak in the standard band it falls in.

```
acoustic_spaces.rotorcraft.pegg_broadband(trim: Trim, *, samples: int, sample_rate: int =
    48000, grid_deg: float = 5.0, seed: int = 0, rotors: tuple[str, ...] = ('main',
    'tail')) → Mixture
```

The rotors' broadband noise in a trim by Pegg's method (see the module's description), `samples` of it, its pattern on a grid `grid_deg` apart in the helicopter's frame: one noise for each of `rotors`.

```
acoustic_spaces.rotorcraft.pegg_level_db(tip_speed: float, sound_speed: float,
    blade_area_m2: float, lift_coefficient: float) → float
```

Pegg's level (dB) of a rotor's broadband noise at 1 m, before its pattern (`10 log (cos^2 theta + 0.1)`) and its spectrum (`S`): `20 log (V_T / c)^3 + 10 log A_b + f(C_L) + 130`.

```
acoustic_spaces.rotorcraft.pegg_lift_db(lift_coefficient: float) → float
```

Pegg's `f(C_L)` (dB), his appendix B.

```
acoustic_spaces.rotorcraft.pegg_peak_hz(thrust_n: float, tip_speed: float) → float
```

Pegg's peak frequency of a rotor's broadband noise (Hz), his equation 7 in SI units (thrust in N, tip speed in m/s).

```
acoustic_spaces.rotorcraft.pitch(trim: Trim, which: str, stations: int = 24, azimuths:
    int = 64) → ndarray
```

A rotor's blade pitch in a trim (radians at the hub, by `twist_deg` less at the tip): its collective `theta_0` and its cyclic `theta_1c`, `theta_1s` (with `cos psi` and `sin psi`, `psi` the blade's azimuth from straight downwind, turning with the rotor), such that its blades carry the trim's thrust and do not flap round the turn: their lift's moment about the hub has no part that goes once round with it. The blades are rigid, their sections' lift linear in the angle of attack (`lift_slope`), the inflow even over the disc (the trim's); the lift runs from the root to the tip loss, as the tones' does.

```
acoustic_spaces.rotorcraft.tones(trim: Trim, which: str, *, samples: int, sample_rate:
    int = 48000, harmonics: int | None = None, grid_deg: float = 5.0, stations: int =
    24) → Periodic
```

The tones a rotor (“main” or “tail”) sends in a trim, `samples` of them, as one period in every direction `grid_deg` apart round its axis; `harmonics` of the blade passing frequency (by default up to the last within `QUIET_DB` of the loudest).

```
acoustic_spaces.rotorcraft.trim(helicopter: Helicopter, airspeed_m_s: float,
    temperature_c: float = 15.0, pressure_kpa: float = 101.325) → Trim
```

The helicopter balanced flying level at `airspeed_m_s` (see the module’s description).

Airfoil self-noise: the noise an airfoil makes in a smooth flow, by Brooks, Pope and Marcolini’s method (NASA Reference Publication 1218, 1989, “BPM”).

BPM scaled wind tunnel measurements of NACA 0012 sections, 2.5 to 30 cm in chord, at speeds up to 71 m/s, into third-octave band levels from five mechanisms:

- **turbulent boundary layer, trailing edge** (TBL-TE): the turbulence of each side’s boundary layer passing the trailing edge, and at an angle of attack the flow separating from the suction side, which takes over past stall;
- **laminar boundary layer vortex shedding** (LBL-VS): a laminar boundary layer’s instabilities feeding back from the trailing edge, at low Reynolds numbers;
- **trailing-edge bluntness**: vortices shed from a trailing edge of thickness `h`;
- **tip vortex formation**: the turbulence where the tip’s vortex forms.

Each segment of an airfoil is its chord, its span, the speed and the angle of attack of the flow, and where the observer is: `r` away, at an angle `theta` from the chord line (pointing downstream, from the leading edge to the trailing edge), and `phi` from the span to the observer’s projection on the plane of the span and the normal, across the chord (the report’s figure B3). The levels are those of the report’s code (its appendix D), checked against its two sample outputs, tables D2 and D3, and are for a segment moving through still air at the speed of the flow: the high-frequency directivity `D_h` (its equation B1) and the low-frequency `D_l` (B2) hold the convective amplification of that motion. The angles of attack are in degrees, from the angle of zero lift.

```
acoustic_spaces.airfoil_noise.FLOOR_DB = -100.0
```

The level (dB) the code puts a mechanism at when it gives less.

```
acoustic_spaces.airfoil_noise.FREQUENCIES_HZ = (100.0, 125.0, 160.0, 200.0, 250.0, 315.0,
    400.0, 500.0, 630.0, 800.0, 1000.0, 1250.0, 1600.0, 2000.0, 2500.0, 3150.0, 4000.0,
    5000.0, 6300.0, 8000.0, 10000.0, 12500.0, 16000.0, 20000.0, 25000.0, 31500.0,
    40000.0)
```

The report’s third-octave band centres (Hz), as its code gives them.

```
acoustic_spaces.airfoil_noise.LIGHTLY_TRIPPED = 2
```

Boundary layers: natural transition, tripped heavily, and tripped lightly.

```
acoustic_spaces.airfoil_noise.TRIPPED = 1
```

Boundary layers: natural transition, tripped heavily, and tripped lightly.

```
acoustic_spaces.airfoil_noise.UNTRIPPED = 0
```

Boundary layers: natural transition, tripped heavily, and tripped lightly.

acoustic_spaces.airfoil_noise.**bluntness**(frequencies, chord, span, distance, theta_deg, phi_deg, alpha, speed, thickness, angle_deg, *, trip: int, viscosity: float, sound_speed: float) → ndarray

Trailing-edge bluntness noise (the report's equations 73 to 85), band levels (dB), for a trailing edge `thickness` thick (m) whose surfaces meet at `angle_deg` (14 degrees for a NACA 0012).

acoustic_spaces.airfoil_noise.**directivity_high**(mach, theta_deg, phi_deg) → ndarray

`D_h`: a trailing edge's noise at high frequencies, 1 at `theta` = `phi` = 90 degrees, with the eddies convected at 0.8 of the flow's Mach number.

acoustic_spaces.airfoil_noise.**directivity_low**(mach, theta_deg, phi_deg) → ndarray

`D_l`: a translating dipole's, for a stalled airfoil's noise.

acoustic_spaces.airfoil_noise.**lbl_vs**(frequencies, chord, span, distance, theta_deg, phi_deg, alpha, speed, *, trip: int, viscosity: float, sound_speed: float) → ndarray

LBL-VS noise (the report's equations 53 to 60), band levels (dB).

acoustic_spaces.airfoil_noise.**stalled**(alpha, mach) → ndarray

Whether an airfoil at `alpha` (degrees) in a flow at `mach` has stalled for the TBL-TE noise (the report's `gamma_0` or 12.5 degrees).

acoustic_spaces.airfoil_noise.**tbl_te**(frequencies, chord, span, distance, theta_deg, phi_deg, alpha, speed, *, trip: int, viscosity: float, sound_speed: float) → tuple[ndarray, ndarray, ndarray]

TBL-TE and separation noise (the report's equations 24 to 30): the band levels (dB, `(..., frequencies)`) of the pressure side, the suction side and the angle of attack's part (the flow separating). Stalled, each side keeps only its scaling term, 10 log10(delta*_s M^5 L D_l / r^2), as the report's code has it (its equations 28 and 29 drop the sides), and the angle of attack's part is that of a translating dipole.

acoustic_spaces.airfoil_noise.**thicknesses**(chord: ndarray, speed: ndarray, alpha: ndarray, trip: int, viscosity: float) → tuple[ndarray, ndarray, ndarray]

The boundary layer at the trailing edge (the report's section 3, as its code has it): the pressure side's thickness, and the suction and pressure sides' displacement thicknesses (m).

acoustic_spaces.airfoil_noise.**tip**(frequencies, chord, distance, theta_deg, phi_deg, alpha_tip, speed, *, rounded: bool, sound_speed: float, lift_ratio: float = 1.0) → ndarray

Tip vortex formation noise (the report's equations 61 to 67), band levels (dB), for a tip at `alpha_tip` (its lift's slope near the tip `lift_ratio` times that of the report's untwisted blade), rounded or square.

Aircraft noise (ECAC Doc 29)

Aircraft noise by ECAC Doc 29, *Report on Standard Method of Computing Noise Contours around Civil Airports* (5th edition, endorsed 19 May 2026): the noise of a single movement of an aircraft at receivers on the ground, from its NPD curves along a flight path built of segments.

- `tables` reads the aircraft, NPD curves, spectral classes and fixed-point profiles from tables laid out as the ANP database's (Volume 2 Appendix G);
- `path` builds the flight path's segments (Volume 2 section 3.6);
- `noise` gives each segment's sound exposure and maximum level and the event's (Volume 2 chapter 4);
- `performance` gives an aircraft's fixed-point profile from its procedural steps, its engine and aerodynamic coefficients and its weight (Volume 2 Appendix B);
- `volume3` holds Volume 3's three hypothetical reference aircraft: Part 1's routes, receptors and reference sound exposure levels, and Part 2's procedures, aerodromes and reference flight profiles;
- `anp` reads a copy of the ANP database that the user has downloaded;
- `atmosphere` adjusts an aircraft's NPD curves to the air at the aerodrome (Volume 2 Appendix D);
- `source` makes an aircraft a source for the engine: its sound power and directivity from its NPD curves, its spectral class and its installation correction, and its flight path as a walk timed as it flies.

The ANP database's own aircraft are not part of this package: its terms (EASA's ANP Database Terms and Conditions, issue 3, 2024) allow their use only for noise contours of an average fleet and traffic and forbid passing them on. The readers of `tables`, `performance` and `anp` take tables with the fields Volume 2 Appendix G lists, as Volume 3's workbook lays them out; they have not been tried on a download of the database.

```
class acoustic_spaces.doc29.Aircraft(name: str, description: str, engine_type: str,
    engines: int, npd: str, power_parameter: str, approach_spectrum: str,
    departure_spectrum: str, installation: str)
```

An aircraft of the ANP database's aircraft table (Volume 2 Appendix G2): what the noise calculation needs of it.

`name``: str`

Its identifier, its description, its engines ("Jet", "Turboprop" or "Piston") and how many it has.

`npd``: str`

The identifier of its NPD curves, and the power parameter they are given against.

`approach_spectrum``: str`

The identifiers of its spectral classes on approach and on departure.

`installation``: str`

Its lateral directivity, "Fuselage", "Wing" or "Prop" (`INSTALLATIONS`): the engine installation correction of Volume 2 section 4.5.3.

`property jet``: bool`

Whether its engines are jets (the start-of-roll directivity of jets).

`property power_in_lb``: bool`

Whether its NPD curves' power parameter is in pounds, as "Corrected Net Thrust (lb)" (a percentage of one is not): the corrected net thrust per engine that Appendix B's procedural steps give.

```
class acoustic_spaces.doc29.Event(sel: ndarray, lamax: ndarray, segments:
    tuple[SegmentLevels, ...])
```

One movement's levels at the receivers (dB(A)), and each segment's part.

```
class acoustic_spaces.doc29.NpdCurves(powers: tuple[float, ...], levels:
    tuple[tuple[float, ...], ...])
```

One set of NPD curves (noise-power-distance, Volume 2 section 4.2): a level (dB) at each of `NPD_FEET` for each power, interpolated linearly in power (eq. 4-3) and in the logarithm of the distance (eq. 4-4), and extrapolated from the last two curves or distances beyond them (eq. 4-5).

`levels``: tuple[tuple[float, ...], ...]`

The levels, a row for each power (as many as `powers`), a column for each distance.

`level``(power: ndarray | float, distance_m: ndarray | float) → ndarray`

The level at `power` and `distance_m` (m), each an array or a number.

exception acoustic_spaces.doc29.**NpdExtrapolationWarning**

A flight's power lies so far outside its NPD curves' powers that their levels, extrapolated in power (eq. 4-3), depart more than `NPD_ENVELOPE_DB` from the levels at the nearest power of the curves (Volume 2 section 4.2).

class acoustic_spaces.doc29.**ProfilePoint**(distance_m: float, height_m: float, speed_m_s: float, power: float)

A point of a fixed-point profile: the distance along the ground track (m; from the start of roll on departure, to touchdown on approach, negative before it), the height above the runway (m), the true airspeed (m/s) and the power (the NPD curves' parameter, per engine).

class acoustic_spaces.doc29.**Segment**(start: tuple[float, float, float], end: tuple[float, float, float], speeds: tuple[float, float], powers: tuple[float, float], banks: tuple[float, float] = (0.0, 0.0), roll: str = '')

A straight segment of a flight path, from `start` to `end` (x, y, z in m), with the speed, power and bank angle at each end, and whether it is part of the take-off roll or the landing roll.

class acoustic_spaces.doc29.**SegmentLevels**(q: ndarray, slant: ndarray, d1: ndarray, d2: ndarray, lateral: ndarray, npd_distance: ndarray, power: ndarray, elevation: ndarray, climb: float, depression: ndarray, installation: ndarray, lateral_attenuation: ndarray, baseline: ndarray, duration: ndarray, maximum: ndarray, fraction: ndarray, start_of_roll: ndarray, impedance: float, exposure: ndarray, peak: ndarray)

One segment's contribution at each receiver, with the terms that make it up (arrays, one value a receiver): the distances (m), angles (degrees) and levels (dB) Volume 3 lists for its reference cases.

peak: ndarray

The segment's maximum level at each receiver (dB(A)), eq. 4-8a.

class acoustic_spaces.doc29.**SpectralClass**(name: str, mode: str, description: str, levels: tuple[float, ...])

A spectral class (Volume 2 Appendix D, G): the spectrum, in third octaves of `SPECTRAL_BANDS_HZ`, of an aircraft's sound 1000 ft (305 m) under it, its level 70 dB at 1 kHz.

class acoustic_spaces.doc29.**Track**(points: tuple[tuple[float, float], ...], start_m: float = 0.0)

A ground track: its points (x, y in m) in the direction of flight, and the distance along it of its first point (m). Doc 29 measures a departure's track from the start of roll and an arrival's from the landing threshold (section 3.3.2).

classmethod **through**(points: Sequence[tuple[float, float]], origin: tuple[float, float]) → Track

The track through `points` measured from `origin`, a point on it (the start of roll or the landing threshold); a point within `SAME_PLAN_POINT_M` of the one before it is left out.

property **node_distances**: ndarray

The distance along the track of each of its points (m).

extended(first_m: float, last_m: float) → Track

The track reaching back to `first_m` and on to `last_m` at least, its first and last legs made longer if they must be (section 3.6.6), the way the first and last that have a length go.

at(s: float) → tuple[float, float]

The point of the track `s` along it (m), on the leg that holds it, or on from the first or last leg that has a length before or after the track.

acoustic_spaces.doc29.**event**(aircraft: Aircraft, exposure: NpdCurves, maximum: NpdCurves, segments: Sequence[Segment], receivers: ndarray, impedance: float = 0.0) → Event

The sound exposure level (eq. 4-11) and the maximum level (eq. 4-10) of the flight `segments` of `aircraft` at `receivers`. A segment whose NPD level at its power departs more than `NPD_ENVELOPE_DB` from the level at the nearest power of the curves gives an `NpdExtrapolationWarning` (section 4.2).

`impedance` is Doc 29's impedance adjustment (dB, eq. 4-6 and 4-7) added to every segment's levels: `impedance_db()` of the aerodrome's temperature and pressure. Its default, 0, is the NPD curves' reference day (25 °C, 101.325 kPa), not the standard atmosphere at sea level, for which the adjustment is 0.074 dB.

acoustic_spaces.doc29.**flight_path**(profile: Sequence[ProfilePoint], track: Track, departure: bool, threshold_m: float = 0.0) → list[Segment]

The segments of the flight of `profile` along `track`, a departure or an arrival. An arrival's profile measures its distances to touchdown and its track from the landing threshold: `threshold_m` is the profile's distance at the threshold (negative, before touchdown). The profile is carried on, at its last speed and power and its last slope, to the end of the track, or back to its start on arrival, and the track on to the end of the profile (section 3.6.6).

acoustic_spaces.doc29.**impedance_db**(temperature_c: float = 15.0, pressure_kpa: float = 101.325) → float

The impedance adjustment of the NPD levels to the air at the aerodrome (dB), eq. 4-6 and 4-7.

acoustic_spaces.doc29.**read_aircraft**(path: str | Path) → dict[str, Aircraft]

The aircraft of the aircraft table `path`, by identifier, each its lateral directivity the installation of `INSTALLATIONS` it names.

```
acoustic_spaces.doc29.read_fixed_point_profiles(path: str | Path) → dict[tuple[str, str,
    str, str], tuple[ProfilePoint, ...]]
```

The fixed-point profiles of the table `path`, by (aircraft, operation mode “A” or “D”, profile identifier, stage length), each its points in order.

```
acoustic_spaces.doc29.read_npd(path: str | Path) → dict[tuple[str, str, str], NpdCurves]
```

The NPD curves of the NPD table `path`, by (NPD identifier, noise descriptor, operation mode): (“JETF”, “SEL”, “D”) for instance, the descriptor “SEL” or “LAmax” and the mode “A” (arrival) or “D” (departure).

```
acoustic_spaces.doc29.read_spectral_classes(path: str | Path) → dict[str, SpectralClass]
```

The spectral classes of the table `path`, by identifier.

```
acoustic_spaces.doc29.segment_levels(aircraft: Aircraft, exposure: NpdCurves, maximum:
    NpdCurves, segment: Segment, receivers: ndarray, impedance: float = 0.0) →
    SegmentLevels
```

The contribution of `segment` of a flight of `aircraft` at `receivers` (an array of x, y, z in m, one a row), its exposure from the SEL curves `exposure` and its maximum level from the LAmax curves `maximum`.

`impedance` is Doc 29’s impedance adjustment (dB, eq. 4-6 and 4-7), added to both levels: `impedance_db()` of the aerodrome’s temperature and pressure. Its default, 0, is the NPD curves’ reference day (25 °C, 101.325 kPa), not the standard atmosphere at sea level, for which the adjustment is 0.074 dB.

Doc 29’s aircraft data read from tables laid out as the ANP database’s (Doc 29 Volume 2 Appendix G, as Volume 3’s workbook lays out its reference aircraft): the aircraft, their NPD curves, their spectral classes and their fixed-point profiles.

A table is a CSV file with a header row; its separator may be a comma or a semicolon. Columns are found by their names, whatever their case, spaces and punctuation, and a length or a speed is read in the unit its column’s name gives in round or square brackets, as Appendix G’s “Distance (ft)” and Volume 3’s “Distance (m)”: m, ft or feet for a length, m/s, kt, kts or knots for a speed. A column that names another unit, or none, is an error.

```
acoustic_spaces.doc29.tables.FOOT_M = 0.3048
```

A foot (m), a knot (m/s) and a nautical mile (m).

```
acoustic_spaces.doc29.tables.LENGTH_UNITS = {'feet': 0.3048, 'ft': 0.3048, 'm': 1.0}
```

The units a column of lengths (m) or of speeds (m/s) may give in brackets, its case and spaces aside.

```
acoustic_spaces.doc29.tables.INSTALLATIONS = ('Fuselage', 'Wing', 'Prop')
```

The engine installations a lateral directivity identifier names by its start, its case and punctuation aside (Volume 2 Appendix G2: “Fuselage-mounted”, “Wing-mounted” or “Prop”).

```
acoustic_spaces.doc29.tables.NPD_FEET = (200.0, 400.0, 630.0, 1000.0, 2000.0, 4000.0,
    6300.0, 10000.0, 16000.0, 25000.0)
```

The distances of the NPD tables (ft), 200 to 25000 ft (Volume 2 Appendix G).

acoustic_spaces.doc29.tables.**SPECTRAL_BANDS_HZ** = (50, 63, 80, 100, 125, 160, 200, 250, 315, 400, 500, 630, 800, 1000, 1250, 1600, 2000, 2500, 3150, 4000, 5000, 6300, 8000, 10000)

The third-octave bands of a spectral class (Hz), 50 Hz to 10 kHz.

acoustic_spaces.doc29.tables.**read_rows**(path: str | Path) → list[dict[str, str]]

The rows of the CSV table `path`, each a dict from its column's name to its cell (stripped).

acoustic_spaces.doc29.tables.**installation_of**(label: str) → str

The engine installation, one of `INSTALLATIONS`, that the lateral directivity identifier `label` names by its start ("Wing-mounted" names "Wing").

class acoustic_spaces.doc29.tables.**Aircraft**(name: str, description: str, engine_type: str, engines: int, npd: str, power_parameter: str, approach_spectrum: str, departure_spectrum: str, installation: str)

An aircraft of the ANP database's aircraft table (Volume 2 Appendix G2): what the noise calculation needs of it.

name: str

Its identifier, its description, its engines ("Jet", "Turboprop" or "Piston") and how many it has.

npd: str

The identifier of its NPD curves, and the power parameter they are given against.

approach_spectrum: str

The identifiers of its spectral classes on approach and on departure.

installation: str

Its lateral directivity, "Fuselage", "Wing" or "Prop" (`INSTALLATIONS`): the engine installation correction of Volume 2 section 4.5.3.

property jet: bool

Whether its engines are jets (the start-of-roll directivity of jets).

property power_in_lb: bool

Whether its NPD curves' power parameter is in pounds, as "Corrected Net Thrust (lb)" (a percentage of one is not): the corrected net thrust per engine that Appendix B's procedural steps give.

acoustic_spaces.doc29.tables.**read_aircraft**(path: str | Path) → dict[str, **Aircraft**]

The aircraft of the aircraft table `path`, by identifier, each its lateral directivity the installation of `INSTALLATIONS` it names.

class acoustic_spaces.doc29.tables.**NpdCurves**(powers: tuple[float, ...], levels: tuple[tuple[float, ...], ...])

One set of NPD curves (noise-power-distance, Volume 2 section 4.2): a level (dB) at each of `NPD_FEET` for each power, interpolated linearly in power (eq. 4-3) and in the logarithm of the distance (eq. 4-4), and extrapolated from the last two curves or distances beyond them (eq. 4-5).

`levels``: tuple[tuple[float, ...], ...]`

The levels, a row for each power (as many as `powers`), a column for each distance.

`level``(power: ndarray | float, distance_m: ndarray | float) → ndarray`

The level at `power` and `distance_m` (m), each an array or a number.

`acoustic_spaces.doc29.tables.`**`read_npd`**`(path: str | Path) → dict[tuple[str, str, str], NpdCurves]`

The NPD curves of the NPD table `path`, by (NPD identifier, noise descriptor, operation mode): (“JETF”, “SEL”, “D”) for instance, the descriptor “SEL” or “LAmax” and the mode “A” (arrival) or “D” (departure).

`class` `acoustic_spaces.doc29.tables.`**`SpectralClass`**`(name: str, mode: str, description: str, levels: tuple[float, ...])`

A spectral class (Volume 2 Appendix D, G): the spectrum, in third octaves of `SPECTRAL_BANDS_HZ`, of an aircraft’s sound 1000 ft (305 m) under it, its level 70 dB at 1 kHz.

`acoustic_spaces.doc29.tables.`**`read_spectral_classes`**`(path: str | Path) → dict[str, SpectralClass]`

The spectral classes of the table `path`, by identifier.

`class` `acoustic_spaces.doc29.tables.`**`ProfilePoint`**`(distance_m: float, height_m: float, speed_m_s: float, power: float)`

A point of a fixed-point profile: the distance along the ground track (m; from the start of roll on departure, to touchdown on approach, negative before it), the height above the runway (m), the true airspeed (m/s) and the power (the NPD curves’ parameter, per engine).

`acoustic_spaces.doc29.tables.`**`read_fixed_point_profiles`**`(path: str | Path) → dict[tuple[str, str, str, str], tuple[ProfilePoint, ...]]`

The fixed-point profiles of the table `path`, by (aircraft, operation mode “A” or “D”, profile identifier, stage length), each its points in order.

`acoustic_spaces.doc29.tables.`**`points_m`**`(rows: Sequence[dict[str, str]], x: str, y: str) → list[tuple[float, float]]`

The points (m) of the columns `x` and `y` of `rows`, in the unit of their names.

A flight path built of straight segments, as Doc 29 builds it (Volume 2 section 3.6): a profile of heights, speeds and powers along a ground track, the take-off and landing rolls cut into segments by speed, the initial climb and the final approach by height, airborne segments that change speed much by speed, and the profile merged with the track's nodes.

```
acoustic_spaces.doc29.path.CLIMB_HEIGHTS_M = (18.8976, 41.4528, 68.2752, 102.108,
    147.5232, 214.88400000000001, 334.97520000000003, 609.6, 1289.6088)
```

The heights (m) that cut the initial climb and the final approach below 4231 ft (1289.6 m), for at most 1.5 dB of change of the lateral attenuation along a segment (Volume 2 section 3.6.4): its heights in feet, which Volume 3's reference cases use (its heights in metres are these rounded).

```
acoustic_spaces.doc29.path.SPEED_CHANGE_M_S = 10.28888888888889
```

How much an airborne segment's speed may change before it is cut (m/s), 20 kt, and the step of speed it is cut into, and a roll on the runway is, 10 m/s (section 3.6).

```
acoustic_spaces.doc29.path.NEAREST_POINTS_M = 10.0
```

How near two points of the same speed and power may be before one goes (m), section 3.6.7, and how low a source may be over the runway (m), section 3.5.

```
acoustic_spaces.doc29.path.SAME_PLAN_POINT_M = 1e-06
```

How near two consecutive points of a ground track may be before the later one goes (m): a walk's places one above the other have one point in plan.

```
class acoustic_spaces.doc29.path.Node(s: float, height: float, speed: float, power:
    float, bank: float = 0.0, ground: bool = False)
```

A point of a flight path: where it is (m), its speed (m/s), its power, its bank angle (degrees, positive in a left turn), and whether it is on the runway.

```
class acoustic_spaces.doc29.path.Segment(start: tuple[float, float, float], end:
    tuple[float, float, float], speeds: tuple[float, float], powers: tuple[float,
    float], banks: tuple[float, float] = (0.0, 0.0), roll: str = '')
```

A straight segment of a flight path, from `start` to `end` (x, y, z in m), with the speed, power and bank angle at each end, and whether it is part of the take-off roll or the landing roll.

```
class acoustic_spaces.doc29.path.Track(points: tuple[tuple[float, float], ...], start_m:
    float = 0.0)
```

A ground track: its points (x, y in m) in the direction of flight, and the distance along it of its first point (m). Doc 29 measures a departure's track from the start of roll and an arrival's from the landing threshold (section 3.3.2).

classmethod **through**(points: Sequence[tuple[float, float]], origin: tuple[float, float]) → Track

The track through `points` measured from `origin`, a point on it (the start of roll or the landing threshold); a point within `SAME_PLAN_POINT_M` of the one before it is left out.

property **node_distances**: ndarray

The distance along the track of each of its points (m).

extended(first_m: float, last_m: float) → Track

The track reaching back to `first_m` and on to `last_m` at least, its first and last legs made longer if they must be (section 3.6.6), the way the first and last that have a length go.

at(s: float) → tuple[float, float]

The point of the track `s` along it (m), on the leg that holds it, or on from the first or last leg that has a length before or after the track.

acoustic_spaces.doc29.path.**speed_between**(v1, v2, f)

The speed `f` of the distance from `v1` to `v2` (m/s), its square linear in the distance, so that the speed is linear in time (eq. 3-2); `f` a number or an array of them.

acoustic_spaces.doc29.path.**interpolate**(a: Node, b: Node, s: float) → Node

The node `s` along the way from `a` to `b`: the height and the bank angle linear in the distance, the speed and the power in its square, linear in time (eq. 3-2).

acoustic_spaces.doc29.path.**power_between**(p1: float, p2: float, f: float) → float

acoustic_spaces.doc29.path.**power_between**(p1: float, p2: float, f: ndarray) → ndarray

The power `f` of the way from `p1` to `p2` (eq. 3-2d to 3-2g), `f` a number or an array of them.

acoustic_spaces.doc29.path.**lowest_approach_node**(profile: Sequence[ProfilePoint]) → float

The distance (m, to touchdown) of the lowest node of an approach above the runway: its last profile point in the air, or the lowest height that cuts its last segment to touchdown (section 3.6.4) if one does. Volume 3's reference cases put the landing threshold there (the jets' point at 15.2 m, 290.2 m before touchdown, and the propeller aircraft's cut at 17.2 m, 328.1 m before it).

acoustic_spaces.doc29.path.**flight_path**(profile: Sequence[ProfilePoint], track: Track, departure: bool, threshold_m: float = 0.0) → list[Segment]

The segments of the flight of `profile` along `track`, a departure or an arrival. An arrival's profile measures its distances to touchdown and its track from the landing threshold: `threshold_m` is the profile's distance at the threshold (negative, before touchdown). The profile is carried on, at its last speed and power and its last slope, to the end of the track, or back to its start on arrival, and the track on to the end of the profile (section 3.6.6).

The noise of one aircraft movement at receivers by Doc 29 (Volume 2 chapter 4): each segment's level from the NPD curves at its power and distance, corrected for the speed, the engines' installation, the lateral attenuation, the segment's finite length and, behind a take-off roll, the start-of-roll directivity; summed over the segments to the event's sound exposure level, or its maximum level taken.

acoustic_spaces.doc29.noise.**NPD_ENVELOPE_DB** = 5.0

How far an NPD level extrapolated in power (eq. 4-3) may depart from the level at the nearest power of the curves before `event()` warns (dB), section 4.2.

class acoustic_spaces.doc29.noise.**Event**(sel: ndarray, lamax: ndarray, segments: tuple[**SegmentLevels**, ...])

One movement's levels at the receivers (dB(A)), and each segment's part.

exception acoustic_spaces.doc29.noise.**NpdExtrapolationWarning**

A flight's power lies so far outside its NPD curves' powers that their levels, extrapolated in power (eq. 4-3), depart more than `NPD_ENVELOPE_DB` from the levels at the nearest power of the curves (Volume 2 section 4.2).

class acoustic_spaces.doc29.noise.**SegmentLevels**(q: ndarray, slant: ndarray, d1: ndarray, d2: ndarray, lateral: ndarray, npd_distance: ndarray, power: ndarray, elevation: ndarray, climb: float, depression: ndarray, installation: ndarray, lateral_attenuation: ndarray, baseline: ndarray, duration: ndarray, maximum: ndarray, fraction: ndarray, start_of_roll: ndarray, impedance: float, exposure: ndarray, peak: ndarray)

One segment's contribution at each receiver, with the terms that make it up (arrays, one value a receiver): the distances (m), angles (degrees) and levels (dB) Volume 3 lists for its reference cases.

peak: ndarray

The segment's maximum level at each receiver (dB(A)), eq. 4-8a.

acoustic_spaces.doc29.noise.**event**(aircraft: **Aircraft**, exposure: **NpdCurves**, maximum: **NpdCurves**, segments: Sequence[**Segment**], receivers: ndarray, impedance: float = 0.0) → **Event**

The sound exposure level (eq. 4-11) and the maximum level (eq. 4-10) of the flight `segments` of `aircraft` at `receivers`. A segment whose NPD level at its power departs more than `NPD_ENVELOPE_DB` from the level at the nearest power of the curves gives an `NpdExtrapolationWarning` (section 4.2).

`impedance` is Doc 29's impedance adjustment (dB, eq. 4-6 and 4-7) added to every segment's levels: `impedance_db()` of the aerodrome's temperature and pressure. Its default, 0, is the NPD curves' reference day (25 °C, 101.325 kPa), not the standard atmosphere at sea level, for which the adjustment is 0.074 dB.

acoustic_spaces.doc29.noise.**fraction_db**(alpha1: ndarray, alpha2: ndarray) → ndarray

The finite segment correction (dB) of eq. 4-20 for the ends `alpha1` and `alpha2` of the segment in scaled distances.

acoustic_spaces.doc29.noise.**impedance_db**(temperature_c: float = 15.0, pressure_kpa: float

= 101.325) → float

The impedance adjustment of the NPD levels to the air at the aerodrome (dB), eq. 4-6 and 4-7.

acoustic_spaces.doc29.noise.**installation_db**(installation: str, depression_deg: ndarray) → ndarray

The engine installation correction (dB) at the depression angle (degrees) under the wing plane, eq. 4-15, of the installation the lateral directivity `installation` names (`installation_of()`); nought for propellers, and at negative angles that at nought.

acoustic_spaces.doc29.noise.**lateral_attenuation_db**(elevation_deg: ndarray, lateral_m: ndarray) → ndarray

The lateral attenuation (dB) at an elevation angle (degrees) and a lateral displacement from the ground track (m), eq. 4-18 and 4-19.

acoustic_spaces.doc29.noise.**segment_levels**(aircraft: Aircraft, exposure: NpdCurves, maximum: NpdCurves, segment: Segment, receivers: ndarray, impedance: float = 0.0) → SegmentLevels

The contribution of `segment` of a flight of `aircraft` at `receivers` (an array of x, y, z in m, one a row), its exposure from the SEL curves `exposure` and its maximum level from the LAmax curves `maximum`.

`impedance` is Doc 29's impedance adjustment (dB, eq. 4-6 and 4-7), added to both levels: `impedance_db()` of the aerodrome's temperature and pressure. Its default, 0, is the NPD curves' reference day (25 °C, 101.325 kPa), not the standard atmosphere at sea level, for which the adjustment is 0.074 dB.

acoustic_spaces.doc29.noise.**start_of_roll_db**(jet: bool, azimuth_deg: ndarray, distance_m: ndarray) → ndarray

The start-of-roll directivity (dB) behind a take-off roll at the azimuth from the aircraft's heading (90 to 180 degrees) and the distance (m), eq. 4-22 to 4-25.

Flight performance by Doc 29 Volume 2 Appendix B: an aircraft's fixed-point profile (its height, true airspeed and corrected net thrust along the ground track) from its procedural steps, its engine and aerodynamic coefficients and its weight, at an aerodrome's elevation, temperature and pressure and into a headwind.

As Appendix B does, the calculation runs in feet, knots and pounds, and its profile points (`Point`) are in those units: `as_profile()` gives them in the metres of `ProfilePoint`, ready for `flight_path()`.

The aircraft's coefficients are read from tables laid out as the ANP database's (Volume 2 Appendix G3): `read_performance()` reads its aircraft, jet engine, propeller engine, aerodynamic coefficients and default weights tables, and `read_departure_steps()` and `read_approach_steps()` its procedural steps.

The aircraft flies straight: the bank angles of turns (Appendix B5) are not part of the calculation.

acoustic_spaces.doc29.performance.**G_FT_S2** = 32.174

The standard gravitational acceleration (ft/s²) and a knot (ft/s), Appendix B2.

acoustic_spaces.doc29.performance.**SEA_LEVEL_INHG** = 29.92

The standard sea-level pressure (inHg) Appendix B3 keeps, and the lapse rate of the temperature (°F/ft), the standard sea-level temperature (°R) and the exponent of its pressure ratio.

acoustic_spaces.doc29.performance.**KPA_PER_INHG** = 3.38653131

An inch of mercury (kPa) as Appendix B2.2 converts it: less precise than 3.386389 on purpose, so that 29.92 inHg is the standard 101.325 kPa (its note 31).

acoustic_spaces.doc29.performance.**BREAKPOINT_C** = 30.0

The breakpoint temperature (°C) of eq. B-11, when the high-temperature jet coefficients are not given.

acoustic_spaces.doc29.performance.**ENGINE_OUT_GRADIENT** = {2: 1.2, 3: 1.5, 4: 1.7}

The engine-out climb gradients (%) of eq. B-13, by number of engines.

acoustic_spaces.doc29.performance.**TRANSITION_FT** = 1000.0

The longest transition segment (ft) where the thrust changes (B6.1.6, B7).

acoustic_spaces.doc29.performance.**REFERENCE_HEADWIND_KT** = 8.0

The headwind (kt) the coefficients are given for (Appendix G3.1).

class acoustic_spaces.doc29.performance.**Aerodrome**(elevation_ft: float = 0.0, temperature_c: float = 15.0, pressure_inhg: float = 29.92, headwind_kt: float = 8.0)

Where and in what air the aircraft flies (Appendix B3): the aerodrome's elevation (ft above mean sea level), its temperature (°C), its pressure reduced to sea level (QNH, inHg) and the headwind (kt). The defaults are the conditions of the ANP data (Appendix G3.1): a sea-level aerodrome in the standard atmosphere, into an 8 kt headwind.

theta(altitude_ft: float) → float

The temperature ratio at `altitude_ft` (ft above mean sea level; eq. B-3).

delta(altitude_ft: float) → float

The pressure ratio at `altitude_ft` (eq. B-4).

sigma(altitude_ft: float) → float

The density ratio at `altitude_ft` (eq. B-5).

pressure_altitude(altitude_ft: float) → float

The pressure altitude (ft) at `altitude_ft` (eq. B-6).

temperature(altitude_ft: float) → float

The air's temperature (°C) at `altitude_ft` (eq. B-1, B-2).

acoustic_spaces.doc29.performance.**STANDARD** = Aerodrome(elevation_ft=0.0, temperature_c=15.0, pressure_inhg=29.92, headwind_kt=8.0)

The conditions the approach procedures' idle steps are given for (Appendix B7.1.2, B7.1.4): the standard atmosphere at a sea-level aerodrome.

class acoustic_spaces.doc29.performance.**JetCoefficients**(e: float, f: float, ga: float, gb: float, h: float)

A thrust rating's jet engine coefficients (Appendix G3.2): E (lb), F (lb/kt), GA (lb/ft), GB (lb/ft²) and H (lb/°C) of eq. B-9.

thrust(calibrated_kt: float, pressure_altitude_ft: float, temperature_c: float) → float

The corrected net thrust per engine (lb; eq. B-9, B-10).

class acoustic_spaces.doc29.performance.**PropellerCoefficients**(efficiency: float, power_hp: float)

A thrust rating's propeller coefficients (Appendix G3.3): the propeller's efficiency and the installed net propulsive power (hp).

class acoustic_spaces.doc29.performance.**Flaps**(b: float | None, speed: float | None, r: float)

A flap setting's aerodynamic coefficients (Appendix G3.4): B (ft/lb), the take-off ground roll's; C or D (kt/√lb), the take-off's or the landing's speed; and R, the ratio of drag to lift. B, C and D are given only for the flaps of take-off and landing (None for the others).

class acoustic_spaces.doc29.performance.**Performance**(name: str, engine_type: str, engines: int, max_landing_weight_lb: float, max_static_thrust_lb: float, jet: dict[str, ~acoustic_spaces.doc29.performance.JetCoefficients] = <factory>, propeller: dict[str, ~acoustic_spaces.doc29.performance.PropellerCoefficients] = <factory>, flaps: dict[tuple[str, str], ~acoustic_spaces.doc29.performance.Flaps] = <factory>, weights: dict[tuple[str, str], float] = <factory>)

What the performance calculation needs of an aircraft (Appendix G2 to G3.5).

max_landing_weight_lb: float

Its greatest landing weight and its engines' greatest static thrust at sea level, each (lb).

jet: dict[str, **JetCoefficients**]

Its jet engine coefficients and its propeller coefficients, by the key of the thrust rating (`rating_key()`): "maxtakeoff", "maxclimb", "idleapproach", and those with the suffix "hitemp", above the breakpoint temperature.

flaps: dict[tuple[str, str], **Flaps**]

Its flaps' coefficients, by (operation "A" or "D", flap identifier).

weights: dict[tuple[str, str], float]

Its default weights (lb), by (operation "A" or "D", stage length).

flap(operation: str, name: str) → **Flaps**

The coefficients of the flap setting `name` on arrival ("A") or departure ("D").

acoustic_spaces.doc29.performance.**rating_key**(name: str) → str

The key of the thrust rating `name`: its letters and digits in lower case, so that "MaxTakeoff", "MaxTakeOff" and "MaxTake-off" are one, with "HighTemp" written "hitemp".

class acoustic_spaces.doc29.performance.**Point**(distance_ft: float, height_ft: float,

```
    speed_kt: float, thrust_lb: float)
```

A point of a fixed-point profile as Appendix B gives it: the distance along the ground track (ft; from the start of roll on departure, to touchdown on approach, negative before it), the height above the aerodrome (ft), the true airspeed (kt) and the corrected net thrust per engine (lb).

```
acoustic_spaces.doc29.performance.as_profile(points: Sequence[Point]) →
    tuple[ProfilePoint, ...]
```

`points` as the profile points (m, m/s) of a flight path.

```
class acoustic_spaces.doc29.performance.DepartureStep(kind: str, thrust: str, flap: str,
    altitude_ft: float | None = None, climb_rate_ft_min: float | None = None, speed_kt:
    float | None = None, accelerating_percent: float | None = None, distance_ft: float |
    None = None)
```

A step of a departure procedure (Appendix G3.6): its type (“Takeoff”, “Climb”, “Accelerate”, “Level” or “Level-Accelerate”), its thrust rating, its flaps, the height it climbs to (ft above the aerodrome; a climb’s), its rate of climb (ft/min) or its share of the thrust that accelerates (%, its accel percentage) and the calibrated airspeed it accelerates to (kt; an acceleration’s), and its length (ft; a level step’s).

```
class acoustic_spaces.doc29.performance.ApproachStep(kind: str, flap: str, altitude_ft:
    float | None = None, speed_kt: float | None = None, angle_deg: float | None = None,
    touchdown_roll_ft: float | None = None, distance_ft: float | None = None,
    thrust_percent: float | None = None)
```

A step of an approach procedure (Appendix G3.7): its type (“Descend”, “Descend-Decel”, “Descend-Idle”, “Level”, “Level-Decel”, “Level-Idle”, “Land” or “Decelerate”), its flaps, the height (ft above the aerodrome) and calibrated airspeed (kt) it starts at, its descent angle (°), the touchdown roll (ft; a landing’s), its length (ft) and the thrust it starts with (% of the greatest static thrust; a deceleration’s on the runway).

```
acoustic_spaces.doc29.performance.read_performance(aircraft: str | Path, jet: str | Path
    | None = None, propeller: str | Path | None = None, aerodynamic: str | Path | None =
    None, weights: str | Path | None = None) → dict[str, Performance]
```

The aircraft of the aircraft table `aircraft` with their coefficients from the jet engine coefficients table `jet`, the propeller engine coefficients table `propeller`, the aerodynamic coefficients table `aerodynamic` and the default weights table `weights` (each optional), by identifier.

```
acoustic_spaces.doc29.performance.read_departure_steps(path: str | Path) →
    dict[tuple[str, str, str], tuple[DepartureStep, ...]]
```

The departure procedures of the departure procedural steps table `path`, by (aircraft, profile identifier, stage length), each its steps in order.

```
acoustic_spaces.doc29.performance.read_approach_steps(path: str | Path) → dict[tuple[str,
    str], tuple[ApproachStep, ...]]
```

The approach procedures of the approach procedural steps table `path`, by (aircraft, profile identifier), each its steps in order.

```
acoustic_spaces.doc29.performance.departure(aircraft: Performance, steps:
    Sequence[DepartureStep], weight_lb: float, aerodrome: Aerodrome =
```

```
Aerodrome(elevation_ft=0.0, temperature_c=15.0, pressure_inhg=29.92,
headwind_kt=8.0), runway_gradient: float = 0.0) → list[Point]
```

The fixed-point profile (Appendix B6) of a departure of `aircraft` at `weight_lb` (lb) by the procedure `steps` from `aerodrome`, its runway's gradient `runway_gradient` (positive uphill).

```
acoustic_spaces.doc29.performance.arrival(aircraft: Performance, steps:
Sequence[ApproachStep], weight_lb: float | None = None, aerodrome: Aerodrome =
Aerodrome(elevation_ft=0.0, temperature_c=15.0, pressure_inhg=29.92,
headwind_kt=8.0)) → list[Point]
```

The fixed-point profile (Appendix B7) of an arrival of `aircraft` at `weight_lb` (lb; by default 90 % of its greatest landing weight, as Appendix B7 takes) by the procedure `steps` at `aerodrome`.

The reference cases of Doc 29 Volume 3 (5th edition, 2026), of three hypothetical aircraft that rely on no ANP data: a rear-engined jet (JETF), a wing-engined jet (JETW) and a propeller aircraft (PROP).

- Part 1, the noise: each aircraft arriving and departing on a straight and a curved route at a hypothetical aerodrome (`CASES`), and the sound exposure levels Volume 3 gives at its receptors (its Appendix A and B, tables A-1 and A-6 to A-12 and B-1, as its workbook gives them).
- Part 2, the flight performance: JETW flying departure and approach procedures and PROP departure procedures (Volume 3 gives no arrivals of PROP) at aerodromes of other elevations, temperatures, pressures and headwinds (`profile_cases()`), and the fixed-point profiles Volume 3 gives for them (its Appendix C and D, tables C-1 to C-8 and D-1 and D-2). The approach procedures are as its table C-6.2 prints them, to 0.1 ft, kt and degree, which its profiles are computed from (its workbook has more digits); the lengths of its Level-Idle steps, which the table gives in the column of the descent angle, are their distances.

The tables are read as the ANP database's are.

```
acoustic_spaces.doc29.volume3.DATA =
PosixPath('/opt/hostedtoolcache/Python/3.12.16/x64/lib/python3.12/site-
packages/acoustic_spaces/doc29/volume3')
```

Where the tables are.

```
acoustic_spaces.doc29.volume3.aircraft() → dict[str, Aircraft]
```

The three reference aircraft (table A-1), by identifier.

```
acoustic_spaces.doc29.volume3.npd() → dict[tuple[str, str, str], NpdCurves]
```

Their NPD curves (table A-7), by (aircraft, "SEL" or "LAmax", "A" or "D").

```
acoustic_spaces.doc29.volume3.spectral_classes() → dict[str, SpectralClass]
```

Their spectral classes (table A-8), by identifier.

```
acoustic_spaces.doc29.volume3.profiles() → dict[tuple[str, str], tuple[ProfilePoint,
...]]
```

Their fixed-point profiles (table A-6), by (aircraft, "A" or "D").

```
acoustic_spaces.doc29.volume3.routes() → dict[str, tuple[tuple[float, float], ...]]
```

The four routes' ground tracks (table A-11; m), by identifier: "AC" and "AS" the curved and straight arrivals, "DC" and "DS" the departures.

acoustic_spaces.doc29.volume3.**runway**() → tuple[tuple[float, float], tuple[float, float]]

The runway's start of roll and end (table A-10; m).

acoustic_spaces.doc29.volume3.**receptors**() → dict[str, tuple[float, float, float]]

The 18 receptors (table A-12; m), by identifier.

acoustic_spaces.doc29.volume3.**sel_results**() → dict[tuple[str, str], float]

Volume 3's sound exposure levels (table B-1; dB(A)), by (case, receptor).

acoustic_spaces.doc29.volume3.**impedance**() → float

The impedance adjustment at the aerodrome's temperature and pressure (dB; table A-9, the standard atmosphere's: 0.074 dB).

class acoustic_spaces.doc29.volume3.**Case**(aircraft: str, operation: str, route: str)

A reference case: an aircraft, an arrival ("A") or a departure ("D"), and a route, curved ("C") or straight ("S"); its identifier is Volume 3's, "JETFDC" for instance.

classmethod **named**(name: str) → Case

The case of Volume 3's identifier `name`.

track() → Track

Its ground track: a departure's from the start of roll, an arrival's from the landing threshold, the runway's start (where the arrivals land, as the departures leave).

segments() → list[Segment]

Its flight path's segments.

event(points: ndarray) → Event

Its levels at `points` (x, y, z in m, one a row).

acoustic_spaces.doc29.volume3.**CASES** = (Case(aircraft='JETF', operation='A', route='C'), Case(aircraft='JETF', operation='A', route='S'), Case(aircraft='JETF', operation='D', route='C'), Case(aircraft='JETF', operation='D', route='S'), Case(aircraft='JETW', operation='A', route='C'), Case(aircraft='JETW', operation='A', route='S'), Case(aircraft='JETW', operation='D', route='C'), Case(aircraft='JETW', operation='D', route='S'), Case(aircraft='PROP', operation='A', route='C'), Case(aircraft='PROP', operation='A', route='S'), Case(aircraft='PROP', operation='D', route='C'), Case(aircraft='PROP', operation='D', route='S'))

The twelve cases, as Volume 3's table 3-1 lists them.

acoustic_spaces.doc29.volume3.**performance**() → dict[str, Performance]

Part 2's two aircraft, JETW and PROP, with their engine and aerodynamic coefficients and their default weights (tables C-1 to C-5), by identifier.

```
acoustic_spaces.doc29.volume3.departure_procedures() → dict[tuple[str, str],
    tuple[DepartureStep, ...]]
```

Part 2's departure procedures (table C-6.1), by (aircraft, procedure).

```
acoustic_spaces.doc29.volume3.approach_procedures() → dict[tuple[str, str],
    tuple[ApproachStep, ...]]
```

Part 2's approach procedures (table C-6.2), by (aircraft, procedure).

```
class acoustic_spaces.doc29.volume3.ProfileCase(number: str, operation: str, aircraft:
    str, procedure: str, aerodrome: Aerodrome)
```

A reference case of Part 2 (tables C-7 and C-8): an aircraft flying a departure ("D") or approach ("A") procedure from an aerodrome; its number is Volume 3's, "2A" or "54" for instance.

```
points() → list[Point]
```

Its fixed-point profile: a departure at the aircraft's default take-off weight (table C-5), an arrival at 90 % of its greatest landing weight.

```
acoustic_spaces.doc29.volume3.profile_cases() → tuple[ProfileCase, ...]
```

Part 2's 26 cases: nine arrivals (table C-7) and seventeen departures (C-8).

```
acoustic_spaces.doc29.volume3.reference_profiles() → dict[str, tuple[Point, ...]]
```

Volume 3's fixed-point profiles of Part 2's cases (tables D-1 and D-2, to 0.1 ft, kt and lb), by case number.

Doc 29 with the aircraft of a copy of the ANP database (Volume 2 Appendix G), the international Aircraft Noise and Performance database, that the user has downloaded from EASA under its terms.

This package ships no ANP data: the database's terms (EASA's ANP Database Terms and Conditions, issue 3, 2024) allow its use only for noise contours of an average fleet and traffic and forbid passing it on, so each user downloads their own copy and accepts those terms. `read_database()` reads the copy's tables from a folder, by the names of their files: "Aircraft", "NPD", "Spectral classes", "Fixed point profiles", "Jet engine coefficients", "Propeller engine coefficients", "Aerodynamic coefficients", "Default weights", "Departure procedural steps" and "Approach procedural steps", in any case, with spaces or underscores, as CSV or text files separated by commas or semicolons.

The readers take the fields Volume 2 Appendix G lists, as Volume 3's workbook lays them out for its hypothetical aircraft; they have not been tried on a download of the database.

```
acoustic_spaces.doc29.anp.TABLES = {'aerodynamic': ('aerodynamic',), 'aircraft':
    ('aircraft',), 'approach_steps': ('approachprocedur', 'approachstep'),
    'departure_steps': ('departureprocedur', 'departurestep'), 'fixed_point_profiles':
    ('fixedpoint',), 'jet': ('jetengine', 'jetcoef'), 'npd': ('npd',), 'propeller':
    ('propeller',), 'spectral_classes': ('spectral',), 'weights': ('weight',)}
```

The tables, each with the words its file's name holds (its case, spaces and punctuation aside), the first that matches taken.

```
acoustic_spaces.doc29.anp.find_tables(folder: str | Path) → dict[str, Path]
```

The files of the ANP tables in `folder` and its subfolders, by the keys of `TABLES`; a table without its file is left out.

```
class acoustic_spaces.doc29.anp.Database(aircraft: dict[str, Aircraft], npd:
    dict[tuple[str, str, str], NpdCurves], spectral_classes: dict[str, SpectralClass],
    fixed_point_profiles: dict[tuple[str, str, str, str], tuple[ProfilePoint, ...]],
    performance: dict[str, Performance], departure_procedures: dict[tuple[str, str,
    str], tuple[DepartureStep, ...]], approach_procedures: dict[tuple[str, str],
    tuple[ApproachStep, ...]])
```

The tables of an ANP database (Volume 2 Appendix G), as the readers of `tables` and `performance` give them.

```
curves(aircraft: str, operation: str) → tuple[NpdCurves, NpdCurves]
```

The SEL and LAmax curves of `aircraft` on arrival (“A”) or departure (“D”).

```
profile(aircraft: str, operation: str, profile: str, stage: str = '1', aerodrome:
    Aerodrome = Aerodrome(elevation_ft=0.0, temperature_c=15.0, pressure_inhg=29.92,
    headwind_kt=8.0)) → tuple[ProfilePoint, ...]
```

The profile `profile` of `aircraft` on arrival (“A”) or departure (“D”), in metres: from its procedural steps where the database has them and the aircraft’s coefficients (a departure at the default weight of its stage length `stage`), else its fixed-point profile. Procedural steps give the corrected net thrust in lb (Appendix B): for an aircraft whose NPD curves’ power parameter is not in lb (`Aircraft.power_in_lb`) they are a ValueError.

```
acoustic_spaces.doc29.anp.read_database(folder: str | Path) → Database
```

The ANP database whose tables are the files in `folder` (`find_tables()`); the aircraft table is needed, the others are empty when their files are missing.

An aircraft’s NPD curves adjusted to the air at the aerodrome (Doc 29 Volume 2 Appendix D).

The NPD curves and the spectral classes of the ANP database are normalised to the “AIR-1845 atmosphere”: the average attenuation rates of SAE AIR-1845, per third octave (`AIR_1845_DB_PER_M`, Volume 2 table D-1), a notional atmosphere of no one temperature and humidity. Appendix D adjusts the curves to another one in three steps:

1. the aircraft’s spectral class (its spectrum 1000 ft under it at the time of the maximum level) is taken back to the source, the AIR-1845 attenuation over 1000 ft added back (eq. D-1);
2. it is taken out to each of the ten NPD distances by spherical spreading and the attenuation of each atmosphere, AIR-1845’s and the aerodrome’s (eq. D-2, D-3);
3. at each distance, both spectra are A-weighted and summed, and the difference between them, `increments_db()` (eq. D-4), is added to the NPD curves’ levels (`adjusted()`), the maximum levels’ and the sound exposure levels’ alike.

The aerodrome's attenuation is ISO 9613-1's (`acoustic_spaces.air.absorption_db_per_m()`), the absorption of SAE ARP-5534, the method Appendix D recommends, averaged over each third octave as a sound of flat spectrum through it loses it (`band_attenuation_db()`): over a long path a band's upper frequencies are lost first, so its attenuation per metre falls as the path grows longer, as ARP-5534's does. It gives Volume 2's worked example (JETW's spectral classes at 10 °C, 80 % relative humidity and 101.325 kPa, its tables D-3c and D-4) within 0.1 dB, the tables' rounding.

```
acoustic_spaces.doc29.atmosphere.AIR_1845_DB_PER_M: tuple[float, ...] = (0.00033,
    0.00033, 0.00033, 0.00066, 0.00066, 0.00098, 0.00131, 0.00131, 0.00197, 0.0023,
    0.00295, 0.00361, 0.00459, 0.0059, 0.00754, 0.00983, 0.01311, 0.01705,
    0.022949999999999998, 0.03115, 0.036070000000000005, 0.05246000000000001, 0.07213,
    0.09836)
```

The AIR-1845 atmosphere's attenuation rates (dB/m) per third octave of the spectral classes, 50 Hz to 10 kHz (Volume 2 table D-1, there in dB/100 m).

```
acoustic_spaces.doc29.atmosphere.A_WEIGHTING_DB: tuple[float, ...] = (-30.2, -26.2,
    -22.5, -19.1, -16.1, -13.4, -10.9, -8.6, -6.6, -4.8, -3.2, -1.9, -0.8, 0.0, 0.6,
    1.0, 1.2, 1.3, 1.2, 1.0, 0.5, -0.1, -1.1, -2.5)
```

The A-weighting (dB) of the third octaves 50 Hz to 10 kHz (IEC 61672-1).

```
acoustic_spaces.doc29.atmosphere.REFERENCE_M = 304.8
```

The spectral classes' distance (m), 1000 ft.

```
acoustic_spaces.doc29.atmosphere.adjusted(curves: NpdCurves, spectral_class:
    SpectralClass, temperature_c: float = 15.0, humidity_percent: float = 70.0,
    pressure_kpa: float = 101.325) → NpdCurves
```

`curves` (an aircraft's NPD curves of one descriptor and operation) in the air at `temperature_c`, `humidity_percent` and `pressure_kpa`: each level plus `increments_db()` of its distance, for `spectral_class`, the aircraft's of that operation.

```
acoustic_spaces.doc29.atmosphere.band_attenuation_db(distance_m: float, temperature_c:
    float = 15.0, humidity_percent: float = 70.0, pressure_kpa: float = 101.325) →
    ndarray
```

The attenuation (dB) of each third octave, 50 Hz to 10 kHz, over `distance_m` of air at `temperature_c`, `humidity_percent` and `pressure_kpa`: ISO 9613-1's absorption at each frequency through the band (its edges a tenth of a decade apart), the band's energy of a flat spectrum averaged.

```
acoustic_spaces.doc29.atmosphere.increments_db(spectral_class: SpectralClass,
    temperature_c: float = 15.0, humidity_percent: float = 70.0, pressure_kpa: float =
    101.325) → ndarray
```

What the air at `temperature_c`, `humidity_percent` and `pressure_kpa` adds to the NPD levels at each of `NPD_FEET` against the AIR-1845 atmosphere's (dB, eq. D-1 to D-4), for an aircraft of `spectral_class`.

An aircraft of Doc 29 as a source of a scene: the sound power and the directivity that give its NPD curves' levels in the engine, and its flight as a walk's keyframes.

Doc 29 gives an aircraft's sound as levels at receivers: its NPD curves (the maximum level and the sound exposure level at distances from 200 to 25,000 ft, at each power, in the AIR-1845 atmosphere, at 160 kt) and its spectral class (the spectrum 1000 ft under it at the time of its maximum level). The NPD curves are measured 1.2 m above flat soft ground (Volume 2 sections 2.5 and 4.5.4), so they hold that ground's reflection. A point source of the scene that sends the same sound gets:

- **its spectrum** from the spectral class, scaled to the NPD curves' maximum level 1000 ft under the aircraft at its power, raised by what a sound level meter's slow time weighting takes off it (`sound_power_db()`): Doc 29 reads its levels through that weighting (Volume 2's glossary; IEC 61672-1's S, a time constant of 1 s), which lags behind the passing aircraft, where the scene's source sends its sound with none. On the NPD curves' own pass (`slow_drop_db()`) it takes 0.29 dB off JETW's loudest moment, at 160 kt in air at 15 °C and 70 % humidity;
- **its level** back at the source: the AIR-1845 atmosphere's attenuation over the 1000 ft added back (Appendix D eq. D-1), and the soft ground's reflection taken out (`ground_gain_db()`): grass (Nord2000's class D, 200 kPa s/m², Miki's impedance) at normal incidence, its reflection's energy added to the direct sound's, as no coherent dip shows in the spectral classes (the first would be near 70 Hz);
- **its directivity** from the engines' installation (eq. 4-15, `installation_db()`), the depression angle under the wing plane measured across the flight path, the same in every band (`directivity()`); nought straight down, where the NPD curves are measured, so the scene's sound power is the power below that much over (`below_offset_db()`).

Doc 29 gives the NPD curves no directivity along the flight path but behind a take-off roll (eq. 4-22), which the scene's source does not hold, nor the impedance adjustment (eq. 4-6): 0.074 dB at sea level in the standard atmosphere, -0.24 dB 2000 ft up at 15 °C and -0.83 dB 5000 ft up at 30 °C. The NPD curves at the powers of a flight differ by nearly the same at every distance: the aircraft's sound power changes with its power as the NPD curves' maximum level 1000 ft away does (`power_change_db()`), its spectrum kept.

`flight()` turns a flight path's segments (`flight_path()`) into keyframes, when the aircraft is where: along a segment its speed's square changes evenly with the distance (a constant acceleration, as Doc 29 interpolates it), cut into legs whose speeds differ by at most `LEG_SPEED_STEP_M_S`, each leg timed for that acceleration. The aircraft faces the way it flies, along each segment's heading and climb, and turns to the next one's over its first `TURN_S`.

`acoustic_spaces.doc29.source.`**`LEG_SPEED_STEP_M_S`** `= 5.0`

How much the speed may change along one leg of a flight's keyframes (m/s).

`acoustic_spaces.doc29.source.`**`MICROPHONE_M`** `= 1.2`

How high the NPD curves' microphone is over the ground (m), Volume 2 sections 2.5 and 4.5.4, and the flow resistivity of its soft ground (kPa s/m²), taken to be grass (Nord2000's class D, `acoustic_spaces.materials.ground.PRESETS`).

`acoustic_spaces.doc29.source.`**`SHORTEST_M`** `= 0.001`

A segment shorter than this (m) is passed over: where the track's nodes and the profile's points nearly meet (the walk's own keyframes as a track), the flight path has segments a few micrometres long, flown in microseconds.

`acoustic_spaces.doc29.source.`**`TURN_S`** `= 1.0`

How long the aircraft takes to turn to a segment's heading and climb (s).

class acoustic_spaces.doc29.source.**Flight**(times: tuple[float, ...], positions: tuple[tuple[float, float, float], ...], orientations: tuple[tuple[float, float, float], ...], speeds: tuple[float, ...], powers: tuple[float, ...])

A flight as a walk's keyframes: when (s from its start) the aircraft is where (x, y, z in m), how it faces there (yaw, pitch and roll in degrees, as a scene's orientation), and its speed (m/s) and power there. It flies at one speed along each leg, from one keyframe to the next, and its power changes along the leg as Doc 29's does along a segment (eq. 3-2).

power_at(times) → ndarray

The power at `times` (s), each in its leg (eq. 3-2, the share of the leg's length the share of its time, at one speed); the first's before the start, the last's after the end.

keyframes(digits: int = 4) → list[dict]

The keyframes after the first, as a scene's walk takes them (the first is the source's own position and orientation), rounded to `digits`.

acoustic_spaces.doc29.source.**below_offset_db**(installation: str) → float

How far the sound power of a source with the installation's `directivity()` is over the power of one that sends each way what it sends straight down (dB), where the installation correction is nought: 10 lg of its energy's mean over the sphere, as the balloon is scaled to average 1.

acoustic_spaces.doc29.source.**depression_deg**(directions) → ndarray

(n,) the depression angle (degrees) under the wing plane of `directions` (n, 3) in the aircraft's frame (x forward, y left, z up), measured across the flight path: `atan2(-z, |y|)`, negative above the wing plane; nought straight ahead and behind.

acoustic_spaces.doc29.source.**directivity**(installation: str, name: str = '')

The directivity (a `Balloon` of octave bands) of the engines' `installation` ("Wing", "Fuselage" or a propeller's, which has none): the installation correction each way (eq. 4-15) at its depression angle (`depression_deg()`), the same in every band, over its mean on the sphere.

acoustic_spaces.doc29.source.**flight**(segments: Sequence[Segment], step_m_s: float = 5.0) → Flight

The keyframes of a flight along `segments`, in order, from the first's start at 0 s (see the module): each segment at a constant acceleration, its speed's square changing evenly with its length, cut into legs whose speeds differ by at most `step_m_s`; each leg's time is its length over the mean of its ends' speeds.

acoustic_spaces.doc29.source.**ground_gain_db**(temperature_c: float = 15.0) → ndarray

(24,) what the soft ground under the NPD curves' microphone adds to the sound of an aircraft 1000 ft from it straight above, per third octave of the spectral classes, 50 Hz to 10 kHz (dB): the reflection's energy added to the direct sound's, `1 + |Q|² (r1 / r2)²`, averaged over the band, Q the reflection factor of a point source's sound off grass at normal incidence (`spherical_reflection()`), r1 the 1000 ft and r2 the image's path, 2.4 m longer, in air at `temperature_c`.

```
acoustic_spaces.doc29.source.power_change_db(maximum: NpdCurves, power, reference: float)
    → ndarray
```

How much louder the aircraft is at `power` (a number or an array) than at `reference` (dB): the maximum levels of its NPD curves `maximum` 1000 ft away.

```
acoustic_spaces.doc29.source.profile_between(profile: Sequence[ProfilePoint], first_m:
    float, last_m: float) → tuple[ProfilePoint, ...]
```

The part of `profile` from `first_m` to `last_m` along its ground track (m, as its distances go), its ends interpolated as Doc 29 does between two points (the height linear in the distance, the speed and the power in its square, eq. 3-2); a profile shorter than that is kept to its own ends.

```
acoustic_spaces.doc29.source.slow_drop_db(spectral_class: SpectralClass, temperature_c:
    float = 15.0, humidity_percent: float = 70.0, pressure_kpa: float = 101.325) → float
```

How far (dB) a sound level meter's slow time weighting lowers the loudest moment of the NPD curves' own pass, which their maximum level is read with (Volume 2's glossary: L_A(t) and L_Amax are measured with slow time weighting): a point source with the spectrum of `spectral_class` flying straight and level at 160 kt (`REFERENCE_SPEED_M_S`) 1000 ft right over the microphone, in a free field of air at `temperature_c`, `humidity_percent` and `pressure_kpa`, each third octave spreading spherically and losing the air's attenuation over its path beyond the 1000 ft (`band_attenuation_db()` over 1000 ft, per metre), heard when its sound arrives; its A-weighted level as it passes over, with no time weighting, less the highest through IEC 61672-1's slow time weighting (`acoustic_spaces.parameters.time_weighted()`).

```
acoustic_spaces.doc29.source.sound_power_db(maximum: NpdCurves, spectral_class:
    SpectralClass, power: float, temperature_c: float = 15.0, humidity_percent: float =
    70.0, pressure_kpa: float = 101.325) → ndarray
```

(24,) the sound power (dB re 1 pW) per third octave, 50 Hz to 10 kHz, of a point source that sends each way what the aircraft sends straight down at `power`: in a free field of the AIR-1845 atmosphere, 1000 ft from it, the spectrum of `spectral_class` at the maximum level of its NPD curves `maximum` there, raised by how far the slow time weighting the curves are read with lowers it (`slow_drop_db()`, in the scene's air at `temperature_c`, `humidity_percent` and `pressure_kpa`), less the NPD's soft ground (`ground_gain_db()`).

Ground-borne vibration

Ground-borne vibration from trains, trams and construction plant, and the rumble it makes in a building, by the US Federal Transit Administration's general method.

FTA, *Transit Noise and Vibration Impact Assessment Manual*, FTA Report No. 0123 (September 2018): the General Vibration Assessment of its section 6.4 for trains and trams (`ground_level()`, `indoor_db()`, `NOISE_DB`), the construction vibration of its section 7.2 for plant (`plant_ppv()`, `plant_level()`), and the criteria of its tables 6-2 to 6-4 and 7-5.

A vibration level here is the manual's: the rms vibration velocity in VdB, decibels re 1 micro-inch per second (25.4 nm/s). Distances are in metres, the manual's feet converted, and speeds in km/h. The method gives the overall level at a building and the A-weighted level of the sound its room surfaces radiate, not their spectra. The manual reads its base curves as "the upper range of vibration levels from well-maintained systems", which ground-borne vibration rarely exceeds "by more than 1 or 2 dB unless there are extenuating circumstances such as wheel- or running-surface defects" (section 6.4, step 3).

```
acoustic_spaces.vibration.BUILDINGS: dict[str, str] = {'large': 'a large building, its
    structure not known', 'large_masonry_footings': 'a large masonry building on spread
    footings', 'large_masonry_piles': 'a large masonry building on piles',
    'light_wood_frame': 'a light wood-frame house, its structure not known',
    'masonry_1_2': 'a masonry building of one or two storeys', 'masonry_3_4': 'a masonry
    building of three or four storeys', 'on_rock': 'a building founded on rock',
    'small_masonry': 'a small masonry building, its structure not known', 'wood_frame':
    'a wood-frame house'}
```

What each building of `COUPLINGS` and `GENERIC` is, in words.

```
acoustic_spaces.vibration.COUPLINGS: dict[str, float] = {'large_masonry_footings': -13.0,
    'large_masonry_piles': -10.0, 'masonry_1_2': -7.0, 'masonry_3_4': -10.0, 'on_rock':
    0.0, 'wood_frame': -5.0}
```

Table 6-12, the coupling loss of a building's foundation to the ground (dB).

```
acoustic_spaces.vibration.CRITERIA: dict[int, dict[str, Criterion]] = {1: {'frequent':
    Criterion(vibration_vdb=65.0, noise_dba=None), 'infrequent':
    Criterion(vibration_vdb=65.0, noise_dba=None), 'occasional':
    Criterion(vibration_vdb=65.0, noise_dba=None)}, 2: {'frequent':
    Criterion(vibration_vdb=72.0, noise_dba=35.0), 'infrequent':
    Criterion(vibration_vdb=80.0, noise_dba=43.0), 'occasional':
    Criterion(vibration_vdb=75.0, noise_dba=38.0)}, 3: {'frequent':
    Criterion(vibration_vdb=75.0, noise_dba=40.0), 'infrequent':
    Criterion(vibration_vdb=83.0, noise_dba=48.0), 'occasional':
    Criterion(vibration_vdb=78.0, noise_dba=43.0)}}
```

Table 6-3, by land use: 1 buildings where vibration would interfere with what is done inside (moderately sensitive equipment such as optical microscopes), 2 residences and buildings where people normally sleep, 3 institutions used mostly by day; and by events (`EVENT_KINDS`).

```
acoustic_spaces.vibration.CURVES: dict[str, Curve] = {'locomotive':
    Curve(what='locomotive-powered passenger or freight trains', coefficients=(92.28,
    14.81, -14.17, 1.65), reference_kmh=80.4672), 'rapid_transit': Curve(what='rapid
    transit or light rail vehicles', coefficients=(85.88, -1.06, -2.32, -0.87),
    reference_kmh=80.4672), 'rubber_tyred': Curve(what='rubber-tyred vehicles',
    coefficients=(66.08, 34.28, -30.25, 5.4), reference_kmh=48.28032)}
```

Table 6-9 and 6-10: the three base curves. The locomotive-powered curve is for trains hauled by diesel or electric locomotives (intercity, commuter and freight trains), the rapid transit curve for heavy and light rail vehicles, electric multiple units and streetcars, the rubber-tyred curve for a bus on a smooth road.

```
acoustic_spaces.vibration.DAMAGE: dict[str, Damage] = {'I': Damage(what='reinforced-
    concrete, steel or timber (no plaster)', ppv_in_s=0.5, level_vdb=102.0), 'II':
```

```
Damage(what='engineered concrete and masonry (no plaster)', ppv_in_s=0.3,
level_vdb=98.0), 'III': Damage(what='non-engineered timber and masonry buildings',
ppv_in_s=0.2, level_vdb=94.0), 'IV': Damage(what='buildings extremely susceptible to
vibration damage', ppv_in_s=0.12, level_vdb=90.0)}
```

Table 7-5: construction vibration damage criteria.

acoustic_spaces.vibration.EVENT_KINDS = ('frequent', 'occasional', 'infrequent')

Table 6-2's kinds of events (`events()`).

acoustic_spaces.vibration.FARTHEST_M = 91.44

How near and how far from the track the manual draws its base curves (figure 6-4): 10 to 300 ft from the track's centreline (m), the diagonal distance for a subway.

acoustic_spaces.vibration.FREIGHT_WAGONS_DB = -3.0

Section 6.2, step 3c: a freight train's locomotives count as infrequent events and its wagons, which pass for minutes, as frequent ones, the locomotives "often" 3 to 8 dB higher; the wagons are taken FREIGHT_WAGONS_DB under the curve, the least of that.

acoustic_spaces.vibration.GENERIC: dict[str, float] = {'large': 0.0, 'light_wood_frame': 3.0, 'small_masonry': 3.0}

Section 6.4, step 2c: where a building's structure is not known, the manual's conservative change from the vibration outdoors to indoors (dB), instead of its coupling loss, floors and amplification: a light wood-frame house's on its first floor and on its second and third, a small masonry building's and a large building's.

acoustic_spaces.vibration.NEAREST_M = 3.048

How near and how far from the track the manual draws its base curves (figure 6-4): 10 to 300 ft from the track's centreline (m), the diagonal distance for a subway.

acoustic_spaces.vibration.NOISE_DB: dict[str, float] = {'high': -20.0, 'low': -50.0, 'mid': -35.0}

Table 6-14: the A-weighted sound level radiated by a room's surfaces (dB(A)) against their vibration level, by the frequency where the ground's vibration peaks: below 30 Hz (most surface track, subways in sandy soil, track on vibration isolation), 30 to 60 Hz (most subways, surface track on very stiff clayey soil) or above 60 Hz (subways founded in rock or very stiff clayey soil).

acoustic_spaces.vibration.PERCEPTION_VDB = 65.0

Section 5.2: "the threshold of perception for humans is approximately 65 VdB".

acoustic_spaces.vibration.PLANT: dict[str, Plant] = {'caisson_drilling': Plant(what='caisson drilling', ppv_in_s=0.089, level_vdb=87.0), 'clam_shovel': Plant(what='a clam shovel drop (slurry wall)', ppv_in_s=0.202, level_vdb=94.0), 'hoe_ram': Plant(what='a hoe ram (a breaker on an excavator)', ppv_in_s=0.089, level_vdb=87.0), 'hydromill_rock': Plant(what='a hydromill (slurry wall) in rock', ppv_in_s=0.017, level_vdb=75.0), 'hydromill_soil': Plant(what='a hydromill (slurry wall) in soil', ppv_in_s=0.008, level_vdb=66.0), 'jackhammer': Plant(what='a jackhammer', ppv_in_s=0.035, level_vdb=79.0), 'large_bulldozer': Plant(what='a large bulldozer', ppv_in_s=0.089, level_vdb=87.0), 'loaded_trucks': Plant(what='loaded

```
    trucks', ppv_in_s=0.076, level_vdb=86.0), 'pile_driver_impact': Plant(what='an
    impact pile driver, typical', ppv_in_s=0.644, level_vdb=104.0),
    'pile_driver_impact_upper': Plant(what='an impact pile driver, upper range',
    ppv_in_s=1.518, level_vdb=112.0), 'pile_driver_sonic': Plant(what='a sonic pile
    driver, typical', ppv_in_s=0.17, level_vdb=93.0), 'pile_driver_sonic_upper':
    Plant(what='a sonic pile driver, upper range', ppv_in_s=0.734, level_vdb=105.0),
    'small_bulldozer': Plant(what='a small bulldozer', ppv_in_s=0.003, level_vdb=58.0),
    'vibratory_roller': Plant(what='a vibratory roller', ppv_in_s=0.21, level_vdb=94.0)}
```

Table 7-4: averages over many sites and soils.

```
acoustic_spaces.vibration.RUMBLE_HZ: dict[str, float] = {'high': 125.0, 'low': 25.0,
    'mid': 50.0}
```

Where each peak of `NOISE_DB` puts the rumble (Hz): the third octave whose A-weighting is the table's adjustment less `UNWEIGHTED_DB` to within 1.1 dB (25 Hz -44.7 dB, 50 Hz -30.2, 125 Hz -16.1), as the manual's note to its table 5-5 says the A-weighted level comes from A-weighting the vibration's spectrum and taking off 5 dB for the room.

```
acoustic_spaces.vibration.SPECIAL_BUILDINGS: dict[str, tuple[Criterion, Criterion]] =
    {'auditorium': (Criterion(vibration_vdb=72.0, noise_dba=30.0),
    Criterion(vibration_vdb=80.0, noise_dba=38.0)), 'concert_hall':
    (Criterion(vibration_vdb=65.0, noise_dba=25.0), Criterion(vibration_vdb=65.0,
    noise_dba=25.0)), 'recording_studio': (Criterion(vibration_vdb=65.0,
    noise_dba=25.0), Criterion(vibration_vdb=65.0, noise_dba=25.0)), 'theatre':
    (Criterion(vibration_vdb=72.0, noise_dba=35.0), Criterion(vibration_vdb=80.0,
    noise_dba=43.0)), 'tv_studio': (Criterion(vibration_vdb=65.0, noise_dba=25.0),
    Criterion(vibration_vdb=65.0, noise_dba=25.0))}
```

Table 6-4, special buildings: frequent events, then occasional or infrequent ones.

```
acoustic_spaces.vibration.UNWEIGHTED_DB = -5.0
```

The manual's unweighted sound pressure level of the rumble against the room surfaces' vibration level (dB): "approximately 5 dB" less (section 6.4, step 2d).

```
class acoustic_spaces.vibration.Criterion(vibration_vdb: float, noise_dba: float | None)
```

An impact criterion: the indoor vibration (VdB) and ground-borne noise (dB(A), or None where the manual gives none) above which there is impact.

```
class acoustic_spaces.vibration.Curve(what: str, coefficients: tuple[float, float, float,
    float], reference_kmh: float)
```

A base curve of the ground's vibration (table 6-10): the overall rms vibration level at the ground's surface `D` feet from the track, a0 + a1 x + a2 x^2 + a3 x^3 with x = log10(D), for vehicles in good condition at `reference_kmh` on welded track at grade, with normal propagation through soil; drawn from 10 to 300 ft (`NEAREST_M`, `FARTHEST_M`), and fitted with these equations only there.

`level``(distance_m: float, speed_kmh: float | None = None, extend: bool = False) → float`

The level (VdB) `distance_m` from the track, at `speed_kmh` (eq. 6-4) or at the curve's own speed; a ValueError nearer or farther than the manual draws the curve, unless `extend`: then nearer it holds its value at `NEAREST_M`, and farther it goes on falling as it falls at `FARTHEST_M` (straight on in log10 of the distance; the cubics themselves turn back up there).

`levels``(distances_m, speed_kmh: float | None = None) → ndarray`

`level()` at each of `distances_m` (m), extended as `extend` does.

`class` `acoustic_spaces.vibration.`**`Damage`**`(what: str, ppv_in_s: float, level_vdb: float)`

A category of building and the peak particle velocity (in/s) from which plant may damage it (table 7-5), with the rms level the manual gives for it (VdB).

`class` `acoustic_spaces.vibration.`**`Line`**`(vehicles: str = 'good', track: str = 'good', special_trackwork_m: float | None = None, treatment: str = 'none', structure: str = 'at_grade', ground: str = 'normal')`

What the manual adjusts its base curve for, at the source and along the path (tables 6-11 and 6-12): the vehicles, the track, its treatment and structure, and the ground.

`special_trackwork_m``: float | None = None`

The distance from the nearest turnout or crossover (m), or None for none near.

`source_db``() → float`

Table 6-11's adjustment for the vehicles, the track and its treatment (dB): the vehicles' and the track's added, but only the greater of the two when the track is worn or corrugated, as the manual says.

`path_db``(distance_m: float) → float`

`path_db``(distance_m: ndarray) → ndarray`

Table 6-12's adjustment for the structure and the ground, `distance_m` from the track (dB), a number or an array of them.

`class` `acoustic_spaces.vibration.`**`Plant`**`(what: str, ppv_in_s: float, level_vdb: float)`

A piece of construction plant's vibration 25 ft away (table 7-4): its peak particle velocity (in/s) and its rms vibration level (VdB), which the manual takes 12 dB under the peak (a crest factor of 4).

`acoustic_spaces.vibration.`**`criterion`**`(use: int | str, kind: str) → Criterion`

The criterion for the land use `use` (1, 2 or 3, `CRITERIA`, or a special building, `SPECIAL_BUILDINGS`) and events of `kind` (one of `EVENT_KINDS`).

`acoustic_spaces.vibration.`**`distance_for`**`(level: Callable[[float], float], limit: float,`

```
    near_m: float, far_m: float) → float | None
```

The distance (m) between `near_m` and `far_m` beyond which `level` (of the distance, falling with it) stays at or under `limit`: `near_m` when it is under all along, None when it is over all along.

```
acoustic_spaces.vibration.events(per_day: float) → str
```

Table 6-2: `per_day` events are frequent above 70 a day, occasional from 30 to 70, infrequent under 30.

```
acoustic_spaces.vibration.floors_db(floor: int) → float
```

Table 6-13's attenuation (dB) up to `floor`, counted from 1 above the basement, as the manual's first floor assumes one: 2 dB a floor up to the fifth, 1 dB a floor from there up to the tenth.

```
acoustic_spaces.vibration.ground_level(curve: str, distance_m: float, speed_kmh: float,
    line: Line | None = None) → float
```

The overall vibration level (VdB) at the ground's surface `distance_m` from the track, of vehicles on the base curve `curve` (`CURVES`) at `speed_kmh`, with the adjustments of `line` (none by default): what reaches a building's foundation.

```
acoustic_spaces.vibration.horizontal_distance(point: Sequence[float], path:
    Sequence[Sequence[float]], closed: bool = False) → float
```

The distance (m) in plan from `point` (x, y[, z]) to the nearest point of `path`, a line through (x, y[, z]) points, or the outline of a footprint when `closed`.

```
acoustic_spaces.vibration.horizontal_distances(points, path, closed: bool = False) →
    ndarray
```

(n,) `horizontal_distance()` of each of `points` (n, 2 or 3).

```
acoustic_spaces.vibration.indoor_db(building: str, floor: int = 1) → float
```

The change in vibration level (dB) from the ground outside `building` to its `floor`. For one of `COUPLINGS`, its foundation's coupling loss, the attenuation up to the floor and the floors' amplification (tables 6-12 and 6-13): the first floor of a wood-frame house gives -5 - 2 + 6 = -1 dB, as the manual's own example says. For one of `GENERIC`, the manual's conservative change where the structure is not known; a light wood-frame house's is given for its first three floors only.

```
acoustic_spaces.vibration.level_of(ppv_in_s: float, crest_db: float = 12.0) → float
```

The rms vibration level (VdB) of a peak particle velocity (in/s), `crest_db` under it: the manual's tables 7-4 and 7-5 take a crest factor of 4 (12 dB).

```
acoustic_spaces.vibration.mm_s(in_s: float) → float
```

A velocity in inches a second, in millimetres a second.

```
acoustic_spaces.vibration.nearest_points(points, path, closed: bool = False) → ndarray
```

(n, 2) the point of `path` (as `horizontal_distance()` takes it) nearest each of `points` (n, 2 or 3), in plan.

```
acoustic_spaces.vibration.noise_level(vibration_vdb: float, peak: str = 'low') → float
```

The ground-borne noise (dB(A)) of a room whose surfaces vibrate at `vibration_vdb`, the ground's vibration peaking at `peak` (`NOISE_DB`).

`acoustic_spaces.vibration.plant_level(plant: str, distance_m: float) → float`

Eq. 7-3: the rms vibration level (VdB) `distance_m` from `plant`, Lv - 30 log10(D / 25 ft).

`acoustic_spaces.vibration.plant_ppv(plant: str, distance_m: float) → float`

Eq. 7-2: the peak particle velocity (in/s) `distance_m` from `plant` (`PLANT`), PPV (25 ft / D)^1.5, a point source with normal propagation.

`acoustic_spaces.vibration.speed_db(speed_kmh: float, reference_kmh: float) → float`

Eq. 6-4: the change in vibration level with speed, 20 log10(speed / reference).

Sound through a façade

The sound outdoors heard in a room behind a closed window: a façade's level difference by EN ISO 12354-3, for the pass-by demos' houses.

EN ISO 12354-3:2017 gives a façade's standardized level difference, from the level 2 m in front of it to the room behind it at the reference reverberation time T0 = 0.5 s, per band as

> D_2m,nT = R' + ΔL_fs + 10 lg(V / (6 T0 S)),

R' the façade's apparent sound reduction index, S its area as the room sees it, V the room's volume, ΔL_fs what the façade's shape adds (none for a flat one). The level 2 m in front of the façade, L_2m, holds the façade's own reflection. The pass-by demos and the road works demo simulate it there, the house standing in the scene: at a listener that stands 2 m in front of the wall, or else at one more listener at the point 2 m in front of it, square to it, at the listener's height (`acoustic_spaces.demos.indoors.with_fronts()`). The room's level is

> L_room = L_2m - D_2m,nT.

Here the closed window is the façade's only path: R' S is the window's R over its own area (the area cancels), so the room hears L_2m - R - 10 lg(V / (6 T0 S_window)). The wall round it, a masonry one, lets through far less, and is left out; so are the window's frame and seals and any vent, so the room is as quiet as the glazing lets it be.

`acoustic_spaces.facade.`**`GLAZING_R_DB`** `= (20.5, 16.8, 25.7, 36.4, 41.4, 36.5)`

AGC Glass Europe's THERMOBEL 4-16-4, double glazing of two 4 mm panes 16 mm apart: its sound reduction index per octave band from 125 Hz to 4 kHz (dB), measured by EN ISO 10140 on a pane of 1.23 by 1.48 m, Rw (C; Ctr) = 30 (-1; -4) dB, "not better than +/- 1 dB" (AGC, *Performances: Planibel, Stratobel, Stratophone, Thermobel*, 9000625 EN, 05/18).

`acoustic_spaces.facade.`**`HOUSE`** `= Facade(what='a living room of 50 m³ behind a closed window of 4-16-4 double glazing, 1.82 m²', reduction_db=(14.55, 20.5, 16.8, 25.7, 36.4, 41.4, 36.5, 36.5), window_m2=1.8204, volume_m3=50.0)`

The pass-by demos' house: a living room of 50 m³ (5 by 4 m, 2.5 m high) behind a masonry wall with one window of the double glazing AGC measured, as large as its pane (1.82 m²), closed.

`acoustic_spaces.facade.`**`PANE_M`** `= (1.23, 1.48)`

The pane AGC measured (m): its width and height.

`acoustic_spaces.facade.`**`REFERENCE_DB`** `= (36.0, 45.0, 52.0, 55.0, 56.0)`

ISO 717-1's reference values for airborne sound insulation in octave bands, 125 Hz to 2 kHz (dB), and its spectra No. 1 (pink noise, C) and No. 2 (urban traffic, Ctr), A- weighted and normalised to 0 dB overall.

`acoustic_spaces.facade.`**`T0_S`** `= 0.5`

EN ISO 12354-3's reference reverberation time of a room (s): a furnished dwelling's.

```
class acoustic_spaces.facade.Facade(what: str, reduction_db: tuple[float, ...],
    window_m2: float, volume_m3: float)
```

A room behind a façade whose closed window is its only way in: the window's sound reduction index per octave band (`reduction_db`, 8 bands from 63 Hz), its area and the room's volume, at the reference reverberation time `T0_S`.

property `room_term_db: float`

EN ISO 12354-3's 10 lg(V / (6 T0 S)) of the room and the window (dB).

`level_difference() → ndarray`

(8,) the standardized level difference D_2m,nT per octave band (dB).

`indoor_db() → ndarray`

(8,) the room's level less the level 2 m in front of the façade, its reflection included, per octave band (dB): -D_2m,nT.

`gains() → ndarray`

(8,) the amplitude gain per octave band from 2 m in front of the façade to the room.

`acoustic_spaces.facade.glazing_r_db() → ndarray`

(8,) the double glazing's sound reduction index in the 8 octave bands, 63 Hz to 8 kHz (dB): AGC's measured values (`GLAZING_R_DB`); at 63 Hz the 125 Hz value less the mass law's 6 dB an octave, as below the panes' resonance on the air between them (some 210 Hz for this unit) they move as one; at 8 kHz the 4 kHz value, where the panes' coincidence dip is (near 3 kHz for 4 mm glass), above which the index rises again.

`acoustic_spaces.facade.weighted(reduction_db) → tuple[int, int, int]`

ISO 717-1's single numbers of a sound reduction index in the five octave bands 125 Hz to 2 kHz: (Rw, C, Ctr), the reference curve shifted in 1 dB steps as high as it goes with its unfavourable deviations summing to 10 dB at most, Rw its value at 500 Hz, and the adaptation terms against the two spectra, rounded to whole dB.

Head tracking

Head tracking: binaural playback of an Ambisonics response that turns with the listener's head, so the sources stay where they are in the room.

A simulated AmbiX response holds the sound field at the listener, arriving from every direction. `HeadTrackedBinaural` runs a dry signal through it in real time, turns the field against the head's latest orientation once per block (crossfading from the previous orientation over the block, so turning does not click) and decodes it to the two ears with `binaural_decoder()`.

The orientation lives in a `HeadOrientation`, which anything may set: the editor's controls, a script, or a head tracker sending OSC over UDP to an `OscReceiver`. The receiver reads the messages head trackers commonly send, whatever comes before the last part of the address (`/ypr` and `/SceneRotator/ypr` alike):

- `.../ypr`: yaw, pitch and roll in degrees;
- `.../quaternion`, `.../quaternions` or `.../quat`: w, x, y and z;
- `.../yaw`, `.../pitch` or `.../roll`: one angle in degrees.

Angles follow the scene's convention: yaw turns left, pitch looks up and roll tilts the head to the right. A tracker that counts an axis the other way round needs that axis in `invert`.

```
acoustic_spaces.headtracking.AXES = ('yaw', 'pitch', 'roll')
```

The axes of a head orientation, as `invert` names them.

```
acoustic_spaces.headtracking.DEFAULT_OSC_PORT = 9000
```

UDP port `OscReceiver` listens on unless told otherwise.

```
acoustic_spaces.headtracking.QUATERNION_ADDRESSES = ('quaternion', 'quaternions', 'quat')
```

Last parts of OSC addresses that carry a quaternion.

```
class acoustic_spaces.headtracking.HeadOrientation(invert: Iterable[str] = ())
```

The listener's head orientation, set from one thread and read from another.

`invert` names the axes (`"yaw"`, `"pitch"`, `"roll"`) whose incoming angles count the other way round. `centre()` takes the current orientation as straight ahead; `rotation()` and `angles()` are relative to it.

```
set_angles(yaw: float | None = None, pitch: float | None = None, roll: float | None =
    None) → None
```

Yaw, pitch and roll in degrees; an angle left out keeps its last value. Angles that are not finite numbers are ignored.

```
set_quaternion(w: float, x: float, y: float, z: float) → None
```

The orientation as a quaternion (normalised first) in the scene's axes: x forward, y left, z up.

```
look(yaw: float = 0.0, pitch: float = 0.0, roll: float = 0.0) → None
```

Turn the head to yaw, pitch and roll in degrees relative to the centre, as `angles()` gives them back (`invert` does not apply).

```
centre() → None
```

Take the current orientation as straight ahead.

```
rotation() → tuple[int, ndarray]
```

(version, rotation (3, 3)) of the head relative to the centre. The version counts the changes, so a reader can tell when to look again.

```
angles() → tuple[float, float, float]
```

Yaw, pitch and roll in degrees relative to the centre.

```
class acoustic_spaces.headtracking.HeadTrackedBinaural(ir: ndarray, sample_rate: float,
    *, hrtf: HRTF | None = None, orientation: HeadOrientation | None = None, block: int
    = 256, decoder: str = 'magls')
```

A dry signal through an Ambisonics response, turned against the head and decoded to the two ears, one block at a time.

`ir` is (taps, channels) AmbiX (ACN / SN3D) of order 1 to 7, at `sample_rate`, as the simulator renders it for a listener: directions relative to the listener's orientation in the scene, which head tracking then turns from. `hrtf` defaults to the spherical head model and is resampled to `sample_rate`; `decoder` is `"magls"` or `"ls"` (see `binaural_decoder()`). `orientation` (a new `HeadOrientation` by default) is read once per block. It works with `acoustic_spaces.render.play()` as a `Convolver` does, with one input and two outputs, and the only latency is the block.

```
process(block: ndarray, out: ndarray | None = None) → ndarray
```

One block (block,) of the dry signal to (block, 2) float32, into `out` when given, as `Convolver.process` takes it. The buffers the size of a block are kept from block to block.

```
run(signal: ndarray, head: Callable[[float], tuple[float, float, float]] | None =
    None) → ndarray
```

A whole signal (samples,) through the renderer: (samples + taps - 1, 2).

`head` gives the yaw, pitch and roll in degrees at a time in seconds, relative to the centre as `HeadOrientation.look()` takes them (`invert` does not apply); it is asked at the start of every block. Without it, the orientation stays as it is.

```
reset() → None
```

Forget all past input.

```
class acoustic_spaces.headtracking.OscReceiver(orientation: HeadOrientation, port: int =
    9000, host: str = '127.0.0.1')
```

Sets `orientation` from the OSC messages arriving on UDP `port` of `host` (0 picks a free port; `port` says which), in a background thread.

The default host takes messages from this computer only; `"0.0.0.0"` takes them from the network too, as from a phone. Use the receiver as a context manager, or call `start()` and `stop()`.

```
received
```

Messages that set the orientation.

```
rejected
```

Packets that were not OSC.

```
stop() → None
```

Stop listening and free the port.

```
acoustic_spaces.headtracking.apply_osc(orientation: HeadOrientation, address: str,
    arguments: list) → bool
```

Set `orientation` from one OSC message (see the module's description); False when the message is not about the head's orientation.

```
acoustic_spaces.headtracking.parse_osc(packet: bytes, _depth: int = 0) → list[tuple[str,
    list]]
```

The messages (address, arguments) of an OSC packet, with bundles unpacked.

Reads 32-bit floats and integers, doubles, 64-bit integers, strings and the argument-less types (true, false, nil, infinity); anything else, or a malformed packet, raises ValueError.

Measurement

Measuring impulse responses with exponential sine sweeps.

The usual chain is: generate an excitation (`Sweep`), play it through a loudspeaker while recording one or more microphones, deconvolve the recording into an impulse response, then analyse it with `acoustic_spaces.parameters`, the same code that analyses simulated responses.

- `Sweep` is a synchronized exponential sweep (Novak et al., 2015): its harmonic distortion lands in separate impulse responses *before* the linear one, so deconvolution leaves the linear response clean. The excitation repeats the sweep with silence after each one, long enough for the room to decay.
- `deconvolve()` divides spectra with Kirkeby regularisation outside the swept band, then averages the repeats, which lowers background noise by 3 dB per doubling.
- `measure()` plays and records through `sounddevice` (the `measure` extra) and returns the deconvolved responses. A loopback channel (an output wired straight back to an input) removes the latency of the audio interface.

Rules of thumb: make the silence after each sweep longer than the reverberation time, and the sweep itself about ten times longer than the reverberation time if distortion products must not overlap the linear response at all.

```
class acoustic_spaces.measurement.Measurement(impulse_responses: ndarray, sample_rate:
    int, recording: ndarray, sweep: Sweep, latency_s: float | None = None)
```

Measured impulse responses, one column per microphone.

latency_s: float | None = None

Round-trip latency removed using the loopback channel (s), or None.

```
class acoustic_spaces.measurement.Sweep(sample_rate: int = 48000, duration_s: float =
    6.0, f_start: float = 20.0, f_stop: float = 20000.0, level_db: float = -6.0,
    silence_s: float = 3.0, repeats: int = 1, fade_in_s: float = 0.05, fade_out_s: float
    = 0.005)
```

An exponential sine sweep excitation, repeated with silence after each sweep.

property **rate**: float

L of x(t) = sin(2 pi f_start L exp(t / L)), rounded so that harmonics stay in phase with the fundamental (the synchronized sweep).

property **sweep_length**: int

Samples in one sweep (the duration after synchronization).

property **period**: int

Samples from one sweep's start to the next.

harmonic_delay_s(order: int) → float

How far before the linear response the `order`-th harmonic's response lands.

sweep() → ndarray

One sweep, faded in and out.

signal() → ndarray

The whole excitation: every repeat, each followed by its silence.

save(path: str | Path) → None

Write the excitation as a float WAV and its settings next to it as JSON.

classmethod **load**(path: str | Path) → Sweep

Read the settings of a saved excitation (its .json, or the .wav next to it).

acoustic_spaces.measurement.**deconvolve**(recording: ndarray, sweep: Sweep, length_s: float | None = None, reference: ndarray | None = None) → ndarray

Impulse responses (samples, channels) from a recording of `sweep.signal()`.

`recording` is (samples,) or (samples, channels), starting when playback started. Each repeat is deconvolved and the repeats are averaged. `length_s` defaults to the silence after each sweep, and a longer one is cut. With repeats, it is cut to the silence: the next sweep's harmonic distortion lands right after it. With one sweep, it is cut to the recording's length less the sweep's, beyond which the recording lacks the response's high frequencies; for a recording as long as the excitation, that is the silence too. `reference` replaces the generated sweep, for instance with a loopback recording of one sweep: (samples,) or (samples, 1), as `read_wav()` gives a mono file.

acoustic_spaces.measurement.**list_devices**() → str

The audio devices sounddevice sees, as text.

acoustic_spaces.measurement.**measure**(sweep: Sweep, *, inputs: Sequence[int] = (1,), output: int = 1, loopback: int | None = None, device: object = None, length_s: float | None = None, play_record: Callable[[ndarray, int, Sequence[int], Sequence[int], object], ndarray] | None = None) → Measurement

Play `sweep` on one output, record `inputs` and deconvolve them.

Channels are 1-based. With `loopback` (an input wired to the output), the latency of the audio interface is measured on it and removed from every response. `play_record` replaces the audio device, for tests or other audio back ends.

```
acoustic_spaces.measurement.play_and_record(signal: ndarray, sample_rate: int, outputs:
    Sequence[int], inputs: Sequence[int], device: object = None) → ndarray
```

Play `signal` on the output channels and record the inputs (1-based channels).

Returns (samples, len(inputs)), as long as the signal.

```
acoustic_spaces.measurement.remove_latency(responses: ndarray, loopback: int) →
    tuple[ndarray, int]
```

Move responses (samples, channels) earlier by the delay of a loopback channel.

`loopback` is the 0-based column of the loopback response, which is dropped. Returns the other columns, padded with zeros at the end, and the delay in samples.

Comparing with measurements

Comparing a simulated response with a measured one, by the parameters of ISO 3382-1.

A model is checked against the room it stands for by comparing its parameters with those measured there, octave band by octave band, in just noticeable differences (JNDs, ISO 3382-1:2009, annex A, table A.1, and annex B for IACC; by analogy for T20, T30 and C50, see `JND`): a difference of less than one JND is too small to be heard.

- `compare_impulse_responses()` analyses two impulse responses, each at its own sample rate, as `from_impulse_response()` does.
- `compare_parameters()` compares two sets of parameters already computed.

Both give a `Comparison`: per parameter and band the simulated and the measured value, their difference and that difference in JNDs; and per parameter, how many bands are within one JND and the mean absolute difference in JNDs from 125 Hz to 4 kHz. A band where either side has no value is missing, with the reason, never zero: a response that decays by less than 20 dB, a decay time whose range does not clear the background noise, G without a reference on both sides, or a band above half a response's sample rate.

```
acoustic_spaces.compare.CONVERGED_JND = 0.5
```

The standard error, in JNDs, past which more rays would be worth tracing: the parameter would change by about a JND or more between runs with other seeds.

```
acoustic_spaces.compare.JND: dict[str, tuple[float, bool]] = {'c50': (1.0, False), 'c80':
    (1.0, False), 'd50': (0.05, False), 'edt': (0.05, True), 'g': (1.0, False),
    'iacc_e': (0.075, False), 'iacc_l': (0.075, False), 'lf': (0.05, False), 'lfc':
    (0.05, False), 't20': (0.05, True), 't30': (0.05, True), 't30_x': (0.05, True),
    't30_y': (0.05, True), 't30_z': (0.05, True), 'ts': (0.01, False)}
```

Just noticeable differences per parameter: the JND, and whether it is a fraction of the measured value (the decay times) rather than a step in the parameter's own unit (dB, s, or none for D50, LF, LFC and IACC). ISO 3382-1:2009 gives those of EDT, C80, D50, Ts, G and LF (annex A, table A.1) and of IACC (annex B); T20 and T30 take EDT's and C50 takes C80's by analogy, and the directional decay times take that of T30. The standard gives none for LJ and stage support, which are not compared.

```
acoustic_spaces.compare.SUMMARY_BANDS_HZ: tuple[int, ...] = (125, 250, 500, 1000, 2000,
    4000)
```

The octave bands the mean difference of each parameter is taken over (Hz).

```
class acoustic_spaces.compare.Comparison(simulated: RoomParameters, measured:
    RoomParameters, parameters: dict[str, ParameterComparison])
```

A simulated and a measured response compared parameter by parameter.

`simulated` and `measured` are the parameters of each side, and `parameters` a `ParameterComparison` for each name of `PARAMETERS`, and of those further ones with a JND that both sides give, which indexing gives as well: `comparison["t30"].difference_jnd`.

as_dict() → dict

Plain lists and numbers, with None for missing values (JSON friendly).

table() → str

A text table: per parameter, the simulated and the measured value in each band and their difference in JNDs, marked * where it is more than one; then why the bands shown as - cannot be compared.

summary() → str

A text table: per parameter, how many of the compared bands are within one JND, and the mean absolute difference in JNDs from 125 Hz to 4 kHz.

class acoustic_spaces.compare.**ParameterComparison**(name: str, simulated: ndarray, measured: ndarray, jnd: ndarray, difference: ndarray, difference_jnd: ndarray, missing: list[str | None])

One parameter of a simulated and a measured response, one value per octave band.

`difference` is the simulated value minus the measured one, and `difference_jnd` the same in JNDs, so both are positive where the simulation reads higher. They are NaN in the bands that cannot be compared; `missing` holds the reason for each of those, and None for the bands that are compared.

jnd: ndarray

The JND in each band, in the parameter's unit (NaN for a relative JND where there is no measured value to take it of).

property compared: int

How many bands have a value on both sides.

property within_jnd: int

How many of the compared bands are at most one JND apart.

property mean_abs_jnd: float

Mean absolute difference in JNDs over the compared bands from 125 Hz to 4 kHz (`SUMMARY_BANDS_HZ`); NaN when none of them is compared.

acoustic_spaces.compare.**compare_impulse_responses**(simulated: ndarray, simulated_rate: float, measured: ndarray, measured_rate: float, *, simulated_reference: float | Sequence[float] | None = None, measured_reference: float | Sequence[float] | None = None, filters: str = 'iec', noise: bool = True) → **Comparison**

Compare a simulated and a measured impulse response, each at its own sample rate.

Each is mono, (samples,) or (samples, 1), and analysed as by `from_impulse_response()` with the same `filters` and `noise` settings. The references are those of its `reference_energy`: the energy of the same source at 10 m in free field, per band or of a broadband impulse (0.01 for responses made by this package, 0.01 * ir_gain**2 for those written with `normalize`). G is compared only where both sides have one; without, it is missing in every band. A band at or above half a response's sample rate is missing (see `from_impulse_response()`).

```
acoustic_spaces.compare.compare_parameters(simulated: RoomParameters, measured:
    RoomParameters, *, simulated_rate: float | None = None, measured_rate: float | None
    = None) → Comparison
```

Compare two sets of parameters, one value per octave band of `OCTAVE_BANDS_HZ`.

NaN values are missing bands, whose reason is read from the parameters themselves: a band with no parameter at all lies above half the sample rate of its response (when `simulated_rate` or `measured_rate`, in Hz, gives it) or else decays by less than 20 dB, G alone is NaN without a reference, and a decay time alone when the decay does not reach the bottom of its fit range (`compare_impulse_responses()` also tells when the noise is in the way). LF, LFC, IACC and the directional decay times are compared where both sides give them.

```
acoustic_spaces.compare.in_jnds(values: RoomParameters, errors: RoomParameters) →
    dict[str, ndarray]
```

`errors` of `values` (such as their standard errors, `standard_errors()`) in JNDs, for each parameter with a JND, per octave band; a relative JND is taken of the value itself. NaN where either is undefined.

Fitting materials to measurements

Material calibration: absorption fitted to the parameters measured in a room.

A model of an existing room starts from absorption coefficients out of tables, which seldom match the room as built. `calibrate()` adjusts the absorption of the materials named, octave band by octave band, until the simulated parameters (T30, EDT and C80 by default) at each measured source and listener pair come as close as they can to those measured, in just noticeable differences (ISO 3382-1, annex A, and by analogy for T30; see `JND`).

It does not trace the room again for every trial. Each round traces it once, keeping the reflections each ray had off each material (`reweight`); the responses for other absorption then come from re-weighting those rays, so a least-squares fit per band (`scipy.optimize.least_squares()`) costs little more than computing the parameters. The next round traces again at the absorption found, until it moves by less than `tolerance`; a last trace gives the parameters reported after the fit.

Measurements seldom pin down every material: what they fix best is the total absorption, and two surfaces that the sound meets in much the same way (a ceiling and a floor of a flat room, say) trade absorption freely. So the materials keep their proportions unless the measurements ask otherwise: the fit is free to scale every material's absorption by a common factor, but a material that moves by a factor more than the others costs as much as missing a parameter, one JND for each factor of `PRIOR_FACTOR`. Without measurements that tell the materials apart this is the fit of a single factor per band, as fitting a diffuse-field formula to the reverberation time does. `Calibration.uncertainty` says, per material and band, how far its absorption could move, the other materials following it, before the sum of the squared misses grows by one JND squared (its standard error): what the measurements leave open.

```
acoustic_spaces.calibration.DEFAULT_PARAMETERS: tuple[str, ...] = ('t30', 'edt', 'c80')
```

The parameters fitted unless others are named.

```
acoustic_spaces.calibration.PRIOR_FACTOR = 2.0
```

A material whose absorption moves by this factor more than the others' does costs as much as missing one parameter by one JND.

```
class acoustic_spaces.calibration.Calibration(scene: ~acoustic_spaces.scene.model.Scene,
    materials: list[str], parameters: tuple[str, ...], initial: ~numpy.ndarray,
    absorption: ~numpy.ndarray, uncertainty: ~numpy.ndarray, measured: dict[tuple[str,
    str], ~acoustic_spaces.parameters.RoomParameters], before: dict[tuple[str, str],
    ~acoustic_spaces.parameters.RoomParameters], after: dict[tuple[str, str],
    ~acoustic_spaces.parameters.RoomParameters], rounds: int, bands_hz: tuple[int, ...]
    = (63, 125, 250, 500, 1000, 2000, 4000, 8000), details: dict[str, float] =
    <factory>)
```

What `calibrate()` found: the absorption per material and octave band, and the parameters before and after, against those measured.

```
uncertainty: ndarray
```

(materials, bands) the standard error of each material's absorption, for misses of one JND: how far it can move, the others following it, before the sum of the squared misses grows by one JND squared. It is what the measurements leave open. Above 0.1 or so they hardly constrain it.

```
fitted_scene() → Scene
```

A copy of the scene with the fitted absorption.

```
errors(which: str = 'after') → dict[str, ndarray]
```

Per fitted parameter, the mean absolute difference from the measurements in JNDs per band over the pairs (NaN where no pair has both values); `which` is "before" or "after".

```
table() → str
```

The fitted absorption, what it started from, its uncertainty and the errors before and after, as text.

```
acoustic_spaces.calibration.calibrate(room: Room | Scene | str | Path, measured:
    Mapping[tuple[str, str], RoomParameters], materials: Sequence[str] | None = None, *,
    parameters: Sequence[str] = ('t30', 'edt', 'c80'), rounds: int = 4, bounds:
    tuple[float, float] = (0.01, 0.99), tolerance: float = 0.01, threads: int | None =
    None, progress: Callable[[str], None] | None = None) → Calibration
```

Fit the absorption of `materials` (default: every material on a surface but openings, at most 16) so that the simulated `parameters` (edt, t20, t30, c50, c80, d50, ts and g) match those `measured` at each (source, listener) pair of the scene, band by band, in JNDs.

The scene's simulation settings (rays, response length, seed, and threads unless `threads` is given) are used, with stochastic ray tracing in octave bands and no wave solver. At most `rounds` traces at new absorption are made; it stops sooner once no value moves by more than `tolerance`. Absorption stays within `bounds` (the lower one above 0, as the prior takes the logarithm of the absorption), and above what a material lets through.

```
acoustic_spaces.calibration.with_absorption(scene: Scene, materials: Sequence[str],
    absorption: ndarray) → Scene
```

A copy of `scene` whose `materials` take `absorption` (materials, octave bands), as calibrated to measurements (`with_absorption()`).

Re-weighting a traced response for new absorption, without tracing it again.

The energy of every arrival is what left the source that way times, at each reflection, the share the surface reflects, 1 - a, and factors that do not depend on the absorption (scattering, air, the source's directivity, what a partition lets through). With other absorption a', the same paths bring each arrival times the product over its reflections of (1 - a') / (1 - a). The paths themselves need not be traced again: where a ray goes depends on the scattering, not on the absorption. (Russian roulette and splitting chose rays by their energies, but those choices only weight the estimate; it stays unbiased.) An early reflection off a ground with an impedance is not of this kind: it takes the impedance's reflection factor in place of 1 - a, and that impedance cannot change without tracing again (`reweighted_result()`). Nor quite one off a ground with zones, which takes the factors of the grounds its Fresnel zone covers and is re-weighted as off the ground it meets alone. With source and listener 0.5 and 1.5 m high and 30 m apart, and a zone without an impedance taken from an absorption of 0.1 to 0.2, a reflection half a metre inside the zone's edge falls by 0.5 dB re-weighted and by 0.13 to 0.20 dB traced again; half a metre outside it, by none and by 0.11 to 0.16 dB. Nor is the path of the solver for whole rows (`row_paths`): it reflects off no face, and its sound depends on the absorption of a ground without an impedance through the admittance fitted to it. A result with such a path is not re-weighted for another absorption of such a ground (`reweighted_result()`); a spread's draws and a calibration's fits, which re-weight response by response, leave that path as it was traced.

The other early paths carry the faces they reflect off, so their energies re-weight exactly. Where they add in pressure (`coherent`), a reflection also shifts the phase, by that of its reflection factor, which comes from the admittance fitted to its face's absorption and the angle it meets the face at; the paths keep those angles (`cosines`), so their phases turn to the new absorption's exactly too (`turned_paths()`). Of a diffracted path only the reflections on its way count, not the faces of the edge it bends round, nor how it interferes with the reflection it shades. The late part's arrivals are far too many to keep, so the tracer sums, per listener and slot of `MOMENT_STEP`, the energy times the reflections each ray has had off each group of faces and their products (`stochastic_energy()` with `groups`): the energy-weighted mean mu and covariance C of the counts n in the slot. With l_i = ln((1 - a'_i) / (1 - a_i)) for group i, a slot's energy becomes E exp(K), K = ln of the mean of exp(n . l) over its arrivals, which mu and C give to second order in l as mu . l + 1/2 l' C l. That form alone holds near the traced absorption only: for much more absorption its square term outgrows the first and the energy rises again, without bound. `moment_exponent()` takes instead the form the counts of a diffuse field lead to. There a ray's N reflections fall on the groups by chance (a multinomial), and with u_i = exp(l_i) - 1, the change of what group i reflects,

> K = N ln(1 + sum_i mu_i u_i / N),

which is Eyring's decay for the new absorption over that for the old. N comes from the covariance: 1 / N = -mu' F mu / (mu . mu)^2 with F = C - diag(mu), which is -mu mu' / N for a multinomial. It is held within plus and minus one over the sum of mu: a ray has had at least the reflections the groups count, and counts that spread wider than by chance make it negative. What of the covariance it leaves, R = F + mu mu' / N, adds 1/2 w' R w, with w_i = u_i for more absorption and l_i for less. K is then the same to second order in l as the form above, exact for the same absorption and for counts that fall by chance, and held to what holds for any counts: at least mu . l (Jensen's inequality), and no gain where no group reflects more. Further from the traced absorption it stays an estimate from two moments. In a box of 8 by 6 by 3.5 m traced with a ceiling that absorbs 0.5 (20 000 rays, at 1 kHz), the late part's decay time for a ceiling of 0.9 is within 1 % of tracing again, for 0.99 within 5 %, and for 0.1 it is 6 % too short; with a ceiling of 0.85 taken to 0.2, 16 % too short. So `calibrate()` traces again at the absorption it arrives at. Its fit within a round keeps the early paths' phases as traced, which that next trace turns.

class acoustic_spaces.engine.reweight.**Redraws**(paths: EarlyReflections, face_group: ndarray, groups: int, old: ndarray, admittance_of=None)

What re-weighting the early paths `paths` of one traced response (`reweighted()`) for draw after draw of the absorption of `groups` groups of faces (`face_group`) takes that does not depend on the draw: how often each path reflects off each group (`path_counts()`); for each set of groups a draw changes, the reflections `turned_paths()` turns and the phases of the absorption `old` as traced that it turns them from, with `admittance_of`; and what the coherent bands' echograms of the draws share (`delays`, a `SameDelays`). A draw comes out the same with it as without it: for other paths, groups, traced absorption or admittances than its own, each is worked out as without it. Made from its arguments alone, for one response's draws, and dropped after them.

counts(paths: EarlyReflections, face_group: ndarray, groups: int) → ndarray

`path_counts()` of these arguments (read-only).

turned(paths: EarlyReflections, face_group: ndarray, old: ndarray, new: ndarray, admittance_of=None) → EarlyReflections

`turned_paths()` of these arguments, for `paths` that are this one's re-weighted (`reweighted_paths()`).

acoustic_spaces.engine.reweight.**cannot_reweight**(room, settings=None) → str | None

Why a run of `room` with `settings` (default the scene's) cannot be re-weighted for other absorption, in words that follow "it needs", or None when it can: another late method, third-octave bands, the wave solver, or more than `MAX_GROUPS` materials on its surfaces.

acoustic_spaces.engine.reweight.**face_groups**(room, materials: Sequence[str]) → ndarray

(faces,) the index in `materials` of each face's material, or -1 for faces of other materials.

acoustic_spaces.engine.reweight.**groups_of**(room, settings=None) → tuple[ndarray, list[str]] | None

Face groups to trace `room` with so that its absorption can change afterwards (`reweighted_result()`): the group of each face and the materials, those on surfaces that absorb less than everything in some band (a band in which one absorbs everything keeps that: no energy is left there to re-weight). None when its simulation settings (`settings`, default the scene's) do not keep what re-weighting needs (`cannot_reweight()`) or no material is left.

acoustic_spaces.engine.reweight.**log_ratio**(old: ndarray, new: ndarray) → ndarray

(groups, bands) ln((1 - new) / (1 - old)): the change in level per reflection; 0 where both absorb everything, which leaves no energy to change.

acoustic_spaces.engine.reweight.**moment_exponent**(moments: ndarray, ratio: ndarray) → ndarray

(bands, slots) the exponent by which each slot of late energy changes, from the moments of the counts (terms, bands, slots) and `log_ratio` (groups, bands): the multinomial's form with the rest of the covariance to second order, within the bounds that hold for any counts (see the module's text).

acoustic_spaces.engine.reweight.**path_counts**(paths: EarlyReflections, face_group: ndarray, groups: int) → ndarray

(paths, groups) how often each early path reflects off each group of faces.

acoustic_spaces.engine.reweight.**reweighted_paths**(paths: EarlyReflections, counts: ndarray, ratio: ndarray)

`paths` with their energies for the new absorption (`counts` from `path_counts()`, `ratio` from `log_ratio()`).

acoustic_spaces.engine.reweight.**reweighted_materials**(room) → list[str]

The materials on `room`'s surfaces that absorb less than everything in some band: those whose absorption a re-weighting can change.

acoustic_spaces.engine.reweight.**reweighted_result**(result, traced, face_group: ndarray, materials: Sequence[str], scene)

`result`, simulated from the scene `traced` with `face_group` for `materials` (`groups_of()`), for the absorption of `scene`: re-weighted, without tracing again. None when `scene` differs from `traced` in more than the absorption of its materials, a material changed that re-weighting does not cover, or one did that the solver for whole rows took for a response (`solver_grounds()`).

acoustic_spaces.engine.reweight.**solver_grounds**(result) → list[str]

The materials without an impedance on the ground of `result`'s scene, where some response of it has the path of the solver for whole rows (`row_paths`) among its early paths; none otherwise. That path's sound depends on those grounds' absorption, through the admittance fitted to it, and reflects off no face that a re-weighting counts.

acoustic_spaces.engine.reweight.**slot_factor**(exponent: ndarray, slots: int, time_step: float, moment_step: float)

(bands, slots) the factor on each of `slots` slots of `time_step`, taking that of the moment slot it lies in.

acoustic_spaces.engine.reweight.**turned_paths**(paths: EarlyReflections, face_group: ndarray, old: ndarray, new: ndarray, admittance_of=None) → EarlyReflections

`paths` with the phase of each reflection off a group of faces (`face_group`) turned, in the bands where they add in pressure, from that of the group's absorption `old` (groups, bands) to that of `new`: from the angle each meets its face at (`cosines`). Reflections whose phase their face's absorption does not give keep theirs. `admittance_of` gives the admittance of an absorption row (`absorption_phases()`; by default the one fitted to it).

Audio files and results

WAV files: 32-bit float output, and input in the common PCM and float formats.

`acoustic_spaces.audio.`**`channel_mask`**`(path: str | Path) → int | None`

> The speaker positions (WAVE_FORMAT_EXTENSIBLE channel mask) of a WAV file, or `None` for a file in the plain format.

`acoustic_spaces.audio.`**`read_wav`**`(path: str | Path) → tuple[ndarray, int]`

> Read a WAV file as float64 (frames, channels) and its sample rate.
>
> PCM 16, 24 and 32 bit are scaled to [-1, 1); float 32 and 64 bit are kept as they are. RF64 and BW64 files (WAV with 64-bit sizes, for files over 4 GiB) are read too. The samples come in a few MiB at a time, so the file takes no copy of itself in memory.

`acoustic_spaces.audio.`**`write_wav`**`(path: str | Path, data: ndarray, sample_rate: int, channel_mask: int | None = None) → None`

> Write samples (frames,) or (frames, channels) as 32-bit float WAV.
>
> Float WAV keeps levels above full scale, so relative levels between files survive. Files with more than two channels, or a `channel_mask` (WAVE_FORMAT_EXTENSIBLE speaker positions; 0 for none, as in AmbiX), use the extensible format. A file of more than 4 GiB is written as RF64 (EBU Tech 3306), the WAV format whose sizes take 64 bits. The samples go out a few MiB at a time, so a long file takes no copy of itself in memory.

Write a simulation's responses to a folder: the files `acoustic-spaces simulate` and the editor's Export produce.

For every (source, listener) pair, one WAV file per output format (`<source>__<listener>.wav` for mono, `<source>__<listener>.<format>.wav` for the others, with `-2`, `-3`… after the names or the format where two files would otherwise be the same), and for the whole run `parameters.json`: the settings, run details and the ISO 3382-1 parameters of every response (with IACC when a binaural response is written, from the first if there are several), its STI, STIPA and U50, and its echo criteria. Outdoors also its level relative to free field, and the insertion loss of the buildings, barriers or vegetation the run was also made without (`simulation.insertion_loss_of`). `acoustic-spaces auralize` reads that folder.

`acoustic_spaces.results.`**`file_name`**`(text: str) → str`

> `text` with anything but letters, digits, dots, dashes and underscores replaced.

`acoustic_spaces.results.`**`results_folder`**`(out: str | Path) → Path`

> The folder `out` ready for results: made where it is missing, and found to take files. Otherwise an OSError says why not: `out` is a file, or the folder cannot be made or written in. `acoustic-spaces simulate` asks before its run, so that a run is not lost at its end.

`acoustic_spaces.results.`**`unique_stems`**`(stems: Sequence[str], suffixes: Sequence[str] = ('',)) → list[str]`

`stems`, each that would name the same file as one before it with any of `suffixes` given `-2`, `-3`… in turn, also where the names differ only in case (the file systems of Windows and macOS ignore it): `file_name()` turns `A/B` and `A_B` into the same name, and names in other scripts into `unnamed`.

```
acoustic_spaces.results.write_results(result: SimulationResult, out: str | Path, formats:
    Sequence[Format] | None = None, *, sample_rate: int | None = None, normalize: bool =
    False, plots: bool = False, progress: Callable[[str], None] | None = None) → dict
```

Write every response of `result` into the folder `out` and return the report saved as `parameters.json`.

`formats` are formats of `acoustic_spaces.spatial` (default: the scene's `outputs`); `sample_rate` defaults to the simulation's. `normalize` scales all files together so the loudest sample is at -1 dBFS; otherwise they keep the simulation's levels, which exceed full scale when a listener is closer than 1 m to a source (the report's `peak` says by how much). `plots` adds the echogram and direction plots of `acoustic_spaces.viz` when matplotlib is installed. Each response's spread from its materials' absorption uncertainty, where the result has it (`spreads`), goes into the report too.

The files are written into a hidden folder in `out` first and moved into place once all of them are, the report last, so that a run that fails or is cancelled while writing (a full disk, say) leaves `out` as it was. Each response is rendered and written in turn (on the run's threads, one each), so memory holds a response's files at a time, not the whole run's, and its late part while its formats are rendered; threads the responses leave free render the channels of each side by side, a channel's band signals in memory for each, the channels in progress held to a quarter of the computer's memory. With `normalize` the files are scaled once all are written, by one rounding to single precision more.

Example scenes

Example scenes that come with the package.

They are the scenes in the repository's `examples` folder, which wheels carry as `acoustic_spaces/example_scenes`:

```
from acoustic_spaces import Room
from acoustic_spaces.examples import example_path, list_examples

for example in list_examples():
    print(example.name, example.title)
room = Room.from_yaml(example_path("concert_hall"))
```

`acoustic-spaces examples` lists them, and `--copy DIR` copies them to a folder you can edit; the editor opens them from its File menu, in the groups of `GROUPS`.

```
acoustic_spaces.examples.GROUPS: tuple[tuple[str, tuple[str, ...]], ...] = (('Rooms',
    ('shoebox', 'room_modes', 'studio', 'listening_room', 'apartment', 'archway',
    'workshop')), ('Halls and churches', ('lecture_hall', 'concert_hall', 'orchestra',
    'church', 'gallery')), ('Sound systems', ('town_hall_pa', 'concert_stage',
    'stadium_concert')), ('Outdoors', ('street', 'roadside', 'railway', 'tree_belt',
    'hillside', 'open_air_stage', 'lakeside_concert', 'helicopter_flyover', 'airport')),
    ('Models from CAD and BIM files', ('rehearsal_room', 'bim_office',
    'leaky_classroom')), ('Measurements and material libraries', ('fitting',
    'material_libraries')))
```

The groups the examples are shown in, in order, each with its examples (folder names) from the simplest. The titles are English; the editor translates them.

```
acoustic_spaces.examples.OTHER_GROUP = 'Other examples'
```

The group of an example that `GROUPS` leaves out.

```
class acoustic_spaces.examples.Example(name: 'str', title: 'str', description: 'str',
    path: 'Path', model: 'Path | None' = None, group: 'str' = 'Other examples')
```

`name`: `str`

Folder name, used to pick the example.

`title`: `str`

The scene's `name`.

`description`: `str`

The comment at the top of the scene file.

`path`: `Path`

The scene file.

`model`: `Path | None = None`

Its model file, or None for a room built from dimensions.

`group`: `str = 'Other examples'`

The group it is shown in (English: see `GROUPS`).

`property extra`: `str | None`

The extra its model is read with (`ifc` for an IFC file), or None.

`property missing_extra`: `str | None`

The extra its model needs when that is not installed, or None.

```
acoustic_spaces.examples.copy_examples(destination: str | Path, names: list[str] | None =
    None) → list[Path]
```

Copy examples (all, or `names`) into `destination`, one folder each, and return their scene files. Raises `FileExistsError` rather than overwrite a folder.

`acoustic_spaces.examples.example_groups(examples: list[Example] | None = None) → list[tuple[str, list[Example]]]`

The examples (default: all) by group, in the order of `GROUPS` and within a group from the simplest; those it leaves out last, in `OTHER_GROUP`. Groups with no example are left out.

`acoustic_spaces.examples.example_of(scene_file: str | Path) → str | None`

The example the scene file `scene_file` is, or is a copy of: its folder's name when that names an example, else the example whose scene has the same `name`, so that a copy saved anywhere keeps its example; None for any other scene, or one that cannot be read.

`acoustic_spaces.examples.example_path(name: str) → Path`

The scene file of the example `name` (a folder name from `list_examples()`).

`acoustic_spaces.examples.examples_dir() → Path`

The folder that holds one sub-folder per example.

Installed packages carry it as `acoustic_spaces/example_scenes`; a source checkout (an editable install) uses the repository's `examples` folder.

`acoustic_spaces.examples.is_installed_example(path: str | Path) → bool`

Whether `path` lies among the examples of an installed package, which an upgrade or an uninstall replaces or removes: nothing is to be written there, even where the folder can be written (a virtual environment). A source checkout's `examples` folder is not.

`acoustic_spaces.examples.list_examples() → list[Example]`

Every example scene, sorted by folder name.

3D view and plots

Pictures of rooms and responses.

- 3D view (PyVista): surfaces coloured by material, sources with their directivity balloons, listeners, and early reflection paths coloured by order (paths diffracted at an edge in a colour of their own); outdoors the vegetation, and the wind: arrows over the ground, the weather zones, and masts with the wind's profile up them.
- Leaks (PyVista): the open edges of a model that is not closed, and the rays of the leak finder that escape through its gaps.
- Animation (PyVista): sound particles spreading from a source and reflecting around the room, in a window with a time slider or saved as a GIF or MP4 video; a walker going along its walk, as a video to go with its sound.
- Response plots (matplotlib): echogram with the early paths, decay curves per band, and arrival directions as the listener faces; the decays of a simulated and a measured response band by band; polar patterns of a source directivity; the wind, temperature and speed of sound over the ground outdoors.
- Coverage maps: a parameter over the audience area, from above (matplotlib) or as coloured tiles in the 3D view.

Needs the viz extra (`pip install "acoustic-spaces[viz]"`, or `uv sync --extra viz` in a source checkout). The 3D view works in a desktop window, in Jupyter, and off screen for screenshots.

```
acoustic_spaces.viz.MATERIAL_COLOURS = ('#8dd3c7', '#ffffb3', '#bebada', '#fb8072',
    '#80b1d3', '#fdb462', '#b3de69', '#fccde5', '#d9d9d9', '#bc80bd', '#ccebc5',
    '#ffed6f')
```

ColorBrewer Set3: light colours, so the saturated source and listener markers stand out.

```
acoustic_spaces.viz.ORDER_COLOURS = ('#ffffff', '#ffd166', '#f78c6b', '#ef476f',
    '#b5179e', '#7209b7', '#4361ee')
```

Reflection paths by order, direct sound first; orders past the end reuse the last.

```
acoustic_spaces.viz.DIFFRACTED_COLOUR = '#06d6a0'
```

Paths that bend over an edge.

```
acoustic_spaces.viz.BAND_COLOURS = ('#5e4fa2', '#3288bd', '#66c2a5', '#abdda4',
    '#e6f598', '#fdae61', '#f46d43', '#d53e4f')
```

Octave band colours for decay curves, low to high.

```
acoustic_spaces.viz.LEAK_COLOUR = '#ff4d4d'
```

Open edges and the rays that escape through them.

```
acoustic_spaces.viz.MAX_ESCAPES_DRAWN = 400
```

Escaping rays drawn at most, so a big hole does not hide the room.

```
acoustic_spaces.viz.COMPARISON_COLOURS = ('#4da3ff', '#ffd166')
```

A simulated and a measured response drawn together.

```
acoustic_spaces.viz.VEGETATION_COLOUR = '#5cb85c'
```

Trees, hedges and belts of trees outdoors.

```
acoustic_spaces.viz.WIND_COLOUR = '#4cc9f0'
```

The wind and the weather zones outdoors.

```
acoustic_spaces.viz.WIND_ARROWS = 40
```

Arrows of the wind over the ground in the 3D view, about this many.

```
acoustic_spaces.viz.WIND_REACH = 0.8
```

The longest arrow of the wind, the strongest's, as a share of the space between them.

```
acoustic_spaces.viz.PROFILE_ARROWS = 5
```

Arrows up each mast of the wind's profile.

```
acoustic_spaces.viz.room_polydata(room: Room, faces: np.ndarray | None = None) →
    pyvista.PolyData
```

Room surfaces (all, or the given face indices) as PyVista PolyData.

The `material` cell array holds the index into `room.material_names`.

`acoustic_spaces.viz.paths_polydata(paths: EarlyReflections, order: int | None = None, *, mask: np.ndarray | None = None) → pyvista.PolyData`

The specular paths of one reflection order, or the paths `mask` picks, as polylines.

`class acoustic_spaces.viz.PathGroup(label: str, colour: str, mask: np.ndarray, order: int | None)`

Paths drawn together: specular paths of one `order`, or (order None) those diffracted at an edge.

`label`: `str`

Alias for field number 0

`colour`: `str`

Alias for field number 1

`mask`: `ndarray`

Alias for field number 2

`order`: `int | None`

Alias for field number 3

`acoustic_spaces.viz.path_groups(paths: EarlyReflections, max_order: int | None = None) → list[PathGroup]`

The specular paths by order, then those diffracted at an edge (with up to `max_order` reflections).

`acoustic_spaces.viz.plot_paths(paths: EarlyReflections, plotter: pyvista.Plotter, *, max_order: int | None = None, line_width: float = 2.0) → list[list[str]]`

Draw specular paths coloured by order and diffracted paths in their own colour; returns their legend entries.

`acoustic_spaces.viz.plot_leaks(leaks: Leaks, room: Room, plotter: pyvista.Plotter, *, render: bool = True) → list[list[str]]`

Draw the open edges of `room`, the rays of `leaks` (from `acoustic_spaces.engine.leaks.find_leaks()`) that escape, from their last reflection to a little past where they leave, and a numbered marker on each gap; returns their legend entries.

`acoustic_spaces.viz.balloon_polydata(directivity: Directivity, center, rotation=None, *, band: int = 4, size: float = 1.0, db_range: float = 30.0) → pyvista.PolyData`

A directivity balloon around `center` in one octave band (`band` indexes `OCTAVE_BANDS_HZ`; 4 is 1 kHz): its radius grows from 0 at `db_range` dB below the loudest direction to `size` there. `rotation` has the source's forward, left and up axes as columns (`Orientation.rotation()`). The point data `level` holds the level in dB re the loudest direction.

`acoustic_spaces.viz.extent_polydata(source, radius: float) → pyvista.PolyData | None`

A line source as a tube of `radius` (m), a surface source as its rectangle, or None for a point source.

`acoustic_spaces.viz.nearest_spacing(points) → ndarray`

The distance from each of `points` (rows of coordinates) to the nearest other one: infinite for a point alone.

acoustic_spaces.viz.**clear_lengths**(points, directions, radius, length) → ndarray

The length of the arrow from each of `points` along its unit row of `directions` (zeros for none): `length`, cut short where the arrow would run into the marker (of `radius`) of another point ahead of it, but no shorter than twice its own marker's radius.

acoustic_spaces.viz.**marker_radius**(positions, size: float) → float

The radius of the markers of sources and listeners at `positions` where they do not crowd, in a model `size` metres across: a fifth of their usual spacing (the median distance to the nearest one), from 1.2 % to 2.5 % of the model's size.

acoustic_spaces.viz.**common_radius**(nearest, full: float, lowest: float = 0.2) → float

One radius for every marker, so that markers of one kind look alike: `full` where no two of them would touch, else as small as the closest two need (0.45 of their distance, `nearest` holding each one's distance to its nearest), down to `lowest` times `full` to stay clearly visible.

acoustic_spaces.viz.**marker_sizes**(positions, size: float, directions=None, *, apart=None) → tuple[ndarray, ndarray]

The radius of the marker of each source or listener at `positions` and the length of its arrow along its unit row of `directions`, in a model `size` metres across, from how far apart the points are: a fifth of their usual spacing (the median distance to the nearest one), from 1.2 % to 2.5 % of the model's size, and an arrow six times that. Where points crowd, every marker shrinks alike so that the closest two do not overlap (`common_radius()`), down to a fifth, and an arrow stops short of a marker ahead of it. `apart` (a boolean per point, default all) picks the points that are not to overlap: those in use, say, where the others are drawn faded.

acoustic_spaces.viz.**vegetation_polydata**(shape) → pyvista.PolyData

The volume of a tree, a hedge or a belt of trees (`PlantShape`) as a closed surface: a lone tree's crown, or an outline raised from its bottom to its top.

acoustic_spaces.viz.**plot_vegetation**(room: Room, plotter: pyvista.Plotter, *, render: bool = True) → list

Add an outdoor `room`'s trees, hedges and belts of trees to `plotter`, as see-through green volumes; returns their legend entries.

class acoustic_spaces.viz.**WindMap**(room: Room)

The weather over an outdoor room's ground as the views draw it: the scene's own (index 0) and each weather zone's (1 on), where each holds, and each one's wind at one `height` above the ground, the height the scene gives its wind's speed at, so that the zones' winds compare with the scene's. Zones are polygons in metres, cut to the ground; one that cannot be (its sides cross) or that lies off the ground is None and holds nowhere.

classmethod **of**(room: Room) → WindMap | None

The map of `room`, or None indoors and outdoors without an atmosphere.

property **windy**: bool

Whether the wind blows anywhere at the map's height.

weather_at(points) → ndarray

(n,) which weather holds at each point (n, 2 or 3): 0 for the scene's own, k for its zone k (the first where zones overlap).

centre(k: int) → ndarray | None

(2,) a point inside weather zone `k`: its centroid, or another inside if the centroid is not (an L-shaped zone); None for a zone that holds nowhere.

label(k: int) → str

What weather zone `k` holds, in words: its name, its wind's speed (if the wind blows anywhere) and its ground temperature (if it has one of its own).

arrows(spacing: float | None = None, clear=None, within=None) → WindArrows

The arrows of the wind over the ground, one in the middle of each cell of a grid no more than `spacing` metres apart (by default about `WIND_ARROWS` cells), each the wind of the weather there: the strongest `WIND_REACH` of a cell long, the others in proportion, none where the air is still. Arrows that would pass near one of `clear` (m, 2), where sources and listeners stand, are left out, and a weather zone the grid misses gets one inside it. `within` (x_min, y_min, x_max, y_max) keeps only the cells it overlaps.

class acoustic_spaces.viz.**WindArrows**(middles: np.ndarray, vectors: np.ndarray, weather: np.ndarray, reach: float, cell: float)

The arrows of the wind over the ground (see `WindMap.arrows()`).

`middles`: `ndarray`

(n, 2) the middle of each arrow (m).

`vectors`: `ndarray`

(n, 3) each arrow, horizontal, from its tail to its tip (m).

`weather`: `ndarray`

(n,) the weather each arrow shows: 0 the scene's own, k its zone k.

`reach`: `float`

How long an arrow is per m/s of wind (m).

`cell`: `float`

The size of the grid's cells (m).

```
acoustic_spaces.viz.plot_wind(room: Room, plotter: pyvista.Plotter, *, field: bool =
    True, profile: bool = False, render: bool = True) → list
```

Add the wind over an outdoor `room`'s ground to `plotter`. `field`: sky blue arrows the way it blows, as long as it is strong, at the height the scene gives the wind's speed at above the ground, and each weather zone as see-through walls up to that height with its name and wind. `profile`: a mast in each weather where the wind blows, with arrows at heights up it and a curve through the wind at every height, how the wind grows with height over the ground. Nothing indoors or in still air. Returns the legend entry of the wind.

```
acoustic_spaces.viz.plot_room(room: Room, plotter: pyvista.Plotter | None = None, *,
    opacity: float = 0.45, show_labels: bool = True, paths: EarlyReflections | None =
    None, max_order: int | None = None, coverage: CoverageMap | None = None, parameter:
    str = 'g', band: int = 4, part: str = 'value', leaks: Leaks | None = None, wind:
    bool = True) → pyvista.Plotter
```

Add the room to `plotter` (a new one if None) and return it.

`paths` adds early reflection paths (from `acoustic_spaces.engine.early_paths()` or a response's `early`), coloured by order up to `max_order`. `coverage` adds a coverage map of one `parameter` and `band`, or of a `part` of its spread (see `plot_coverage()`). `leaks` adds the open edges and the rays that escape through them (see `plot_leaks()`). Outdoors, `wind` adds the wind and the weather zones (see `plot_wind()`).

```
acoustic_spaces.viz.show_room(room: Room, screenshot: str | Path | None = None,
    window_size: tuple[int, int] = (1280, 800), **kwargs) → None
```

Open the room in a window, or save a PNG to `screenshot` without one.

Keyword arguments go to `plot_room()` (for example `paths`).

```
acoustic_spaces.viz.animate_particles(room: Room, particles: Particles, path: str | Path
    | None = None, *, fps: int = 25, color: str = 'energy', db_range: float = 40.0,
    window_size: tuple[int, int] = (1024, 640)) → None
```

Show sound particles (from `acoustic_spaces.engine.trace_particles()`) moving through the room: in a window with a time slider, or saved to `path` as an animated GIF or an MP4 video.

`color` is `"energy"` (level in dB relative to a particle's share at emission, down to `-db_range`) or `"order"` (reflections so far).

```
acoustic_spaces.viz.check_video(path: str | Path) → None
```

Raise what saving an animation to `path` would, before the work of making it: ValueError unless it is a .gif or .mp4, ImportError without PyVista or, for an MP4, without imageio and imageio-ffmpeg.

```
acoustic_spaces.viz.animate_walk(room: Room, walk: WalkResponses, path: str | Path, *,
    fps: int = 25, window_size: tuple[int, int] = (1024, 640), sound: str | Path | None
    = None) → None
```

Save a walk (from `acoustic_spaces.walkthrough.simulate_walk()`) as an animated GIF or an MP4 video: the room, the path and the walker going along it, in time with the sound of `render_walk()`. `room` is the scene's room. `sound`, a WAV file of that sound in one or two channels, becomes an MP4's sound track, and the video then lasts as long as the sound (a GIF has none).

```
acoustic_spaces.viz.animate_movement(room: Room, movement: Movement, path: str | Path, *,
    fps: int = 25, window_size: tuple[int, int] = (1024, 640), sound: str | Path | None
    = None) → None
```

Save a movement (from `acoustic_spaces.walkthrough.simulate_movement()`) as an animated GIF or an MP4 video: the room, every mover's path and the movers going along them on one clock, each with an arrow the way it faces, the still sources and listeners where they stand. `sound`, a WAV file of one listener's sound in one or two channels, becomes an MP4's sound track, and the video then lasts as long as the sound (a GIF has none).

```
acoustic_spaces.viz.plot_response(response: Response, path: str | Path | None = None, *,
    band: int = 4)
```

Echogram and decay curves of one response; saved to `path` if given.

The echogram shows the energy per millisecond in one octave band (`band` indexes `OCTAVE_BANDS_HZ`; 4 is 1 kHz) with the early specular paths as stems coloured by order. The decay panel shows the backward-integrated curve of every band. A response in third-octave bands shows them averaged into octave bands. Returns the matplotlib figure.

```
acoustic_spaces.viz.plot_directions(response: Response, path: str | Path | None = None,
    *, band: int = 4)
```

Where the sound comes from, as the listener faces: early reflections (dots sized by level and coloured by delay) and the late energy per arrival direction, in one octave band (`band` indexes `OCTAVE_BANDS_HZ`; 4 is 1 kHz). Azimuth runs from the left to the right of the listener. A response in third-octave bands shows them averaged into octave bands. Returns the matplotlib figure.

```
acoustic_spaces.viz.plot_directivity(directivity: Directivity, path: str | Path | None =
    None, *, db_range: float = 30.0)
```

Polar patterns of a source directivity in every octave band, in dB re the loudest direction of each band: the horizontal plane (seen from above, front at the top and the source's left on the left) and the vertical plane through the front (seen from the side, front on the right and up at the top). Returns the matplotlib figure.

```
acoustic_spaces.viz.plot_atmosphere(atmosphere: Atmosphere, ground_c: float, path: str |
    Path | None = None, *, top: float = 100.0)
```

An outdoor atmosphere from the ground to `top` metres up: the temperature and the wind's speed (left), and the speed of sound (right) in still air and, with a wind, as sound going with it and against it, the speed that bends sound: towards where it is lower. `ground_c` is the temperature at the ground (the scene's air temperature). Returns the matplotlib figure.

```
acoustic_spaces.viz.plot_impulse_response(ir: ndarray, sample_rate: float, path: str |
    Path | None = None, *, title: str | None = None, filters: str = 'iec', noise: bool =
    True)
```

Level and decay curves of a measured (or any) impulse response.

The top panel shows the squared response in dB over 1 ms windows and, per octave band with `noise`, the background noise level found for it. The bottom panel shows each band's backward-integrated decay up to where it sinks into the noise, with the noise taken off and the missing tail added, as the parameters are computed. Returns the matplotlib figure; it is saved to `path` if given.

```
acoustic_spaces.viz.plot_comparison(simulated: ndarray, simulated_rate: float, measured:
    ndarray, measured_rate: float, path: str | Path | None = None, *, title: str | None
    = None, labels: tuple[str, str] = ('simulated', 'measured'), filters: str = 'iec',
    noise: bool = True)
```

Decay curves of a simulated and a measured impulse response, one panel per octave band.

Each panel overlays the backward-integrated (Schroeder) decays of the two, each from its own direct sound on, as `plot_impulse_response()` draws them: up to where the band sinks into its noise, with the noise taken off and the missing tail added. `labels` name the two responses in the legend. Returns the matplotlib figure; it is saved to `path` if given.

```
acoustic_spaces.viz.MAP_COLOURS = 'turbo'
```

Colour map of coverage maps.

```
acoustic_spaces.viz.MAP_OPACITY = 0.7
```

Opacity of a coverage map in 3D: see-through, so the sources and listeners at listening height, half under its tiles, stay in sight.

```
acoustic_spaces.viz.MAP_PARTS = {'at_least': 'at least', 'at_most': 'at most', 'high':
    'upper end of the interval', 'low': 'lower end of the interval', 'standard':
    'standard uncertainty', 'value': 'value'}
```

What a map shows of a parameter: its value, or a part of its spread (`spread_of()`), each with the name it has in a title.

```
acoustic_spaces.viz.map_values(coverage: CoverageMap, parameter: str = 'g', band: int =
    4, part: str = 'value')
```

The values a map shows of `parameter` (in `band` when it has bands, as an index into `OCTAVE_BANDS_HZ`), in the units of its label, and its title. `part` (a key of `MAP_PARTS`) takes, in place of the value, a part of its spread on a map run with one: its standard uncertainty, an end of its interval, or the value it is at least or at most, the last four with their probability in the title.

```
acoustic_spaces.viz.map_polydata(coverage: CoverageMap, parameter: str = 'g', band: int =
    4, part: str = 'value') → pyvista.PolyData
```

The map's columns as tiles at listening height, over the audience surfaces (`tiles()`); the cell data `value` holds the parameter as `map_values()` gives it (NaN where undefined) and `point` the map point of each tile.

```
acoustic_spaces.viz.plot_coverage(coverage: CoverageMap, plotter: pyvista.Plotter, *,
    parameter: str = 'g', band: int = 4, render: bool = True, part: str = 'value')
```

Add a coverage map to a 3D view: coloured tiles at listening height over the audience area, with a colour bar (`parameter`, `band` and `part` as `map_values()` takes them). Returns the actor of the tiles.

```
acoustic_spaces.viz.plot_map(coverage: CoverageMap, path: str | Path | None = None, *,
    parameter: str = 'g', band: int = 4, room: Room | None = None, part: str = 'value')
```

A coverage map seen from above: one panel per level (stalls, balcony, …), columns coloured by `parameter` in `band` (an index into `OCTAVE_BANDS_HZ`; 4 is 1 kHz) and labelled with their values when there is room; with `part`, by that part of its spread (`map_values()`). With `room` the walls and sources are drawn too. Returns the matplotlib figure; it is saved to `path` if given.

Export to Blender (glTF)

Scenes as glTF 2.0 binary files (`.glb`) for Blender and other 3D tools.

The file holds the room, one object per surface name with a see-through material per scene material (its absorption and scattering per octave band as extras, which Blender shows as custom properties), an outdoor scene's trees, hedges and belts of trees as see-through green volumes, and the sources and listeners turned the way they face, with an arrow along the aim. It can also hold early reflection paths as lines, one object per order, and sound particles as an animation in slow motion: by default 100 ms of sound play over 4 s. Blender opens it with File > Import > glTF 2.0.

Scenes are Z-up and glTF is Y-up: a point (x, y, z) is written as (x, z, -y), which the importer turns back.

```
acoustic_spaces.gltf.PARTICLE_COUNT = 1000
```

Particles the command line and the editor export: about 3 MB at 100 frames.

```
acoustic_spaces.gltf.SLOW_MOTION = 40.0
```

Seconds of animation per second of sound.

```
acoustic_spaces.gltf.export_gltf(room: Room, path: str | Path, *, paths: EarlyReflections
    | None = None, particles: Particles | None = None, slow_motion: float = 40.0,
    db_range: float = 60.0) → None
```

Write `room` with its sources and listeners to `path` as glTF 2.0 (`.glb`).

`paths` (from `acoustic_spaces.engine.early_paths()` or a response's `early`) adds the early reflection paths, coloured by order as in the 3D view. `particles` (from `acoustic_spaces.engine.trace_particles()`) adds an animation with a small sphere per particle that moves with it and shrinks with its level: full size at its share of the emitted energy, gone `db_range` dB below. The animation runs `slow_motion` times slower than the sound, with a keyframe at each of the particles' frames.