AmbiRR2

Ambisonic Room Reverb

A 16-source image-source & diffuse-field reverb
in 3rd-order AmbiX (or stereo)

User and Technical Manual

Olivier Doaré, ENSTA
https://perso.ensta.fr/~doare
github.com/odoare
FX-Mechanics

Document version 0.1 — July 10, 2026

Contents

Chapter 1

User guide

1.1 Overview

1.1.1 What AmbiRR2 is

AmbiRR2 is an audio plugin (VST3, AU and Standalone) that places up to 16 mono sources in a virtual parallelepipedic (box) room and renders the room's response as either

- a 1st, 2nd or 3rd-order ambisonic stream in AmbiX format (ACN channel ordering, SN3D normalisation, 4 to 16 channels); or
- a stereo recording made by two virtual microphones placed anywhere in the room, with selectable polar patterns.

It is a rewrite of the earlier AmbiRR with a different DSP structure and a realtime-safe, thread-aware architecture. The physics is split into two mechanisms that are computed and updated independently:

- the direct sound and early reflections, from an image-source model (exact, per-source, and directional);
- the late reverberation, as a diffuse field shared by every source and rendered by fast partitioned convolution.

The whole design turns on one property (Section 1.4): moving a source or the listener never rebuilds an impulse response, so those positions can be automated in your DAW, while the room's dimensions and wall materials are fixed per scene.

1.1.2 Signal flow at a glance

Figure 1.1 shows the per-block processing. Each of the 16 inputs feeds one source through its own multitap delay line (direct + early reflections). The active sources are also summed into a mono diffuse send that drives the shared late-reverb convolution. Both fields are laid down in the room frame; when the output is ambisonic, a final rotation stage turns the whole field into the listener's frame according to the head orientation.

1.1.3 Plugin format and channel layout

- Formats: VST3, AU (macOS), Standalone.
- Bus layout: one 16-channel input bus (the 16 mono sources) and one 16-channel output bus (the ambisonic stream). In stereo mode only output channels 1 and 2 carry signal.
- Plugin type: audio effect (not a synth, no MIDI).

The output is AmbiX (ACN/SN3D) so it drops straight into any ambisonic decoder or renderer that expects that convention (see the primer in Appendix A if the format is new to you).

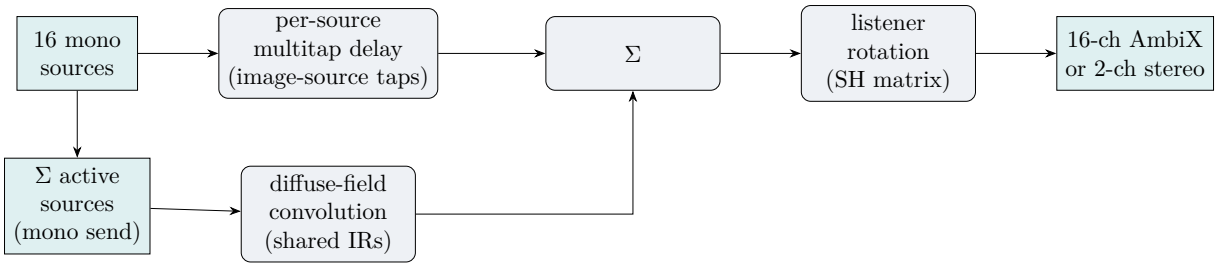

Figure 1.1: Per-block signal flow. The early field is rendered per source by a multitap delay line; the late field is one convolution shared by all sources, fed by their mono sum. In stereo mode the taps carry two microphone signals instead of ambisonic channels, and the rotation stage is bypassed.

1.2 The interface

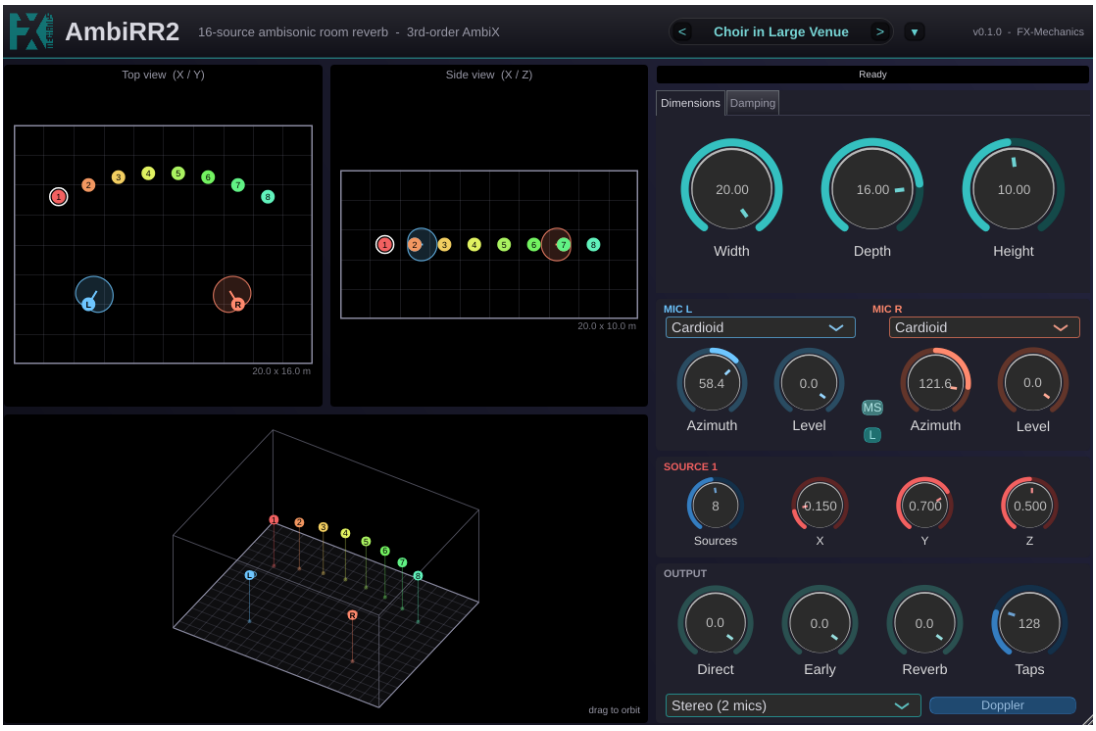

Figure 1.2: The full AmbiRR2 window: header bar on top; the two 2D plane views and the orbiting 3D room on the left; the stacked control column on the right.

1.2.1 Layout

The window has a dark header bar (logo, name, description, version), a left column with the spatial views, and a right column of stacked control panels.

The left column holds, top to bottom, two 2D plane views side by side and an orbiting 3D room view beneath them. The right column holds, top to bottom: a recompute progress bar; a tabbed Room panel; the listener block (or the microphone controls, in stereo mode); the selected source controls; and the Output row.

1.2.2 The 2D plane views

Two draggable maps of the room:

- the Top view looks straight down, drawn "map-style" with the listener's rest orientation (azimuth 0) pointing up: the vertical axis is the room's X (front), the horizontal axis is Y (mirrored, so $+Y$ is to the left as seen from above);
- the Side view shows X (horizontal) against Z (height).

Both show a metre grid and the room dimensions. Sources are numbered coloured dots; the listener is an amber dot with a gaze arrow (the projection of its true look direction, so it shortens as the gaze tilts out of the plane). Drag any dot to move it; the change is written straight back to the parameter.

Mouse-wheel rotation. Rolling the wheel over the listener turns its Azimuth in the Top view and its Elevation in the Side view. Over a microphone (stereo mode) it turns that mic's azimuth.

Clicking a wall. Click one of the four room edges to select that wall: its Damping and HF Damping knobs pop up in a small overlay right at the click, the edge is highlighted in both 2D views, and the matching face is shaded in the 3D view. Click empty space to deselect. All six walls are reachable (the X/Y walls in the Top view, the X walls plus floor and ceiling in the Side view).

1.2.3 The 3D room view

An orbiting orthographic view of the box with a floor grid, the listener (with its gaze arrow), and every active source; a thin stem drops each object to the floor so its height is readable. Drag to orbit the camera. This view is display only (positions are edited in the 2D views). In stereo mode the two microphones are drawn here too, each with a directivity lobe.

1.2.4 The Room panel (tabbed)

Two tabs:

Dimensions
the room Width, Depth and Height in metres (1–20 m per edge).

Damping
the per-wall material grid: six Damping knobs (broadband absorption, 0–0.99) and six HF Damping knobs (extra high-frequency loss). Each row has a Link toggle: while it is on, editing any wall writes the same value to the other five, so a whole room material can be dialled in with one gesture; unlink for per-wall tweaks.

1.2.5 The listener / microphone block

In ambisonic mode this block shows the listener's X, Y, Z position knobs and its Azimuth and Elevation. The active-Sources count sits on the source row below.

In stereo mode the same slot shows the two-microphone controls instead (Section 1.3.2): a polar-Pattern dropdown, an Azimuth knob and a Level knob per mic, plus two small toggles between them: MS (mid/side output) and L (link the pair).

1.2.6 The source controls

The Sources count (how many of the 16 inputs are active) and the X/Y/Z position of the currently selected source. Click a source dot in a 2D view to select it; its knobs re-tint to the source's colour.

1.2.7 The Output row

The three output volumes (Direct, Early for reflections and Reverb for the diffuse tail) plus the Taps count. Beneath them, the mode / order dropdown and the Doppler switch:

- the dropdown selects 1st / 2nd / 3rd order ambisonic output, or Stereo (2 mics);
- Doppler upgrades moving-source smoothing to a true pitch glide (Section 2.6.2).

1.3 Output modes

1.3.1 Ambisonic (AmbiX) output

The default. The room's field is encoded as 3rd-order ambisonics at the listener's position and orientation, in AmbiX (ACN/SN3D) format. Lower orders (1st, 2nd) use fewer channels and are cheaper; the choice is in the Output dropdown. Feed the 16 (or 4, or 9) channels to your ambisonic decoder, binaural renderer or speaker-array decoder.

Because the field is computed in the room frame and only rotated into the head frame at the very end, the listener's Azimuth and Elevation are essentially free to move (turning the head is click-free and never triggers a recompute, Section 2.2.3).

1.3.2 Stereo: two virtual microphones

Selecting Stereo (2 mics) in the Output dropdown replaces the ambisonic encoding with two virtual first-order microphones at their own points in the room. This is a true spaced-pair model: the reflections are computed separately for each microphone, so a spaced pair captures genuine inter-mic time and level differences (AB, ORTF, XY and Blumlein-style recordings all emerge from where you put the mics and how you aim them).

Patterns. Each mic has a first-order polar Pattern: omni, cardioid, supercardioid, hypercardioid or figure-8. The pattern weights every reflection by its direction of arrival; the rear lobe of the tighter patterns (and of the figure-8) is a genuine phase inversion, preserved per reflection.

Placing and aiming. The two mics are dragged in the 2D views like sources (the listener is hidden: the mics are the reception points), and drawn with their directivity lobe in every view. The wheel over a mic turns its azimuth. The L (link) toggle keeps the pair rigid: a drag or an azimuth turn is applied as the same delta to the other mic, so a configured ORTF/AB pair moves and rotates as one object.

Mid/Side. The MS toggle mid/side-encodes the output for M/S recording or downstream width control: channel 1 becomes $\frac{1}{2}(L+R)$ (mid) and channel 2 becomes $\frac{1}{2}(L-R)$ (side). The $\frac{1}{2}$ keeps the matrix unity-gain, so M/S sits at the same level as the L/R feeds.

1.4 Parameters, presets and automation

1.4.1 The Taps control (reverb detail vs. CPU)

Taps sets how much of the early field is rendered as discrete image-source reflections before the smooth diffuse tail takes over. Low values (the default is 64) give a short, economical early field; raising it extends the detailed, directional, per-source reflection pattern deeper into time, at higher CPU. Because the tail is a genuinely diffuse field (below), positional detail in the reverb is bought with Taps, not by making the tail itself position-dependent.

Taps is a target: it fixes the time at which the early field hands over to the diffuse tail. The actual number of reflections rendered varies a little with the room's shape (Section 2.5).

1.4.2 What can be automated (and what cannot)

This is the single most important thing to understand about AmbiRR2. Anything that rebuilds an impulse response cannot be automated: a rebuild is a background job of tens of milliseconds to seconds and cannot follow a control-rate parameter. Everything else can.

The dividing line is clean and deliberate: no position ever reaches an impulse response. Table 1.1 is the summary.

Parameter	What a change rebuilds	Automatable
Listener X/Y/Z	early taps only	yes
Source X/Y/Z (each)	that source's early taps	yes
Mic positions / azimuth / pattern	early taps only	yes
Listener Azimuth / Elevation	nothing (realtime rotation)	yes
All Levels, MS, Doppler	nothing	yes
Room Width/Depth/Height	early taps and the reverb IRs	no
Damping / HF Damping	early taps and the reverb IRs	no
Taps count	early taps and the reverb IRs	no
Order / output mode	early taps and the reverb IRs	no

Table 1.1: Automation map. Listener and source positions are fully automatable; the room's shape and materials are set once per scene.

So, in practice: set the room dimensions and wall materials once for a scene, then automate the listener and the sources moving around inside it. If you hear the reverb "freeze" or the CPU spike while automating, you are almost certainly automating a room parameter by mistake.

1.4.3 Why the reverb tail does not follow the sources

The late reverberation is modelled as a diffuse field: above the Schroeder frequency the reverberant energy in a room depends on the source's power, not on where it stands, and it is statistically the same everywhere in the room. AmbiRR2 therefore shares one set of reverb impulse responses across all sources and never rebuilds them when something moves. The direct-to-reverberant balance still changes correctly with distance, because the early reflections carry the $1/r$ distance law while the reverb send does not.

The one thing this model gives up is per-source decorrelation of the tail: every source's reverb is the same noise realisation. In practice this is inaudible against the fully position-dependent early field, and it is what buys the automation and the low CPU.

1.4.4 Levels

Direct, Early and Reverb are independent output trims (-90 dB to 6 dB). Direct is the level of the direct sound (the un-reflected image), Early the level of all the discrete reflections, and Reverb the level of the diffuse tail. They are smoothed, so automating them is click-free.

1.5 Workflows

1.5.1 A: A moving source in a fixed room (ambisonic)

1. Set Order to 3rd, and dial the room Dimensions and wall Damping for the space you want. Leave them alone from here.
2. Place the listener; set Reverb to taste against Direct and Early.
3. Automate a source's X/Y/Z (draw it in your DAW or drag the dot while recording automation). The early reflections and their directions follow smoothly; the tail stays put.
4. Feed the 16 output channels to your ambisonic decoder / binaural renderer.

1.5.2 B: Fast movement with Doppler

As above, but turn on Doppler. Now a source swept quickly towards or away from the listener produces a genuine pitch glide on the direct sound and each reflection (each with its own radial velocity). Costs roughly twice the early-reflection CPU while things move; leave it off for static or slow scenes.

1.5.3 C: A stereo room recording

1. Set the output mode to Stereo (2 mics).
2. Choose a technique: a spaced omni pair (AB), a near-coincident cardioid pair toed out (ORTF), coincident figure-8s at 90° (Blumlein)...
3. Turn on L to move/aim the pair as a unit while you audition placements; turn it off for fine per-mic tweaks.
4. Use MS if you want a mid/side feed to width-control downstream.

1.5.4 D: Turning the head

In ambisonic mode, automate the listener's Azimuth and Elevation freely (these are realtime and never rebuild anything), so head-tracking-style motion is smooth at any speed.

1.5.5 E: Realistic binaural scenes

Because AmbiRR2 outputs 3rd-order AmbiX, it feeds a binaural decoder directly, and the combination makes a convincing headphone room: the image-source early reflections give the ear the geometry cues it uses to localise and to judge distance, and the diffuse tail supplies the enveloping reverberation, all decoded to the two ears through head-related transfer functions.

A decoder that works very well here is the BinauralDecoder from the IEM Plugin Suite (`plugins.iem.at`, free and open source). Place it in the same channel strip, immediately after AmbiRR2:

1. Build the scene in AmbiRR2 as in workflow A (fix the room, place the listener, position and automate the sources).
2. Set Order to 3rd (16 channels) for the sharpest imaging; lower orders also decode, with softer localisation.
3. Insert BinauralDecoder right after AmbiRR2 on a 16-channel track, and check that its order and normalisation are set to 3rd-order AmbiX (ACN/SN3D) to match AmbiRR2's output.
4. Monitor the decoder's stereo output on headphones.

Everything that is automatable in AmbiRR2 stays automatable through the decoder: move the sources, and turn the listener's Azimuth and Elevation (workflow D), and the binaural image follows in real time. This is the most direct way to audition an AmbiRR2 scene without an ambisonic loudspeaker rig.

Chapter 2

Technical reference

This chapter documents the DSP as implemented. Code references are given as `File::symbol`; the reusable maths lives in the FxmeTools shared library (`lib/FxmeTools/FxmeTools/dsp/`), the plugin-specific orchestration in `Source/Dsp/`. It mirrors, and is kept in step with, `doc/implementation.md`.

2.1 Architecture and threading

2.1.1 Two computation paths

	Early field	Late field
Model	image-source method	statistically diffuse field
Realtime unit	multitap delay line per source	partitioned convolution (WDL)
Recomputed	a source / listener / room change	a room change only
Position dep.	full (per source, per listener)	none (shared by all sources)
Orientation	not baked in (applied by rotation)	not baked in (isotropic)
Automatable	yes (Sec. 2.5, 1.4.2)	n/a (it never moves)

Table 2.1: The two independent paths that feed the realtime renderer.

2.1.2 Thread model

Four kinds of thread touch the engine (`AmbiRoomEngine`):

Audio thread
`process()` only. Never locks (bar one non-blocking try-lock), never allocates. All per-tap and per-source state is preallocated in `prepare()`.

Message thread
parameter push at 30 Hz: copies the APVTS into POD snapshots, updates atomics, rebuilds the rotation matrix on orientation changes.

Worker thread
woken on dirty flags; snapshots the request, fans per-source tap jobs out to the pool, waits, then rebuilds the late IRs only if the room changed.

Thread pool
(`physical cores − 1`) one job per dirty source, plus one job per channel of the diffuse IR.

Hand-offs are lock-free. Tap sets and the rotation matrix cross the message/worker → audio boundary through a single-slot single-producer/single-consumer mailbox (`SpscMailbox`), an `std::atomic<bool>` with acquire/release ordering and a pointer-cheap `std::swap` of the payload. Levels and flags are plain atomics. The rare diffuse-IR swap is guarded by a spin try-lock, so the audio thread skips a block rather than ever blocking.

2.1.3 Split invalidation

What a parameter change invalidates is split by two predicates on the parameter snapshot (`RoomParams`), and this split is exactly what makes automation possible (Section 1.4.2):

`tapsEquals`
: false ⇒ rebuild the early taps. Sensitive to the room and every position. Cheap, per-source, smoothed on arrival, so a parameter that only breaks this is automatable.

`reverbEquals`
: false ⇒ rebuild the diffuse IRs. Sensitive to the room alone (no position appears in it). A parameter that breaks this swaps the convolver's IR and is not automatable.

A per-source dirty flag additionally lets a single source's move recompute only that source's taps.

2.2 Ambisonic implementation

All ambisonic maths is in `FxmeTools/dsp/Ambisonics.h` (header-only, JUCE-free). It uses the AmbiX conventions: ACN ordering $\text{acn} = l(l+1) + m$ for degree l and order m; SN3D normalisation; a right-handed frame with x = front, y = left, z = up.

2.2.1 Encoding

`encodeSN3D` evaluates the real spherical harmonics up to degree 3 directly as polynomials of the unit direction (x, y, z), with no trigonometry at encode time. Degree 1 is $(Y, Z, X) = (y, z, x)$; degree 2 includes $\sqrt{3}\,xy$, $(3z^2 - 1)/2$, $\frac{\sqrt{3}}{2}(x^2 - y^2)$; and so on. The W channel is 1. Each early-reflection tap stores its 16 encoding gains premultiplied with its broadband gain and crossfade factor, so the realtime cost is 16 multiply-accumulates per tap per sample.

2.2.2 Diffuse-field order weighting

For an isotropic diffuse field in SN3D the per-degree channel energy must scale as $1/(2l+1)$, so the late IRs are weighted by

$$g_{\text{diff}}(l) = \frac{1}{\sqrt{2l+1}}, \tag{2.1}$$

which keeps the reverb level consistent across output orders and with the encoded early field.

2.2.3 Listener rotation (SH rotation matrices)

Head rotation must not trigger a recompute, so everything is computed in the room frame and rotated at the output:

1. the listener's (azimuth, elevation) becomes a 3×3 rotation whose rows are the head's forward/left/up axes in room coordinates (`headRotation`; a roll argument exists but is not yet exposed);

2. that matrix is lifted to a real spherical-harmonic rotation with the Ivanic–Ruedenberg recurrence [1] (degree 1 is a re-indexed copy of the 3×3 matrix, each higher degree built recursively from degree 1 and degree $l-1$; `ShRotation::fromMatrix`).

The matrix is block-diagonal per degree (rotations never mix degrees), so rotating a 16-channel sample costs $3^2 + 5^2 + 7^2 = 83$ multiplies; W passes through. The construction satisfies

$$\text{encodeSN3D}(M\mathbf{d}) = \text{ShRotation}(M)\ \text{encodeSN3D}(\mathbf{d}), \tag{2.2}$$

verified numerically against direct re-encoding over 2000 random rotations (max error $\approx 8 \times 10^{-7}$), plus per-block orthogonality and identity exactness. At runtime the message thread rebuilds the matrix on azimuth/elevation change and publishes it lock-free; the audio thread cross-fades from the old matrix to the new one over exactly one block (both applied, outputs interpolated), which is click-free at the 30 Hz push rate.

2.2.4 Stereo mode: virtual microphones

The stereo mode replaces the SN3D encoding with two first-order microphones rendered to channels 0 (L) and 1 (R). The polar family is

$$g(\theta) = \alpha + (1-\alpha)\cos\theta, \tag{2.3}$$

with $\alpha = 1$ (omni), 0.5 (cardioid), 0.366 (supercardioid), 0.25 (hypercardioid), 0 (figure-8); θ is the angle between the mic axis and the arrival direction. Gains are signed, so rear lobes keep their phase inversion. The image-source enumeration runs once per microphone position, so a spaced pair carries true inter-mic time and level differences. The rotation stage is bypassed (orientation is meaningless for a mic pair); the mid/side matrix, when engaged, is applied last as $(\frac{1}{2}(L+R),\ \frac{1}{2}(L-R))$.

2.3 The early field: image-source model

The room model is in `FxmeTools/dsp/RoomAcoustics.h` (header-only, JUCE-free).

2.3.1 Image enumeration

For a box $[0, r_x] \times [0, r_y] \times [0, r_z]$ the image of a source at $\mathbf{s}$ for grid index (i_x, i_y, i_z) is, per axis,

$$x_{\text{img}} = 2\lceil i_x/2 \rceil\, r_x + (\, i_x \text{ odd } ?\ -s_x : s_x\,), \qquad \text{likewise } y, z. \tag{2.4}$$

The enumeration bound is always a radius in metres, converted per axis ($\text{steps(dim)} = \lceil r/\text{dim} \rceil + 1$), never a cube of grid indices, because the image lattice inherits the room's aspect ratio and a cube badly under-samples the thin axis of a flat or tall room. The radius follows the truncation policy (Section 2.5.3).

2.3.2 Per-image acoustics

Each image contributes one tap:

- delay = $\text{round}(\text{dist}/c \cdot f_s)$ samples;
- broadband gain = $(\prod_w (1-\text{damp}_w)^{n_w})/\text{dist}$, combining per-wall amplitude reflection and $1/r$ spreading, where n_w is the number of reflections off wall w (derived exactly from $|i|$ and the sign of the index);
- HF damping: each wall hit adds its hfDamp_w to a sum H, giving a one-pole low-pass with cutoff $\omega = 2\pi \cdot 20\,\text{kHz} \cdot e^{-H}$ (transparent at $H = 0$);
- direction of arrival, a unit vector in the room frame, encoded by `encodeSN3D` (ambisonic) or weighted by the polar pattern (stereo);
- a stable id packing (i_x, i_y, i_z), so the same physical reflection keeps its identity across recomputes (the key to the Doppler renderer, Section 2.6.2).

2.3.3 Reverberation time

Damping is an amplitude loss (images reflect with the factor $1-\text{damp}$), so the fraction of energy absorbed per reflection is $\alpha = 1-(1-\text{damp})^2$ (not damp). Both RT60 formulae take α:

$$\text{Sabine:}\quad \mathrm{RT}_{60} = \frac{0.161\,V}{\sum_w S_w\,\alpha_w}, \tag{2.5}$$

$$\text{Eyring:}\quad \mathrm{RT}_{60} = \frac{0.161\,V}{-\sum_w S_w \ln(1-\alpha_w)} = \frac{0.161\,V}{-2\sum_w S_w \ln(1-\text{damp}_w)}. \tag{2.6}$$

The plugin uses Eyring (clamped to $[0.05, 8]$ s), because the diffuse tail has to continue an image-source field whose energy decays exactly as $(1-\text{damp})^{2n}$; Sabine on the raw Damping made the tail decay 2.2× too slowly, so it sat under the early field as a separate, slower event rather than continuing it.

2.4 The late field: diffuse convolution

The late reverb is one set of 16 impulse responses (`LateReverb`), one per ambisonic channel (two in stereo mode), shared by every source and driven by the mono sum of the active sources. Convolution uses Cockos' WDL engine (`WDL_ConvolutionEngine_Div`), a partitioned FFT convolver.

2.4.1 Why one shared, static IR is correct

Above the Schroeder frequency the reverberant field is diffuse: its level follows the source's power, not its distance, and it is statistically uniform in the room. So a single position-independent IR is not an approximation of convenience but the right model, and it is what lets the IR stay static while sources and the listener move (Section 1.4.2). The direct-to-reverberant ratio still tracks distance because the early taps carry $1/r$ and the reverb send does not.

2.4.2 IR construction

Per channel: decaying, per-channel-decorrelated noise. The channels use distinct deterministic seeds (reproducible decorrelation), a one-pole low-pass whose cutoff follows the average HF damping, and the envelope e^{-ki} with k from RT_{60} (Section 2.3.3) referenced to absolute time $i = 0$.

Physical amplitude. The IR amplitude is not a normalisation of convenience; it is set so the diffuse field continues the early field's energy. An image shell at radius $r = ct$, thickness $c\,dt$, holds $4\pi r^2 c\,dt/V$ images (density $1/V$) each of energy $(\text{refl}/r)^2$, so the early field's energy density is $4\pi c\,\text{refl}^2/V$ per unit time (i.e. $4\pi c/(V f_s)$ per sample at $t = 0$). With uniform noise of variance $\frac{1}{3}$ through a one-pole low-pass of steady-state variance $\mathrm{var}_{\mathrm{lp}}$,

$$\text{norm}^2 = \frac{4\pi c}{V\,f_s\,\mathrm{var}_{\mathrm{lp,ref}}}, \qquad \text{weighted per degree by } \tfrac{1}{\sqrt{2l+1}}. \tag{2.7}$$

Verified against the measured image-source density: within ±1.2 dB (mostly ±0.5 dB) across $18\,\mathrm{m}^3$ to $4320\,\mathrm{m}^3$ and 64–512 taps. (The earlier unit-total-energy scale $\sqrt{2k}$ made the density $\propto A/V$, so the reverb drifted 15 dB with room size relative to the early field.)

2.5 The early ↔ late crossover

This is the join between the discrete early taps and the diffuse tail, and getting it seamless and position-independent at once is the subtlest part of the design. `Source/Dsp/Crossover.h` owns one window [crossStart, crossEnd] that both sides use.

2.5.1 An exact, power-complementary join

Over the window,

$$\text{early taps:}\quad \text{gain 1 before crossStart, linear to 0 at crossEnd, truncated by time at crossEnd;} \tag{2.8}$$

$$\text{diffuse IR:}\quad \text{silent before crossStart, } \sqrt{w(2-w)} \text{ up to 1 at crossEnd,} \tag{2.9}$$

where $w \in [0, 1]$ is the normalised position in the window. These two are power-complementary:

$$(1-w)^2 + w(2-w) = 1. \tag{2.10}$$

The taps and the noise are uncorrelated, so their energies add and the sum is unity at every instant (no hole, no bump, at any room shape or source / listener placement). (A naïve linear w fade-in would leave $(1-w)^2 + w^2 = \frac{1}{2}$ at the midpoint: a 3 dB dip.) This only holds because the taps are truncated by time, not by count.

2.5.2 Fixing the window from the room alone

Two constraints set the window, both position-independent (which is what keeps the IR static and the positions automatable):

- The tap target sets the time. Taps $= N$ is a target image count, evaluated once at a canonical reference geometry; the arrival of the N-th tap there gives crossEnd, and crossStart $= 0.9 \cdot$ crossEnd.
- The crossover can never precede the direct sound. A source in the far corner has its direct path arrive at diagonal$/c$. Were crossStart earlier, that direct path would be faded away (impossible, since the diffuse field is caused by it). So the window is pushed out until crossStart $\geq 1.05 \cdot$ diagonal$/c$.

2.5.3 Time truncation and the tap count

Because the taps are truncated by time at crossEnd, their count varies with room shape and placement (measured 1.0–1.5× the target at 256–512 taps, more at low targets where the diagonal clamp dominates). The delay-line and Doppler buffers carry headroom for this (`maxTapsPerPoint`); the worst case measured was 572 taps for a "512" target.

2.5.4 Bug history

The crossover took three iterations to get right, recorded here so the failures are not repeated:

1. A fixed 20 ms floor on the fade span pushed "diffuse full" ∼19 ms past the last tap in every room (an audible hole).
2. A closed-form $r = \sqrt[3]{3nV/4\pi}$ estimate was wrong twice: it ignored that stereo halved the per-mic tap budget (a 50 ms hole at 512 taps in stereo, none in ambisonic; exactly how the bug was reported), and it assumed a spherical image set while the enumeration walked a cube of indices, firing the diffuse ∼25 ms early in flat rooms.
3. Even taking the window from the real enumeration, the join could not be exact while the taps faded by tap index (position-dependent) and the IR by absolute time (position-independent). Only fading the taps by time, over the shared window, makes it exact.

The enumeration is now radius-derived per axis in both its time- and count-bounded forms; fixing only the time-bounded path once left the reference window wrong and inflated the tap count 4× in a flat slab room.

2.6 Tap-update smoothing

Moving a source (or the listener, or a wall) recomputes its early taps on the worker thread; the new set replaces the live one on the audio thread. A raw swap would click, so two modes smooth it (`EarlyReflections`).

2.6.1 Cross-fade (default)

On arrival, the outgoing set (and its per-tap filter states) moves to a fading slot, and for ∼20 ms both sets are rendered from the same delay line with complementary linear ramps. Old and new are strongly correlated (same source, slightly moved), so an amplitude-complementary fade is correct. Cost: one extra tap-set render per source, only during the fade. Fast continuous motion becomes a chain of 20 ms fades between snapshots (click-free, but with no pitch shift).

2.6.2 Doppler (optional)

Per-tap parameter interpolation with fractional delays. Taps are matched across recomputes by their stable id; each tap's fractional delay and gains glide to their new targets over ∼40 ms, and the delay line is read with linear interpolation. A sliding delay resamples the source, so a fast move produces a genuine Doppler pitch shift on the direct sound and on every reflection (each with its own radial velocity). Mode transitions are click-free both ways. Cost: roughly 2× the static per-tap work while moving, plus preallocated per-tap state (hence the switch).

2.7 Memory and CPU

At 48 kHz, defaults in parentheses:

- Delay lines: 16× next-pow-2(2 s · f_s) floats ≈ 8 MB.
- Diffuse IRs: ≤ 16 ch × 8 s × f_s floats ≈ 24 MB in the convolver at the RT_{60} cap (usually far less). One instance, always.
- Per-sample audio cost: taps × sources × (order+1)^2 MACs for the early field, 83 multiplies for the rotation, and the partitioned convolution for the tail. Cross-fades double the early cost transiently; Doppler roughly doubles it while anything moves.

Appendix A

Ambisonics primer

If AmbiX, ACN and SN3D are new, this is the minimum you need.

A.0.1 The idea

Ambisonics does not carry loudspeaker feeds; it carries a description of the sound field itself, as a set of coefficients on the spherical harmonics. A decoder later turns those coefficients into feeds for whatever loudspeaker rig (or headphones, binaurally) you actually have. AmbiRR2 produces the field description; you supply the decoder.

A.0.2 Spherical harmonics and order

The spherical harmonics Y_l^m are a basis for functions on the sphere, organised by degree $l = 0, 1, 2, \ldots$ and order $m = -l, \ldots, l$. Degree 0 is the omnidirectional pressure (W); degree 1 is the three figure-8s along the axes (X, Y, Z); higher degrees are finer angular lobes. An ambisonic stream of order L keeps every degree up to L, so it has $(L+1)^2$ channels: 4 (1st), 9 (2nd), 16 (3rd). Higher order = sharper spatial resolution = more channels.

A.0.3 AmbiX: ACN and SN3D

A convention has to fix two things: the channel order and the normalisation. AmbiX uses

ACN
: Ambisonic Channel Number: channel acn $= l(l+1) + m$, so W=0, then Y=1, Z=2, X=3, and so on.

SN3D
: Schmidt semi-normalisation: a specific per-degree scaling of the harmonics that keeps the numbers well-behaved (the W gain of a plane wave is 1). AmbiRR2 encodes directly in SN3D (Section 2.2.2).

Feed the 16 channels, in this order and normalisation, to any AmbiX decoder.

Appendix B

Glossary

AmbiX
The ambisonic convention AmbiRR2 outputs: ACN channel ordering with SN3D normalisation (Appendix A).

Diffuse field
The statistical model of late reverberation: energy depends on source power not position, uniform in the room, valid above the Schroeder frequency. The basis for one shared, static reverb IR (Section 2.4).

Doppler mode
Optional per-tap interpolation that produces a true pitch glide on fast moves, instead of a cross-fade between static snapshots (Section 2.6.2).

Early reflections
The discrete image-source taps: direct sound plus a controllable number of reflections (Taps), each with a delay, HF damping and a direction of arrival (Section 2.3).

Eyring formula
The reverberation-time formula AmbiRR2 uses, correct at high absorption, fed the energy absorption $\alpha = 1 - (1 - d)^2$ (Section 2.3.3).

Image-source method
Modelling reflections as mirror-image copies of the source across the walls (Section 2.3).

Schroeder frequency
The rough boundary (typically 150 Hz to 300 Hz domestically) above which a room's response is statistically diffuse rather than a few discrete modes.

SH rotation matrix
The block-diagonal-per-degree matrix that rotates an ambisonic field, used to apply head orientation in realtime without a recompute (Section 2.2.3).

Tap
One image source as rendered by the delay line: a delay, an HF one-pole, and its encoding gains (Section 2.3).

Crossover window
The shared [crossStart, crossEnd] over which the early taps fade out (by time) and the diffuse tail fades in power-complementarily (Section 2.5).

Appendix C

Building from source

JUCE is expected as a sibling of the repository; FxmeTools (which bundles the WDL it needs) is an in-tree submodule:

```
git submodule update --init --recursive
cmake -B build -DCMAKE_BUILD_TYPE=Release
cmake --build build -j
```

with the layout `../JUCE/` and `lib/FxmeTools/`. Targets: `AmbiRR2_Standalone`, `AmbiRR2_VST3`, `AmbiRR2_AU` (macOS).

C.0.1 License

C.0.2 Acknowledgements

The late-reverb tail is rendered by the fast partitioned convolution engine of WDL (`WDL_ConvolutionEngine_Div`), the same engine used by REAPER's ReaVerb. WDL is written and maintained by Cockos Incorporated (© 2006 and later) and is distributed under the permissive zlib license. Warm thanks to Justin Frankel and the Cockos team for releasing it. Project page: https://www.cockos.com/wdl/; source: https://github.com/justinfrankel/WDL.

Bibliography

[1] J. Ivanic and K. Ruedenberg, "Rotation Matrices for Real Spherical Harmonics. Direct Determination by Recursion," J. Phys. Chem. 100(15), 6342–6347, 1996 (corrections 102(45), 1998).

[2] H. Kuttruff, Room Acoustics, 6th ed., CRC Press, 2016.

[3] J. B. Allen and D. A. Berkley, "Image method for efficiently simulating small-room acoustics," J. Acoust. Soc. Am. 65(4), 943–950, 1979.