

A FREQUENCY-DOMAIN ARTIFICIAL REVERBERATOR PLUG-IN

Jonas Roth, Nishanth Kumar, Silvan Krebs, David Wieland, and Christoph Studer

Department of Information Technology and Electrical Engineering, ETH Zurich, Switzerland

joroth@ethz.ch | studer@ethz.ch

ABSTRACT

We present *FDverb*, a frequency-domain artificial reverberator, based on the idea of a vocoder with a noise carrier signal. Using a short-time Fourier transform (STFT) for analysis and synthesis, *FDverb* generates late reverberation by weighting spectral noise components with envelopes. We extend *FDverb* with early reflections, nonlinear decay, and pitch shifting. These extensions enable creative sound-design applications. We provide *FDverb* as an open-source DAW plug-in, using the JUCE framework.

1. INTRODUCTION

Artificial reverberation is a fundamental tool in audio production. Although traditionally used to simulate the acoustic properties of physical spaces, artificial reverberators are increasingly valued as creative effects that produce reverberation with no physical counterpart. The acoustic response of a physical room is typically divided into two parts: (i) early reflections (ERs), where individual reflections are still distinguishable, and (ii) late reverberation tail, where the reflection density is high and the response takes on a stochastic character. As observed in [1], some impulse responses of concert halls sound remarkably similar to white noise with an exponential decay of amplitude. This observation motivated synthetic reverberation methods such as [2, 3], which model late reverberation as noise with frequency-dependent decay. More specifically, [2] describes a frequency-domain reverberator based on recursive spectral magnitude decay, derived from the phase vocoder. Related, [3] adopts bandwise exponentially decaying noise as an ideal reference and proposes a computationally efficient frequency-domain reverberator that is perceptually equivalent to the ideal reference. There also exists a number of so-called “spectral processing” DAW plug-ins, such as [4], that seem to implement similar techniques, but technical details are not available.

1.1. Contributions

Inspired by the idea of a vocoder with a noise carrier, we derive a late reverberation generator, further referred to as *tail reverberator*. The tail reverberator operates in the frequency domain using a short-time Fourier transform (STFT), and is conceptually related to the spectral magnitude decay reverberator of [2]. In addition, we implement nonlinear decay and pitch shifting for the tail reverberator, which enables creative sound-design applications. Finally, we combine the tail reverberator with an early-reflection unit; together, they form our proposed hybrid reverberation algorithm, which we

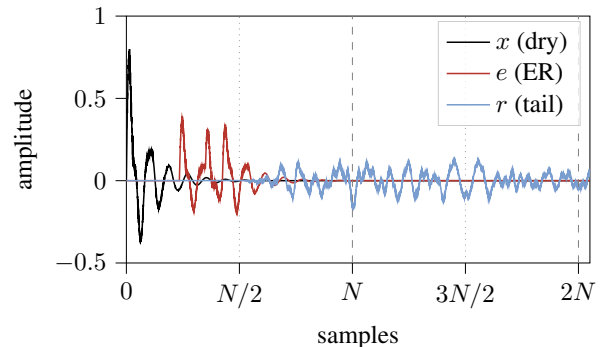


Figure 1: Illustration of signal components of *FDverb* applied to a drum signal: (black) input x , (red) early reflections e , and (blue) reverb tail r . The dashed and dotted vertical lines mark the boundaries of the overlapping blocks processed by the tail reverberator (here, we used a block size of $N = 8192$).

call *FDverb*. An early version of *FDverb* was demonstrated on *gratli*, a microcontroller-based audio DSP platform [5], but the underlying algorithm was not described. We provide *FDverb* as an open-source DAW plug-in together with a Python reference implementation, both available in the *FDverb* Git repository.¹

2. THE FDVERB ALGORITHM

We now explain the hybrid *FDverb* algorithm that contains an ER unit and a tail reverberator, as depicted in Figure 2. *FDverb* receives a discrete-time *dry* input signal $x[n]$, $n \in \mathbb{Z}$ (we will use the shorthand x). The ER unit generates the ER signal e by convolving x with a sparse impulse response, as described in Section 2.1. The tail reverberator generates the tail signal r by shaping the spectrum of a noise source to match the spectrum of the input signal x , as described in Section 2.2. The *FDverb* output signal y is a weighted sum of x , e , and a delayed version of r ; Figure 1 illustrates these three components for a drum signal. In the following discussion, we display the parameters of the *FDverb* user interface in a monospaced font, e.g., `parameter-name`.

2.1. Early Reflections

We obtain the ER signal e by convolving the input x with a sparse early-reflection impulse response (ER-IR) h_{ER} . We implement the convolution in the time domain to avoid the block latency inherent to frequency-domain processing, and constrain the impulse response h_{ER} to be sparse and of finite length, in order to limit both the computational cost and the memory usage.

¹<https://github.com/IIP-Group/FDverb/>

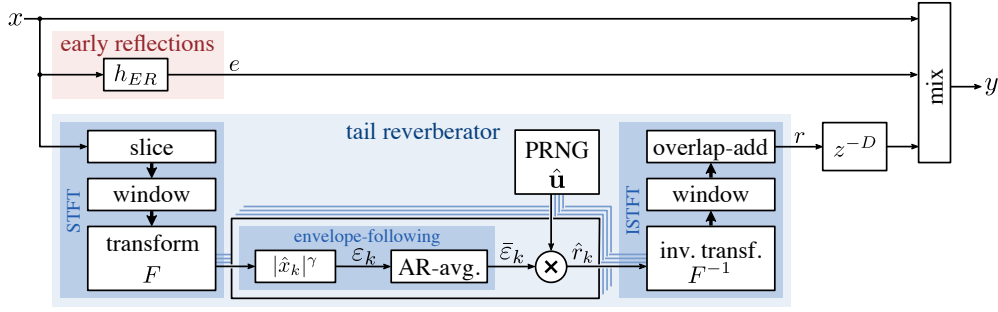


Figure 2: Block diagram for FDverb.

More specifically, we synthesize h_{ER} based on the following parameters: (i) E (echoes), the number of early reflections to be generated; (ii) T_{pre} (pre-delay), the delay of the first early reflection; (iii) T_{tail} (tail-delay), the maximum delay across all early reflections; and (iv) $T_{60,ER}$, the ER decay time².

First, we draw the tap delays randomly from a quadratic distribution over the interval $[T_{pre}, T_{tail}]$, except the first tap, which is fixed at T_{pre} . Based on the tap delays, we compute the tap gains to decay exponentially, with decay time $T_{60,ER}$. We implement the random draw of the tap delays using a pseudo-random number generator (PRNG) and use T_{pre} as the seed for the PRNG, so that the user gets repeatable results for the impulse response h_{ER} .

2.2. Reverb-Tail Algorithm

At a high level, the FDverb tail reverberator implements the following steps:

1. Apply STFT to the tail-reverberator input signal x .
2. For each spectral bin k , extract the magnitude envelope $\bar{\varepsilon}_k$.
3. Generate the noise carrier spectrum $\hat{u}$.
4. For each spectral bin, modulate the amplitude of the noise with the envelope, to get the reverb-tail spectrum bins $\hat{r}_k$.
5. Apply inverse STFT (ISTFT) to $\hat{r}$, which leads to the tail-reverberator output signal r .

In the following, we detail a mono implementation of FDverb; later we will extend FDverb to a stereo implementation.

2.2.1. Forward Short-Time Fourier Transform

Consider the signal x , which enters the tail algorithm on the left side in Figure 2. We realize the STFT by slicing the input signal into overlapping blocks of N samples, which we denote as $\mathbf{x}[\ell] \in \mathbb{R}^N$, where ℓ is used to index the blocks over time. We use an overlap of $\frac{N}{2}$ between the blocks and, for simplicity, constrain N to powers of two. To reduce blocking artifacts, we apply a sine window to each block. Next, we apply a spectral transform $F(\mathbf{x}[\ell]) = \hat{\mathbf{x}}[\ell] \in \mathbb{C}^K$ from N time-domain samples to K frequency bins. For our explanation, we consider the case of a unitary *discrete Fourier transform* (DFT), where we only consider the non-negative frequency bins, so $K = \frac{N}{2} + 1$.³

²In the FDverb plug-in, $T_{60,ER}$ is not exposed in the user interface, but is derived from the nominal decay time parameter instead.

³When implemented using a fast Fourier transform (FFT) algorithm, the above transform is sometimes referred to as a “real-FFT” (RFFT).

2.2.2. Envelope-Following

We now perform envelope-following for each bin $\hat{x}_k, k = 1, \dots, K$. The envelope follower comprises a *detector* followed by an *averager*. The detector is implemented as follows. For each block ℓ , we compute the detector signal $\varepsilon_k[\ell] \in \mathbb{R}$ for the bin k as

$$\varepsilon_k[\ell] = |\hat{x}_k[\ell]|^\gamma, \quad (1)$$

where $|\cdot|$ denotes the magnitude and $\gamma > 0$ shapes the amplitude dynamics of the reverb tail. The default value $\gamma = 1$ corresponds to amplitude detection. In the user interface, we refer to γ as the `color` parameter.⁴

The averager is implemented as a so-called *attack-release (AR)-averager*: a first-order IIR low-pass filter with two distinct time constants for the attack and release phases of the signal $\bar{\varepsilon}_k[\ell]$, as described in [6, 7]. For each block ℓ , we compute the AR-averager signal $\bar{\varepsilon}_k \in \mathbb{R}$ for the bin k as

$$\bar{\varepsilon}_k[\ell] = \begin{cases} \alpha_k \varepsilon_k[\ell] + (1 - \alpha_k) \bar{\varepsilon}_k[\ell - 1] & \text{if } \varepsilon_k[\ell] \geq \bar{\varepsilon}_k[\ell - 1] \\ \beta_k \varepsilon_k[\ell] + (1 - \beta_k) \bar{\varepsilon}_k[\ell - 1] & \text{if } \varepsilon_k[\ell] < \bar{\varepsilon}_k[\ell - 1], \end{cases} \quad (2)$$

where the coefficients $\alpha_k, \beta_k \in (0, 1)$ control the attack- and release-time constants of the envelope, respectively.

Through the user interface, the release-time coefficients β_k are controlled as follows. The `decay` parameter specifies a nominal decay time constant (given as T_{60}). We then use the `tilt` parameter to derive a frequency-dependent decay time constant $T_{60}(k)$ for each bin, from which we determine β_k .⁵ The attack-time coefficient, on the other hand, is not exposed to the user: we found that varying the attack time does not noticeably affect the character of the reverb tail. We therefore fix $\alpha_k = \alpha$, independent of frequency, to achieve an attack time of 10 ms.

Figure 3 illustrates the effect of the envelope follower on an input signal containing a sequence of organ chords, whose distinct harmonics are visible as horizontal bands in the spectrogram. Comparing the input (top) and output (bottom), the AR-averager smooths the signal in time, i.e., along the horizontal direction.

2.2.3. Spectral Noise Generation and Weighting

In a conventional vocoder-reverberator setup, one would generate noise in the time domain and apply the same spectral decomposition

⁴When the color parameter is set to $\gamma \neq 1$, then the detector also influences the spectral balance of the tail signal, since typical input signals have a $1/f$ spectral distribution.

⁵We set $\beta_k = 1 - 10^{\frac{-3 N/2}{T_{60}(k)}}$, where $T_{60}(k) \in (0, 2 T_{60}]$.

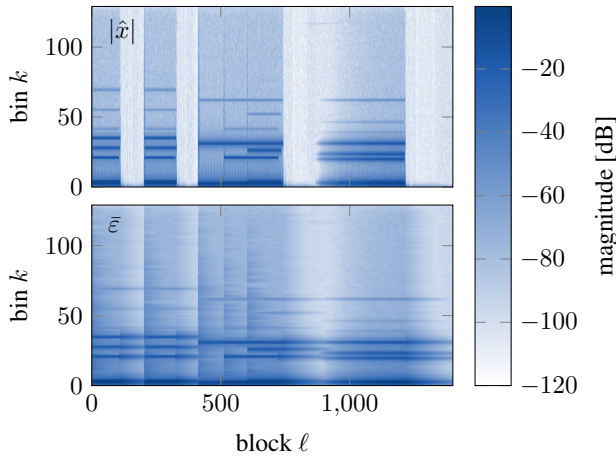


Figure 3: Time-frequency magnitudes before and after the envelope follower ($N = 256$, decay time 750 ms), for an organ-chords input. Top: input magnitude $|\hat{x}|$ (spectrogram); bottom: AR-averager signal $\bar{\varepsilon}$ (“envelope-gram”).

as for the modulator signal. For FDverb, since the DFT of a noise signal will be noise, we generate it directly in the spectral domain. For each block ℓ , we generate a noise spectrum $\hat{u}_k \in \mathbb{R}$, $k = 1, \dots, K$ as uniformly distributed samples $\hat{u}_k[\ell] \stackrel{\text{i.i.d.}}{\sim} \mathcal{U}_{[-1,1]}$.

Next, we use the extracted envelopes $\bar{\varepsilon}_k$ from (2) to weight the spectral noise bins. For each block ℓ , we compute the reverb-tail spectrum as $\hat{r}_k[\ell] = \hat{u}_k[\ell] \cdot \bar{\varepsilon}_k[\ell]$ for $k = 1, \dots, K$.

2.2.4. Inverse Short-Time Fourier Transform

With the weighted noise $\hat{r}$, we apply the inverse spectral transform $F^{-1}(\hat{r}[\ell]) = \mathbf{r}[\ell] \in \mathbb{R}^N$ from K bins to N samples. We apply a sine window again to reduce blocking artifacts and then perform overlap-adding of consecutive blocks with an overlap of $\frac{N}{2}$, to obtain the time-domain reverb-tail signal r .

2.3. The Impact of Block Size N

We discuss two important repercussions of the block size N : (i) the effect on the sound character of the reverb tail and (ii) the latency introduced in a real-time implementation.

Together with the sample rate f_s , N sets the frequency resolution f_s/N and the block duration N/f_s , two reciprocal quantities that trade off against each other. A large block size (i.e., large N/f_s) provides a fine frequency resolution and a tonal tail at the cost of temporal smearing, whereas a short block (small N/f_s) provides a coarse frequency resolution and a noise-like tail that follows the input more sharply in time.

In a real-time implementation, block-wise processing delays the tail signal relative to the input, since the input must be buffered and processed before the corresponding tail emerges. This delay has two contributions: the per-block processing time T_{proc} , which is small (a few milliseconds or less), and the algorithmic latency $T_{\text{algo}} = \frac{N}{2f_s}$. Their sum is the total latency, which imposes a lower bound on the delay between the dry signal and the reverb tail; we denote this bound by $T_{\text{tail, min}} = T_{\text{proc}} + T_{\text{algo}}$, the minimum possible tail-delay.⁶

⁶For example, $N = 8192$ at $f_s = 48$ kHz results in $T_{\text{algo}} \approx 85$ ms.

Without early reflections, this minimum tail-delay would also be the onset of the reverb itself and, hence, the perceived pre-delay; for large N , such a long pre-delay audibly detaches the reverb from the dry signal and sounds unnatural. In FDverb, we use early reflections to avoid this issue: implemented by time-domain convolution (see Section 2.1), their pre-delay has no such lower limit, so the reverb can begin with a short pre-delay even when the tail arrives late. To provide the user with independent control over the pre- and tail-delays, we delay the tail signal so that it aligns with T_{tail} , the position of the last early reflection. In the user interface, $T_{\text{tail, min}}$ then acts as the lower limit of the `tail-delay` parameter.

2.4. Adding Bells and Whistles

We now introduce several enhancements that extend the basic FDverb tail algorithm with additional creative controls.

2.4.1. Alternative Envelope Modes

The reverb tail of a physical room typically follows an exponential decay curve (which gave rise to (2), further referred to as *exponential envelope mode*). Beyond this physically motivated envelope mode, FDverb offers two additional envelope modes.

In the *arithmetic envelope mode*, we replace (2) by

$$\bar{\varepsilon}_k[\ell] = \begin{cases} \min\{\bar{\varepsilon}_k[\ell-1] + \alpha_k, \varepsilon_k[\ell]\} & \text{if } \varepsilon_k[\ell] \geq \bar{\varepsilon}_k[\ell-1] \\ \max\{0, \bar{\varepsilon}_k[\ell-1] - \beta'_k, \varepsilon_k[\ell]\} & \text{if } \varepsilon_k[\ell] < \bar{\varepsilon}_k[\ell-1] \end{cases} \quad (3)$$

where β'_k are the release-time coefficients for the arithmetic mode.⁷ The arithmetic envelope mode produces a reverb tail that sounds unnatural and resembles *gated reverb* (sometimes also referred to as *nonlinear decay reverb*).

In the *freeze mode*, we replace (2) by

$$\bar{\varepsilon}_k[\ell] = \max\{\varepsilon_k[\ell], \bar{\varepsilon}_k[\ell-1]\}, \quad (4)$$

which results in a reverb tail that does not decay, i.e., is sustained infinitely. This mode is similar to *time-freeze* described in [2].

In the FDverb user interface, the alternative envelope modes are controlled as follows: The mode selector chooses between exponential and arithmetic envelope modes, while the *freeze* button allows one to momentarily enable the freeze envelope mode. The freeze envelope mode also features an input gate, which is controlled with the *gate* toggle. With the gate open, the input spectrum is continuously accumulated into the reverb-tail envelopes; with the gate closed, the tail-reverberator input is muted and the reverb-tail envelopes are held fixed. The dry signal x is unaffected in either case.

2.4.2. Pitch Shift and Pitch Drift

Taking creative freedom further, we extend FDverb with two pitch-shifting features for the reverb tail.

With the *shift* parameter, we offset the bins of the analysis side and the synthesis side of the tail algorithm. In detail, we introduce an input bin offset $\sigma \in \mathbb{Z}$ and modify the envelope update equations (i.e., (2), (3), (4)) to use $\varepsilon_{k-\sigma}$ in place of ε_k , treating out-of-bounds elements as zero. The offset σ applies a static pitch shift to the reverb tail (upward or downward, depending on the sign

⁷We set $\beta'_k = 4 \frac{N/2}{f_s T_1}$, where T_1 is the time required for a decrease in amplitude of 1 and the factor 4 is an empirical choice.

of σ). However, since this shift is linear in frequency, harmonic intervals are not preserved, resulting in an inharmonic reverb tail.

With the `drift` parameter, we offset the state-holding elements of the AR-averager units in the algorithm. In detail, we introduce a state bin offset $\delta \in \mathbb{Z}$ and modify the envelope update equations (i.e., (2), (3), (4)) to use $\bar{\varepsilon}_{k-\delta}[\ell-1]$ in place of $\bar{\varepsilon}_k[\ell-1]$, treating out-of-bounds elements as zero. The offset δ applies a dynamic pitch shift to the reverb tail, i.e., the pitch of the reverb tail rises or falls over time, depending on the sign of δ . Similarly to the shift parameter, the linear-in-frequency property again yields an inharmonic reverb tail.

2.4.3. Brick-Wall EQ

We extend the reverb-tail algorithm with a brick-wall bandpass equalizer that can be used to tailor the frequency range of the reverb tail using the parameters `low-cut` and `high-cut`. The equalizer is implemented in the frequency domain where it sets the noise bins outside the passband to zero

$$\hat{u}_k = 0, \quad k \in \{1, \dots, K_{\text{HP}}, K_{\text{LP}} + 1, \dots, K\}. \quad (5)$$

2.4.4. Stereo Options

So far, we have only detailed mono audio processing. We extend FDverb to stereo signals by running two tail-reverberator instances in parallel, one per channel. The `width` parameter controls the correlation of the noise spectrum $\hat{u}$ between the two channels, ranging from fully correlated (mono, i.e., *mid* only), to fully uncorrelated (full stereo, i.e., *mid* and *side*), to fully anti-correlated (out-of-phase, i.e., *side* only). The `link` parameter controls whether the envelopes of the two channels are processed independently or averaged into a shared mono envelope.

Furthermore, the early-reflection taps are distributed across the stereo field using amplitude panning. The panning follows a fixed random pattern, with the `width` parameter controlling the spread.

3. DAW PLUG-IN IMPLEMENTATION

FDverb is implemented as a plug-in for digital audio workstations (DAWs) using the JUCE framework⁸ and is provided in the VST3 and Audio Unit (AU) formats. The ER unit is implemented using a delay line, and up to 32 echoes are supported with a maximum delay of 500 ms. The spectral transform from Section 2.2.1 is implemented using the real fast Fourier transform (RFFT) provided by JUCE and block sizes from $N = 2^5$ to $N = 2^{14}$ are supported. To support DAW buffer sizes larger and smaller than the block size N , the tail reverberator runs on a dedicated background thread, communicating with the real-time audio callback through lock-free queues. In the current version, we measured the average per-block reverb-tail processing time to range from 0.011 ms at $N = 2^5$ to 3.81 ms at $N = 2^{14}$. Across all supported block sizes N , the per-track CPU meter of REAPER reported less than 2.6 % for one instance of the FDverb plug-in.⁹

The user interface of the FDverb plug-in features all of the parameters mentioned in Section 2 and also adds a real-time “envelope-gram” of the envelopes $\bar{\varepsilon}_k$, a visualization similar to

⁸<https://github.com/juce-framework/JUCE>

⁹Measurements performed on a MacBook Pro (M2 Pro), FDverb VST3 in REAPER v7.76, 512 samples I/O buffer size, $f_s = 48$ kHz; the CPU meter reading is taken from REAPER’s FX window.



Figure 4: User interface of the FDverb plug-in with the real-time envelope visualization found at the top right.

a rolling spectrogram; see Figure 4 for a screenshot of the user interface. At the time of writing, the FDverb plug-in described here is version v1.0 of the Git repository.

4. VISITOR EXPERIENCE

Visitors are invited to explore FDverb hands-on via a laptop running the DAW plug-in, connected to an audio interface and headphones. We provide a set of parameter presets for FDverb as a starting point, and visitors are encouraged to tweak the parameters in real time. As an audio input source, visitors can choose from pre-recorded instrument samples and music, or connect their own device.

5. REFERENCES

- [1] J. A. Moorer, “About this reverberation business,” *Computer Music J.*, vol. 3, no. 2, pp. 13–18, Jun. 1979.
- [2] E. Vickers, J.-L. L. Wu, P. G. Krishnan, and R. N. K. Sadanandam, “Frequency domain artificial reverberation using spectral magnitude decay,” in *Proc. 121st Audio Eng. Soc. Conv.* Audio Eng. Soc., Oct. 5–8, 2006.
- [3] J. Vilkamo, B. Neugebauer, and J. Plogsties, “Sparse frequency-domain reverberator,” in *Proc. 40th Intl. Audio Eng. Soc. Conf. on Spatial Audio*. Audio Eng. Soc., Oct. 8–10, 2010.
- [4] W. A. Production. Venom. Accessed: Apr. 10, 2026. [Online]. Available: <https://www.waproduct.com/plugins/view/venom>.
- [5] J. Roth, S. Krebs, and C. Studer, “graetli: A microcontroller-based DSP platform for real-time audio signal processing,” in *Intl. Conf. Digital Audio Effects (DAFx25)*, Ancona, Italy, Sept. 2–5, 2025, (demo paper).
- [6] U. Zölzer, Ed., *DAFx: Digital Audio Effects*, 2nd ed. Chichester, UK: John Wiley & Sons, 2011.
- [7] G. W. McNally, “Dynamic range control of digital audio signals,” *J. Audio Eng. Soc.*, vol. 32, no. 5, pp. 316–327, 1984.