# State Space Filters for Artificial Reverberation Effects

Warren L. G. Koontz
*Rochester Institute of Technology*
Rochester, NY USA
wlkmet@rit.edu

***Abstract*—This paper shows the relation between state space filters and both feedback delay networks and more complex structures that can add a reverb effect to an audio signal. The paper goes on to show how one of these structures, known as Multiverb, can be implemented in a straightforward manner as both a MATLAB streaming audio script and a VST audio plug-in.**

***Index Terms*—audio signal processing**

## I. Introduction

In a live musical performance, or any other situation involving live sound production, sound travels from the source to the listener over multiple paths, creating a sometimes pleasing and sometimes confusing effect known as *reverberation*. Efforts to produce a pleasing artificial reverb effect began in 1947 with the recording of ”Peg o’ My Heart” by The Harmonicats. The ”reverb chamber” used to produce this recording was the studio bathroom. Since that time, many approaches to artificial reverberation have be developed, including mechanical, electronic, and, most recently, digital signal processing (DSP) techniques.

In a previous paper [1], my colleagues and I presented a DSP algorithm for creating an artificial reverberation (AR) effect. The algorithm is an extension of the feedback delay network (FDN) approach described in [2], [3]. This paper shows how both our algorithm, which we call “multiverb”, and the FDN algorithm can can be viewed as extensions of a standard state space filter. In addition, this paper provides information about implementing multiverb in MATLAB and as a VST audio plug-in.

## II. DSP Approach to Artificial Reverberation

Figure 1 shows an example of the impulse response (IR) of a reverberant space. The initial part of the IR is referred to as *early reflections* and the latter part is referred to as *late reverberation*. The IR decays more or less exponentially and the echo density increases with time. Given the IR of a reverberant space, we can add an AR effect to an audio signal by convolving the signal with the IR. Fast convolution algorithms, described in [4] and elsewhere, make this approach quite practical. Moreover, we can measure the IR of a given space by producing and recording a sound in that space. The swept sine method described in [5] provides an accurate measure of the IR.

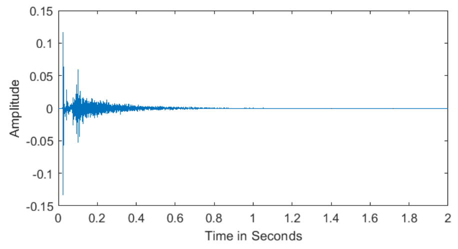


Fig. 1. Impulse Response of Reverberant Space.

Another approach is to design a digital filter with a reverberant IR and pass the audio signal through this filter. In the 1960s, Schroeder and Logan [6] used combinations of comb filters, all-pass filters, and mixing matrices to add AR to a digital audio signal. They combined comb filters in parallel and all-pass filters in series and sometimes used the mixing matrix to extract one or two output channels from individual filter outputs. A descendant of their work, known as “freeverb,” is still widely used. Freeverb was developed by someone known as “Jezar at Dreampoint,” and source code for freeverb is currently available at http://blog.bjornroche.com/2012/06/freeverb-original-public-domain-code-by.html and perhaps elsewhere. Feedback delay networks as described in [3] were introduced more recently.

In the next section, we will present the FDN as a logical extension of a standard state space filter. We will then show how the FDN can be extended to a two-channel structure that is multivefrb.

## III. State Space Filters for Artificial Reverberation

All of the filters that we will consider are discrete-time, linear, time-invariant systems. The basic filter is described by the following pair of equations:

$$\begin{aligned}\mathbf{x}(n+1) &= \mathbf{A}\mathbf{x}(n) + \mathbf{B}\mathbf{u}(n) \\ \mathbf{y}(n) &= \mathbf{C}\mathbf{x}(n) + \mathbf{D}\mathbf{u}(n)\end{aligned} \quad (1)$$

where $\mathbf{x}$ is the $N \times 1$ state vector, $\mathbf{y}$ is the $M \times 1$ output vector, $\mathbf{u}$ is the $L \times 1$ input vector, $\mathbf{A}$ is the $N \times N$ state transition

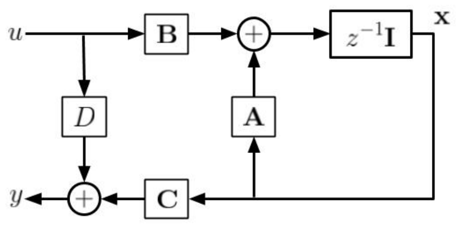


Fig. 2. SISO State Space Filter.

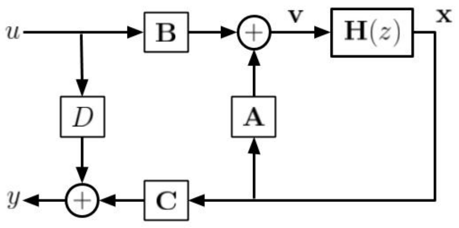


Fig. 3. Feedback Delay Network.

matrix, $\mathbf{B}$ is an $N \times L$ matrix, $\mathbf{C}$ is an $M \times N$ matrix, and $\mathbf{D}$ is an $M \times L$ matrix. In the $z$ domain, (1) becomes

$$\begin{aligned} z\mathbf{x}(z) &= \mathbf{A}\mathbf{x}(z) + \mathbf{B}\mathbf{u}(z) \\ \mathbf{y}(z) &= \mathbf{C}\mathbf{x}(z) + \mathbf{D}\mathbf{u}(z) \end{aligned} \tag{2}$$

Figure 2 displays (2) for the single input, single output (SISO) case as a signal flow diagram. Figure 3 introduces an extension of the state space filter where the single sample delay $z^{-1}\mathbf{I}$ is replaced with a more general $N \times N$ matrix $\mathbf{H}(z)$. This extended filter is described by the following set of equations:

$$\begin{aligned} \mathbf{v}(z) &= \mathbf{A}\mathbf{x}(z) + \mathbf{B}u(z) \\ y(z) &= \mathbf{C}\mathbf{x}(z) + Du(z) \\ \mathbf{x}(z) &= \mathbf{H}(z)\mathbf{v}(z) \end{aligned} \tag{3}$$

A more direct relation between the input and the output is given by

$$y(z) = \{\mathbf{C}[\mathbf{I} - \mathbf{H}(z)\mathbf{A}]^{-1}\mathbf{H}(z)\mathbf{B} + D\}u(z) \tag{4}$$

To construct a basic FDN, we start by defining $\mathbf{H}(z)$ as a set of delay lines with varying delays:

$$\mathbf{H}(z) = \begin{bmatrix} z^{-M_1} & 0 & \cdots & 0 \\ 0 & z^{-M_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & z^{-M_N} \end{bmatrix} \tag{5}$$

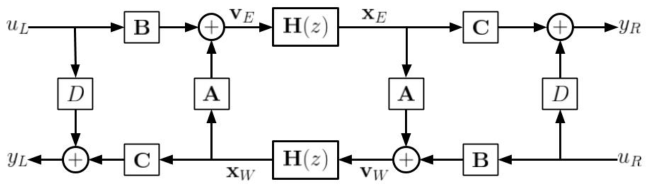


Fig. 4. Multiverb Filter.

We complete the basic FDN by defining the remaining parameters:

$$\begin{aligned} \mathbf{A} &= \frac{1}{N}\mathbf{O} - \mathbf{I} \\ \mathbf{B} &= \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \\ \mathbf{C} &= \frac{r}{N}\begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} \\ D &= 1 - r \end{aligned} \tag{6}$$

where $\mathbf{O}$ is an $N \times N$ matrix of all ones. Defined this way, the FDN is a stable filter and the parameter $r$ can be used to vary the output from fully filtered ($r = 1$) to unfiltered ($r = 0$).

We can take the FDN a step further by defining $\mathbf{H}(z)$ as

$$\mathbf{H}(z) = \begin{bmatrix} z^{-M_1}A_1(z) & 0 & \cdots & 0 \\ 0 & z^{-M_2}A_2(z) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & z^{-M_N}A_N(z) \end{bmatrix} \tag{7}$$

where $A_k(z)$ is an attenuation function given by

$$A_k(z) = \frac{g_k}{1 - d_k z^{-1}} \tag{8}$$

Now $\mathbf{H}(z)$ models a set of delay lines with frequency-dependent losses. The attenuation varies between $g_k/(1 - d_k)$ at 0 Hz and $g_k/(1 + d_k)$ at the Nyquist frequency. This leads to a relatively straightforward method to compute $g_k$ and $d_k$ as described in [4]. The $g$ and $d$ parameters will be called *gain* and *damping*, respectively.

In the time domain, this FDN with lossy delay lines is expressed as

$$\begin{aligned} \mathbf{v}(n) &= \mathbf{A}\mathbf{x}(n) + \mathbf{B}u(n) \\ y(n) &= \mathbf{C}\mathbf{x}(n) + Du(n) \\ x_k(n+1) &= d_k x_k(n) + g_k v_k(n - M_k) \quad k = 1, 2, \ldots, N \end{aligned} \tag{9}$$

As discussed in Section V, the last line of (9) can be implemented efficiently using buffers.

## IV. The Multiverb Filter

The multiverb filter is shown in Figure 4. It is a combination of two FDN filters and provides two inputs and two outputs labeled left and right. The internal signals $\mathbf{v}$ and $\mathbf{x}$ are labeled

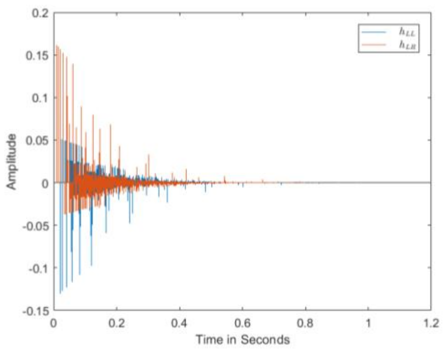
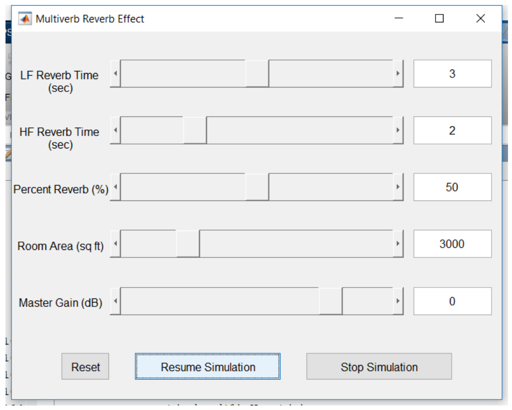


Fig. 5. Multiverb Filter Impulse Response.



LF Reverb Time (sec) 3
HF Reverb Time (sec) 2
Percent Reverb (%) 50
Room Area (sq ft) 3000
Master Gain (dB) 0

Fig. 6. MATLAB Prototype GUI.

east and west and there are corresponding eastbound and westbound delay lines. The filter is governed by the following set of equations:

$$\begin{aligned}\mathbf{v}_E(z) &= \mathbf{A}\mathbf{x}_W(z) + \mathbf{B}u_L(z)\\ y_L(z) &= \mathbf{C}\mathbf{x}_W(z) + Du_L(z)\\ \mathbf{v}_W(z) &= \mathbf{A}\mathbf{x}_E(z) + \mathbf{B}u_R(z)\\ y_R(z) &= \mathbf{C}\mathbf{x}_E(z) + Du_R(z)\\ \mathbf{x}_E(z) &= \mathbf{H}(z)\mathbf{v}_E(z)\\ \mathbf{x}_W(z) &= \mathbf{H}(z)\mathbf{v}_W(z) \qquad (10)\end{aligned}$$

The inputs and outputs are related by

$$\begin{aligned}y_L(z) &= G_{LL}(z)u_L(z) + G_{LR}(z)u_R(z)\\ y_R(z) &= G_{RL}(z)u_L(z) + G_{RR}(z)u_R(z) \qquad (11)\end{aligned}$$

where (dropping the references to $z$)

$$\begin{aligned}G_{LL} = G_{RR} &= \mathbf{C}[\mathbf{I} - (\mathbf{HA})^2]^{-1}\mathbf{HAHB} + D\\ G_{LR} = G_{RL} &= \mathbf{C}[\mathbf{I} - (\mathbf{HA})^2]^{-1}\mathbf{HB} \qquad (12)\end{aligned}$$

Although (12) is not a practical means of implementing the filter, it provides a nice view of the filter as a two-port device. The time-domain version of (10) is similar to (9), however, and also leads to an efficient implementation using buffers. The time domain (difference) equations can be used to determine the impulse responses corresponding to $G_{LL/RR}$ and $G_{LR/RL}$. Examples of these are shown in Figure 5.

## V. Implementation

The multiverb filter has been implemented as both a MATLAB prototype and a VST audio plugin. Both implementations include a GUI with the following control sliders:

- Low and high frequency reverb time seconds.
- Room area in square feet.
- Percent reverb.
- Gain in decibels.

The reverb time and room area control values are used to determine the delay line lengths ($M_k$) and the gain and damping coefficients. The details of these calculations will not be presented here. The percent reverb is converted to the parameter $r$ in (6) and the gain is converted from decibels and applied to the filter output.

### A. MATLAB Prototype

The MATLAB prototype takes advantage of MATLAB's streaming audio capabilities. Using MATLAB's audio file reader and audio player objects, one can easily create a script to read, process, and play audio from many standard audio file types. Each iteration of the script's main loop does the following:

- Check for and handle updates from the GUI.
- Get next block of audio.
- Process block.
- Set block to player.

Except for updating the filter and processing the block, much of the script is copied from MATLAB examples. The GUI is shown in Figure 6.

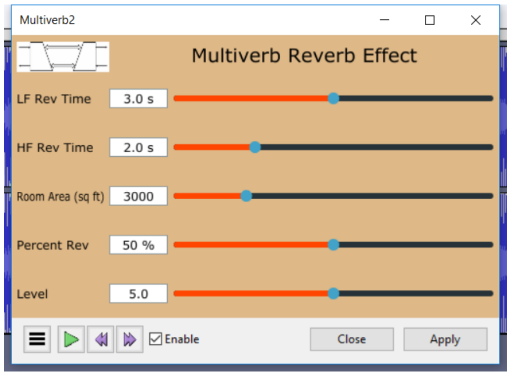
The block processing code is listed here:

```
for k=1:NS
    Veast=A*Xwest+B*uL(k);
    Vwest=A*Xeast+B*uR(k);
    y(k,1)=ampl*(rmix*C*Xwest+(1-rmix)*uL(k));
    y(k,2)=ampl*(rmix*C*Xeast+(1-rmix)*uR(k));
    for n=1:N
        Xeast(n)=damp(n)*Xeast(n)+...
            gain(n)*east(n,tail(n)+1);
        Xwest(n)=damp(n)*Xwest(n)+...
            gain(n)*west(n,tail(n)+1);
        east(n,head+1)=Veast(n);
        west(n,head+1)=Vwest(n);
        tail(n)=mod(tail(n)+1,dmax);
    end
    head=mod(head+1,dmax);
end
```

The first part of the loop matches nicely with (10). The nested loop handles the delay lines using the buffers `east` and `west`.

### B. VST Audio Plug-in

The VST audio plug-in implementation of the multiverb filter was developed using the JUCE framework [7]. JUCE



LF Rev Time 3.0 s
HF Rev Time 2.0 s
Room Area (sq ft) 3000
Percent Rev 50 %
Level 5.0

Fig. 7. VST Plug-in GUI.

provides a straightforward procedure to develop audio plug-ins, including VST plug-ins, in C++ and creates much of the necessary code automatically. As a developer, your main concern is to layout the GUI and code your parameter update and processing algorithms.

JUCE is available as a free download (there are also "pro" versions) and works with the free version of Visual Studio (I used VS 2017).

When you start JUCE and select the audio plug-in option, JUCE creates the frameworks for two classes: `PluginEditor` and `PluginProcesssor`. These two frameworks provide a starting point for creating the GUI and processing the audio data. The multiverb GUI is shown in Figure 7. I will focus on the processing part.

The multiverb algorithm is coded in a C++ class called `Mverb`. The class includes an `update` method and a `process` method. The `update` method computes the delay line lengths and the gain and damping coefficients based on the current values of the reverb times and room size and the `process` method applies the filter to a block of two-channel audio samples. The JUCE `PluginProcessor` class includes numerous methods, many of which you can safely ignore. Two key methods are `prepareToPlay` and `processBlock`. You add code to `prepareToPlay` to get the parameter values from the GUI and to call the `Mverb` `update` method. You add similar code to `processBlock` to get the latest parameter values and call `update` if either the reverb times or the room area have changed. Finally, you add a call to the `Mverb process` to `processBlock`.

Of course there are other programming details to attend to, but these are covered adequately by online JUCE tutorials. You do the actual coding in the IDE, which in my case is VS 2017. The end result is a `.dll` file, which you copy to the appropriate plug-in folder. I have tested multiverb extensively with Audacity[1].

## VI. Conclusion

This paper shows how a classic state space filter can be extended to create a feedback delay network and a more complex structure that we call multiverb. Both the FDN and multiverb can be designed to produce a reverb effect on audio signals. The paper also outlines how to implement multiverb both as a MATLAB prototype and as a VST plug-in. The VST plug-in implementation is especially useful, since VST plug-ins are accepted by a variety of hosts[2].

## References

[1] W. L. G. Koontz, S.-Y. Kim, and M. J. Indelicato, "A digital reverberation simulator based on multi-port acoustic elements," *JMEST*, vol. 2, no. 1, pp. 185–191, January 2015.

[2] J.-M. Jot and A. Chaigne, "Digital delay networks for designing artificial reverberators," in *Audio Engineering Society Convention 90*. Audio Engineering Society, 1991.

[3] J. O. Smith, *Physical Audio Signal Processing: For Virtual Musical Instruments and Digital Audio Effects*. Julius Smith, 2006.

[4] W. Koontz, *Introduction to Audio Signal Processing*. RIT Press, 2016. [Online]. Available: https://books.google.com/books?id=xOAYMQAACAAJ

[5] A. Farina, "Simultaneous measurement of impulse response and distortion with a swept-sine technique," in *Audio Engineering Society Convention 108*. Audio Engineering Society, 2000, pp. 18–22.

[6] M. R. Schroeder and B. F. Logan, "Colorless artificial reverberation," *IRE Transactions on Audio*, no. 6, pp. 209–214, 1961.

[7] "Juce framework for audio applications." [Online]. Available: https://juce.com/

[1]Audacity® software is copyright ©1999-2016 Audacity Team. The name Audacity® is a registered trademark of Dominic Mazzoni.

[2]Pro Tools is a notable exception