

Composing The System “Filter” And The System “Reverberator”

Arun Chandra

Computer Music Project, U of I

INTRODUCTION

As a composer, I investigate and compose the dialectical relationship between components and the systems they inhabit: the system determines the significance of its components, and the components determine the behavior of their system.

Filtering and reverberation are two closely linked techniques of digital signal processing. The mathematics of DSP are daunting, and will not be discussed in this brief overview. Rather, I would like to present to other composers the structural relations that define the system “filter” and the system “reverberator”, and some potentials for creative work that remain to be tried out. Locally, Michael Hamman, Adam Cain, and Camille Goudeseune have explored some possibilities of networks of filters and signal processors. I hope this overview will encourage others to make experiments in this area.

In the following discussion, these abbreviations hold:

n	the current sample number
y	the output (the filtered sound)
x	the input (the source sound)
$y(n)$	the n -th output sample
$x(n)$	the n -th input sample
sr	the sampling rate

Other abbreviations are defined in their contexts.

FILTERING

The simplest filter can be built by taking the average of every two samples:

$$y(n) = 0.5 * x(n) + 0.5 * x(n - 1)$$

This is a low-pass filter with one “tap”. The current input sample $x(n)$ and the previous input sample $x(n-1)$ are both multiplied by the coefficient 0.5, and added together.

To make the above into a high-pass filter, take the difference between adjacent samples:

$$y(n) = 0.5 * x(n) - 0.5 * x(n - 1)$$

To use either of the above algorithms in a program, one would use a syntax similar to:

```
y[0] = x[0] ;                               /* set the initial value */
for ( n = 1 ; n < numSamples ; n++ )       /* process rest of samples */
    y[n] = 0.5 * x[n] - 0.5 * x[n-1] ;
```

(The above should be optimized for speed in any final program.)

Filters are divided up into two general types: recursive and non-recursive. Non-recursive filters (also called “Finite Impulse Response” filters (FIR)) modify only the input samples. Recursive filters (also called “Infinite Impulse Response” filters (IIR)) modify both the input samples and the past output samples.

An example of a recursive filter is:

$$y(n) = a * x(n) - b * y(n - 1)$$

Here, the previous output sample $y(n-1)$ is subtracted from the current input sample $x(n)$ to create the new output sample $y(n)$. The choice of coefficients (a and b) is important, and should almost always be less than 1. If the coefficients are not chosen carefully, it is easy to make a recursive filter “blow-up”, i.e., the output values get larger and larger until they exceed the limits of the output type, usually 16-bits.

The composer Kirk Corey (a former student from here, currently teaching at the University of Iowa) wrote some programs running on a PC, in which his compositional investigation was to explore the sounds created before recursive filters blow-up, given a variety of initial coefficients.

An intriguing aspect of Butterworth filters is that merely by changing the coefficients, you can change the behavior of the filter from low-pass to high-pass to band-pass to band-reject. All Butterworth filters use the following algorithm:

$$y(n) = a1 * x(n) + a2 * x(n - 1) + a3 * x(n - 2) - b1 * y(n - 1) - b2 * y(n - 2)$$

As you can see, this filter has two input taps $x(n-1)$ and $x(n-2)$, and two output taps $y(n-1)$ and $y(n-2)$. To get a low-pass filter, the coefficients a1, a2, a3, b1, b2 are calculated thusly:

```
fc = cutoff frequency
```

```
C = tan(pi*fc/sr)
```

```
a1 = pow((1 + sqrt(2)*C + C*C), -1)
```

```
a2 = 2*a1
```

```
a3 = a1
```

```
b1 = 2 * (1 - C*C) * a1
```

```
b2 = (1 - sqrt(2)*C + C*C) * a1
```

But for a band-pass filter, the coefficients are:

```
fc = center frequency
```

```
bw = band width
```

```
C = pow( tan(pi*bw/sr), -1 )
```

```
D = 2 * cos(2*pi*fc/sr)
```

```
a1 = pow((1+C), -1);
```

```
a2 = 0
```

```
a3 = -a1
```

```
b1 = -C * D * a1
```

```
b2 = (C - 1) * a1
```

The coefficients for a high-pass or a band-reject filter are calculated in a similar, mind-numbing manner.

A compositional possibility offered by the above is an investigation into what might happen if the coefficients were to be linearly varied between one set into another over the duration of the sound. One would then have a filter that would change its properties from low- to band-pass, or high-pass to band-reject, etc. One way to do it would be this:

```
/* first set the magnitude of change */
```

```
alchange = (alfinal - a1) / (numSamples - 1) ;
```

```
a2change = (a2final - a2) / (numSamples - 1) ;
```

```
a3change = (a3final - a3) / (numSamples - 1) ;
```

```
b1change = (b1final - b1) / (numSamples - 1) ;
```

```
b2change = (b2final - b2) / (numSamples - 1) ;
```

```
y[0] = x[0] ;          /* initialize the samples */
```

```
y[1] = x[1] ;
```

```
for ( n = 2 ; n < numSamples ; n++ ) {      /* process the sound */
```

```
    y[n] = a1*x[n] + a2*x[n-1] + a3*x[n-2] - b1*y[n-1] - b2*y[n-2] ;
```

```
    a1 = a1 + alchange ;          /* increment the coefficients */
```

```

    a2 = a2 + a2change ;
    a3 = a3 + a3change ;
    b1 = b1 + b1change ;
    b2 = b2 + b2change ;
}

```

Now, if a_1 , a_2 , a_3 , b_1 and b_2 represented the coefficients for a low-pass filter, and a_{1final} , a_{2final} , a_{3final} , b_{1final} and b_{2final} represented the coefficients for a band-pass filter, this filter would change its characteristics from being low-pass to band-pass over the duration of the input sound $x[n]$ (assuming that `numSamples` is the duration in samples of $x[n]$). This system of change can be raised to higher levels of complexity. It might be amusing to find out what this would sound like using the input sound of a spoken text.

REVERBERATION

The simplest reverberator is the comb filter, which is a variant of the recursive filter shown above:

$$y(n) = x(n - D) + g * y(n - D)$$

“D” represents the number of samples of the “delay”. So, one can see that the current output $y(n)$ is equal to a delayed input $x(n-D)$ plus a previous output $y(n-D)$ multiplied by a gain factor “g”. If the value for D is high (say a tenth of a second), the comb filter will act as a kind of “echo”.

The gain “g” (usually between 0 and 1) determines the “reverberation time” of the filter. Charles Dodge has suggested that the relationship between g and the “reverberation time” (as composers might prefer to think of it) is:

D	delay in seconds
rvt	reverberation time in seconds
g	$\text{pow}(0.001, (D/sr/rvt))$

So, a composer can specify the reverberation time he or she would like the comb filter to exhibit, and thusly calculate the value of “g”. (This is why in ReFill, if you change the gain factor for a comb or allpass filter, the reverberation time also changes, and vice-versa.)

An interesting property of this filter is that a previous output $y(n-D)$ is scaled by g, and becomes part of the current output $y(n)$. This means that if $g = 0.9$, the first time an output sample will be multiplied by 0.9, the second time by $0.9 * 0.9$ (0.81), the third time by $0.9 * 0.9 * 0.9$ (0.729), etc., meaning that the previous output will be steadily decreasing in amplitude, i.e., the echoes will be “dying away”. If the initial value of g is smaller, the sound will die away faster:

$g = 0.5$
 $0.5 * 0.5 = 0.25$
 $0.5 * 0.5 * 0.5 = 0.125$
 $0.5 * 0.5 * 0.5 * 0.5 = 0.0625$
 (etc.)

Metaphorically speaking, the magnitude of D determines “the size of the room” one is emulating, and the magnitude of g determines “the type of material the walls are made of”, i.e., whether they highly reverberant walls (creating a long reverberation time), or whether they are muffling walls (creating a short reverberation time).

For composers, what could be amusing is to continually change the setting for g , meaning that one is changing the “walls of the room” in which the sound is imagined to occur, or change the setting for D , meaning that one is changing the “size of the room”, i.e., a speaker is shoved from the Krannert Great Hall into a garbage can. Or vice-versa. That is usually called “getting a job.” (Or is it the other way 'round?)

M.A. Schroeder, in the 1960s and 1970s, had suggested two models for “realistic reverberation”. The first model was to use five allpass filters cascaded together, i.e., the output of the first was the input to the second, the output of the second was the input to the third, etc. (This filter is called Schroeder1 in ReFill.) The algorithm for an allpass filter is:

$$y(n) = -g * x(n) + (1 - g * g) * (x(n - D) + g * y(n - D))$$

As you can see, the latter part of an allpass filter is a comb filter.

The second model he suggested was to use four comb filters in parallel, sum their outputs, and then pass their outputs through two allpass filters in sequence. (This filter is called Schroeder2 in ReFill). These two filters designed by Schroeder are the most commonly used ones in digital synthesis.

James A. Moorer built on the work of Schroeder. He noticed that Schroeder’s filters tended to keep the high partials of a sound ringing, and so suggested that the comb filter be used with a built-in low-pass filter:

$$y(n) = x(n - D) + g2 * (y(n - D) + (g1 * y(n - (D + 1))))$$

He then suggested a model for reverberation that involved six combs with built-in low-pass filters run in parallel, their outputs are summed, and then sent to an allpass filter. (This filter is called “Moorer Reverb” in ReFill.)

A problem manifest by all three of these reverberation units (first noticed by Schroeder) is their “lack of early echoes”. Remember that the samples are delayed by the magnitude of D . This means samples that are less than D are ignored by the reverberators. Schroeder suggested a solution, which was to add to the source sound some initial delays in the form of an FIR filter:

$$y(n) = a1 * x(n) + a2 * x(n - D1) + a3 * x(n - D2) + \dots aN * x(n - N)$$

These delays would range from 0 to 80 milliseconds. Moorer found some appropriate coefficients for either 7 or 19 “early echoes”. This has been implemented in ReFill under the option “Add Early Echoes”.

REFILL

The program ReFill is a NeXTStep application, currently residing in `arunc/Apps/ReFill`. Please feel free to copy it and use it! The source code is not part of it, but if anyone is interested in looking at the source code and playing with it, drop me a line and I'll send it to you. And please let me know if it crashes on you, or gives other problems. I've tested it with three minute, stereo sounds at $sr = 44100$, and it has behaved properly, so it should work with anything you can throw at it.

SOURCES

Charles Dodge and Thomas Jerse, *Computer Music*

James A. Moorer, *About this reverberation business*

M.A. Schroeder, *Digital simulation of sound transmission in reverberant spaces*

M.A. Schroeder, *Natural sounding artificial reverberation*

F. Richard Moore, *Elements of Computer Music*

Charles P. Bernardin, *C/Math Algorithms for Engineering and Scientific Applications*

Paul Embree and Bruce Kimble, *Algorithms for Digital Signal Processing*