Effectsecution:

a journey into DSP sound manipulation

Physics 498pom Final Paper

Farsheed Hamidi-Toosi
Jason Noah Laska

Introduction

For our final project, we researched and implemented DSP effects algorithms in C. Initially, we anticipated on implementing these algorithms on a real-time Texas Instrument C6000 series DSP processor, but due to time constraints and equipment problems, abandoned the idea. Instead, we coded these algorithms in C and used a free, open source DSP audio API called PortAudio. The advantage to using PortAudio was that it provided real-time audio input/output routines for a number of popular platforms, allowing the code we wrote to be very portable. PortAudio will work on Mac, Windows and Unix machines. And since the DSP coding is in C, it would be easy to port the code to the Texas Instruments DSP, which also compiles C. Once we had decided on using PortAudio, it was time to decide what audio algorithms to implement.

We wanted to learn about commercial audio algorithms that are in use today, as well as experiment with our own audio DSP algorithms. Since many audio effects are based on variations of a few simple concepts, we decided to try and implement the basic concepts and then use those as building blocks for more complex effects. The basic effects can be categorized under time, frequency, or amplitude effects.

Delay

The simplest audio effect algorithm is delay. This effect consists of adding a delayed version of the input mixed with the current input itself. The block diagram for a simple delay is shown below.

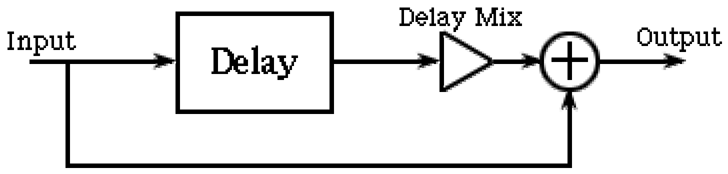

fig. 1. simple delay

The block called "delay" is often denoted as z^{-D}, where D is the length of the delay in samples, because dsp transfer functions often are shown using z transforms, which is simply short hand notation for $z^{-d} = e^{jdwt}$, In either case, this delay is usually implemented using a circular memory buffer of length D. The delay mix multiplier scales the delay output to provide control over the volume of the delayed input. A variation on the simple delay is the feedback delay, which additionally takes the output from the delay and adds it back into the input of the delay.

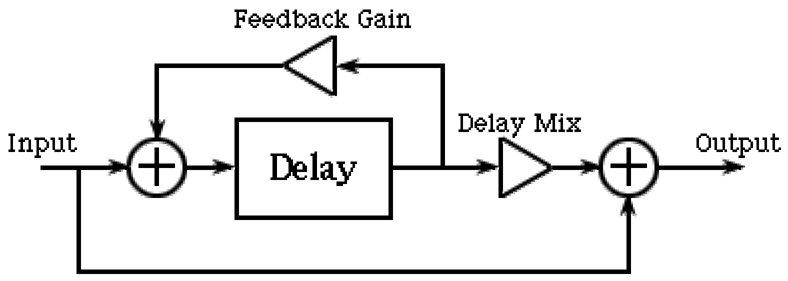

fig. 2. delay with feedback [2]

For delay without feedback, the real world analogy would be standing rather far away from a wall and then yelling. The sound wave would take a certain amount of time to travel and come back, which can be calculated from distance = rate x time, where rate is the speed of sound in air (~343 m/s at NTP). The addition of this feedback component is a more accurate model if the echoes are able to bounce back and forth between two walls. If we think about yelling into a large canyon, when we yell we hear a delayed version of our initial yell, but then we hear the secondary reflections as the sound waves bounce back and forth between the walls of the cave until they die out. In real life, the feedback gain is always less than one because part of the energy is absorbed when it bounces off a wall. However, on the computer we can set this gain to unity (infinite echoes that never die out) or greater than unity which will cause instability and will result in the gain of the overall output to go to infinity very quickly, which will cause distortion and/or clipping.

Yet another variation on delay is the stereo ping-pong delay which has the following block diagram:

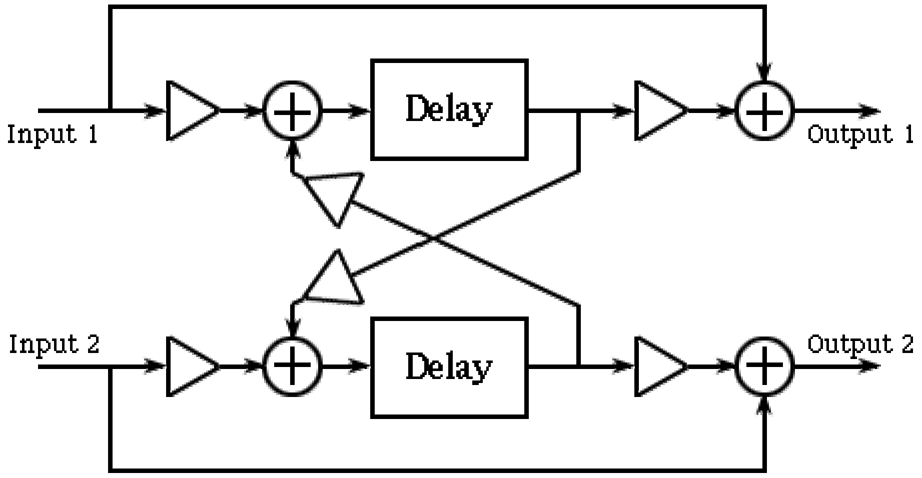

fig. 3. Ping-Pong delay[2]

This variation essentially takes the feedback component and feeds it into the input of the delay unit of the opposite channel. The result is that each iteration of an echo flips from the right channel to the left channel.

Implementation of the Delay Block

The delay block is implemented by saving samples into a circular buffer. The circular buffer is a data structure similar to a fixed length queue. Items are placed into the top of the buffer and read out through the bottom. The first items to go into the buffer are the first items that come out, and we read and write at a one to one rate (ie: for everyone sample read from the buffer, we write a new sample to the buffer). This can be seen intuitively seen in the example case of delay. If we initialize a length D circular buffer by filling it with zeros, and then enqueue samples onto the buffer, D samples will be read followed by the delayed data. The circular buffer concept is fundamental to implementation of many of these effects as most of them require some form of delay block.

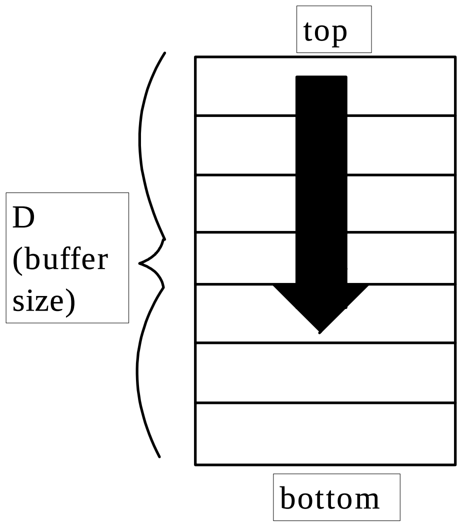

fig. 4. Circular Buffer

Amplitude Modulation (or Ring Modulation)

This effect modifies the frequency content of the input. All it does is simply multiply the input by a cosine at a particular frequency in the time domain. The result of such an operation modulates the input frequency, which creates sum (w1+w2) and difference (|w1-w2|) frequencies. These sum and difference tones may not be harmonic, and thus the result of this effect can be very dissonant. If we consider the input to be x(t)=cos(w1) and the modulating frequency to be c(t)=cos(w2) then our output would be y(t)=x(t) * c(t).

Mathematically this looks like:

$$\begin{aligned} y(t) &= x(t) * c(t) \\ &= \cos(w1)*\cos(w2) \\ &= \tfrac{1}{2} [\cos(w1-w2) + \cos(w1+w2)] \end{aligned}$$

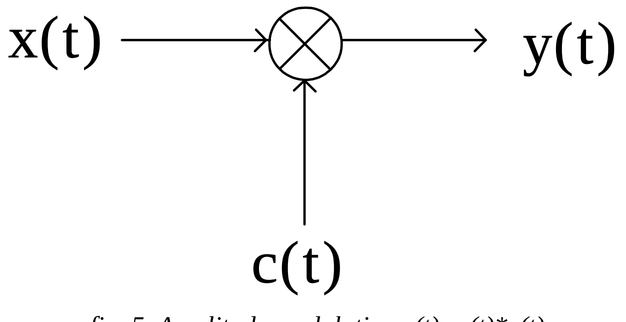

*fig. 5. Amplitude modulation y(t)=c(t)*x(t)*

This effect is also called ring modulation, most likely because sounds that ring (i.e. bells, cymbals) have dissonant, chaotic, or non-harmonic spectrums.

Tremolo

An interesting thing happens though, when the amplitude modulation frequency of c(t) drops below ~10 Hz. The sensation of hearing sum and difference tones magically disappears and instead we hear what is often referred to as tremolo. This sounds like the original input sound pulsating in amplitude at the modulating frequency, but no extra tones are present. The reason for this has to do with the auditory physiology of the human ear. If the modulation is able to create sum and difference tones that fall outside of the critical bandwidth at that frequency, then a second tone appears. However, if the modulation frequencies fall within one critical bandwidth, only the original tone is perceived. Another way to think about it is that since the lowest frequency the human ear can detect is around 20 Hz, any amplitude variations less than 20 Hz will sound like amplitude fluctuations instead of another frequency. Human hearing is most definitely non-linear.

All-Pass Filter

One of the most useful building blocks in digital audio effect design is the all-pass filter. This filter has the characteristic of having a gain equal to unity but causes a phase shift at a particular frequency. The reason for this can be seen by simply examining the transfer function for a first order all-pass filter:

$$\mathrm{AP} = \frac{g_i + z^{-1}}{1 + g_i z^{-1}}.$$

The unique property of the all-pass filter comes from the fact that all the poles cancel the zeros, exactly. However, this filter modifies the phase of the input at particular frequencies. By adding the input with the all-passed version of the input, those frequencies that are 180^o out of phase will cancel and there will be a notch or frequency attenuation. This can be utilized to make an equalizing algorithm or phasor effect. The block diagram of an all-pass filter looks very similar to delay with feedback:

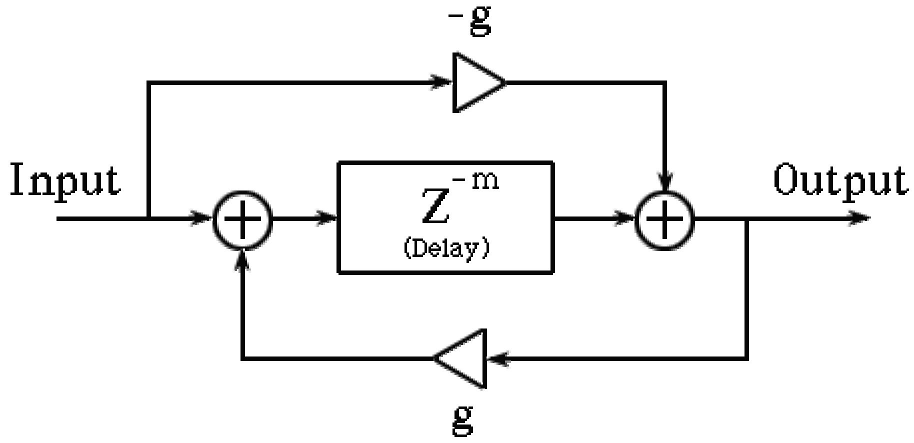

fig. 6. 1st order All-Pass Filter [2]

The only difference between this diagram and the delay with feedback diagram are the g and –g multiplier terms. The condition that the feedforward multiplier is the opposite of the feedback multiplier must be true in order for the resonant frequencies to cancel the zero frequencies. You can imagine if you took away the feedback portion, the resulting filter (comb filter) would create zeros or notches in the spectrum at frequencies equal to 1/delay [Hz]. If you took away the feedforward portion, you would have resonances equal to 1/delay [Hz].

Distortion

A very popular effect that is often desired is that of distortion. Distortion is an effect that applies quadratic, cubic, or other higher order non-linearities to a signal, which results in an increased number of harmonics-per-input frequency. Since these harmonics are integer multiples of the input frequency, the perceived output is fuzzy and "warm". Distortion is often implemented in the digital world via a simple output/input amplitude transfer function. With fancier digital distortion, you can actually draw the transfer function with lines and achieve some pretty bizarre (and usually bad/gritty sounding) distortion. For our distortion, we implemented a simple cubic distortion routine. A cubic distortion brings extra harmonics (notably the third harmonic) in addition to the original input frequency. In general, digital distortion sounds much harsher and colder than analog distortion, most likely because the amplitude for a typical audio wave file has a bit depth of 16 bits, which limits one to 2^{16}=32768 different amplitude values. Thus mapping a cubic distortion will involve some quantization error when compared to an analog (and therefore continuous) distortion equivalent. When mapping original samples to a cubic function, the magnitude of the audio can easily grow to large scales causing clipping. To reduce this clipping effect, the magnitudes are first scaled, then mapped with a cubic function, and then rescaled to that the sound follows a non-linear pattern.

Harmonic Exciter/Resonator

Many of today's audio DSP algorithms were developed in the 70's and the 80's and still remain in use today. One such algorithm is the Karplus-Strong physical model of a string. This model basically consists of a delay with feedback, scaled and attenuated by a low-pass filter.

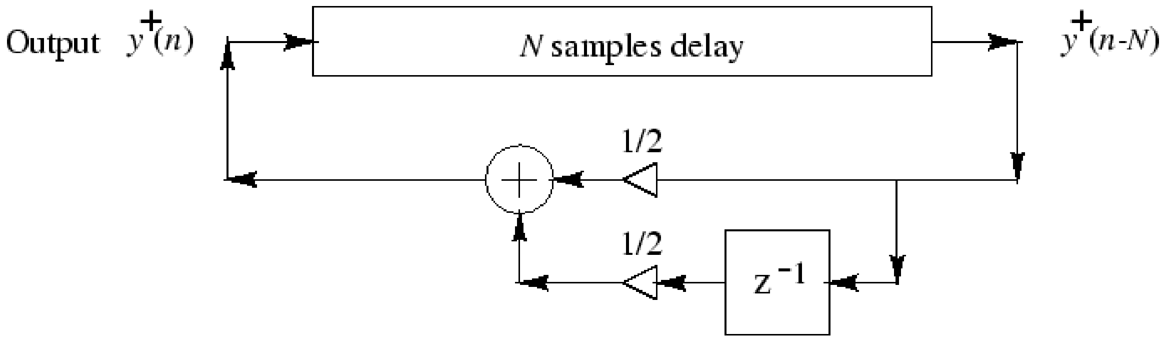

fig. 7. Karplus Strong Harmonic Resonator with averaging filter applied to feedback.[7]

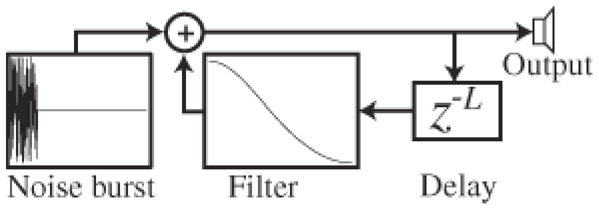

fig. 8. A more intuitive diagram of the Karplus- Strong model.

Each component of this algorithm represents a physical aspect of wave propagation. The delay buffer represents the time it takes for the wave to propagate down the length of the string, and the feedback represents the wave bouncing back up the string, while the two ½ multiplier terms and z^{-1} implement an averaging filter which represents the body of the instrument. By loading the N samples delay buffer with harmonically rich content, such as white noise, a sharp resonance will occur centered upon the frequency determined by the length of the delay. This frequency can be determined as f = 1/delay and can be thought of as a pole resonance at this frequency. One thing that we found out and had fun with was that by loading the delay buffer with different shapes, we could achieve various initial conditions similar to plucking a guitar. That is, if we loaded a symmetric triangle function into the buffer, it would be analogous to plucking a guitar string from the middle. This resulted in a mellow-like resonance frequency tone. If the triangle is made more asymmetric, by shifting the peak to the left side, we essentially created a significantly sharper slope on one side, which introduced higher harmonics. This is similar to plucking a guitar string near the bridge, resulting in a brighter tone signal. By accident, we also loaded the buffer with a sinc^2 waveform, which actually made the waveform sound like an old 80's synthesizer with a weird phasor/flanger effect. This simple algorithm is still one of the best sounding string synthesis algorithms available, in our opinion.

Digital Waveguide Models

An expansion upon Karplus Strong's model is to simply create a more exact replica of waves traveling on a string. By creating two delay buffers, one for the P^+ traveling wave and one for the P^- traveling wave, creating reflection coefficients for the boundary conditions, and using more complex filters to represent the body of the instrument, a more "exact" physical model can be created. What about string thickness, material-type, attenuation variables, environmental conditions and nonlinearities? These physical models can become extremely complicated very quickly, but the tradeoff is computational cost. In trying to model a spring reverb unit, we tried to use a digital waveguide but found it did not sound too similar to the real thing. However, during our demonstration Professor Errede informed us that a real spring reverb unit has three springs, not just one. Since we only modeled one spring, we had greatly simplified our model and thus were missing some important synthesis parameters.

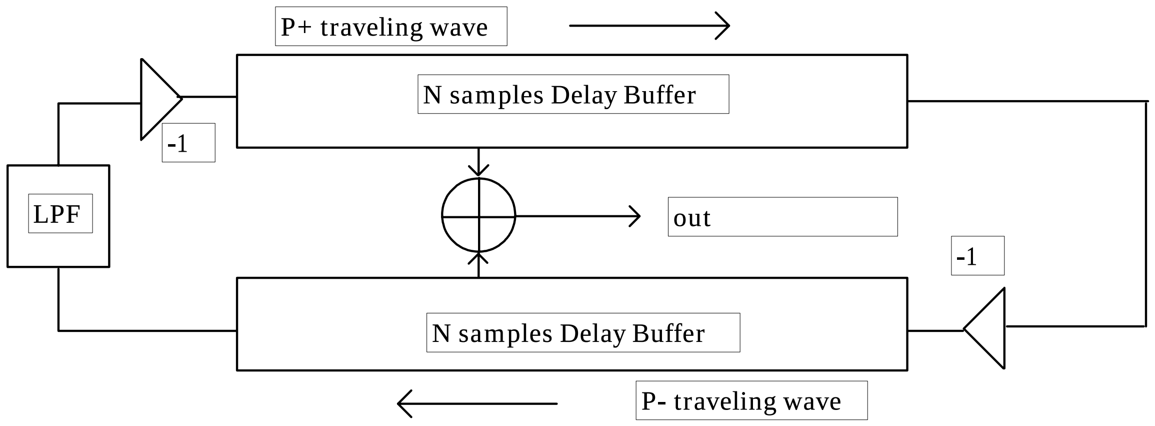

fig. 9. Spring Reverb Digital Waveguide Model

Reverb

Reverb is the result of acoustic echoes bouncing and dissipating in a room. Inspired by our lackluster success of our spring reverb digital waveguide model, we felt inclined to implement a simple reverb filter model based on comb filters and all-pass filters. A comb filter is simply a feedforward delay, which creates notches in the spectrum. The way this reverb filter algorithm works is that it models the early reflections as delayed versions of the input, much like our simple delay algorithm, and then it models the dissipation of echoes using all-pass filters. This transition from early reflections to white noise like dissipation "air" occurs after a period of time referred to as Schroeder's time. This is when the sound waves bouncing off the walls are occurring so frequently and there are so many reflections that it does not sound like echoing but rather like decaying band-pass filtered white noise. The impulse response of a particular room might look like the following picture.

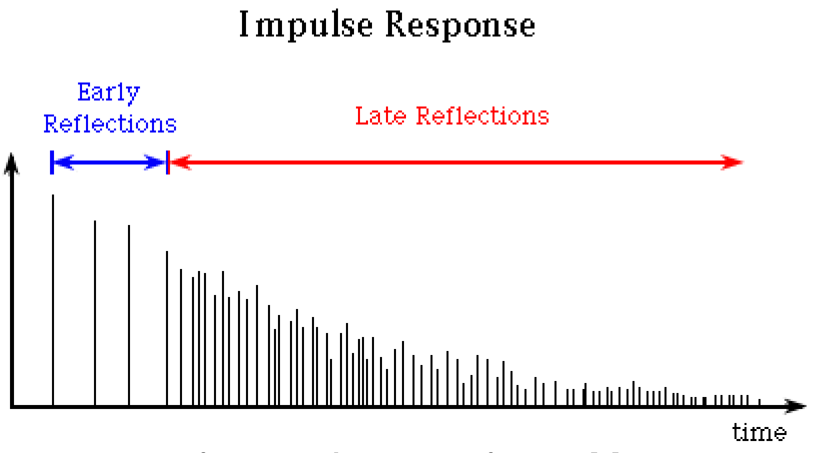

fig. 10. Impulse response of a room. [2]

You can see the early reflections are much more widely spaced out and discernable than the late reflections. The algorithm we implemented was Schroeder's reverberator algorithm.

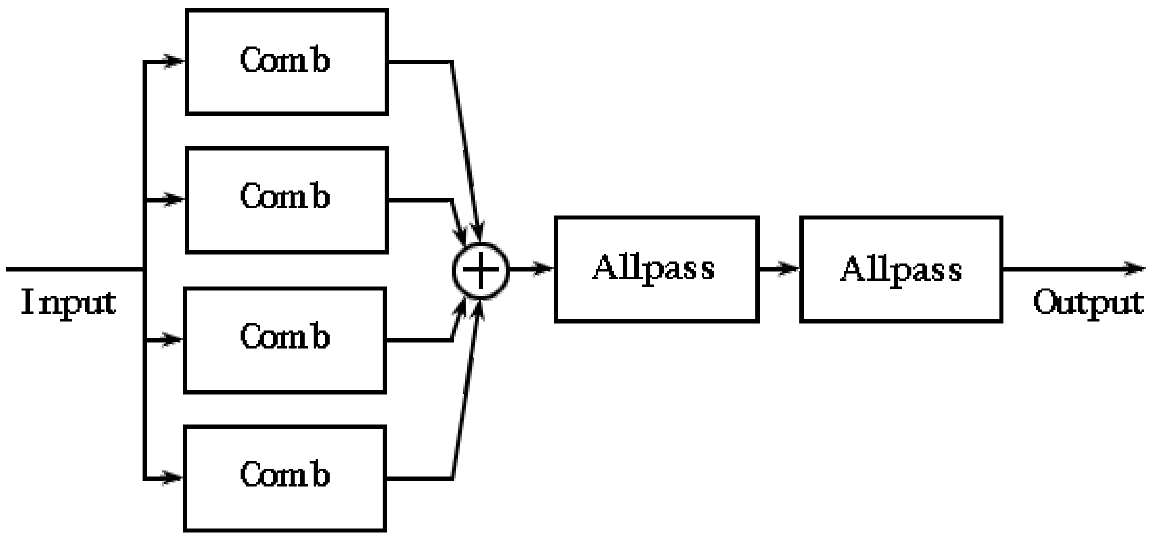

fig. 11. Schroeder's comb filter reverb.[2]

The comb filters represent the input bouncing off the walls to form the early reflections, and uses the all-pass filters to model the late reflections. The only major problem with this algorithm is that transient signals sound very unnatural, which is most likely due to the settling time of the IIR all-pass filters.

Reverse

Reverse simply takes an audio signal, stores it in memory, and reads it out in the opposite order. This means that if you said the word "hello", it would end up sounding like "olleh" after a specified delay amount. The longer you set the delay time, the longer the reverse effect will occur.

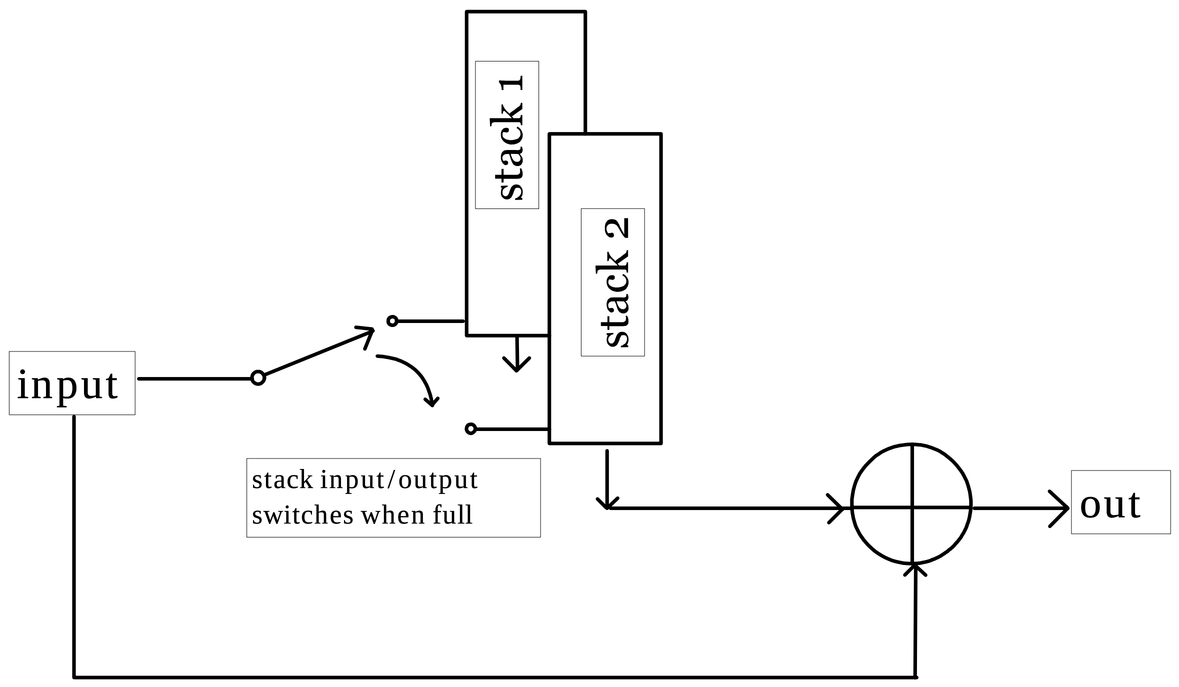

fig. 12. Reverse Algorithm

This effect uses two stacks in memory, and is very simple to implement. You could probably do this with analog circuitry but it would not be easy. The idea is that while the input fills one stack, the contents of the other stack is emptied. Once the stack is full, the input starts filling the other stack, and the recently filled stack empties.

Phasor

With a simple variation of the delay effect, we can achieve a phasor effect. A simple phasor consists of making the delay time of a simple delay unit to be less than 50 ms and varying this delay length either linearly or sinusoidally. Since the lowest frequency that is audible to the human ear is 20 Hz, then any delay that is less than that period (~50 ms) will not be heard as an echo, but rather a phase shift. The output then is the phase-shifted version of the input is added to the original input. If the phase shift happens to be 180^o of the input, the output will go to zero. However if the phase shift happens to be 360^o then the output for that frequency will be amplified by two. The result is that certain frequencies are boosted and attenuated for various phase shifts. If we vary this phase shift over time, it is equivalent to "sweeping" over all frequencies. Another way to look at a phasor is that it is simply creating a notch filter at a particular frequency, and then sweeping the filter over a specified range. A simple delay unit with no feedback is equivalent to a digital filter with a zero occurring at a frequency of 1/delay length. Varying this delay length varies the location of this zero point. We also experimented with trying to sweep a 2nd order IIR bandpass filter to see if it sounded like phasor, and it sounded pretty close, but the simple 1st order allpass configuration sounded best.

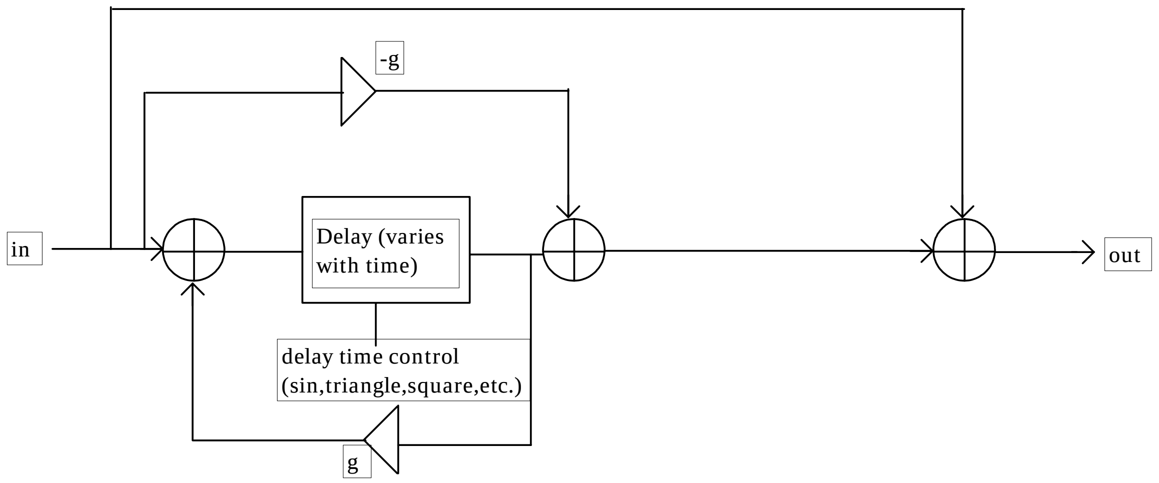

fig. 13. Phasor using 1st order All-Pass filter.

Parametric EQ

This algorithm is based on a paper by Mitra-Regalia. It implements a very nice bandpass/bandstop filter with independent control over the bandwidth, center frequency, and boost/attenuation. It uses a second-order all-pass filter to create a 180^o phase shift at a given frequency, and when this is mixed with the input, it causes an attenuation at that frequency. In addition, this filter uses a lattice implementation of the all-pass filter instead of Direct form II because using Direct Form II creates instability due to the feedback factor. Essentially a lattice structure is a single order all-pass filter nested inside of another single order all-pass filter. Lower order digital IIR filters are in general more stable than higher order digital filters, and thus it is advantageous to cascade lower order filters whenever possible.

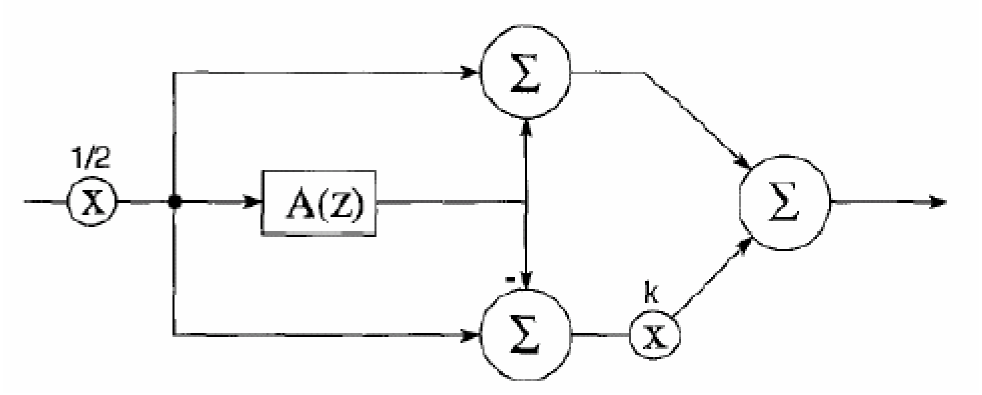

fig. 14. Mitra Regalia Topology.

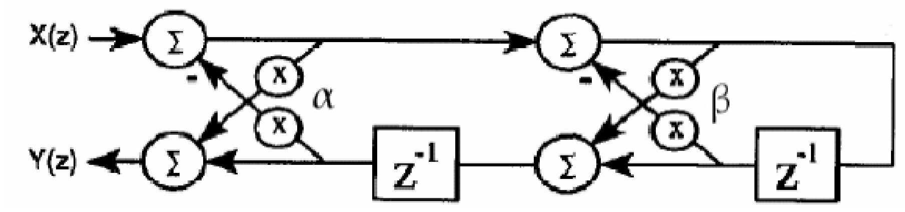

fig. 15. A(z) = 2nd order all-pass lattice.

Design Equations for this filter:

$\beta = -\cos(\omega_c)$ *this sets the center frequency of the filter in radians*

$k = 10^{(\text{GAIN}/20\text{ dB})}$ *this determines the GAIN of the filter (can boost or cut)*

$\alpha = \dfrac{(1 - \tan(\text{BW[rad]}/2)}{(1 + \tan(\text{BW[rad]}/2)}$ *this coefficient sets the 3dB bandwidth of the filter*

Some example plots detailing how the filter changes when the parameters change:

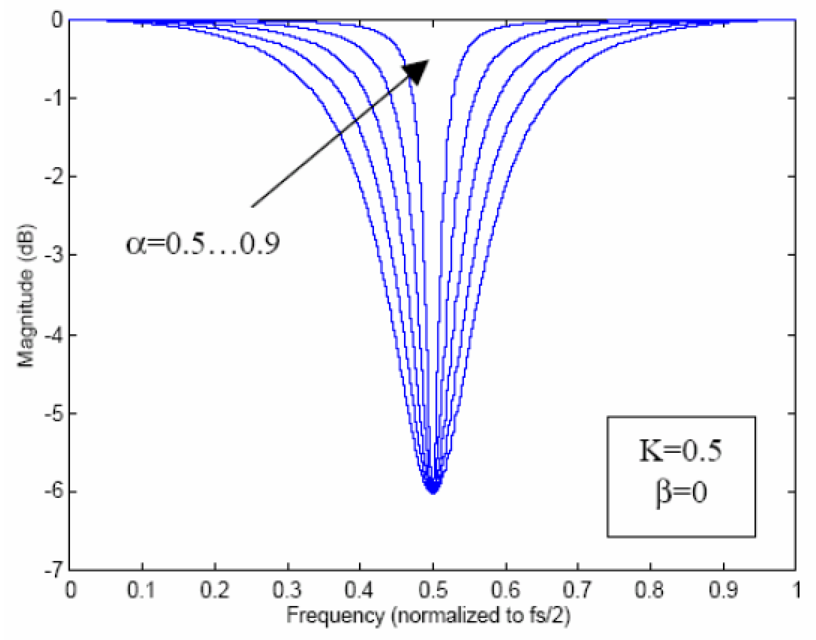

fig. 16. Changing alpha changes the 3dB bandwidth of the filter.[8]

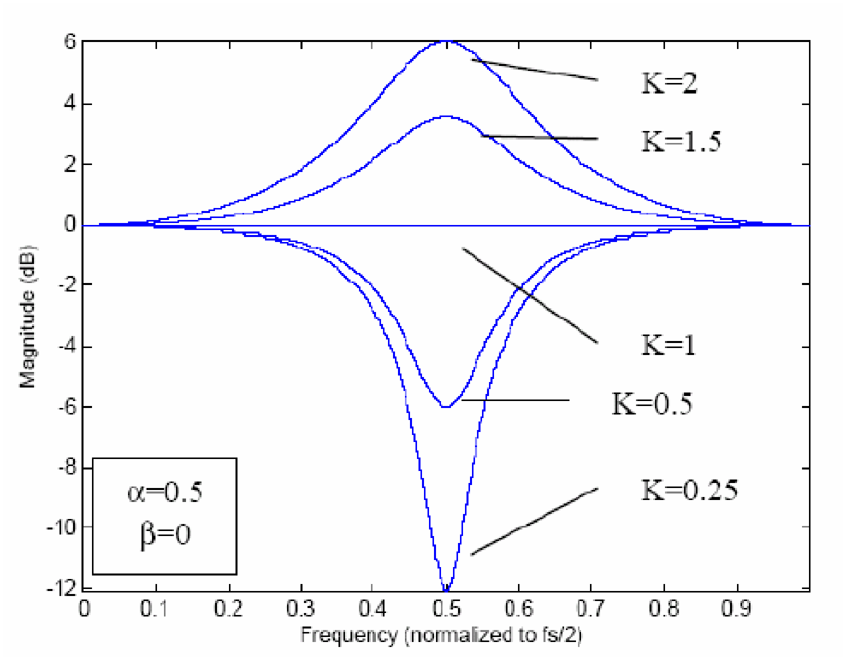

fig. 17. Changing K boosts or cuts by a certain amount.[8]

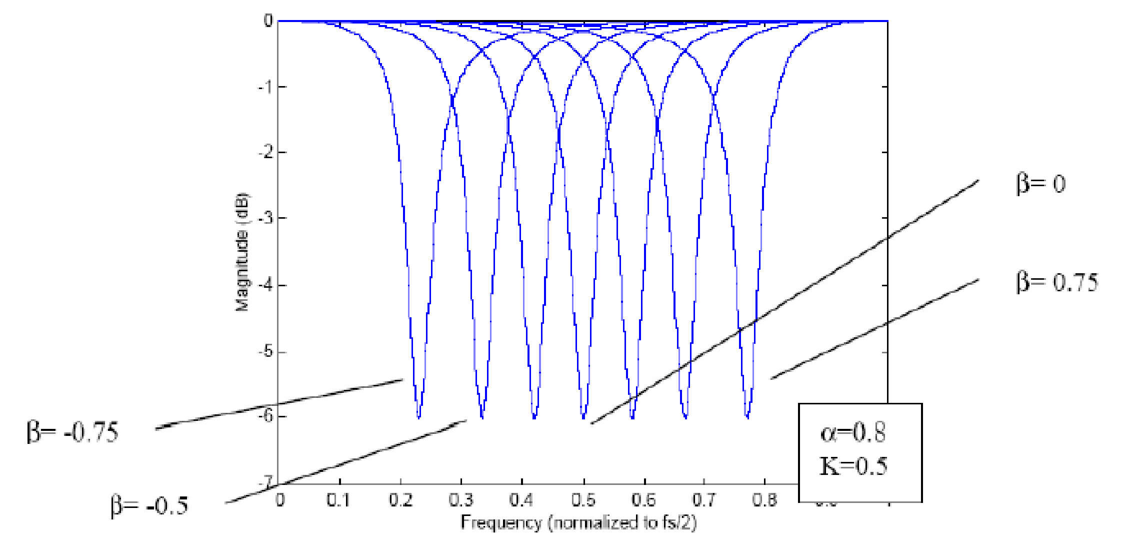

fig. 18. Changing beta changes the center frequency of the filter.[8]

Conclusion

We had a lot of fun researching and implementing these algorithms, especially since we grew up playing with programs on our computer that implemented these algorithms. Because we researched and experimented with these algorithms ourselves our understanding of audio DSP is much stronger than before.

Thanks

Thanks to Professor Errede for the overwhelming amount of knowledge he shared with us all throughout the semester, and the MusicDsp.com list, whose members created the outstanding PortAudio package.

Resources:

[1] Montana University Web Site.
http://www.coe.montana.edu/ee/rmaher/ECEN4002/lab4_020226.pdf

[2] Harmony Central Web Site.
http://www.HarmonyCentral.com/Effects/

[3] Errede, Steven. *Phys 498pom Notes, Spring 2005*.

[4] Haken,Lippold. *ECE 402 Notes, Fall 2004.*

[5] Hasegawa-Johnson, Mark. *ECE 403 Notes, Spring 2005.*

[6] PortAudio, Cross-Platform Audio API.
http://www.portaudio.com

[7] CCRMA, Stanford University.
http://ccrma.stanford.edu/~jos/SimpleStrings/Karplus_Strong_Algorithm.html

[8] Colorado State University.
http://ece-www.colorado.edu/~ecen4002/lab4_2004.pdf