DIGITAL EFFECTS PROCESSING

Final Report

Submitted to

The Faculty of Operation Catapult LXVIII

Rose-Hulman Institute of Technology

Terre Haute, Indiana

by

Group 31

Cheryl Fang	Monta Vista High School Cupertino, California
Jason Kaeding	Center Grove High School Greenwood, Indiana
Evan Ramey	Franklin Community High School Franklin, Indiana
Mark Slee	Conant High School Hoffman Estates, Illinois

July 28, 2000

Introduction

Digital signal processing has a wide variety of applications. Although its more common uses reside in cell phones, modems, voice recognition, and other telecommunications systems, digital signal processors (DSPs) are also widely used in the music industry. Almost all forms of music heard on the radio have been altered using digital signal processors. These alterations include noise reduction and amplitude scaling as well as many other effects. Some effects make music sound more natural, while others make music sound as if it has been turned upside-down. When used simultaneously, effects create even more unique sounds. In the world of musical effects, the cliché applies: "the possibilities are endless."

Effect Design

Before instructing a DSP to process audio data, it is necessary to understand exactly what must be done to the sound. For the most common effects, such as an echo, the process is fairly straightforward. However, more advanced effects like flanging, require research into what makes the effect possible.

The first effect we designed was a tremolo. A tremolo effect causes sinusoidal fluctuations in the amplitude of the signal, as if someone is turning the volume knob repeatedly back and forth. A tremolo effect is relatively easy to create because its main component is gain. A gain function multiplies the input signal by a constant, thereby changing the amplitude. Different DSPs implement gain in different ways, but the concept is the same. In order to create a tremolo, an oscillator must be used in conjunction with the gain function. The oscillator modifies the gain setting, and manipulating oscillator parameters such as rate (frequency) and depth (amplitude) changes the effect. An oscillator is typically implemented in the form of a sine wave generator, or through some form of trigonometric coding.

After learning the fundamentals of gain and sine generation, we began work on the delay function. Digitally delaying the output of a signal can be accomplished through the use of a circular buffer. When analog audio is converted to a digital signal, it is divided into samples. Typical digital audio has anywhere from 8,000 to 44,100 samples per second. A circular buffer works by storing these samples and later releasing them. When a sample enters the buffer, it becomes the first sample in the buffer. When the next sample enters, the original becomes the second sample. As expected, the sample moves up in order as each new sample is received. The settings of the buffer determine how many samples will be stored until the oldest sample is sent out. In order to create a 10 millisecond (ms) delay with a sample rate of 8000 hertz (Hz), it is necessary to have a buffer size of 80 samples. Therefore, a 10 ms delay circular buffer will store 80 samples, and then will send out the oldest stored sample each time a new sample is received.

A typical application of the delay function is an echo. In order to create an echo, the original signal must be combined with a delayed version of itself. Typically, this delayed signal has amplitude somewhere in the range of one half to three quarters times the original. A simple echo algorithm would resemble Diagram 1 (see next page). However, there is a problem with this algorithm. As anyone who has experienced a real echo knows, the echo is heard many times, not just once. The first algorithm only allows for the delayed signal to be heard once, and

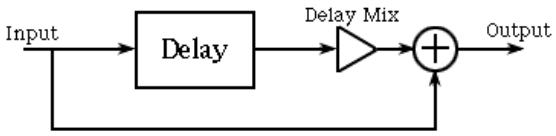


Diagram 1

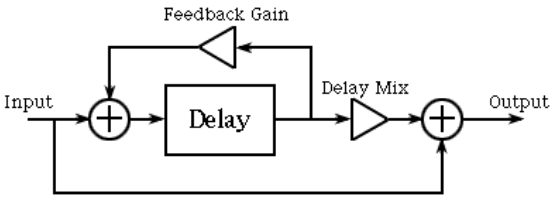


Diagram 2

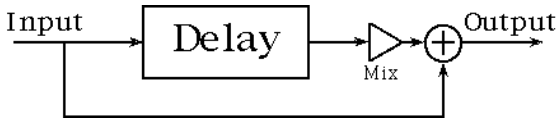


Diagram 3

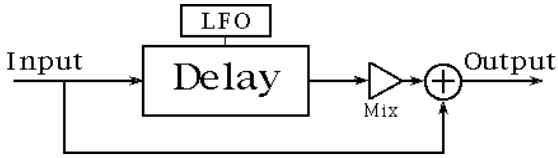
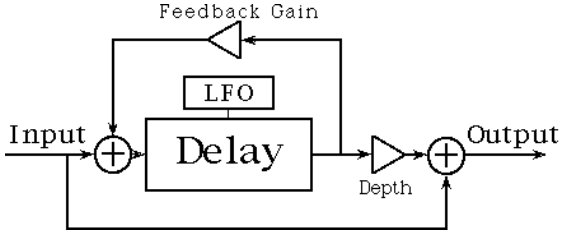


Diagram 4

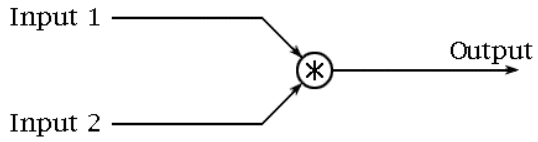


Flanger

therefore is referred to as a “slapback” echo. In order for multiple delays to be heard, a feedback loop must be incorporated. A feedback loop sends the output signal from the delay back through itself; so that delayed signal is repeatedly delayed until it dies out. Because the output is being fed into the input, it is absolutely necessary to insert a gain with a value less than one. Otherwise, the amplitude will build quickly and continuously, which may damage equipment. Diagram 2 correctly incorporates a feedback loop, and this delay will repeat until the sound dies out, just like a real echo.

Delay is not limited to echo effects. Instead, very short delays can be used to create very interesting effects. In these effects, it is often impossible for the listener to notice the sound being played back a second or third time. One commonly used delay effect is chorus, which is used in order to make one musician sound like an ensemble. The delays in a chorus are typically only about 30 ms, making them hardly noticeable. A chorus combines the input signal with multiple delayed versions of itself. Diagram 3 shows one branch of a chorus. This algorithm, however, would not sound very realistic, as a real musician’s timing is never perfect. For this reason, an oscillator is usually implemented in order to make the delay time vary slightly. Diagram 4 shows a basic chorus algorithm with an oscillator implemented, and it simulates multiple musicians much more accurately and effectively.

Another interesting delay-based effect is called flanging. In a flanger, delay times typically vary between 1 ms and 10 ms. These delays are so short that they create an effect entirely different from a typical echo. A flanger works due to interference between the original signal and the slightly delayed signal. The delay time oscillates sinusoidally, creating periodic changes in the interference pattern. The result of these oscillations is a sweeping effect, somewhat reminiscent of an airplane. In order to create a deeper effect, a feedback loop is incorporated just like in the echo algorithm. Flangers can create a variety of effects



Ring Modulator

based on the properties of the oscillator. A typical algorithm will provide its user with control of the frequency and amplitude of the oscillator, and the strength of the gain in the feedback loop.

An entirely different effect that works due to the combination of signals is called ring modulation. A ring modulator uses a process called heterodyning, or the multiplication of audio signals. When two signals are added, the original qualities of each are preserved, resulting in a very clean combination. Multiplication, however, alters the properties of the original signals, often creating entirely new frequencies and sounds. Because they create discordant frequencies, modulators are rarely used in popular music. The result of ring modulation is a somewhat mechanical or robotic sound.

Implementation

Once the fundamental design of an effect is known, the effect must somehow be created. The effect must undergo meticulous testing, because changing just one value can often alter the sound remarkably. Often, the same value must be changed numerous times until the desired sound is produced. We used two different pieces of software to create and test our effects.

Visual Application Builder (VAB) is a software application that uses an external DSP board to create effects. Specifically, we used two Texas Instruments boards, the C31 and the 621DSK. Algorithms are created in VAB by stringing blocks together on a worksheet. The finished product resembles a flow chart. VAB is easy to use because it enables the user to visually follow the path of the signal. The majority of the blocks used in VAB perform single mathematical operations. Examples of these are addition, subtraction, multiplication, division, and gain blocks. These simple single-operation blocks, however, are not sufficient for all effects. Many effects require oscillators, and VAB includes sine and cosine generators specifically for this purpose. These can be used to change parameters, as in a flanger, or to generate frequencies, as in a ring modulator. For more complex operations, such as a circular buffer, VAB uses hierarchies. Hierarchies are complex systems of single-operation functions that are used to perform a common operation. Because these complex operations are used frequently, it is much more convenient to create a single block that does not clutter the diagram. VAB allows you to create your own hierarchies that can be used in any new effect algorithm. This aspect of VAB helps make the creation process easy and natural.

Because VAB uses an external board that is dedicated entirely to DSP, it can process audio signals and output them immediately, in real-time. Real-time is advantageous because the user can hear the processed signal as soon as it is played. Real-time effects are ideal for live applications, where it is impractical to wait for a signal to be processed. Real-time, however,

also has many limitations. Any effect using a delay must store multiple samples of data, often as high as 1000 samples. Because real-time processing requires immediate output, the DSP board must have a large memory to operate successfully even with a medium-length delay. In these situations, programs that do not operate in real-time are the optimal choices because they have no time constraints to process data.

An excellent application for non-real-time processing is MATLAB. MATLAB is a software application that contains its own programming language. Although MATLAB was not designed primarily for audio, it is ideal for many effect applications. MATLAB is also incredibly powerful because there are no limits on the amount of time it can take to process a signal. MATLAB works mainly through manipulation of matrices. Audio passages in the form of .wav files can be stored into MATLAB as a matrix variable. Once in MATLAB, other variables can be used to create delays, flangers, and other effects. The following code is an echo algorithm.

```
infile = input('Input filename: ','s');              (1)
outfile = input('Output filename: ','s');
[x,fs]=wavread(infile);
xlen=length(x);
y = zeros(size(x));
d = round((fs/100)*2.5);                             (6)
a = .7;
for i=1:1:d
y(i)=x(i);
end
for i=d+1:1:xlen                                     (11)
y(i)=x(i)+a*y(i-d);
end
wavwrite(y,fs,outfile);
```

Lines one and two prompt the user for the input and output file names. Line three reads the input file and stores the sample frequency, and line four stores the length of the file. Line five creates a new matrix, y, the same size as x, which will be the output file. Line six stores the delay time, as a function of the sample frequency, to a variable, and line seven stores the value of the delay gain. The following sequence, lines eight to ten, instructs MATLAB to store the original signal to the output signal for the time before the delay begins reading samples. Next, lines eleven to thirteen tells MATLAB to add the original signal to the signal from the current time minus the delay time, thus creating the delay effect. Finally, MATLAB writes a wave file from the new matrix y.

The following is an example of a flanger, which works in the same way as the echo. However, the flanger incorporates a trigonometric function, which modifies the delay time.

```
infile = input('Input filename: ','s');              (1)
outfile = input('Output filename: ','s');
[x,fs]=wavread(infile);
xlen=length(x);
y = zeros(size(x));
delay = zeros(size(x));                              (6)
d = round(fs/100);
a = .6;
```

```
for i=1:1:d
   y(i) = x(i);
end                                                    (11)
for i=d+1:1:xlen
   delay(i) = round(d*(.5*cos(1*pi*i/fs)+.5));
   y(i)=x(i)+a*y(i-delay(i));
end
wavwrite(y,fs,outfile);                                (16)
```

Sound Synthesis

Neither MATLAB nor VAB is limited to processing external audio signals, yet both include powerful utilities for creating sounds. The VAB software includes a variety of sound generators. The most basic generators are sine and cosine generators. These generators have parameter settings for amplitude, frequency, sample rate, DC offset, and phase offset. Amplitude controls volume; frequency controls pitch; sample rate determines the samples per second; DC offset adds a constant to the sine wave, causing it to lie above or below the axis; and phase offset shifts the sine wave horizontally, ideal for creating interference with other sine waves. By manipulating these settings and combining multiple generators, a relatively wide range of sounds can be made. Ultimately, any waveform can be made through precise combination of sine waves. VAB also includes a sweep generator, which is a sine wave with frequency modulation. The frequency sweeps from a minimum frequency to a maximum frequency and then back down. The sweep generator has settings for these two frequencies, as well as for amplitude, sample rate, DC offset, and phase offset.

Moreover, MATLAB is capable of producing audible waveforms. This can be accomplished by programming in a trigonometric function, as shown by the following code:

```
xlen=input('Length(s): ');                             (1)
f=input('Frequency: ');
A=input('Amplitude: ');
while A>1
   A=input('Amplitude: ');
End                                                    (6)
fs=8000;
i=0:(1/fs):xlen;
x=A*sin(2*pi*f*i);
sound(x,fs);
```

Line one prompts the user for the length of the sound in seconds, and line two prompts the user for the frequency in Hz. Line three prompts the user for the amplitude, and the sequence from four to six ensures that the user does not exceed a ceiling limit of one. The sample frequency is defined to be 8000 samples per second by default. Line eight creates a series of variables i, which represent each sample that will be created. Finally, line nine creates the matrix x, which is the actual sinusoid. MATLAB can also combine sine waves through simple mathematical addition. The following example will produce a series of “beats,” caused by two slightly different frequencies interfering with each other.

```
xlen=input('Length(s): ');                              (1)
f=input('Frequency: ');
f2=input('Frequency 2: ');
A=input('Amplitude: ');
fs=8000;
i=0:(1/fs):xlen;                                        (6)
x=A*sin(2*pi*f*i)+A*sin(2*pi*f2*i);
sound(x,fs);
```

The difference between the frequencies f and f2 determines the frequency of the beat pattern that will be heard. Therefore, if f2 is only 5 Hz above f, the amplitude will fluctuate five times per second.

Ultimately, MATLAB is more powerful than VAB because it can take time to perform an unlimited amount of addition and multiplication in order to create very complex waves. VAB, however, is a much more practical application for most synthesis applications, because the sound is created in real-time, and characteristics such as frequency can be modified in real-time. VAB includes input devices such as graphical piano keyboards and knobs specifically for the purpose of allowing the user to create musical phrases.

Conclusion

When working with DSPs, there are often many choices and sacrifices that have to be made. One of the most important decisions to make is whether to use real-time or non-real-time processing. Our experience with DSPs suggests that both are ideal in certain situations. When using DSPs to alter audio signals in performance-altering ways, real-time processing is absolutely necessary. Many effects change sounds in ways that make it impossible to play correctly without hearing the effect itself. In a tremolo, for example, it is necessary to play in time with the amplitude oscillations, which is incredibly difficult to do if the oscillations cannot actually be heard. In a flanger, music is typically played in time to the sweep, and even echo effects have a particular tempo associated with them.

Non-real-time effects, however, are ideal for making powerful changes to audio when time or performance tempo is not an issue. When recording a song, effects such as chorus and artificial reverberation can easily be added after recording. These effects do not radically alter the sound itself; they merely enhance it, adding realism and fullness. They also have no tempo associated with them, which makes them easy to apply to any signal.

Generally, good results can be achieved with nearly any DSP. However, it is absolutely essential to have a full understanding of how the DSP works and how to implement your effect. When creating digital versions of analog effects, it is vital to research how the analog circuits work and decide how to accomplish the same tasks digitally. It is also always necessary to understand the limitations of your DSP. If you consistently try to get a DSP to do more than it is capable of, good results will be difficult to achieve. Ultimately, digital effect processing takes meticulous preparation and implementation.