**CSc 637 Spring 2009**

**Project 1: Echo, flange and sine generator**

(11% of your grade)

due in class (by 6pm), Monday 2/23/2009

You will write C/C++ code to implement three classic modules in computer audio:

1) an echo effect

2) a sinewave generator

3) a flange effect (which is pretty much a combination of an echo and a sine generator)

Both the echo and the flange effect take some input sound and transforms it into some desired output sound. The sinewave generator does not take any input sound.

For simplicity, your code only has to work with mono soundfiles, at sampling rate of 44.1 kHz.

I have provided an example that implements the simple delay effect,

sproc.cc, in http://unixlab.sfsu.edu/~whsu/csc637/P1/Code , or on thecity.sfsu.edu, in ~hsu/637/P1/Code . It uses the portable soundfile handling library libsndfile, and should work with most soundfile formats with one channel. The reference platform will be thecity.sfsu.edu (an x86 system running Linux). A Makefile is provided for thecity.sfsu.edu and most Unix flavors; Makefile.OSX is a Makefile for Mac OS X. If you'd like to work on other platforms, you'll need to get my approval before proceeding. You are implementing the modules as C++ classes, your code should be easily portable to different systems; the only necessary changes should be Makefiles. (I will consider other approaches to implementing the three modules, but you will need to get my permission first.)

You will need a sound editor/viewer to display and play the input soundfiles, and the soundfiles that you generate. Some useful applications can be found at <http://unixlab.sfsu.edu/~whsu/csc637/Software.html> . As mentioned earlier, I recommend Audacity: <http://audacity.sourceforge.net> . More on testing later.

In audio programming, you should always be careful not to create samples that are beyond the min/max range of the sample format. In sproc.cc, all audio samples are doubles, and should be clipped to a range of –1.0 to 1.0, before writing to an output soundfile.

For this project, you are required to write two effects: echo and flange. Flange is an extension of echo; you should work on echo first, get it working perfectly, then work on flange.

**1) Echo**: as described in lecture slides. (Your life will be easier if you work from the DelayEffect class and sproc.cc main program given in the code examples.) Your program takes an input sound, accepts delay (same as min delay, in ms), feedback gain, wet gain, and dry gain as parameters. Ignore all other parameters. It generates an output sound containing the original sound plus echoes. Your echo effect should be able to handle delays of up to two seconds.

Your program should stop generating echoes when it runs out of input sound samples. Hence, you should test your echo effect with a soundfile that has a short sound and a long period of silence, such as emma\_silence.wav from the example sounds. The output file should fill the silence with echoes.

**2) Sine generator:** Your program accepts a frequency, initial phase and duration as parameters, and generates a soundfile containing a sine tone with those characteristics. You should implement a sine generator class that generates a full-scale (i.e., amplitude = 1.0) sine wave at a specified frequency, with a given initial phase phi. (See Chapter 3 slides 2 and 3 for mathematical details.) The discrete time variable n should be treated as an internal state variable that each sine generator object keeps track of. To generate the next sample, call a *tick* method, similar to the delay class. The makesine.h header file in the code samples is an example of a (relatively) clean and modular implementation of a sine generator class.

**3) Flange:** This is identical to the echo effect, except the delay varies like a sinusoid; *do not implement the triangle wave-based variation described in the lecture slides*.

min delay

+ delay depth

time

min delay

1 / delay rate

You should allow the user to specify these parameters: wet gain, dry gain, feedback gain, delay rate (which is the frequency of the sinusoidal variation, in Hz) and delay depth (in ms), and min delay (in ms). Initially, the flange delay is at max delay. When you are sweeping the delay buffer, you may get a non-integer index for a buffer slot. Use linear interpolation to approximate slots with non-integer indices, as shown in lecture.

**Testing your code:**

You can make soundfiles for input data, recording from the microphone, from CDs etc. Some soundfiles are provided for testing your effects, in

<http://unixlab.sfsu.edu/~whsu/csc637/P1/Sounds/> or on thecity.sfsu.edu at ~hsu/637/P1/Data .

Some sounds may be more appropriate for testing one type of filter, but not for another type. You should pick test sounds so that it is obvious whether your filter is working.

A classic way to test a filter/effect in DSP is to use a *unit* *impulse* as an input; this is a “sound” that has a sample of maximum amplitude (1.0 for floats/doubles) at time = 0, and zero amplitude elsewhere. With this simple input, it is very easy to trace the behavior of the filter by monitoring each output sample; the output stream completely captures the functionality of the filter. A unit impulse soundfile (2 second duration), click.wav, is included online as one of the potential input sounds. You can’t hear very well click.wav or its output, but you can examine the actual samples in your favorite sound viewer.

With sample rates on the order of 44.1 kHz, even small delay times imply huge numbers of samples. (100 ms is 4410 samples!) Obviously it’s very tedious to browse through and count thousands of samples when debugging. A good approach is to specify unnaturally small delay times for debugging (such as 0.1 ms), then examine the output sample by sample. These tiny delay times do not yield audible results. After initial debugging, re-run the effect with larger delay times and acoustic input sounds, and listen to the output sounds for the expected results.

(I am happy to help with debugging, but when you come show me your code, you should already have utility code set up to print every input and output sample, and relevant parameters for your module.)

For example, to test "echo", developers usually use a short and distinct sound with a lot of trailing silence. click.wav is clearly the *extreme* of short and distinct sounds. A short and distinct sound that you can actually hear is emma\_silence.wav from the example sounds.

For each of the parameter settings below, use the specified input soundfiles, and compare your output soundfile with the soundfiles generated by my flange effect. Some implementation details might differ, so your result may not be identical to mine, sample for sample. However, the audible result should be very similar.

1) slow flange

input: violin.wav output: slowflange.wav

min delay (ms) = 0

delay rate = 1 Hz, delay depth (ms) = 6

dry gain = 0.0, wet gain = 1.0, feedback gain = 0

(You should hear a subtle distortion and throbbing "sheen" added to the original violin tone. The sheen will be more noticeable if you set the dry gain to 0.)

2) chorus

input: emma16.wav output: chorus.wav

min delay (ms) = 20

delay rate = 0.25 Hz, delay depth (ms) = 20

dry gain = 1.0, wet gain = 1.0, feedback gain = 0

(When tested on a single human voice input, the output should sound like several people talking at the same time.)

3) robot

input: emma16.wav output: robot.wav

min delay (ms) = 13

delay rate = 0 Hz, delay depth (ms) = 0

dry gain = 0, wet gain = 1.0, feedback gain = 0.75

(Makes human voice input sound like a robot.)

4) wacky pitchbend

input: emma16.wav output: wacky.wav

min delay (ms) = 40

delay rate = 2 Hz, delay depth (ms) = 40

dry gain = 0.0, wet gain = 1.0, feedback gain = 0.3

(Changes the pitch of human voice input in wacky ways.)

**Submission** **and format details:**

Submit a paper copy of your code, and (by email) a tar file containing the source code of each filter, any necessary header files, and a Makefile. You may assume that the libsndfile library is installed at ~hsu/libsndfile-1.0.17. Your tar file should expand into a directory with your username as part of the directory name, containing all your code. If there are “special” quirks I need to know about when compiling and running, there should be a README file.

I will create a script to compile and test your effects. The script will cd into the top-level directory that your tar file expanded into, and start with two lines to make echo and flange:

make Echo

make Sine

make Flange

Command line format for echo:

Echo [inputfile] [outputfile] delay dryGain wetGain feedbackGain

where

[inputfile] any input soundfile, e.g. in.aiff

[outputfile] output soundfile, same format as

input soundfile

delay echo delay in milliseconds (int)

dryGain dry gain (float)

wetGain wet gain (float)

feedbackGain feedback gain (float)

Command line format for sine:

Sine [outputfile] frequency phase duration

where

[outputfile] output soundfile, SR = 44.1 kHz,

1-channel, .wav format, 16-bit

samples

frequency frequency of sine wave in Hz

(float)

phase initial phase in radians (float)

duration duration of output sound in ms

(float)

Command line format for flange:

Flange [inputfile] [outputfile] minDelay delayRate delayDepth dryGain wetGain feedbackGain

where

[inputfile] any input soundfile, e.g. in.aiff

[outputfile] output soundfile, same format as

input soundfile

minDelay min flange delay in millisec (int)

delayRate delay rate in Hz (float)

delayDepth delay depth in millisec (int)

dryGain dry gain (float)

wetGain wet gain (float)

feedbackGain feedback gain (float)

If you would like to use a different platform, or make other changes to the specifications, you must get my permission first, or penalties will be applied.

**Documentation and Style**

I expect basic standards of documentation (descriptive file headers, moderate amounts of comments where appropriate, indentation etc); nothing fancy, but you should not be embarrassed to show your code at a job interview. Obviously ugly code may be penalized.

Score breakdown: 90% for correctness, 10% for style/documentation.