MUSC 208 Winter 2014
John Ellinger Carleton College

Lab 15 Time Domain Effects

Overview

Lab 15 will create a SndBuf utility class in ChucK that contains methods to report information about a wav file's minimum, maximum, peak to peak, and root mean square (RMS) amplitude, and whether there are clipped samples. The class also contains functions to convert amplitude to decibels , decibels to amplitude; remove DC offset; normalize the file so the maximum amplitude is 1.0; and amplify or attenuate the file to a set dB level. The class will read the sound samples into an array and then operate on the array to collect information and modify the wav file.

The class will be declared as a public class so that it can be used by other ChucK programs or projects you use in the future. A public class must be written in a separate file and only one public class can be defined in a single file. The public class file must be executed before executing other files that use that public class.

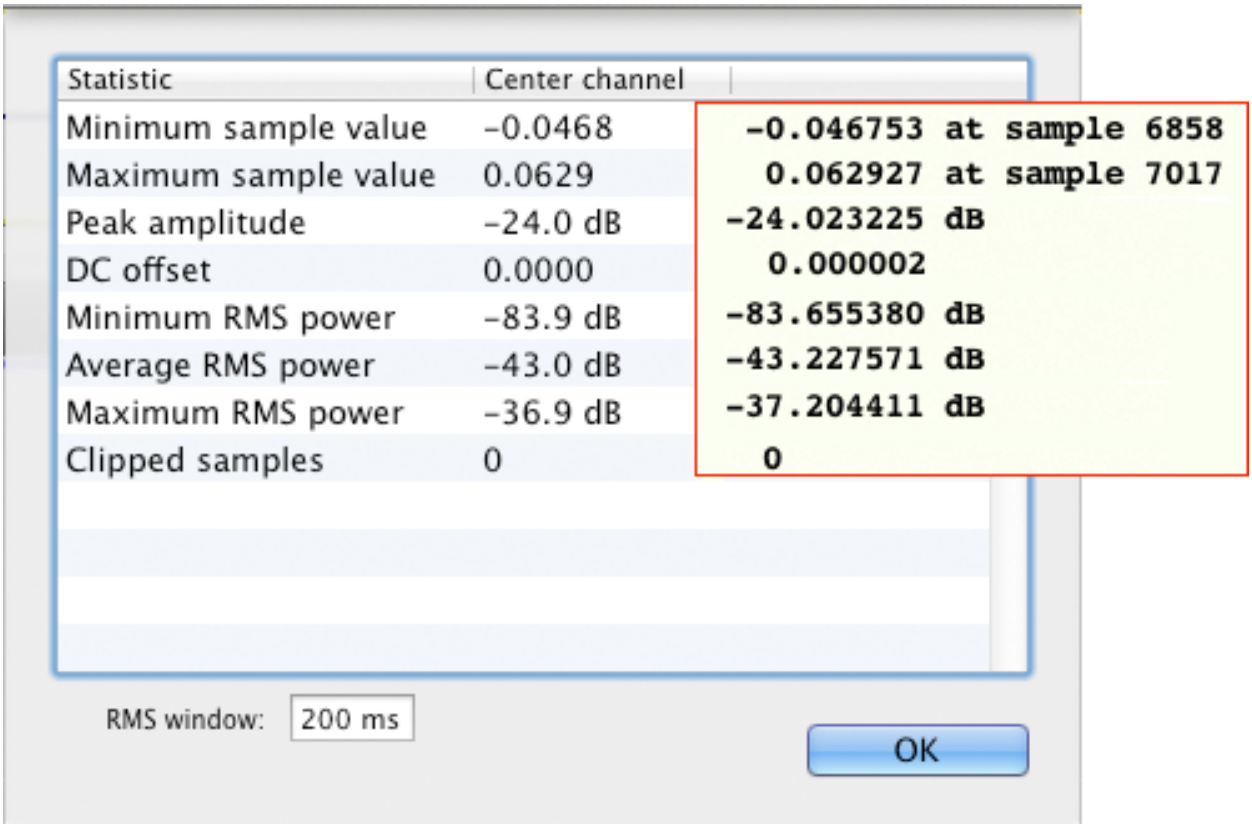
The Audio file we'll be working with is "music208minus24dB.wav" a file that is deliberately attenuated (reduced in amplitude).

The SndBufUtils class we'll create during Lab15 reports this information about the music208minus24dB.wav.

```
Sound file name
          music208minus24dB.wav
Length in samples
          108861
Duration
          2468.503401 ms
          2.468503 seconds
          0.041142 minutes
Minimum amplitude
          -0.046753 at sample 6858
Maximum amplitude
          0.062927 at sample 7017
          -24.023225 dB  Hence the name
Peak to peak amplitude
          0.109680
DC Offset
          0.000002
Minimum RMS amplitude
          0.000066
          -83.655380 dB
Min RMS sample range
          44100 - 52920
Average RMS amplitude
          0.006896
          -43.227571 dB
Maximum RMS amplitude
          0.013797
          -37.204411 dB
Max RMS sample range
          0 - 8820
Maximum RMS amplitude
          0.013797
          -37.204411 dB
Clipped sample count
          0
```

Sound Editing Software

AmadeusPro is a commercial sound editor available at http://www.hairersoft.com. It is only mentioned here because it shows similar statistics to those we'll be developing in Lab 15. Our class is shown in the yellow box.



Audacity

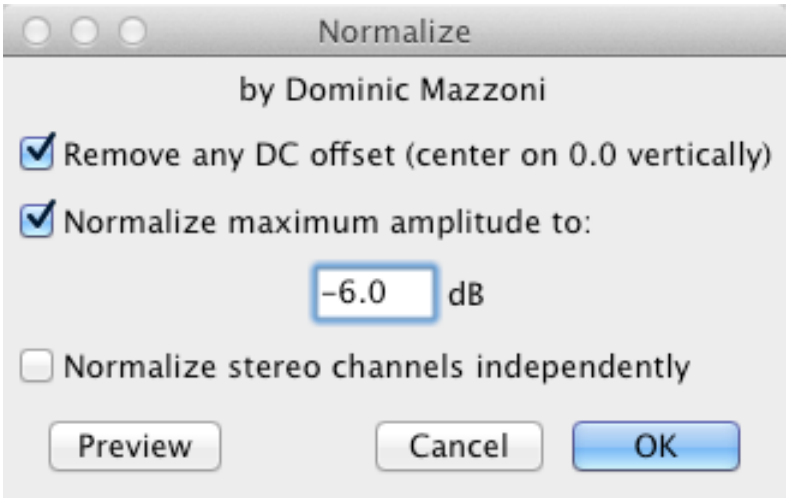
Lab 15 will implement Audacity information and three Audacity Effects.


```
Length in samples
          108861
Duration
          2468.503401 ms
          2.468503 seconds
          0.041142 minutes
```

Remove DC Offset and Normalize

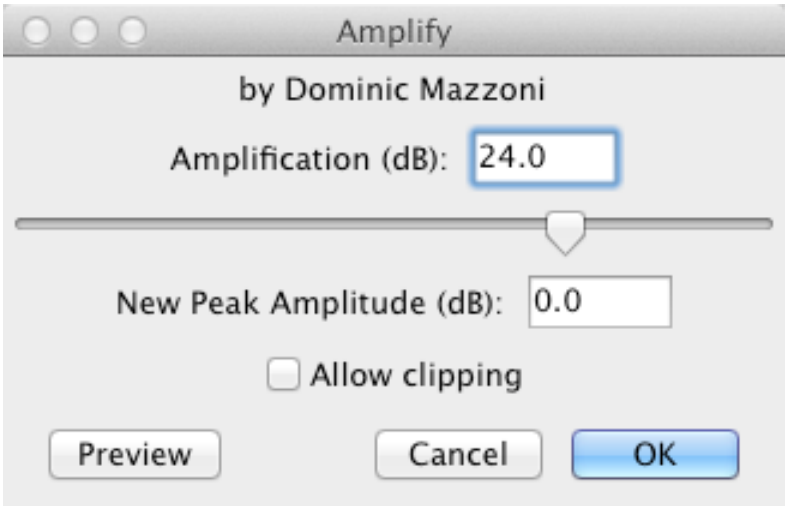
DC offset removes any vertical amplitude offset to center the amplitude values between ±1.

The Normalize command will scale all amplitude values so that the largest amplitude peak is at a given decibel leve. In this example -6.0 is roughly 0.5 on the ±1 amplitude scale.



Amplify Command

The amplify command shows the amount of amplification necessary to reach a given decibel level. For our example file the maximum amplitude was -24 dB and Audacity shows that we need to amplify by 24 dB to reach 0 dB.



Create The SndBufUtils Test Framework

Select and copy this code into a new miniAudicle file. Save it as SndBufUtil.ck. We'll build the functions in the class one step at a time.

Setup

Download m208Lab15files.zip. Open the SndBufUtils.ck file in miniAudicle or in your text editor if you prefer running ChucK from the Terminal.

Incremental Implementation

We're going to build the code one function at a time, testing each one before doing the next. We'll begin with the init function.

Step 1

Write the init function: `function void init( string filename )`

Write the info function: `function void info( string header )`

Test 1

Add these lines after the end of the SndBufUtils file.

```
// TESTING
function void playIt( int play )
{
     if (play != 0) // not equal to zero
     {
          0 => buf.pos;
          buf => dac;
          buf.length() => now;
     }
}

init( me.sourceDir() + "/music208minus24dB.wav" );
info( "Test 1" );
// set to 0 to not play, 1 to play
playIt( 1 );
```

The info function should display this information

```
==== Test 1 ====
Sound file name
      /Users/je/Sites/m208w2014_scrivener/_labs/Lab15/
music208minus24dB.wav
Length in samples
      108861
Duration
      2468.503401 ms
      2.468503 seconds
      0.041142 minutes
```

Step 2

Write the fillSampRA function: `function void fillSampRA( )`

Test 2

Add this line to the info function

```
<<< "sampRA[0] to sampRA[4]", sampRA[0], sampRA[1], sampRA[2], sampRA[3] >>>;
```

Run the program. Oh, Oh, looks like there's an ArrayOutOfBounds error.

```
==== Test 1 ====
Sound file name
      /Users/je/Sites/m208w2014_scrivener/_labs/Lab15/music208minus24dB.wav
Length in samples
      108861
Duration
      2468.503401 ms
      2.468503 seconds
      0.041142 minutes
[chuck](VM): ArrayOutofBounds: in shred[id=1:SndBufUtilsJE.ck], PC=[43], index=[0]
```

The problem is that the variable sampRA was defined with only one element.

```
float sampRA[0]; // necessary to initialize one element
```

This was necessary so we can resize it later. Add these lines to the end of the init() function.

```
// increase sampRA[] size
lenSamples => sampRA.size;
fillSampRA();
```

Run the program again. It should work this time.

```
==== Test 1 ====
Sound file name
       /Users/je/Sites/m208w2014_scrivener/_labs/Lab15/music208minus24dB.wav
Length in samples
       108861
Duration
       2468.503401 ms
       2.468503 seconds
       0.041142 minutes
sampRA[0] to sampRA[4] -0.001221 -0.001221 -0.001129 -0.001190
```

Important: All steps from now on will work with the sampRA[] and leave SndBuf buf unchanged.

Step 3

Write the two utility functions db2amp() and amp2db().

Hint: Use the Math.pow() and Math.log10 functions in ChucK.

Refer to http://chuck.cs.princeton.edu/doc/program/stdlib.html if needed.

```
function float db2amp( float dB )

function float amp2db( float amp )
```

Test 3

Change the playIt() function parameter to zero. Add these lines at the end of the file the playIt(0) function.

```
<<< "amp2db", amp2db( 0.00001 ) >>>;
<<< "amp2db", amp2db( 0.5 ) >>>;
<<< "amp2db", amp2db( 1.0 ) >>>;

<<< "db2amp", db2amp( 0 ) >>>;
<<< "db2amp", db2amp( -6 ) >>>;
<<< "db2amp", db2amp( -20 ) >>>;
```

You should see these results in addition to the previous output.

```
amp2db -100.000000
amp2db -6.020600
amp2db 0.000000
db2amp 1.000000
db2amp 0.501187
db2amp 0.100000
```

Step 4

Write the getMinAmp() function: `function float getMinAmp()`

Test 4

Add this line at the end of the init() function.

```
getMinAmp() => minAmp;
```

Modify the info() function. Comment the samp[0] line and add this line

```
// <<< "sampRA[0] to sampRA[4]",sampRA[0],sampRA[1],sampRA[2],sampRA[3] >>>;
<<< "Minimum amplitude", "" >>>;
<<< "\t", getMinAmp(), "at sample", minSampIndx >>>;
```

Comment out the amp2db and db2amp tests at the end of the file. Run the program.

```
init( me.sourceDir() + "/music208minus24dB.wav" );
info( "Test 4" );
// set to 0 to not play, 1 to play
playIt( 0 );
/*
<<< "amp2db", amp2db( 0.00001 ) >>>;
<<< "amp2db", amp2db( 0.5 ) >>>;
<<< "amp2db", amp2db( 1.0 ) >>>;
<<< "db2amp", db2amp( 0 ) >>>;
<<< "db2amp", db2amp( -6 ) >>>;
<<< "db2amp", db2amp( -20 ) >>>;
*/
```

Output

```
==== Test 4 ====
Sound file name
       /Users/je/Sites/m208w2014_scrivener/_labs/Lab15/music208minus24dB.wav
Length in samples
       108861
Duration
       2468.503401 ms
       2.468503 seconds
       0.041142 minutes
Minimum amplitude
       -0.046753 at sample 6858
```

Step 5

Write the getMaxAmp() function: `function float getMaxAmp()`

Test 5

Add this line to the end of the init() function.

```
getMaxAmp() => maxAmp; // also sets maxSampIndx and maxDB
```

Modify the info() function. Add this line at the end.

```
<<< "Maximum amplitude", getMaxAmp(), maxDB, "(dB)", "at sample", maxSampIndx
>>>;
```

Run Test 5

```
init( me.sourceDir() + "/music208minus24dB.wav" );
info( "Test 5" );
// set to 0 to not play, 1 to play
playIt( 0 );
```

Output

```
Maximum amplitude
       0.062927 at sample 7017
```

Step 6

Write the getPeak2PeakAmp() function:

```
function float getPeak2PeakAmp()
```

Hint: Look up the Math.abs and Math.fabs methods.

http://chuck.cs.princeton.edu/doc/program/stdlib.html

Test 6

Add this line to the end of the init() function.

```
getPeak2PeakAmp() => p2pAmp;
```

Add this line to the end of the info() function.

```
<<< "Peak to peak amplitude", getPeak2PeakAmp() >>>;
```

Run Test 6

```
Peak to peak amplitude
       0.109680
```

Step 7

Write this function: `function float getDCOffset()`

As a general rule you want all audio signals to be equally centered above and below zero. DC offset refers to the amount the mean (average) of all amplitudes varies from zero. When DC equals zero the sum of positive amplitudes plus the sum of negative amplitudes equals zero.

Test 7

Add this line to the end of the init() function.

```
getDCOffset() => dcOffset;
```

Add this line to the end of the info() function.

```
<<< "DC Offset", getDCOffset() >>>;
```

Run Test 7

```
DC Offset
       0.000002
```

Step 8

Write this function: `function float getAvgRMSAmp()`

RMS stands for Root Mean Squared. The regular mean (DC offset) is not a very good indication of the power or intesity of the samples because the positive and negative amplitudes cancel each other. The most common method to see if one sound file is louder than another is to square all the amplitude values and add them up resulting in a positive number. Take the square root of that sum and you have the RMS average amplitude. You'd use this method if you were

creating a playlist and wanted all the songs to be at the same relative volume to one another.

Test 8

Add this line to the end of the init() function.

```
getAvgRMSAmp() => avgRMSAmp; // also sets avgRMSdb
```

Add this line to the end of the info() function.

```
<<< "Average RMS amplitude", getAvgRMSAmp(), avgRMSdb, "(dB)"
>>>;
```

Run Test 8

```
Average RMS amplitude
       0.006896
       -43.227571 dB
```

Step 9

The average RMS value reports the average power in the whole file. If the file has several loud sections and several soft sections you wont get sense of where the maximum power is. This function will find the RMS maximum amplitude over small "windows" of samples specified in millisecond width.

Write this function: `function float getRMSMaxAmp( float millis )`

Test 9

Add this line to the end of the init() function.

```
getRMSMaxAmp( 200 ) => maxRMSAmp;  // also sets maxRMSdb, maxRMSstartSamp,
maxRMSendSamp
```

Add these lines to the end of the info() function.

```
    <<< "Maximum RMS amplitude", "" >>>;
    <<< "\t", getRMSMaxAmp( 200 ) >>>;
    <<< "\t", maxRMSdb, "dB" >>>;
    <<< "\tsamples", maxRMSstartSamp, "-", maxRMSendSamp  >>>;
```

Run Test 9.

```
Maximum RMS amplitude
       0.013797
       -37.204411 dB
      samples 0 - 8820
```

Step 10

Write this function: `function float getRMSMinAmp( float millis )`

Similar to getRMSMaxAmp();

Test 10

Add this line to the end of the init() function.

```
getRMSMinAmp( 200 ) => minRMSAmp;  // also sets minRMSdb, minRMSstartSamp,
minRMSendSamp
```

Add these lines to the end of the info() function.

```
    <<< "Minimum RMS amplitude", "" >>>;
    <<< "\t", getRMSMinAmp( 200 ) >>>;
    <<< "\t", minRMSdb, "dB" >>>;
    <<< "\tsamples", minRMSstartSamp, "-", minRMSendSamp  >>>;
```

Run Test 10

```
Minimum RMS amplitude
       0.000066
       -83.655380 dB
      samples 44100 - 52920
```

Step 11

Write this function: `function void removeDCOffset()`

As a general rule you should always remove the DC offset from audio files before doing any processing on them.

Test 11

Modify the test code.

```
// TESTING
init( me.sourceDir() + "/music208minus24dB.wav" );
// set to 0 to not play, 1 to play
playIt( 0 );
info( "Test 11 before removeDCOffset" );
removeDCOffset();
info( "Test 11 after removeDCOffset" );
```

Run the program

The DC Offset should be zero after this step

Step 12

Write the normalize() function.

This function will make the sound as loud as possible without clipping.

```
function void normalize()
{
    // call removeDCOffset();
    // get the maximum amplitude
    // get the minimum amplitude
    // find the absolute value of the minimum amplitude
    // choose the larger of maxA or Math.fabs( minA )
    // determine the amplitude scale factor
    //      1/maxA => scaleBy;
    // loop through sampRA and multiply all values by scaleBy
```

```
}
```

Test 12

Modify the testing code.

```
// TESTING
init( me.sourceDir() + "/music208minus24dB.wav" );
// set to 0 to not play, 1 to play
playIt( 0 );
info( "Test 12 before normalize" );
removeDCOffset();
info( "Test 12 after normalize" );
```

Run the program

Step 13

Write the amplify() function.

```
function void amplify( float dB )
```

Test 13

Modify the test code.

```
// TESTING
init( me.sourceDir() + "/music208minus24dB.wav" );
// set to 0 to not play, 1 to play
playIt( 0 );
info( "Test 13 before amplify" );
amplify(-6);
info( "Test 13 after amplify" );
```

Run the program

Step 14

Now that everything is working let's turn SndBufUtils into a public class that can be used by other files.

Add public class definition at the beginning of the fiel

```
public class SndBufUtils
{
    SndBuf buf;
    string fname;
    // reporting variables
    int lenSamples;
```

Add the end class closing bracket just before the TESTING commeng.

```
}; // end class

// TESTING
```

Delete the TESTING section.

Save the class as SndBufUtilsClass.ck

Use the SndBufUtilsClass Public Class

There is are a few things to know about public classes.

1. Only one public class is allowed in any one file.

2. Once the public class is running, you cannot make changes to it unless you quit and restart the Virtual Machine.

3. It is a good way to share global variables between several files.

Stop the Virtual Machine. Start the Virtual Machine.

Run SndBufUtilsClass.ck

Create a new miniAudicle file called SndBufUtilsDemo.ck and add this code to

play the samples in sampRA[].

Run it. It is a very low volume sound file.

```
// SndBufUtilsDemo.ck
// Run SndBufUtilsClass.ck before running this demo
// John Ellinger, Music 208, Spring2013

Impulse imp => dac;
"music208minus24dB.wav" => string fname;
SndBufUtils sbu;
sbu.init( fname );

sbu.fillSampRA();
for ( 0 => int n; n < sbu.lenSamples; n++ )
{
    sbu.sampRA[n] => imp.next;
    1::samp => now;
}
sbu.info("Original");
```

Let's make it louder using the normalize() function. Add this code to the demo.

```
// Normalize
sbu.normalize();
for ( 0 => int n; n < sbu.lenSamples; n++ )
{
    sbu.sampRA[n] => imp.next;
    1::samp => now;
}
sbu.info("Normalized");
```

Let's set the loudest amplitude level to -6 dB. Add this code to the demo.

```
27 // Amplify
28 sbu.fillSampRA();
29 sbu.amplify( -6 );
30 for ( 0 => int n; n < sbu.lenSamples; n++ )
31 {
32     sbu.sampRA[n] => imp.next;
33     1::samp => now;
34 }
35 sbu.info("Amplify to -9 (dB)");
```

End of Lab 15