COSC 4P98 Assignment 1: Intro to Digital Audio

Due date: 9:00am Monday October 17, 2022. No lates accepted.

Hand in: Electronic submission of source code, audio files, example output, plots. See the end of assignment for format of electronic hand-in.

General comment: It is recommended that you use an ASCII TXT format audio file. Audacity handles TXT files, and the older Audition format has been used with "txt-to-wave" tools available on the 4P98 web site. This format will let you inspect sample values, and can be easily imported into Excel for plots. However, you may also use a suitable WAV I/O library if you like, so long as it doesn't do the processing tasks required in the assignment. (See final page of assignment for more details.)

1. Audio sampling and synthesis

The basic ideas of sampling and sample playback were discussed in the lectures. This question will have you generate your own sound files using various concepts in sample generation assignment.

(a) Generate separate audio TXT files that have the following. You should write code that creates (synthesizes) samples when appropriate. Also include a graphical time-amplitude plot of each wave, perhaps by importing the TXT files into Excel and plotting the data. Clearly label each plot.

i) Sine wave at 440 Hz.
ii) Square wave at 440 Hz.
iii) Sine wave at 880 Hz.
iv) Sawtooth wave (any audible frequency)
v) Random noise (generate 2 seconds duration)
vi) A 2 second sample from a recorded source. You can record your own, or convert part of an MP3 file into a WAV file.
vii) Using sample repetition, convert the sample in (vi) to one that is half the pitch (twice the duration).
viii) Repeat vii, but this time use linear sample interpolation between samples. In other words, rather than repeating each sample, you should insert new samples that are mid-way (average) between previously existing samples.

(b) Implement a utility that takes a sample table (TXT file), phase index, and duration, and generates samples according to the following parameters. The duration can be in seconds or # samples. Use linear interpolation of samples for fractional phase indexes. You can write the generated output directly into a TXT file. Be sure to give the utility the information necessary to generate a complete TXT file that can be imported into (say) Audacity, or can be converted with the TXT-to-WAVE tools on the 4P98 web site.

Test your utility on one of the tone samples you used in part (a) (other than (v) noise), by generating a TXT file that has short durations of 24 notes for the sample from -12 semitones to +12 semitones from your source sample (ie. -1 octave to +1 octave).

(c) Bit crushing (quantization): Convert a sample to a lower quantization level that is less than 8 bits. (Note: 2 bits is too much).

2. Low-Frequency Oscillator and Sample Mangling

Low frequency oscillators (LFO's) are signal generators that generate a sub-audible frequency (much lower frequency than audible sounds). For example, a sine wave whose frequency is 1 cycle/second (1 Hz) is a typical signal of an LFO. This signal can be used to control other parameters used in samplers and synthesizers, such as pitch, amplitude, or sample index (as in this question).

This question involves enhancing your code in question 1b, to make a more dynamic sample playback engine. The idea is that you will supply a LFO source that will take the place of the static phase index value in question 1b. Imagine replacing the single phase index (perhaps set to "2") to be the value of a LFO sine wave. This sine wave may go from 0.5 to 4, at a particular frequency. You should have your playback engine accept some suitable parameters for this LFO sine wave, indicating its amplitude (min and max) as well as rate (in time units or # samples per second).

Note that when the phase index is negative, you traverse the sample table in reverse. For example, an index of -1 means you are playing the sample backwards. Some wild audio effects arise when you allow the phase index to fluctuate between positive and negative values!

Test your LFO by generating TXT audio files that use various LFO parameters. You should include one test that oscillates the LFO between -2 to 2, using a frequency of 1 Hz. Test this on an audio recording (e.g. the long sample in 1a(vi)).

3. DSP

Write a utility that takes a sample table, and applies a DSP effect to it. You are free to make any reasonable effect here, so long as it isn't too trivial. (Trivial effects are: increase the amplitude, reverse the sample, etc.). Remember: audio is data, and so there are lots of interesting algorithmic transformations that are possible. Some ideas...

- Reverb: this is a fast-response delay (< 35 ms), which does not mix with itself more than once. Weights should be used when mixing the signals, or else saturation (overflow) will occur.
- Echo: This is a longer-response delay (> 35 ms). It also mixes onto itself. In other words, you can mix the echoed signal repeatedly with a (weighted) version of itself. Weights are important!
- Auto-Panning: this moves the left and right channels of a stereo signal around with an LFO signal. The LFO may be a sine wave, square wave, etc.
- Wave chopper: Slice and dice a sample table! For example, break it up into 0.2 second chunks, and reverse every other chunk. Or, repeat chunks. Or shuffle their order. Or...?
- Normalize: Normalize the sound file, as discussed in class.
- Hiss and flutter removal (as discussed in class).
- ???

Whenever possible, parameterize your DSP. For example, for the reverb, an obvious parameter is the delay time. Also try to avoid saturated audio (overflow), which will create distortion (unless that is an intended effect!).

Test your DSP with a couple of recorded sample files, and parameter variations if appropriate. Include dry (original pre-processed sample) and wet (processed) samples.

General advice:

You can use any programming language for these questions that is supported in the department's labs (Windows or Linux). The marker should be able to compile and run your assignment in our lab. Please ensure this before submitting your assignment.

Rather than work directly in an audio file format, the Adobe TXT audio format is recommended. It was originally supported by Adobe Audition – an audio editor. It lets you easily generate and modify samples as ascii text, without the need for special audio libraries and audio compression algorithms. It uses an integer ASCII representation. You can even create audio data with a simple text editor. Please look here at examples of TXT files, to see the file format (which is easy to figure out):

http://www.cosc.brocku.ca/Offerings/4P98/waves/Audio_txt_format_files/

Also available on the 4P98 web site is a set of TXT-to-WAV utilities ("wavtxt_tools") for Linux and Windows. These convert between the WAV and TXT audio file formats (the WAV format is a common standard). Then you can use any audio editor to listen to your audio, or edit and create new audio. The source code (Linux) is available, should you want to see how the conversion is done using the *sndfile* library. The "readme.txt" file in the tools zip file shows how to run these command-line tools.

Alternatively, you can use the TXT format that is supported by Audacity. Descriptions on web sites are included on the 4P98 page. It is not quite as self-contained as the Adobe TXT above, however, as you need to supply important parameters to Audacity's TXT import/export tool. Its format uses floating point ASCII.

Submission Requirements:

The entire assignment is to be submitted electronically on Sakai. The submission should be in the following format, in order to organize the submission for the marker:

1. The 3 questions of the assignment should be in 3 top folders, suitably labelled. There should be a signed department cover page at the top level (scanned or photographed).

2. There should be a "Readme" file in each folder. This file will describe everything required to compile and run that assignment question. It should also describe all the files relevant to your own executions of the system. It should include:

a. A list of the relevant source code file(s), with descriptions of each file. If the question involves your own design of an application (e.g. DSP effect), you should describe the effect, and how it works.

b. Description of how to run the application, including the means for setting new arguments (files, values,...).

c. A list of example input/output cases for the question. These are the executions that you have already done for the question. You should carefully describe each case:
   i. the parameters used
   ii. the input and output files involved.
   iii. Excel plots, etc.