Assignment 3: Granular, CSound, VST ...

Due date: **9:00am Monday December 5. No lates accepted.**

Hand in: Electronic submission of all source code, data files, example output, plots. Clearly name all files with respect to the question and part number that they are solutions to. (See assignment 1 for hand-in requirements).

Comments:
- This assignment will require you to do some independent reading on relevant topics, such as VST programming or music composition environments. Make good use of online tutorials and other resources. But do not submit a tutorial as your work!
- You are permitted to work either solo or in groups of 2 on this assignment. However, **both group members must contribute equally to everything submitted.** It is not acceptable for groups to split the questions between themselves (for example one person does question A, the other does question B).

Choose 2 of the following:

1. Granular synthesis engine

Implement an asynchronous wavetable granular synthesis engine! It will generate an audio file using granular synthesis. The engine will use a few ideas from your sample playback engine from assignment 1(b). However, one difference is that grains can mix together, depending on the rate (density of cloud). This means you should mix them in a buffer before dumping to a file. Your engine should minimally have the following parameterized options:

a) duration of individual grains.
b) rate of grain generation (density of cloud)
c) pitch of grains
d) location of grain cloud within wave table: center location, and span of cloud in the table (ie. min and max locations from center, from which grains are generated).
e) overall duration of grain cloud (total number of grains generated)

You must also apply an envelope to each grain generated. This can be a simply linear ramp which makes the grain start at zero, and end at zero.

In the above, parameters a, b, and c, should also have some sort of selectable random deviation parameter for them. For example, duration in (a) might be in milliseconds, plus/minus some random percentage (25 plus/minus 3 ms, randomly determined per grain). These can be turned off too (default is zero?).

Feel free to add more parameters that you can think of.

Test your engine by generating a variety of audio files from source files, using various parameter settings. Hand in documentation that reports on the design and architecture of your engine, describes the parameters and examples of their effect (TXT files). Be sure to include the original unprocessed audio file as well. Document your example files in detail, so that the marker can reproduce them.

Note: If you add a suitable GUI, file I/O dialogs, real-time audio playback, (and hey, what about a port to a Rasp-pi and controller) this can be a suitable course project.

2. CSound composition

First, work through the exercises in chapter 1 of *The Csound Book*. This chapter is available online at: http://csounds.com/chapter1/index.html

Next, create a composition with Csound! You can do whatever you want, and use as much of Csound as you wish. Go wild! You should do the entire composition in CSound. Although external audio files will be used within the composition, please don't do the major composition in another system, e.g. FruityLoops).

The minimal requirements are:

a) The composition should be between 1 minute and 12 million years long.
b) Invent at least one unique instrument with CSound. Feel free to use many more.
c) You should process at least one wave file as input.
d) Use at least one of the granular synthesis functions (grain or granule).
e) Your final composition should use stereo channels (L and R).
f) Convert your final wave file output to MP3. Audacity will convert to mp3.
g) Describe your composition on a document, eg. PDF or Web page. (see “Hand in” below)

Some ideas for inspiration...

1. Make an experimental EDM composition. One genre to consider is Glitch, which sounds like a Nintendo Gameboy that somebody dropped on the floor. Use samples of machines, devices, bleeps, clicks, etc., which you have recorded yourself, or found on Creative Commons sample sharing web sites.

2. Make an experimental composition that uses processed variations of one single wave file as an input source. With the use of creative processing, filters, and granular synthesis, you can make a very interesting and complex composition.

3. You may use other tools that help you with your composition. There are a number of free midi/audio sequencers (eg. Rosegarden on Linux, Garageband on Mac, Protools First on Mac/Win, and many time-limited demos of commercial products) and CSound environments. There are also fractal composition programs available. You can use wav editors to help tailor sound files. I will permit you to use these tools to organize, sequence, and post-process (reverb, echo,...) your final composition, so long as you do use CSound as the primary means for creating source sound files, as described in the minimal requirements above. Be sure to document all the tools you use, and how you use them.

4. Remember that you can take audio files created from CSound, sequence them in a sequencer (or CSound itself), generate the resulting output to a wave file, and then put that wave file back into CSound for more processing, for example, granular synthesis effects. This processed file can be mixed with the original in a sequencer if you wish.

Hand in: all Csound files, data files, audio files, MP3 output, as well as a **written web page** or **PDF document** describing your composition: its structure, special techniques and effects, and a description of your invented instrument (including a **dataflow definition diagram**). Please acknowledge the sources for all borrowed instruments and wave files. Past examples are on the 4P98 web site.

3. VST Audio Effect Plug-in

This question requires that you implement a small VST 2 dll plug-in that will plug into Ableton Live or other VST host. Although VST supports both MIDI and audio effects, you should implement an audio effect here. Some effects to consider are:

- **Delay** (echo or reverb): Mix a sample with earlier samples in the audio stream. The "adelay" in the tutorial(s) is a basis for this effect. If you implement a delay effect, create some new parameters to enhance the tutorial's example. Note that you should be able to control the parameters with Ableton's interface.
- **Distortion**: think of ways to add distortion to audio. At a basic level, it may involve adding random values to samples. But different techniques will give different kinds of distortion, for example, bit crushing.
- **Hiss removal**: implement the "moving average" filter as a VST plugin. The window size and weights would be parameters.
- Implement your audio effect from assignment 1 (might be one of the above!).
- Implement a **granular delay** effect. It's like a delay, but it mixes delayed grains (similarly to question 1). Then you can process the sample data from the host application. See Ableton's built-in Grain Delay effect.

Besides the above, you are free to try any idea you might think of. But reserve more significant effects or instruments for your 4P98 project (talk to me).

Implementation options...

<u>Option 1</u>: Implement the plug-in in **C++ and VST 2**. This will require that you use the VST 2 SDK auxiliary library, which is available on the 4P98 web site. The SDK gives you access to a number of features within the host. However, for the purposes of this assignment, you will need only a few utilities. To begin, you should work through the excellent VST programming tutorial by Rick Strom (see end of assignment for location). The tutorial takes you through the complete process of implementing a basic plug-in in the Visual Studio C++ environment. The VST SDK also has some basic introductory documentation as well.

<u>Option 2</u>: Implement the plug-in in **C++ and VST 3 SDK**, within the **JUCE** environment. JUCE is a popular framework for commercial desktop and mobile applications. It has very good support for audio, midi, and VST development. In fact, Korg Gadget was developed with it. If you select this option, please go to the JUCE website and study the online tutorials.

<u>Option 3</u>: Implement the plug-in in Java, using **jVSTwrapper**, which is an environment for Java-based VST plug-ins. As above, you should consider implementing an audio effect or instrument. Although Java programmers might prefer this wrapper, a disadvantage is the CPU overhead involved in running Java (the C++ plug-ins are much more efficient). You will have to read the jVSTwrapper documentation for information on its use. The tutorials mentioned in options 1 and 2 are still worth studying.

<u>Hand in</u>: Source code, audio examples of the plugin working (if possible), and a small written report describing all aspects of your plug-in, including architecture, user parameters, algorithms, etc.

4. SuperCollider composition

This is identical to (2) above, except use SuperCollider instead of CSound. Work through introductory tutorials first.

5. Processing and Java Beads composition

Ditto to (3) and (4), except you are to use Processing and Java Beads. Work through introductory tutorials first.

6. Pure Data (Pd) composition.

Ditto, but use Pure Data. Work through the online tutorials first.

Comment: If you do one of question 4, 5, or 6, then the platform used may be a great seminar topic!

Resources

VST 2 SDK (local):	http://www.cosc.brocku.ca/Offerings/4P98/vst/
VST 2 Tutorials:	... vst/strom_VST_tuts.zip
	... vst/vst20spec.pdf
VST 3 SDK download:	https://www.steinberg.net/en/company/developers.html
JUCE:	https://www.juce.com/
JUCE tutorials (incl. VST):	https://www.juce.com/tutorials
jVSTwrapper:	http://jvstwrapper.sourceforge.net/
CSound:	http://www.csounds.com/
CSound book, chapter 1:	http://csounds.com/chapter1/index.html
SuperCollider community:	http://supercollider.sourceforge.net/community/
SuperCollider tutorials:	http://supercollider.sourceforge.net/learning/
Java Beads project:	http://www.beadsproject.net/
Beads demos:	http://www.beadsproject.net/?page_id=68
Processing:	http://www.processing.org/
Pure Data	http://puredata.info/

See 4P98 web site for additional resources.