Nick Creel

Sound Design

3 May 2017

Final Project: Music and Visualizations

For my final project, I wanted to continue experimenting with GEM. I was inspired by, and borrowed some elements from, the live coding example that Lysha shared with us, and attempted to create something similar. I also wanted to attempt to make a richer sounding music sequence; rather than just having an oscillator doing something random, I decided to use two different types of oscillators and incorporate a simple drum kit to add to the sound. Nearly everything that produces sound also has an effect on the GEM object, creating a visualization that reacts to the music.

Frequency Modulation Synth and Step Sequencer

For the lower frequency oscillator, I created a frequency modulation synth that runs on a four step sequencer. This produces the deep “wobble” in the background. A step sequencer is composed of a metro object, a counter, and a route object. Every time the metro object sends a bang to the counter, it adds 1 to the total amount that has been counted. Using mod division on this value gives a range of numbers that the counter can produce. Creating a route object that takes these numbers and bangs the object that is connected to the corresponding number value allows for easy sequencing of events. For this specific step sequencer, a metro object sends a signal to the counter every 250 milliseconds. The values generated by the counter are subject to mod division by 4. Therefore, the sequencer has the option to choose between values 0, 1, 2, and

3 (there's also a fifth option for any input that does not equal 0, 1, 2, or 3, but I do not use that option here).

Each step on the step sequencer changes the FM index and the modulator frequency. The FM index controls how much the modulator frequency affects the carrier synthesizer, and the modulator frequency controls the frequency of the carrier synthesizer. Changing the index without changing the modulator frequency will generally not affect the synth – however there will be a slight change here because the sliders I have created for controlling the modulation frequency have a minimum value higher than 0. This frequency modulation synth is also interesting because the values for FM index and modulation frequency are not the only values going into the osc~ objects – there are line~ objects, which cause the output to ramp between two (or more) values rather than just relying on one value.

The cos~ object is very important in this synth. Without it, the synth would be a amplitude modulation synth rather than a frequency modulation synth. The wave would also have the form of a sawtooth wave rather than a sine wave. Without the cos~ object, the FM index and modulation frequency values would cause a ramp in amplitude rather than frequency.

Drum Kit

I've wanted to explore creating a drum kit in Pd for a while, but have been intimidated by the prospect. I did not have a clear understanding of the line~ objects and their function and thought that my lack of knowledge would be a hindrance, but making a simple drum kit actually helped me understand how line~ objects worked.

The simplest way to make a drum kit is to try applying different attack and delay signals to a noise~ object. This is done using line~ objects. Rather than the line~ objects affecting the frequency as they did with the FM synth, here they affect the amplitude. The line~ object accepts messages containing various values. One value on its own causes a “jump” to that value, while two values next to each other without commas causes a “ramp.” For example, the shorter drum is created by sending the message [1, 0 50(. The amplitude of the wave produced by the noise~ object starts at a value of 1, but decreases gradually to 0 over the course of 50 milliseconds. This creates the quick, crisp sound of the shorter drum.

This drum kit consists of two variations on the noise~ signal: one with a long decay and one with a short decay. The resulting drum kit reminds me of the difference between an open and closed hi-hat; a closed hi-hat will not produce sound that lasts as long as an open hi-hat. Although the drum kit is fairly simple, it provides a nice variety to the layers of sound.

Simple Oscillator with Step Sequencer

The melody is produced by an osc~ object with no frequency or amplitude modulation. The most interesting thing about the oscillator is the step sequencer (which also triggers the drum kit). I wanted an easy way to repeat the same melody over a period of time, so I created a similar step sequencer to the one used with the frequency modulation synth. The metro~ object for the counter is set to the same speed of 250 milliseconds, but there are eight options to trigger rather than just four, so it takes twice as long for this sequencer to complete a loop.

GEM Window and Object

This was the most interesting part of the project to me, as I really enjoy working with GEM. I attempted to manipulate various arguments attached to a GEM object in order to manipulate color, alpha channels, volume, and location of the object in 3D space. I also attempted to recreate the effect from the live coding video that produced many cubes from one cube, which was much simpler than I imagined.

For each argument I attempted to have various sources from sound producing objects. For example, the amplitude of the frequency modulation synthesizer affects the volume, rotation in the x-axis, and translation in the y-axis, and the red color value for the cube object. The drum kit's amplitude controls translation in the x-axis and the z-axis, as well as the transparency of the cube. The amplitude of the melodic synthesizer controls the blue value of the synthesizer. Having the values come from various sources and affect different components of each attribute creates interesting variance.

The effect of having multiple cubes is produced by using mod division on the results of a counter, just like the counters producing the melody and powering the step sequencer. In this case, the counter is sending a value directly into the translation object, creating copies of the cube that are affected by the same object but translated a certain number of units away from the original cube, depending on the value set for the mod division. For example, if the mod division object contains [% 4], four cubes appear in the GEM window rather than just one. Because the cubes are all affected by the same arguments, they appear to be in a chain.

Conclusion

I was very pleased with the outcome of this project. I felt that it gave me the opportunity to expand on concepts I had explored before, as well as helped me clear up some confusion I had on certain Pd objects and use those objects effectively. I feel like I've reached a place in my Pure Data knowledge that I am able to execute most concepts well and am eager to use Pd as a tool in future projects of mine.

Bibliography

Barknecht, Frank. (2005, August 8). *Building Drums in PD.* Retrieved from http://impala.utopia.free.fr/pd/patchs/doc_tut_workshop/Fr_Workshop_giair/2.synthese/fotils_pddrums/pddrums.html.

Farnell, Andy. (2010). *Designing Sound.* Cambridge, MA: The MIT Press.

Kreidler, Johannes. (2013). *Loadbang: Programming Electronic Music in Pure Data.* Hofheim en Tanus, Germany: Wolke Verlag

Park Sung Min. (2014, November 7). Pd/GEM -live coding [Video File]. Retrieved from https://www.youtube.com/watch?v=kua7TZtroY4.