Sound Design Final: Live Sampler/Looper

Lysha Smith

Sound Design: Jim Mahoney
Submitted: May 1st, 2017

A brief paper explaining my final project.

For my final project of the semester I settled (after some false starts—more on this later) on creating a versatile sampler/looper that would be capable of being useful in both live performance settings as well as for creating interesting samples for studio/production work. There a good many moving parts to the patch and in general, despite it not being my initial thought/idea, I feel it displays much of the mastery with Pure Data that I have acquired over the course of the semester. The following is a brief explanation of the patch and the work that went into crafting it.

Before I begin to describe the patch itself, it is important that I give a bit of background information on my initial idea and efforts for a final project. For another class, Sound Art, I created a sound installation that was essentially a space for one viewer/listener to occupy that showered them with sound and light. The sound itself is mainly comprised of a slew of recorded samples of friends and family saying affirming, loving things as well as some drone like tones. Another feature of the sound is the chair that the listener sits in is wired into the sound as well however such that it vibrates the chair and the chair and person in the chair become the resonant body for the sound. The sound is triggered from an Ableton Live session and sent out over seven channels via a MOTU interface that places the various sounds at various locations as there are seven six speakers spread out in the space. In addition to the spacialization of the sound, was a highly visual component. I used a visualization software called VDMX which functions much like a video version of Ableton in that it triggers and manipulates video clips in real time. This feeds to a TV in front of the viewer/listener.

The final portion of the installation, and here is where I had hoped to implement some Pd patching, is a set of Phillips Hue lights which are capable of receiving OSC data over a wireless network. The idea was to synchronize the sound and lights to inform and shape a concise narrative for the piece. There were two main stumbling blocks that despite my many efforts, I could not overcome. The first and ultimately main issue was that I was unable to hack into the Hue API. There were/are various sources for how to go about this that I found online, but my lack of aptitude with some of the more in depth programming/hacking bits was ultimately debilitating. Additionally, I could not seem to find the right set of externals to get OSC messages communicating properly from Pure Data. I was able to create a patch that received OSC messages from my iPhone, but for some reason struggled to get a signal from Pd to the iPhone. Here again I feel that some of my inability to get this off the ground was a result of my not fully grasping certain key CS concepts, particularly how to make use of the terminal in meaningful way, as well as how to acquire and utilize installation packages from sources like GitHub, many of which rely on the terminal to compile and execute the files so that they are functional. After spending several days and countless hours on this, I decided to abandon it for lack of time and understanding. I surely will be spending a great deal of time over the course of the summer and next year trying to get a higher degree of aptitude with the terminal and generally with compiling files for installation so that I am able to implement structures that other users have created.

Now (finally) on to the sampler/looper patch. Loopers first came into being in the 1970s with artists like Brian Eno and Robert Fripp taking cues from early sound artists like John Cage and using physical tape recordings to playback and record small snippets of material recorded in real time. This patch seeks to emulate that sort of action. There are a few main components of the patch as well two sub-patches that facilitate it so that it functions properly. The recorded samples are stored temporarily in a table that is played repeatedly, based on a time/tempo signature that is able to be manipulated in real time, shortening or lengthening the sample clip. The tap tempo contains a sub patch that essentially reads the incoming bangs from the user and then dictates the length of the loop. There are two toggles in the patch, one that enables/looks for incoming audio signal, and another which enables the loop recording. If the audio is enabled but the loop recording is not, then nothing is recorded or played back. Once some audio is recorded the looper can be switched off so that the loop/clip plays back. Another feature is the ability to store a loop as a wav file if the user is liking what they are hearing.

Let's look at the tap tempo function a bit more in depth as this is a key component to the patch. This is the part of the patch that I struggled with a great deal and relied on the work of another programmer's patch[1] A bang immediately (via "inlet") goes into the "tempo" sub-patch, and this is where most of the work happens. The tempo of the tapping is read and interpolated so as to determine the length between the taps. Once the distance between taps is determined it is piped back to the main patch (via "outlet"). From there things are divided by four, as it is assumed that we are in 4/4 time. I suppose this could cause issues if we were in a different time signature. The other sub-patch is within the tap tempo function as well and this is another trick I needed help with. Pd essentially processes audio in blocks of 64 samples and so the table's length must be a multiple of 64, otherwise there would be very audible clicks and pops. The blocks sub-patch takes care of this problem. One of the great parts about the tap tempo function is that you can start things out with a really short loop length, record a couple of parts and then change the length to a longer one (or vice versa), this leads to very musical results.

Once the length of the loop is set, the audio is able to be dealt with. Things come in via the loop record function and an "adc~" is immediately processed with a low pass filter and there is also a line in level so as to be able to better control the signal that is being looped. I tried messing around with some limiters at first, but was having issues with the limiter not effecting the sound so much that it was difficult to hear the sample well, especially the higher frequencies. All the while the tap tempo function is changing the manner in which samples are being recorded as long as bangs are coming into it. Originally I was working with my laptop's microphone, but this of course was less than ideal and caused a good many issues. I eventually settled on using a standalone software synthesizer that I routed to the patch via soundflower. This software synth allows for multiple instruments in a bank so that I can create really interesting loops and patterns,

1 guitarextended. "Making a Looper with Pure Data." *Guitar Extended*, August 5, 2013. https://guitarextended.wordpress.com/2013/08/05/making-a-looper-with-pure-data/.

all the while avoiding any sort of feedback issues. I utilized a great deal of “s”-sends and “r” receives to move the signal around the patch without convoluting things too much visually.

The last bits were mostly aesthetic and functional additions. I added a VU meter as well as various volume and low pass filter sliders to the main outputs. I also added some MIDI control, though this is less than essential to the patch and likely I will not present this in class, but it was important both for my own cementing of the understanding of how to implement MIDI, which is key for any sort of live performance.

All things considered this was a bit of a frustrating venture as the ideas I had been trying so hard to implement just never really panned out. The sampler/looper ended up being a good way to sort of tie in some ideas that I had been toying with as well as to create a patch that I can use in a meaningful way in both live and studio settings. For lack of time, I had to rely a good deal on the “Guitar Extended” template for the patch, but personalized it for my own use. This emulation allowed me to not only craft a nice patch but also to better understand some objects and concepts I had been working on. Overall it has been a really enjoyable tutorial and I feel I have a real good sense of how to use the program in a way that I hope to move forward with.

Bibliography

Farnell, Andy. *Designing Sound | The MIT Press*. MIT Press. Accessed March 27, 2017.

"3.4 Sampling." Accessed May 1, 2017. http://pd-tutorial.com/english/ch03s04.html.

"3.9 Amplitude Corrections." Accessed May 2, 2017. http://pd-tutorial.com/english/ch03s09.html.

"Best Way to Build a Sampler." *PURE DATA Forum~*, October 10, 2014. http://forum.pdpatchrepo.info/topic/8700/best-way-to-build-a-sampler.

cheetomoskeeto. *PURE DATA: 19 Amplitude*, 2009. https://youtu.be/6O2PK4IaEPY.
———. *PURE DATA: 20 Smoothing Amplitude*, 2009. https://www.youtube.com/watch?v=I9db3hmA96U.

"Documentation — Pd Community Site." Accessed May 2, 2017. https://puredata.info/docs.

guitarextended. "Making a Looper with Pure Data." *Guitar Extended*, August 5, 2013. https://guitarextended.wordpress.com/2013/08/05/making-a-looper-with-pure-data/.