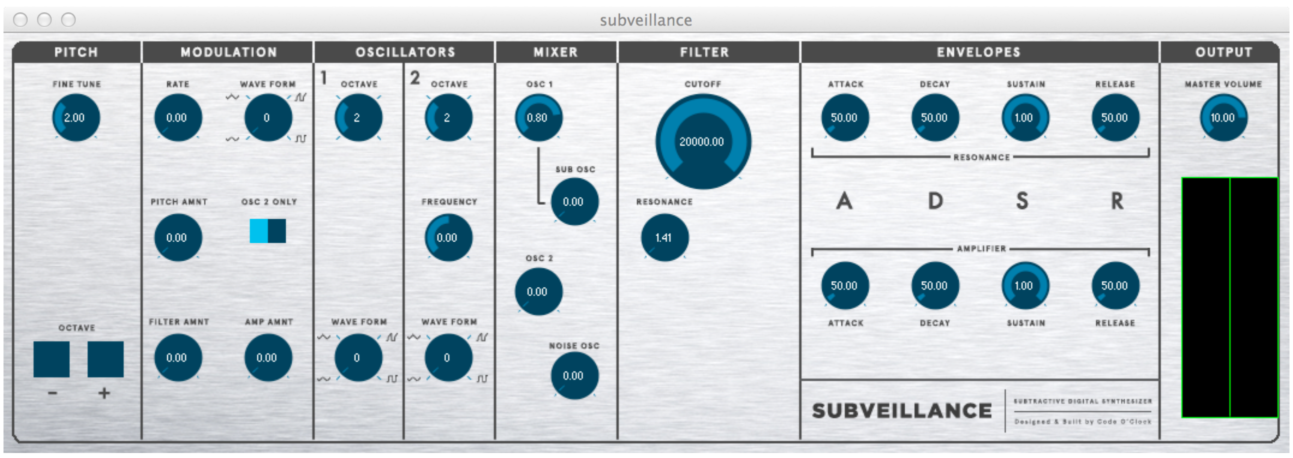
Subveillance

A Processing based subtractive synthesiser

Hugh Rawlinson

Mini Project for Introduction to Programming

Academic year 2012-2013



Introduction

For my Mini-Project, I decided to create a subtractive synthesiser in Processing, using digital signal processing rather than a pre-made audio library. I began by researching existing subtractive synthesisers. I decided to look into hardware synths rather than software synths, because the most widely acclaimed synths are hardware, rather than software. Originally I intended to model a specific synth, Moog Music Inc.'s recently released Sub Phatty because I thought it's entry level features would be easy to replicate digitally than a full scale large modular synth. However, in the design of the synth I found several features that would be extremely difficult to replicate in software particularly with no prior tuition in digital signal processing. In particular, the main two features were the iconic Moog ladder filter and analog distortion. After much research, I concluded that both would require very advanced modelling of analog signal path, and given that I do not have access to a Moog synthesiser to analyse these features, nor the electronics expertise to derive an algorithm to accurately model them, I decided to implement simpler, more generic versions of those effects in my software synth. I concluded that I could implement a subtractive synthesiser reminiscent of the Moog Sub Phatty with two oscillators with variable waveforms and relative pitches, a standard Butterworth filter, two envelopes controlling the amplitude of the output and the cutoff frequency of the filter, and a modulator with the ability to modulate pitch, amplitude, and filter cutoff frequency. Once I had decided upon the basic requirements for my synth, I began designing and implementing the component classes.

Design

Software

I began researching the correct method to access the sound-card in Processing/Java, and found Dr Matthew Yee-King's audioThread Java class which I began the project with. It accepts float arrays, or buffers, of a pre-defined length which it then passes to the sound card to be played. This worked very well for what I intended to do.

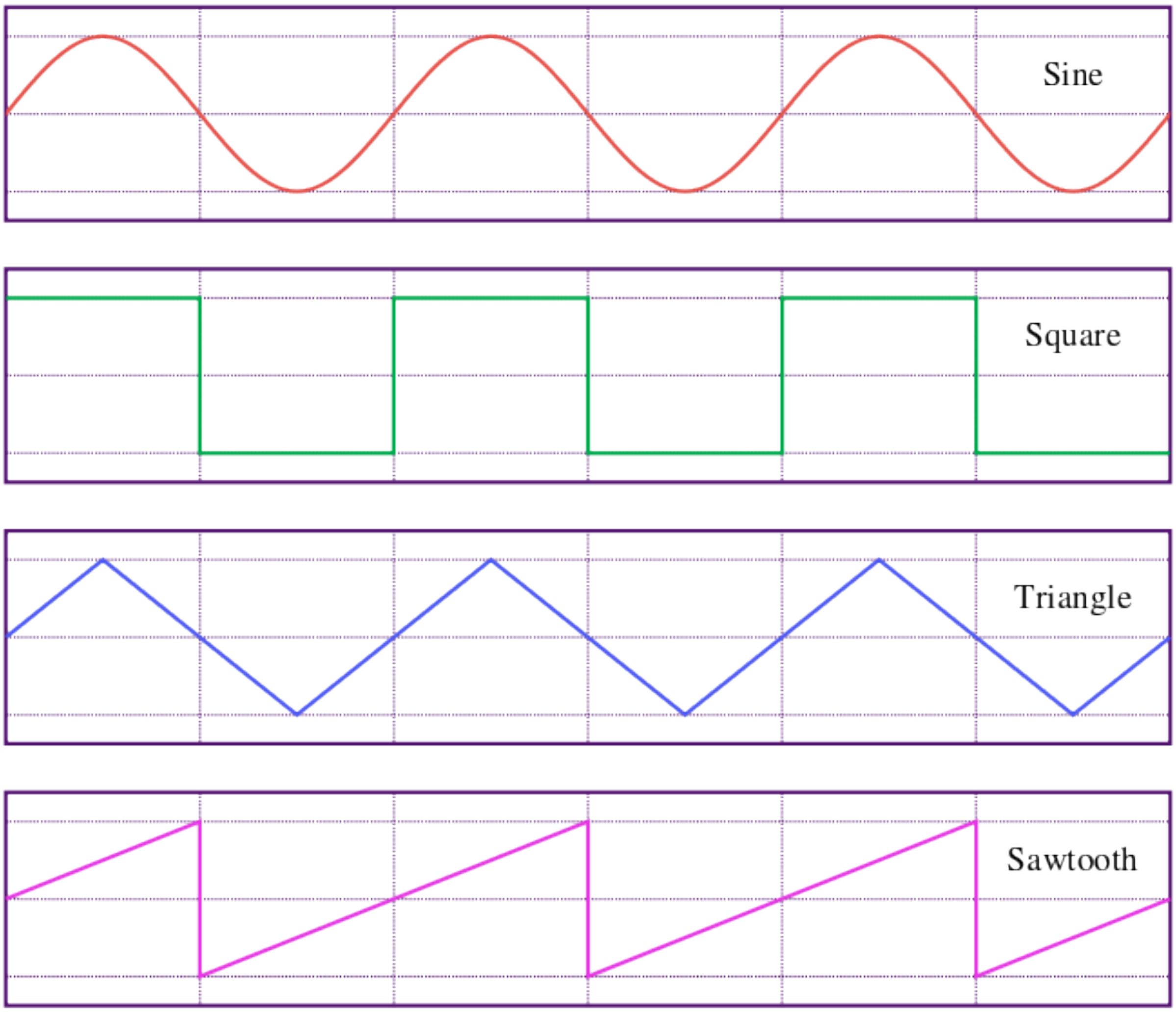
MIDI is a standard protocol for sending note data between audio devices. It has been around since it was standardised by major synthesiser manufacturers in the 1980s, and is still the most widely used protocol for that application today. Therefore, I decided to use MIDI as a a control protocol for my synth. This has the additional benefit of allowing musicians to use their own preferred human interface device - be it a keyboard, a MIDI enabled string instrument, or something entirely different. I use The Midi Bus by Severin Smith, a Processing library to enable access to MIDI hardware that is connected to the host computer, which works across multiple platforms.

I also decided to use Andreas Schlegel's Control P5 Library to display a functional interface. Originally I was using it as a temporary stop-gap, to be able to manipulate variables during runtime. I had intended to implement my own class of controllers, however as my work diverged from the physical version of the Sub Phatty I became more and more inclined to keep the Control P5 scheme. I finally decided not to implement my own control classes - I had intended to model the original knobs on the Sub Phatty - when I realised that a software synth is quite removed from a hardware synth in that a physical interface was not required, and that skeuomorphic design in fact hindered the layout of a software synth. There were undoubtedly better ways to display parameters in a digital medium than simply copying the hardware version. I decided that Control P5 was a good choice, because it displays the numerical values of its parameters on screen.

I began to design the class structure of the software once I had decided upon the features I wanted to implement. A filter class would essentially need to be an object with dynamic parameters, and a function to apply its effect to incoming audio values. It was at this stage of my research that I realised the difficult to model nature of Moog's critically acclaimed filters. I could not find an algorithm I could implement that accurately modelled them, so I decided to use a simpler filter algorithm which would achieve a similar effect, while keeping a low processing footprint. Given the extraordinarily high rate at which samples must be generated and passed to the sound card in time to prevent a buffer underrun error in which the sound card is not being fed samples fast enough to keep from running out of them while playing, I was mindful of the total time it would take for each sample to be generated. Java is not a language widely used for audio processing specifically because these operations are not generally as efficient as that of a more low level language like C ++ , so I decided a simpler filter would suffice rather than risk buffer underruns with a more complex algorithm. During my research I found a great site, www.musicdsp.org, which contains hundreds of algorithms that are used in DSP. I used the lowpass filter algorithm that can be found at http://www.musicdsp.org/archive.php?classid=3#38. I chose this algorithm for its comparative simplicity for ease of implementation.

My envelope class is quite straightforward. When a midi key is pressed, it triggers an envelope generator to begin, ramping up from 0.0 to 1.0 in the time specified in the parameter on the interface, and then from 1.0 to the sustain level. When a key is released, it triggers a released function, and ramps the value back down to 0.0 in the time specified by the user.

The design of my Oscillator classes is quite complex. In a subtractive synthesiser, it is necessary to have access to a selection of fundamental waveforms that have become standardised over the years in the design of both hardware and software subtractive synthesisers. These waveforms are the sine wave, triangle wave, square wave, and sawtooth wave, as shown in the image below.



Source: http://en.wikipedia.org/wiki/File:Waveforms.svg

Each of these waveforms share some common properties, including phase, frequency, and amplitude. I therefore wrote a base class for each of the oscillators. As I would use them in both the oscillator class - a master class for an oscillator module of the synth - and in the modulation class, I kept them out of just one class to maximise the reusability of the code. The main oscillator class instantiates 4 oscillator objects each with a different waveform, and integrates them into one complex class for use directly in the audio generation code. In the getValue() function, which returns the current value of the selected oscillator, you will notice that I call the getValue() method of each of the component oscillator objects. This is to ensure that the phase remains the same in each of the oscillator classes to prevent artefacts in the audio.

I also wrote a separate noise oscillator class. As it is not used in a tuned oscillator, and not part of the oscillator module on the layout of the GUI, there was no need to integrate it into the main oscillator class. The signal generation simply returns a random float between -1 and 1.

As previously mentioned, the modulation class instantiates each of the 4 oscillators as a modulation source. As in the oscillator class, it calculates each waveform so that they remain in phase, so that when modulation source is changed there is cohesion between the previous and the new waveform patterns.

The remaining files serve as the main sketch, and the Control P5 config file - I found that there was far too much code in setting up Control P5 it would be best to relegate it to another file in order to preserve readability of the main sketch file.

Layout

The visual layout of Subveillance was originally intended to mimic that of the Moog Sub Phatty, but once I had made the background image with the help of a graphic designer friend of mine, I decided to take the actual control of the synth down a less skeuomorphic route, and use Control P5 for the control elements. Despite the illegibility of Control P5's documentation, I found that wading through automatically generated JavaDocs to figure out a way of doing what I wanted to do (most notably, removing the caption labels from the elements) was quite a rewarding task, and undoubtedly a skill that will serve me well as I continue to develop both in Java and in other languages.

Usage

I intended to make the synthesiser as easy to use as possible to those who know how synthesisers work, however as it is intended for use by people who already have an understanding of synthesisers, I did not include any design elements that indicate how it works on a deep level, and therefore Subveillance is unlikely to be suitable for synth beginners.

Working our way left to right section by section, the fine tune knob sets the main tune of the synthesiser. In early hardware synths that relied on non-linear electronic components whose responsiveness changed as valves warmed up, the tune knob is for making sure the synthesiser is in tune with the rest of a musical ensemble. The octave selectors at the bottom of the pitch section are used to select the octave of the synthesiser. While most hardware and indeed software MIDI controllers have this function build in, I thought it would be a nice feature to have. Moving on to the modulation section, where the rate knob controls the frequency of the low frequency oscillator that drives parameter modulation. "Wave Form" allows the user to choose from a selection of waveforms the waveform of the low frequency oscillator. The "Amount" buttons control the level at which the modulation section affects other parameters of the synth - namely pitch, filter cutoff, and amplitude of the final waveform. "Osc 2 Only" toggles whether or not the pitch amount controls the pitch of both oscillators, or only the second oscillator. The oscillator section is divided into two parts, one for oscillator one and the other for oscillator 2. The octave and waveform knobs of each oscillator set the respective parameter for the corresponding oscillator. In the mixer section, the user can set the volume of each oscillator before it is passed in to the filter. Connected to oscillator one is a sub-oscillator that uses the same waveform as oscillator one, but is always one octave below, which is useful for creating bass sounds. At the bottom is a knob controlling the noise oscillator volume. As this is independent of every other oscillator, and requires no pitch parameter, this is the only knob that affects the noise oscillator.

The Filter section allows the user to set the cutoff frequency and the resonance of the filter. The default cutoff frequency was chosen deliberately - as human hearing ranges from about 20hz to 20khz, when the filter is fully open, it should be impossible for humans to tell the difference as to whether or not the filter is bypassed. The envelopes section handles envelope generators for the filter and for the amplitude of the sound, allowing the musician to control those parameters over time in relation to a keypress. The output section houses a master volume knob (which goes up to 12, or 120%), and an oscilloscope I used for testing but decided to keep in the finished product as I found it was an interesting feature.

While I understand that the average user does not have a MIDI control device, the target audience of this program is clearly synthesiser enthusiasts who are considerably more likely to have access to one. However, because I used MIDI as a standard for input, it is possible to use an external program to use a standard PC keyboard as a MIDI input device. I would recommend MidiKeys on the Mac OS X platform.

Limitations

The main limitation I encountered during development was attempting to reduce the buffer size of the array to be passed to the sound card. This would have achieved a more responsive feel to playing the synthesiser with a MIDI keyboard, as it would take less time for pitch changes to come into effect. As a result, compared to flagship software synthesisers by companies such as Propellerhead and u-he, Subveillance feels quite sluggish to play. I attempted to counter this by optimising the various dsp algorithms in an attempt to lower the buffer size without running into buffer underrun errors. Unfortunately, my efforts did not work, and I am left with a minimum possible buffer size of approximately 1500 samples. Subveillance seems to take quite a lot of processing power, as do many Processing applications. I began stripping down the application to see if I could minimise its CPU usage, however removing my DSP objects did very little. However, when I removed all the GUI elements, particularly the background picture, CPU usage dropped by an astonishing 90%. This clearly indicates that the main processing cost of the program is in fact the visual elements rather than the audio processing elements, which surprised me. Unfortunately, even without the visual elements of the program, the buffer size would not go below 1500 samples without running into buffer underrun errors. With the help of a friend of London Hackspace where I did much of the coding of this project, we ascertained from various Stack Overflow answers on the topic, that Java's implementation of sound on OS X is particularly inefficient when compared to other operating systems, in particular Linux. Again, I found this surprising. All my prior experience told me that OS X is the best operating system on which to use audio.

Another limitation I ran into was my knowledge of DSP algorithms. As these do not get taught to Music Computing students until 2nd year, I had to independently research these concepts. Luckily, I found many great resources both online and in books (in particular, The Computer Music Tutorial by Curtis Roads), and thanks to these resources, going beyond what is expected of Music Computing students at first year level was not as difficult as I had anticipated it would be.

Future Plans

I intend to rewrite this synthesiser in C++, using the WDL-OL library so that I can export it as a VST and use it in a digital audio workstation so that I can incorporate it into a workflow suited to standard musical composition. I hope that in doing so I will be able to dramatically increase the performance and reduce the buffer size for a more responsive feel to the synthesiser. Over the summer I also intend to research and implement more features including an overdrive effect, and possibly into other synthesis techniques such as frequency modulation synthesis, and wavetable synthesis which was popular in the early days of computer music synthesis when considerably less processing power and RAM were available.

I also intend on releasing Subveillance as an open source project, both in Processing and in C++ in the hopes that I can continue developing and optimising both versions with other developers who are interested in digital audio synthesis. I have gained considerably from open source projects, including The Midi Bus, and Control P5 in this project alone.

References

ACE Synthesiser. (n.d.). *u-he*. Retrieved March 22, 2013, from http://www.u-he.com/cms/ACE

Doty, M. (n.d.). The Moog Sub Phatty Part 1- Oscillators - YouTube. *YouTube*. Retrieved March 1, 2013, from http://www.youtube.com/watch?v=nLKiGwdMgig

Doty, M. (n.d.). The Moog Sub Phatty Part 2- The Filter - YouTube. *YouTube*. Retrieved March 1, 2013, from http://www.youtube.com/watch?v=Y5_Ox-3GPG8

Doty, M. (n.d.). The Moog Sub Phatty Part 3- Modulation Part 1 - YouTube. *YouTube*. Retrieved March 1, 2013, from http://www.youtube.com/watch?v=e42_ou_OMkg

Doty, M. (n.d.). The Moog Sub Phatty Part 5- Envelope and Presets - YouTube. *YouTube*. Retrieved March 1, 2013, from http://www.youtube.com/watch?v=vOPoZjntOlY

Doty, M. (n.d.). The Moog Sub Phatty Part 4- Modulation Part 2 - YouTube. *YouTube*. Retrieved March 1, 2013, from http://www.youtube.com/watch?v=1wIhqY2NJyU

Language Reference (API) Processing 2+. (n.d.). *Processing.org*. Retrieved January 31, 2013, from http://processing.org/reference/

Roads, C. (1996). *The computer music tutorial*. Cambridge, Mass.: MIT Press.

Schlegel, A. (n.d.). Javadocs: controlP5. *Andreas Schlegel, sojamo. Index*. Retrieved January 18, 2013, from http://www.sojamo.de/libraries/controlP5/reference/

Smith, S. (n.d.). The MidiBus. *Small But Digital*. Retrieved January 23, 2013, from http://www.smallbutdigital.com/themidibus.php

SUB PHATTY | Moog Music Inc. (n.d.). *Moog Music Inc*. Retrieved February 1, 2013, from http://www.moogmusic.com/products/phattys/sub-phatty

Sub Phatty Photograph. (n.d.). *GearNuts.com*. Retrieved February 1, 2013, from www.gearnuts.com/images/items/1800/SubPhatty-xlarge.jpg

Tarrabia, P. (n.d.). LP and HP Filter. *http://music-dsp.org/*. Retrieved February 17, 2013, from www.musicdsp.org/archive.php?classid=3#38

user Omegatron. (n.d.). File:Waveforms.svg - Wikipedia, the free encyclopedia. *Wikipedia, the free encyclopedia*. Retrieved March 22, 2013, from http://en.wikipedia.org/wiki/File:Waveforms.svg

Yee-King, M. (n.d.). Simple Audio in Processing Repository. *GitHub*. Retrieved January 16, 2013, from https://github.com/yeeking/SimpleAudioInProcessing

Yee-King, M. (n.d.). AV Lab 3 Lecture Notes. *yeeking.net*. Retrieved January 23, 2013, from yeeking.net/cc/AV_Lab_3.pdf