Audio Programming: Submission 3

I created a bass synthesizer, modeled after Native Instruments' Reaktor ensemble, the TRK-01 Bass synth's West-style oscillator. My synth consists of a main oscillator, a sub oscillator, a modifier section, and a filter section. The main oscillator morphs between three wave shapes: sine, spike, and sawtooth. The spike wave is created by high-passing a square wave. This oscillator is further wave-shaped via foldback distortion. The sub morphs between sine, square, and sawtooth, and has an octave selector, from unison to two octaves below the main.

The modifier section consists of a ring modulator, frequency shifter, and sample-and-hold distortion. These run in series, each with its own dry/wet mix. The modifiers all track the current note frequency, so pitch parameter adjustments are relative to the note being played by the main oscillators. The sub oscillators bypass the modifiers.

Finally, the filter section has a selectable filter type. There is some confusion about what JUCE's lowpass IIRFilter is, a Butterworth or Biquad. A biquad would roll off –12dB per octave. But, looking at the definition, it only uses the previous input and output, which leads me to believe it is a Butterworth, making it –6dB per octave. I cascaded two instances to create a –12db roll-off low-pass. I then made child classes that cascaded the parent class to create –24dB and –48dB lowpass filters. There is also a notch filter available.

The filter has an envelope that can control the cutoff frequency and resonance. There is also an LFO built using the sub oscillator class that can control the cutoff frequency.

I chose this particular synth because it is one of my go-to synths when producing electronic music. I wanted to recreate it, with some variations from the original. Building it from scratch would help me better understand it, while also allowing me to change some elements I wish were different. For example, the Native Instruments version can only use one modifier selection. My version can use all three simultaneously.

Reaktor's appears to use real-time oscillators. I chose to use wavetables because I wanted experience using them and they are much lighter on resources than live oscillator functions. Most of the synth plugins I use are wavetable based and understanding them is vital.

Ideally, this synth could be used for any form of electronic dance music currently being produced. Of course, it has the flexibility to create sounds used for a vast array of music styles. It does create the type of aggressive bass sounds used in house, techno, and breakbeat styles. The main oscillator blends to create some straightforward wave shapes that are conducive to low frequency reproduction in speakers.

The foldback distortion and three modifier oscillators add a lot of aggressive harmonics that are common in today's electronic music. These can be either tamed or exaggerated using the filters and resonance. This is especially true when using an envelope or Low Frequency Oscillator (LFO) on the notch filter with a high resonance. It can create sounds similar to Frequency Modulation Synthesis running through formant filters.

The wavetables are all built with sine waves. I found this to be an efficient way to generate band limited sawtooth and square waves additively. You can create dozens of instances of sine waves at the appropriate harmonics, and then print them to the wavetable. After that it is just a matter of reading the values in the table.

Since wavetables are more efficient, it opens us up to creating more synth sections that utilize oscillators derived from the wavetables, such as the modifiers or LFOs.

Another interesting bit of code is the morphing control on the main and sub oscillators. The slider at zero gives a pure sine wave. To the far right, at 2, the slider gives a pure sawtooth wave. When centered at 1, it gives either a spike or square wave, depending on the oscillator. The float values between integers blend the adjacent waveshapes. I only use it to morph between three waves, but it can be scaled to any number of waveshapes or blendable values.

I was able to follow the data flow inside the Reaktor Core modules used to build the TRK-01 synth to derive the algorithm to achieve this. Each wave is assigned an integer control value from zero to n – 1, where n is the number of waves. The morph parameter slider also goes from zero to n – 1. Subtract the control value from the morph value, and take the difference's absolute value. Subtract the result from one. Then clip any negative values at zero. Then, every integer returns a pure waveshape, with blends in between, for as many integers as you have.

One thing that was oddly satisfying was using the different header files almost as components in a modular synth. It helped organize and compartmentalize the tasks needed to create the synth. Each header encompassed a module with different classes that performed tasks for that module. This also allowed for logical class inheritance because the classes within each synth module require similar things to operate.

When the synth was essentially functioning, I began adding value smoothing to the parameters that clicked. This led to a fair amount of type issues between atomic float pointers and floats, and the arguments and methods that used them. In order to smooth the values, I had to dereference the atomic pointers. I then had to find all the class methods' arguments that required the atomic pointers and revamp the mechanics and variables within the classes to use the smoothed float.

In the future I plan to add the ability to adjust the voice count with a parameter instead of in the code. I want more usable parameter control using either logarithmic or exponential values. The filter cutoff LFO frequency clamping is rough and needs improving. I would love to have the LFO track the BPM. Pitchbend wheel functionality is needed as well. I also would like to have the modifiers run in either series or parallel.