COLLEGE OF ENGINEERING



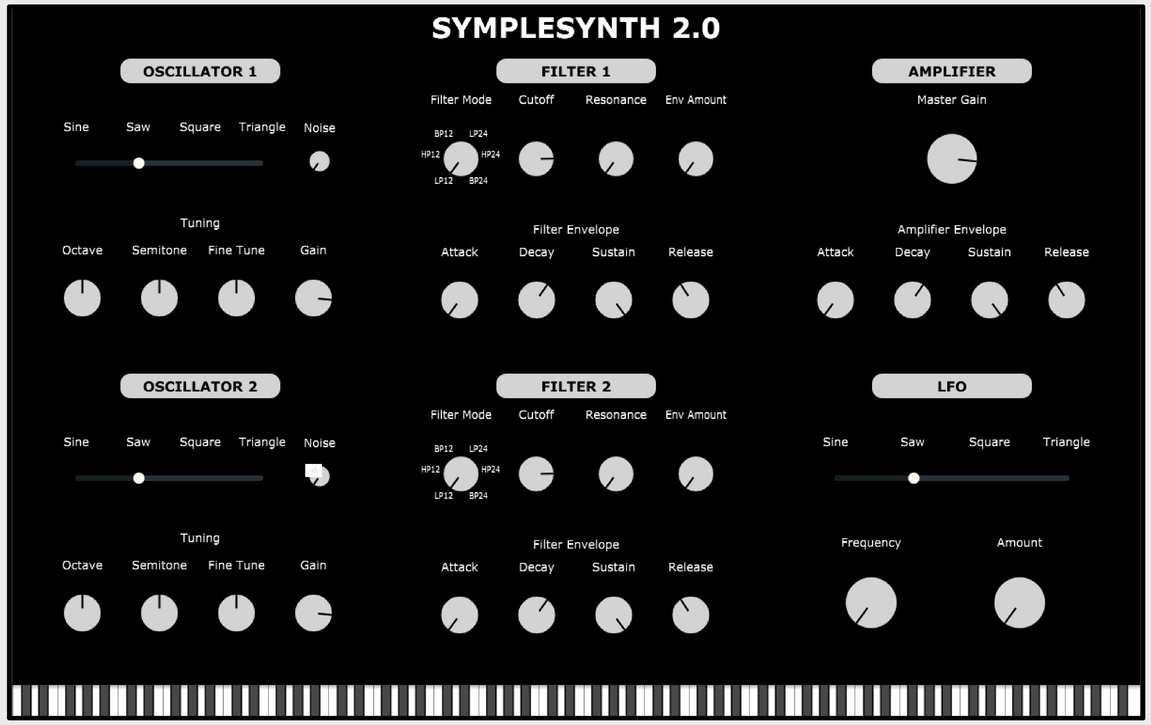
## THE TEAM

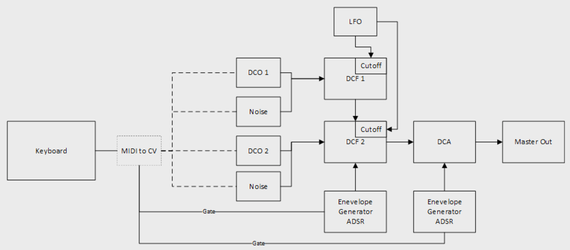

- NATHAN JOHNSON *johnsna7@oregonstate.edu*
  Currently a graphic designer in the Chicago suburbs, Nathan is graduating from the Post-Bacc program in December with a degree in Computer Science to transition to a career in software development. His previous degree was in Music Performance with a minor in Music Technology. Prior to working as a graphic designer, he worked as an audio engineer in a small recording studio, so coding a synth drew his interest as a natural synthesis of his past experiences.
- GLENN OBERLANDER *oberlang@oregonstate.edu*
  Glenn got his first degree in Kinesiology, and after a stint in that career, is currently in the first steps into his tech career. He hopes to use his new degree to excel past the moon. He chose to build a synth because he's been producing music off and on for the past 8 years and was really interested to see how a software synth works.
- JESSE WOZNIAK *wozniakj@oregonstate.edu*
  Jesse Wozniak is a full-stack web developer based in Detroit, MI. Previously, Jesse studied Music with a Jazz Guitar concentration at Wayne State University. He's always interested in finding new ways to combine music and technology, so synthesizer programming was a perfect fit.

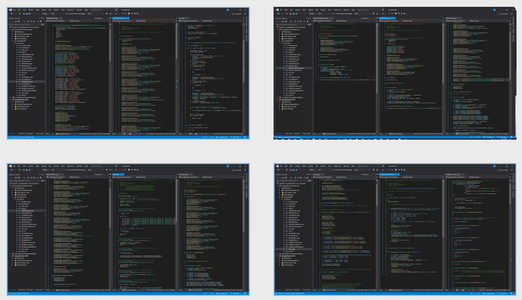

Oregon State University

# SYMPLESYNTH

Digital Synthesizer built using C++ & the JUCE library



## DESCRIPTION

SympleSynth is a digital synthesizer, an instrument which accepts MIDI signal input and produces audio signal output inside the application.

Our synthesizer will allow the user to adjust and modulate 2 different signal oscillators and a noise signal for output. In addition to adjusting the output signal attack, decay, sustain, and release, the user will also be able to adjust the frequency content of the output with 2 independent filters.

## FEATURES

- Polyphonic Synth with 2 Independent Oscillators
- Available as a standalone application for Mac or PC
- Full MIDI support
- Independent filters, one for each oscillator
- Global LFO and Amplifier Envelope

## UNDER THE HOOD

- SympleSynth works similarly to a website. There is a front end and back end. The front end graphical user interface (juce::AudioProcessorEditor, aka PluginEditor) interacts with the back end (juce::AudioProcessor, aka PluginProcessor). In this case, instead of the back end being a database, it is an "audio thread", that the front end (graphical interface: dials, sliders, etc) gets and sets values via a reference to the thread. If any other graphical components (knobs, sliders, etc) want to get and set from the audio thread, they are given a reference to it.
- PluginProcessor has a function, processBlock, which is where the main audio processing occurs. This is where we process the Oscillator, Filters, and LFO, then output sound to the speakers. This function can get called slower than 86 times per second or faster than 689 times per second, depending on your buffer size. You can imagine the drastic difference in processing load for different buffer sizes. Typically the smaller the buffer size the better, but the human ear cannot detect latency less than 10-12ms, and such low buffer sizes can cause very terrible clicks and pops in your audio if your hardware can't keep up. So, a safe bet is to start with 512 samples per buffer and adjust accordingly.
- Since processBlock takes care of all the audio processing, these tasks need to be delegated to other classes and functions. One such class is juce::Synthesiser, which controls the juce::SynthesiserVoice (Voice). Imagine a choir of 10 people, each person has a voice, and each person can only sing one note at a time, but all 10 people (voices) could play at the same time. But, if a new note needs to be voiced, someone needs to stop their current note to play the new one… The Synthesiser class works in the same way. In our case, our Synthesiser class manages 64 voices. (For perspective, an analog synth typically doesn't manage more than 16 voices).
- Each Voice in SympleSynth has its own copy of an Oscillator and Filter (If the Voices shared 1 or 2 Oscillators or Filters, the sound would be quite a bit different). Each Oscillator is generated and each Filter filters in juce::SynthesiserVoice::renderNextBlock which is fairly similar to processBlock. renderNextBlock takes the X number of samples (audio buffer), and renders each sample based on the Oscillator and Filter settings. For example, if you are using the Saw wave for Oscillator 1, then the samples will be "drawn" to be a Saw wave. Then, the buffer, which is now very Saw-like, gets passed to the Filter to get its frequency spectrum shaped accordingly.
- SympleSynth has 2 Oscillators and 2 Filters. Osc1 feeds to Filter1, Osc2 feeds to Filter2, then Filter1 and 2 feed to Master Volume (back in juce::AudioProcessor::processBlock).
- The LFO affects the cutoff of Filter 1 and Filter 2. The LFO works like this: From the cutoff frequency, open up to X number of semitones (LFO Amount), then close back to the cutoff frequency. The rate that this happens is determined by Frequency. The wave type will affect how it will open.
- Once the renderNextBlock has finished rendering each sample in the buffer, processBlock will output this buffer to each channel. Then the next buffer goes through the same process.