Virtual Synthesizer Development in C++

Will Sieber

Department of Computer Science, The College of Wooster

May 27th, 2024

Contents

1 Abstract

Virtual synthesizers are tools used by musicians to generate sound for use in digital audio workstations. They are repackaged dynamic-link libraries under one of several existing specifications that ensure they are compatible with host applications. Several software development kits and libraries exist with the purpose of developing these tools, but this paper will focus on development under Steinberg's Virtual Studio Technology (VST3) specification. This will serve as a quick-start guide to those interested in real time audio processing by first providing the theory behind computer audio, and then using this knowledge in a practical setting by developing a virtual additive synthesizer under the VST3 specification. This synthesizer will be able to generate several different waveforms with two different oscillators, alongside an envelope generator to control the articulation of the resulting waveform. Additional functionality common to other synthesizers will be included and discussed, but are not required to develop a functional program.

2 Introduction

2.1 Project Goals

This project initially had one goal in mind: create sound. In order to do so, one must first process MIDI data, generate the desired waveform, and then create the user interface while ensuring the program's state changes accordingly. Upon overcoming this hurdle, several additional features could be implemented on this foundation that include additional oscillators, envelope generators, and more. Maintaining good practice while coding was an additional goal, so that others may take this project and change or modify it to their needs. That being said, prior to beginning development, it is important to understand how each element of a synthesizer works and how each fits together.

3 Synthesizer Basics

3.1 What is a Synthesizer?

In essence, a synthesizer is a musical instrument that generates waveforms and provides several parameters to alter them in interesting ways. These parameters can interact with one another to create complex timbres, such as by introducing additional frequencies to a sound. Development of the modern synthesizer can be traced to the late 1950s, where the Columbia-Princeton Electronic Music Center became the prime location for composers interested in new and developing technologies. [1]

3.2 Components

There are several components of a synthesizer that work together in order to generate sound. This section will cover in detail the specific components that were implemented in software. Following this section, there will be another that provides further components that are commonly found in other synthesizers.

3.2.1 Oscillators

Oscillators can be generally broken into two categories: tone generators and controllers. Tone generators repeat periodic waves in order to generate sound. Simple tone generators can be expressed as mathematical expressions, such as $f(x) = sin(x)$, while complex tone generators allow for any waveform to be repeated, regardless if they can be easily expressed as a single expression. Complex oscillators generally allow for a number of different waveforms to be represented in a single table, allowing for a user to select any number of waveforms in real time. For the purposes of this project, simple oscillators are used as tone generators. Controller oscillators, on the other hand, are much lower frequency than tone oscillators, and are used to control other parameters of a synthesizer. These will be discussed in brief at a later section.

There are four simple waveforms that are considered ”fundamental” for a synthesizer. These are the oscillator's *sine*, *square*, *sawtooth*, and *triangle* waveforms. Each have their own characteristics distinguishing them from one another, but can be loosely described in the following ways:

1. Sine waves are considered ”pure”, as they contain no overtones. Should every instrument be isolated to its fundamental frequency, this is the tone that would result.
2. Square waves are ”buzzy” and [].
3. Sawtooth waves are [].
4. Triangle waves are [].

3.2.2 Envelope Generators

In order to provide articulation, envelope generators describe how the amplitude of a particular sound should change over time. Several parameters are used to accomplish this goal, and those are the synthesizer's *attack*, *decay*, *sustain*, and *release*. *Attack* describes how much time it takes for the synthesizer to reach full amplitude upon receiving the signal to generate sound. Note that this is not the s A short attack will reach full amplitude quickly, such as playing on a piano or organ. Longer attacks will take more time to reach full amplitude, which create a ”swelling” effect that is similar to that of a violin increasing in volume. *Decay* describes how much time the synthesizer should take to reach a particular amplitude once the attack is finished. A longer decay will prolong a sound prior to the sustain, and vice versa. *Sustain*, unlike attack and decay, describes the amplitude itself to which the decay will reach upon completion. When reached, the synthesizer will continuously play at this amplitude until a signal is sent that it should stop generating sound. Upon this signal, *release* describes how long the synthesizer should take to stop generating sound by gradually decreasing the amplitude to zero.

In more technical terms, the envelope generator of our synthesizer is its own finite state machine that modifies our signal. The signal is modified by multiplying our source to a constantly changing value. This value has different behavior as to how it changes depending on if it is in the attack state, the decay state, and so on.

One may notice a problem with this model - the rate at which the attack, decay, and release change amplitude itself is not described. Powerful synthesizers allow for this change to be represented as a polynomial function, but for the purposes of this project, the rate at which each parameter changes will simply be linear.

With these two components in mind, one can create a simple synthesizer in the same manner that is described in this project. What follows are additional components found commonly in synthesizers, but are not required to be present. That being said, it is important to understand them to provide a complete picture of what synthesizers are capable of.

3.3 Extra Components

Digitally Controlled Amplifiers

Digitally Controlled Amplifiers (DCAs) control the final amplitude or volume of the synthesizer. A good DCA will prevent the final output signal from going beyond the range [-1, 1] thus preventing unwanted artifacts. They can handle global panning as well, allowing the user to choose where the resulting sound should play from a pair of speakers. [2] For the purposes of this project, volume is controlled from each source oscillator.

Low-Frequency Oscillators

Low-Frequency Oscillators (LFOs) are generally used as an input to automate other parameters of a synthesizer, allowing for a greater range of timbres. They generally exist in a range much lower than human hearing (<20 Hz) and usually consist ofs the same basic waveforms that tone generator oscillators use (sine, square, sawtooth, and triangle).

Filters

Filters are a form of subtractive synthesis that remove frequencies based on a particular *cutoff*. There are several types of filters that exist, with the most common being *lowpass*, *highpass*, and *bandpass*. Lowpass filters remove higher frequencies, allowing lower frequencies to "pass through" the filter, resulting in a "dark" sound. Highpass filters do the opposite, removing lower frequencies and passing through higher ones based on a particular cutoff frequency. Bandpass filters remove both upper and lower frequencies from a particular cutoff.

3.4 Additive Synthesis

Strictly speaking, this synthesizer will not be a fully functional additive synthesizer. By definition, an additive synthesizer works by adding sine waves of proper frequencies and amplitudes in order to generate the resulting wave one desires. [2] Doing so this way is considered "cleaner" and removes aliasing from the resulting wave.

3.5 Extra Synthesis Techniques

Subtractive Synthesis

Subtractive synthesis is done by taking a wave and removing frequencies from the waveform. Filters are an example of subtractive synthesis, which have been covered in a previous section.

Wavetable Synthesis

Wavetable synthesis is a type of synthesis that allows a musicians to manipulate the waveform being used in real time. Instead of there being one of several simple waveforms to choose from, the user can select an entire array (or table) of waveforms that can be morphed from one to the next. This is known as *digital interpolation*.

FM Synthesis

Frequency modulation (FM synthesis) is a type of waveform manipulation that uses one waveform to manipulate the frequency of another. An example of this can be see below:

4 Implementation

4.1 Theory

There are a number of problems one faces when attempting to process real time audio. They can be summed up by asking the following questions:

1. "How do we create a digital representation of sound?"

2. "How do we process audio fast enough to ensure there are no unwanted artifacts in our software?"

The following section will address these concerns and provide the theory of what is occurring in the background of our software.

4.1.1 Discrete Signals

Sound is represented digitally by taking samples of the original source. An audio sample measures the amplitude of the source wave at a point in time, which is represented in code as a floating point number in the range $[-1, 1]$. Given a piece of audio, the number of samples per second represent the *sample rate* of our recording. As a result, a higher sample rate will result in audio that more closely resembles the original source. Consider the following sine wave:

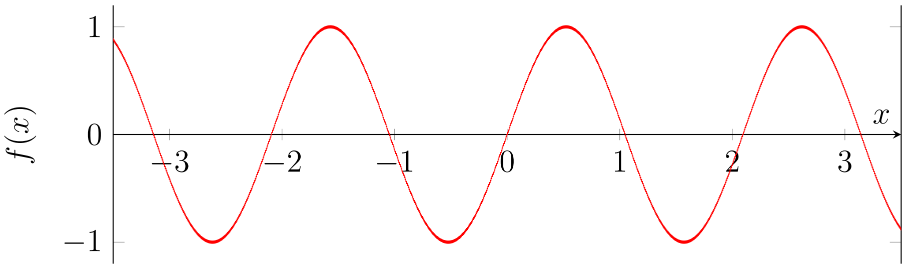

Figure 1: A graph of the sine wave $f(x) = sin(3x)$.

A computer would interpret this signal by taking a number of samples at regular intervals:

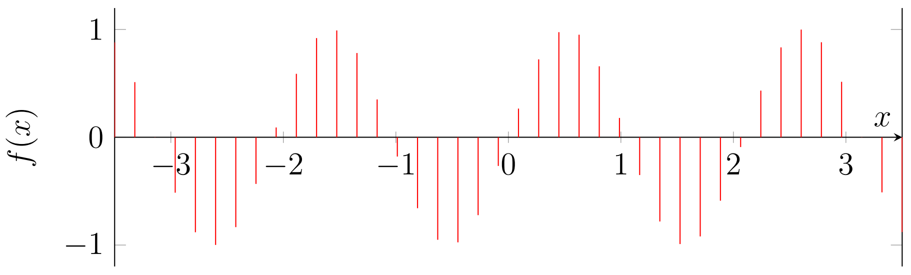

Figure 2: A discrete graph of the sine wave $f(x) = sin(3x)$ taken with 40 samples.

Thus, the higher number of samples per second would result in a sound that more closely resembles Figure 1:

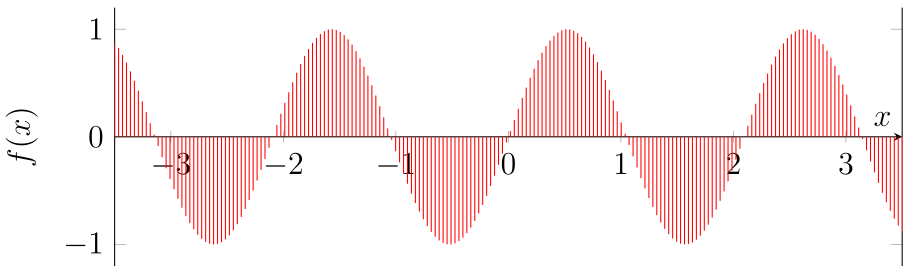

Figure 3: A discrete graph of the sine wave $f(x) = sin(3x)$ taken with 200 samples.

Most commonly, audio files will have a sample rate of 44.1 kHz or 48 kHz. That means our software will need to process that number of samples per second, continuously. For the purposes of this project, there are two audio sources: *OSC1* and *OSC2*. Each oscillator will generate a sound sampled at the rate that is decided by the host application.

4.1.2 Buffers

When processing, there are a stack of processes that one's audio information must traverse before it is played on the system's speakers. Section 4.1.3 describes this stack. At regular intervals, a system's integrated sound card requests audio data to be sent to the system's speakers. This request does not wait for anything. As such, it would be incredibly inefficient for this call to be done for every single audio sample, so to process this large amount of data quickly, *buffers* are used. A buffer is a length of audio samples, such as [].

4.1.3 Flow of Information

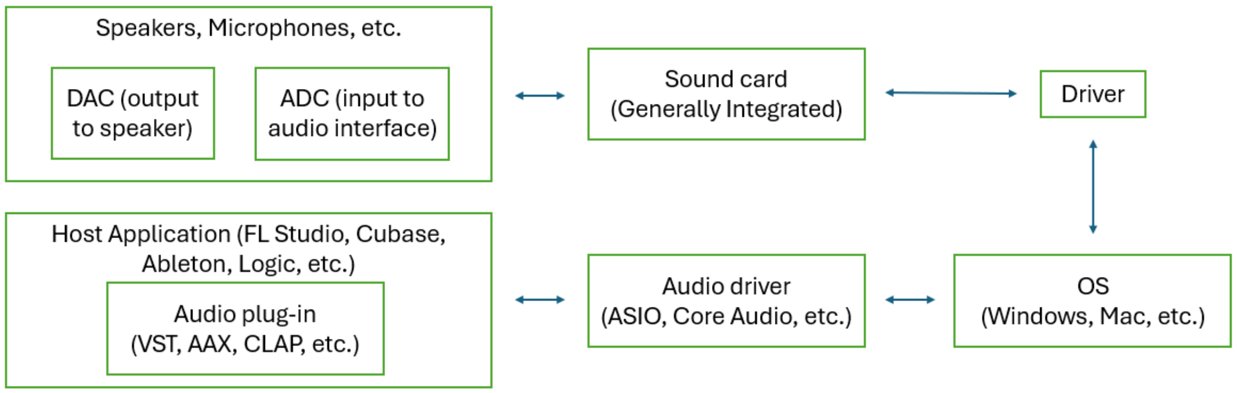
Speakers, Microphones, etc.
DAC (output to speaker)
ADC (input to audio interface)
Sound card (Generally Integrated)
Driver
Host Application (FL Studio, Cubase, Ableton, Logic, etc.)
Audio plug-in (VST, AAX, CLAP, etc.)
Audio driver (ASIO, Core Audio, etc.)
OS (Windows, Mac, etc.)

Figure 4: A graph of how audio information is handled throughout the system.

4.2 Comparison of Frameworks

There are several competing standards to create virtual synthesizers (among other pieces of music software). They include - but are not limited to - VST(3), AU, AAX (DSP/Native), LV2, and CLAP. Each have their own benefits and drawbacks, but can largely be broken down into "what operating system are you writing for". VST is supported on both Windows and MacOS, while AU and AAX are supported exclusively on MacOS.

4.3 Development with VST3 SDK

4.3.1 Development Tools

The VST3 SDK contains several tools and programs that assist in developing the boilerplate code necessary for creating VST applications and ensuring they fall under the required specification. These are the project generator, GUI editor, and validator. By using the project generator, a CMakeLists text file is generated. The program automatically compiles it, which creates a Visual Studio solution pre-configured with several components to test VST applications easily.

When the Visual Studio solution is started, a host application is started that can be used to quickly test MIDI input and audio output.

4.3.2 Workflow and UML diagram

Functions and classes under the VST3 SDK are organized under namespaces, in order to avoid name collisions [2]. The Steinberg namespace contains the Vst namespace, which in turn contains the programmer's namespace created by the project generator.

4.4 Creating Oscillators

There are several methods of generating sine, square, saw, and triangle waves.

4.5 Creating An Envelope Generator

5 Conclusion and Future Work

References

[1] A. Pejrolo. *Creating Sounds from Scratch: A Practical Guide to Music Synthesis for Producers and Composers.* Oxford University Press, 1st edition, 2017.

[2] W. C. Pirkle. *Designing Software Synthesizer Plugins in C++ : for RackAFX, VST3, and Audio Units.* Routledge, 1st edition, 2015.