# Virtual Reality Modular Synthesizer

Matthew Messina
Colorado State University
Project type: 2

## Abstract

For my final project I opted to build a prototype for an Analog Modular Synthesizer Simulator in a Virtual Reality (VR) environment. An analog synthesizer is a type of musical instrument where all the sound is generated by electrical, analog signals. These synthesizers are made up of individual modules which are capable of altering said electrical signals [11]. Physical analog modules can be quite expensive and take up a lot of space, however, in a VR environment none of the modules take up any physical space and are far less expensive to use or create. My prototype is meant to act as a non-physical medium of interacting with analog synthesizers, making experimentation with these devices far more accessible.

**Keywords**: Analog, Modular, Synthesizer, Oscillator, Filter, Reverb.

## 1 Introduction

Analog Modular synthesizers are a type of musical instrument which produce sound in a rather non-traditional way, or at least when compared to other musical instruments that came before it. All the sound is produced by electrical, analog signals which can be passed into individual "modules" to be altered [11]. More specifically, an "oscillator" module produces electrical signals in the form of waves which can be passed into other modules. The oscillator that is currently implemented in the prototype has support for 3 different wave forms: sine, triangle, and square. The signal produced by the oscillator can then be passed into other modules such as a Filter, which is responsible for cutting out specified frequencies, or a reverb which can add "space" and "atmosphere" to a sound.

Even though the idea of modular synthesizers may seem dated to some, individual modules are still being widely produced today, many of which are highly sought after. A paper published by LMU Munich entitled "The Modular Backward Evolution – Why to Use Dated Technologies", claims "We consider the current device development of many manufactures as a strong indication for a return of tactile experiences in this domain reflected by the interface components losing their pure digital nature." [2]. In other words, they see this sudden surge of interest in modular synthesizers amongst musicians as derived from a desire for a tactile response while making electronic music. One of the primary goals of this prototype is to develop a digital modular synthesizer that still manages to deliver somewhat of a tactile feel that traditional simulators fail to provide.

Most physical synthesizer modules that one might buy tend to be relatively expensive, making it hard for beginners to dive in and really learn about the ins and outs of a modular synthesizer. Not to mention the inability to save patches to a disk or memory since they are constructed with physical wires and electrical signals. As analog synthesizers have become more popular amongst experimental musicians within the last few years, new software-based solutions to the aforementioned problems have come up. A prime example of one of these solutions would be an open-source application known as "VCV Rack". One of the key differences between my prototype and VCV Rack, however, is that Rack is entirely set in a 2D environment, whilst my prototype is in a VR environment.

The prototype currently supports 4 different modules, 3 of which feature fully adjustable parameters and settings. The first 3 modules include an Oscillator, Reverb, and High/Low-Pass Filter, while the 4th module is simply a custom representation of an output module, and it is only responsible for routing the audio from the module chain to the output specified by Unity. The prototype was almost entirely constructed using the Unity Game Engine and its built-in XR interaction toolkit. All the modules are built off of Unity's Audio Filters API, with some of them being entirely custom game objects (the oscillator), and others being filters provided by Unity out of the box that can be modified by custom Game Objects. More specifically the High/Low-pass filters and the Reverb module come from built-in Unity filter objects.

The typical modular synthesizer is interconnected with cables known as "patch cables". These patch cables connect the output of one module to the input of another. For example, an oscillator would have power going in as the input, and the electrical signals (waves) would be sent from the output port, over the patch cable, into the input of the next module. The prototype implementation does not utilize wires, as modules are instead connected to each other in the order that they are spawned into the game world. This simplification of the connections makes the simulator far less cumbersome and confusing for users to interact with. From the perspective of the Unity engine, modules are simply game components that can be attached to the oscillator object. The ordered nature of the components found in Unity's inspector view is actually very similar to the way in which modules are chained together, so implementing the "chain" functionality was trivial.

A small user study was also designed to go along with this prototype which involves measuring the degree to which users experiment with different modules. The idea is to track the amount of time users spend on one parameter or setting in a given module, and determine which modules and parameters of the 3 inspire the most experimentation.

## 2 Related Literature

Physicist Michael J. Ruiz wrote a paper discussing the physics and mathematics behind a basic modular synthesizer setup (Oscillator, Oscilloscope, and Filter) [1]. Modular synthesizers are actually one of the original manifestations of the synthesizer we know today, and provide an interface for making music based entirely around physics and mathematics [1].

Several works have previously been written on the use of Virtual Reality to simulate physical instruments including one that provides a detailed overview of the current and past landscapes of Virtual Reality Musical Instruments (VRMIs)[3,4]. In it, the author mentions several guidelines (13 to be exact) for developing VRMIs, "categorized either as user-centric or aesthetic, with some

additional observations" [3]. My VR synthesizer simulation is meant to be a balance between both of those two categories as some minor modifications needed to be made to the interface in order for it to feel more intuitive to the user. For example, the lack of wiring in the prototype is partially due to the fact the handling large amounts of interconnected wires in VR can be very cumbersome and tedious.

In my research I also found a some other literature reviews discussing the potential for virtual reality as a medium for hands-on education in both academic and personal settings [5,7]. The combination of VR and musical instruments may offer people a powerful approach to involving themselves in music production in grades K-12 as music education in those grades is often subject to budget cuts [7]. According to Michael Ruiz, "The music synthesizer is very useful in demonstrating a variety of principles of sound and has become quite popular in physics courses"[1].

In my research, I discovered that another group of researchers happens to have designed a similar modular synth application based on Unity [9]. However, their implementation seems to be based on a modular synth simulation platform called "VCV Rack". My application utilizes the native Unity audio filtering functionality, more specifically, the OnAudioFilterRead() method that they provide for generating sound waves. I also found that another group of researchers created a full body interfaces for controlling the parameters of modular synthesizers via dance moves [10]. While this idea for a synthesizer interface is interesting, it seems to be a complete overhaul of the modular synth interface, and my prototype is meant to reflect traditional modular synthesizer functionality as much as possible. Finally, a paper entitled "The JD-1: an implementation of a Hybrid Keyboard/Sequencer Controller for Analog Synthesizers" describes the creation of a capacitive MIDI interface for controlling the voltage going into an Oscillator and thus, a modular system [6].

## 3 Implementations of Modules

As mentioned in the previous section, the prototype features 4 different digital representations of analog modules. The 3 most feature-rich of those modules include the oscillator, high/low-pass filter, and reverb. While the 4th module that was included is a representation of what would normally be a physical output (speaker, headphones, etc.). Although each module has its own 3D model associated with it, virtually all of the functionality behind them is added to the oscillator module in the form of Unity Component objects.

### 3.1 Oscillator

The oscillator is arguably the most important component of any modular synthesizer setup. It is responsible for generating all the sound going through the chain and there are many different kinds of oscillators. In the case of this prototype, the oscillator is capable of producing three different types of sound waves. These wave-forms include Sine, Triangle, and Square, each with their own unique shapes and characteristics. Below is a figure showing what the oscillator model looks like in the prototype, including labels which define what parameter each item interacts with.

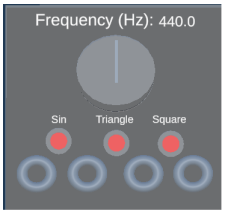


**Figure 1: Front view of Oscillator module's 3D model**

As for the code itself, the sound is generated inside of a function provided by the Unity engine with the name "OnAudioFilterRead()". According to the official Unity documentation, if the function is implemented in a script attached to an object "Unity will insert a custom filter into the audio DSP chain" [14]. In other words, Unity constantly calls this function and places your modifications defined in the script on top of the currently running audio chain. Passed into OnAudioFilterRead() is a buffer containing float values which represent audio features (amplitude) from the previous audio source in the chain. If no audio is currently playing on the filter, the buffer will be filled with 0.0 values, thus, to generate a wave we need to perform various calculations on the values in the buffer depending on the wave that needs to be generated.

For the sine wave, the formula would look similar to the following in code form (C#).

```
data[i] = (float)(gain * Mathf.Sin((float)phase));
```

**Figure 2: From Oscillator.cs in my source-code. Acquired from: https://www.youtube.com/watch?v=GqHFGMy_51c**

To explain further, "data[i]" is meant to represent a given value inside of the buffer passed into OnAudioFilterRead(). The gain is essentially the max volume set for the wave and the "phase" is equal to: **(2 x frequency * pi) / sample frequency**, where sample frequency is the rate at which Unity places points on the wave and plays them back.

As for how the square wave is being generated by the prototype, the formula would look something like the following in code format.

```
if (gain * Mathf.Sin((float)phase) >= 0)
    data[i] = (float)gain * 0.6f;
else
    data[i] = (-(float)gain) * 0.6f;
```

**Figure 3: From Oscillator.cs in source code. Acquired from: https://www.youtube.com/watch?v=GqHFGMy_51c**

Above one can see that the formula for defining a buffer item on a sine wave is being used as the basis for a conditional statement. The given buffer item is then updated according to whether the result of the formula is positive or negative. One can also see that the amount by which gain is being multiplied by in the final calculation is a fixed amount. This fixed multiplier is what gives the square wave it's unique shape.

Finally, the triangle wave is being generated in our prototype by the following code.

```
data[i] = (float)(gain * (double)Mathf.PingPong((float)phase, 1.0f));
```

**Figure 4: Triangle wave formula from Oscillator.cs. acquired from: https://www.youtube.com/watch?v=GqHFGMy_51c**

As one can see, this method of generating waves is incredibly similar to how sine waves are generated. However, the key difference here is the use of Unity's PingPong method, which according to their official documentation page "returns a value that will increment and decrement between the value 0 and length"[15]. In the case of the prototype and the code shown above, the value for length is simply 1.0. Below is a figure demonstrating what each of the previously mentioned waveforms might look like.

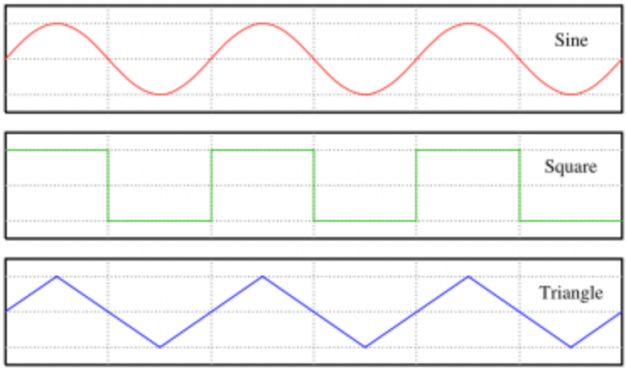


**Figure 5: Differences between waves produced by prototype [12]**

### 3.2 High/Low-Pass Filters

Apart from the oscillator, filters are the most popular analog module that synthesizer enthusiasts purchase, as they allow for an incredible amount of fine-tuned control over the frequencies being produced by the chain. Normally, filters can be divided into two separate categories, high-pass and low-pass. According to somanytech.com, a high-pass filter "allows the higher frequency above cutoff frequency and attenuates all frequency below the cutoff frequency" [16]. In other words, a high pass filter drops any frequencies that fall below the specified cutoff value, letting in only higher frequencies. Low-pass filters work almost the exact same way only instead of dropping frequencies that fall above the specified cutoff value, letting in lower frequencies.

In the case of the prototype, the decision was made to combine both the high and low pass filters into the same module, as it is easier on the user when they are dealing with fewer modules. In order to maintain the customizability that comes with separate filter modules, the knobs are set to a neutral position by default when the filter module is spawned in and labelled accordingly. While the effects of this module may not be immediately apparent to the user, it is essential for anyone who wants the ability to fine tune the timbre of their sounds and keep certain frequencies in check. Depicted below is the 3D model used for the High and Low-Pass filters in the prototype.


**Figure 6: Front view of Oscillator module's 3D model**

### 3.3 Reverb

The reverb module is the final module that was implemented in the prototype with fully adjustable parameters. This is another module that has the majority of its functionality defined by the built-in Unity filter components. The specific parameters that I chose to implement for this module include the Dry/Wet signal, Decay Time, and Diffuse. Dry/Wet can be defined as a ratio of how much the reverb affects the original sample. If the knob is set all the way to dry, the reverb has no effect over the original sample. As the user turns the knob toward the "wet" position, the signal becomes more and more affected by the reverb. Decay time is the amount of time it takes for the reverb effect to dissipate and "dry up" the signal. Lastly, the diffusion parameter is responsible for defining the space between echoes on a reverberated effect. Below is an image depicting the 3D model that is used for the reverb module in the prototype.

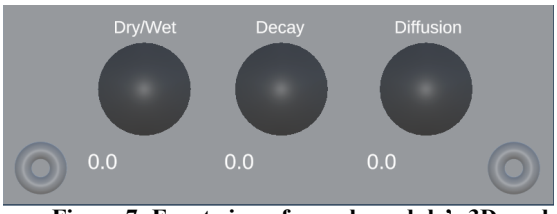


**Figure 7: Front view of reverb module's 3D model**

The effect that the reverb module has on the rest of the chain can be quite dramatic depending on what the parameter values are set to. It can give almost any sound a feeling as if it is being played inside of a massive empty room, effectively giving it "space to breath" and making it sound less robotic. For example, increasing both the decay and wet values over a sine wave will produce an almost haunting ambient tone that seems to drag along as one changes the frequency value of the oscillator.

## 4 Comparing to Similar Works

As mentioned previously, there are quite a few similar projects that have been developed over the years which aim to provide a convenient and smooth digital interface for constructing analog synthesizer patches. Perhaps one of the most prominent examples of such an application would be one known as "VCV Rack". VCV Rack is, according to their website, an "open-source Eurorack Modular synthesizer simulator" [13]. The interface of VCV is 2-dimensional (2D), and the modules are connected via dedicated input and output zones along with digital patch cables. The figure below is an example of a very basic setup in VCV Rack to help with later comparison to the prototype.


**Figure 8: VCV Rack Basic Patch**

The patch shown above in VCV Rack consists of an oscillator (VCO-1), a filter (VCF), and a mixer/output device combination. As one can see, each of the modules' inputs and outputs are interconnected with virtual patch cables. There are quite a few key differences between the VCV interface and the prototype's interface with a very similar patch. To start, since the prototype works in a 3-dimensional (3D) VR environment, the modules can be placed anywhere in the 3D space. Whereas VCV requires modules to be placed on a 2D "track", which is visible on the top and bottom of Figure 8. The other major difference between VCV Rack's and the prototype's interfaces would be the use of patch

cables. It was determined that the VR prototype would be better off without the wire functionality because it could be quite cumbersome attempting to wire the modules together with the Oculus Rift Touch Controllers. As an alternative, whenever a module is spawned into the environment, it's basic effect (if any) is immediately applied to the chain. One of the primary limitations of this lack of wires is that multiple oscillators are not able to be utilized within the patch.

## 5 User Study

When this project was initially proposed, the intention was to create a user-study with the goal of determining which type of tutorial would be best for teaching users how to setup a basic synthesizer. However, it quickly became apparent that this idea for a user study did not reflect the intention of the prototype in any way, and that was to build an improved, Virtual Reality interface for an analog synthesizer simulator. Instead, I designed a different study that is less focused on education, and more focused how my prototype is interacted with by a user.

The goal was to design a system which tracks the parameter adjuster interactions on each module, both the number of times it was interacted with, and the amount of time a user left that parameter set in seconds. The hypothesis was that if a specific parameter is set for a long time with relatively high interaction counts, the module and its parameters inspire experimentation from the user, and provide some sort of tactile experience for the user.

To test my Hypothesis, I planned to provided participants with a survey asking them questions regarding the tactile experience of interacting with the synth they created, as well as whether they felt the desire to experiment more after a timer (set to 3 minutes for each participant) has completed. It would also ask the participants questions regarding which module(s) they felt were the most enjoyable or interesting to use.

Unfortunately, due to technical issues with implementing both the user study and the prototype, I was not able to fully complete the implementation of the user study, as the prototype itself ended up taking priority. While there are some traces of code in the oscillator which track the amount of time in seconds each oscillator type is used, there is no other working code for this experiment present in any of the other scripts.

## 6 Difficulties with the Audio Engine

Throughout the development of this project, I happened to run into quite a few difficulties. The most significant of which was the difficulty with the framework that the prototype originally utilized, known as JUCE. JUCE is a C++ framework for developing audio plugins for Digital Audio Workstations (DAWs), which is just another moniker for modern music production software. JUCE happens to have incredible support for the types of process chains necessary for building an application such as the prototype, and even has Unity plugin support. Unfortunately while attempting to integrate some basic modules into a Unity plugin, I realized just how limited JUCE's Unity integration really is at this point in time. I was unable to access and modify a lot of the data about the chain itself from the unity interface, and due to the lack of documentation on JUCE's side regarding Unity integration, it became apparent that a new audio solution was necessary in order to get the prototype completed in the allotted amount of time.

## 7 Conclusion

In summary, I have developed a prototype of a VR modular analog synthesizer simulator which is intended to provide an improved interface for constructing basic patches. In hindsight, it might have been more beneficial to utilize a different framework for developing the modules themselves as opposed to native Unity or JUCE. The unity Filter system turned out to be fairly limiting in terms of the number of modules that could be potentially created. Overall, I am quite pleased with the result of the prototype, even if it may be slightly more limited in functionality that I initially pictured.

## 8 References


[1] Michael J. Ruiz. 1985. Modular synthesizers. (March 1985). Retrieved May 13, 2021 from https://aapt.scitation.org/doi/abs/10.1119/1.2341749?journalCode=pte

[2] Beat Rossmy and Alexander Wiethoff. The Modular Backward Evolution – Why to Use Outdated Technologies.

[3] Stefania Serafin, Cumhur Erkut, Juraj Kojs, Niels C. Nilsson, and Rolf Nordahl. 2016. Virtual Reality Musical Instruments: State of the Art, Design Principles, and Future Directions. *Computer Music Journal*40, 3 (2016), 22–40. DOI:http://dx.doi.org/10.1162/comj_a_00372

[4] Teemu Mäki-Patola, Juha Laitinen, Aki Kanerva, and Tapio Takala. Experiments with Virtual Reality Instruments.

[5] D. Gu and F. Li, "The Exploration of the Application of VR Technology in Music Education under the Background of Internet+," *2020 International Conference on Modern Education and Information Management (ICMEIM)*, 2020, pp. 9-12, doi: 10.1109/ICMEIM51375.2020.00010.

[6] Jeff Snyder and Andrew McPherson. The JD-1: an Implementation of a Hybrid Keyboard/Sequencer Controller for Analog Synthesizers.

[7] C. D. Wickens, "Virtual reality and education," [Proceedings] 1992 IEEE International Conference on Systems, Man, and Cybernetics, 1992, pp. 842-847 vol.1, doi: 10.1109/ICSMC.1992.271688.

[8] Nino Auricchio and Paul Borg. 2016. New modular synthesizers and performance practice. (November 2016). Retrieved May 13, 2021 from http://repository.uwl.ac.uk/id/eprint/2963/

[9] Matthew Hughes and Andrew Johnston. URack: Modular audio-visual composition with Unity and VCV Rack.

[10] Yanto Browning, Stephanie Hutchison, and John McCormick. Performing High Voltage: Full body gestural control of Eurorack modular systems.

[11] Kieron Brown. 2021. How Do Analog Synths Work? - Oscillators, Filters & Envelopes Explained. (April 2021). Retrieved May 13, 2021 from https://integraudio.com/how-do-analog-synths-work/

[12] Ucancode Software.Retrieved May 13, 2021 from http://www.ucancode.net/real-time-wave-sine-square-triangle-signal-gererator-csharp-source-code.htm

[13] Anon. VCV Manual. Retrieved May 13, 2021 from https://vcvrack.com/manual/

[14] Unity Technologies. MonoBehaviour.OnAudioFilterRead(float[], int). Retrieved May 13, 2021 from https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnAudioFilterRead.html

[15] Unity Technologies. Mathf.PingPong. Retrieved May 13, 2021 from https://docs.unity3d.com/ScriptReference/Mathf.PingPong.html

[16] Anon. 2020. What is High Pass Filter? Its response curve, types, design. (January 2020). Retrieved May 13, 2021 from https://somanytech.com/high-pass-filter/