Modeled Modules: A SuperCollider Extension Project

Aiden Benton

Senior Capstone Project

Dr. Maxwell Tfirn

04/26/2024

When deciding on the content of my capstone project, I knew that I wanted to combine software engineering and music in some capacity. There are numerous ways to accomplish this union, so the sheer number of possibilities was rather daunting. Deciding on SuperCollider extension development was the best fit for numerous reasons. My technical knowledge and capabilities did contribute to the decision, however the creative relevance of the project was the main driving force.

The extensions themselves are implementations of certain analog synthesizer modules as extensions in SuperCollider. Specifically three modules were selected: Wavefonix Boolean Logic, Mutable Instruments Ripples, and Erica Synths DIY Delay. This selection would require very different implementations for each extension and resulted in unique programming challenges. These modules were also selected due to the new compositional uses their implementations would provide as part of the SuperCollider environment.

The cultural relevance of my project is rooted in the history of synthesizers and computer music in the modern age. The modeling of analog modules in a digital environment is not just an exercise in possibility, but one which highlights current trends and demands in music making. The history provides the necessary context of how far technology has evolved and how aspects throughout history are embodied in modern day uses, including this project.

From Analog to Digital and Back Again

The earliest forms of electronic instruments were analog synthesizers by definition. Simply put, an analog synthesizer is a machine which uses electricity as an analog for sound waves. Modern day hardware will often represent this as one volt per octave. By this definition, the earliest analog synthesizer was Thaddeus Cahill's Telharmonium in 1896. This massive machine was intended to share music over telephone lines. It had many drawbacks like its size, the amount of electricity it required, and issues with crosstalk on the telephone lines. All the teleharmoniums were eventually scrapped and no recordings or original models exist today.[1]

In the 1920s the theremin and the Ondes Martenot come out. Both have similar sonic qualities as they both used heterodyning as their form of synthesis. These instruments offered rudimentary control over certain aspects of sound. The Orgues des Ondes which was unveiled in 1929 was the first polyphonic analog synthesizer. It used an unwieldy 700 vacuum tube oscillators with 10 oscillators for each of the 70 keys. It was capable of rudimentary forms of additive and subtractive synthesis due to the polyphony and individual tuning of key timbres. Unlike the theremin and Ondes Martenot which are still used today, the Orgue des Ondes did not catch on.[2]

The next major early analog synthesizer was the Trautonium invented in 1930 by electrical engineer Dr. Freidrich Tratuwein. The goal was to provide more expressive possibility than a traditional keyboard instrument by using a metal wire which was pressed on to play. Oskar Sala was the foremost Trautonium virtuoso, and he was modifying and upgrading the instrument until his death. The Trautonium had a limited commercial run with manufacturer Telefunken, but the price of the units prevented their popular adoption.[3] Price of technology is a theme across the history of electronic music.

[1] "Synthmuseum.Com - Magazine."
[2] Crab, "The 'Orgue Des Ondes' Armand Givelet & Edouard Eloi Coupleux, France. 1929."
[3] Crab, "The 'Trautonium' Dr Freidrich Trautwein. Germany, 1930."

The first machine to be called a synthesizer was the RCA Synthesizer Mk I in 1951 and later the Mk II. RCA was one of the leading record companies at the time, and they wanted to develop a way to generate pop hits and instrument sounds without having to deal with pesky musicians and artists who wanted to be compensated. RCA funded the research project run by Harry Olson and Herbert Belar which resulted in the creation of the synthesizer. The technology of the time was still quite behind what RCA had originally envisioned; however, the RCA Synthesizer was an important milestone in computer music and analog synthesizers. The synthesizer worked like mainframe computers of the time working with a roll of hole punched paper. This allowed for potentially millions of different programmable settings.[4]

Around the same time, Bell Labs hired Max Mathews. He would go on to invent the first music programming languages collectively called MUSIC-N. The first iteration, MUSIC I, is best described by Mathews himself quote: "[MUSIC I was] terrible—it had only one voice, one waveform, a triangular wave, no attack, no decay, and the only expressive parameters you could control were pitch, loudness, and duration."[5] MUSIC II improved on this by adding multiple oscillators and inventing the concept of a wavetable oscillator which was much less computationally intensive.

MUSIC III is considered the first fully implemented version of the language and created two important concepts which would be used by later music computing languages. The first is the unit generator or UGen. Digital processing of sound uses samples, or discrete single values that represent a sound wave at a specific time. These samples are units. UGens take input parameters and output units. For example, a sine

[4] Crab, "The 'RCA Synthesiser I & II' Harry Olson & Herbert Belar, USA, 1951."
[5] "Chapter 5: Computer Music," 109.

wave UGen may take a frequency parameter and output the necessary values to construct that sine wave. The other important concept is that of the instrument versus the orchestra. Instruments were collections of UGens or the sound generators and the orchestra is the structure of how they play.[6]

MUSIC IV didn't add new features but was the version of MUSIC-N that Mathews shared with John Chowning who would go on to invent FM synthesis. MUSIC V was the final iteration of the MUSIC-N languages and was notable because it was implemented in FORTRAN.

At this time in history, computers still take up a whole room and are programmed with punch cards. One problem with early programming is that programs would only work on the computer they were written for as different computers handled instructions differently. FORTRAN solved this problem as the first programming language. If computers could translate FORTRAN into their own instructions, the computer could run the FORTRAN program. This made MUSIC V the most accessible of all the MUSIC-N programming languages and made it possible for countless others to work with the language.[7]

Around the time MUSIC IV and V are being conceived and polished, Bob Moog goes from selling theremin kits to designing his own analog synthesizers. His friend and composer Herb Deutsch asked him to create a new kind of instrument, so Moog put together the first modular synthesizer prototype lovingly called "The Abominatron". The modular synthesizer would soon become one of the most powerful music making tools. Moog would launch his "Synthesizer" series in the late 1960s. These modular

[6] Wang, "A History of Programming and Music," 61–62.
[7] Wang, 63.

synthesizers came in studio and portable versions, even though they were quite large. Each new iteration added more features and modules to work with, with the Synthesizer 3 being the first fully realized synthesizer.[8]

Early Moog synthesizers fell into the same cost trap that its analog predecessors did. The units were so expensive, many musicians couldn't get access to them. That was until the Minimoog Model D came out in 1970. It is considered the most iconic synthesizer of all time. The unit was much smaller than other synthesizers and had common routings hard-wired into the circuitry. This made it not only cheaper to produce, but also more user friendly. It is the archetype for all synthesizers that follow.

In the advancing computer world, the advent of the microprocessor kickstarts the advent of modern desktop computers and personal workstations. Now you could perform sound synthesis from the comfort of your desk typing on a keyboard rather than going into a hot room with a mainframe and painstakingly inserting punch cards. Computers could now be used by people who don't have degrees in mathematics and engineering. The C programming language is invented around this time as well and is used to create many new sound processing tools. Csound is of note due to its invention of the concept of control rates and audio rates. For audio to sound good to our ears, tens of thousands of samples must be generated to replicate that sound signal. It wouldn't make sense to run a UGen which just changes the frequency at the same rate. The control rate calculates values much less frequently than audio rates, saving on limited computer resources.[9]

[8] "Evolution of Moog Synthesizers 1964-2002."
[9] Wang, "A History of Programming and Music," 65.

With microprocessors, you get the Prophet-5 the first synthesizer which combined analog and digital technology. Invented by Dave Smith and John Bowen, the Prophet-5 used analog technology for sound synthesis but used digital technology to store presets and perform keyboard scanning. It was also one of the best sounding, early polyphonic synthesizers. The combination of analog and digital allowed it to outclass its competition and set a digital precedent moving forward.[10]

The 1980's sees a decline of analog instruments with the advent of cheap and powerful digital synthesizers like the ubiquitous Yamaha DX7, but the new expensive digital synthesizers drive new musicians towards second-hand analog equipment. This in turn drives the price of analog synthesizers back up due to the new demand. To this day, it is very difficult to purchase synthesizers like the Roland TR-808 and TB-303. Again, digital technology comes to the fore, modeling the highly sought after analog models. This is also when software modeling analog synthesizers starts becoming a popular alternative like Native Instruments Reaktor and Reason Studies (formerly Propellerhead Studios) ReBirth, which existed solely to model the Roland synthesizers.[11]

There are a few important musicians who put into context the cultural importance of this development in music making technology and how it relates to my project, the first being Wendy Carlos. She skyrocketed the analog synthesizer into the mainstream with her album *Switched on Bach*. This daring project which tool thousands of hours to complete due to the monophonic nature of the synthesizers became the second classical music album to go platinum and won three Grammys. Carlos would go on to compose music for incredibly influential films like *The Shining*, *A*

[10] Bloderer, "Sequential Prophet-5 - Milestone and Musical Legend."
[11] "Reason Studios"; "Top 10 Pivotal Moments in REAKTOR Music History."

Clockwork Orange, and *Tron*. Her music is influential to this day, recently providing inspiration for the *Stranger Things* soundtrack.[12]

The next group which rocketed forward electronic music is Kraftwerk. This group founded by Florian Scheider and Ralf Hütter provided the foundation for numerous electronic music genres from electro to synth pop. The group's breakout hit was the single edit of "Autobahn". This song set a new trajectory for popular electronic music. When it came to this new form of music making Schneider said "We are more like vehicles, a part of our mensch machine, our man-machine. Sometimes we play the music, sometimes the music plays us, sometimes... it plays." This sentiment echoes modern algorithmic music and DJ culture.[13]

The last and most recent figure is Deadmau5 an EDM artist. His studio contains a 30-foot wall of analog synthesizers. Analog has never gone out of fashion. He also uses digital software very frequently when making music. In an interview, he talks about the software VCV Rack which emulates popular analog synthesizer modules for patching together in a digital rack.[14] There is a demand for digital implementations of analog synthesizers, and my project directly contributes to that growing community.

Extending SuperCollider

SuperCollider was invented in 1996 by James McCarthy and was released as open-source software in 2002.[15] By making the project open source, the community had the ability to directly influence and modify the program to their liking. The most current

[12] Villalba, "Wendy Carlos."
[13] Reynolds, "How Florian Schneider And Kraftwerk Created Pop's Future."
[14] Matt Mullenlast, "Deadmau5."
[15] "SuperCollider Home."

major version of SuperCollider is version three. Despite its age, SuperCollider is still seeing updates, and this can largely be attributed to its open-source nature.

SuperCollider as an audio processing programming environment has three distinct components: scsynth, sclang, and scide. scsynth is the back end of the SuperCollider language. It is where the actual sound synthesis and processing occurs. What is unique about it is the ability to use the server independently from the other components. The server communicates with the Open Sound Control protocol; as long as the device or programming language used can format its messages in a way the server understands, it can use the server.

SuperCollider's packaged way of communicating with the server is sclang. This is an interpreted language, meaning that it can be executed line by line in real-time. This facilitates sound synthesis by allowing the adjustment of parameters with quick feedback. The language itself is based on C and Smalltalk with syntax that is accessible to those who may not have an extensive background in programming yet is familiar to those who are.[16]

The final part of the SuperCollider package, scide, is the most optional of the three parts. IDE stands for integrated development environment which is akin to a program like Microsoft Word but for specific programming tasks rather than document editing. The IDE provides very useful functions for working with SuperCollider like syntax highlighting which helps distinguish between different components of a program at a glance. It also provides easy access to server functions like booting and metering. It also shows the current system resource usage of the server. The most useful part of the

[16] Wang, "A History of Programming and Music," 69.

IDE is the built-in documentation browser. This makes working with SuperCollider very accessible. It is very easy to look up the implementation and information about everything in the language. It also comes with built-in tutorials and guides to help beginner's get started. Overall, SuperCollider is a very accessible introduction into sound programming.

SuperCollider extensions are not written in the SuperCollider language. Extensions are specifically used to extend the functionality of the server which is written in the C programming language. Modern extensions, like the ones created for this project, use C++, itself an extension and improvement of the C language, to add to the functionality of the server. The design of the server is highly conducive to adding extensions as plugins. There is a well-documented API, application programming interface, which allows access to all the necessary components to create new functionality. In fact, nearly all of the core components of the server are implemented as plugins. This gave me many examples to look over. In this case, the extensions are new UGens, which are a concept inherited from MUSIC-N as mentioned above.

Each of the UGens I set out to create contained unique challenges in their implementation. I intended to program them in order based on difficulty, but as seen later, my assessment was off. In this spirit, I started with the Boolean Logic module.

Wavefonix Boolean Logic

ModularGrid. "Wavefonix Boolean Logic (BL)." Accessed April 14, 2024. https://modulargrid.net/e/wavefonix-boolean-logic-bl.

The nature of this module made it very simple to implement digitally, likely much simpler than the analog implementation. Boolean means that there are only two possible states, two possible inputs and outputs. These are represented as true and false or digitally as 1 and 0. Logic is a method of comparing two values and outputting another value. For example, the AND logic gate compares two values and only outputs true if both are true. How Wavefonix implemented this style of comparison using analog circuitry is unknown, however the digital implementation for SuperCollider is very simple. Creating basic Boolean operations is often one of the first exercises one does when learning computer science.

The module has AND, OR, NAND, and NOR comparisons available. I picked an arbitrary number of operations to implement. The code uses a switch statement to process the selected logic operation. The SuperCollider server API provides functions for accessing user inputs, so the inputs are evaluated logically depending on the selected operation.

What this module offers is a unique way to make pseudorandom impulses. Because of the nature of logic, one can make estimations of how often the UGen will output a 1 based on the inputs and the selected operation. This can allow for more control over the aspects like frequency. Within the SuperCollider language, the UGen can be used to trigger events sporadically. This can be useful when making breakbeats or with granular synthesis.

This was a very good module to start of with programming due to its simplicity. It allowed me to familiarize myself with the SuperCollider extension development toolchain and API functions. Rather than having to focus on the nitty gritty of signal processing, I could understand how fitting functionality into SuperCollider worked.

Mutable Instruments Ripples

ModularGrid. "Mutable Instruments Ripples," April 14, 2024. https://modulargrid.net/e/mutable-instruments-ripples.

The Ripples module is a filter-based module. I thought that filters would be easier than delay to implement due to delay requiring memory allocation as discussed later. I was very wrong. Digital filters in digital signal processing are very complicated from a layman's point of view. There is a lot of math that goes into filter design. Filter design could take up the entirety of a project. Trying to cram it into one third of mine did not turn out very successful. I did my best to understand how to implement digital filters using YouTube videos, "Digital Filters for Everyone" by Rusty Allred, and "The Computer Music Tutorial" by Curtis Roads. While I do know much more about how digital filters work, it's more accurate to say I know more about how much I don't know.

The Ripples module has two lowpass filters, and one bandpass filter. From the named adjustable parameters on the module, they are all specifically resonant filters. I was going to use the equation's provided in Allred's book, but there are no equations for resonant filters. I decided on using Type I Chebyshev filters because they have a parameter called the ripple. I attempted to implement the mathematical equations from the book, but every attempt resulted in an output of noise, not a filtered signal. I spent much of my time trying to troubleshoot why this was an issue, but my limited knowledge of filters prevented me from even knowing where to start. Rather than give up, I decided to use a filter library I found on GitHub which implemented the filters I needed.[17]

After moving the hassle of digital filtering elsewhere, the next difficulty seemed rather mundane in comparison. SuperCollider allows for the implementation of UGens with multiple separate outputs. I wanted to implement this structure so that each filter output could be manipulated separately or together. The implementation of this structure happens in the SuperCollider language class declaration which there is significantly less documentation on than C++. I looked through how SuperCollider implemented core UGens with the same functionality, and with a little trial and error, I was able to get it functioning.

The module provides new filters which weren't previously part of the SuperCollider environment. The new implementation also allows for expanded creativity due to its multiple outputs. One output can be delayed, another distorted, and one not played at all. It adds many new avenues for composition and creativity.

Erica Synths DIYDelay

I left the delay to last due to the memory allocation requirements, as mentioned above. Dealing with system resources adds an additional level of stress to writing a program. No longer is a failure state simply the code not working, it could make your

[17] Porr, "Berndporr/Iir1."

computer crash. Fortunately, the SuperCollider API provides helper functions for real-time memory allocation. My worries about memory issues were promptly averted.

Erica Synths module has several standard delay options like delay time and feedback. It also has a hold button which repeats the same delay line, a reverse button which reverses the delay line, and an add button. The add button acts similarly to a loop station by storing the delay and continuing to play it back. I was not able to implement this functionality in my module due to time constraints. The module also has the option for digital or tape delay. With tape delay on, it applies a saturation filter to the signal. This feature needs debugging as of writing.

ModularGrid. "Erica Synths DIY Delay," March 2, 2024. https://modulargrid.net/e/erica-synths-diy-delay.

The initial delay algorithm used came from the SuperCollider example-plugins repository on GitHub. I had to alter the implementation to work with the modern C++ interface and add additional functionality. The algorithm has built in efficiencies I may not have thought of like using a bitwise and operation as a faster modulo operator. I added the reverse function by implementing a specific pointer which read in reverse, and the hold function simply doesn't update the stored delay buffer if hold is on. It really turned out much simpler than I thought.

Conclusion

I plan on continuing development of this project in my free time to iron out bugs and potentially add new features and modules. Knowing I'm contributing to a long history of computer music making makes a real impact on me. This form of music making opens the door to totally new avenues of creativity, and adding to that potential is incredibly gratifying. I hope to see more developments in the realm of computer music as time goes on, especially due to the increasing accessibility of computers.

SuperCollider aims to be beginner friendly while also offering complex functionality to advanced users. It is a jumping off point into the vast ocean of creative potential found in various other computer music environments and software. Getting people interested in the software will open a door to a whole world to new creative endeavors that we have yet to hear.

Bibliography

Allred, Rusty. *Digital Filters for Everyone*. CreateSpace Independent Publishing Platform, 2013.

Bloderer, Theo. “Sequential Prophet-5 - Milestone and Musical Legend.” *GreatSynthesizers* (blog), February 7, 2016. https://greatsynthesizers.com/en/review/sequential-prophet-5-milestone-and-musical-legend/.

“Chapter 5: Computer Music.” In *Electric Sound: The Past and Promise of Electronic Music*. Prentice Hall, 1997.

Crab, Simon. “The ‘Orgue Des Ondes’ Armand Givelet & Edouard Eloi Coupleux, France. 1929.” *120 Years of Electronic Music* (blog), December 16, 2013. https://120years.net/the-orgue-des-ondes-armand-givelet-edouard-eloi-coupleux-france-1929/.

———. “The ‘RCA Synthesiser I & II’ Harry Olson & Herbert Belar, USA, 1951.” *120 Years of Electronic Music* (blog), September 21, 2013. https://120years.net/the-rca-synthesiser-i-iiharry-olsen-hebert-belarusa1952/.

———. “The ‘Trautonium’ Dr Freidrich Trautwein. Germany, 1930.” *120 Years of Electronic Music* (blog), September 23, 2013. https://120years.net/the-trautoniumdr-freidrich-trautweingermany1930/.

Google Arts & Culture. “Evolution of Moog Synthesizers 1964-2002.” Accessed April 10, 2024. https://artsandculture.google.com/story/evolution-of-moog-synthesizers-1964-2002/agUh-zsKII1HOQ.

Matt Mullenlast. “Deadmau5: ‘The Moog Voyager - I Can’t Seem to Escape That Synth. It’s the Last Good Moog.’” MusicRadar, September 12, 2022. https://www.musicradar.com/news/deadmau5-interview.

ModularGrid. “Erica Synths DIY Delay,” March 2, 2024. https://modulargrid.net/e/erica-synths-diy-delay.

ModularGrid. “Mutable Instruments Ripples,” April 14, 2024. https://modulargrid.net/e/mutable-instruments-ripples.

ModularGrid. “Wavefonix Boolean Logic (BL).” Accessed April 14, 2024. https://modulargrid.net/e/wavefonix-boolean-logic-bl.

Native Instruments Blog. “Top 10 Pivotal Moments in REAKTOR Music History,” July 10, 2017. https://blog.native-instruments.com/top-10-pivotal-moments-in-reaktor-music-history/.

Porr, Bernd. “Berndporr/Iir1.” C++, April 25, 2024. https://github.com/berndporr/iir1.

“Reason Studios.” Accessed April 20, 2024. https://www.reasonstudios.com/about.

Reynolds, Simon. “How Florian Schneider And Kraftwerk Created Pop’s Future.” *NPR*, May 7, 2020, sec. Music Features. https://www.npr.org/2020/05/07/852081716/how-florian-schneider-and-kraftwerk-created-pops-future.

Roads, Curtis. *The Computer Music Tutorial, Second Edition*. MIT Press, 2023.

SuperCollider. “SuperCollider Home.” Accessed April 27, 2024. https://supercollider.github.io/.

“Supercollider/Example-Plugins.” C++. 2016. Reprint, SuperCollider, April 9, 2024. https://github.com/supercollider/example-plugins.

Villalba, Juanjo. “Wendy Carlos: The Brilliant but Lonely Life of an Electronic Music Pioneer.” EL PAÍS English, December 12, 2022. https://english.elpais.com/culture/2022-12-12/wendy-carlos-the-brilliant-but-lonely-life-of-an-electronic-music-pioneer.html.

Wang, Ge. “A History of Programming and Music.” In *The Cambridge Companion to Electronic Music*, edited by Nick Collins and Julio d’Escrivan. Cambridge Companions to Music. Cambridge: Cambridge University Press, 2007. https://doi.org/10.1017/CCOL9780521868617.