California Institute of the Arts

Sensitive Dependence: An Exploration of the Unpredictable

by

Naomi Mitchell

A thesis submitted in partial fulfillment for
the degree of Master of Fine Arts

Herb Alpert School of Music
Music Technology: Interaction, Intelligence & Design

2023

Supervisory Committee

Committee Member

Committee Member

Committee Member

Committee Member

Abstract

This thesis examines controlled randomness, how to generate it, how it behaves, and its applications. After beginning with the history and context of these methods, this thesis focuses on ways of generating pseudo-random and chaotic behavior. Through the research and development of hardware and software implementations of unpredictability, I highlight the possibilities of audio and visual compositions that rely on attenuated uncertainty. The hardware consists of a collection of modular synthesizers conceptualized, designed, and realized by me during the past two years. Software comes in the form of explorations of chaotic and pseudo-random algorithms that includes different implementations of chaotic equations. Finally, the thesis addresses applications of these concepts through an interface between a modular synthesizer and a computer, the creation of larger feedback systems using elements explored in previous chapters, and examples of audio and visual work created with the various methods explored herein.

Randomness offers artists and musicians a way of creating works directly influenced by a non-human entity in the form of code and circuitry. While not conscious in a way recognized by biological entities, these systems of uncertainty seem to contain a life of their own. A collaboration between artists and algorithms brings new possibilities to their work, creating new possibilities that would otherwise not exist without the other. The intention of the user informs the application and direction of the randomness, but not necessarily the end behavior of it. But the artist always has the option to disregard the influence of the uncertain, something the algorithm lacks in its limited behavioral agency. We look primarily not at 'true randomness' but rather controlled randomness, pseudo-randomness, feedback, and chaos in the mathematical sense.

Acknowledgments

I would like to thank my wife, Lauren, for her undying support during my time at grad school and beyond. I would like to thank Eric Heep and Mike Leisz for their extensive assistance with coding and for being good sounding boards. Thank you to Audrey and Sophia, who listened to me talk about chaos and synthesizers even if they didn't understand all of it. I would like to thank Jim and Loren for supporting me creatively and intellectually. Thank you to Andrew Fitch for your research and advice. Thank you to the agents of chaos, benevolent and otherwise.

Contents

List of Figures

Chapter 1 Introduction

One of the things that have most captured my attention and drew me towards technology as a focus is controlled randomness and unpredictability. It has been ingrained in my artistic and musical practice over the past ten years, providing a structural backbone that informs other elements. My focus has primarily been on modular synthesizers, although I've expanded and incorporated software such as Max/MSP. The idea of a give and take with a musical or artistic system where I, as the performer, have a hand in guiding the course of a piece, but without the certainty that comes from composing every single element, captivates me. A system organized around uncertainty, a machine with its own agenda, an electronic ghost who steers the course of a tangled mess of patch cables or software connections.

In this thesis, I will explore different methods of generating controlled unpredictability from several angles. First, I dive into the history of modular synthesis and an overview of randomness in terms of theory and technology. Then I get into Eurorack modular synthesizer modules I designed and built, how they operate, and my design goals. After that, I move on to different ways of generating and analyzing randomness in software. Finally, I explore merging the worlds of hardware and software with custom-designed interfaces and practical applications for these concepts with both audio and video.

I don't focus primarily on 'true randomness' or completely uncorrelated systems of random values or numbers but rather on controlled randomness, pseudo-randomness, feedback systems, and chaos in the mathematical sense. Humans are pattern-seeking creatures. Our ancestors looked at the uncorrelated light shining from stars galaxies away and saw whole stories illuminated in the sky. Most music, and to a lesser extent visual art, is pattern-based. True randomness lacks some of the repetition that humans find so pleasing. Chaos and pseudo-randomness tend to form patterns that our brains find pleasing and latch on to.

While chaos and randomness are often used interchangeably, there are a few fundamental differences between the two. Randomness has no underlying order, and the behavior of a truly random system is both non-deterministic and not influenced by previous states of the system that creates it. If I flip a coin ten times, regardless of how many heads or tails, the eleventh flip of the coin is not influenced by the previous ten. Chaos is mathematically deterministic but displays sensitive dependence on initial conditions as well as sensitivity to small perturbations to the system. This is well illustrated with a double pendulum. A pendulum consists of a weight hung from a pivot point so that it may move and swing freely. A double pendulum suspends a second weight from the bottom of the first pendulum, creating a second pivot point. If you have two double pendulums and release them from ever so slightly different positions, they will initially follow a similar trajectory. However, they begin to diverge in behavior rather quickly and will follow different paths through their swinging motions, as if they were dropped from wildly different initial positions. This is of course, until the forces of friction and gravity deplete the energy and they come together again at rest.

Pseudo-randomness may appear random to observers, but the methods for generating the seemingly random string of numbers or values is fundamentally deterministic. After a period of time determined by the method of generating the pseudo-random signals, the string of seemingly uncorrelated values repeats. However, if this happens over a long enough period, you may not be consciously aware of it. For example, a 24-bit linear feedback shift register (LFSR) in a maximal configuration does not repeat for a period of 16,777,215 timing increments, much longer than the average person can keep track of.

My road to this program started in 2015, when I purchased my first Eurorack module. I had been interested in electronics, starting in 2012 during the end of my undergraduate degree with early experiments in circuit bending and noise boxes. Circuit bending is a process of opening electronics, usually battery-powered toys or instruments, and making connections the designers never intended, usually in order to glitch, corrupt, distort, or otherwise make results that may be 'unwanted' in a commercial product. Pioneered by Reed Ghazala in the 1960s, circuit bending saw a great deal of attention in the early 2000s, with artists such as Casper Electronics, Gijs Gieskes, Get LoFi, Circuitbenders.co.uk, and many more creating circuit-bent

instruments for electronic music artists. Though it saw a bump in popularity during this time, many people who were involved in circuit bending began to move towards not just modifying other people's electronics but making their own.

This happened concurrently with the rise of Eurorack modular synthesizers in the early 2010s. Eurorack is a modular synthesizer format popularized by the German company Doepfer in the 1990s that has become the most popular modular format, with hundreds of companies making synthesizers that are all compatible with each other, sharing a common form factor and power specifications. Many of these companies are just one or two people working out of their apartments or garages, which is how I run my company, omiindustriies.

Due to the fact that a whole marketplace of modular synthesizers exists in the Eurorack format, companies don't have to contend with making an entire synthesizer. This allows manufacturers, especially smaller ones, to make esoteric and specialized instruments. If a user can get a great-sounding oscillator from one company, then another company can make a distortion effect that uses an actual container of dirt without having to worry about making all the other parts of a synthesizer. This is where I entered the landscape of Eurorack as a small maker making unique and specialized instruments that fit very particular niches. I make them, first and foremost, for myself, and selling them is secondary.

One of the unique things about the Eurorack marketplace is the willingness of the participants to help each other. Much of the community comes from the realm of DIY, or Do It Yourself, a practice of building, repairing, altering, or otherwise changing things without the aid of so-called professionals, professional credentials, or even training. Many designers or engineers freely put up their schematics, code, or even PCB layouts online. One company, Mutable Instruments, which by the time you're reading this does not exist anymore, is one of the best-known and best-selling manufacturers of Eurorack modules, headed by lead engineer Émilie Gillet. She made a conscious choice to make all the Mutable Instruments open source, to the point where an entire marketplace has emerged that either directly clone the modules or redesign the circuit board and front panel, so it takes up less space. I should note, of course, most companies follow these open-source practices.

The modules I examine are the Cascading Register and Ya Jerk, both of which were developed and realized during my time at CalArts after years of research. The Cascading Register uses what is called a shift register in order to generate pseudo-random voltages and binary signals, known as gates, typically used in a modular synthesizer as timing events. Ya Jerk is an implementation of a circuit researcher J.C. Sprott laid out in his paper "A New Chaotic Jerk Circuit," with some modifications to make it more suited to musical applications, influenced by researcher and fellow Eurorack manufacturer Andrew Fitch, aka Nonlinear Circuits. These two modules provide two different ways of generating unpredictability, with the Cascading Register occupying the realm of pseudo-random and Ya Jerk sitting in the world of chaos.

Another focus of the methods explored is generating complexity from relatively simple building blocks. The Cases A-S equations, as defined by J.C. Sprott in his paper "Some Simple Chaotic Flows" and explored in the software section, boil down to three differential equations which define the X, Y, and Z parameters as they relate to each other over time. These equations consist of five terms and two quadratic non-linearities or six terms and one quadratic nonlinearity. A non-linear equation is a set of equations that share common variables but at least one of the equations includes a nonlinear element. Linear refers to a straight line with a constant change, whereas nonlinear functions change over time and bend or slope.

The custom designed interface, known as Introductions, went through several iterations and continues to grow and evolve. Through all its forms, it boils down to a microcontroller connected to jacks and potentiometers that take analog voltages and gate signals and convert them into messages recognized by the computer, typically MIDI note on/off and CC messages. It acts as a bridge between a modular synthesizer and a computer, allowing the two to communicate. Most often when discussing an interface between a synthesizer and a computer, people refer to a way to send MIDI information out from a computer and into the synthesizer, but Introductions works the other way, allowing modulation and timing signals from a modular synthesizer to control software on the computer.

Feedback refers to a self-influencing system that creates a loop by routing an output back into an input. Anyone who has had their microphone pick up their speakers on a Zoom call knows the screeching effect of feedback as the audio runs back in on itself, quickly building

to a piercing tone. However, feedback does not only exist in the realm of video conferencing but has many artistic applications. Creating a system that informs itself, particularly when containing elements of chance, provides a treasure trove of creative possibilities. Many of the techniques covered in the following pages require an amount of feedback in order to operate and may also operate as one element in a larger feedback system.

Over the course of this thesis, I dive deep into these different techniques in order to examine a microcosm of the possibilities of unpredictable behavior as it relates to audio and video compositions. It is not intended to be an exhaustive overview of all the ways to generate chaos and pseudo-randomness but rather to document my process of exploration over my time at CalArts. This thesis seeks to provide context and analysis of these techniques generated through experiments in both hardware and software to introduce a broader understanding of stochastic processes, ones that can be studied for emergent patterns but not precisely predicted.

Chapter 2

History and Context

In 1895 Thaddeus Cahill submitted the first patent for "The Art of and Apparatus for Generating and Distributing Music Electrically," for what we now know as the Telharmonium. Although there were some early experiments in electronic music instruments, the Telharmonium was arguably the first successful implementation of these concepts. Through electromagnetic synthesis methods, it could transmit music over telephone lines in Victorian America. The name comes from Telegraphic Harmony, hence Telharmonium. Concerts were not performed in person, but rather over telephone lines, allowing listeners to tune in at home or in public places equipped with telephones and loudspeakers.

Cahill's dream was to make a 'universally perfect instrument that could perfectly synthesize tones with scientific accuracy'. He imagined that this instrument would make all acoustic instruments obsolete, as it contained elements from existing instruments without the defects he perceived were inherent to their design.

Instead of simple waveforms, which one might expect from an early electronic instrument, the Telharmonium created complex harmonies from a series of sine waves generated by electrical dynamos. These dynamos, or tone wheels, included the fundamental tone and six ascending partials. The first version included 12 rotors spun at a speed determined by a belt-driven motor and allowed for six octaves of range, covering the 12 chromatic notes of western tuning. Using organ-style stops, a performer could select which partials were heard, making the Telharmonium an early example of electronic additive synthesis. The pure sound generated by the rotors, particularly in the first version of the Telharmonium, was particularly harsh, and so

Cahill included filtering in the form of secondary inductors that softened the sound and made them sound more pure.

There were three versions of the Telharmonium that Cahill made between 1895 and the final concert of the Telharmonium in 1912. All three versions were gigantic, occupying entire buildings in order to house all the parts necessary for the instrument. Unfortunately, the public's interest in the novelty of the Telharmonium waned after their initial delight in the sound, and eventually, all three versions were sold for scrap parts.

As the vacuum tube proliferated in the early 20th century, radio engineers began experimenting with different applications for them, and, quite by accident, discovered beat frequencies and heterodyning oscillators. When two radio frequency waveforms of similar but not identical frequency are played simultaneously, they combine and create a third frequency based on the difference between the oscillators. Several engineers found this idea to be inspirational, but none remembered as much as Russian engineer Lev Sergeivitch Termen, better known as Leon Theremin.

One issue many engineers ran into with vacuum tube heterodyning synthesis was the human problem. Meaning that as a person came close enough to actually perform on a vacuum tube instrument, the capacitance of the body caused variations in the pitch of the oscillators, causing instability. However, Theremin found an opportunity in this limitation, realizing that this could be a way for a performer to interact with an instrument. Thus, the first Theremin was born in 1917, also known as the Aetherophone. The original design included a foot pedal to control the amplitude and a switch mechanism to control the pitch. However, by 1920, the Theremin began to resemble the instrument we recognize today.

The Theremin includes an antenna and metal loop. The performer does not touch the theremin but controls the sound by moving their hands in the proximity of the instrument. The antenna controls the pitch of the sound, while the loop controls the amplitude or volume of the sound. The sound is reminiscent of a violin, both in the timbre and the continuous sliding between pitches that comes from moving your hand back and forth to control the sound.

The Theremin was first shown at the Moscow Industrial Fair in 1920 to astonished audiences. Vladimir Lenin was so infatuated with the Theremin that he requested lessons and eventually commissioned 600 Theremins to be built and toured around the USSR.

In 1927, Leon Theremin left the Soviet Union for the United States. The Theremin received a patent in 1928. By the 1930s, RCA began selling both kits and finished instruments to the public. However, it was not seen by many as a serious instrument, but rather a novelty or sound effect device. However, anyone who has seen Clara Rockmore perform on a Theremin knows that it is a versatile and expressive instrument, although one needs a tremendous amount of skill in order to master it.

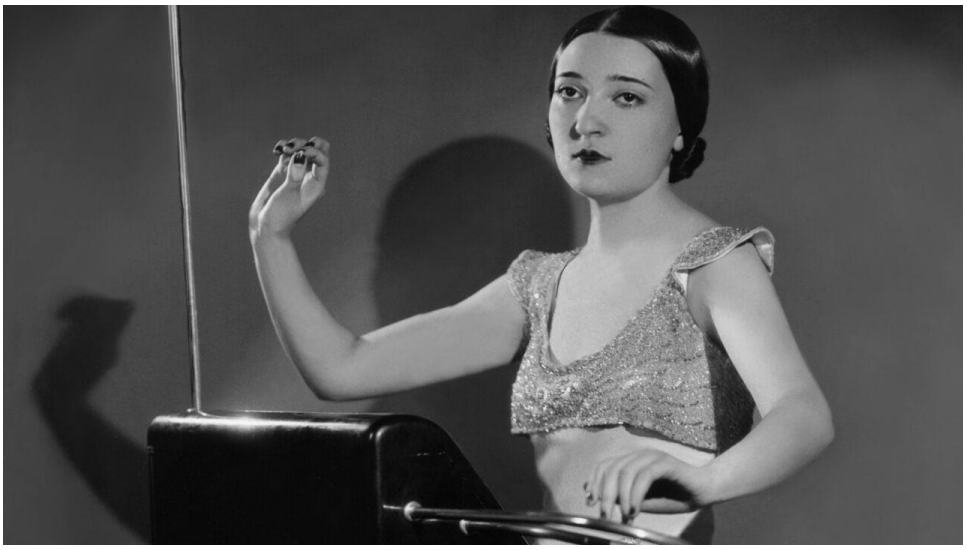


Figure 1: Clara Rockmore Playing the Theremin

Jumping forward a bit to the mid-1950s, RCA was a huge force in the world of electronic entertainment and technology. They produced everything from televisions to record players and oversaw the production of the Theremin in the United States. There was some interest in the company of analyzing the popular music of the time to figure out what made a song a hit. They thought if they could scientifically deduce the properties of popular songs, they could create a formula and crank out top 10 songs. They also wanted a way to circumvent the cost of unionized orchestras, so they sought out alternatives. RCA engineers Harry Olson and

Herbart Belar were tasked with this project, which would develop into a huge endeavor that would change the landscape of music and synthesis.

The first programmable synthesizer was born from years of hard work and hundreds of thousands of dollars. It took up an entire room in Columbia/Princeton's computer music center, known at the time as the Columbia-Princeton Electronic Music Center. The entire instrument was essentially an analog computer designed for musical purposes.

Composers would punch holes in pieces of paper that the machine interpreted through a series of relays as instructions on pitch, amplitude, envelope, and timbre for each individual note in the composition. Each parameter had four columns of holes, making 16 possible values for each parameter. The paper moved through the device at 100mm/sec and allowed for compositions of up to 240BPM.

The Mark II version of the synthesizer added several features, including doubling the number of oscillators from 12 to 24, high and low pass filters, noise, and glissando, which opened compositional freedom to many more possibilities. In addition to the punch card-controlled parameters, over 250 manually controlled sound shaping parameters were available, grouped into clusters around the 10 19" racks that the instrument took up.

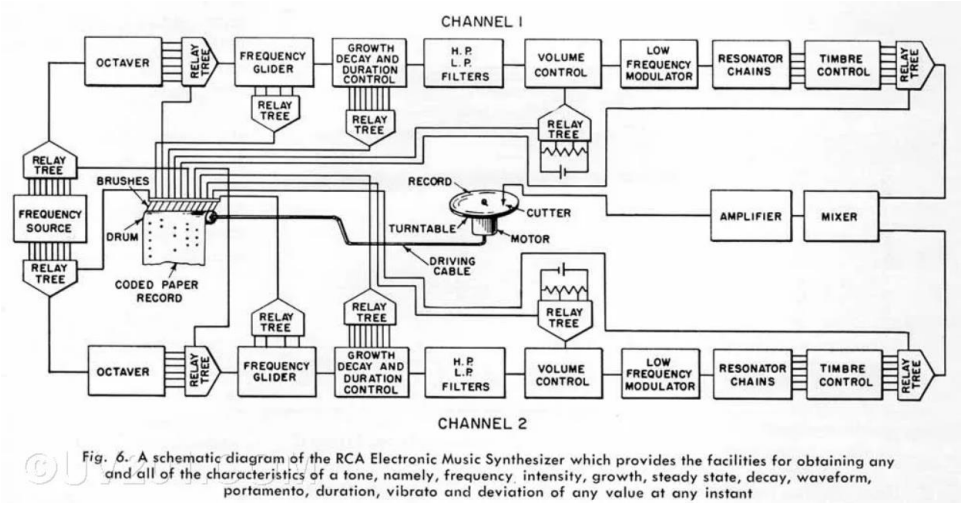


Fig. 6. A schematic diagram of the RCA Electronic Music Synthesizer which provides the facilities for obtaining any and all of the characteristics of a tone, namely, frequency, intensity, growth, steady state, decay, waveform, portamento, duration, vibrato and deviation of any value at any instant

Figure 2: Block Diagram of the RCA MKII

On both versions, the primary sound sources were vacuum tube oscillators that allowed for four voice polyphony over several octaves. It also included a dual-tier record cutting lathe, one with six cutting heads that could record two notes at a time. After recording on the first lathe, the second lathe would then mix down and cut a final record of the programmed sounds. However, by 1959, the impracticability of cutting individual records was replaced by a tape recorder.

While the synthesizer was groundbreaking in a technical sense, it was not well received by the public at large or many practicing musicians. The interface was obtuse, the sounds simplistic, and was housed in a room in a university, unavailable to the general public. As the transistor and integrated circuits began to proliferate and take over the market of electronics, electronic devices, including synthesizers, could be made much smaller, cheaper, and more reliable. While the Mark II still exists, housed in a small room at Columbia's computer music center, it has fallen into disuse.

In the early 1960s, a young man in upstate New York was selling kits for theremins and tinkering with electronics. He would go on to become *the* name in synthesizers, even to this day. That man, of course, was Robert 'Bob' Moog. With a $200 grant from Columbia University in 1963, Bob Moog collaborated with musician Herb Deutsch on the early design of what would become the Moog Synthesizer.

The spark of genius in the Moog synthesizer was both the modular nature, where the reprogramming was done with patch cables, and the voltage-controlled nature of the synthesizer. Control voltage, or CV as it's commonly referred to, is a way of controlling modular synthesizers using electricity. This allows composers to automate different parameter changes over time without having to physically turn knobs. This can be the pitch, volume, or timbre of a sound, or any other parameter within the system. The other primary signal within a modular synthesizer is a gate signal. Gate signals only have two states, high or low, and are commonly used as timing signals. In a Moog synthesizer, this is commonly generated by pressing down on a key on a keyboard and then releasing it. This on/off signal could then be used to generate an event within the modular synthesizer.

One breakthrough of the Moog was the standard of one volt per octave scaling standard. What that translates to is that an increase of voltage of one volt translates to a doubling of frequency for an oscillator, which we know as an octave. This scale is exponential, as every octave is double the frequency as the previous octave.

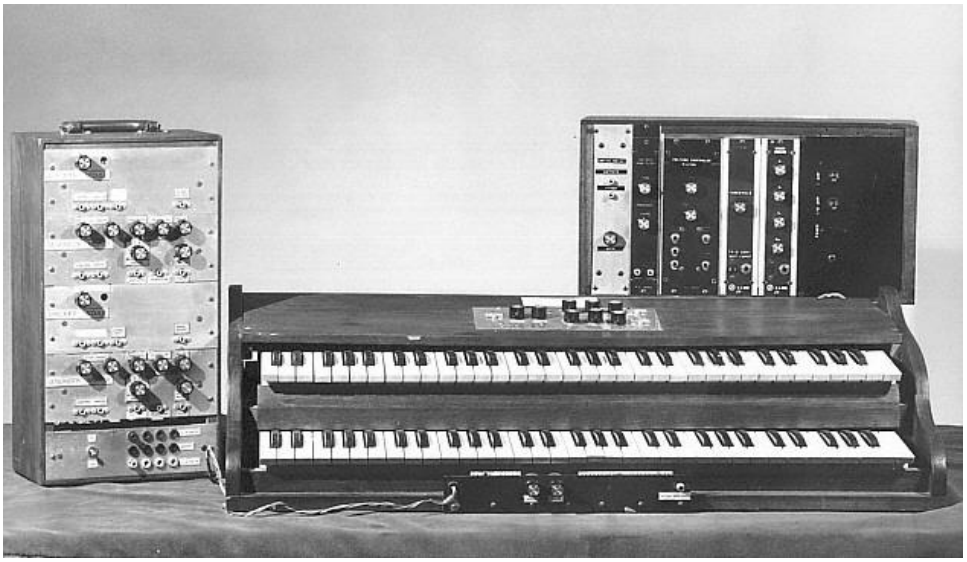


Figure 3: Early Version of the Moog Modular Synthesizer

Deutsch was responsible for the design of the ADSR envelope generators on the Moog modular that shape sound over time. An ADSR envelope has controls for the time of an attack, which is the time it takes to go from the off position to all the way on, making the difference between the hard percussive hit of a drum or the slow build-up of a bowed sound such as a violin. Decay is the time it takes to fall down from that top-most point in the signal to a level set by the sustain. The envelope stays at the level set by the sustain control as long as a signal is active or until a key is let go, and then the release parameter sets the time it takes for the envelope to fade away completely.

Moog was able to patent only one part of his Moog modular, the filter, which is the most recognizable part of the Moog modular. I'm sure you have heard the sound of a filter sweep; that characteristic "waaahhh" sound of an analog filter being swept is iconic in many different forms of music.

Around the same time, on the other side of the country, the members of the San Francisco Tape Music Center were looking for new ways to generate sounds for their compositions. Started by Morton Subotnik and Ramon Sender, the Tape Music Center included composers such as Pauline Oliveros, Steve Reich, and Terry Reilly. Subotnik put out an ad looking for an engineer who could design new ways of synthesizing sound, moving away from the large industrial laboratory equipment found in electronic music studios such as WDR Studio in Köln, Germany, and GRM in Paris, France.

Don Buchla, a former NASA engineer, took Subotnik up on this project. With a $500 grant from the Rockefeller Foundation, he sent about designing what would become to be known as the 100 Series Buchla system, although also sometimes referred to as the Buchla Box or the Electronic Music Box. While Moog made modular synthesizers that sought to synthesize sounds reminiscent of acoustic instruments and included a black and white organ-style keyboard, Buchla wanted to create a new paradigm for creating electronic music. The Buchla system utilized sequencers, randomness, and tunable touch plates in order to create synthesized sounds. Moog used filters to subtract harmonics from harmonically rich waveforms such as sawtooth and pulse, known as subtractive synthesis. Buchla, on the other hand, used additive synthesis, taking harmonically simple waveforms such as sine and triangle and adding harmonics through frequency modulation, wave folding, and audio rate amplitude modulation, commonly referred to as ring modulation.

Another major difference between Buchla and Moog was the separation of signals. In Moog and many other modular systems, all signals were treated the same, occupying the same range of voltages and using a single connection. Buchla, on the other hand, separated the modulation and audio signals in the system. Audio signals used tini-jax, a shielded cable similar to 3.5mm/1/8th cables found on your wired headphones and were referred to as Performance Modules. Modulation sources, referred to as Compositional Modules, used banana cables, which included the added bonus of being able to be stacked, allowing one signal to be sent to multiple destinations without using a specialized splitter module. While Moog and many other synthesizer makers used a 1V/oct standard, Buchla instead implemented a 1.2V/oct, making a change of 0.1V correspond to a semi-tone.

Buchla's first forays into randomness came in the form of the Model 160 and Model 165 modules released in the 100 Series. The Model 160 was a noise generator with two pairs of outputs. One pair output white noise, marked as flat, and the other pair output pink noise, marked 1/F. White noise is named as such because it has equal power throughout the spectrum of its bandwidth, containing a series of random phases, amplitudes, and frequencies within the bounds of the signal. It was thought of as akin to white light which contains all other colors. Other kinds of noise took the naming convention of kinds of light and their energy and while the metaphor doesn't exactly work, it is an easy shorthand for describing kinds of noise. Pink noise does not have equal power density through its spectrum, but if examined on a spectrogram, slopes downward at a rate proportional to the frequency of the noise. It is sometimes called musically flat noise since the energy contained in each octave interval is the same. The energy contained in the interval between 100 and 200Hz is the same as the interval between 1000 and 2000Hz. The rate at which pink noise's energy loss slopes is -3dB/octave. Pink noise is generally regarded as more pleasant to listen to than white noise, as white noise is perceived to have more high frequency content due to the way human hearing works. Many devices sold as white noise machines actually produce pink noise.

Figure 4: Buchla Model 165

The Buchla Model 165 was a two-channel random voltage generator that produced a pair of uncorrelated random voltages whenever it received a timing signal. It is a self-contained device with no user-accessible controls or any way to affect the output besides processing it with another module later down the patch chain. These random voltages were stepped and not free running, only generated or changed when the module received a timing signal. It used relays, which function both to produce the stepped voltages from a noise signal, such as white noise, and let you know when it receives a timing signal as the movement of the electro-mechanical switches gives off an audible click when it engages. These relays were used to create a simple sample and hold circuit.

A sample and hold is a device that is typically used to turn a continuous signal into a discretely stepped signal. It has a timing input and a sampling input; when it receives a timing signal, the sample and hold looks at the signal present at the sampling input and stores that voltage level. It takes that stored voltage and passes it to the output, where it is held until the sample and hold receives another timing signal. At this point, it looks at the sampling input again, moves that voltage level into the sample and hold buffer, and the old voltage level is lost. Typically, the voltage source on a sample and hold is noise, usually white noise, which gives a random collection of uncorrelated values at the output, useful for true random sequencing. It is the most common way of generating randomness in hardware as it fairly simple to implement and produces continuous randomness. While that is useful for many contexts in synthesis, true randomness is outside the scope of this thesis.

The models 160 and 165 were fairly simplistic, due to the fact that this was the mid-1960s. None of the early Buchla designs used integrated circuits, known colloquially as chips, as they were not in widespread use at the time. In lieu of that, all designs were fully discrete, meaning they were composed of basic electronic components such as transistors, resistors, capacitors, and diodes. Buchla was also charting unknown waters, producing a whole new paradigm of generating electronic music and art, and early works in any new field seem lacking and basic when looking back with hindsight. However, Don Buchla was not satisfied with the original 100 series, and through the 1960s into the 1970s worked on a new collection of modules, known as the 200 series. The 200 series would take the lessons learned from the 100

series, both from an engineering and musical point of view, and expand on their functionality, control, interaction, and reliability.

While there are many aspects of the 200 Series that could be explored deeply in their own right, I'll be looking specifically at two modules, the 265 and the 266, both named Source of Uncertainty. The Model 265 was the first iteration, followed by the 266.

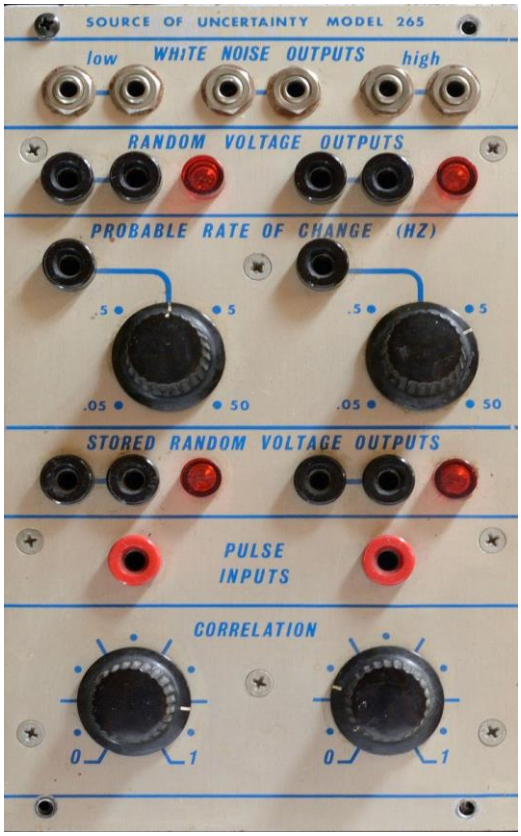


Figure 5: Buchla Model 265

The 265 contained three sections; noise, random voltage outputs, and stored random voltage outputs. Noise comes in three varieties, low, high, and flat. A fourth iteration of audio noise used in the module that does not have a direct output is a so-called 'noisy triangle' waveform. This is a 100Hz triangle oscillator that is synchronized to white noise in order to produce an equal distribution of random signals. This noisy triangle is used in both sections of the random modulation. The random voltage outputs, which would come to be known as fluctuating random voltages, output a constantly changing random voltage whose period is

determined by the Probable Rate of Change controls which vary the speed from 0.5Hz to 50Hz. Each channel outputs a pair of signals and includes CV modulation over the rate of change.

The Stored Random Voltage section retained the timing signal input and pair of distinct outputs seen on the Model 165, but added a unique feature, marked Correlation on the front panel. The correlation control is simple in its design, basically just a cross fader that fades between the incoming noisy triangle and the output of the random voltage fed back into itself. This allows users to sculpt the direction of the chaos to make it more or less related to the system's previous state. Turned all the way up, the output is always the same as it just samples itself. Turned all the way down, the output constantly changes based on the noisy triangle. The sweet spots are in the middle, where the module influences itself but includes new information to generate the random outputs.

While the 265 is a powerful source of unpredictability, the 266 is the better-known Source of Uncertainty. It retains the three pairs of noise outputs and the fluctuating random voltages from the 265 but adds several other function blocks. These are Quantized Random Voltages, an Integrator, Sample and Hold, and a new version of Stored Random Voltages, which shares the name from the 265 but is implemented in a completely different fashion.

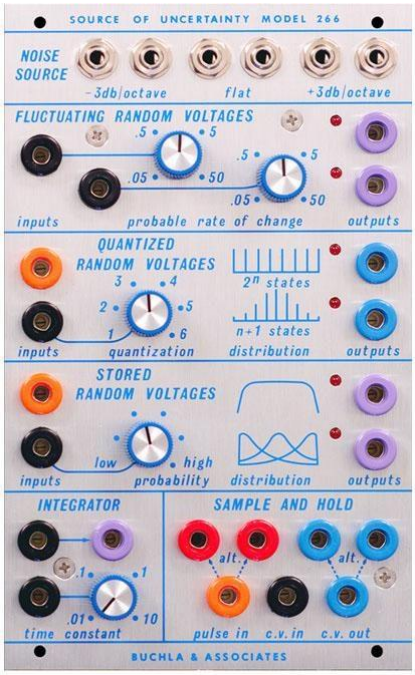


Figure 6: Buchla Model 266

The Stored Random Voltage section has a timing input and a probability CV input with a pair of outputs. The outputs each have different probabilities of outputting a random value. The top output produces signals that include an equal probability of output level. The probability distribution for the bottom output is determined by the probability control and CV input, favoring high, mid-range, or low values. This allows users to set the range of random values generated by the module and therefore the amount of change that output contributes to another module. The CV input allows the probabilistic distribution to be varied by an external source, or even another signal from within the 266 itself if patched that way.

The Quantized Random Voltage section also includes a timing input and CV input over the quantization level with a pair of related but separate outputs. These are marked 2^n and n+1, which correspond to their distribution of random voltages. If the control is at 1, then n = 1 and both the n+1 and 2^n produce two possible values. 1+1 and 2^1 both equal two. If n= 3, then n+1 outputs one of four possible values (3+1), while 2^n outputs eight possible values, 2^3 = 8. N+1 scales linearly, while 2^n scales exponentially. The n+1 output tends to favor the values in the center of the probability distribution, while the 2^n has equal weighting across the distribution. Both sections use what are called shift registers to generate their random, or pseudo-random, signals. I'll be going deeper into what a shift register is in the section dealing with hardware implementations of pseudo-randomness. The 266 also includes an integrator and sample and hold section. The sample and hold section creates a stepped voltage based on an input signal and timing signal and the integrator smooths out incoming voltages.

An unconfirmed urban legend surrounds several red-paneled Buchla modules, that said that the paint of these modules contained LSD. Rubbing your finger along them and licking it was thought to give users an extra boost of creativity. Whether or not this synthesizer folklore is accurate is up for debate, but it lends to the air of chaos that surrounds Don Buchla's legacy.

In the 1970s, CalArts had several Buchla systems in their studios, but few had access. A professor at the time, Serge Tcherepnin, wanted to create a more accessible brand of synthesizer that retained the experimental nature of Buchla synthesizers. Tcherepnin and students Rich

Gold and Randy Cohen eventually set out on a goal to make their own synthesizer. The first Serge systems, as they would be known, were designed and soldered at a kitchen table in Tcherepnin's home. Although it started small, word eventually got out to CalArts faculty and students, as well as other musicians. They set up a pseudo factory on campus, and for the fee of $700, you got all the parts necessary to build a six-panel system, all put together on-site, assembly line style.

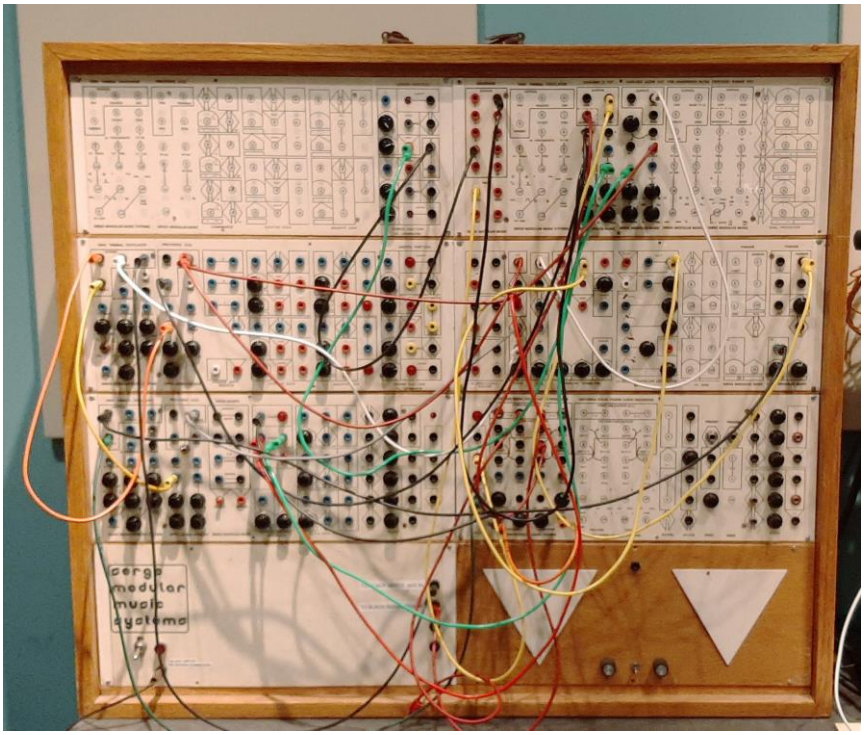


Figure 7: An early 'Paperface' Serge at CalArts

The Serge paradigm breaks down the parts of synthesis into their barest elements. All connections are made via banana cables, with a color coating of the jacks to indicate if the signal is AC, DC, or a timing pulse. Serge modules often use technical terms to explain musical concepts, which may be daunting to some musicians, but often the constituent elements are relatively simple. Serge's function generator, known as the Dual Universal Slope Generator (DUSG), is one of the best-recognized parts. On a Moog system, you have an ADSR envelope generator and on a Buchla, you have the Quad Function generator, both of which strictly generate envelopes, usually used for varying amplitude or timbre. The DUSG, on the other hand, can be used to generate envelopes, but can also be used as a slew generator to add

portamento, add delay to a pulse signal, follow the amplitude of an audio signal, cycles like an LFO or audio oscillator, or even be used as a simple non-resonant low pass filter.

Serge also pioneered the concept of patch programming. Patch programming uses one of the outputs from a module patched directly to one of the inputs on the same module in order to change the functionality. This could be making a function generator oscillate or turning a 16-step sequencer into a 13-step sequencer.

The two modules that are most used for generating randomness within a Serge system are the Noise Source and the Smooth and Stepped Generator, known as the SSG. The Noise Source outputs white and pink noise, plus another noise source called S/H Source that is inspired by the noisy triangle found on the Buchla 265. Some iterations of the Noise Source also include a stepped random output generated by either an external timing signal or with an onboard button.

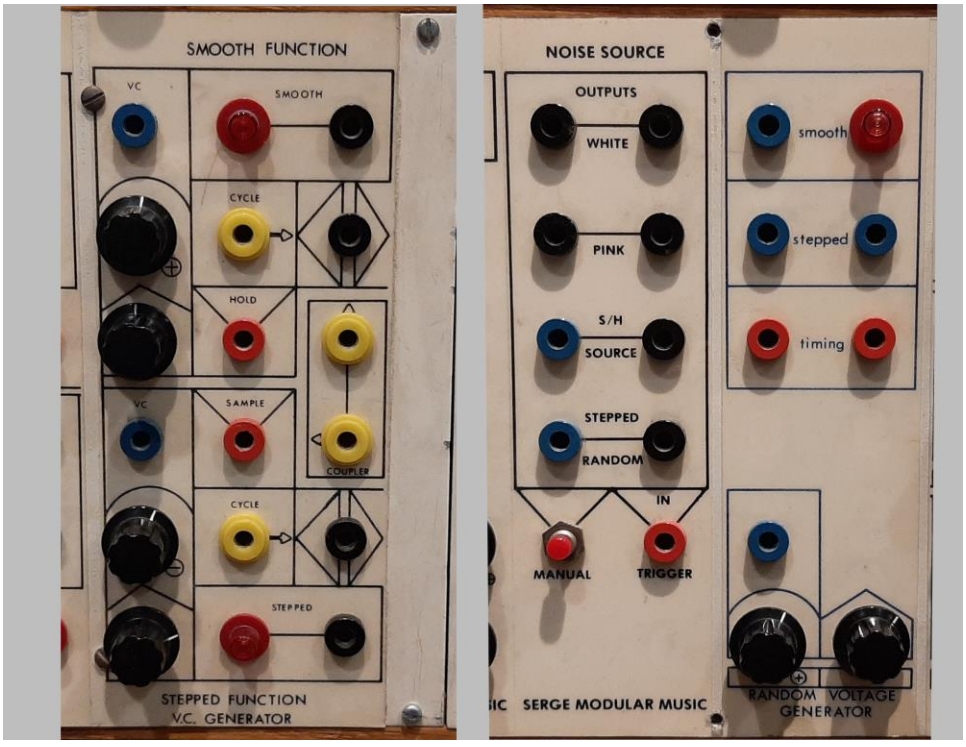


Figure 8: Paperface Serge Smooth and Stepped Generator and Noise Source

The SSG is broken down into three sections, Smooth, Stepped, and Coupler. The Smooth section smooths out incoming signals at a rate set by a control, turning stepped signals into smoothly varying signals. The Smooth section uses what is known as a track and hold. A track and hold is almost the opposite of a sample and hold. Instead of only allowing signals to pass to the output at clearly defined stepped intervals, a track and hold lets a signal present at its input to pass freely to the output. That is until the track and hold receives a timing signal, at which point it holds the level of the signal passing through it and does not let any signal through until the timing signal goes inactive.

The Stepped section also includes a slew generator to smooth out signals but uses a timing signal to generate a stepped signal based on an input signal using a sample and hold. Both sections include cycle gate outputs, which, when patched into their respective inputs, allow the sections to oscillate and be used for modulation or an audio source. The third section is the coupler output which compares the signals present at the smooth and stepped side and outputs a high signal when the stepped side is at a higher voltage level than the smooth side.

An SSG cannot be used on its own as a source of randomness, but it is a great way to expand the functionality of the Noise Source. The Serge Fans website includes the following suggestion on how to patch up a complex source of random modulation: Patch the S/H noise signal into the in input on the stepped side, patch the coupler out to the timing input on the stepped side and the in input on the smooth side. This creates stepped random voltages, smooth random voltages, and random timing signals. Varying the controls of the smooth and stepped sides affects the amplitude and timing of the random signals. There also exists a module that does that already, known as the Random Voltage Generator, which outputs stepped, smooth, and timing signals. If you opened your case and took a look at the circuitry behind the front panel, you would notice that the RVG actually uses an SSG PCB, prewiring that common patch-programmed configuration.

The rise of low-cost digital synthesizers in the 1980s led to many musicians casting off their analog synthesizers in favor of the newer, more reliable all-in-one keyboard synthesizers. Analog was seen as a thing of the past, a curiosity or steppingstone to the next iteration of technology. However, in the mid-1990s, a German company named Doepfer had the idea of a

new modular format, which came to be known as Eurorack. Eurorack refers to a standardization that defines the size of the front panels, 3U or three rack units high, 128.5mm or just over five inches. It also specifies the power requirements, both in terms of what voltages the power supply generates and the connections from the power supply to the individual modules. The connections between the modules are made on 3.5mm/ 1/8th inch monophonic cables, similar to the ones seen on headphones, but only carrying a single signal instead of a dual/stereo one.

Since the mid-90s, Eurorack has become the dominant modular format, with well over 100 companies of various sizes making and selling their own synthesizer modules. With the wide range of available modules, there is bound to be variation on existing paradigms and new ways of synthesizing and modifying sounds.

Two modules that take direct or indirect inspiration from the Buchla 265/266 Sources of Uncertainty are the Doepfer A-149-1 and the Wogglebug, originally made by Wiard with versions from Make Noise, Erica Synths, After Later Audio, and others. The Doepfer A-149-1 can be seen as a direct link to the Buchla 266, stripping it down to its Stored and Quantized random voltages. It adds controls to vary the amplitude of incoming CV for the N and distribution for the Quantized and Stored random voltages, but otherwise is near identical, even retaining the same panel graphics seen on the original Buchla module. It also has an optional expander, which adds eight gate outputs based on the state of signals within the Quantized Random Voltage section.

Figure 9: Doepfer A-149

The Wogglebug was originally designed by Grant Richter of Wiard Synthesizers. It could be seen as a continuation of the ideas laid out in the 265 Source of Uncertainty. It includes the fluctuating smooth and stored stepped CV outputs, complete with correlation control, but adds a third unique smoothed output, known as the woggle CV. This CV signal follows the smooth random voltage and when it catches up to it, it bounces around that CV level with decaying sinusoidal wiggles and woggles. The Wogglebug features an internal clock and clock input to synchronize all outputs to a common timing signal. In addition to providing modulation, it also includes audio-rate oscillators connected to the random CV with outputs for the Smooth VCO, Woggle VCO, and the result of ring modulating the two VCOs against each other.

Figure 10: Make Noise Wogglebug

In 2012, Music Thing Modular released the Turing Machine as a DIY project. While it shares the name Turing from the pioneering researcher in computer science, Alan Turing, it is not a true Turing Machine in the way Turing described it. Rather, it's a random sequencer that allows users to guide the direction of the randomness without being able to control exactly what notes are played. One unique feature that has attracted many users is the large knob that adorns the front panel, which allows you to interact with the randomness. By varying the position of the knob, you can allow the module to introduce more or less randomness into the sequence, even to the point where no new information is passed into the data buffer, and it repeats indefinitely. This could be seen as harkening back to the correlation control on the 265 but implemented differently. The length of the sequence is user definable between 16 and 2 steps long. Several expanders are available, which add additional modulation outputs, timing signals based on the state of the random sequence, or a simple matrix mixer hooked up to the output of the random voltages.

Figure 11: Music Thing Modular Turing Machine

Marbles from Mutable Instruments is a powerhouse of random timing and modulation signals. It has three main sections; T, X, and Déjà vu. The T section controls the timing of the module, generating three timing signals with variable rate and jitter, or the amount of randomness in the clock timing. T2 is the steady clock signal set by the rate knob, while T1 and T3 are controlled by a Bias knob. Bias has three modes: coin toss, random ratio, and kick/snare; in all of these modes, the Bias controls the likelihood of the chance operation to affect T1 or T3. The X section is a collection of modulation signals with variable range, probability distribution, distribution bias, smoothness or steppiness, and quantization to a musical scale. Both the T and X sections are connected to the Déjà vu section, which allows users to recycle the random data within the module for looping behavior.

Figure 12: Mutable Instruments Marbles

Nonlinear Circuits is an Australian-based modular synthesizer company run by Andrew Fitch. Fitch has had a prolific career putting out strange and esoteric modules with personality baked into their front panel and PCB silkscreens. Modules have names such as Poultry in Motion, Bindubba, and Brain Custard, and while some fall into standard synthesis categories, many exist outside of traditional behavior seen in synthesis. Fitch's output has been prolific, releasing a new design every month or two and running a periodic DIY synthesis workshop in Perth. Much of the NLC output comes in the form of unique chaotic and random modules which have been a large inspiration in terms of my own designs. We'll explore the direct inspiration from his work to mine in the hardware chapter.

Figure 13: Nonlinear Circuits Brain Custard

Chaos

In the beginning, there was nothing. At least that's how the story goes for many creation myths. Chaos was seen as the prehistory of the world, a formless mass of nothingness that predated humans or even gods. In Ovid's Metamorphoses, the chapter on The Creation begins:

> Before the ocean and the earth appeared—
> before the skies had overspread them all—
> the face of Nature in a vast expanse
> was naught but Chaos uniformly waste.
> It was a rude and undeveloped mass,
> that nothing made except a ponderous weight;
> and all discordant elements confused,
> were there congested in a shapeless heap.

Some god force, or nature, personified with he/him pronouns, took this formless state of the universe and gave it direction and a recognizable form. This conception of chaos seemingly influenced the Christian creation myth as well, as before the earth existed, there was a formless void or abyss. The disorder of emptiness stands in contrast to the divine orderliness of creation.

Chaos is often used as shorthand for disorder and confusion, something antithetical to the rigors of science and mathematics. However, in the middle of the 20th century, researchers began to apply the term to an emerging realm of thought that touched the realms of physics, mathematics, and even the life sciences. Chaos theory can broadly be seen as studying the seemingly random or unpredictable using deterministic rules. Its evangelists came from a diverse range of fields but were all drawn in by the allure of the unknown.

One of the major researchers in the field was a man named Edward Lorenz, a mathematician and meteorologist. In the 1960s, he was working at MIT, trying to figure out a way to predict weather patterns. To many meteorologists, the idea of forecasting the weather was not a realm of serious consideration, it was pure fantasy and conjuncture. The idea of a system that could predict weather patterns had been a tantalizing pipe dream, something not worth considering or spending time pondering. However, Lorenz was not one of these nay-saying scientists.

Lorenz had constructed a crude weather simulation using a Royal McBee LGP-30 computer. It was a primitive thing, able to perform about 60 calculations a second, but it was what was available at the time. He sketched out 12 rules to govern his hypothetical weather system that determined the relationship between temperature and pressure, and then pressure and windspeed. Every few minutes, the machine would print out a row of numbers, an incredibly abstracted version of a day's weather and winds. If you could decode this numeric code, you could see the behavior of weather emerge, but never quite the same way twice.

Lorenz would eventually change the way the machine represented its findings. He picked a variable and represented it by the letter 'a' with a certain number of spaces on either side. The machine would print a series of 'a's to chart the changes of that parameter over time, moving

back and forth across the printout. Up and down the 'a's marched, plotting information that corresponded to some variable such as wind direction. There was an order to the seeming disorder of it, recognizable patterns that didn't even quite repeat.

In 1961, he wanted to get a better look at a particular behavior, so instead of starting the simulation over from scratch, he started partway through. He typed the initial conditions of the system by hand, and let the machine run unattended while he did something else in the office. When he returned, he had a shocking discovery. The machine had not repeated the behavior he saw beforehand but had an entirely new printout that deviated from the first. Slowly at first, but as it progressed, the discrepancy grew and grew. He first thought something had malfunctioned with his computer, as it often did, but he realized where the discrepancy came from.

The printout showing the numbers only printed the first three decimal places, but the computer stored six decimal places in its memory. He had typed 0.506 instead of the 0.506127 stored in memory. This tiny deviation and its resulting behavior became known as one of the fundamental aspects of chaos theory: sensitive dependence on initial conditions. The machine was governed by deterministic rules, but small changes to the parameters led to great changes as the system progressed. Lorenz saw a delicate order in his unpredictability, a complex system that was nevertheless governed by specific laws.

Chaos theory is sometimes mentioned together with the Butterfly Effect. The butterfly effect uses the metaphor that the simple flap of a butterfly's wings in a remote location then disturbs the weather and causes a tornado halfway across the world weeks later. This is the best-known analogy of sensitive dependence on initial conditions.

Lorenz decided to keep pursuing complexity that arises out of a simplistic set of rules, and eventually settled on a system of just three nonlinear equations. Rather than linear equations that can be solved and plotted as a straight line on a graph, nonlinear equations are not proportional and curved when graphed, if they can be solved at all. Only nonlinear equations can be chaotic, but not all nonlinear equations are chaotic. Lorenz took a series of equations that described convection, or the rising of a hot liquid or gas, and stripped them down to their barest

elements so they no longer applied to real world conditions of convection. He kept the nonlinearity, of course, but threw out much of the other elements that made up the equations.

Plotting the three equations in three-dimensional space creates what is called the Lorenz Attractor. Fittingly, it almost appears to trace the shape of a butterfly's wings as it rotates around a bounded space never quite the same way twice. Lorenz documented his findings in a paper entitled "Deterministic Nonperiodic Flow," which researchers would cite with excitement for years to come. The attractor was chaotic but not unstable. Noise or other perturbations would not throw it off its long-term trajectory. The system was stable in the long term and when viewed holistically, but any one point or on a microscale was unpredictable.

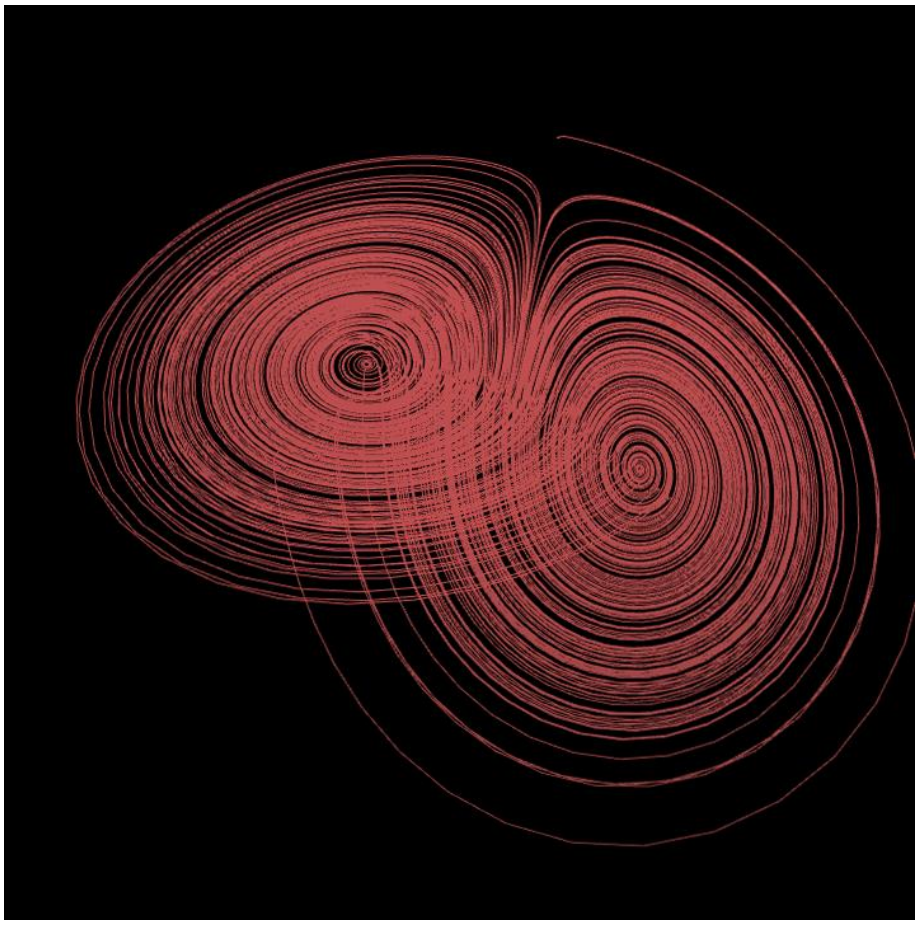


Figure 14: Lorenz Attractor

A surprising place where chaos shows up is in an equation that models population growth, known as the Logistic Map. Using a simple equation, biologists could map the rate of change in a population of animals over time. The simplest way to calculate the change in population is to take this year's population, symbolized by the variable 'n' in this case, and then multiply by a growth rate, 'r'. Let's say you start with a population of deer, n = 10, and a growth rate of r=2. x(next) = n * r, or next year's population is equal to this year's population times the growth rate. Then you repeat, so x is equal to the previous year's x(next). The first year you would have 10 deer, in year two you would have 20 deer, in year three, 40 deer, etc. Obviously, this runs away quickly, doubling in size every year without any consideration for death by disease or predation. The feedback loop is infinite, last year's population becomes next year's at a rate with no decrease in population.

You need to reign in the numbers and account for mortality, and a simple change in the equation does that. The new equation is x(next) = r*x(1-x), with the addition of (1-x) setting a boundary to the growth. As x rises, (1-x) falls. With a growth rate under 1, the population declines to extinction as it decreases every year. With an r growth rate between 1 and 3, the population stabilizes, no runaway growth or decline. However, once r goes above 3, the population begins to oscillate between two points, reaching a different kind of stability. This is known as period double bifurcations, meaning it takes twice as long to repeat a value. As you increase further, the population splits again into a four-year cycle, then eight, and then 16. Eventually, when r is greater than r=3.57, the doublings give way to chaos. However, the chaos does not exist at all r values over 3.57, as r approaches 3.83, there are three-period cycles, then six, then 12, and then back to chaos again. In fact, hidden in the logistic map are periods of every length. Hidden in the chaotic nonperiodic oscillations of the logistic map are islands of order, small windows where the unpredictable gives way to the stable.

It should be noted that this way of calculating growth is a different type of equation than the Lorenz attractor. The Lorenz attractor and equations like it are differential equations that happen continuously over time, while the Logistic Map is a difference equation, with clearly defined intervals. Like the difference between a smoothly moving second hand on a watch and the minute hand that jerks forward a set interval every full rotation of the second hand. This

works nicely when charting population growth in animals, as many animals have a distinct breeding season and populations can be charted in yearly intervals.

In 1998, Dan Slater published a chapter in the Computer Music Journal entitled Chaotic Sound Synthesis. This was one of the first instances of bringing the worlds of chaos theory and modular synthesis in a formalized manner. Many early researchers in chaos theory used analog computers in order to study chaos. In contrast to the much more popular digital computer, analog computers have the capability to calculate continuous signals, whereas digital computers store values as discrete and set binary intervals. Mathematical equations could be constructed from operational amplifiers and common components such as resistors and capacitors. However, due to the nature of physical parts, there are potential issues in small variations in the characteristics of the materials making the parts. This is part of the reason that digital computers have become the ubiquitous machine of the end of the 20th century into the 21st century.

One thing that attracted the attention of the authors of Chaotic Sound Synthesis is the ability to interface many analog computers with hardware modular synthesizers. Many computers output analog signals in the +/-10V range and are compatible with Buchla, Serge, or Moog modular synthesizers. By programming analog computers, you can get any number of behaviors, from simply scaling a voltage to complex chaotic equations not available in modular synthesizers at the time.

One application was the construction of nonlinear filters that produce chaotic behavior. These filters are sensitive to an incoming signal's amplitude and waveform, not just its frequency. Additionally, the nonlinear chaotic filters may produce frequencies not present in the input signal. One chaotic system like a state variable filter is the Ueda attractor. The Ueda attractor can be constructed out of a modified state variable filter where one of the inverting stages is swapped for a circuit that produces an x^3 function.

While an analog computer can be included with a modular synthesizer in order to calculate chaotic behavior, common synthesis elements in a modular synthesizer can also be used to generate chaotic behavior. One example of this is an analog implementation of the

logistic map discussed above. The author of the paper reordered the logistic map from x(next) = r*x(1-x) to x(n+1) = k(xn - x^2n).

The implementation of the logistic map was created using several Moog Modular modules, the 902 VCA and the 928 Sample and Hold. The 902 VCA is a voltage-controlled amplifier with both linear and exponential response curves. A voltage-controlled amplifier can be compared to automated volume control. Instead of manually turning up and down the volume for a signal, a second signal can be used to automate the amplitude or volume level of that signal. The 902 includes two inputs that are differential, where the signal present at the lower jack is subtracted from the signal at the upper jack. When only using one input signal, the two inputs correspond to inverting and noninverting behavior. Two 902s are used to generate the x^2 term, while a third generates the k(x-x^2) term. The k or chaos level is controlled by the CV input on the third VCA. All the VCAs are set to respond linearly to incoming voltages.

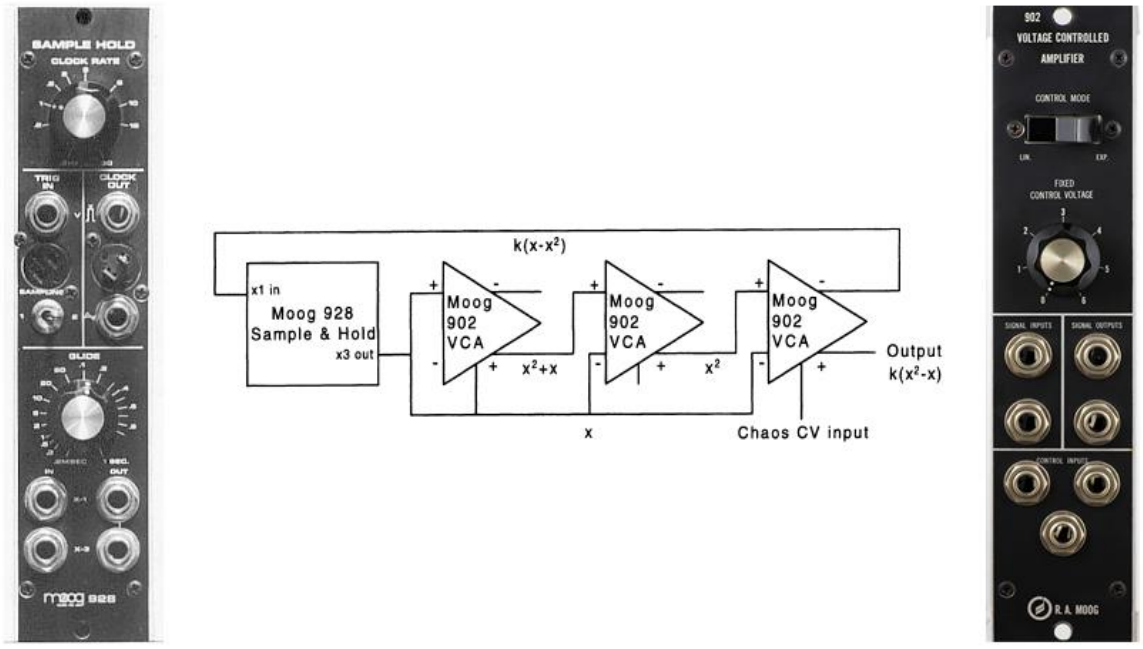


Figure 15: Method for Generating a Logistic Map with Moog Modular Synthesizers

The inverted output from the third VCA is patched to the Moog 928 Sample and Hold into the x1 input. The 928 provides a single-sample delay with a sampling rate set by the internal clock oscillator. The x3 amplified output from the 928 is fed back into the VCAs. Adjusting the slewing on the 928 alters the discretely time-stepped nature of the logistic map where instead of jumping from one year or time interval to the next, it bleeds the intervals together. It's not a one-to-one recreation of the logistic map, but it probably makes for some interesting results.

With these considerations in mind, Slater proposes a hypothetical chaos module in the Buchla 200 series format. The module includes a Poincaré control voltage processor, Ueda attractor audio processor, and a pair of Logistic equation circuits, one for audio and one for control voltages. The Ueda audio processor blurs the line between chaos module, audio filter, and quadrature oscillator. A quadrature oscillator is an oscillator that generates two or more outputs that sit out of phase with each other, usually 90 degrees. This is commonly implemented with sine and cosine outputs. The attractor includes CV inputs to modulate the frequency, Q (a shorthand for resonance, but labeled here as damping), and nonlinearity with a control for exponent that ranges from 1-4. It includes lowpass, highpass, bandpass, and band-reject filter outputs, with the lowpass and bandpass outputs sitting 90 degrees out of phase with each other, acting as quadrature outputs.

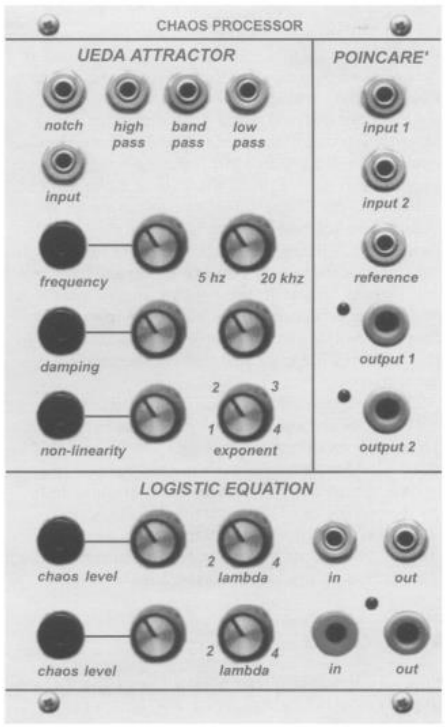


Figure 16: Dan Slater's Hypothetical Chaos Module in the Buchla Format

The Poincaré map takes two audio signals, by default the quadrature out of phase bandpass and lowpass filters from the Ueda filter and generates a short pulse when the audio signal transitions from negative to positive. This drives two sample and hold circuits with the reference voltage coming from the input signal present at the Ueda filter. If only using a single chaotic input signal, the Slater suggests an allpass filter, delay network, or dome filter in order to generate a second chaotic signal, derived from the first. The Poincaré map would in this case be used to generate the X and Y positions of a chaotic signal as they relate to phase space, which could be easily viewed if used with an oscilloscope in XY mode. When running an oscilloscope in XY mode, the oscilloscope takes two of its inputs and maps them onto a 2D space, with one input corresponding to the horizontal axis and the other the vertical axis.

The pair of Logistic Map Circuits act as nonlinear waveshapers for both audio and control voltage signals. When passing audio signals to the circuits, it generates outputs varying from pulse waves to different subharmonic and chaotic noise signals. It does the same thing to control voltages, but on a sub-audio level. The outputs would be stepped and not smooth, because, as we've seen, the Logistic map operates in discrete time-stepped intervals.

Dan Slater laments the fact that chaos theory, as a formalized concept, had not been fully explored during the major heyday of modular synthesis development from the mid-1960s to the early 1980s. While there is some overlap in the timelines, the idea of bringing the two together was seemingly not on the minds of the early developers of modular synthesis. The paper "Chaotic Sound Synthesis" was published in 1998, right around the time that Doepfer was introducing their iconic Doepfer A-100 series Eurorack modular synthesis format. As I've mentioned, the proliferation of this ubiquitous style opened the marketplace for radical experimentation in synthesis techniques, including chaos. In the next chapter, I detail my explorations in creating modular synthesizers that occupy the realm of chaos and pseudo-randomness.

Chapter 3 Hardware

I first released a hardware synthesizer in the spring of 2018, but that was after several years of planning, research, trial, and plenty of error. The first module released was the Dual Digital Shift Register, a simple shift register-based module that generates pseudo-random signals after beginning work in 2016. In 2019, I released Illyana, a two-channel Boolean logic module. Over the next few years, I went on to release the R2Rawr, lo-fi digital to analog converter; Curtail, an audio to video level shifter; and the Quad Mute, a four-channel passive mute switch. I also developed several modules that have not been released but have stayed in the prototyping phase until I deem them ready for release.

When I enrolled in CalArts in 2021, I was in the middle of development of one of those modules that I had been working on for almost two years at that point, and just figured out I needed to do a massive revision to the design to make it both more stable *and* more chaotic. I decided I needed a break, and my mind wandered to the design that would become the Cascading Register.

The Cascading Register generates pseudo-random voltages and gates using what is called a shift register. It was inspired by several different kinds of shift register implementations found in both within modular synthesis and outside of modular synthesis. These are digital shift registers, analog shift registers, linear feedback shift registers (LFSR), and runglers (a specialized esoteric shift register implementation). It grew and evolved from the first Eurorack module I designed to completion, the Dual Digital Shift Register (DDSR).

Figure 17: omiindustriies Cascading Register

A shift register is a series of simple data storage elements, known as flip-flops, connected sequentially, which share a common clock or timing information. Typically, shift registers are digital, meaning that the information they store occupies two states, off or on, 0 or 1, high or low. A flip-flop is a simple data storage device that stores the state of an external data source, typically binary on/off information. I'll be talking about a serial-in-parallel-out shift register, but there are several other implementations of the basic shift register concept. For simplicity's sake, I'll just call the SIPO shift register a shift register.

A shift register has a data input connected to the first flip-flop and a clock input connected to every flip-flop in the chain. On the rising edge of a clock pulse, the shift register looks at the data input and moves that state into the first stage of the shift register. If the data input is high, it moves a high state into the first stage, and if it's low it moves a low state into the first stage. Concurrently, whatever data was in the first flip-flop gets shifted to the second stage,

whatever was in the second stage moves to the third, and so on. When a bit of information reaches the final stage of the shift register, it is shifted out and lost. This is the basic function of a digital shift register.

A rungler is a particular implementation of a shift register, designed and invented by Dutch engineer Rob Hordijk. He wanted to make a module that created unpredictable signals but was disillusioned with just simple noise generators. In the 1980s, he experimented with CMOS (complementary metal-oxide-semiconductor) circuits, a family of digital logic circuits. A rungler circuit is a particular implementation of a shift register, coupled with a pair of oscillators. One oscillator provides the clock information and the other is the data source. The last three bits of the eight-bit shift register are fed into a DAC circuit to create a waveform that Hordijk describes as a "Stepped Havoc Wave." Hordijk was inspired by Dutch composer Jan Boerman, who described all sound as appearing on the continuum between a pure sine wave that contains only the fundamental frequency and no harmonics, and pure noise, which contains every frequency, amplitude, and phase relationship of sound; even going so far as to say that all sound is contained in the static hiss of pure uncorrelated noise.

The rungler's stepped havoc wave is routed to modulate the frequency of both the clock and data oscillators to create a feedback loop. The clock and data oscillators also include triangle wave outputs, which in addition to the stepped havoc wave, is routed to modulate the frequency of the opposing oscillator.

The most famous implementation of the rungler is in the Benjolin, a DIY workshop instrument Hordijk designed to be built in workshops. It also appears in his Hordijk modular system, and the Blippoo Box. The Benjolin design has been adapted by several other Eurorack companies, including After Later Audio and Epoch Modular, as well as standalone synthesizers by companies such as Macumbista. One element missing from some of the adaptations, but not all, is the ability to substitute external signals for the internal clock and data oscillators.

Another aspect that informed the design of the Cascading Register came from the world of cryptography, a linear feedback shift register, known as an LFSR. An LFSR is a common way to generate pseudorandom numbers in both hardware and software. Their simplicity means

they're easy to implement and can be constructed without using thermal noise used for analog noise to generate randomness. They are deterministic and given the same initial state and feedback configuration will generate the same string of pseudo-random values. An LFSR takes two of more of the stages of the shift register, known as taps in this case, and combines them through logical operations before routing the resulting signal back to the data input, creating a linear feedback path of binary information. Usually, the logical operation is a Boolean XOR logical operation. Boolean logic takes binary signals and compares them against each other, turning the output of the logic on or off based on the states of the inputs. The output of an XOR logic gate is on if one input or the other are on, and off if both or neither are on. Put another way, the output is on if the inputs do not match, and off if the state of the inputs do match.

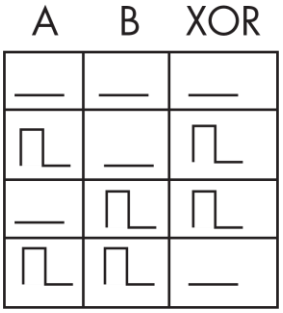


Figure 18: XOR Truth Table

A maximal length shift register configuration refers to the configuration of taps that create the longest string of values and vary based on the length of the shift register. The length of the sequence can be calculated as 2^n-1, or two to the length of the shift register, minus one. It's minus one because if all the stages of the shift register are off, no data could be recycled, creating an invalid state. So, in an eight-bit shift register, the maximal length is 255 steps. In a 32-bit shift register, however, the length is 4,294,967,295 steps long. If you calculated the numbers sixty times a second, it would take over two years before it repeated itself.

In an eight-bit shift register, the configuration of taps that creates the longest string of pseudo-random values takes the eighth, sixth, fifth, and third stages of the shift register and XORs them together. The Cascading Register is not a maximal length shift register, as only the eighth, sixth, and fifth stages are used as feedback taps. One could patch up a maximal length LFSR by manually patching the third stage into the external data input.

In addition to the external data input and the three feedback taps that supply data, a button on the bottom of the module allows users to manually enter data into the data stream. This button, labeled Seed, gives a direct manual input to influence the course of the shift register. However, it is in the XOR path, meaning that pushing the button may or may not enter a high state into the first stage of the shift register, depending on the other parts of the data path. In order, the eighth stage is XOR'd with the sixth, that is XOR'd with the fifth, that is XOR'd with the seed button, and finally that resulting signal is XOR'd with the external data input.

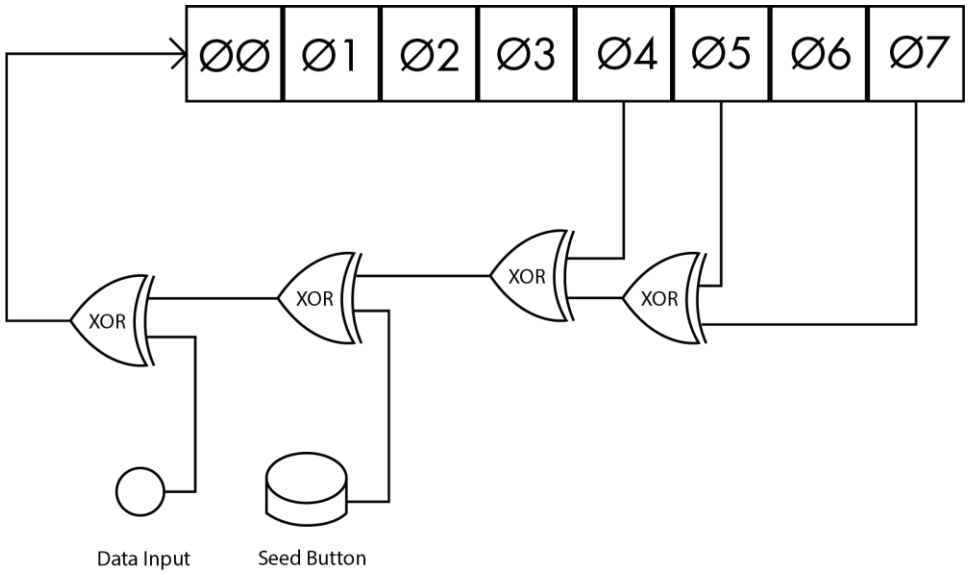


Figure 19: Data Feedback Path with XOR Logic Gates

Another kind of shift register that inspired the Cascading Register is an analog shift register. An analog shift register is essentially a series of sample and hold function blocks. An analog shift register marries the function of a sample and hold with the idea of a shift register. It is a series of sample and hold function blocks. Instead of losing the original sampled signal, on every clock pulse, that voltage level passes from one sample and hold to the next, creating what is known as an arabesque musical form, where a melody passes down a line of voices. It was first implemented by Barry Schrader with help from Dr. Fukushi Kawakami, known to his friends as "Fortune." They collaborated on a series of modules to extend the functionality of Schrader's Buchla synthesizers, known as the Fortune Modules. Kawakami made four modules for Schrader, Control Voltage Smoother #1, Control Voltage Smoother #2, Control Voltage Matrix

Gate, and the Analogue Shift Register. The first commercially available version of the Analog Shift Register was made by Serge Tcherepnin and included in Serge Modular System.

The Cascading Register contains no sample and holds but was inspired by the concept of an analog shift register. The three control voltage outputs available on the Cascading Register come from the state of the gate signals that pass through the eight-bit shift register. CV1 is determined by the first four gates, CV2 comes from the middle four gates, and the last four gates inform CV3. Each set of four gate signals runs into a simple digital-to-analog converter, creating a stepped CV signal determined by the state of the gates. As I mentioned before, the Cascading Register contains no sample and holds, but the core idea of a set of information passing down a series of outputs comes from the realm of the analog shift register.

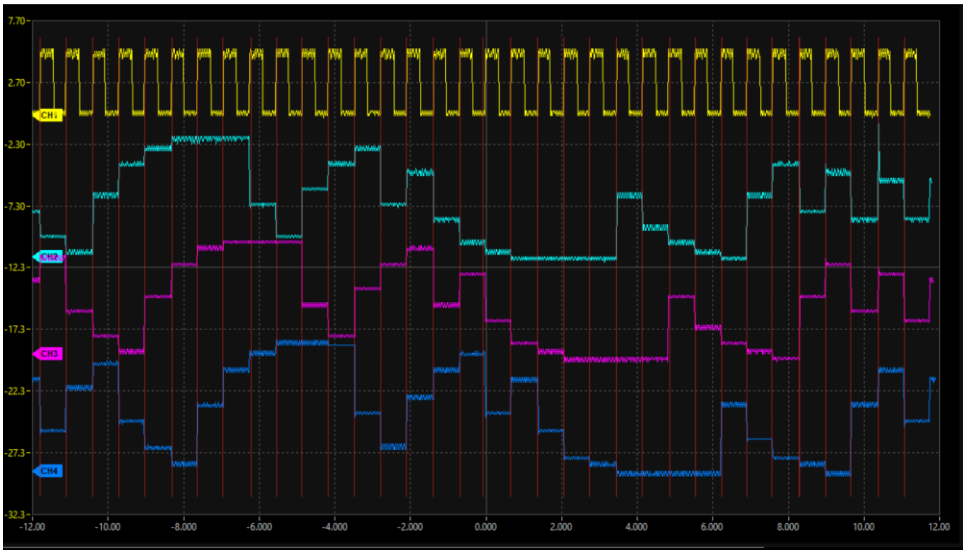


Figure 20: Oscilloscope showing Clock and CV Signals 1-3

The Cascading Register has three stepped analog control voltage (CV) outputs, whose output level is tied directly to the three white knobs on the panel. CV1 and CV2 pass through attenuverters before their outputs, and CV3 has an associated attenuator. An attenuator can be thought of as analogous to a standard volume control on a stereo. All the way counterclockwise, the signal level is off, and as you turn clockwise, the voltage level increases until it reaches its maximum level. Attenuverters, on the other hand, are off when the knob is in the center of its range. Turning clockwise increases the voltage level in the positive domain, while turning

counterclockwise inverts and increases the voltage level in the negative domain. For example, if a user patches CV1 or CV2 to the pitch input of a voltage-controlled oscillator, turning clockwise increases the pitch of the oscillator, while turning counterclockwise decreases the pitch of the oscillator.

In addition to the three analog CV outputs, the Cascading Register includes individual outputs for each stage of the shift register. The zero-indexed labeling goes from ØØ to Ø7 arranged vertically from top to bottom. The outputs are all binary gate signals, typically used in modular synthesis as timing signals. Constructing a simple pseudo-random rhythm is a matter of patching the first output to a kick sound and a snare from a lower gate output, which will generate a call-and-response rhythm offset by the selection of the gate output. While gate signals are most commonly used as timing signals, they also work as modulation signals. The sharp onset and near-immediate decay makes them excellent as modulation sources for percussive sounds, creating a defined accent in timbre or volume when the gate is active.

The Cascading Register also comes equipped with an internal voltage-controlled clock oscillator, normalized to the clock input. Normalized in this case means an internal connection from an output to an input on the same module, that is interrupted when you patch a cable into the input jack. The clock oscillator is a square wave oscillator that ranges from sub-audio to low-audio frequencies. The output of CV3 is also normalized to the CV input of the clock oscillator, creating a feedback loop that generates a clock signal that slows down and speeds up depending on the output voltage from CV3. This normalized connection between CV3 and the CV input on the clock oscillator is the reason that CV3 used an attenuator instead of an attenuverter. It's much easier to zero out the modulation of an attenuator than an attenuverter, as the attenuator's off position is a function of the mechanical connection on the potentiometer. An attenuverter's off position is close to the middle of the range of the potentiometer, but finding the exact off position is a difficult art.

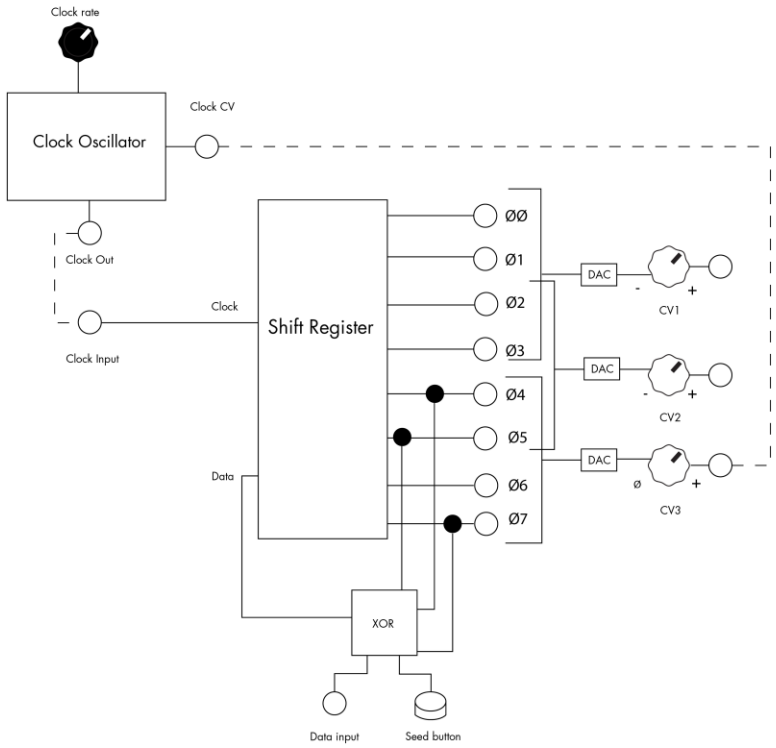


Figure 21: Block Diagram of the Cascading Register

While the Cascading Register features an internal clock oscillator, it also includes an input for an external clock source. This accepts signals from sub-audio to audio, up into the range used in video synthesis, well above the range of human hearing, entering the realm of the scanline speed on a television. While this is not a necessary addition for audio synthesis purposes, and increased the overall cost of the module, it allows the Cascading Register to fit into the world of video synthesis, becoming a complex noise generator and rough down sampler for video.

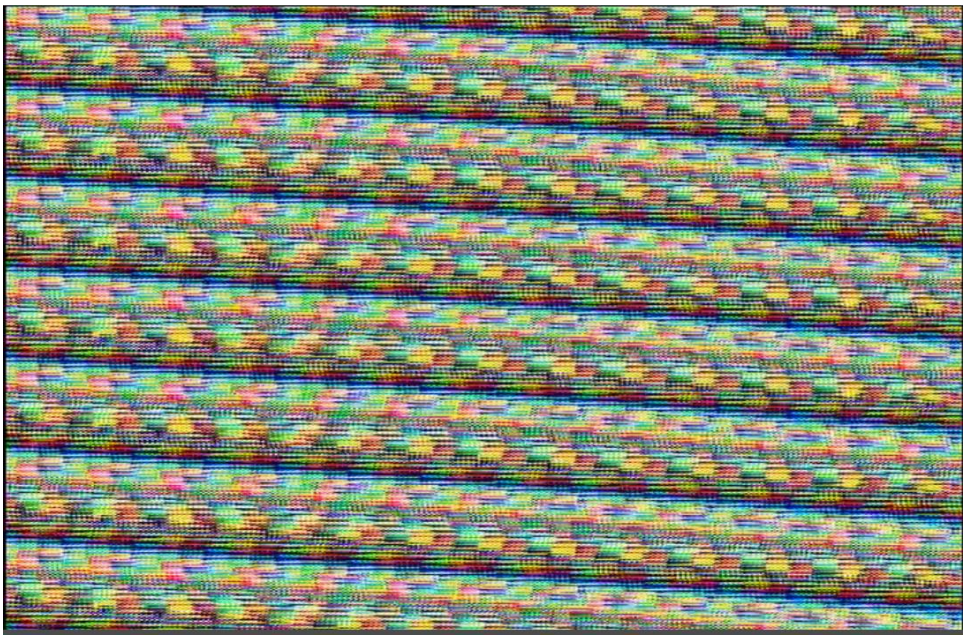


Figure 22: Video Noise Generated by the Cascading Register

So far, I've described the Cascading Register in terms of the front panel, but I turn now to the interior design of the module, examining the schematic and design of the printed circuit board.

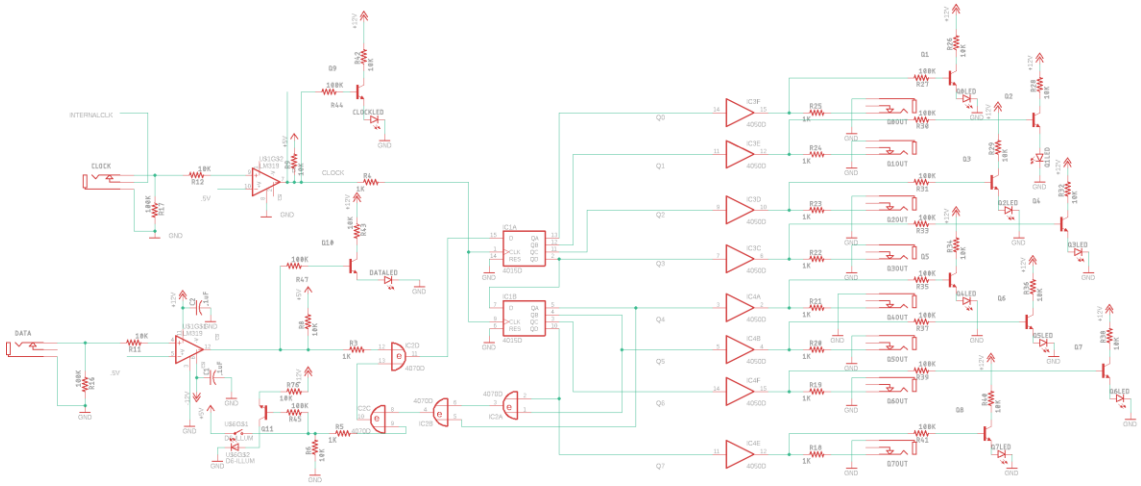


Figure 23: Schematic of the Shift Register, Clock Input, Data Inputs, and Buffer Sections of the Cascading Register

This first sheet shows the main shift register implementation. I used a CD4015 dual four-stage shift register connected together with a common clock and the four data bit (QD) out from the A shift register into the B shift register. The clock and data inputs include an LM319 comparator with a reference voltage of approximately 0.5V. The shift register outputs binary signals, with the fifth, sixth, and eighth XOR'd together as seen in this collection of logic gates. In addition, these binary signals run into a pair of CD4050 hex buffers before reaching the outputs, in order to ensure a steady +5V signal level. I used transistors in order to buffer the signals going into the LEDs which allow signal to flow through the LEDs to ground when they receive an active signal.

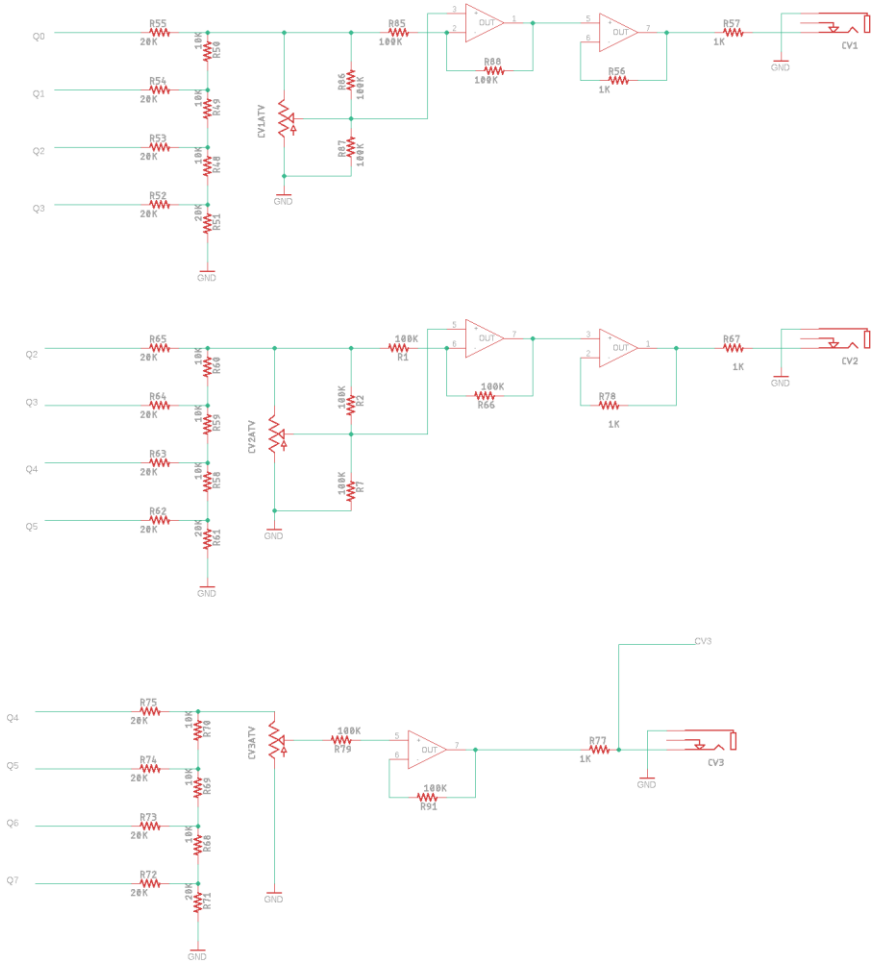


Figure 24: Schematic of the Digital to Analog Converters and CV outputs

Next, we have the three digital to analog converter circuits. I used R2R digital to analog converter circuits which involve two resistor values, R and 2R, or 10K and 2 x 10K or 20K. CV1 and CV2 pass into these attenuverter circuits. I used a pair of 100K resistors connected to the potentiometers, which gives a slight curve to the linear response of the potentiometer, giving more of a center 0V spot to allow users to zero out the modulation. CV3 simple runs through the potentiometer connected to ground. The LM6172 ICs require a 1K resistor in the feedback path when used as a buffer.

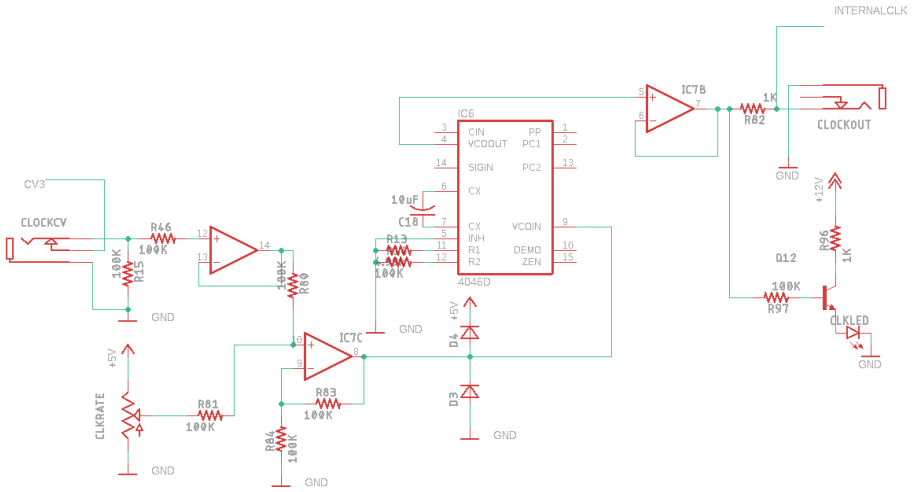


Figure 25: Schematic of the Clock Oscillator

The clock oscillator uses a CD4046 Phase-Locked Loop configured as a Voltage Controlled Oscillator. C18, R13, and R14 determine the frequency range of the oscillator. The voltage present at pin9 VCOIN controls the frequency of the oscillator within this frequency range. The clock rate potentiometer and external CV input are summed in a non-inverting op-amp before passing by a pair of diodes that ensures the signal running into the IC does not go above +5V or below 0V.

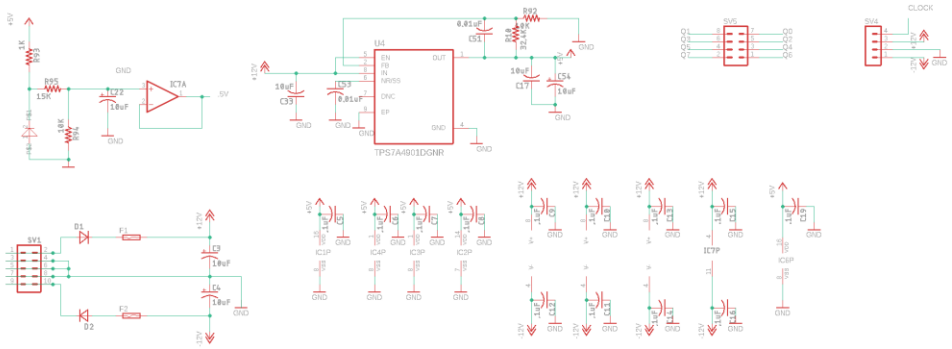


Figure 26: Schematic of the Power Sections

The final part of the schematic is the power section. This shows the power connected to each of the ICs with 0.1uF capacitors connected to ground to ensure clean power is supplied. The Eurorack power connector is SV1 which supplies the +12V, -12V, and ground signals. +12V and –12V pass through D1 and D2 which ensure that if the pin header is plugged in backward that the module isn't damaged. F1 and F2 are ferrite beads which add filtering to the power. C3 and C4 are polarized capacitors that also filter the power rails. The TPS7A4901DGNR is a low noise and low dropout voltage regulator that takes the +12V power rail and converts it to a +5V signal. SV4 and SV5 are pin headers connected to the shift register gates, clock signal (either internal or external), +12V, -12V, and ground. These are for an expander board that would allow the Cascading Register to connect to other modules that expand the functionality. At present, this is not included in the Cascading Register, but there are plans for future modules.

My next goal for developing a Eurorack module was to create a chaotic circuit. I had been working on a self-designed chaotic signal generator as part of a multi-function module I mentioned I needed a break from, but I wanted to simplify and create a single purpose module instead of one with four function blocks. I primarily looked at the work of J.C. Sprott, Ian Fritz, and Andrew Fitch during those planning stages. But how does one go about creating chaos in circuitry?

Constructing a chaotic circuit involves creating an analog computer that solves the chaotic differential equation. This is done with operational amplifier (op-amp) integrators, as an

integrator's output voltage is the input voltage's negative integral. An op-amp integrator consists of an op-amp whose positive input is grounded with a feedback path from the output to the inverting input with a capacitor between the output and the input. A resistor comes before the feedback loop. The ratio of the size or capacitance of the capacitor and the resistance of the resistor determines the rate of change of the output voltage. The voltage level is determined by the time a signal is present at the input that allows the feedback path across the capacitor charges and discharges. As the op-amp is configured with a negative-feedback path, it produces an output voltage that attempts to maintain a virtual ground at the inverting input. Virtual ground refers to a signal that is at the level of ground, a 0V reference voltage, without being connected together.

The result is a linear ramp of increasing voltage at a rate set by the RC (resistor/capacitor) ratio that increases until the voltage reaches saturation or the maximum output voltage of the op-amp. If the input signal to the integrator is a square wave, the integrator will create a triangle wave that follows the square wave at a slew rate determined by the RC ratio. In the case of chaotic applications, the integration creates the integral of the signal passing into it. However, a nonlinear element is necessary to create a chaotic circuit. This is because a linear system cannot be chaotic. This nonlinear element comes in various forms, but the simplest is a diode. A diode is an electronic component whose primary function is to allow current to flow in one direction. The forward direction of a diode offers close to no resistance, while the reverse direction of the diode provides near-infinite resistance. An ideal diode has two states, on or off, allowing current to flow or not. When the diode is off, the current flow through the diode is zero; when on, the voltage drop is zero as the current flows freely. However, the world does not operate in an ideal way. The current does not change linearly as the voltage increases across a diode. By inserting a nonlinear element, such as a diode, into the feedback path of several integrators, you get a nonlinear differential equation in circuitry.

Ya Jerk is a two-channel chaotic signal generator based around Jerk chaos, as described in J.C. Sprott's paper, "A New Chaotic Jerk Circuit" and as modified by Andrew Fitch of Nonlinear Circuits for use in the context of modular synthesizers. It includes two channels that by default are arranged in a self-influencing feedback loop but can easily be disconnected from each other. The term jerk in jerk chaos comes from physics and describes how an object's

acceleration changes with respect to time. Jerk is the third derivative of position after velocity (first derivative) and acceleration (second derivative).

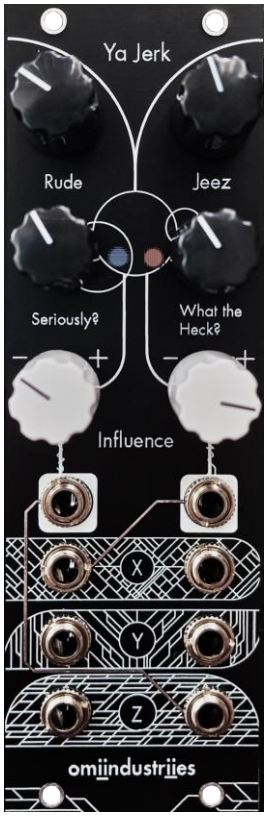


Figure 27: omiindustriies Ya Jerk Front Panel

The two channels of the module are arranged side by side, the left being the Blue channel, and the right the Red channel, based on the color of the associated LEDs. The controls are "Rude", "Jeez", "Seriously?", and "What The Heck?", chosen specifically for their vague meanings with regards to synthesizer functions, but playing on a theme of an argument. The module was conceived as a chaotic argument between two similar but slightly different chaotic entities, and by argument, I refer to the self-influencing feedback loop the two sides are arranged in by default. "Rude" and "Jeez" control the shape of the chaos, while "Seriously?" And "What The Heck?" Control the rate of integration within the feedback loop and relative speed of the chaos. In many settings, the chaos is non-periodic, meaning it doesn't oscillate at a set frequency like an oscillator, but rather oscillates up and down on a path that does not repeat. In other settings, the module oscillates at a periodic rate.

Each side has three outputs, X, Y, and Z, which each behave slightly differently. I changed the naming convention of the X, Y, and Z outputs from their labeling in Sprott's paper based on an early version of Fitch's design of the Jerk chaos circuit for modular synthesizers. In Sprott's version, the X and Z are swapped. After asking Andrew Fitch why he made that change, he mentioned that the X output of a chaotic signal generator is often the most irregular while the Z output is the smoothest. The names originally come from defining an X variable and then calculating the two successive derivatives of that variable. When replacing the X with a Z, the results are the same, but X is the third derivative of Z instead of the other way around. The nonlinear equations that describe this Jerk chaos system when written as three first-order differential equations are as follows (written twice to show both variable configurations with Sprott's equations on the right)

$\dot{Z} = Y$ or $\dot{X} = Y$

$\dot{Y} = X$ or $\dot{Y} = Z$

$\dot{X} = -X - Z - 10^{-9}[\exp(y/0.026) - 1]$ or $\dot{Z} = -Z - X - 10^{-9}[\exp(y/0.026) - 1]$

I got more into first-order differential equations in the software section of this thesis as I use them to generate different chaotic signals in code, so a more detailed explanation is forthcoming. The variables with dots over the variable correspond to the time derivative, or that value over time.

The outputs sit somewhere between an LFO (low-frequency oscillator) and a fluctuating random voltage seen by smoothing out a stepped random voltage from a sample-and-hold. The X outputs are the most complex, creating a sharp onset and decaying sinusoidal bounces. The Y outputs are less violent and complex, but still retain a dynamic movement as they oscillate. The Z outputs are the smoothest and most sinusoidal, but still display chaotic behavior and a less violent bouncing behavior.

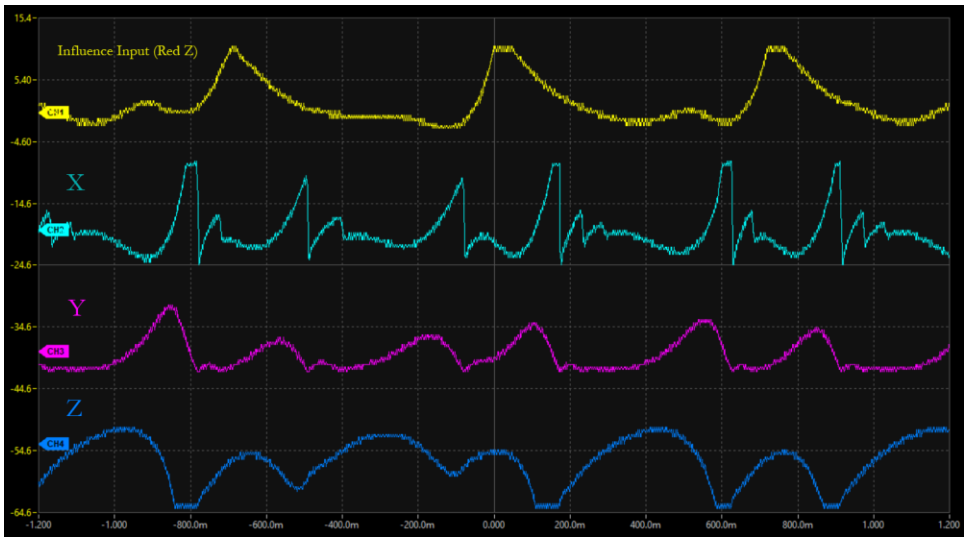


Figure 28: Oscilloscope View of Red Z, and Blue X, Y, and Z

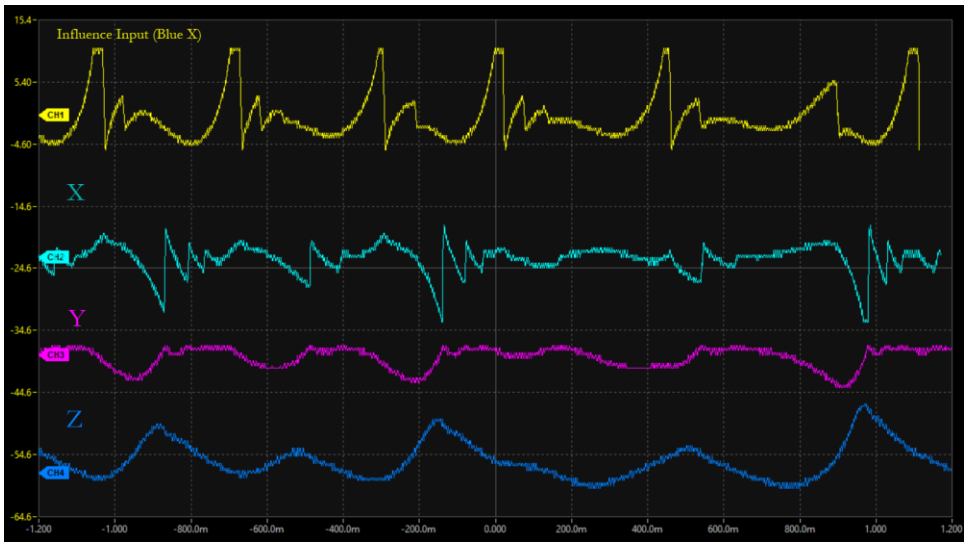


Figure 29: Oscilloscope View of Blue X**, and** Red **X, Y, and Z**

The outputs are bipolar, meaning they go both into the realm of positive and negative voltages. The outputs have a wide voltage range that changes depending on the settings of the controls and range up to a little more than +/-10V. For many applications this is a little too wide a modulation range, so users are advised to keep an attenuator handy for processing the voltage level down to a more manageable range.

As I've mentioned, by default, the two sides are connected in a feedback loop, through normalization into the Influence inputs. The X output from the Blue channel is normalized into the influence on the Red channel, while the Z output from the Red channel is normalized into the Influence input on the Blue channel. The influence inputs include attenuverters to scale and invert the incoming modulation. The Influence inputs don't change the frequency of the chaos, but rather sort of inject energy into the internal feedback path of the chaos. If users input gate or trigger signals into the Influnce inputs, the chaos roughly syncs to the incoming signals, but does not produce perfectly on-tempo modulation you might find on another modulation source with a dedicated clock input such as a tempo-synced LFO, envelope, or stepped random voltage. These inputs don't work particularly effectively with audio rate signals but will work with a wide range of sub-audio signals. The Jerk chaos could work as a crude audio filter with a different configuration of parts, as was mentioned in the Slater paper, but that was not the goal of this particular project.

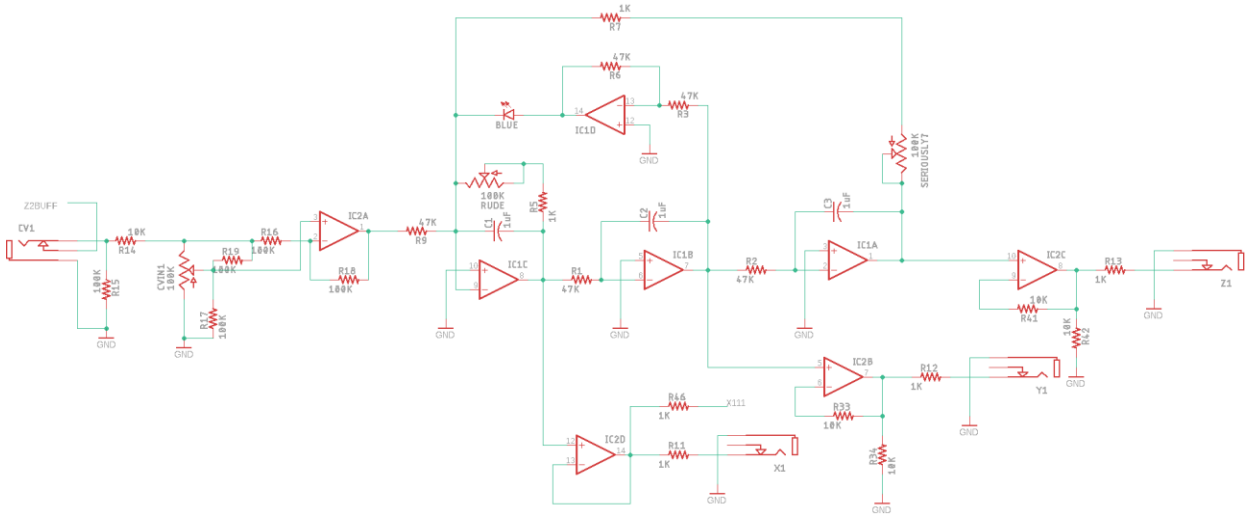


Figure 30: Schematic of the Blue Channel

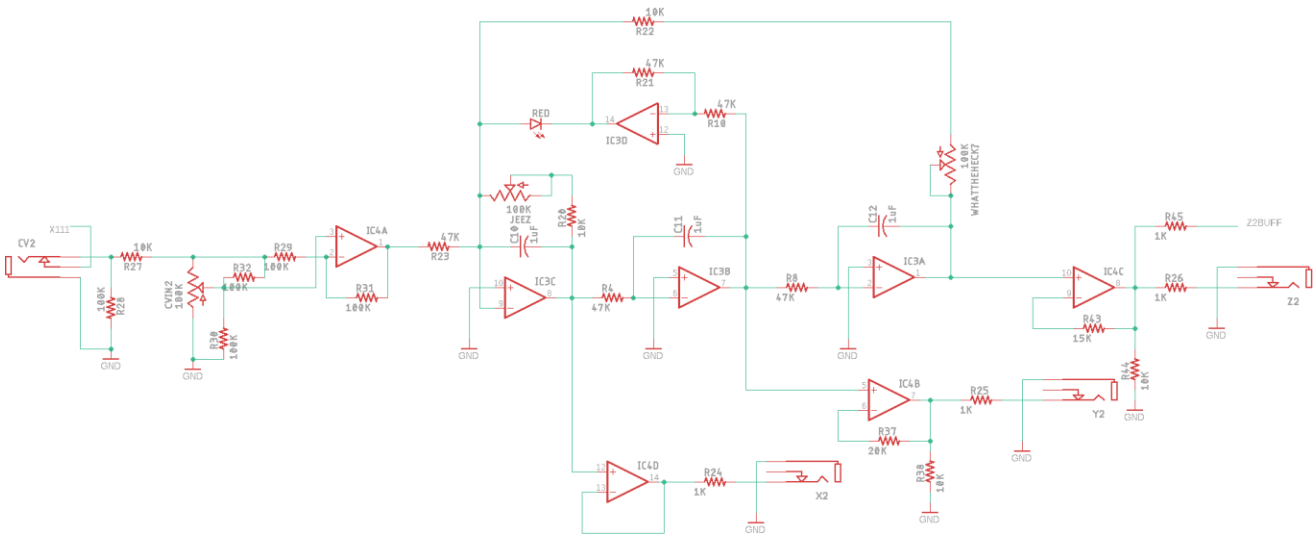


Figure 31: Schematic of the Red Channel

Now, moving from the front panel to the interior circuitry. These two schematics show the design of Ya Jerk. As you can see, the two sections are nearly identical, but the orientation of the LED is swapped between the two sides and there are a few changed resistor values. This is the cause of the differences between the outputs of the Blue and Red channels. The LED orientation of the Blue channel is consistent with the original Sprott paper and the first available version of the Fitch design. The orientation of the Red channel comes from the newer version of the Fitch designs, such as Stooges, a three-channel jerk chaos module. The resistor values for R5/R7 and R20/R22 are different, also giving slightly different behavior. This comes from an earlier version of the Ya Jerk where the integrator capacitors were different values, 1uF and 10uF. The 10uF capacitor configuration was originally chosen to allow the two sides to run at different rates but was scrapped because the chaos had dead spots where the outputs would completely turn off. While the capacitor values were changed to match, the resistor values were kept as is to add small variation in the overall behavior.

Three op-amps are configured as integrators, while the fourth op-amp acts as an inverter. An inverter takes the state of a signal passing through it and changes the gain to –1, swapping its polarity without adding or removing any amplitude. While an oscillator traditionally increases in frequency as you turn the potentiometer in the clockwise direction, Ya Jerk works

the other way, slowing down as the "Seriously?"/ "What the Heck?" Controls are turned clockwise. This is because those controls set the rate of integration, so turning clockwise increases the rate of integration, much like you would see on a slew generator or portamento control.

The "Seriously? / "What The Heck?" Controls determine the shape of the chaotic outputs and sit in a feedback loop that do not contain the nonlinear LED elements. The shape of the chaotic signals does not just mean how smooth or jagged they are but also includes the amplitude of the signals, as the amplitude of the outputs is not constant in all potentiometer positions.

Each of the stages of the integration pass into op-amps that buffer or amplify the outputs before their final output jacks. The X outputs are simply buffered, meaning that no additional amplification is added to their output voltage. The Y and Z outputs travel through non-inverting amplifier configured op-amps. A non-inverting amplifier is a way of adding amplitude to a signal while retaining the phase of the signal. The gain can be calculated as 1+ R2/R1, where R1 is the feedback resistor and R2 is the resistor connected to ground. If the two are equal, as with X and Y of the Blue channel, the gain is two. The Red Y has a gain of 2.5, while the Red Z has a gain of three. For the Blue X and Red Z, a resistor is placed between the output of the op-amp and the normalization into the Influence CV input jacks, as well as before all the output jacks. This is to prevent the accidental problem of connecting two outputs together, which may damage a module if not protected by a simple 1K ohm resistor.

The Influence CV input section uses the same attenuverter design as the Cascading Register but on the input stage rather than the output stage. The amount of change that the influence input imparts on the behavior of the module was surprising when first testing the module. Even just slight changes to the attenuverter settings cause massive changes in behavior as the chaos continues to move through its attractor orbits, thus relating to one of the fundamental elements of chaos, sensitive dependence to initial conditions.

In the next chapter, I will explore chaos on more of a mathematical level in software. By implementing chaotic equations in code, I can better visualize their behavior using computer graphics software capabilities that I do not have access to in analog hardware.

Chapter 4 Software

In order to better illustrate the idea of how chaos behaves over time and sensitive dependence on initial conditions, I chose to not only implement chaos in hardware, but also in software. This started as experiments in Max/MSP and Processing, before adding TouchDesigner and Python. While implementing chaos in a hardware modular synthesizer enables me to use it in a rather straightforward musical application, I chose to also explore chaos in software to better visualize and attempt to understand its behavior in a more straightforward environment. While the oscilloscope shots shown in the previous chapter allow you to see how the chaos moves over time in one dimension, I lack the hardware to properly display chaos in more than two dimensions. However, by solving chaotic equations in software, I could plot the X, Y, and Z parameters in a 3-dimensional environment.

The first place I wanted to start was with the Lorenz attractor, as it is one of the most ubiquitous and well-known examples of chaotic behavior. Not only that, but its simplicity makes it more accessible to someone without a background in mathematics, such as me.
The Lorenz Attractor can be described with the following equations

$dx/dt = s(y - x)$

$dy/dt = -xz + rx - y$

$dz/dt = xy - bz$

With the typical parameters being $s = 10$, $r = 28$, $b = 8/3$. The variable d_ over dt indicates the change of this variable over time. These equations are plotted continuously and each time the equation is solved, the resulting parameters are fed back into the equation and describe a rate of change.

Where did these equations come from though? Well, as I mentioned in an earlier chapter, Lorenz came across these three equations that describe convection, or the rising of hot

gases or liquids. He had originally been working on a set of 12 equations as they related to weather in his simplified simulated world, but after months of working, had narrowed it down to three equations that correspond to three dimensions. After all, humans are not well adapted at visualizing phenomena in 12 dimensions. The narrowed down equations correspond to the stream function, change in temperature, and deviation in linear temperature.

$dx/dt = s(y - x)$ corresponds to the stream function. The variable "s" (or sometimes written as "p") corresponds to the Prandtl Number, a dimensionless number that is used to indicate the ratio of viscosity of a fluid to thermal conductivity. This number changes with the fluid being studied. For example, sodium has a Prandtl number of 0.01, water has a Prandtl number of 6.90, argon has a Prandtl number of 22.77, and xenon has a Prandtl number of 674.91. The standard variable in a Lorenz equation is 28, so much greater than water and somewhat greater than argon. The lower than Prandtl number, the more effective the given gas or liquid is at conducting heat.

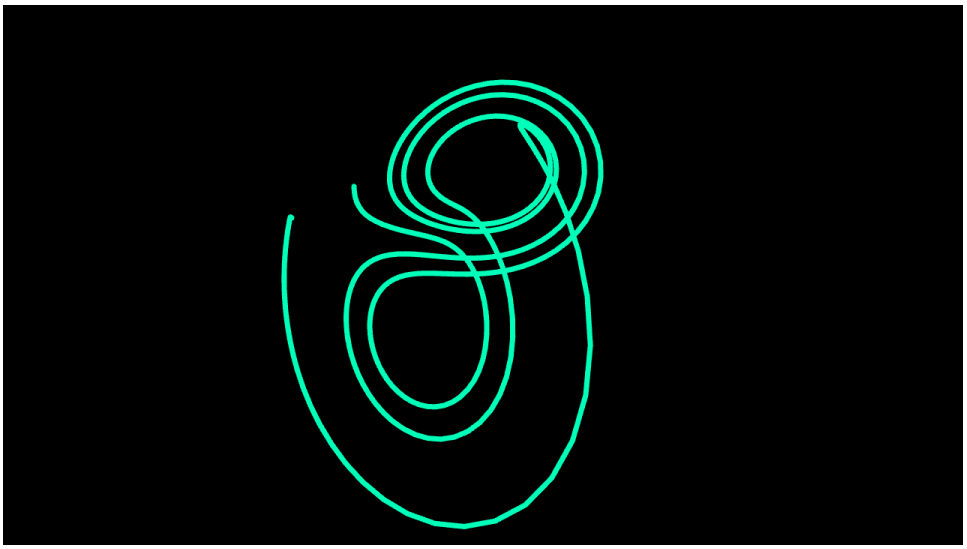
$dy/dt = -xz + rx - y$ corresponds to a change in temperature. The variable "r" corresponds to the Rayleigh Number, another dimensionless number that relates to heat transfer in convection. It can be calculated by multiplying a Prandtl number by the Grashof number. The Grashof number is a ratio of the buoyancy and viscous forces in a fluid. In this equation, -xy is a nonlinear term, an essential element in chaos.

Finally, $dz/dt = xy – bz$ corresponds to the deviation in linear temperature. This equation has to do with the ratio between the height of a fluid layer with the width of the convection rolls, or the rolls of counterrotating air in the atmosphere that sit approximately horizontal to the earth. In this equation, the 'xy' element provides the nonlinearity.

When plotted, these simple equations result in a complex shape that traces the outline of two points in space, poetically mirroring the shape of a butterfly's wing. Could the smallest flaps of a butterfly's wing in the jungle result in a hurricane two weeks later? We can't know for sure, but what is known is the small perturbation in initial state of a system as varied as the weather can result in huge changes over time.

Over time, a Lorenz attractor settles down into a discernible and well-recognized shape. Given the same *exact* initial condition, the system behaves the same way and unfolds tracing its way through space. However, if the initial conditions are varied, even slightly, the results may be wildly different.

I decided to investigate that by plotting a Lorenz attractor, not for an extended period of time, but on a small timeline, say 500 iterations of the equations. For this application, I chose to implement it in TouchDesigner with the script SOP providing Python code in order to calculate the attractor over time. The initial conditions of X, Y, and Z are generated randomly every time the script is called with a range of –0.5 to 0.5. While this seems like a small range of initial conditions, the concept of sensitive dependence becomes very clear on this timescale. Below are several images of the resulting 500 iterations of the Lorenz attractor.



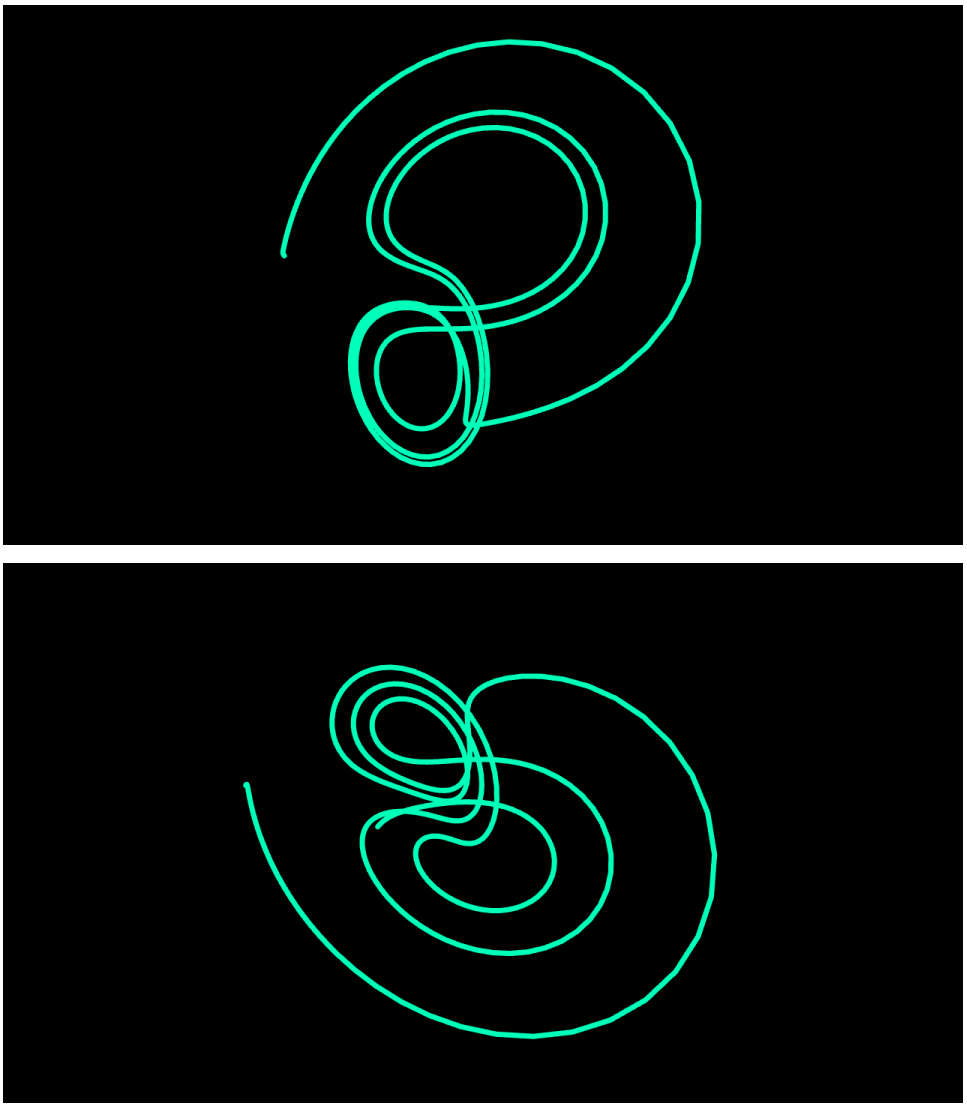


Figure 32: Three Versions of the Lorenz Attractor At 500 Iterations

As you can clearly see, they look strikingly different from each other when plotted at this scale. If you let these run for, say 25,000 or 50,000 iterations, they would appear more closely related. Granted, each iteration would, in fact, be unique and behave differently, but when viewed on a large enough scope, the behavior begins to settle down into an almost predictable pattern as it traces its way through its faux-butterfly wing path. Below is an example of the Lorenz attractor when iterated at 25,000 steps showing this behavior, as seen in an earlier chapter

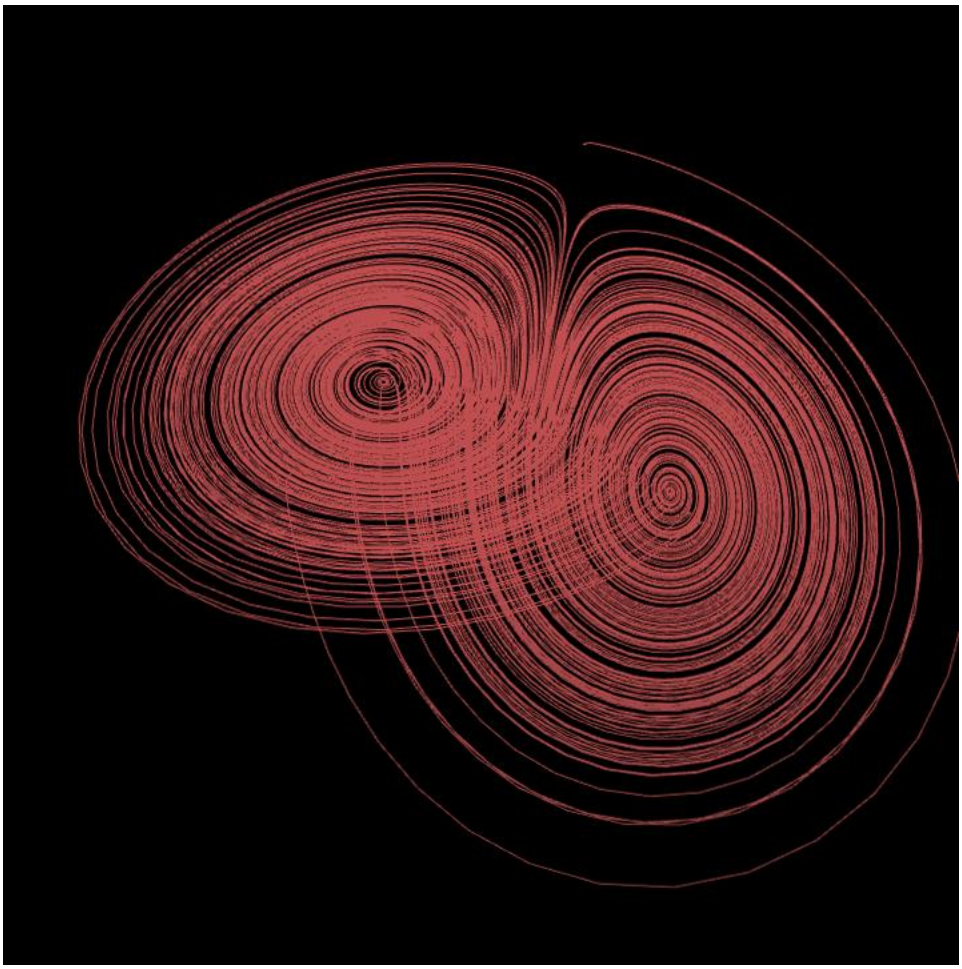


Figure 33: Lorenz Attractor at 25,000 Iterations

A paper caught my eye while browsing J.C. Sprott's website, entitled "Some Simple Chaotic Flows." His goal with the paper was to study the existence of some of the simplest chaotic equations. Lorenz's conjecture was that the Rössler attractor was the simplest example of a chaotic equation, in the algebraic sense rather than the processes that the equations describe.

The Rössler attractor can be described as

$dx/dt = -y - z$

$dy/dt = x + ay$

$dz/dt = b + z(x - c)$

Where typically a and b = 0.2 and c = 5.7

The Rössler attractor was proposed by Otto Rössler in 1976 as a way to produce chaos in a manner mathematically simpler than the Lorenz attractor. In contrast to the Lorenz attractor, it only has one nonlinearity in it. The nonlinearity appears as z*x in the equation dz/dt = b + z(x - c). When comparing the attractor to the Lorenz model, the most obvious difference is that the Rössler attractor only revolves around one point, whereas the Lorenz attractor has two points of attraction. In contrast to the Lorenz model, the Rössler attractor does not come from a simplified physical phenomenon, but purely from the realm of mathematics.

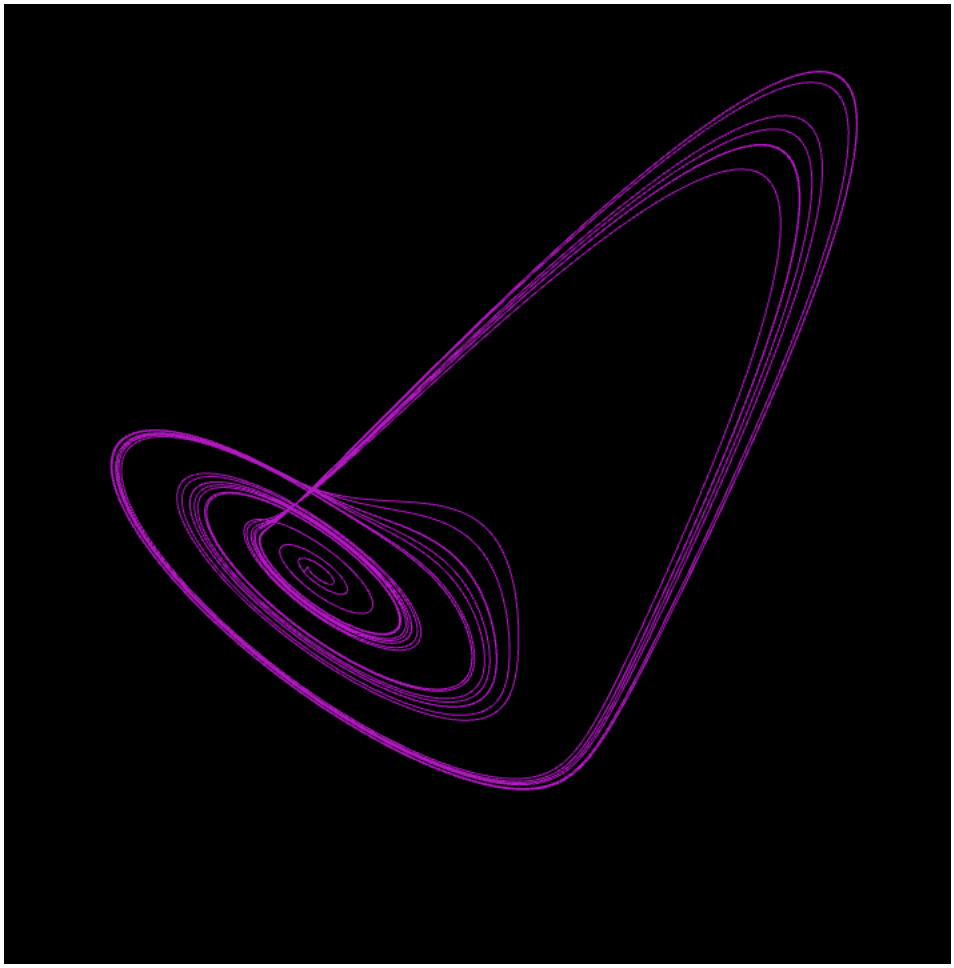


Figure 34: Rössler Attractor

The Rössler and Lorenz attractors are composed of seven terms and either one or two quadratic nonlinearities respectively. It's incredible the amount of complexity that is formed from these simple differential equations solved over time. But could chaos emerge from simpler

equations? Sprott ran a series of computer-aided experiments on differential equations with six or fewer terms that returned positive Lyapunov exponents.

A Lyapunov exponent measures how two points that start near each other change over time. By near, I mean incredibly close to each other, almost on top of each other. The Lyapunov exponent is represented by λ. If the Lyapunov exponent is less than zero, a negative number, a system falls into periodic behavior. The more negative the number, the more stable the system is. If the Lyapunov exponent is equal to zero, you get a fixed point or an eventual fixed point. If greater than zero, the small distances between two points grow indefinitely over time and you get chaotic behavior.

After examining several thousand potential chaotic equations, and throwing out the vast majority of them, Sprott came up with 19 examples of chaotic flows comprised of either six terms and one nonlinearity or five terms and two nonlinearities. He named them simply Case A through Case S. Cases A-E have five terms and two nonlinearities, and cases F-S have six terms and one nonlinearity. Many equations with six terms and two nonlinearities were discovered during this process, but this experiment was tied to algebraic simplicity in chaos and were thus discarded. No cases of systems with five terms and one nonlinearity or cases with less than five terms were found in this study.

A	dx/dt = y	dy/dt = -x + yz	dz/dt = 1 - yý
B	dx/dt = yz	dy/dt = x - y	dz/dt = 1 - xy
C	dx/dt = yz	dy/dt = x - y	dz/dt = 1 - xý
D	dx/dt = -y	dy/dt = x + z	dz/dt = xz + 3yý
E	dx/dt = yz	dy/dt = xý - y	dz/dt = 1 - 4x
F	dx/dt = y + z	dy/dt = -x + 0.5y	dz/dt = xý - z
G	dx/dt = 0.4x + z	dy/dt = xz - y	dz/dt = -x + y
H	dx/dt = -y + zý	dy/dt = x + 0.5y	dz/dt = x - z
I	dx/dt = -0.2y	dy/dt = x + z	dz/dt = x + yý - z
J	dx/dt = 2z	dy/dt = -2y + z	dz/dt = -x + y + yý
K	dx/dt = xy - z	dy/dt = x - y	dz/dt = x + 0.3z
L	dx/dt = y + 3.9z	dy/dt = 0.9xý - y	dz/dt = 1 - x
M	dx/dt = -z	dy/dt = -xý - y	dz/dt = 1.7 + 1.7x + y
N	dx/dt = -2y	dy/dt = x + zý	dz/dt = 1 + y - 2z
O	dx/dt = y	dy/dt = x - z	dz/dt = x + xz + 2.7y
P	dx/dt = 2.7y + z	dy/dt = -x + yý	dz/dt = x + y
Q	dx/dt = -z	dy/dt = x - y	dz/dt = 3.1x + yý + 0.5z
R	dx/dt = 0.9 - y	dy/dt = 0.4 + z	dz/dt = xy - z
S	dx/dt = -x - 4y	dy/dt = x + zý	dz/dt = 1 + x

Figure 35: Equations for Sprott's Cases A-S

Complex behavior from simple building blocks and rules has and continues to be a point of interest for me in my research and artistic practices. I decided to recreate the chaotic equations in the Max/MSP coding environment Gen~. This was firstly because of the ability to load gen~ patches onto a Daisy microcontroller, but also because gen~ provided a sample-level way to manipulate code that lent itself to patching up differential equations. The equations could also be developed into a Max for Live (M4L) device in order to bring chaotic modulation into an Ableton Live environment.

I paired the gen~ code with Processing code in order to visualize the resulting chaos in 3D space. While Max/MSP has the capability to run the code and display it in 2 dimensions using the scope object, Processing includes the ability to plot the resulting signals in 3 dimensions with the additional parameter of color to better illustrate the behavior of the chaos.

Below are a few examples of the resulting chaos.

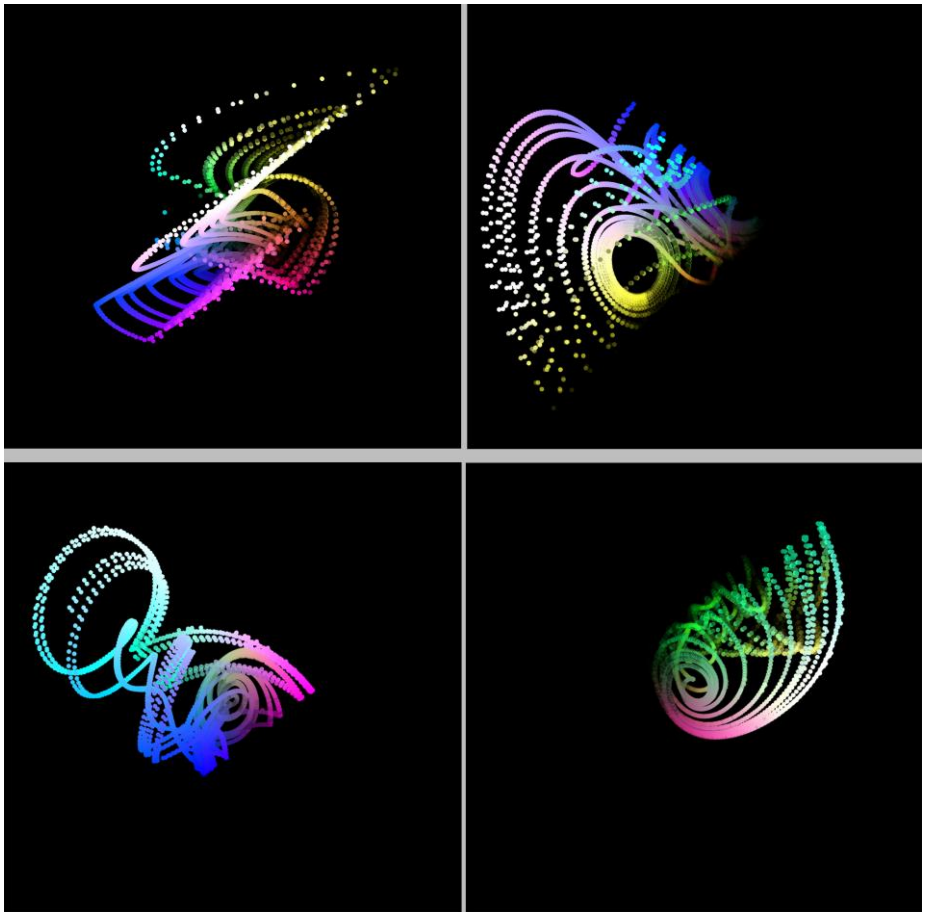


Figure 36: 3D Views of Cases B, C, F, and H

The equations are selected by a dropdown menu which outputs an integer and a bang message. The integer from the dropdown menu selects which equation is active by a nested 'if' statement on input two. The bang message has two purposes. One is to send a 1 and then a 0 with a 25ms delay to input three; and the other is to select a random number between 0-1000, which is then scaled to a number between –0.03 and +0.03. When input three receives a 1, it resets the X, Y, and Z values to a new value. This new value is determined by the three random objects added to an initial value of 0.05, which Sprott indicated in his paper was a good initial condition. By adding this small amount of variation, the initial values are close to that provided working initial condition, with enough variation to generate different behavior every time the equation is selected.

Here is an example of some of the code in the Gen~ codebox. Commented out are the equations source from Sprott's website, which just act as reference for the resulting code. The xgain, ygain, and zgain lines set the overall amplitude of the resulting chaos, which I tried to contain to the range of +/-1, but with some room for deviation. I used the variable "a" in the cases to represent the integers in the chaotic equations, in other equations I used additional variables, with the labels "b" and "c", although one could use any variable names they see fit.

Figure 37: Gen~ Implementations of Cases B, C, and D

The equations given could not be calculated in the form listed on Sprott's website, so they were refactored. For example, in Case C, dx/dt = yz is changed to x = x + dt * (y * z). This could be broken down to x is equal to itself plus some time factor (dt) and multiplied by the y and z variables. After that the Y and Z variables are calculated, and the new X, Y, and Z variables are fed back into the equation, replacing the old values. This process is repeated continuously to generate the resulting chaos.

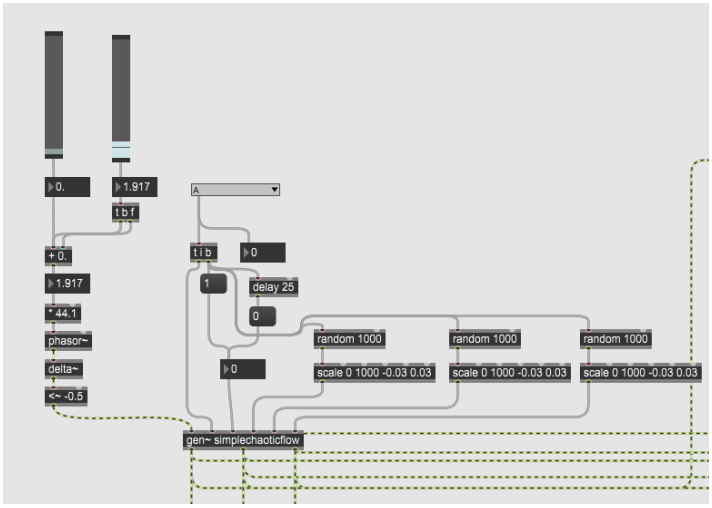


Figure 38: Max/MSP Controls outside of Gen~ Codebox

The speed of the chaos is determined by a pair of sliders, one with a range of 0-20, and the other with a range of 0-499, which are added together. This corresponds to Hertz or cycles per second; giving users the ability to set the chaos to behave as audible noise or sub-audio modulation. The slider values are multiplied by 44.1, or the sampling frequency, and passed into a phasor~ object, a delta~ object, and finally a <~ object before running into the gen~ codebox to run the equations. The dt or time constant was left static at dt=0.05. I ran into issues with three of the equations and they were thus left out of the final selection of equations. Cases I, L, and Q tended to fly off into infinity and crash the program.

I then converted the resulting output from the gen~ code, which was not output as numbers but Max's signal format, usually used for audio signals. I rescaled it from –1.0-+1.0 to 0-1.0, ran it into a meter object which turned it from a signal to a float, and then rescaled it back

from 0-1.0 to –1.0-+1.0. These floats were then packed, prepended, and sent as OSC messages to Processing.

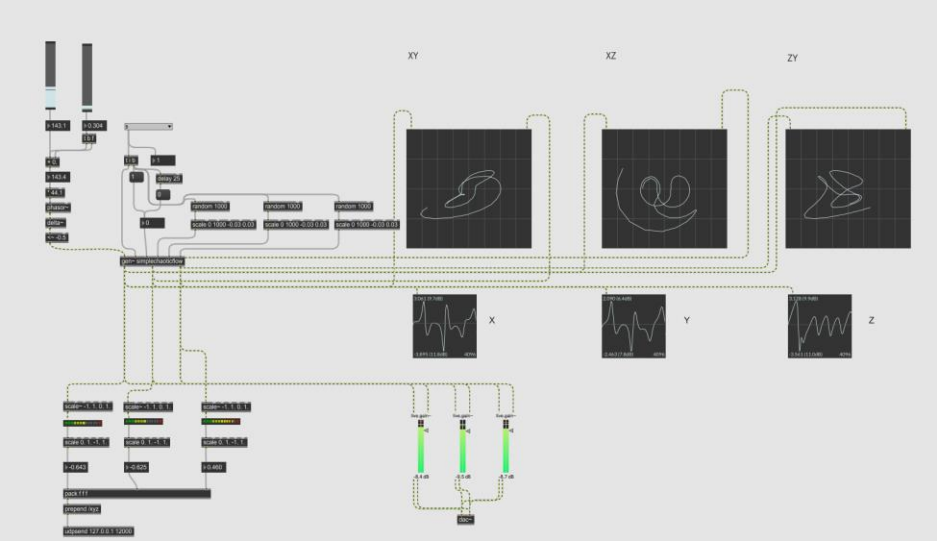


Figure 39: View of the Entire Max/MSP Patch

Processing is a coding environment that is very well suited to visual applications. It uses a simplified version of JavaScript with a focus on generating visuals from code. It is a flexible tool and is suited for both 2D and 3D applications. The Processing code is broken down into two parts, the main code and an external function in order to keep the code concise. The canvas, or area where the visuals are generated, has a width and height of 1000 pixels and the code runs at 60 frames per second.

scope basic | Particle

```
import oscP5.*;
import peasy.*;

OscP5 oscP5;

PeasyCam cam;

ArrayList<Particle> particles = new ArrayList<Particle>();

void setup() {
  frameRate(60);
  size(1000, 1000, P3D);
  oscP5 = new OscP5(this, 12000);
  cam = new PeasyCam(this, 2000);
  background(0);
  strokeWeight(10);
}

void draw() {
  background(0);

  for (int i = 0; i < particles.size(); i++) {
    Particle part = particles.get(i);
    part.draw();
  }

}

void oscEvent(OscMessage msg) {
  if (msg.checkAddrPattern("/xyz") == true) {
    float x = msg.get(0).floatValue();
    float y = msg.get(1).floatValue();
    float z = msg.get(2).floatValue();

    particles.add(new Particle(x, y, z));
  }
}
```

Figure 40: The Main Processing Code

The function, named Particle, scales the OSC messages from –1.0 and 1.0 to the width and height of the canvas divided by four in order to keep the signals within the bounds of the drawing area. Naturally, the X, Y, and Z signals from Max are mapped to the X, Y, and Z axis of the 3D space. Additionally, the X, Y, and Z signals get mapped between 0-255 in order to generate red (X), green (Y), blue (Z), and alpha (also Z) for the drawn stroke. The draw function takes the XYZ points in space and the resulting color and generates a series of circular points that trace the shape of the chaos in space.

scope basic | Particle

```
class Particle {

  float x, y, z;
  float mappedX, mappedY, mappedZ;
  float mappedXc, mappedYc, mappedZc;
  float mappedZf;

  Particle(float _x, float _y, float _z) {
    x = _x;
    y = _y;
    z = _z;

    mappedX = map(x, -1.0, 1.0, -width/4, width/4);
    mappedY = map(y, -1.0, 1.0, height/4, -height/4);
    mappedZ = map(z, -1.0, 1.0, -width/4, width/4);

    mappedXc = map(x, -1.0, 1.0, 0, 255);
    mappedYc = map(y, -1.0, 1.0, 0, 255);
    mappedZc = map(z, -1.0, 1.0, 0, 255);
    mappedZf = map(z, -1.0, 1.0, 0, 255);
  }

  void draw() {

    stroke(mappedXc, mappedYc, mappedZc, mappedZf);
    point(mappedX, mappedY, mappedZ);
  }
}
```

Figure 41: The Particle Processing Code

From there, I chose to explore these simple chaotic equations in TouchDesigner, another visual programming environment, but much more suited to real-time visualization. Using the Script SOP, I wrote external Python code in order to run the same equations I explored in Max/MSP and Gen~. TouchDesigner also has the added benefit of a suite of video processing capabilities, which I harnessed for artistic applications. The results of my experiments in coded chaos generate visually interesting results, which appeal to me not just as someone interested in the behavior of chaos but also as a visual artist.

```python
#initialize points array with random starting point
def initPoints(scriptOp):
    rndX = rng.random() * 2 - 1
    rndY = rng.random() * 2 - 1
    rndZ = rng.random() * 2 - 1

    points = []
    points.append([rndX, rndY, rndZ])
    scriptOp.store('points', points)
    return
```

Figure 42: Python Initial conditions

First to initialize the code, I generated an empty array to store the points of the XYZ signals. The XYZ starting positions are randomized in a similar way to the Max MSP, but inside the code instead of externally. I chose to allow control over the dt time variable, as well as the number of points drawn by the program and the active equation. The Python code took the empty array and populated it with the XYZ points from the equations. Then took the previous point and a new point and drew a line between the two of them. When the array of points, defined by the NumPoints parameter, reaches the maximum number of points, the last point in the array is lost. It continuously updates, tracing the results of the equations in a 3D environment. An external operator, the Timeline CHOP is called to generate the code in real time, much like the Phasor~ object in Max.

```
def onCook(scriptOp):
    scriptOp.clear()

    counter = op('timeline1')
    print(counter[0])

    #grab all ui parameters
    dt = scriptOp.par.Dt.eval()
    num_steps = scriptOp.par.Numpoints.eval()
    equation = scriptOp.par.Equation.eval()

    #grab points from storage
    points = scriptOp.fetch('points')
    #get most recent point from array
    prev_point = points[len(points) - 1]

    #convert most recent point to numpy format (because all chaos functions expect numpy arrays as input)
    prev_point = np.array([prev_point[0], prev_point[1], prev_point[2]])

    #calculate next point
    next_point = calculateNextPoint(equation, prev_point, dt)
    #append next point to points array
    points.append(next_point)

    #add all points to script op for drawing
    for i in range(len(points)):
        x, y, z = points[i]
        point = scriptOp.appendPoint()
        point.P = (x, y, z)

    #remove oldest point when array is too large
    if (len(points) >= num_steps):
        points.pop(0)

    #draw all poly-lines
    for i in range(1, len(scriptOp.points)):
        poly = scriptOp.appendPoly(2, closed=False, addPoints = False)
        poly[0].point = scriptOp.points[i - 1]
        poly[1].point = scriptOp.points[i]

    return
```

Figure 43: Main Processing Code For Drawing

There are slight variations in how the code is written due to the differences between Python and Gen~. The largest difference is the omission of the "dt" parameter within the equations. This happens outside the functions when the next point is calculated from the previous point in the array. The equations also are formatted slightly differently; instead of the x, y, and z variables equaling each of the equations, I used xp, yp, and zp to store the state of the x, y, and z variables before passing those values back into the equations. Unfortunately, many more of the equations tended towards infinity or otherwise crashed after running for a short while. I'm unsure of what caused these equations to behave erratically, I would imagine that TouchDesigner is not intended to solve differential equations in real time.

```python
def SCe(xyz, a = 1, b = 4):

    x, y, z = xyz

    xP = z * y
    yP = pow(x, 2) - y
    zP = a - x * b

    return np.array([xP, yP, zP])

def SCf(xyz, a = 0.5, b = 1):

    x, y, z = xyz

    xP = y + z
    yP = -x + a * y
    zP = pow(x, 2) - b * z

    return np.array([xP, yP, zP])


def SCg(xyz, a = 0.4, b = 1):

    x, y, z = xyz

    xP = a * x  + z
    yP = x * z - y
    zP = -x + y


    return np.array([xP, yP, zP])
```

```python
def calculateNextPoint(equation, prev_point, dt):
    #check equation string and apply corresponding function to calculate next point
    if (equation == 'a'):
        next_point = prev_point + SCa(prev_point) * dt
    elif (equation == 'b'):
        next_point = prev_point + SCb(prev_point) * dt
    elif (equation == 'c'):
        next_point = prev_point + SCc(prev_point) * dt
    elif (equation == 'd'):
        next_point = prev_point + SCd(prev_point) * dt
    elif (equation == 'e'):
        next_point = prev_point + SCe(prev_point) * dt
    elif (equation == 'f'):
        next_point = prev_point + SCf(prev_point) * dt
    elif (equation == 'g'):
        next_point = prev_point + SCg(prev_point) * dt
    elif (equation == 'h'):
        next_point = prev_point + SCh(prev_point) * dt
    elif (equation == 'j'):
        next_point = prev_point + SCj(prev_point) * dt
    elif (equation == 'k'):
        next_point = prev_point + SCk(prev_point) * dt
    elif (equation == 'm'):
        next_point = prev_point + SCm(prev_point) * dt
    elif (equation == 'n'):
        next_point = prev_point + SCn(prev_point) * dt
    elif (equation == 'o'):
        next_point = prev_point + SCo(prev_point) * dt
    elif (equation == 'p'):
        next_point = prev_point + SCp(prev_point) * dt
    elif (equation == 'q'):
        next_point = prev_point + SCq(prev_point) * dt
    elif (equation == 'r'):
```

Figure 44: Equations for Cases E, F, and G, and a nested elif Tree with Each Case

The results of the Python script then pass into a Geo COMP, which takes the drawn points and puts them in a 3D environment. Always paired with the Geo in TouchDesigner are the Camera and Light COMPs, which act much in the way you might expect, lighting and capture a view of the geometry. A Line MAT gives the drawn points more of a defined shape and a light purple color. The camera rotates around the drawn chaos points, varying the view over time and making it more dynamic. These elements all combine in the Render TOP, which takes the 3D environment and converts it to 2D images.

Figure 45: TouchDesigner 3D Rendering

After the Render TOP, the resulting 2D image passes into a Composite TOP. This TOP combines two or more video signals together in a variety of operations, from multiplying them to taking the XOR to burning the color of one signal onto another. I composited the render with a Feedback TOP. As the name implies, the Feedback TOP is used for generating feedback loops. By default, in TouchDesigner, you can't connect and output back into an input further back in the signal chain; but the feedback TOP is the exception to that rule, specifically designed in order to generate feedback loops.

I've discussed different implementations of feedback loops during the course of this thesis. The TouchDesigner internal feedback loop is just one example of a video feedback loop. The other is a camera/monitor feedback loop, which I will discuss further in the next chapter. The feedback TOP in TouchDesigner offers certain functionality not available in physical feedback loops but has certain limitations as well. The feedback loop takes the output and routes it back to the input, creating a self-influencing feedback loop.

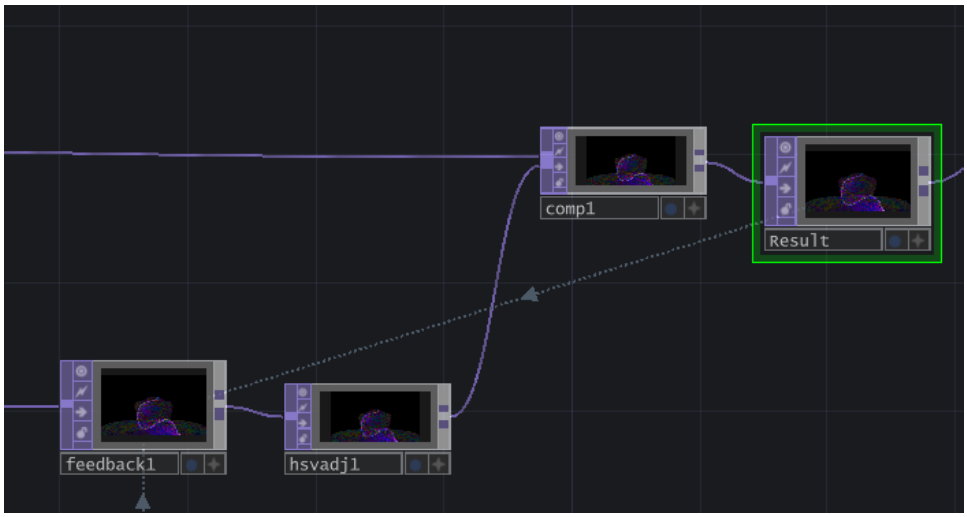


Figure 46: TouchDesigner Feedback path

The source for the feedback loop is a Null TOP on the other side of the composite TOP, an empty container that can store information. The Feedback runs into a HSV Adjust TOP, or Hue, Saturation, and Value adjuster. Any changes between the feedback TOP and the null where it takes information from changes the behavior of the resulting image. Changes may include rescaling, coloring, rotating, or otherwise affecting the behavior of the signal passing through the loop.

I made only slight adjustments to the saturation and value multiplier, adding just a little bit of saturation and subtracting just a little bit of value. This is to make sure the feedback doesn't get out of hand but adds a bit of interesting variation to the resulting signal. The composite TOP is set to difference, outputting the difference between the incoming chaotic figure and the resulting feedback loop. This results in trails following the curls of the chaos as it traces its path through the 3D space. With the addition of saturation enhancement, the result is textured and colorful, tracing where the 3D movement overlaps with the 2D feedback.

The final result of this process is like the results from the Max/MSP and Processing, but intentionally different and distinct. While the actual drawn shapes lack the vibrant colors generated from Processing, the addition of the feedback and constantly turning camera angle adds interesting variation that separates the two chaotic operations. Below are some of the resulting images.

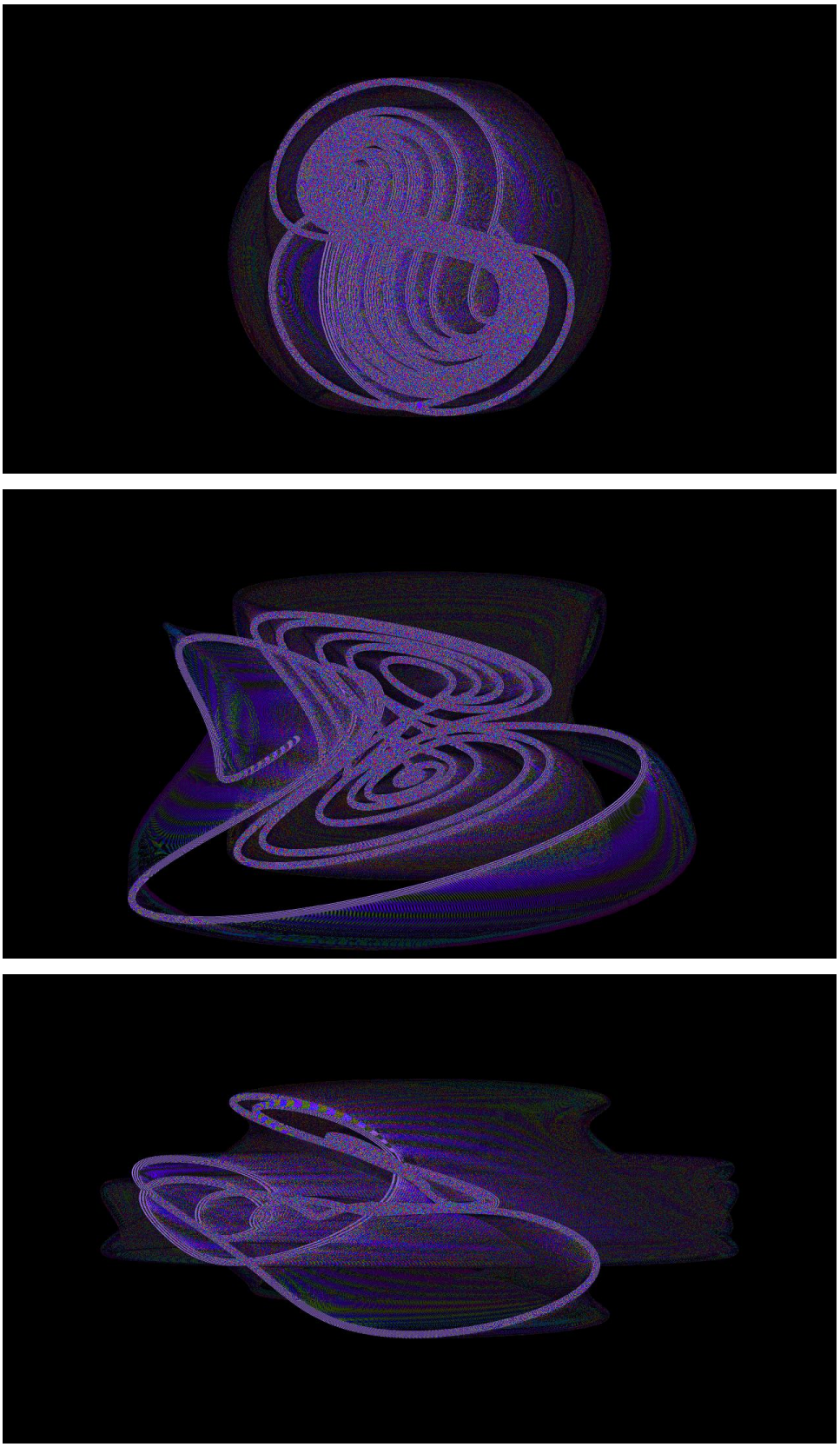


Figure 47: Cases A, B, and C

I took the experiments in TouchDesigner and decided to apply them to a larger project, not just as a self-contained image or video generator, but as part of an installation. The next chapter details my applications of both hardware and software and chaotic applications.

Chapter 5 Applications

In order to bring together the worlds of hardware and software, I chose to exhibit both in an installation entitled *Feedback Loops, All the Way Down*. The goal of the installation is to showcase some of the ways to harness chaos and unpredictability in an audio/visual artistic context. The name comes from a mantra repeated in order to center and calm me down when I become overwhelmed by anxiety. The universe is a series of complex and interconnected feedback loops that influence every aspect of our lives and the existence of everything in the universe.

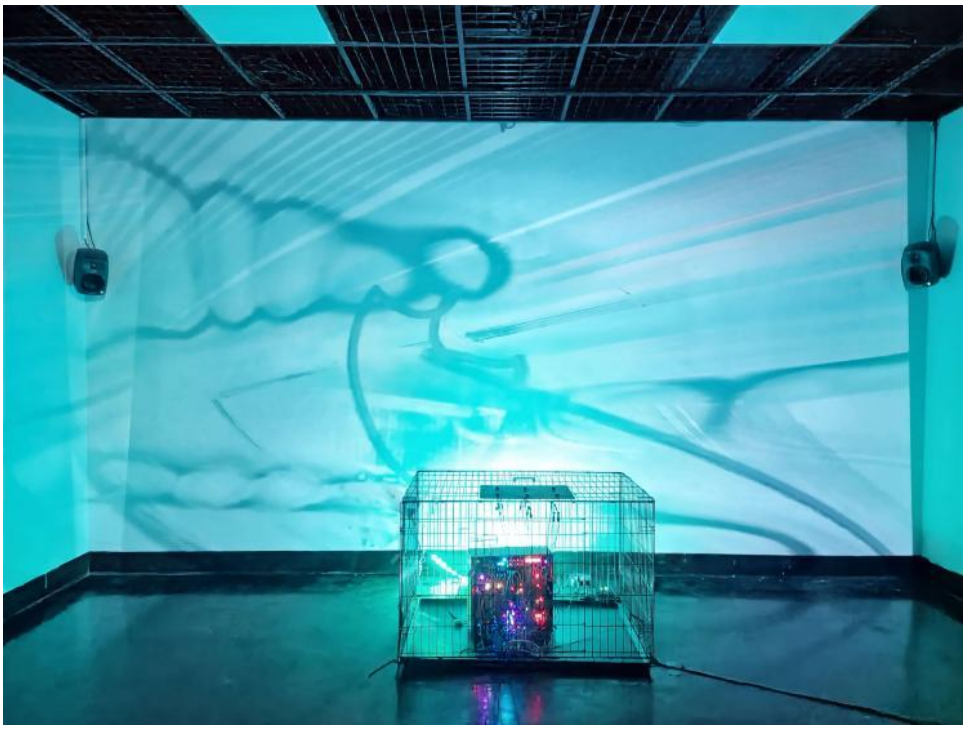


Figure 48: View of the Installation

The installation combines both audio synthesis from a modular synthesizer and video synthesis and manipulation from TouchDesigner. The goal was to combine and explore the techniques of software chaotic signal generation, creating unpredictability in hardware, and how to apply audio and video feedback for artistic applications. However, the hardware and software elements are not separate, but brought together with the help of a converter named Introductions.

Introductions is a small interface that turns analog CV and gate signals into information that a computer can understand. It grew and evolved several times starting in the fall of 2021 and to this day is still being refined. The idea was to be able to interface a modular synthesizer with a computer in order to extend the functionality of both. In the current iteration, it takes CV and gate signals and turns them into MIDI note on/off and CC messages, although it could also send other kinds of serial information, i^2C, or other data types with some tweaks to the code.

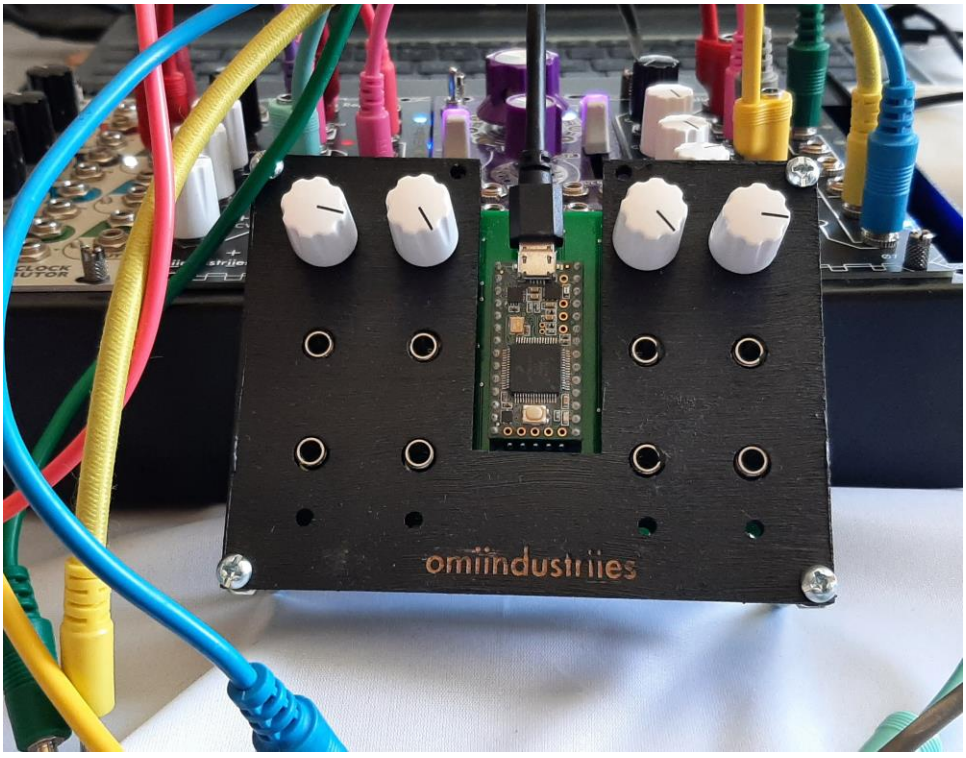


Figure 49: Introductions in Front of Small Modular Synthesizer

The concept started as a way to send information both to and from a computer, but the scale of the project was adjusted for the time being. Instead of sending and receiving data, the data flow is one way. The hardware uses a Teensy 3.2 microcontroller and a collection of jacks and potentiometers. The potentiometers allow users to adjust the overall range of signals coming into the microcontroller. With nothing plugged into the corresponding jack, a 3.3V signal is normalized into the signal path, allowing the potentiometer to act as an offset or manual controller over signal level. The Teensy 3.2 microcontroller accepts signals up to +5V, so the potentiometers act to attenuate the signal level to a range the microcontroller expects. Because the Teensy 3.2 only accepts signals between 0V (gnd) and +5V, I am unable to patch the outputs of Ya Jerk directly into the CV inputs as its outputs swing up to +/-10.5V, but it can modulate and influence other signals passing into Introductions.

I used a Eurorack case that housed a collection of modules from different manufacturers but included two Ya Jerks and two Cascading Registers, as well as another module of mine, the R2Rawr. The R2Rawr is much like the signal path from CV1 and CV2 in the Cascading Register, except that it has five inputs for external signals. These are primarily intended to be gate signals, and from those gate signals, it generates stepped CV signals. Other manufacturers included Nonlinear Circuits, Doepfer, Make Noise, Noise Engineering, 4ms, ADDAC, Qu-Bit Electronix, Mystic Circuits, and Bastl Instruments.

Figure 50: The Modular Synthesizer Setup Used in the Installation

(L) Patched (R) Unpatched

There were three primary sound sources, a collection of oscillators from Noise Engineering (Loquelic Iteritas and Sinc Iter) and 4ms (Ensemble Oscillator). These ran into a number of effects modules, including one called Dark Matter from Bastl Instruments. This module was key in the patch and for the theme of feedback loops. It is a module inspired by no-input mixer techniques and was designed in partnership with Peter Edwards of Casper Electronics. No-input mixing is a technique where instead of connecting external instruments to an audio mixer, you feed the outputs of a mixer back into itself in order to generate sounds from the resulting feedback loops. While I did use an external oscillator patched into the Dark Matter, it includes a send and return loop. With nothing patched, it simply creates a feedback loop; but you are able to patch from the output of the feedback loop to external effects and then back into the feedback loop. I did so with the help of a module called the Data Bender from Qu-Bit Electronix. This module was created to model some of the ways that audio equipment can fail, inspired by circuit bending. Patching a feedback loop into a module intended to amplify the inherent flaws in audio equipment leads to some interesting results.

I also patched one of the oscillators into the Tapographic Delay from 4ms, a delay module which includes a feedback control to set the amounts of repeats in the delay line. I sent

the output of the Dark Matter into a module called the Make Noise Phonogene, a digital recreation of the tape machine as a musical instrument, as pioneered by Musique Concrète. This module can record audio into a buffer much like a tape machine and includes further controls that take it into the realm of granular synthesis. By periodically triggering the recording of loops of audio, I capture the result of the Dark Matter feedback loop. However, there are imperfections in the recording, and I modulated the parameters of the module to further change the resulting sound. It creates an imperfect loop of audio, altered through the process of recording and modulation.

Modulation of these audio processes are primarily done by the pair of Cascading Registers and the pair of Ya Jerks. Additional modulation came from the Nonlinear Circuits Sloths, a chaotic signal generator whose primary purpose is to generate extremely slow modulation and the Make Noise Maths, a function generator based on the Serge Dual Universal Slope Generator. While Ya Jerk is arranged in a two-channel feedback loop by default, the pair of Ya Jerks were connected in a larger feedback loop which included all four channels of chaotic signals in a self-influencing feedback loop. One of the outputs from the Cascading Register ran into a Make Noise Function, a smaller version of Maths. This module acted as a slew generator, smoothing out the stepped CV signals from the Cascading Register.

The Cascading Registers' clock outputs acted as a pair of master timing sources, sending timing information to the 4ms Quad Clock Distributor and the Olivia Artz Modular Uncertainty, two modules that take gate signals and alter them. The QCD is a clock divider/multiplier, which takes a clock signal and either speeds it up or slows it down based on an integer division of the incoming gate signal. The Uncertainty also affects a timing signal, it alters the probability that the timing signal will pass to one of its eight outputs. The top output has the highest probability of passing the input signal, while the bottom output has the lowest probability of passing the input signal.

In addition to modulating the audio modules, several outputs from the modular synthesizer were patched into Introductions. Two outputs from Maths and two outputs from the Cascading Register (one smoothed by Function) were passed into the CV inputs on

Introductions. One gate output from each of the Cascading Registers and one of the outputs from the Uncertainty provided timing information for Introductions.

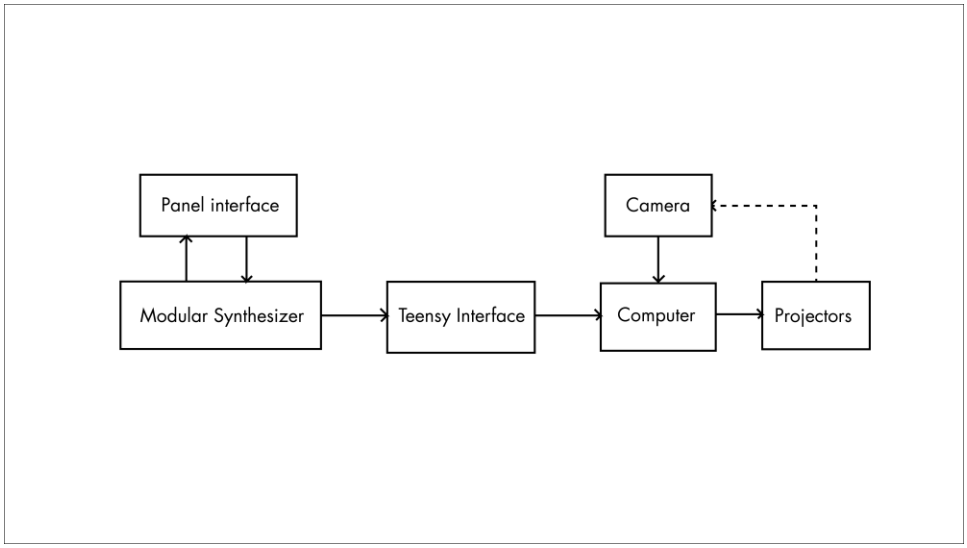


Figure 51: Block Diagram of Signal Flow through the Installation

The modular case sat in a cage in the middle of the installation space. This was to prevent access from the viewers and would-be thieves. However, there was some limited interaction between the audience and the modular synthesizer. On top of the case sat a piece of wood with six knobs able to be touched by the viewers. These were connected to the delay time, oscillator spread, the Maths CV output connected to Introductions, and two were connected to the Ya Jerk feedback loop between the two modules.

In the previous chapters, I've discussed several systems that involve feedback processes in order to generate unpredictable results. One commonly seen iteration of feedback comes in the form of video feedback, where a camera is pointed at a monitor that is displaying the output of the camera. While it often shows up accidentally, it has been a tool for artistic expression and even a tool for the exploration of temporal and spatial dynamics. James P Crutchfield, in his paper Space-Time Dynamics in Video Feedback, published in 1984, outlines the possibilities for using camera feedback to model different dynamical systems and reaction-diffusion systems.

Video feedback is a self-influencing system that takes its output and feeds it back into its input. Due to imperfections in electronics, lenses, and monitors, the feedback path is nonlinear, an essential element for chaotic behavior. The camera takes an image, converts it to electrical signals, and passes it to the monitor, which interprets the electrical signals in order to reconvert them to an image. This repeats over and over in a continuous loop. Each time it passes through this loop, it is altered, and each part of the image influences the areas around it. Parameters within this loop can be altered at will by the operator, such as rotating the camera, adjusting focus, zooming in or out, changing brightness on the monitor, or other image processing techniques.

The processes involved in a video feedback system are what is known as a dissipative dynamic system. The energy in the system is lost to small amounts of error and freedom and the system contracts over time. Often, the video feedback creates an attractor, either a fixed point, limit cycle, or a chaotic attractor. The fixed-point attractor rests in equilibrium, while the limit cycle oscillates in a predictable repeating pattern. The chaotic attractors amplify noise within regions of the system while remaining globally stable, a mix of stability and instability. These attractors may be disturbed and may or may not return to their previous form, or morph into entirely new attractors, depending on the amount of change over time. Crutchfield's writing describes an analog system of video feedback, but feedback also exists in the form of digital feedback, as I described in the previous chapter.

In my installation, I used a combination of camera and node-based feedback loops in order to generate visual complexity and interest. There were several node-based feedback loops, one connected directly to the chaotic shapes and the other composited with the output of a webcam. A switch object selected between this second feedback loop and the output of a displace object that combined the output of a complex video oscillator device, chaotic 3D shape, and the processed output of the webcam. The webcam was connected to cache TOP which stores the information of an incoming video signal. By adjusting the cache position with a smoothed random signal, the timing of the video signal can be adjusted, scrubbing back and forth through time, creating a sort of simple glitchy video delay.

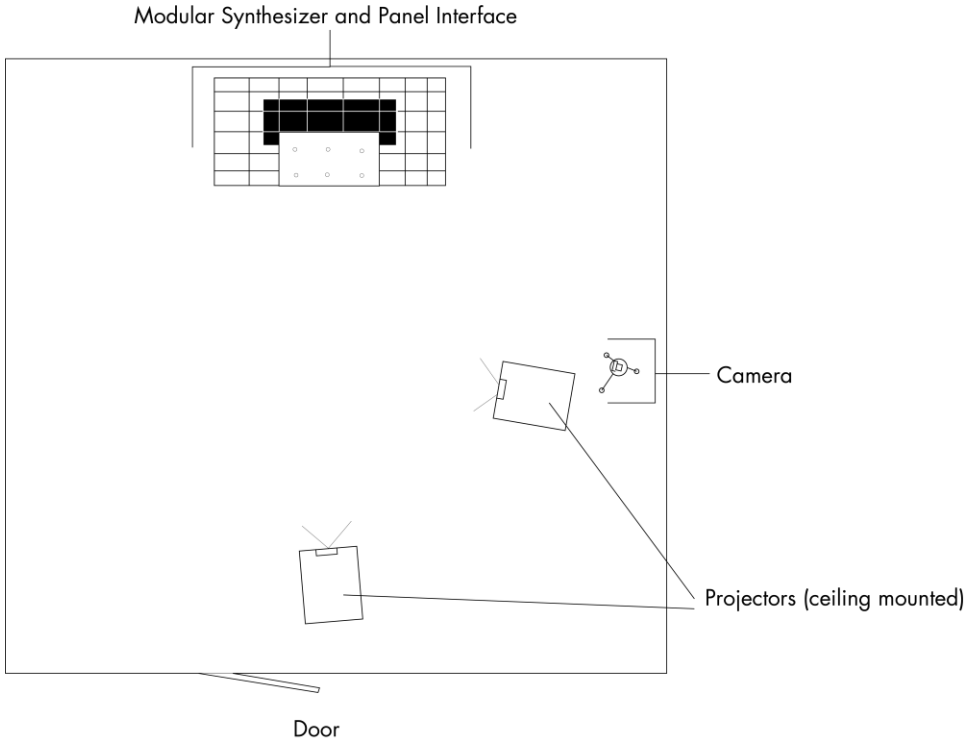
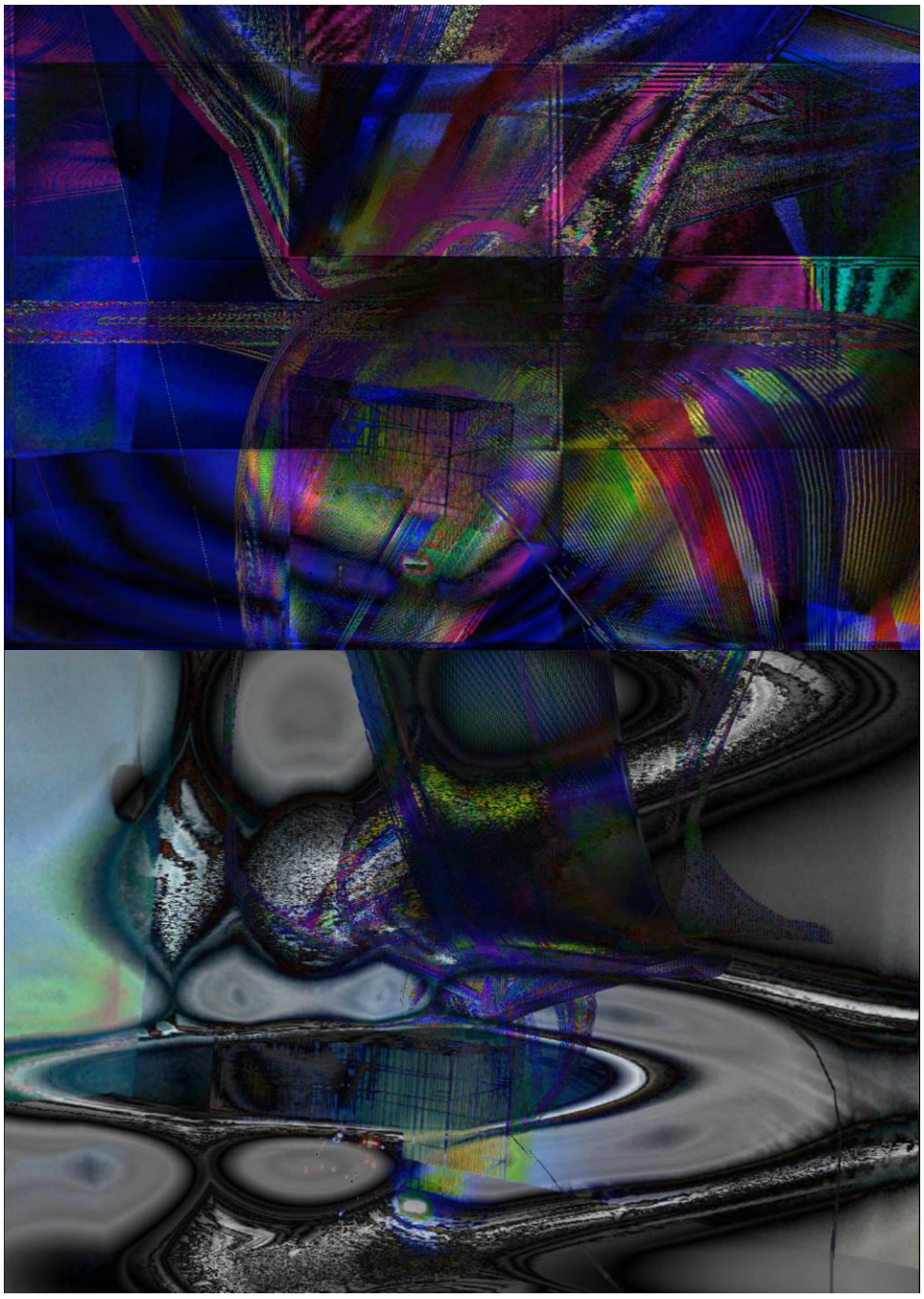
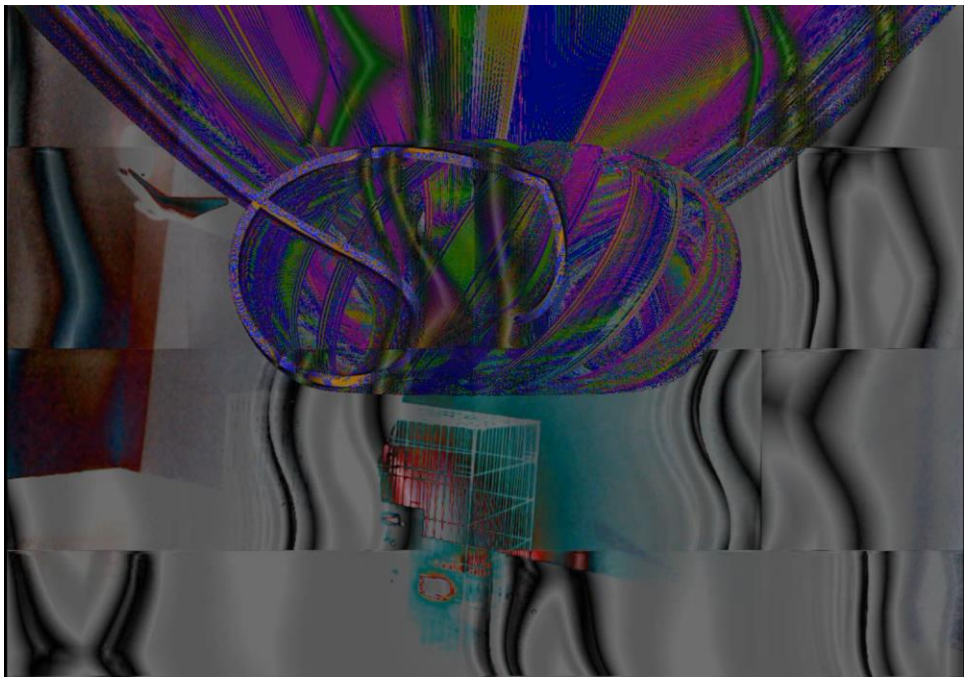


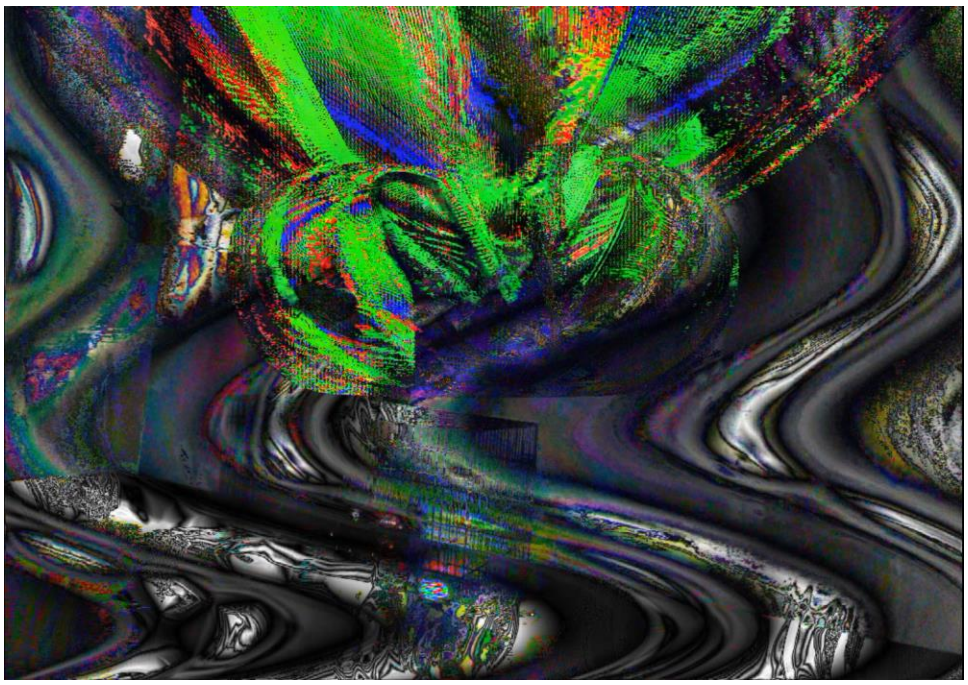
Figure 52: Top-down View of the Installation

The output of the TouchDesigner project was sent to a pair of projectors mounted on the ceiling of the installation space. The webcam was pointed at the wall where one of the projectors displayed its image, creating yet another feedback loop. When viewers walked up to the modular synthesizer cage, they interrupted the feedback loop and influenced it simply by walking through the space.

The signals interpreted by Introductions were sent to several destinations within the project. One of the Maths outputs controlled the switch that blended between the displaced signal and the feedback loop. The smoothed output from the Cascading Register selected which chaotic equation was displayed, while the signal from Uncertainty reset the equations to clear the buffer and start the process over. The value parameter on both HSV adjusts on the feedback loops were also modulated, as well as the displace weight on the displace TOP.







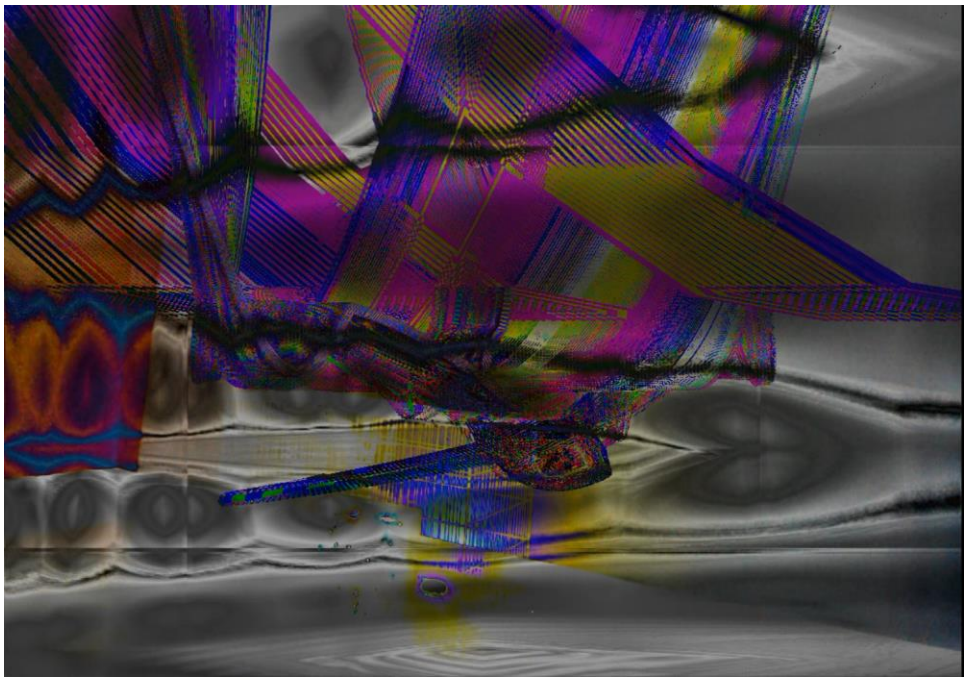


Figure 53-57: Stills from a Video Recording of the Installation

There were several issues that I ran into while setting up the installation. Originally, I had a webcam and a CCTV camera both connected to the patch to capture video from two angles, but the computer in the installation space did not accept two camera inputs. Several of the chaotic equations caused the program to crash and/or behave erratically. The sound interface in the space was not clearly labeled, and I had issues connecting the outputs of the modular synthesizer to the four-channel sound system. The projectors in the space were not as powerful or bright as the one I tested with, causing the image to be washed out when displayed. Despite these setbacks, the installation was successful in an artistic sense from my view.

You can see a video recording of the installation on my YouTube channel at this link if you're reading this online https://youtu.be/YqrN4Up7czA or by searching 'omiindustriies Feedback Loops All the Way Down" if you're reading a printed version.

Chapter 6 Conclusion

The universe tends towards entropy, becoming more and more disordered as it continues to spread out. Chaos is all around us, influencing our daily lives. If you forgot to turn the lights off when leaving the house, and go back to turn them off, you might be delayed in your journey to work to narrowly avoid an accident on the freeway; one example of how small changes to the initial state of your day may result in large changes as you go about it. The weather, the contractions of our heart muscles, the fluctuations of stock prices, all examples of chaos that influence our everyday lives.

Is there determinism in the universe? I would argue that the trends of entropy knock off any sort of predefined fate that may lie in store for humans. Small perturbations echo out of even the smallest of choices, spreading like ripples. These ripples in the pond of reality come into contact with the 8 billion other ripples that each human contributes to the overall direction of the flow of this metaphorical water. Constantly shifting, we influence each other in a species-wide feedback loop.

However, some things that do appear to us as random and disordered have an underlying order to them. The human mind and imagination, which sometimes seem unbounded, have limits to what we can conceptualize. We can know something is true, but sometimes it just doesn't feel on an emotional level to be accurate. As I mentioned, a 32-bit LFSR would take years to repeat itself if you left it running 60 times a second. Sometimes, what we perceive as random simply has to do with the limits of human perception; numbers and timescales that defy our conceptions of reality. Chaos, as I've explored, has an underlying determinism, rules that define its behavior. There is a certain amount of unpredictability, these rules appear simple but occupy the realm of complexity that baffle and intrigue many researchers and artists.

Modular synthesizers as they exist today are uniquely suited for harnessing chaos for artistic applications. Eurorack, in particular, is ripe for experimentation due to the plethora of manufacturers in the marketplace. There is a small and dedicated contingent of makers that seek to delve into the realm of chaos, producing modules that lie outside the confines of traditional synthesis. You couldn't dedicate the capabilities of a factory city in China to produce chaotic synthesizers, there's just not the market for it.

Composing with modular synthesizers sometimes is more of a two-way street. Instead of the composer strictly inputting their compositional style onto sheet music or a DAW, a linear input to output, a modular synthesizer may behave in ways that seem erratic or esoteric, nonlinearly creating music. This is, of course, dependent on the modules used, as some are particularly well suited for imposing a compositional paradigm onto them. My compositional processes, however, are dependent on a give-and-take relationship with the electronic synthetic behavior of chaotic and pseudo-random elements. Can electricity have a will? Are components imbued with the powers of thought? Is my modular synthesizer alive in some non-organic sense? In a scientific sense, probably not; but giving up some artistic will to a machine, a non-human entity, can result in surprisingly organic feeling outcomes.

Is there inherent meaning in this exploration of the unpredictable? I'm not sure, but perhaps the process is the meaning. Taking the time to look deeper into some of the underlying systems that exist both in the real world and the abstract, delving into the depths of esoteric electronics and complex simplicity. Perhaps this thesis has raised questions in you, the reader, about things you may have taken for granted or have not examined. I hope to have left you with both conclusions and curiosity. There is no end to this process, no final questions explored completely, no grand unifying truths. Simply, I leave you with the words of James Gleick:

"Where chaos begins, classical science stops."

Bibliography

Barton, Todd. “Buchla Architecture.” Synthesizer Resource. *Todd Barton* (blog), n.d. https://toddbarton.com/2014/02/buchla-architecture/.https://toddbarton.com/2014/02/buchla-architecture/.

Crab, Simon. “120 Years of Electronic Music.” WordPress Blog. *120 Years* (blog), 2021 1996. https://120years.net/wordpress/.https://120years.net/wordpress/.

Crutchfield, James P. “Space-Time Dynamics in Vide Feedback.” *Physica D: Nonlinear Phenomena* 10, no. 1–2 (1984): 229–45.

Electronics Tutorials. “The Integrator Amplifier.” Educational Resource. *Electronics Tutorials* (blog), n.d. https://www.electronics-tutorials.ws/opamp/opamp_6.html.https://www.electronics-tutorials.ws/opamp/opamp_6.html.

Fitch, Andrew. “Nonlinear Circuits.” Store and Documentation. *Nonlinear Circuits Modules* (blog), n.d. https://www.nonlinearcircuits.com/modules.https://www.nonlinearcircuits.com/modules.

Fritz, Ian. “Chaos: General Remarks and Implementation.” *Chaos Theory for Synthesizers* (blog), 2007 2006. http://ijfritz.byethost4.com/Chaos/ch_close.htm.http://ijfritz.byethost4.com/Chaos/ch_close.htm.

Gillet, Émilie. “Marbles Manual.” Github Manual Repository. *Mutable Instruments* (blog), n.d. https://pichenettes.github.io/mutable-instruments-documentation/modules/marbles/manual/.https://pichenettes.github.io/mutable-instruments-documentation/modules/marbles/manual/.

Gleik, James. *Chaos Making A New Science*. 2nd ed. Penguin Books, 2008.

Neal, Meghan. “The Many Colors of Sound.” *The Atlantic* (blog), February 16, 2016. https://www.theatlantic.com/science/archive/2016/02/white-noise-sound-colors/462972/.https://www.theatlantic.com/science/archive/2016/02/white-noise-sound-colors/462972/.

Nolte, David. *Galileo Unbound*. Oxford University Press, 2018.

Ovid. *Metamorphoses*. 1st ed. Vol. Book 1. 15 vols. Boston: Cornhill Publishing, 1922. http://data.perseus.org/citations/urn:cts:latinLit:phi0959.phi006.perseus-eng1:1.5-1.88.http://data.perseus.org/citations/urn:cts:latinLit:phi0959.phi006.perseus-eng1:1.5-1.88.

Random Numbers with LFSR (Linear Feedback Shift Register) - Computerphile. Youtube Video. Computerphile. Computerphile, 2021. https://youtu.be/Ks1pw1X22y4.https://youtu.be/Ks1pw1X22y4.

Rapp, Bastian. *Microfluidics: Modelling, Mechanics, and Mathematics*. Elsevier, 2017.

Rob Hordijk Rungler Demo // Modular Meets Leeds 2017. YouTube Video. Modular Meets Leeds 2017, n.d.

Rössler, Otto. “An Equation for Continous Chaos.” *Physics Letter* 57A, no. 5 (May 27, 1976): 397–98.
Slater, Dan. “Chaotic Sound Synthesis.” *Computer Music Journal. MIT Press* 22, no. 2 (1998): 12–19.
Sprott, J.C. “A New Chaotic Jerk Circuit.” *IEEE Transactions on Circuits and Systems_II:Express Briefs* 58, no. 4 (April 2011): 240–43. https://doi.org/10.1109/TCSII.2011.2124490.https://doi.org/10.1109/TCSII.2011.2124490.
———. “Common Chaotic Systems.” Blog. *Common Chaotic Systems* (blog), April 18, 1998. https://sprott.physics.wisc.edu/chaos/comchaos.htm.https://sprott.physics.wisc.edu/chaos/comchaos.htm.
———. “Some Simple Chaotic Flows.” *Physical Review E* 50, no. 2 (August 1994): R647–50.
Strange, Allen. *Electronic Music: Systems, Techniques, and Controls*. 2nd ed. Responsive Ecologies Lab, 2022.
This Equation Will Change How You See The World (The Logistic Map). YouTube Video. Veritasium, 2020. https://youtu.be/ovJcsL7vyrk.https://youtu.be/ovJcsL7vyrk.
Weng, Cho Chew. “ECE 255, Diodes and Nonlinear Circuits.” Purdue Engineering, January 18, 2018. https://engineering.purdue.edu/wcchew/ece255s18/ece%20255%20s18%20latex%20pdf%20files/ece255Lecture_4_Jan18_Diode_Nonlinear_Circuit.pdf.https://engineering.purdue.edu/wcchew/ece255s18/ece%20255%20s18%20latex%20pdf%20files/ece255Lecture_4_Jan18_Diode_Nonlinear_Circuit.pdf.
Whitwell, Tom. “17 Things to Know about Turing Machine.” Synthesizer Documentation. *Music Thing Modular* (blog), n.d. https://www.musicthing.co.uk/Turing-Machine/.https://www.musicthing.co.uk/Turing-Machine/.