EMID Project Report 2: Synth-esthesia

Kym, Jacob, and Trevor

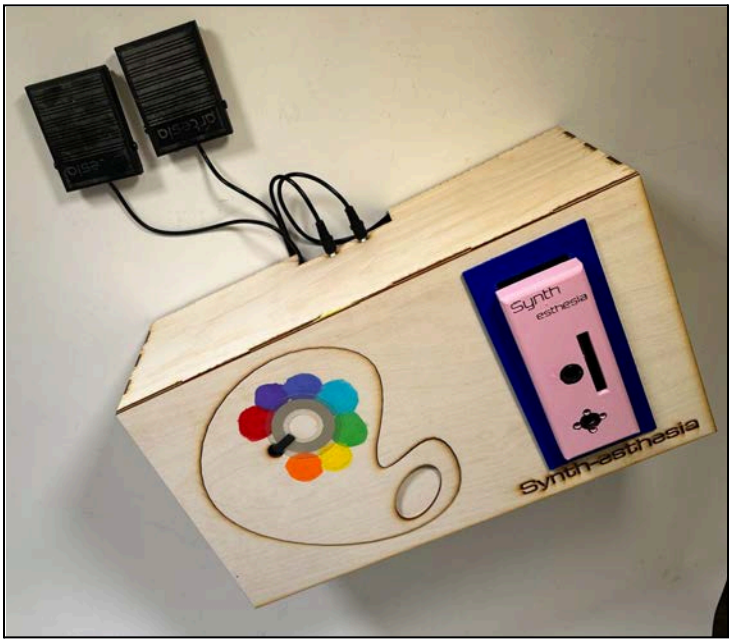
Dec. 14, 2025

Introduction

"Synth-esthesia," is a visually and sonically rich musical instrument that allows artists to blend the worlds of painting and music making. Inspired by the multisensory experience of synesthesia, our instrument is geared towards visual creators who wish to bring sight and sound to life within each other.

Design Goals

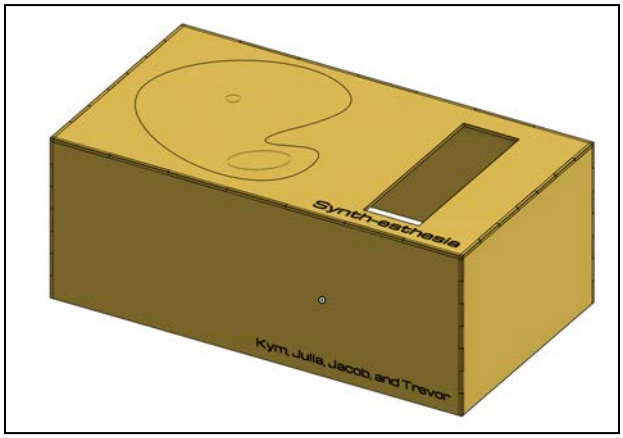
Our goal was to create an instrument that allows users to draw on a digital canvas and simultaneously embed a musical or sonic composition. For phase 2 of this project, we aimed to expand on the capabilities of our prototype and improve the prior issues and bugs we experienced. The prototype consisted of a painter's desk with a circular softpot acting as a color palette and a Wii remote that controlled our first sound bank. The Wii remote was able to create colorful drawings on an LCD screen through the programming of our Max msp patch. This was done with the up, down, left, and right arrow keys for movement and the B button to enable drawing. This was generally successful, but we still wanted to incorporate foot pedals, three more soundbanks and, varying canvases, and more tools for expression. With our final design, we kept the original painter's desk and color palette builds, but created a more elaborate project. In the end, we created a PCB with a 3D-printed casing that allowed users to paint wirelessly to four contrasting digital canvases programmed in Max msp and corresponding sound banks built in Reason. Expression is also extended with foot pedals that enable drawing capabilities and change the pen size and brightness of the color hue selected from the color palette softpot.



Physical Construction

<u>Laser-cut box & color palette:</u>

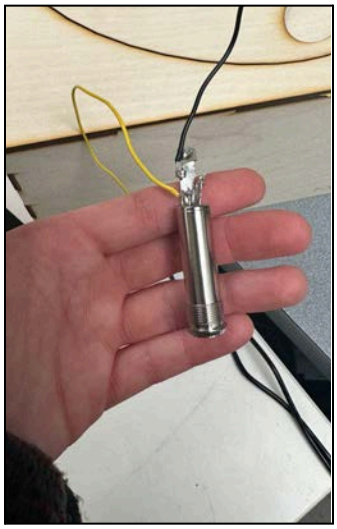
The desk design was 3D modeled and laser cut to include a holding place for the color palette and paintbrush. It was fabricated by

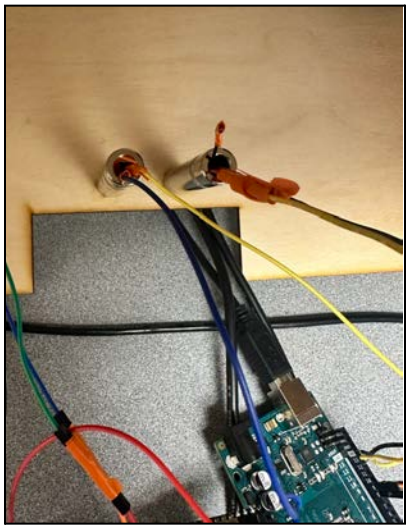


laser-cutting birch wood and using wood glue to assemble. The top of the box is double-layered so that the color palette is detachable and can be carried as the user is engaging with the instrument, along with the brush. The corresponding colors from the rainbow are painted around the softpot to visually signal to users what color they are selecting. The connector pins of the softpot feed through the hole in the middle of the palette into our Arduino hidden under the box. To help eliminate extra noise from the softpot that we were receiving in our Max msp programming, we used a 10k ohm pull-down resistor on the middle source pin of the sensor. This improved our reading by some degree, but the color readings were still somewhat noisy. Possibly a different sensor could have been more suited for this purpose (more details in forward looking section).

Foot Pedals:

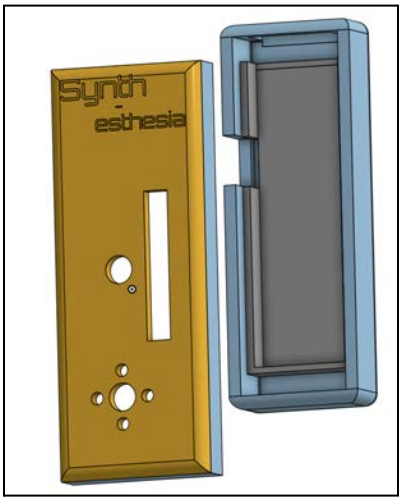
For our foot pedals, we soldered TRS female connector jacks to wires that connected to our Arduino, and laser cut two holes into the back of the box so that we could screw in the connectors to the box, and the user would only have to plug in the TS cables connected to our foot pedals to these jacks. Since the foot pedals were TS and the connectors TRS, we just soldered the hot and ground pins and left the cold pin open. This allowed us to get a "1" in our Max patch through the Arduino when the pedal is pushed down.





PCB casing/paintbrush:

Due to the new addition of our PCB to phase 2 of this project, we needed to create some type of casing that allowed for the two buttons and linear slide pot to be accessible and the four LEDs to shine through. We also wanted something that would be ergonomic and fit easily into the human hand. After a few iterations of designs, we decided to model our brush into more of a

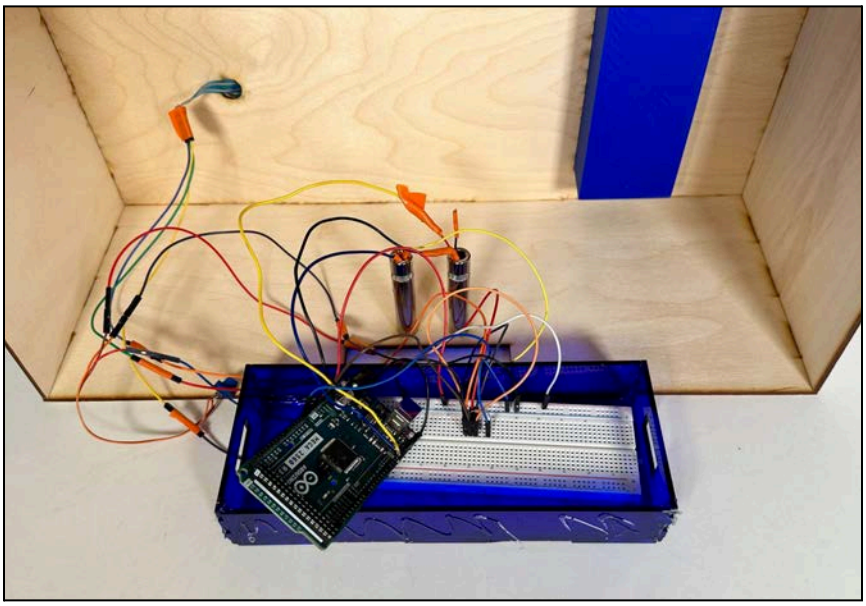


pen shape and followed the inspiration of how a Wii remote works as an interface. Since our PCB is battery-powered, we also needed to be able to remove the top of the casing to switch and charge the battery. With these considerations in mind, we modeled four separate parts–two for the bottom and two for the top. The bottom includes an outside shell (similar to the size of a Wii remote) with an internal sleeve properly sized for the PCB to sit snug inside. The top was designed with the correct holes to allow the necessary components through with an extruding bottom parameter that snapped on to the bottom. After CAD-ing, we 3D printed each of the four parts and assembled them together using hot glue where needed.

Electrical Components

Arduino:

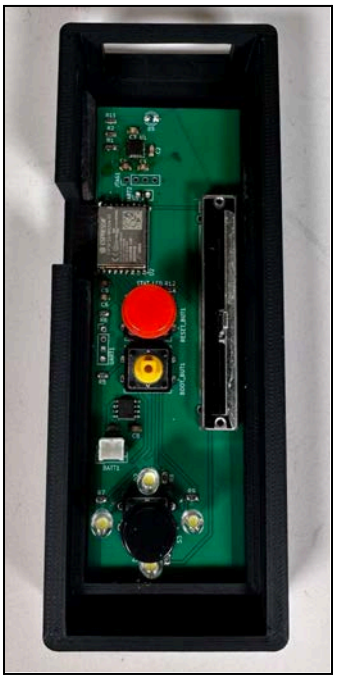
Our circular softpot and foot pedals were connected to an Arduino Mega under our box and routed to our Max programming. The pins of the softpot and foot pedals are going directly into the pins of our Arduino so that we receive a continuous stream of data from a finger on the softpot and receive a 1 when a foot pedal is pressed. As mentioned previously, we used a 10k ohm pull-down resistor to streamline the data from the softpot. Also, since the



linear slide potentiometer on our PCB failed before the project demo, we added a rotating potentiometer and similarly connected it to the Arduino to control pen size and brightness.

PCB:

In order to account for the failure of the Wii remote's orientation and movement data to connect to our Max patch, Jacob designed a custom PCB with an embedded IMU that would allow us to achieve our original goal of painting wirelessly on a digital canvas with 3 axes (pitch, yaw, and roll) of movement reflected in the visualizer and sonic feedback. As Jacob modeled the intricacies of the circuit and wrote the code needed to connect the PCB successfully, Trevor and I decided what controls would be necessary and desirable for this highly immersive instrument. To us, the most important components were that there would be a clear function, a way to shuffle between sound banks and canvases, and a slider for pen size and brightness. We had more ideas about how other sensors, such as an infrared sensor or a rotary encoder, could be used to further manipulate sound, but we decided to focus more on making these crucial functionalities work well and communicate effectively between our different systems. With this, we were left with two buttons and a linear slide potentiometer on the PCB, as well as the embedded IMU and other necessary components to support these functions.



Parts Used

Part	Quantity
Arduino Mega	1
ESP32-C3-Wroom-02 CPU	1
MPU-6050 IMU	1
Slide potentiometer	1
Circular soft potentiometer	1
Foot pedals	2
TS cable female sockets	2
Rotating potentiometer	1
Custom PCB	1
White LEDS	4

Rotating potentiometer	1
3.7 Lithium Ion Battery	1
3.7 to 3.3 Linear Voltage Regulator	1
Custom SMD Buttons	3
SMD 0805 Resistors and Capacitors	Many components with many different values
PLA/TPU filament	Several meters
⅛” birch wood	A few square feet

Max MSP

Our Max patch consists of four digital canvases, each one with its corresponding sound bank (*waterfall*, *cityscape*, *geometric*, and *atmosphere*). These are selected using the LED cycling button on the remote, and the corresponding window is opened on the main screen. A central design goal was that the performer should never need to look “behind the scenes” of the patch while playing. Instead, the user interacts solely through physical controls (hands, feet, and body orientation) and receives feedback through a single active visualizer displayed on the screen. The active visualizer and sound bank are selected using an LED-cycling button on the handheld remote, and only the selected visualizer window is shown on the main screen at any given time.

In the top left corner of the main patcher, **udpsend** and **udpreceive** take in four data points from the remote’s ESP32 microcontroller: IMU pitch angle, IMU yaw angle, the currently active LED (which determines the bus), and the linear sliding potentiometer value. Below this, **serial** logic from the Arduino2Max patcher reads values from the circular soft potentiometer and the foot pedal. These inputs are routed into shared logic that scales the raw sensor data into parameters usable by both the Jitter visualizers and the MIDI note output logic.

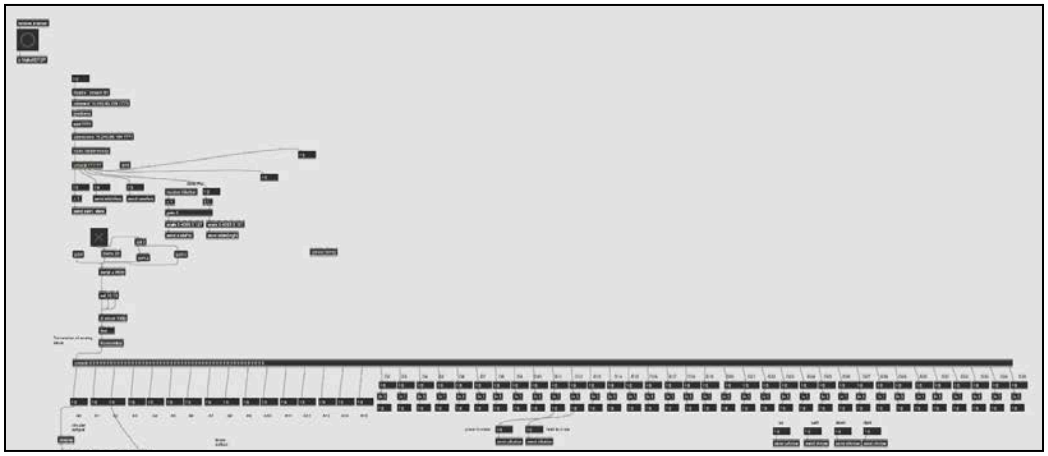


Figure 4: ESP32/Arduino Read-In

The circular softpot is wired to the color selection logic, enabling the visualizer to smoothly transition across the color wheel from red to violet. The sensor range is first mapped to a continuous range from zero to 2π using **scale**. This value is then sent through trigonometric functions using **expr** to turn this single value into three RGB values between 0 and 255. A separate linear slider, mapped to the sliding potentiometer, scales the brightness of the output color from black to fully vibrant. This final set of values is **pack**ed and sent via **prepend frgb** and **prepend color** to the two different kinds of **jitter** objects used by the visualizers. This approach allows color to change continuously rather than in discrete steps, creating fluid visual transitions.

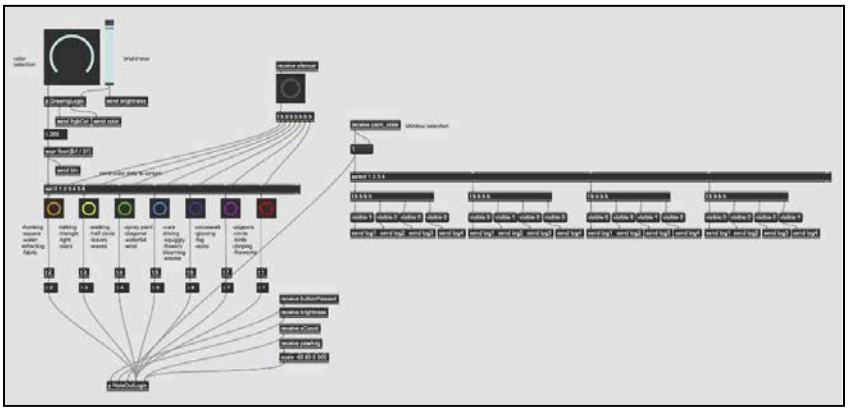


Figure 5: Color Selection Logic and NoteOutLogic Inputs

For playing music, the color selection logic is evenly partitioned into red, orange, yellow, green, blue, indigo, and violet bins, one for each of seven channels on the four soundbanks. The **mousedrawing** subpatcher sends its x-location to a **scale** from 0 to 127, which sends this value to a **bendout** object to modify pitch bend based on position. The button press logic is sent to a **noteout** command, which simply plays a middle C (note 64) on the color-selected Reason patch. Velocity is mapped to the linear slider that controls note brightness, such that black is velocity zero and fully vibrant is velocity 127. See Reason Patch section for more information on the NoteOutLogic subpatcher.

Bus A: Waterfall

Figure 6: Bob Ross Visualizer

The Bob Ross visualizer is based on Max Basic Tutorial 9: Mouse Drawing.[1] A **jit.lcd** object in the patcher is mapped to a **jit.window bobross**, with external arguments *pensize*, *frgbCol*, *bButton*, *yawAng*, and *pitchAng* controlling pen size, color, whether the cursor is drawing, and X/Y cursor location, respectively. The **mousedrawing** subpatcher is built foundationally from the Max tutorial, so that the mouse can be used to draw while debugging. Functionality was added to enable a virtual "mouse" to be controlled using the pitch and yaw angle of the IMU as X/Y coordinates, and the foot pedal as the "draw" button. When the cursor leaves the screen to the left, right, up, or down, it reappears on the opposite side of the screen and continues drawing. The *xCoord* cursor location and *buttonPressed* logic are sent from the **jitter** visualizer to the note out logic. The slide potentiometer on the remote controls brush size.

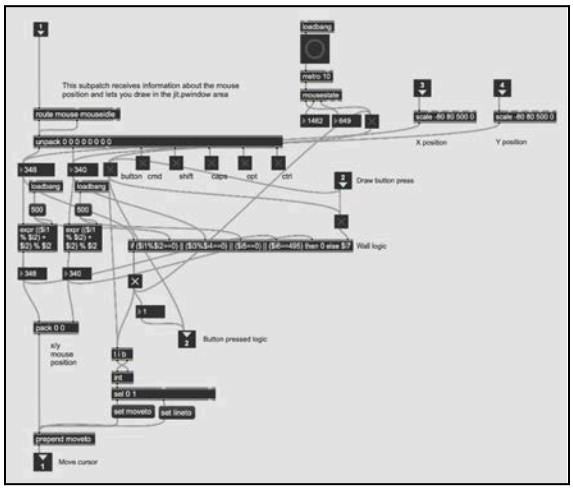


Figure 7: **mousedrawing** Subpatcher

Bus B: Cityscape

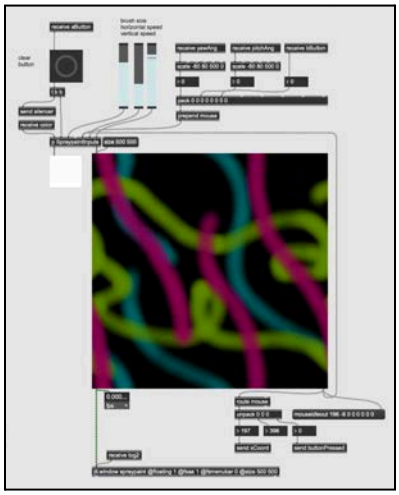


Figure 8: Spray Paint Visualizer

[1] Cycling '74. *Max Tutorial 9: Mouse Drawing*. Cycling '74 Documentation, https://docs.cycling74.com/legacy/max5/tutorials/max-tut/basicchapter09.html.

The Spray Paint visualizer was based on Max Jitter Recipes Book 4, Recipe 44: ScrollyBrush.[2] As in the Bob Ross visualizer, a **jit.lcd** object is mapped to a dedicated **jit.window spraypaint**. External arguments control brush size, horizontal and vertical speed, cursor position, and drawing state. The **SpraypaintInputs** subpatcher again retains mouse functionality for debugging purposes, while adding IMU-based control for performance. Pitch and yaw from the IMU control cursor position and motion speed, and the foot pedal toggles drawing. Compared to the Bob Ross visualizer, this patch emphasizes motion and momentum rather than direct stroke placement. The x-coordinate cursor location and button press logic are sent to the note output system, for audiovisual coupling. The sliding potentiometer continues to control brush size.

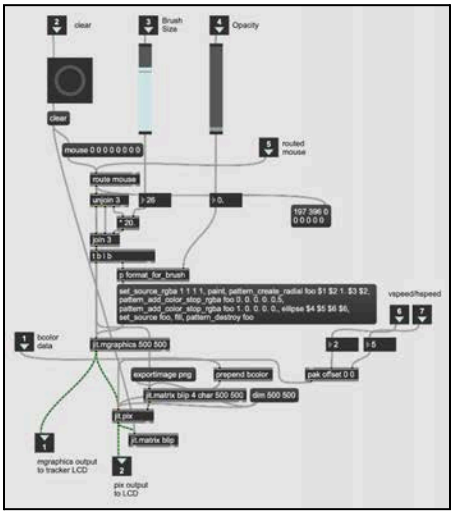


Figure 9: **SpraypaintInputs** Subpatcher

Bus C: Geometric

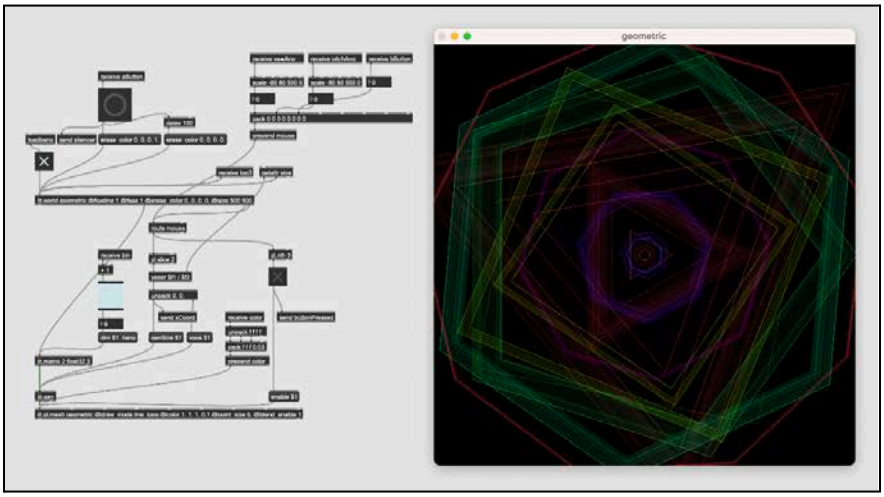


Figure 10: Geometric Visualizer

[2] Cycling ’74. *Jitter Recipes Book Four*. Cycling ’74 Articles, https://cycling74.com/articles/jitter-recipes-book-four.

The Geometric visualizer was based on HearingGlass's YouTube video "Generative Geometry - Max/MSP Tutorial."[3] **jit.world** and **jit.gen** are used to generate and draw a line-loop mesh whose geometry is procedurally updated every frame, creating layered rotating polygonal shapes. X/Y cursor location controls the radius and rotation angle of the shapes while the draw button is being held down. A trailing, kaleidoscopic visual effect is produced with alpha blending enabled and a low opacity value for the input lines. Holding in one spot then lets the shape slowly materialize from the black background.

Bus D: Atmospheric

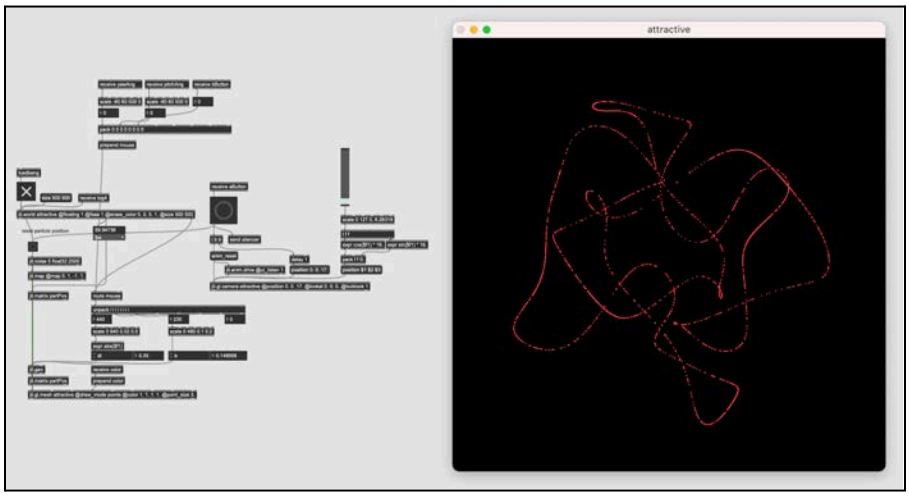


Figure 11: Atmospheric Visualizer

The Atmospheric visualizer was based on HearingGlass's YouTube video "Chaotic Particles - Max/MSP Tutorial."[4] It implements a dynamical systems physics simulation in **jit.world** where particle positions are iteratively updated inside **jit.gen** using the Thomas attractor differential equation (see below). Virtual mouse input is routed and scaled to control key parameters of the system, the time step *dt* and coefficient *b*, reshaping the attractor's trajectory in real time. A noise-based initialization seeds particle positions, and this continuous feedback based on the differential equation causes them to trace chaotic, looping paths. Rendering uses **jit.gl.mesh** in point mode with blending enabled, producing dotted trails that accumulate visually over time. The slide potentiometer on the remote controls the orbital angle of the 3-dimensional **jitter** object.

[3] HearingGlass. *Generative Geometry – Max/MSP Tutorial*. YouTube, https://www.youtube.com/watch?v=eMfxO_PxZlk.
[4] HearingGlass. *Chaotic Particles – Max/MSP Tutorial*. YouTube, https://www.youtube.com/watch?v=xBM9C9ADK8w.

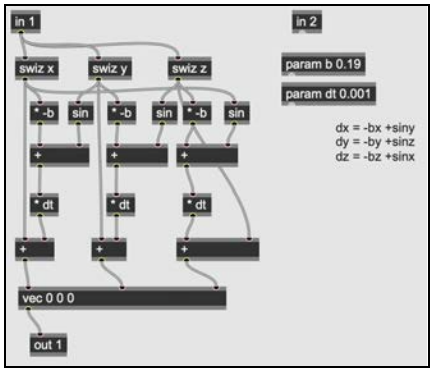


Figure 12: **jit.gen** Object

$$\frac{dx}{dt} = \sin y - bx, \qquad \frac{dy}{dt} = \sin z - by, \qquad \frac{dz}{dt} = \sin x - bz$$

Figure 13: Thomas' Cyclically Symmetric Attractor Differential Equation[5]

Connection to Reason:

In order to support the functionality of having 4 different soundbanks with 7 sounds each, the Reason patch was set up with 28 independent instruments split among 4 buses. In order to enable Max to send data to 4 different buses, 4 virtual buses were enabled in the Mac's IAC Driver application, labeled *s*, *t*, *u*, and *v* in Max. In Max, each **noteout** is triggered with **gate** logic to the proper bus and channel based on the remote's LED selector and the circular softpot color selector, respectively. A **noteout** with velocity zero is sent when the virtual mouse button is released, and **bendout**s are sent to the corresponding instruments based on the scaled X-coordinate of the cursor. In Reason, then, we four soundbanks, each inspired by a specific visual artwork or conceptual environment, encouraging users to "paint" immersive sonic scenes. The banks are organized in order of increasing abstraction and visual complexity.

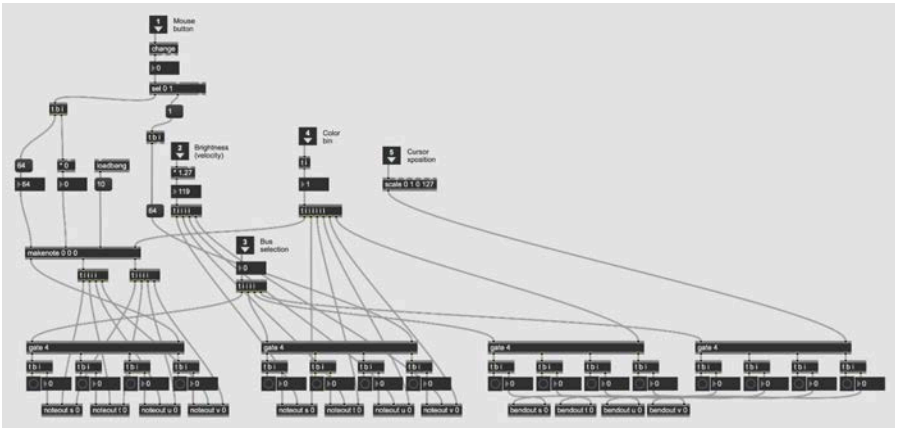


Figure 14: NoteOutLogic Subpatcher

[5] Wikipedia contributors. *Thomas' Cyclically Symmetric Attractor*. Wikipedia, https://en.wikipedia.org/wiki/Thomas%27_cyclically_symmetric_attractor.

The “clear” button was mapped indirectly to the “reset” button on the PCB. When this button is pressed, the IMU recalibrates and sets yaw and pitch angle to zero. When yaw and pitch angle are both set to zero, which only really happens during recalibration, Max logic **bang**s the “clear” button of all four visualizers, opens the Bob Ross visualizer again, and **bang**s the MakeItSTOP subpatcher, which shuts all of the notes off. Initially, the MakeItSTOP subpatcher utilized **uzi** objects to send **noteout**s with velocity zero to all notes, channels, and buses. However, this proved to be glitchy in Reason, so we pivoted to using **uzi** to bang each channel and send a **ctlout 123**, which worked better.

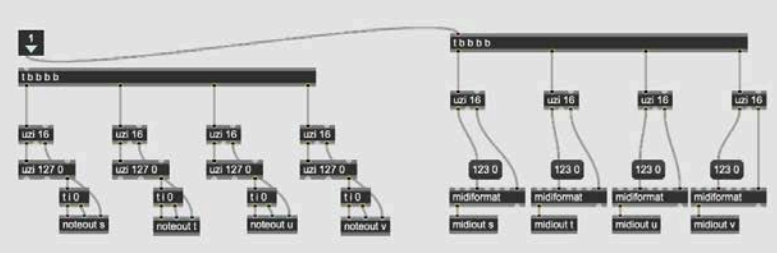


Figure 15: MakeItSTOP Subpatcher

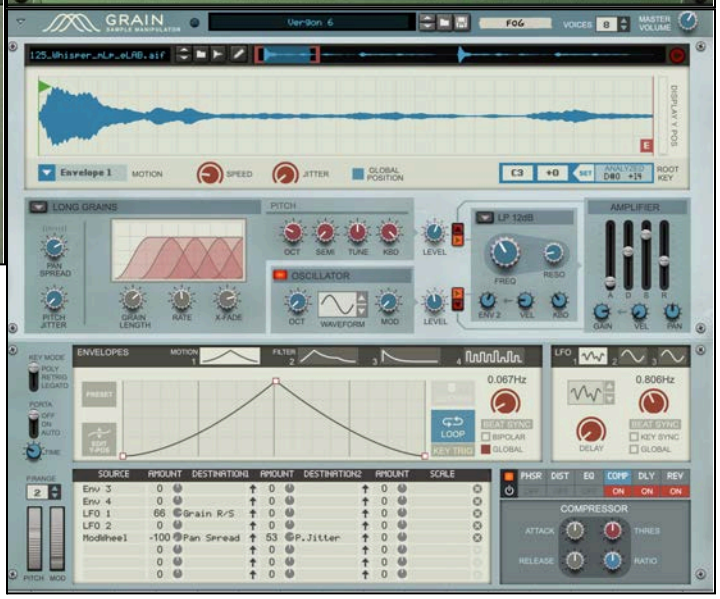
Reason

For our final design, we used four different sound banks made up of seven sounds each, one for every color of the rainbow. Each of these banks was inspired by a painting or concept. Our first bank is called *Waterfall* and is based on Bob Ross’, “Mountain Waterfall” (1980). The second is called *Cityscape* and was created to showcase the sounds of an urban landscape in a musical way. *Geometric* is our third bank, inspired by Vasily Kandinsky’s “Composition 8” (1923). Lastly, our fourth bank is titled *Atmosphere* and actualizes the elements of James Abbott McNeill Whistler’s painting, “Nocturne in Black and Gold: The Falling Rocket” (1875). The idea was to allow users to build their own unique sonic landscapes with these expressive sounds. Each soundbank was mapped to the four Reason buses (A, B, C, and D) respectively. This allowed us to then separate each of the sets of seven channels so that they could be independently triggered in Max depending on which bank is selected. For a majority of the sounds, we modified the amplitude envelope by raising attack, sustain, and release so that there were minimal perceivable start and end times to each sound. However, this took some This allowed for an enveloping of a sonic landscape rather than simply playing notes and hearing them back for a short moment. With this same evolving sound idea, we did not want our instrument to be constrained by expected or “pleasant” harmonic relationships, but still wanted to incorporate some type of pitch mapping to the canvas. We mapped pitch bend to the x-axis of our canvas so that movement left to right correlates to a continuous increase in pitch. This allows for microtonal pitches to play and more experimentation with pitch relationships. In all of the modules, I decreased the pitch bend range to 2 octaves to limit the number of possible pitches played.

<u>Waterfall/Bus A</u>

1. Red - Birds Chirping
2. Orange - Water Reflecting
3. Yellow - Sparkling Light/Shining Sun
4. Green - Rustling Leaves
5. Blue - Waterfall
6. Indigo - Flowers Blooming
7. Violet - Fog/Clouds on the Mountains

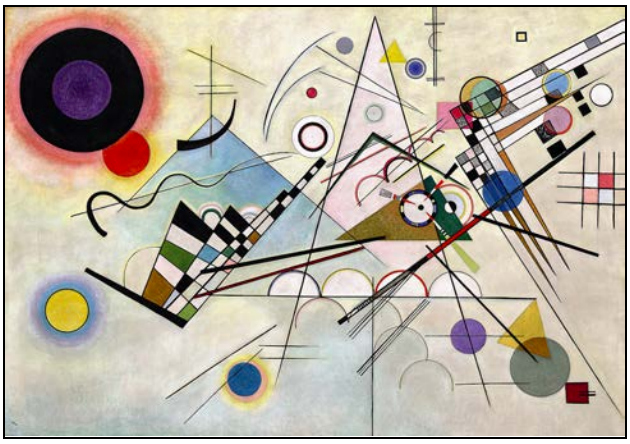
For the sounds that I wanted to resemble more realistically such as *birds chirping* and *waterfall*, I used the NN-19 sampler and loaded in samples from a free online source called *Pixabay*. I used this same strategy for the following sound banks and included a variety of these realistic sounds versus more symbolic and evolving sounds. With the NN-19, I cut the samples into my desired length (usually about a half a second to a second long), selected “solo sample” to solo my loaded-in sample, and modified the amplitude envelope to suit the sample. I also found Malström Graintable synthesizer to be a good module for creating more realistic sounds with further adaptive controls, as it allowed for the synthesis of two different patch sounds to generate more complicated waveforms. For our more ambient sounds, I found that the Grain Sample Manipulator (for this bank, and later I also expanded in subsequent sound banks to add Europa Shapeshifting synthesizer and Thor Polyphonic synthesizer) module provided complex evolving soundscapes that helped the remainder of the sounds blend together. Specifically, I used this module for the *sparkling light/shining sun* and *fog/clouds on the mountains* sounds as seen to the left.



Cityscape/Bus B

1. Red - Pigeons
2. Orange - Cars honking
3. Yellow - People talking
4. Green - Walking in snow
5. Blue - Spray paint
6. Indigo - Cars driving by
7. Violet - Walk signal sound

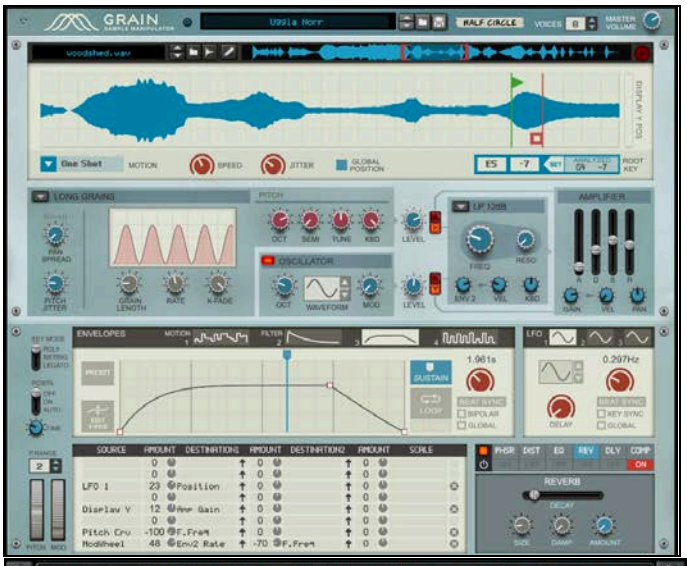
Similar to the strategies used in *Waterfall*, I used many recorded samples from *Pixabay* and loaded them into the NN-19, also modifying the sample length and amplitude envelope. This sound bank included the most 'realistic' samples in order to emulate the sounds one may here in an urban setting. For many of the samples I sourced, I had to meticulously edit the amplitude envelope and actual cutout of the imported sample since looping the sound would cause unwanted hanging notes in Max. I found that it was important to make sure that the sustain and release values were not completely at 127, although having relatively high values for these parameters did allow for sounds to blend better with each other. For other sounds, such as *spray paint* and *cars driving by* I decided to use Grain and Malström to achieve more symbolic representations of what it may sound like if a person were using spray paint to create a graffiti mural in the distance and how the Doppler effect changes the way sound reaches the human ear as a car drives by.



Geometric/Bus C:

1. Red - Circle
2. Orange - Square
3. Yellow - Triangle
4. Green - Half Circle
5. Blue - Diagonal Lines
6. Indigo - Squiggly Lines
7. Purple- Glowing

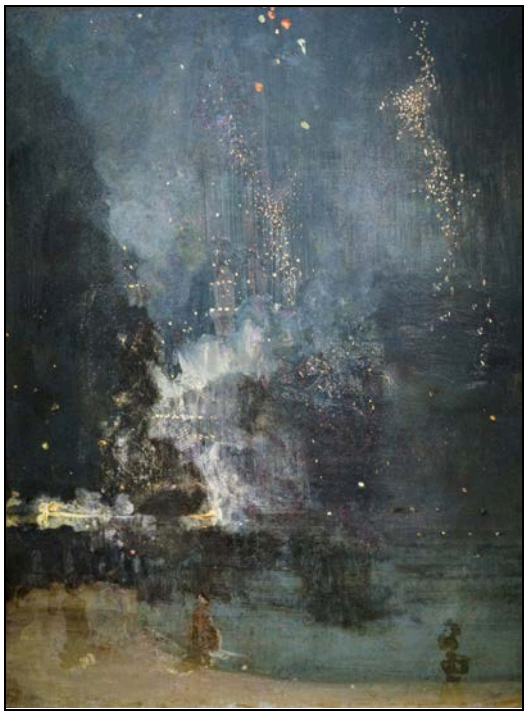
For the Geometric bank, I wanted to create sounds that would emulate different shapes and elements as depicted in the Kandinsky piece. Furthermore, I attempted to build each shape/sound by somehow incorporating the shape into the synthesizer. For example, I used a triangle-shaped envelope for *triangle*, an LFO for *squiggly lines*, and sine wave oscillators for *half circle*. I found that granular synthesizers such as Thor, Europa, and Grain were the most suitable for this bank since I wanted to layer multiple oscillators and generate sounds that aurally resembled classic granular synthesizer sounds.



Atmosphere/Bus D:

1. Red - Fireworks
2. Orange - Fabric Rustling (woman's dress)
3. Yellow - Sparkling Stars
4. Green - Waves
5. Blue - Wooshing Sky/Wind
6. Indigo - Smoke/Phantom
7. Violet - Mysterious Gong

Finally, the last soundbank was created to be the most mysterious of them all. I wanted to include very abstract sounds in this bank to complement the corresponding chaotic particles as seen in the



Atmospheric visualizer. With inspiration from Whistler's painting, I used a combination of all the modules previously utilized to generate a complex array of evolving and generally ambient sounds. For this bank, it was the most difficult to eradicate hanging notes, as many of the modules looped and sustained samples for longer periods of time to create an evolving effect. By manipulating other parameters of the modules, such as the amplitude envelope, reverb, and delay.

Looking Forward/Conclusion

For a future design of this instrument, I would propose adding more opportunities for manipulating sound, such as adding more sensors to change resonance, LFO, overall volume, and panning. This way, more parameters of sound could be controlled and users would have more creative autonomy as to how the sounds develop. I would also polish the physical design,

especially the paint brush case, and embed a charging port within the brush so that users could just plug it in instead of changing the battery. Overall, the process of making this instrument was very rewarding and helped me learn about the areas that my partners focused on. The complexity of our system and its reliance on Max and wireless communication tested our debugging and problem-solving skills heavily. In the end, I feel that we were able to successfully reach our goals and produce a creative, complex, and playable instrument that has virtuosic capabilities. In the future, I would suggest more in-depth planning and prototyping before building and connecting all of the necessary components, but I feel that our group did a great job of collaborating and meeting consistently to work on the project over time.