California Institute of the Arts

Sound Is Everywhere

by
Scott Cohen

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

Herb Alpert School of Music
Music Technology: Interaction, Intelligence & Design
2015

Supervisory Committee

Ajay Kapur

Michael Leisz

Kai Luen Liang

Cristian Amigo

Sound is everywhere.

This thesis is a simplification. It's taking a complicated process and trying to make it as easy as possible. A physical space can be defined with infinite variables, a snapshot in time defined as a point with infinitesimal decimals. Live spatialized sound performance is almost entirely limited to the high academic level. There are some exceptions to this such as Max Cooper's use of Paul Oomen's *4DSOUND* or Suzanne Ciani's performances in quadrophonic sound on her Buchla. But for the most part, performances that are sound in space are limited to installations and conceptual ideas that are supposed to challenge the listener to think and to criticize.

Long gone are the days of cavemen playing on bone whittled proto instruments with modern amplifiers. We can generate pressure strong enough to simulate the force of a rocket ship taking off. Average hobbyist music producers can now tune their rooms with nothing but a microphone and a laptop. Computers can iterate on scales so large that the speed of sound is no longer a hindrance on the world's largest stages. LEDs have gotten so advanced that screens the size of buildings and spotlights strong enough to shine into space captivate the eyes of concert goers at your popular nightclub or basic music festival. And yet, sound in its natural form, surging all around us, is excluded from the capitalism on our senses.

As an electronic music performer have played on large stages and small, basements and festivals. I have played projects so complicated that no matter how much I rehearsed I could learn something new each time I opened my laptop, and some so simple that nothing could go wrong. As a producer, I've spent hours listening to white noise, buying plugins, taking classes, and learning from mentors to hone my ears. In my life, I've built hundreds of guitar amplifiers in the pursuit of finding the ultimate tone. But one day as I sat outside on a hiking trail to identify birds for a theater class, my ears were opened to the waves of sound around me.

Parameters can be set for volume, for filters, for electrical current, for water, for lights, even for other parameters. But when I get up on stage to perform, when I write music in my studio, I still can't control *where I'm listening from* unless I physically move myself. Modern science has proven that sound can rewire the brain, and life is a sound bath. A sound bath is a conglomerate of instruments, melodic and percussive, arranged in a circle around the listeners. How can I perform my music like that?

So, I decided to make it simple, like listening to the birds, so simple that anyone can do it. Throw it all out and use your ears. Take the math and cut it to pieces. Grab the bits and treat them like they're alive, let the sounds fly around freely like the birds. One parameter that doesn't require automation or modulation that can take my sounds and set them free.

Acknowledgments

Not in any particular order, I'd first like to thank my family. My father for always lending me his sage advice and providing me with the confidence and emotional maturity to try and achieve my goals. My mother for constantly surprising me with her caring and never take no for an answer nature, reminding me to focus on my goals and to put myself first. And I would like to thank my future wife, Anais for her never-ending support and curiosity. Without her I wouldn't have had someone to explain my wacky ideas to, providing a necessary means to trouble shoot. And I'd like to thank her parents for being patient with me and strong supporters of my artistic nature and always lending a helping hand whenever necessary.

I'd like to thank Amy Knoles for making a mistake during my audition regarding the location I was supposed to set up, which allowed me to enter Cal Arts. My good friend Sacha Robotti for giving me a platform to work on and a recommendation to start my education. And I'd like to thank my piano teacher, Tata for her unending patience in listening to and dealing with my "mashed potato fingers" when I was preparing for my audition.

I'd like to thank Kai Luen for being the inspiration that pushed me to change majors. An incredible teacher and a friend who is not only talented but supportive and treats everyone with an insatiable kindness. I'd like to thank Michael Leisz with whom I've shared a multitude of wacky conversations. A mentor who not only humored my good days but also my bad. Mike provided a place to talk safely about anything, whether we got down to coding or just talked about issues. Without Mike my experience at Cal Arts wouldn't be the same. I'd also like to thank Jake Penn for answering my emails during time that he wasn't teaching courses at Cal Arts, when I was frantically grasping at straws to understand basic concepts. And I'd like to thank Zane Golas his TA for spending so many hours with me to understand the concepts we covered in class.

I'd like to thank Cristian Amigo for giving his students an environment to just think, to learn about sound from a perspective that isn't often offered. Without Cristian I wouldn't have found my direction. I'd like to thank John Tejada for being a grounding human being who always found a way to make assignments fun and add a little bit of reality to the other dimension that Cal Arts is.

And finally, last but most definitely not least, I'd like to thank Ajay Kapur. Not just the head of the program, but the teacher who asked me when I first joined Cal Arts if I might want to change majors. Ajay, who when I did ask to change programs made sure to remind me of what kind of effort it would take instead of just saying yes. Thank you, Ajay, for bringing out the best in your students, myself included.

Contents

List of Figures

Chapter 1
Evolution Of Electronic Spatialized Sound

The Spatialized Sound Institute in Hungary refers to performing in multi-dimensional listening as “an emerging area of study.” [1] This is simply not true. We have been reaching up towards this ceiling for as long as we have been listening to sound with electric transducers. However, there is something to be said regarding the performance of music composed for spatialized sound as novel. The inventions of the vital technologies associated with transduction and amplification arose from the minds of scientists, not artists.

The invention of the speaker created a new necessity. The amplification of recorded sound as a product to consumers. On top of that, the concept of sound in space was approached early in the journey towards modern sound systems. Additionally, technologies seem grow more in a marriage than when left alone and the modern developments in amplification and spatialization came through sounds application to movies. Just like many of the best 20th century composers worked for movie studios, most of the best sound engineers worked for the likes of Disney or Cinerama.

Much of what we take for granted today, was invented by a brilliant mind nearly one hundred years ago. Even multi-dimensional listening has a history nearly as long as that of the loudspeaker itself. Starting from the beginning:

Sound is everywhere.

1.1 The Invention of The Loudspeaker

The first loudspeaker was invented as a means for humans to communicate with each other over distance. Installed in a telephone in 1861, Johann Philipp Reis had created the loudspeaker. [2] Unfortunately, it couldn’t produce much more than sinewaves and muffled speech. Fifteen years later, Alexander Graham Bell patented the first loudspeaker capable of producing human speech.[3]

[1] “About — SSI.”

[2] “The Forgotten Johann Philipp Reis - Integrated Network Cable - National or Local IT Rollouts and Installation Servcies.”

[3] “The Auxetophone.”

1.1.1 Bonus: The First Use of Amplified Sound Spatialized

In 1881, Clément Ader created the first spatialized sound system referred to as the Théâtrophone. Its invention came less than 4 years after the patent of the first working loudspeaker. It goes to show how ingrained into our brains it is that sound must exist in three dimensions. The Théâtrophone used multiple telephone receivers placed in a row to capture a form of binaural stereophonic sound. It was able to broadcast Opera performances in stereo using two headphones. [4]

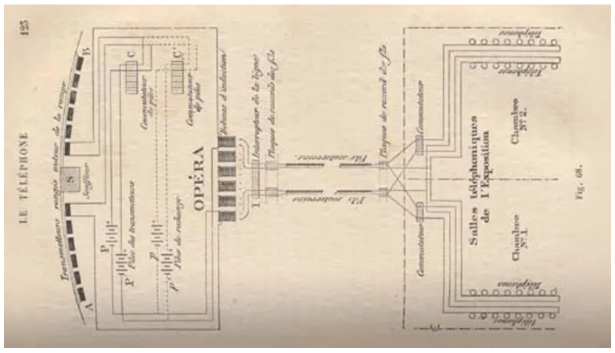


Figure 1.1: Theatrophone Schematic[5]

The Theatrephone was commonly called the Electrophone during the time it was used in London. It arrived in the UK in towards the end of 1893 and was a subscription-based telephone service that provided the listeners with headphones to catch their favorite "programs" before the invention of modern telephone cable networks. [6] However, starting at fifty dollars per year, this kind of service was primarily for the wealthy.[7] This is demonstrated through its use for a children's choir to sing happy birthday to Queen Victoria on her final birthday before her passing in 1903.[8]

1.1.2 Turn of the century inventors

In 1874 Ernst W. Siemens imagined the idea of a "dynamic" speaker, better described as a moving coil transducer. While he was unable to fully realize his idea, Siemens did file for a patent of a "magneto electric apparatus in 1874. And later, in 1877 his patent for the

[4] *Scientific American 1925-09.*
[5] "Du Moncel - Système Ader."
[6] "The Electrophone (1893)."
[7] "Electrophone in England (1901)."
[8] "The Queen and the Electrophone (1899)."

parchment diaphragm became what would be the most widely used invention in modern speakers. Despite this the invention of the dynamic loudspeaker was first filed for as a patent by Oliver Lodge on April 27th, 1898. [9]

Peter L. Jensen, along with Edwin Pridham, Jensen began to manufacture moving-coil speakers and marketed them for radios and public address systems under the name *Magnavox* in 1915. This marketing strategy began the steady push towards having electrified entertainment in the average household. [10] However, it would still be a few years before the invention of what we now know as the modern loudspeaker.

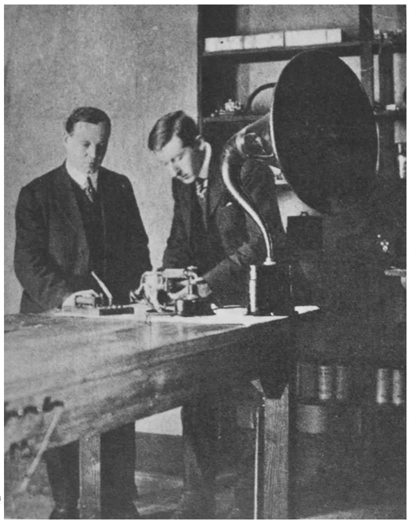
F

Figure 1.2 Magnavox

C.W. Rice and E.W. Kellog created the first modern loudspeaker prototype in 1921. They filed for a patent in 1925 and sold it commercially for the first time as the Radiola Loudspeaker #104 under the company Radio Corporation of America (RCA). RCA had been created under the pressure of the U.S. government as a collective research company to develop better technologies. The use of electromagnets was more widespread at the time because permanent magnets weren't available at a reasonable price. Also, the electromagnets served as both the choke coil and the power supply filter. [11]

1.1.3 Advancement in Theater Systems

In the 1920s commercial theaters used large single cone electromagnet speakers, and some minds had the opinion they needed advancement. This advancement would arrive in 1933 in the form of John Hilliard's Shearer Horn.

[9] "Loudspeaker History."
[10] "Timeline."
[11] "History and Types of Speakers."

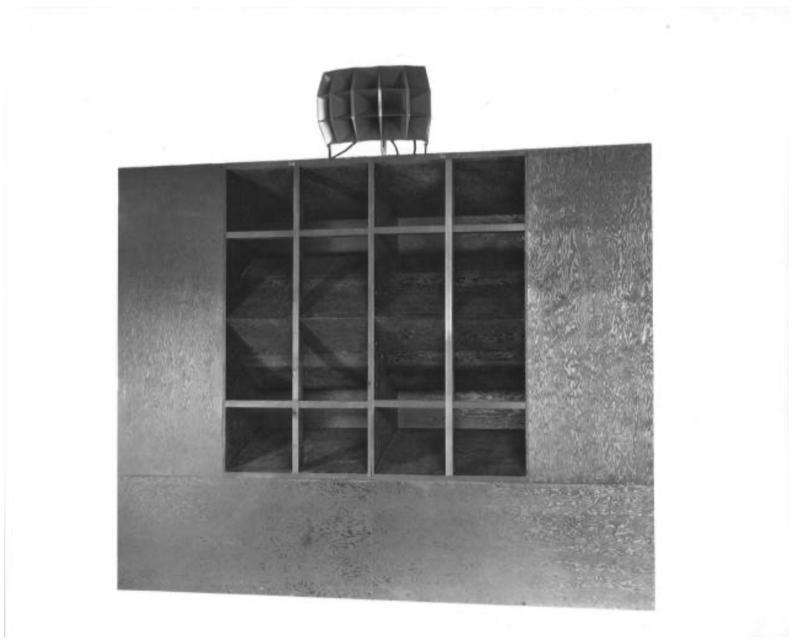


Figure 1.3: The Shearer Horn[12]

Hilliard, an acoustic engineer at Metro-Goldwyn-Mayer (MGM) began a collaboration with Jim Lansing and they met with the head of MGM's sound department, Douglas Shearer. Hilliard and Lansing proposed the creation of a sound-system produced by MGM and Lansing Manufacturing. The system would be the first design to use multiple drivers. The speaker used four fifteen-inch low frequency drivers and two compression drivers for the high frequency. This would be the first time that the phrase "crossover frequency" would be used as the speaker had a crossover network at 375hz.

Over the course of the next three years, the Shearer Horn would be demoed at the Capitol Theater on Broadway which would lead to the purchase of twelve separate systems for additional theaters. By 1936 the Shearer Horn would be the industry standard for speaker manufacturers. The Shearer Horn was also the recipient of the technical achievement award in 1936, celebrating its achievement in progressing the motion picture industry, by the Academy of Motion Picture Arts and Sciences. [13]

Sound is one of the primary forms of human communication.

1.2 Imagining the Stereo-field

1.2.1 Alan Blumlein

Around the same time Lansing and Hilliard were pioneering the quality of sound, a man named Alan Blumlein was reimagining the way we listened. Blumlein, annoyed at current

[12] "SHEARER HORN."
[13] Nordost, "The Evolution of HiFi."

monophonic speaker systems not providing the effect of distance when actor's spoke on screen began his foray into developing what we now know as modern stereophonic sound.

In 1931, Blumlein, a British engineer at EMI, filed his patent for *Improvements in and relating to sound-transmission, sound-recording, and sound-reproducing systems* on December 14th [14]. His patent provided the first take on the "Shuffler System," envisioning a stereo system as not just Left and Right but also as Middle and Sides. He came up with the equation $M = L+R$. While this sounds like something that is standard for all audio engineers today, understanding that Left and Right as a sum equal the sound in the middle wasn't a common idea back in the early 1930s. Treating all the signal processing as one total signal was monumental to the creation of spatialized sound. Using this equation, Blumlein was able to recover both the left and right fields of sound through the additional equation $2L = M + S$ and $2R = M – S$ where L is Left, R is right, M is middle, and S is sides. [15]

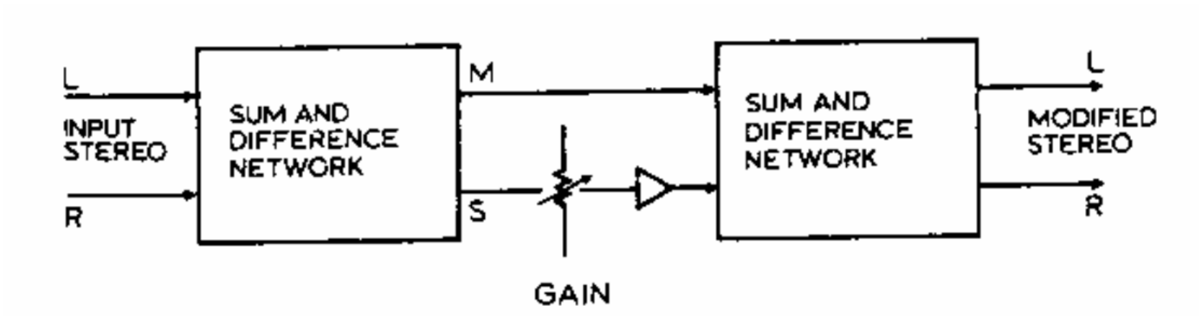


Figure 1.4 Basic Shuffling System

Part of the patent also described the use of dual microphone's in opposing right angles next to each other, which to this day in audio engineering is still known as the Blumlein Pair or Blumlein Technique. This recording technique records a representative and authentic performance that can be replicated in the shuffle system to produce an accurate stereo field. [16]

[14] "Espacenet – Search Results."
[15] "Stereo_shuffling_A4.Pdf."
[16] McDermott, "What Is the Blumlein Technique?"

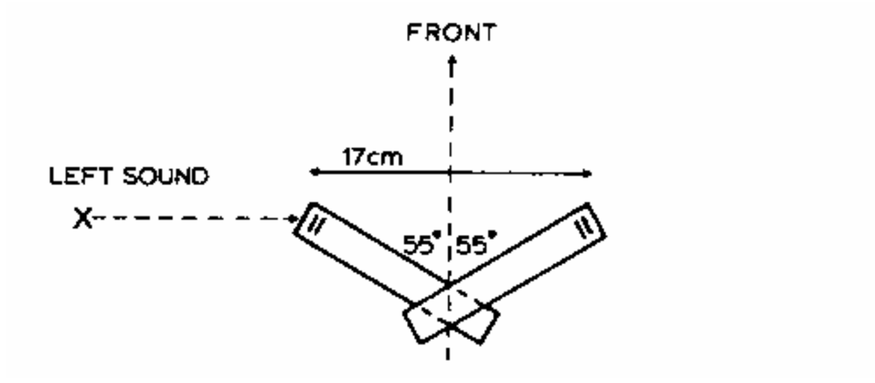


Figure 1.5 Blumlein Techique

Although much of what Blumlein intended to accomplish wouldn't be realized until the year 1935, the first stereo records were cut and distributed by 1933 using Blumlein's recording techniques. The printing method used the walls of a single groove, perpendicular to each other to have dual sides read by a special head. In 1934, Abbey Road studios recorded, Mozart's Jupiter Symphony in stereo, for the first time. [17]

1.2.2 The United Stereo of America

Meanwhile, at Bell Telephone Laboratories, an American by the name of Harvey Fletcher was researching solutions to stereophonic sound. Unlike Blumlein who used a calculated and mathematical approach, Fletcher began investigation through the "wall of sound." The initial testing used a staggering eighty microphones, each corresponding to its own unique loudspeaker in a listening environment.

Eventually this was reduced to two microphones, connected to two separate styli to cut out grooves on a wax recording. Funnily enough, although Blumlein was ahead of the curve in much of the pioneering of stereo sound, Fletcher achieved the first stereophonic recording in March of 1932 when he recorded Scriabin's *Prometheus: Poem of Fire* performed by Leopold Stokowski and The Philadelphia Orchestra. [18]

[17] "Abbey Road Studios Celebrate Alan Blumlein's Birthday."
[18] "Harvey Fletcher – the Father of Stereophonic Sound | SciHi Blog."

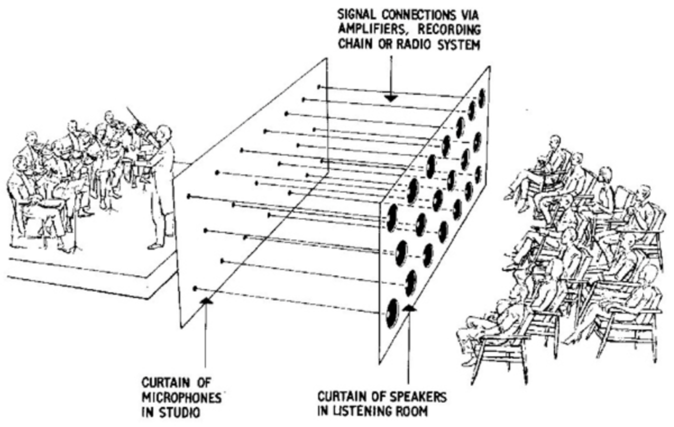
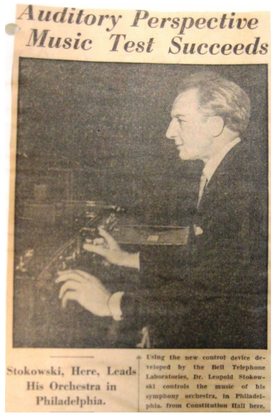

SIGNAL CONNECTIONS VIA AMPLIFIERS, RECORDING CHAIN OR RADIO SYSTEM
CURTAIN OF MICROPHONES IN STUDIO
CURTAIN OF SPEAKERS IN LISTENING ROOM

Figure 1.6 The wall of sound[19]

On April 27th, 1933, Fletcher and Stokowski demonstrated the first performance of their new recording techniques holding a preview for a total of three hundred people. However, this time Stokowski was not conducting the orchestra and instead was serving as the mixing engineer for a momentous broadcast. While Alexander Smallens conducted and Stokowski mixed the sound, Fletcher and his team of engineers broadcasted to a total of four thousand and five hundred notable individuals to demonstrate the new recording and listening technologies.

The Philadelphia Inquirer described the broadcast as "For those alive and alert to the significance of the episode, it took rank as an epochal event in the history of musical performances." Due to the success of the performance Stokowski continued his collaboration with Bell Labs throughout the 1930s and by the end of the decade had begun to apply the theory to film.


Auditory Perspective Music Test Succeeds

Stokowski, Here, Leads His Orchestra in Philadelphia.

Using the new control device developed by the Bell Telephone Laboratories, Dr. Leopold Stokowski controls the music of his symphony orchestra, in Philadelphia, from Constitution Hall here.

Figure 1.7 News Article of April 27th, 1933, Performance [20]

[19] "80 Years On & Counting: Progress In 'Getting It Right' With Speech Reinforcement - ProSoundWeb."
[20] "At The Academy, The Birth Of Wire Transmission Of Music."

1.3 Fantasound

1.3.1 Stokowski

During the 1930s, if one could believe it, Popeye had grown more popular than Mickey Mouse. Walt Disney, looking for a way to spring Mickey back into the limelight, had the idea to utilize composer Paul Dukas' *The Sorcerer's Apprentice* in a short called *Silly Symphony.* Funnily enough, the world of sound would be transformed permanently by a chance meeting of two brilliant minds.

On an evening like any other evening, Walt Disney was dining by himself and happened upon Leopold Stokowski, also on his own. Disney took the opportunity to invite Stokowski to eat with him and brought up the "Sorcerer's Apprentice." Stokowski, deeply interested, offered to conduct music for the film for free. Stokowski also suggested that perhaps the film should be a full-length feature. Disney ignored that. However, when the film's budget soon grew far beyond the original, to a staggering $125,000, Disney decided the only way to make the money back would be to produce a full-length feature film.

The film itself was met with harsh criticism during the production, Igor Stravinksy dismissed the visuals as "an unresisting imbecility" and that Stokowski's performance was "execrable." Critics were divided, some called the final project a "milestone in the motion picture." Franz Hoellering called it a "promising monstrosity." Fantasia ended up costing $2,280,000 to produce, which in today's terms would be worth close to $500,000,000. The film lost a lot of money during its initial release.[21]

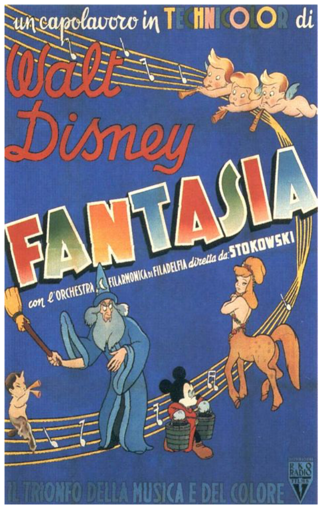

un capolavoro in TECHNICOLOR di
Walt Disney
FANTASIA
con l'ORCHESTRA FILARMONICA di FILADELFIA diretta da STOKOWSKI
IL TRIONFO DELLA MUSICA E DEL COLORE
RKO

Figure 1.8 Poster of Fantasia [22]

Fantasia, along with being the first recording to make use of a click-track or metronome for cinematic timing was also the pioneer for what we know of today as modern surround sound.

[21] Solomon, "Fantastic 'Fantasia.'"
[22] "Fantasia Movie Poster (#4 of 9)."

1.3.2 Fantastic Solutions equal Fantastic Problems

During the production of *Fantasia,* the primary problem faced by Disney and Stokowski was the limited volume range of recording technology at the time, and while recordings could be amplified, high volume playback was plagued by excess noise and distortion. Additionally, Disney wanted to create the sensation of Point of Source Sound. This meant that the center of the screen would be heard as directly ahead, while as objects moved across the screen their sound would also move along with them. In the past, using broad sound sources caused phase and the movement would have to be extremely gradual if it would ever be done at all.

The first solution that was reached by over half a dozen engineers and their teams was to create a two-gang volume control to attenuate circuits of two speakers on opposite ends of the room. The attenuations, expressed as power ratios would equal that of a constant, or:

$$\mathbf{A = \frac{B - 20 \log |2 \sinh 0.115B|}{2}}$$

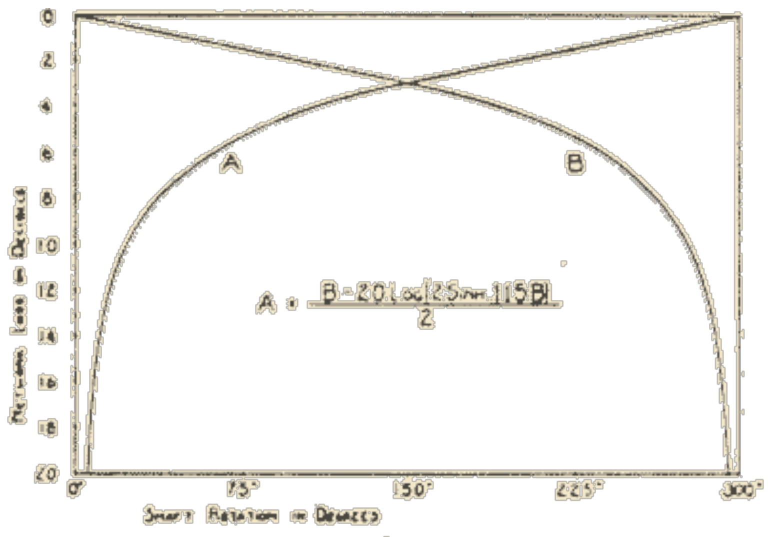


FIG. 1. Curves showing loss vs. shaft rotation of a typical two-circuit differential junction network. The cross-over point for the two losses is at −3 db.

Figure 1.9 Attenuation expressed as a constant

However, two speakers weren't quite enough, and the array grew to a set up of three separate tracks, which are represented as physical horns. This three-circuit set up was nicknamed "The Pan Pot," with pan standing for panoramic, and could be used to record or dub a recorded track to any or all the three Fantasound program tracks with a smooth transition at any desired volume level. In an even over simplified sense, it could be viewed as two sets of stereo pairs sharing a center channel. By using this technique, volume could move throughout the speaker array at a steady volume at any desired speed. It also had the unintentional effect of controlling recording reverberation and microphone pick up noise without effecting the desired output level.

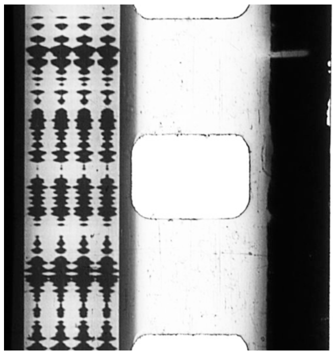


Figure 1.10 Image of Film with Optical Sound on the Left Side[23]

On the left of Figure 1.9, are four photocells which scan the three program tracks and a pilot control-track. A pilot tone is an extremely low frequency, small intensity modulation applied to an optical channel to identify channels as well as monitor power. [24] Each track feeds a variable gain amplifier (VGA) and then the three stage horns.

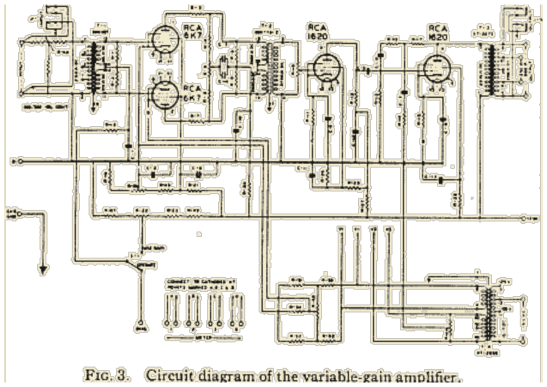


FIG. 3. Circuit diagram of the variable-gain amplifier.

Figure 1.11 Diagram of VGA

Each VGA was also granted its own separately biased tone rectifier and thus each speaker's volume can vary based on the amplitude of the associated control tone. This technology was applied directly into Fantasounds gain adjustment device dubbed the *Togad.*

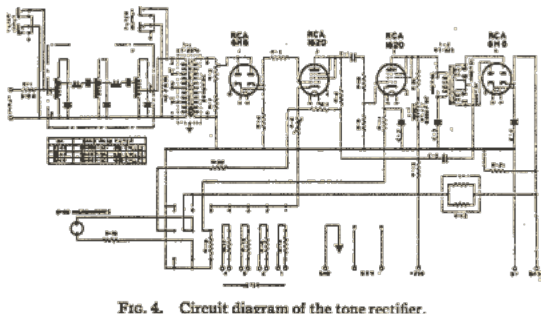


FIG. 4. Circuit diagram of the tone rectifier.

Figure 1.12 Diagram of Tone Rectifier

[23] "Optical Sound Examples."

[24] Jiang et al., "Progresses of Pilot Tone Based Optical Performance Monitoring in Coherent Systems."

The *Togad* is composed of two VGAs and the tone rectifier. The pilot control tone is converted into bias voltage and applied to the VGA and is configured so that a 1db change in control tone is a 1db change in program transmission via the VGA.

The Fantasound system went through eight different variations before being mass produced for the public. The Mark I system used three horns and two subwoofers, with two tracks, one feeding the center horn and subwoofers and the other feeding the side horns using a four-circuit differential junction network controlled manually. The Mark II system added three horns, one on each sidewall and one in the center, also controlled manually. Mark III maintained the speaker set up of Mark II but was the first iteration of the *Togad* expansion, and with *Togad* the myriad issues of crosstalk balance, amplitude tone characteristics, distortion, time constants and noise compromise began being attacked.

Mark IV used the same set up as the eight channel Mark II but was monumental in that *Togad* had officially replaced all manual control at this point allowing for what we know today as modern automation. This version used eight separate control tone tracks and was installed in Hyperion Studios for research. Mark V was an improvement on Mark IV, however it was only in service for one day because with the advanced development of the channel separation, none of the composers or mixing engineers could remember where each sound was supposed to go, and by the time they remembered Mark VI had been released, which reduced everything back to three horns, three program tracks and a 3 tone control tracks. This allowed for six separate mixing engineers to work during the dubbing of *Fantasia.*

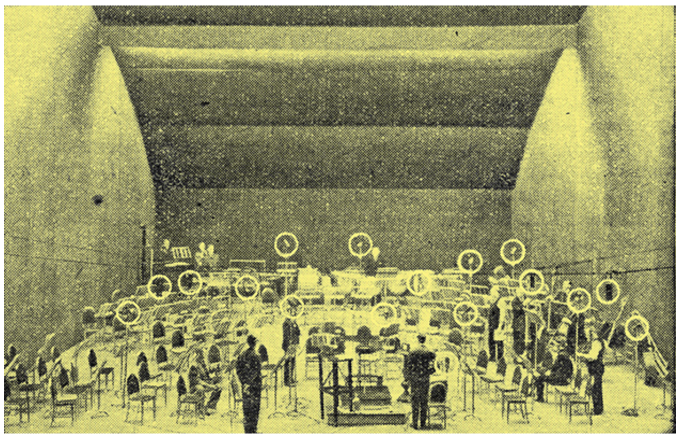


Figure 1.13 Fantasound Recording [25]

While Mark VII was not the final iteration of Fantasound, it was the first large scale manufactured version, it used a linear phase tone rectifier rather than logarithmic. [26] This was installed in multiple theaters across the United States. Fantasound continued to Mark X before eventually being discontinued. However, the seven-channel iteration of the Mark X was installed at Carthay studios, technically making it the first fully automated seven channel surround sound system.

*Fantasia'*s initial Fantasound release only ran from 1940-1941 due a multitude of hiccups. Theaters had to be shut down during the installation of the Fantasound system,

[25] "'Fantasound.'"

[26] "FANTASOUND*."

which caused many of them to decide against hosting the program. The production of the Fantasound system cost Disney and exorbitant amount for each theater as well.[27] Finally, with the breakout of World War II hurting foreign sales and Disney's animators going on strike, Fantasound was officially shut down.[28] The film was then mixed down into mono and sent out for national release.

You have never experienced pure silence.

1.4 Cinerama

The first of the many changes that occurred in film, and thus sound recording, after World War II was the beginnings of the switch from Optical Recordings to Magnetic Tape. In 1952, *This is Cinerama* debuted. Cinerama was filmed by synchronously recording on three 35mm cameras and shown using three separate projectors to create a Panoramic moving image. Fred Waller won an Academy Award of Merit for this creation. [29] On the edge of sound however, Hazard E. Reeves had found something entirely new. Instead of recording sound onto the optical tape along with the film, Reeves decided to switch to magnetic tape.

Reeves had begun his investigation into magnetic tape in 1943 when he noticed that the population had started to adopt an interest in listening to high fidelity sound. Magnetic tape allowed for better fidelity and more accurate representation of sound when played back. Magnetic film was first used in 1948 for Joseph Lerner's *C-man,* but it wasn't until *This is Cinerama* did Reeve's true talent shine. To Reeves it was a real challenge, but to the rest of the world it was the first step towards moving into multiple channel magnetic sound recording.

Reeves created a system that used five loudspeakers behind the screen and two speakers off to the side when played back, a layout like modern Sony Dynamic Digital Sound. However, the microphones could be placed anywhere of to the sides during recording to accomplish the same effect, moving sound from stage to offstage. Initially, Reeves technology was met with backlash from investors but after the debut of *This is Cinerama,* magnetic stereophonic sound was a reality.[30]

[27] "Fantasound," April 12, 2017.
[28] Cantrell, "The Making of Fantasia."
[29] "AFI|Catalog - This Is Cinerama."
[30] "Adding the Sound to Cinerama."

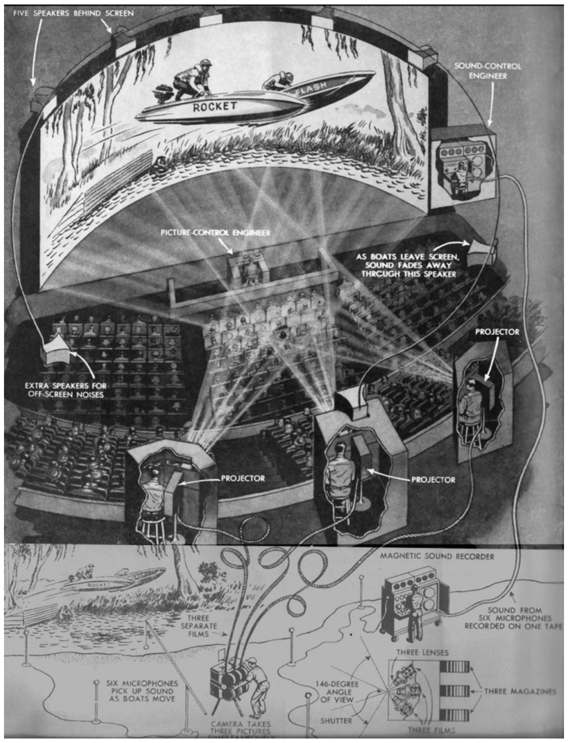

FIVE SPEAKERS BEHIND SCREEN
ROCKET
FLASH
SOUND-CONTROL ENGINEER
PICTURE-CONTROL ENGINEER
AS BOATS LEAVE SCREEN, SOUND FADES AWAY THROUGH THIS SPEAKER
PROJECTOR
EXTRA SPEAKERS FOR OFF-SCREEN NOISES
PROJECTOR
PROJECTOR
MAGNETIC SOUND RECORDER
THREE SEPARATE FILMS
SOUND FROM SIX MICROPHONES RECORDED ON ONE TAPE
THREE LENSES
SIX MICROPHONES PICK UP SOUND AS BOATS MOVE
146-DEGREE ANGLE OF VIEW
THREE MAGAZINES
SHUTTER
CAMERA TAKES THREE PICTURES
THREE FILMS

Figure 1.14 Cinerama[31]

Another up-and-coming company was Todd-AO, having initially been a part of the Cinerama company, Mike Todd followed a similar method. However, unlike Cinerama, Todd AO developed a camera that filmed onto a 65mm negative instead of a 35mm negative, so instead of requiring multiple projectors to play the film, only one specialized projector was required. The projector would run a single tape of 70mm magnetic film which would center it nicely on the screen. Additionally, Mike Todd, having come from a sound background, added a total of 6 separate soundtracks to each of his films that could be played simultaneously out of different loudspeakers on the remaining 5mm space left on the tape track.[32]

[31] "A Silver Screen Special Edition: Ladies and Gentlemen: This Is Cinerama."
[32] "The Todd-AO Process."

Chapter 2
Adventures into Modern Surround Sound

2.1 Post WWII Sound

2.1.1 The Little House in Gravesano, Switzerland

In 1954, Herman Scherchen turned things upside down. Scherchen was known as one of the most prominent conductors of the time, having founded one of the earliest known radio orchestras in 1928. He conducted his first radio broadcast concert in 1924 and was unhappy with the shortcomings of the technology. Scherchen himself had said "That was really the birth of my mission – the beginning of my search, which I am still carrying on and which sometimes I seem to have almost succeeded in realizing." The dream was: the acoustical existence of music in modern recording. [33]

In Gravesano, Switzerland, during the summer of 1954, Hermann Scherchen turned a barn at his summer home into an electroacoustic research center. After receiving funding from UNESCO, he invited some of the most promising musicians, technicians and theorists from Europe and US to hold a conference. A conference on music and technology isn't anything out of the ordinary, but what set this set of meetings apart was the focus: Spatial Electroacoustics.

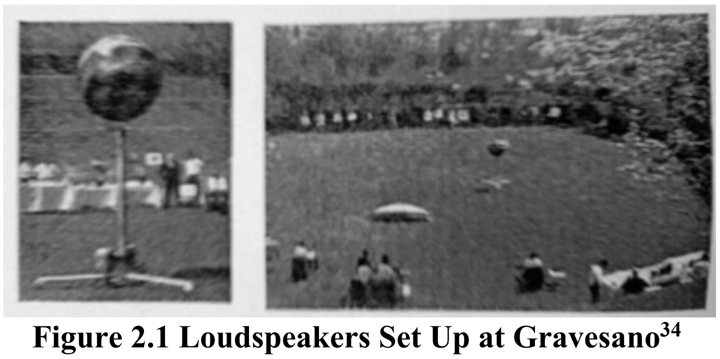


Figure 2.1 Loudspeakers Set Up at Gravesano[34]

[33] Bornoff, "Hermann Scherchen - Youth and Music / Hermann Scherchen — Jeunesse Et Musique / Hermann Scherchen — Jugend Und Musik."
[34] Eck, "An Active Loudspeaker by Hermann Scherchen."

The lectures and discussions focused on multi-projection film, broadcast demonstrations and concerts. Many of the engineers, such as Pierre Schaeffer and his tech Jacques Poullin from Paris and his counterparts from Cologne and Berlin had already begun to improve upon the systems developed by Bell Labs and Disney. The presentations themselves started the conversation that electroacoustic studios were not just capitalistic fancies for the masses, but also radically new structures and experiences.

Poullin himself gave a particularly relevant, mostly technically, presentation on what he referred to as a "space potentiometer." This device was not just a fader but a fundamentally new form of experience. It had already been implemented as a fixture in musique concrete, most famously in *Symphonie pour un homme seul* and *opera concrete, Ophee* in 1951 and 1953 at New Music Festival in Donaueschingen. The "space potentiometer" was developed not only as a spectacle, utilizing the conductor's movements to distract from the machinery on stage, but also to move sound from multiple tape decks into a quadrophonic listening experience for the audience, with two speakers at the front and two in the back.

Schaeffer and Poullin considered dynamic manipulation of the sound and its location in space to be revolutionary in the new era of art. However, Schaeffer was not interested in creating any form of realism, which he considered boring. Schaefer pursued the idea, that all sounds should be their own individual objects and through a spatial development they would be coextensive with their own forms.

2.1.2 Cage and Stockhausen

In 1951, the American composer John Cage began work on a collective tape music project and in 1952 seven tapes produced at Schaeffer's studio were delivered to Cage in New York. Cage had requested these tapes to fill up time during a concert of his at the University of Illinois. The concert was to be presented on eight speakers, all of which hooked up to their own mono tape machine to be arranged around the audience. All the pieces in the program were performed in either eight individual channels, some playing one track through all eight speakers and some of the performances simulated panning with volume knobs to send the sounds whizzing around the audience. The rig was originally conjured up for Cage's pieces *Octet 1* and more famously *Williams Mix.*

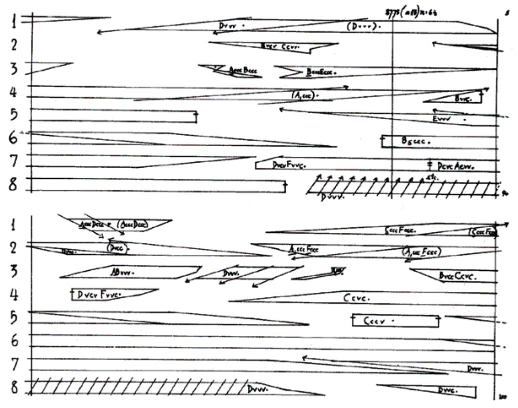


Figure 2.2 William's Mix[35]

Cage believed that multi-channel systems left no room for anything but immediate listening, however unlike his European peers he had found it hard to get funding for his work. Cage envisioned a concert hall with speakers on the ceilings, in the entryways and even on the roof, but art critics at Donaueschingen wrote his performances at the festival that year off as absurd, eccentric, and anarchistic. However, Cage was received at the WDR studios second concert two nights later where Karlheinz Stockhausen performed pieces along with Herbert Eimert.

Prominent figures Robert Beyer and Werner Meyer-Eppler, invited as guests the Gravesano congress, were cofounders of WDR studios that by 1954 had long been set up with multi-channel equipment. The facilities could be grouped into a three-channel arrangement, to mimic a standard PA system or each speaker used individually for varying effects. Beyer had written in 1928 that "at the end of development stands the space-intoning, space-filling, stationary sound, oscillating around a central core, changeable illuminated from a tone-color world of cosmic expanse, a sound almost visible." WDR had long been a firm believer that by bringing the score into real space through technology would be the revolution in listening.

Strangely, it wasn't until Cage visited WDR in 1954 that anyone at the studios had begun to actually work on creating a multi-channel composition. Stockhausen with the assistance of Gottfried Michael Koenig began work on a piece planning to use the available stereophonic resources. Although Koenig didn't personally take stereophony seriously, Stockhausen approached it with a form of reverence, referring to it as his next great task. Stockhausen believed that his piece *Gesang der Jünglinge* would herald in a 'new living art-form of musical composition and musical listening.' In this way, unlike much of the music concrete that came before, using the multi-channel design at Schaeffer studios, Stockhausen created the first multi-channel musical piece that was derived in its most basic structure only to be performed on multiple channels.

Although *Gesang der Jünglinge* would ultimately be reduced from six channels down to four, Stockhausen continued to lecture and publish works on multi-channel compositions. Stockhausen found that by de-spatializing his forebears and contemporaries he could start a new dimension in the modern music experience. [36]

"The tone's property of being generated in a definite space and in definite locations in this space has - disregarding the few aforementioned entries as exceptions - until now not been further differentiated at all: the tone location in the concert hall remained fixed (in front of the listener on the stage), not only during one composition, but for all music written until now, and it played no role in composition."

2.1.3 Philips Pavilion

Brussels 1958 saw the completion of the Phillips Pavilion in honor of the World's Fair and while the building itself was a marvelous feat of architecture, we're going to examine

[35] "LISTENING SESSION."

[36] Valiquet, "The Spatialisation of Stereophony."

something a little different. Inside the Phillip's Pavilion, were numerous performances including one concert showcasing a piece by Edward Varese. The piece was called *Poème èlectronique.* The piece itself was in a way inspired by Stockhausen's early spatialized electronic pieces like *Gesang der Jünglinge* and *Kontakte* but the application of it at the World's Fair was entirely different.

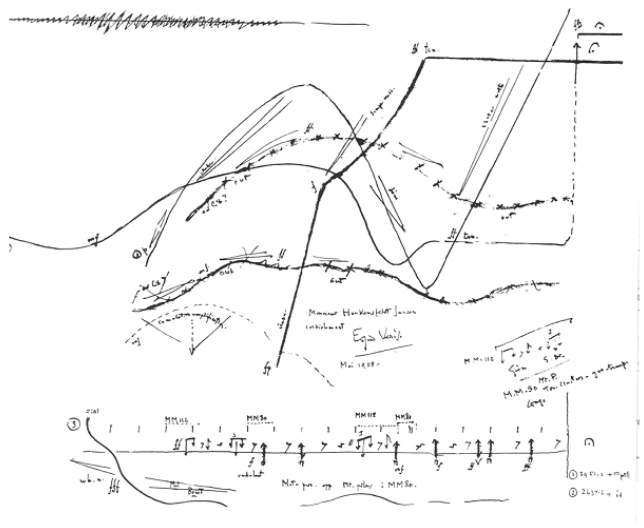


Figure 2.3 Spatialization of *Poème èlectronique*

The Phillip's Pavilion's interior had been constructed entirely out of asbestos and was thus extremely sound reflective due to its solid nature (and extremely carcinogenic). On top of this, there are estimates that Varese used upwards of four hundred loudspeakers during the presentation of his piece, a lowest estimate says at the minimum three hundred and fifty. The speakers were broken up into twelve separate banks of at least five individual lines. The sounds were moved around by a team of engineers using rotary telephone dials. While there are ideas of how the sound was spatialized, the only actual remaining musical score is above as Figure 2.3. [37]

2.1.4 The Vortex Concerts

Around the same time a Berkeley student named Jordan Belson began another historic foray into the spatialized sound installation world. At the California Academy of Science's Morrison Planetarium, Belson held the first Vortex Concert on May 28th, 1957. Belson hosted music from composers such as Henk Badings, Gordon Longfellow, David Talcott and Karlheinz Stockhausen. The planetarium had a total of thirty-eight speakers in a circular configuration and was able to handle many of the early spatialized pieces. Additionally, Belson would project psychedelic kaleidoscopic patterns on the planetarium ceiling to go along with the music to create a full immersive experience.

[37] "ANALOG Arts » Recreating the Philips Pavilion."

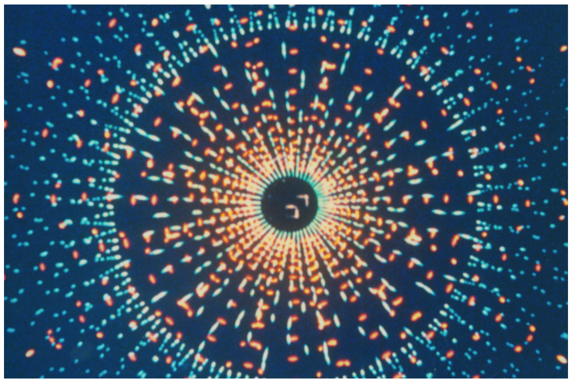


Figure 2.4 Example of Belson's visual work

The name Vortex was derived from the ability of Belson to move the sound around the dome in both clockwise and counterclockwise motions, at whatever speed desired. This was advertised as an entirely new aural experience. To accomplish this, Belson had hired an electronics expert to design him a rotary mechanism with one handle so that he could manually swing the sound around the room to achieve what he called the Vortex Effect.

The experience was an immediate success and became immensely popular, exceeding all the attendees' expectations. Not surprisingly, the management had to work around the academics and the unruly "bohemian" audience members who caused some trouble initially. The domed theater of sound and light ran for over two years and thirty-five performances. There had been initial plans to take the experience overseas, but it never came to fruition and while Belson had tried to recreate the experiences he made at the Vortex series later in life, he feels that it was impossible to ever recreate something of that caliber. [38]

2.2 Dolby Myth

2.2.1 Apocalypse When?

Apocalypse Now's soundtrack is lauded as one of the most influential pieces of mixed audio ever produced. The foley was so detailed that it led the movie along as if it was a soundtrack, and the only thing that reminds you of the lack of modernity is the actual musical score itself. The realism in the sonic field was so huge that the jump in audio quality and composition was so great that it shamed all movies that came before it, setting a new standard for every movie that came after it. However, it wasn't an isolated incident, unbeknownst to even the creators themselves *Apocalypse Now* had been in gestation since the days of Cinerama and Todd AO.

Ray Dolby, the inventor of Dolby's A-type noise reduction, claimed that he had personally solved the issue of playback noise himself, but that couldn't have been further from the truth. During this time high frequency surface noise during recordings, particularly with vinyl records, remained an issue. So, the industry had devised a technique. As early as the late 1940s, studios had begun to adjust recording levels in a "pre- and post-equalization"

[38] Keefer, "'RAUMLICHTMUSIK' - EARLY 20TH CENTURY ABSTRACT CINEMA IMMERSIVE ENVIRONMENTS."

process in which they manually recorded high frequencies at much louder volumes and then reduced the gain to temper the high frequency noise pollution. This process reduced noise and optimized a film's dynamic range.

What Dolby had achieved was an automation of this process. The Dolby system divided the original signal into four individual frequency ranges, increased the high frequencies during recording and then reduced them in playback effectively reducing hiss. Additionally, Dolby's split-surround sound invention had also been pilfered, or inspired by at the very least, *This is Cinerama* and Todd A-O.

Dolby stored extra surround tracks on film prints with only one extra track for surround options. Cinerama had originally used this format so that they would only have to produce one type of film roll for all theaters and so theaters wouldn't have to keep extra inventory if they had rooms with surround sound and some without. Also, Dolby, used control tone, like the original experiments with *Fantasia,* to send the tracks themselves to the different speakers within the array. And while not particularly unique, IMAX had been implementing their own system utilizing a separate film tape and audio tape to feed a room of six tracks to six individual speakers in the early 1970s. However, Walter Murch's sound design for *Apocalypse Now* along with Ray Dolby's combination of previous technologies changed the industry forever.

FRANCIS FORD COPPOLA PRESENTS
Apocalypse Now
MARLON BRANDO ROBERT DUVALL MARTIN SHEEN in APOCALYPSE NOW
FREDERIC FORREST ALBERT HALL SAM BOTTOMS LARRY FISHBURNE and DENNIS HOPPER
Produced and Directed by FRANCIS COPPOLA
Written by JOHN MILIUS and FRANCIS COPPOLA Narration by MICHAEL HERR
Co-Produced by FRED ROOS, GRAY FREDERICKSON and TOM STERNBERG
Director of Photography VITTORIO STORARO Production Designer DEAN TAVOULARIS Editor RICHARD MARKS
Sound Design by WALTER MURCH Music by CARMINE COPPOLA and FRANCIS COPPOLA AN OMNI ZOETROPE PRODUCTION
R RESTRICTED UNDER 17 REQUIRES ACCOMPANYING PARENT OR ADULT GUARDIAN
TECHNICOLOR®
Copyright © 1979 Omni Zoetrope. All rights reserved.
Original Soundtrack Album available on Elektra Records and Tapes
DOLBY STEREO
70mm Six-Track IN SELECTED THEATRES
United Artists
An MGM/UA Communications company

Figure 2.5 Apocalypse Now [39]

[39] "Apocalypse Now (1979)."

The release of *Apocalypse Now* was met with heavy praise from critics, editors, and sound mixers, claiming that it had "changed forever the way we think about film sound." The film was the culminating achievement of Ray Dolby and his team's implementation of previously discovered technologies. The technical innovations such as the "whump whump of helicopters at the opening," were the sound design of Walter Murch and much of the reverence for the movie was attributed to his contributions. Murch, having been a listener of experimental multi-channel music fell in love with sound coming from all directions. Murch loved the idea of sounds flying through space and likened the experience to being like the era's 'psychedelic haze.' This listening experience is what inspired him to try and create a realistic soundscape of war-torn Vietnam.

So, then what was so special about Dolby? ***Walter Murch claimed that he could only achieve success in multi-channel recording through his partnership with Dolby Stereo.*** Walter Murch, in congruency with Dolby, came up with the term 5.1 format, before that it had no name it was just "what we were doing for *Apocalypse Now."* Murch claimed that the history of sound in film could be divided in half, film sound from Before Dolby (BD) and film sound from After Dolby (AD). According to Murch's testimony, he attributed this new force in transforming film sound directly to Dolby and not to his sound design. This began what is known as the Dolby Myth, or the belief that it takes a genius of unprecedented artistry to mix in surround sound.

2.2.2 *Tron* and Comparisons to similar sound design

Tron was undoubtedly a groundbreaking film at its release in 1982. Wendy Carlos's score resonates in the brain of any reader who has witnessed the original spectacle that was *Tron.* Like other Dolby releases, the surround channels are used throughout the film, not just for anecdotal interludes. During *Tron*, when the main characters visited the arcade, or when a disc slices through the antagonist's head, the soundtrack at the arcade is played through all the channels just as the impact against the head. The use of both front and rear loudspeakers was common in the Dolby 5.1 releases, but this technique was designed by *Fantasia,* with its array of six loudspeakers, three in front and three in back. Additionally, the whistling in *Westside Story (1961)* and many numbers in *Sweet Charity (1969).*

In another comparison, to create an atmosphere of closeness, when the main characters in *Tron* talk in their small apartment, the conversation is kept only in the front two speakers along with the city noises. However, when the characters converse in the digital world, the sounds emanate from all the channels to recreate the magnitude that is the inside of the arcade game. In *Journey to the Center of the Earth (1959)* the quadrophonic mix differentiates between aboveground and belowground. When the characters are aboveground, the atmospheric noise of a pub is only in the stereo front channels, versus when they are underground the rumbles come from all four channels.

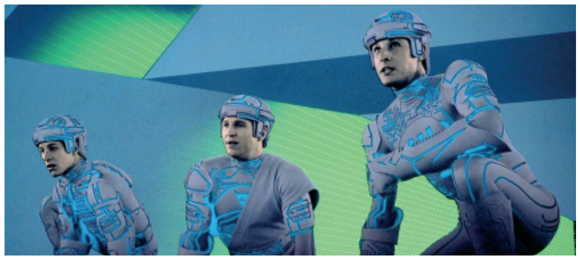


Figure 2.6 The World of Tron

Rick Altman of the University of Iowa says that with the emergence of Dolby "the surrounds began a new career (especially in fantasy or horror films) as purveyors of spectacular effects," where "every menace, every attack, every emotional scene seemed to begin or end behind the spectators." Another movie *The Sand Pebbles (1966),* used multi-channel mixing to create motion, having voices and footsteps reverberate from the front of the stage to the back in long stone hallways. *The Battle of the Bulge (1965)* used panning from front to back to simulate the feeling of planes flying overhead. So, while Dolby created and defined a new powerful niche within the industry, it wasn't responsible for the technology that it was peddling. There were plenty of films and pioneers whose visions for cinema and sound paved the way for Dolby 5.1.

2.2.3 Call me a sound designer, not a producer…

In 1948 the supreme court had ruled against Paramount Pictures in *United States v. Paramount Pictures,* in summary: motion picture companies could no longer own the theaters that they released movies inside of due to their block-booking and price-fixing practices. This led to the rise of independent theater chains, as well as more frequent releases of independent films. Independent films lacked the budget of the larger movie studios and used smaller labor forces which meant that for the most part they didn't have their own sound departments. Additionally, many of the large-scale production studios dismantled their own long standing permanent sound departments. This led to the era of freelance editors and mixers, many of whom formed their own independent postproduction studios.

Due to the nature of film making, and its visual nature, producers often forgot to reserve funds for postproduction sound. Sound workers were now competing for fewer jobs, while receiving less money. Walter Murch had a solution for this, elevate your status, and instead of calling himself a sound editor or a re-recording mixer, he was now a "Sound Designer." This title didn't only have the purpose of being more glamorous and sounding more artistic, it also allowed practitioners to skirt sound union rules without violating them. The term didn't last long, as all good things get abused, many novice engineers and manufacturers tried to purport themselves as Sound Designers when truly they weren't a shadow of Murch. However, due to Murch's prowess and affiliation with Dolby, a stigma stuck, that the editors who worked with Dolby were different.

Much of what the sound engineers drew from were popular culture, history, and advice from peers, but the myth of the solitary genius continued, with many claiming inspirations from Pierre Schaeffer and other avant-garde musical composers. While the engineers at Dolby had a hard time admitting their indebtedness to popular practices, they did manage something extraordinary… they managed to turn opinions on sound mixing from a boring technical skill into an artform. This led to more jobs, higher pay, and general reverence to film sound allowing for higher budgets and more general acknowledgement. So, while Ray Dolby wasn't exactly a visionary, his partnership with Walter Murch changed the world of spatialized sound for the better. [40]

Figure 2.7 Ray Dolby's Star of Fame [41]

2.2.4 Atmos

In 2012, Dolby changed the game again with the release of Pixar's *Brave* and the unveiling of the new Dolby Atmos system. The goal of Dolby Atmos is twofold; to create a high-fidelity sound system that also functions as an immersive listening experience. Atmos was so successful that it became the most widely adopted immersive sound solution in the world. So how did Dolby do it again?

Atmos, like the original variants of surround sound has increased its variability by an axis, adding sounds that can travel above and below. Atmos treats sounds as objects, instead of speakers and channels, allowing for dynamic mixing of these sound objects along the X, Y and Z axis of a plane. This allows for a full three-dimensional experience with plane sounds flying overhead, subways rumbling below and actors looking at sounds they are talking about off screen.

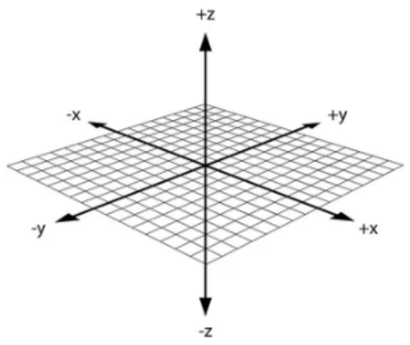


Figure 2.8 Representation of X Y Z Axis

Dolby Atmos can render up to sixty-four speaker feeds and each auditorium undergoes rigorous testing to make sure it's provided the proper technology. Atmos can handle up to one hundred and twenty-eight individual sound objects at a time. The Dolby

[40] Dienstfrey, "The Myth of the Speakers."
[41] "Ray Dolby."

rendering software has speaker protection built in and Dolby themselves produce the amplifiers themselves. Dolby even offers their own line of speakers and provides discounts to theaters to outfit themselves in entirely Dolby produced equipment, minus the projector. This is reminiscent of the old anti-trust laws against film companies, although on a much smaller scale.

What also makes Atmos different is its scalability. Atmos, can be scaled almost unlimitedly. Atmos can be played on hundreds of speakers or on a classic 5.1 system without the need for additional mixing. This is useful for older movie theaters, and home sound systems. Atmos is scalable from the largest most modern theaters to an older home theater system, or even a home that doesn't use one at all. With Atmos, a sound engineer can mix the movie with the sound objects that will translate into any speaker set up.

However, what if someone at home wants to upgrade and they really want to have height channels in their own home? Will they have to purchase speakers to put on the floor and the ceiling? Absolutely not. Upgrade your old TV and your older sound system to an applicable streaming device and speaker system. Atmos home theater speakers exist in something as basic as a soundbar. The soundbar can decode the height channels and uses a separate upwards facing speaker to project sound onto the ceiling and back down to the listeners ears to give the illusion of height. More advanced systems might have each of the 5.1 speakers projecting sound to the ceiling to give the feeling of height all around the room. In just ten years this technology has become so adopted that most films, many video games and even companies like Apple are implementing it into their products, such as headphones. [42]

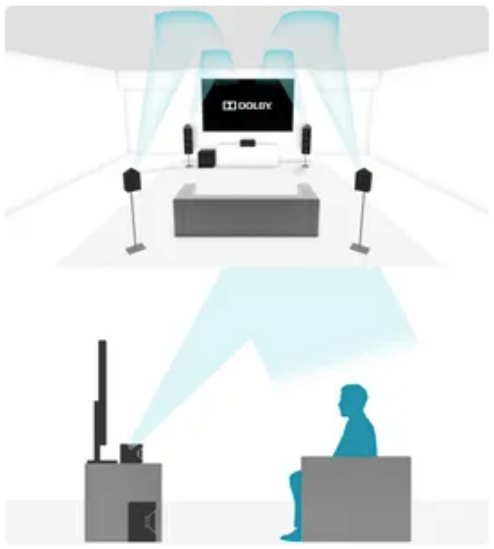


Figure 2.9 How Dolby Home Theater Works

Dolby themselves have built over one hundred and eighty mixing facilities for both high end and local creative talent to mix in, adding to the exclusivity and chokehold that Dolby has on their own market. To mix for Dolby, you have to work with Dolby, there is not an in between. However, this does allow for Dolby, and it's contracted mixing engineers to constantly critique their own mixes to better adapt the technology for visual media. Dolby

[42] Morrison, "Surround All Around."

continues to pump out amplifiers that allow for simpler set up using more advanced “brains” that can allow for the same sound with less speakers.

SMPTE (Society of Motion Picture and Television Engineers) published two standards for immersive audio in 2018. Both were derived from Dolby Atmos, there’s no competition. Dolby believes that its positive step toward simplified distribution that allows for cinemas to have improved audio experiences on their choice of *COMPLIANT* immersive sound systems. If I can just throw an opinion out here, the use of the word compliant by Jed Harmsen, one of Dolby’s vice presidents is genuinely scary. Who is setting the standards? Is it the motion picture industry or is it Dolby?

Dolby genuinely believes that their quality control is superior, but to be honest, it probably is. Due to Dolby’s widely adopted 5.1 system in the 1970s and its subsequent market manipulation Dolby Atmos brings consistent quality sound to theaters all over the world. They also provide mixing tools for the at home producer, all hail our great sound overlords Dolby! [43]

2.3 Not for the average listener

2.3.1 Ride the Waves

There have always been dissenters of common ideas in the world of science, in this case we will start by looking at Thomas Young. Young in the year 1802 had the *cajones* to try and re-argue that light was not a particle, but a wave. He was called “destitute of every species of merit,” for his subjective inductions. Wave theorists derived their ideas on an assumption, that light was a wave. Afterwards they would use this assumption to explain different optical phenomena such as rectilinear propagation, reflection, refraction, diffraction and so on. Then they would argue that the particle theory leaves gaps in the science that can’t be explained away and that the idea of light being a wave does not leave those gaps.

In modern times we are aware that light both functions as a wave and particle. [44]

Now despite being extremely bored in the nineteenth century, many scientists were grappling with new ideas and new technology, including the French Augustin Jean Fresnel. Fresnel started working on the concept of diffraction where he was up against highly successful theorists of the corpuscular, or particle theorists. In France, Thomas Young’s ideas were entirely ignored. However, Fresnel began amassing a small following including Francois Arago. Arago, being an insider to the French scientific community was able to give Fresnel an opportunity to gain support, unlike Young. But like Young, in 1815 when Fresnel published his first memoir on diffraction, he was obscure and unwelcomed.

Fresnel was immediately attracted to wave theory, leaning into the undulatory wave concepts. Funnily enough, by the time he had published his first paper Fresnel had already worked out the concept of transverse waves. With Arago, Fresnel had already begun to study

[43] “Dolby Atmos.”

[44] Achinstein, “Waves and Scientific Method.”

the effects of interreference on polarized light proving many of his undulatory wave ideas. Bringing back the concepts originally postured by Christaan Huygens, Fresnel based his research on the concept that wavefronts will be composed/described as a multitude of basic elementary spherical waves. [45]

In 1875, Gustav Kirchhoff arrived in Berlin and through his research with Hermann von Helmoltz discovered an algorithm simply dubbed the Kirchhoff-Helmoltz integral. [46] The solution of the wave equation describes the response of a given wavefield, measured upon a closed surface, at a given observation point within the volume enclosed by that surface. Kirchhoff-Helmoltz is used to model primary reflection in smooth layered models inside the bounds of smooth surfaces. The integral provides a tool to simulate quantitatively: wave propagation. [47]

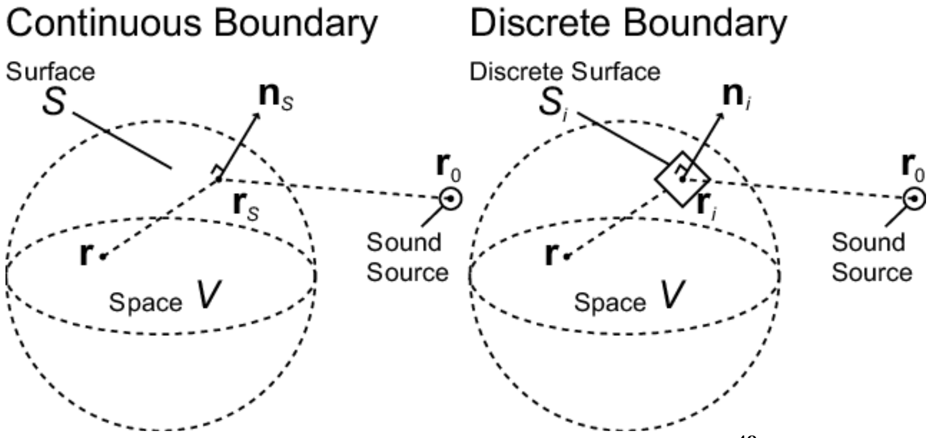


Figure 2.10 Coordinates for Kirchhoff Helmoltz [48]

Wave Field Synthesis came to prominence in the late 1980s, conceptualized by a professor A.J. Berkhout. Of course, the underlying base is the Kirchoff-Helmholtz integral that states that sound pressure and particle velocity in any point at the surface of a source free volume is known, then the sound pressure at any point within this volume is determined. This technique can be used to simulate natural sound like wave fronts by assembling larger sounds into their respective elementary waves. A computer synthesizes the sounds into a vast number of loudspeakers at the exact right moment following a similar equation to that of the Kirchhoff-Helmholtz. However, unlike the spherical nature of the Kirchhoff-Helmholtz, WFS (wave field synthesis), unlike Ambisonics (which we won't touch on) doesn't need to use spherical arrays to create fully immersive sound spaces, all it requires is one linear array along a wall space.

[45] Frankel, "Corpuscular Optics and the Wave Theory of Light."c
[46] "The Doctrine of Description."
[47] Tygel et al., "The Kirchhoff-Helmholtz Integral Pair."
[48] "Fig. 1."

$$\boldsymbol{P}(w,z) = \iint_{dA} \left(G(w,z|z') \frac{\partial}{\partial n} P(w,z') - P(w,z') \frac{\partial}{\partial n} G(w,z|z') \right) dz'$$

Figure 2.11 Wave Field Synthesis Principle

Wave field synthesis begins on the grounds that each point of a wave represents the starting point of an elementary wave. Using multitudes of small speaker sources, WFS can accurately portray the location of a sound from almost any point in space. It does this by breaking down sound objects and their spread, size, distance, and volume into accurate bite sized elementary waves. An incredible graphical understanding of this principle can be found at http://www.syntheticwave.de/Wavefieldsynthesis.htm, also found in my sources below. This animated depiction of WFS shows how sounds in front of listeners can accurately be represented by a single array of small speakers and also how using reflections, similar to Dolby Atmos, a single array of speakers can actually simulate full surround throughout the entire room.

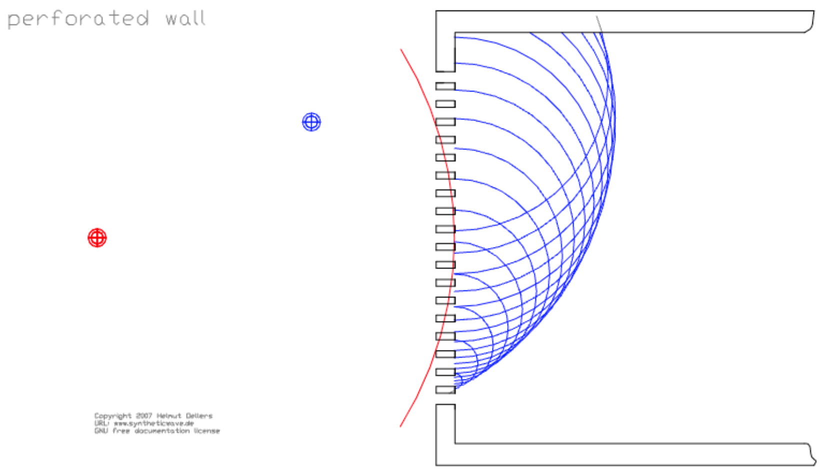

perforated wall
Copyright 2007 Helmut Oellers
URL: www.syntheticwave.de
GNU free documentation license

Figure 2.12 Snapshot of Helmut Oellers Graphical Representation

This specific animation takes Christian Huygens principle that each infinite point of a wave is the starting point of an elementary wave and breaks it down into a sample size. In this case, it is with holes in a perfectly soundproofed wall. Each hole represents a loudspeaker membrane. In this case, so long as the holes are small and close together, the larger wave, broken down into its smaller elementary waves, will maintain the same frequency, size, and amplitude. All that is needed in the case of a WFS array is the positional starting point of a recorded sound to replicate this effect.

Additionally, a large portion of what makes up sound is also reflections. Through WFS, these single loudspeaker arrangements can create more than one virtual sound source. What makes WFS so special is that it can take the position of the source in any position inside the room and utilize virtual reflections and real reflections to pinpoint its starting position through the use of either model-based approach or data-based approach.

The model-based approach starts from calculating the reflections of the recording room geometry. The distance of the virtual sound source positions regarding each loudspeaker position is where we determine the levels. Wall reflections are included in this calculation as well as the general radiation of the wave itself. However, this approach is difficult to use when working in multiple different spaces.

The data-based approach is the common practice. First, an impulse signal is broadcast in the room and recorded in such a way that the reflections are properly considered. This is done using a line array of microphones either in the same set up or similar set up as the wave field array that is to be installed. Through the physical delay recorded by the line of microphones, the computer can synthesize the room itself. These calculations get complicated when you include mirror source positions and reflections, temperature, and other environmental factors, even today these calculations are tough on our fastest computers. While WFS hasn't been widely adopted like Dolby Atmos, it is superior in some ways. However, what makes WFS special is what it has inspired in modern spatialized sound composers and inventors.[49]

2.3.2 Four-Dimensional Sound

In 2012, the same year Dolby Atmos was released, Paul Oomen demonstrated his opera, Nikola (named after Nikola Tesla) in the first theatrical presentation of 4DSOUND. Inspired by Wave Field Synthesis, 4DSOUND systems are described as a "spatial sound instruments with ability to project Sound Holograms; sonic vibrations articulated through controlled dimension, position, movement, resolution and interaction within space." What the concept can be broken down to is, sound can be pinpointed into individual spots within a space to create an environment where the listener can hear a sound as if it is right next to them. [50]

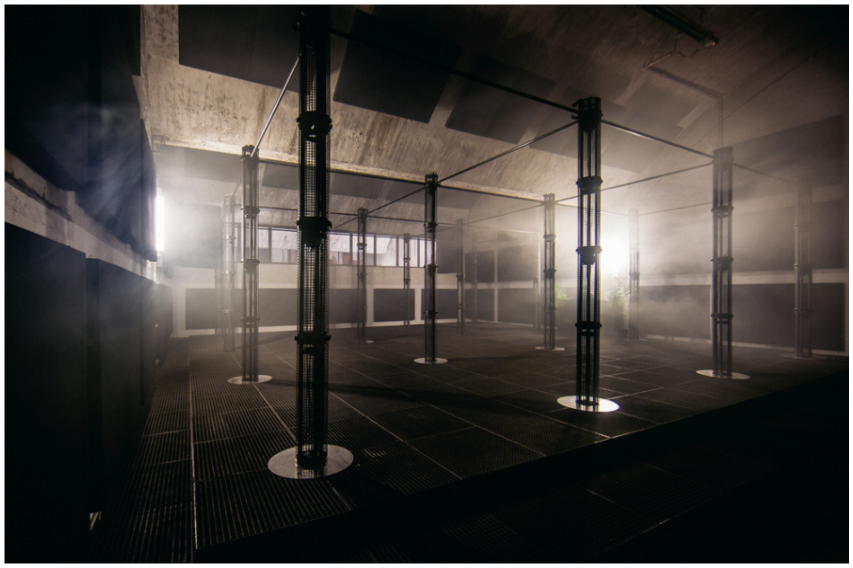


Figure 2.13 Picture of a 4DSOUND system (MONOM Studios)

The idea behind 4DSOUND began in 2008. Tired of club music and inspired by Nikola Tesla's writings about space and its relationship with our surroundings, Oomen got to

[49] Oellers, "Wave Field Synthesis."
[50] "4DSOUND."

work. He wanted to take the natural perception life and sound into the club, to immerse the listener in an environment rich with sound but also separated from it. In a specific performance done with techno artist Stimming in 2013, Oomen divided the room into sixteen separate sections. From there three separate loudspeakers were placed per section to project sound from all partitions of the space.

Performances like this are complicated, and the performer must adapt to the technology itself. The performance by Stimming was primarily performed on an iPad. Stimming found it frustrating to control sound objects and build spaces within the room. It was not only challenging to grasp visually in the software, but also aurally it required a lot of brain power. In the end he found that simple sounds sounded best because the complexity of the sound system itself was more than enough to stimulate any audiences' ears. Stimming even went so far as to play the intro in stereo before starting to spiral sounds all throughout the space. [51]

"Imagine listening to a symphony - not seated in the audience, but walking amongst the orchestra as it plays. In passing through, you hear violins on your left. Directly in front of you, the flutes are playing. You stand still and close your eyes as the melodies flow from the instruments. Suddenly, from the right-hand corner, the voice of a soprano gradually appears, drawing closer until she's right beside you. It seems to pass through you before disappearing. Then out of nowhere, drops of water start to crash on several spots on the theatre floor. It has started to rain. Slowly the orchestra pirouettes around you, while lifting itself up in the air." [52]

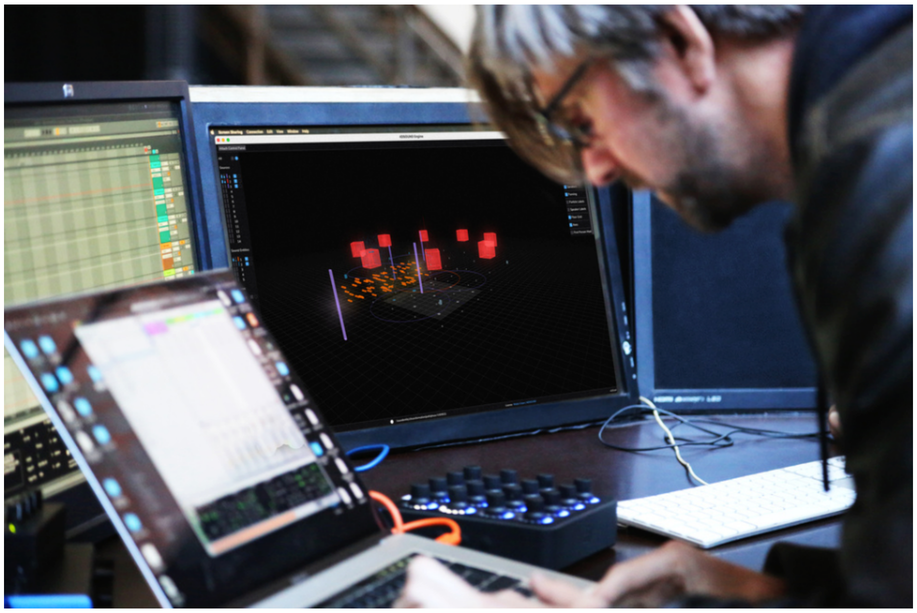


Figure 2.14 4DSOUND Software

What makes 4DSOUND the titan that it is in live performance spatialized sound all boils down the powerhouse that is its software. The user interface is adaptable to whatever

[51] "ONES TO WATCH."
[52] "P_Sound Holograms — SSI."

set up someone might be interested in using, and studio recordings can be saved as presets to be adapted from at home to auditorium set ups. The suite itself offers a multitude of resizable shapes, that can be chosen, and the user can also create their own spaces as well. Instead of sound being treated as the object, in this case the performer creates objects in space that are literally and physically sound.

By treating sounds as particles, sounds can be broken up and brought together in a way that feels like they are physically moving through space. This is the backbone behind the Sound Hologram. These particles are mapped as physical objects within the space and pan through each individual speaker creating the illusion that the sound is moving within the actual space. It's truly something that is hard to describe when it hasn't been heard. Sounds can appear and vanish from different spots with phase as well, which means that depending on where you are the sound experience an will be an entirely different performance than had you been standing in another location.[53]

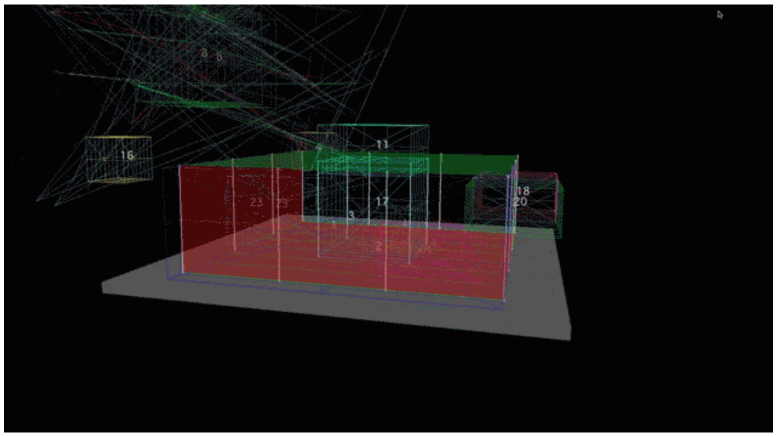


Figure 2.15 A look at Max Cooper's Composition in 4DSOUND [54]

One of the earliest adopters of 4DSOUND was Max Cooper. He debuted on the 4DSOUND system in 2013 with his album *Human.* His performances were done originally before the software had been made intuitive for the end user and his performances have been instrumental in streamlining it. Cooper is credited at Oomen's Spatialized Sound Institute as faculty as he continues to perform with and contribute to the system.

4DSOUND began as a Max MSP patch, that treated the channels of the speakers as individual instrument tracks or busses. These busses could be switched between by activating MIDI clips. Additionally, Cooper was provided with a visual representation of the space with boxes, if a sound was to get louder instead of going up in volume it would have to grow, in size. The larger the box the louder the sound. The sounds were represented as numbers along the grid that represented the speaker set up. Sound can also be moved up and down, which is scalable to the system. As Cooper puts it, it's halfway between a museum gallery experience and club experience. [55]

[53] "Software — 4DSOUND."
[54] "The 4D Soundsystem."
[55] "Max Cooper on the 4DSOUND System (2013) — SSI."

4DSOUND is about as fresh as it gets when it comes to spatialized sound in musical performance, with its applications not even fully realized. One of the most recent applications was taking the soundscapes of different cultural nations and bringing them into the studio space. This posed the question, if you closed your eyes were you in the studio or were you in Portugal? The piece titled SONOTOMIA used sounds from Portugal, Spain, and Budapest in an attempt to accurately recreate the outside world, in sound.[56]

Much like the excitement felt by many artists around the world after Scherchen's congress in Gravesano, Switzerland, modern artists are entranced by 4DSOUND. Electronic artists are itching to perform and create in new ways to immerse their listeners, elevating the grungy warehouse parties to eloquent gallery openings.

[56] "P_Sound Holograms — SSI."

Chapter 3
Audio Programming

3.1 History Of Digital Audio

3.1.1 Turning our backs on hardware

Only a little over forty-five years ago, in the year 1976, it was practical and inexpensive for any amateur engineer or musician to tinker with hardware. Building unique, specialized hardware for individual purposes was something that could be designed, built, and wired on cheap point to point boards in a garage with a soldering iron and a few hand tools. As this was the commonality between all musicians at the time, building, inventing, and producing hardware received much of the effort in the audio production industry. This was the way to achieve high performance tech.

If high performance was necessary for commercial success, or even just research purposes using hardware was the way to go. However, over time hardware began to have diminishing returns. With the rise of new software-based computing systems, engineers began to opt for a new means to produce their end goals. Throughout the 1970s and 80s computers grew more powerful at an exponential rate. As far back as the 1980s processor speeds were doubling every eighteen months. This is due to denser circuitry, lighter fabrication materials and the use of computers to produce the parts themselves instead of human hands.

In the past, synthesis algorithms had been based entirely on hardware capabilities, requiring constants and fixed sequences. With the introduction of software musical tools, these mathematical systems could change into more computational equations. Instead of sound and modulation being the same thing, such as the example case of control tones in systems such as Fantasound and Cinerama, modulation could be treated as a separate beast than the audio generation itself. This allowed for more control and started to break the cage of hardware-based programming.

In 1989 Chris Chafe processed a string score and used synthesis to develop the sound scape that was the quartet. He developed control events for bow-string contact, bow reversal, finger placement and vibrato. These functions could, albeit very primitively mimic live sound. And while he wasn't the first to attempt such compositions, since then things have drastically changed. The world has been introduced to different types of information such as "control rate," "audio rate and "note rate" functions just to name a few.[57] To top it all off, forty years ago in 1983 MIDI was unveiled at the winter NAMM show and it hasn't changed since.

[57] Dannenberg, "A Perspective on Computer Music."

3.1.2 What is a computer program?

To those who haven't spent time looking to understand what drives the tools they use daily, a computer program might as well be a giant piece of graph paper with some holes filled black and some left white. To some degree they would right, however there is a fundamental issue with this thought process. Just as humans speak in sentences, computation utilizes section of binary in the form of strings, digital patterns. These are arrays of cells or locations that contain either ones or zeros. Analog patterns and digital patterns differ in that analog patterns follow continuous forms and digital patterns are broken into cellular bits.

What makes computer programs so useful is that they can grow nearly infinitely in complexity. The strings and bits can be expanded and compressed to where the formal articulation becomes blurred. Analog patterns can be reproduced this way with a level of accuracy. Peter Suber refers to this in his paper "Software"as the Digital Principle. A photo of a face could be broken into small dots, which could be represented digitally, as if they were the filled in squares on that piece of graph paper. The smaller the dots, the more numerous they become, the more accurate the representation of the picture of the face. Perhaps a sound? A sound can be represented as single points, the smaller and more numerous the points, the more accurate the representation of the acoustics.

Under the Digital Principle, software can either function digitally or analog and is not limited to either. The only limitation is the power of the device itself. A computer can take instruction from an analog pattern such as sound and translate it into its digital counterpart. This must be done through a process called Linear Pulse Code Modulation.

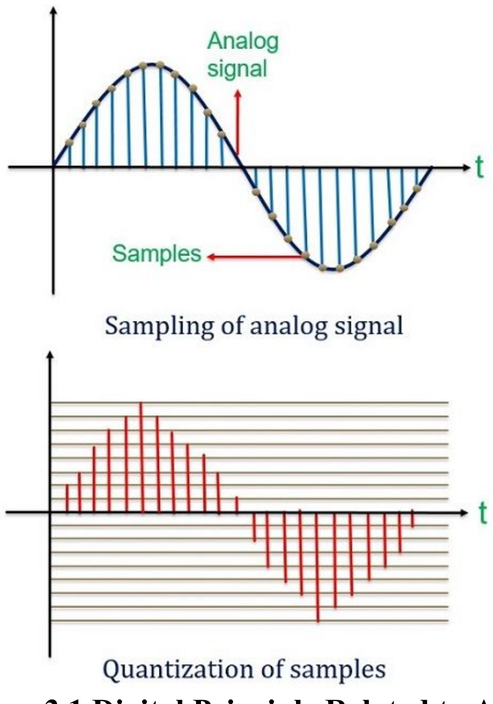


Figure 3.1 Digital Principle Related to Audio

However, software must be readable by a machine. This requires two things: First that the pattern must have proper material form and that the pattern much also be in a "machine language." A pattern has proper material form if the computer reads in that type of system. If a machine reads magnetic fields, as opposed to say punch cards, switches or grooves like a record, the pattern must be represented in magnetic norths and souths representing 1s and 0s. Programs are written in specific languages with uniform syntax, then translated by compilers which break it down the rest of the way to the most basic machine level. If a program meets the physical but not grammatical requirements it will crash. If it meets the grammatic requirements but the pattern isn't the proper material form the program will not perform.

Part of what makes software so appealing is that when it is properly prepared, it can run on any machine. Hardware is programmable, software is portable. Machines are built in a way that the patterns they recognize are uniform, so software that works on one machine can be lifted and used on another. Also, individual programs can be easily altered just through text, if the user knows what they are doing, unlike the physical nature of hardware that is hard wired. Using firmware, computers can all understand the same patterns and run the same programs.[58]

3.1.1 MIDI is mid?

MIDI is a communications scheme for digital musical devices, a type of firmware that all musical devices can recognize. MIDI has mixed reviews, but one thing is for certain it works. It was, if not the most, one of the most important advancements in fusing the physical and the digital when it comes to digital programming. MIDI has created a way to simplify the process of signal path between devices at the sacrifice of frequency and resolution, parameters, and modification. However, MIDI still managed to become the most widely adopted industry standard due to its ability to serve a variety of tasks.

MIDI is serial data transmission, known more simply as transmission protocol for computer terminals. The concept is that two devices are connected, and code is transferred between a channel. The channel must be robust, heavy duty, able to transfer from distance, resistant to interference as well as not interfering with other signals as well. While MIDI sends 5 separate strings of data it only uses three as two are opto-isolators.

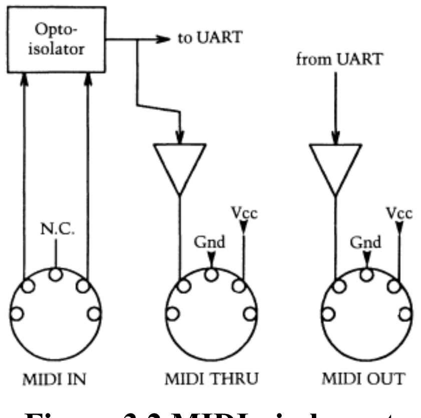


Figure 3.2 MIDI pin layout

[58] Suber, "What Is Software?"

MIDI sends data in ten-bit code bites consisting of a start bit, eight data bits and a stop. One argument against MIDI is that two of the bits in a MIDI packet are sacrificed for just start and stop messages, reducing the resolution drastically. However, MIDI does maintain the bandwidth to allow for human performance. The three packages sent via midi are a single status message and two data messages. A status message could be something like NOTE ON, the first data message could the NOTE VALUE and the last data message could be the NOTE PRESSURE. This is a wonderful and simple way to send data and receive data using a uniform connection and uniform language.[59]

3.2 Language

3.2.1 A+ for C++

C++ was developed from an earlier version of C called "C with Classes" and began its development in 1979 by Bjarne Stroustrup. However, development on the language had begun years prior. While working on his Ph.D. at Cambrige, Stroustrup wanted to study alternatives to the organization of system software in a distributed system. The focus ended up being the composition of a software of well-delimited modules. Stroustrup originally ran his code in Simula at Cambridge and found that his code was more readable than any language he had ever worked with. However, Simula didn't scale properly and almost caused C++ to crash and burn.

In Simula, link times for separately compiled classes were beyond sluggish. It took longer to compile 1/30th of the program and link it to a precompiled version than it took to compile the code as a whole. Additionally, the run-time performance was so poor that there was no way to obtain any real time data. While Simula has improved since, the improvement relative to other systems programming languages hasn't particularly caught up.

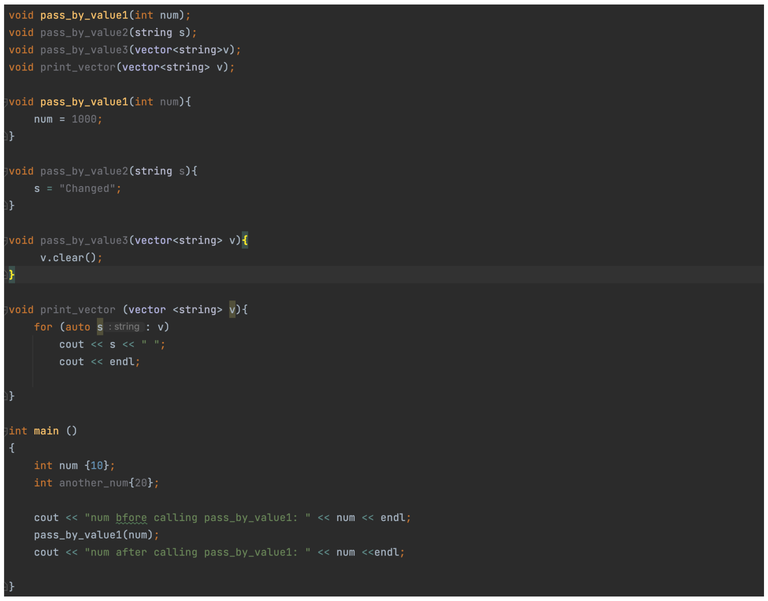
Stroustrup rewrote the simulator in BCPL and ran it on an experimental CAP computer. BCPL was in his own words, horrible. It made the original version of C look like a very high-level language. The higher the level of a language, the further away it from compiling directly into machine code, which uses valuable processing power. BCPL provided no type checking or run-time support. However, it ran suitably fast and provided useful results for Stroustrup. After these experiences Stroutrup came up with his own notions of what constituted a suitable tool for projects such as writing a simulator, operating system, or similar programming tasks. The three criteria described in Stroustrup's words:

1. "A good tool would have Simula's support for program organization--that is, classes, some form of class hierarchies, some form of support for concurrency, and strong (that is, static) checking of a type system based on classes. This I saw as support for the process of inventing programs, as support for design rather than just support for implementation."
2. "A good tool would produce programs that ran as fast as BCPL programs and shared BCPL's ability to easily combine separately compiled units into a program. A simple linkage convention is essential for combining units written in languages such as C,

[59] Loy, "Musicians Make a Standard."

ALGOL 68, FORTRAN, BCPL, assembler, and so on, into a single program and, thus, not to get caught by inherent limitations in a single language."

3. "A good tool should also allow for highly portable implementations. My experience was that the "good" implementation I needed would typically not be available until "next year" and only on a machine I couldn't afford. This implied that a tool must have multiple sources of implementations (no monopoly would be sufficiently responsive to users of "unusual" machines and to poor graduate students), that there should be no complicated run-time support system to port, and that there should be only very limited integration between the tool and its host operating system."



```cpp
void pass_by_value1(int num);
void pass_by_value2(string s);
void pass_by_value3(vector<string>v);
void print_vector(vector<string> v);

void pass_by_value1(int num){
    num = 1000;
}

void pass_by_value2(string s){
    s = "Changed";
}

void pass_by_value3(vector<string> v){
     v.clear();
}

void print_vector (vector <string> v){
    for (auto s : v)
        cout << s << " ";
        cout << endl;

}

int main ()
{
    int num {10};
    int another_num{20};

    cout << "num bfore calling pass_by_value1: " << num << endl;
    pass_by_value1(num);
    cout << "num after calling pass_by_value1: " << num <<endl;

}
```

Figure 3.3 Some Simple Code in C++

Stroustrup was interested in modularization and communication. The C++ model of protection was even based on the idea of granting and transferring access rights, an integral part of capitalizing on software. While Stroustrup hadn't completed his work by the time he left Cambridge, these ideas were the foundation of C++.

In April of 1979, Stroustrup began working on analyzing UNIX kernels to determine how they could be distributed over local network at the Computing Researching Center of Bell Laboratories in Murray Hill, New Jersey. He found two problems: how could he analyze resulting network traffic from the kernel transfer and how could he modularize that? By 1980 he had created Cpre which used Simula like classes, to run several experiments and one true project: the C++ task library.

At the time C with Classes was still thought of just an extension of classic C. At the time the language did not contain any methods for providing concurrency, or completing multiple tasks at once, but it did have the ability to define class member functions with special meanings by the preprocessor. So, the application was useful for organizing programs rather than specific application areas. This is what has made C++ a general-purpose language rather than a C variant. To this day even when the language is modified, abstraction is prioritized over specificity.

Stroustrup was not convinced that C with Classes and its better organization was necessarily better than C. C was still faster, with less run-time overhead. So Stroustrup aimed to match C with Classes run-time, code compactness and data compactness to the speed of C. Even through the run-time differential was less than three percent, that was too much for Stroustrup. Also, Stroustrup didn't want to sacrifice original classes found in C to achieve this run-time efficiency and wanted C with Classes to be safer with better debugging environments.

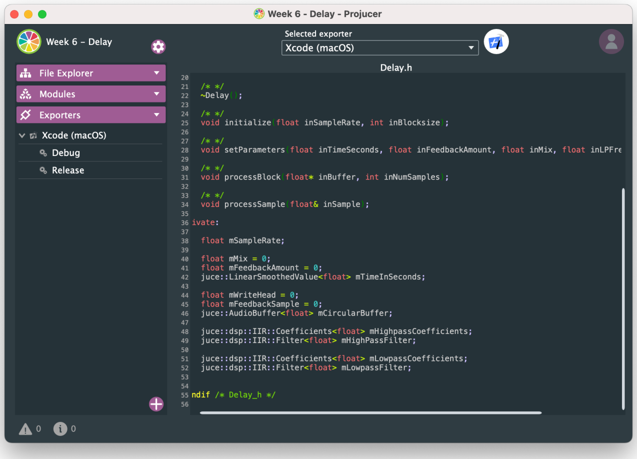
So instead of allowing C with Classes and later C++ to become a higher-level language, C++ maintains the path of a low-level language. It keeps the unsafe features of C; however, it systematically eliminates the need to use the features except when it is absolutely necessary. These unsafe operations are only performed at the request of the programmer. Stroustrup though it was wrong to force programmers to use one style, which informed his decision to leave the options available for the user.

What made C++ so monumental at its release in 1983 was its run-time efficiency. People refer to it as the C++ Principle. C++ was an affordable language in that it was common among developers. It didn't require high end equipment and wasn't too advanced for the computers at the time. Its specificity was part of its power, also the use of inlining saved valuable computer power by making sure the programmer was leading the compiler, and not the other way around.

Over time Stroustrup's creation has been customized and bastardized. The rise of libraries has helped to add to the language, allowing it to be used for a plethora of application support types. However, the challenge remains in teaching users of C++ good habits in the code syntax, since minor errors can cause huge computational problems after going through the compiler due to the lack of safety protocol. However, in the case of this thesis what makes C++ so unique and special is its speed. [60]

3.2.2 JUCE

JUCE is a framework and library for C++ for use in developing cross-platform audio applications. Since its release in 2004 JUCE has become the primary means to produce audio software. [61]



```

  /* */
  ~Delay();

  /* */
  void initialize(float inSampleRate, int inBlocksize);

  /* */
  void setParameters(float inTimeSeconds, float inFeedbackAmount, float inMix, float inLPFre

  /* */
  void processBlock(float* inBuffer, int inNumSamples);

  /* */
  void processSample(float& inSample);

ivate:

  float mSampleRate;

  float mMix = 0;
  float mFeedbackAmount = 0;
  juce::LinearSmoothedValue<float> mTimeInSeconds;

  float mWriteHead = 0;
  float mFeedbackSample = 0;
  juce::AudioBuffer<float> mCircularBuffer;

  juce::dsp::IIR::Coefficients<float> mHighpassCoefficients;
  juce::dsp::IIR::Filter<float> mHighPassFilter;

  juce::dsp::IIR::Coefficients<float> mLowpassCoefficients;
  juce::dsp::IIR::Filter<float> mLowpassFilter;


ndif /* Delay_h */
```

Figure 3.4 JUCE's Projucer

[60] Stroustrup, "A History of C++."
[61] "JUCE Announces Acquisition by PACE | JUCE."

JUCE is operated from its Projucer application. It also contains additional tools to test your applications connections and quality. The library itself was initially developed by one person Julian Storer. However, since the product is open source and operated in C++ the community of amateur and commercial users have added to it over the years. The applications built with JUCE are used by millions of people every day in the form of applications, plugins, and firmware.

The library has uses in the entire multi-media industry which means that pretty much any piece of music, film or TV made recently will have used something built in JUCE.[62] Music is played to a tempo. Audio is heard in continuum. Computers follow the Digital Principle, following language and patterns. C++ offers us the speed to create the illusion that sound is flowing fluidly like a river, and that changes happen instantaneously like a lightning strike. The customizable nature of C++ through frameworks such as JUCE are what sets it apart from other languages when it comes to audio programming.

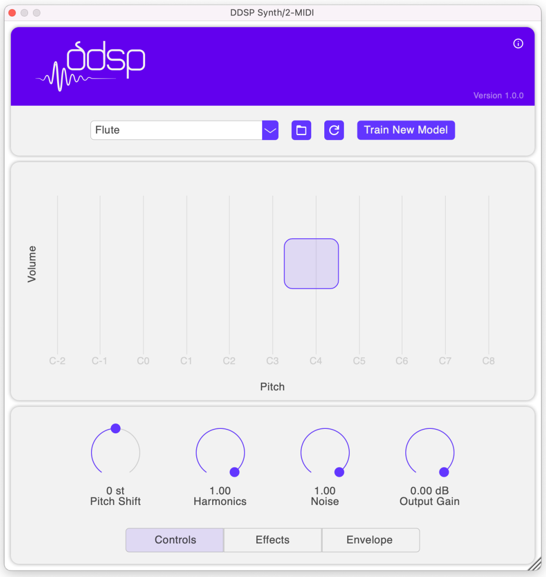


Figure 3.5 Example of a Plugin Built in JUCE[63]

[62] "Ecosystem | JUCE C++ Library."
[63] "DDSP-VST."

Chapter 4
More history but make it Personal.

As I sit here and write this paper, I contemplate the journey that brought me here and the ideas that have shaped my experience at Cal Arts. While this chapter is not an opinion piece or a layout of my subjective ideas, moving forward the words will stray from objective history and academia. In this pinpoint of a moment, I will write my story. My journey as a musician is like many but different from most. My relationship to technology is as vast as it is shallow. If I could go back in time and tell my middle school self where I am today, he wouldn't believe it.

There's something to be said about what you gain from school and what you gain from experience. Whatever that is, isn't applicable here. I'm going to say my own piece.

4.1 Personal Choices

What are we if not an amalgamation of our personal choices? The only thing that we truly have control of is how react to our surroundings. Let this chapter inform the next.

4.1.1 Start from the Top

As a performer, I was always in it for the fun. As a musician, I was never in it as an academic. My childhood piano teacher eventually grew so tired of my lack of practicing that she began to teach me song structure at the age of eight. Whether it was stubbornness, or ADHD I would rather noodle for hours upon hours than practice a piece for five minutes. Also, piano wasn't cool by any means. My friends played instruments like guitar and bass, their parents listened to Jane's Addiction or the Offspring, mine Andrea Bocelli or The Beatles. To this day I know more Beatles lyrics than any other band.

So, at the age of eleven, the year 2003, my grandmother bought me a nylon string guitar, not cool, but somewhere to start. Again, despite my parent's asking if I would like lessons I just noodled, blissfully unaware of even the concept of guitar tabs. I continued to write my own music. I performed in talent shows and I even put together a band for my bar-mitzvah where I performed, and composed parts for drums, bass, and guitar for a total of three individual songs as well as (I still cringe at this) Greenday's *When I come Around.*

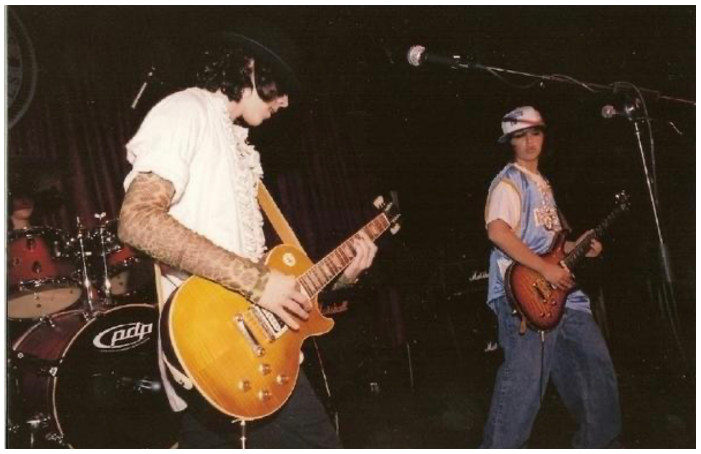


Figure 4.1 My Skin Crawls in Embarrassment

We continued to perform at local venues around the westside LA, so long as our parents allowed, and at school related events. However, as hormones began to set in my bandmates interests strayed from practicing together. These interests including going to shows, one of which being a particularly hot and crowded experience at Coachella. I couldn't believe my eyes. What I was seeing on stage was one, or maybe two, individuals performing music on their own. My uninformed fourteen-year-old brain couldn't comprehend it.

So, I went a deluge of research about how this was possible. My high school even offered a class, Modern Music Creation and Recording that seemed like it might lead me in the right direction, so I took it. The teacher was Tadashi Namba, an accomplished Japanese music producer with credits on *Dragon Ball Z* and *Hello Kitty.* While he wasn't a particular hands-on teacher, he provided us with some software and room to create digitally on our workstations. Two hours, twice I week I had access to Reason Studios (built originally in C++ I might add).

While Namba may have been relatively hands off, my personal journey with electronic music production began. I quickly learned that these specific DJs did *not* produce music on stage and just genuinely pressed play on tracks they had already produced. Instead of pressing further, I shelved the idea of creating music by myself on the fly and tried to wrap my head around new concepts such as MIDI and quantization. Some of the music I made during this period, as a fourteen-year-old, I think, was my most creative.

4.1.1 Firebelly

Around Fourteen, I was still performing with my band and while I didn't necessarily practice scales on my guitar all the time, I was playing it often enough. My dad, a ham radio connoisseur, decided that the aggressive loudness emanating from my room was too much and devised a solution. With a background in building tube radio amplifiers, he purchased a headphone guitar amplifier kit that we put together casually on a Saturday. It was my first introduction to circuit boards, soldering iron, capacitors, and resistors, I wish I had known how fascinating that was at the time. I just felt it was a chore, a task I had to waste my day on with my dad. (I would include a picture of this arts and craft project but it's resting next to my father in his oak box at the cemetery.)

So, I continued playing my guitar at home, banging on my drums when allowed and twice a week I made bizarre concoctions that someone might have been able to call music at school. I take no credit for the idea but having found himself a new joy in life my father decided to build a Fender tube amplifier clone, specifically a Princeton Reverb. He purchased a kit from the now long-gone Allen Amplifiers website, built it, played on it for a couple months and then decided he wanted a two-channel amplifier. So, he sold his first amp on eBay for double what it cost. What a novel concept? A hobby could make him some money.

I always hated the name *Firebelly,* but having seen these grubby little toads at the pet store I guess I equated the name to the firebelly toad. I didn't quite grasp what the name would come to mean. After buying, building, and selling a few kits with my help here and there my father had decided to start an amplifier company. We'd build them together. He grew up building ham radio amplifiers, so he had no qualms putting his fourteen-year-old son near 700 volts of pure DC. I just followed the schematics and the simple point to point layouts, but overtime we started to adjust the kits. We figured out the best ways to shield for hiss, the best ways to improve the tone and every amp we built was better than the last.

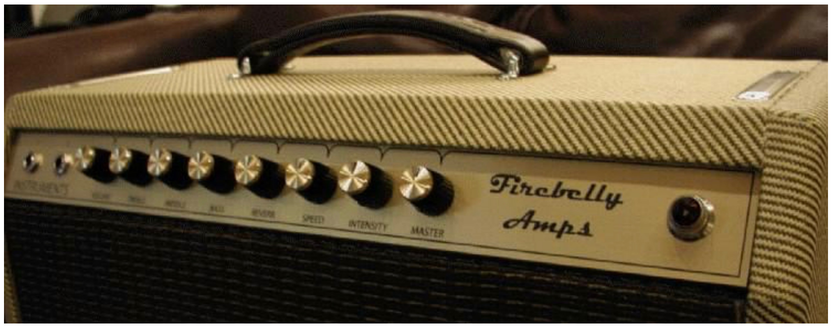


Figure 4.2 A Firebelly PR-35 (Princeton Reverb Based)

With a handful of articles in gear magazines, glowing reviews online and a partnership with Jensen speakers we decided we would aim for the big times. We signed up for the summer NAMM show in 2009 and built out a whole line of amplifiers to ship ahead for show. Our booth was sat in between Orange Amps and a talented unknown luthier, and we had more traffic than either. The name *Firebelly* was hot. By the end of the humid weekend in Nashville, we had more offers for distribution than we could count.

<u>We didn't take a single one of them.</u>

Prices for guitar amps, good hand wired tube amps, at the time were already reaching towards three or four thousand dollars. Funnily enough, subsequently prices have gone down since then due to the decline in Rock N' Roll's popularity. Our amps started at one thousand dollars, and we made half of that in profit, but the distributors also wanted to make a 50% margin as well. Taking a one-thousand-dollar product, making it two thousand and letting the distributors take all that cash it just didn't sit right. The next year when we exhibited at the LA AMP show, it was much the same, even with local stores.

So, we decided to maintain our business model. We wouldn't open a shop; we'd stay in our garage. We wouldn't build base models and leave custom jobs for the high-end clients; every amp was a custom build. Funnily enough, orders never dwindled. When I went to college, we raised the prices substantially because my father no longer had time to build amps, so that there wouldn't be too many orders to fill. We still got them. We increased the wait times for the amps to be built, the orders didn't stop. My summer vacations were spent building amps to fulfill orders for our customers, and I hated it. However, whenever he could my dad would come help me.

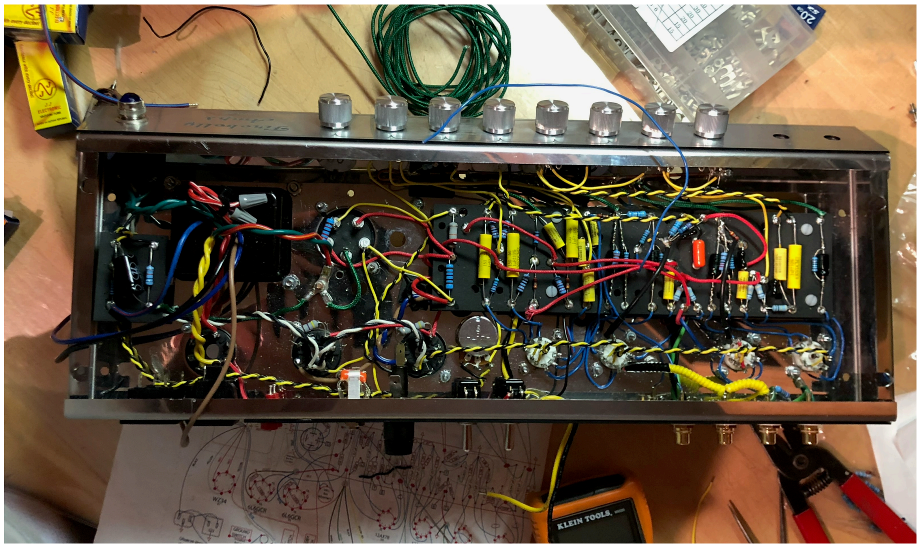


Figure 4.3 The Inside of the last PR-35 I built in 2020.

When I came back from college. I continued to build amps, I'd be working on five or six a time. The holidays got even crazier. But it wasn't until I had worked a job I hated for 2 years, that used up all my time, quit and came back to building amps that I realized how good I had it. Sure, the money wasn't quite enough to live on, but it got me by as I applied to grad school. But *Firebelly* was never about the money, it was about the joy of playing electric guitar on an incredible amp. It was about the meditative nature of filling the board, soldering connections, and crafting a piece of artistry.

When I started Cal Arts in 2019, we changed the wait time from eight to twelve weeks to six months to a year and raised our prices to try and reduce the amps I would have to build. We still received orders. We received orders up until the day I deactivated the website.

When my dad passed in April of 2020, I barely finished up my year at Cal Arts. It was hell. It took my mother and I over a year to pick up the pieces that he left behind. We *still* received orders. Usually, customers would send an email asking about pricing for a custom amp, or they would call and ask directly. However, here and there someone would just jump for one on the site.. No matter what I did, posting on Facebook, replying to messages, nothing stopped the orders. I had to deactivate the site, at least for now.

The time I spent, late evenings just soldering and testing and playing guitars with my dad, I'll hold onto those together. The skills, the knowledge and the understanding of the artistry that goes into building electronics, I'll never forget. Nothing will compare to the days standing over a toolbox, prodding the inside of a hot amp with a chopstick trying to figure out where the tiniest amount of hiss was coming from. I would do anything to be able to go back to figuring out how to build an amplifier with Marshall and Fender voicing on two separate channels, or how to properly tone cap an amp for a keyboardist. I learned more than

I realized, and it came from a place of love. I seriously had no f-Ing idea how much I was learning, how much we were learning together.

I've had friends work for me in the past and some of the techs I met in the process are still close with me. When my dad passed, I've had countless offers on places to move shop, or people wanting to help me get the company back up and running. I get messages on a weekly basis, asking if we are temporarily closed or permanently closed? Will Firebelly reopen? I'd like that. I also think it would mean a lot to the man who started it all, Steven Cohen. I still have two chassis ready to be prepped in my parent's garage, so who knows! I wonder if our partnership with Jensen is still valid…

Figure 4.4 Our Very Professional Business Photo

4.1.1 Everybody and their mother is a DJ

I mean it when I say it. Everybody knows how to DJ these days. A USB is all you need to rock a club. Not when I was fifteen, back in 2007. I started on a couple old vinyl turntables I was lent and a crummy Numark mixer, but due to my lack of funds, I didn't have a very strong record collection. So, I took all of my birthday money, along with a few bucks from my parents and purchased a pair of Numark CDX. They functioned like vinyl but read CDs or from an internal hard drive (that skipped all the time because it was 2007 didn't have a solid state hard drive).

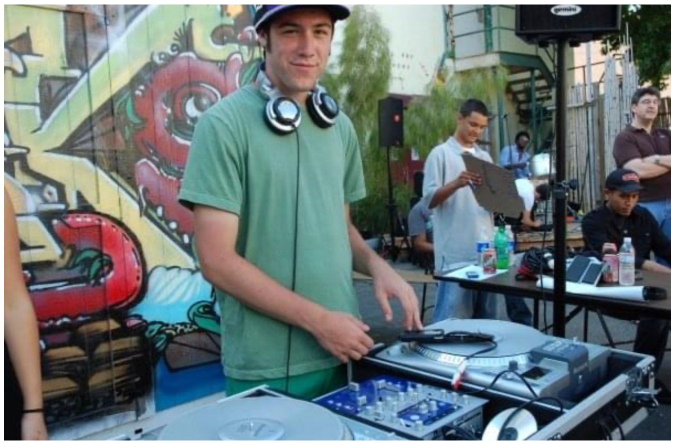


Figure 4.5 Rocking out a BMX event with Numark CDXs

So that was it. No more practicing guitar. I danced in my garage trying out mashups and transitions. I'd bring home songs I made in school on Reason and try them out in my sets. They rarely worked. But I tried. I got a PA system, started DJing parties. I started making money, improving my gear. I bought an iMac and got a free copy of Logic from my school. So now I could make music at home, press it to mp3 and put it right into my CDXs. I was making money; I was playing big events and I was even putting out my own music, but something still didn't feel right.

It wasn't really composition, was it? It wasn't the same as playing with a band, really relying on your skills and your bandmates to make sure it all turned out alright. If you messed up a transition, it might sound bad for a second but most of the audience wouldn't even notice. Was it the lack of risk involved? I'm not sure. One day I watched a video from Future Music on deadmau5's live set up. So, I sold everything and bought an Allen & Heath Xone:4D, the same device he used and bought Ableton live. The Xone:4D had four audio channels that could be used either with RCA inputs for CDJS or Turntables or as individual audio cards. It also had over 200 mappable parameters including knobs, faders, rotary encoders, spinning plates and lots and lots of buttons.

The way deadmau5 used it was: He ran two channels straight from Ableton to Audio channels one and two on the Xone:4D like turntables. Whole songs he had loaded in to play as if he was DJing. He used a lemur and monome to control beat repeats and other effect functions. Additionally, he had a third track in Ableton that he used for various loops and clips sent to channel three. Last he used a final channel in Ableton for a 'live performance' channel using loops and clips that could be triggered, muted, or effected with parameters routed to the fourth output on the Xone.[64] This was it; this was what I was looking for.

The Xone:4D was a staple in my performances from when I purchased it in early 2009 all the way until my first-year grad recital at Cal Arts in 2020. It's variable modularity and audio capabilities opened so many opportunities to build and refine DJ sets that felt alive. I could sit back and play my favorite songs, or I could compose bits on the go as if I was the whole band. It was something different than what everyone else was doing and I really enjoyed playing music again.

[64] Sundius, "Weekend Rewind."

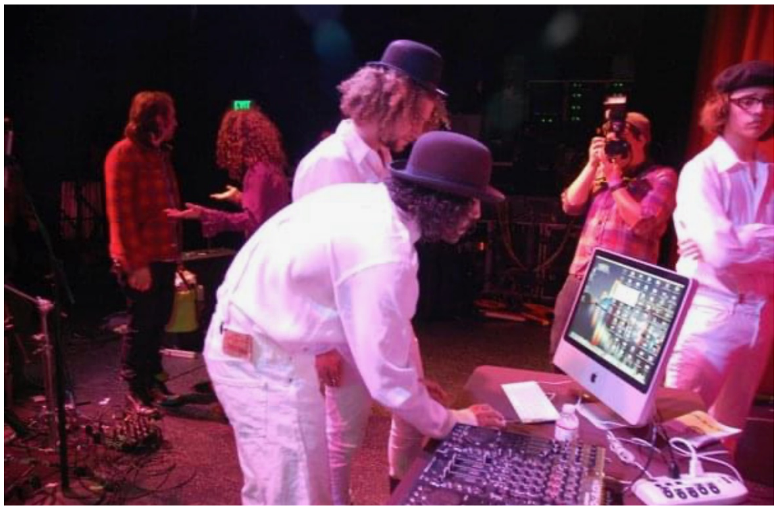


Figure 4.6 Xone:4D at Sound Check at the El Rey in 2010

The Xone:4D traveled with me all the way to New York where I spent a year studying economics at SUNY Purchase for my undergrad. Then it followed me to Boston where I had transferred to Boston University School of Business. I had purchased a MacBook pro by this time as the iMac was unwieldy, and that's all I needed. Countless events on a MacBook pro and Xone:4D, each different, always fun.

The Xone and I played art galleries, basements, nightclubs, and festivals. We remixed and created so many songs that will only live on in the ears of people who listened to them that night. This is what music was supposed to be. Could I even call it DJing anymore?

As time drew on, more and more people picked up Serrato, other software options or learned how to use Ableton. I didn't have the market cornered the way that I had before. But I still played shows. I made new friends, built up relationships and honed my production skills. Soon enough, as the shows got bigger the allowances for equipment got smaller.

Pioneer CDJs were the norm for clubs and warehouses, and often the answer was "no" when asked if I could bring additional equipment. A new project I had started with a partner, tau0n, was starting to gain traction so it was back to producing music and pressing play. It's funny how things go full circle.

4.1.1 Applying to Cal Arts

After quitting my job at the end of 2017, I was fulltime working as an audio engineer and building amps. My days were split between soldering and listening. Something all my peers who were having success in the field of music had that I didn't, was an education. I changed my mind back and forth a couple times, not sure if I had what it took to get a graduate degree.

Again, I had never studied music outside of some piano lessons in my early years.

I did have an idea. A plan as you might say. My vocabulary was limited and my sheet music skills abysmal, but I could write music. I could play music. So, I took piano lessons from a crusty old Russian lady to fulfill the basic requirements for the Performer-Composer track. I learned a couple jazz songs, that I absolutely failed miserably at playing when it came down to the actual audition. I wrote an essay, sent in my grades, and got to work.

What makes Parameter Based Performance so interesting is that the sky is the limit. The instrument is yourself and a piece of hardware, for me that was my laptop and Xone:4D. But this time, that wasn't enough. Times had changed. Artists were bringing more and more gear on the road, setting up more complicated rigs and normalizing what it meant to be an electronic artist rather than a DJ.

So, I switched out of Logic and started making music in Ableton. If I wanted to get better at live performance with Ableton, I knew I had to learn it inside and out. By the end of it all, I had come up with something I'd never done before.

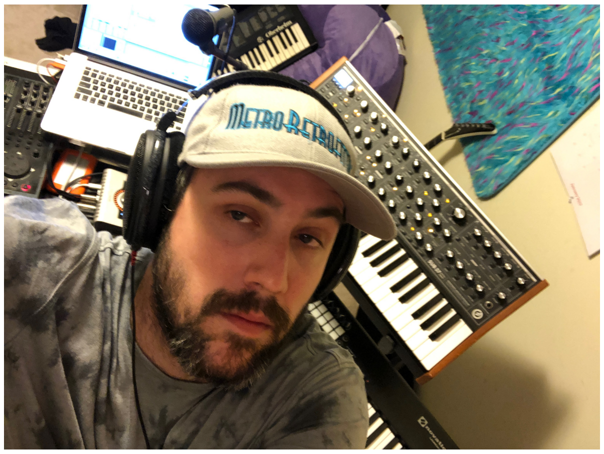


Figure 4.7 Exhausted Practicing with Audition Set Up

I added a Moog Sub 37, a Novation Midi Keyboard a Microphone, and my Apollo Twin. I'll dive more into the full routing and set up later, but this expanded live set gave me the ability to play the music I wrote, broken down by section in a way I had never done before. Music that could be melodic and moving rather than just classic techno.

On the actual day of the audition, something funny happened. All auditions were to be held in the Wild Beast, but I was told that the panel was actually moving around from classroom to classroom. That wasn't the case. However, once I had fully set up, far far away from the Wild Beast, I was told I would have to move. It was panic inducing, until Amy Knowles, having seen the set up I had put together said she would ask the teachers to come to the classroom.

As I already said, I did a terrible job in my piano skills. But when I provided each teacher with a pair of headphones to preserve the mix that I had spent all those months on, their eyes lit up a bit. That was it, the mistake that got me into Cal Arts Performer Composer program and the performance that I spent my first year expanding on.

4.2 Mid Residence Recital

4.2.1 Early Conceptions

My first year at Cal Arts started off weak, having failed to test out of fundamental music theory. Despite this, I was more than excited. I was learning new concepts, meeting new people. Every day was an overload of knowledge and I realized that I had so much room to grow. Many of the classes I had signed up for, weren't even exactly what I thought they would be. One of these very classes was the Introduction to Synthesis, which I thought would be an easy A. Instead, I got my feet wet in my first programming class.

I was never bad at math, but I never particularly enjoyed it either. I also was never really a language person, so in many ways learning a new language was a challenge on its own. All that aside, there was something gripping me about the concept of building a program from scratch. Sure, a lot of what I was building was just tutorial from my professor, but there was always something a little different between his and my own.

Never having the money, and truly the interest to dive into modular. This was my first foray into randomization, different synthesis types and the functions that make the instruments I love work. Let's put it this way, I never once visited TA hours in my undergrad for business, but I was there every single week for Reaktor.

During these music technology TA hours, I witnessed something different than I had seen in the other departments. A sense of community, working together to solve problems and build, in the simplest of terms, super cool toys. When I initially applied to Cal Arts, I couldn't dream that I might learn to code. It seemed so far out of reach, but when I saw the comradery and friendship, I started to think maybe I could do this. I could take the intro courses and walk out of Cal Arts a little more knowledgeable. I might try to learn how to make my own visuals. I might try to get a little bit more of an understanding in how interfaces work.

At the time, I was also working Bob Clendenen, helping students with their recitals and classes with their performances. Some of them I really resonated with, some of them I didn't but no matter what style of music they were they all had the same flavor. The same room, with the same lights, the same sounds, and the same college feel. I knew that I wanted my mid-residency recital to be different.

4.2.2 Setting Up

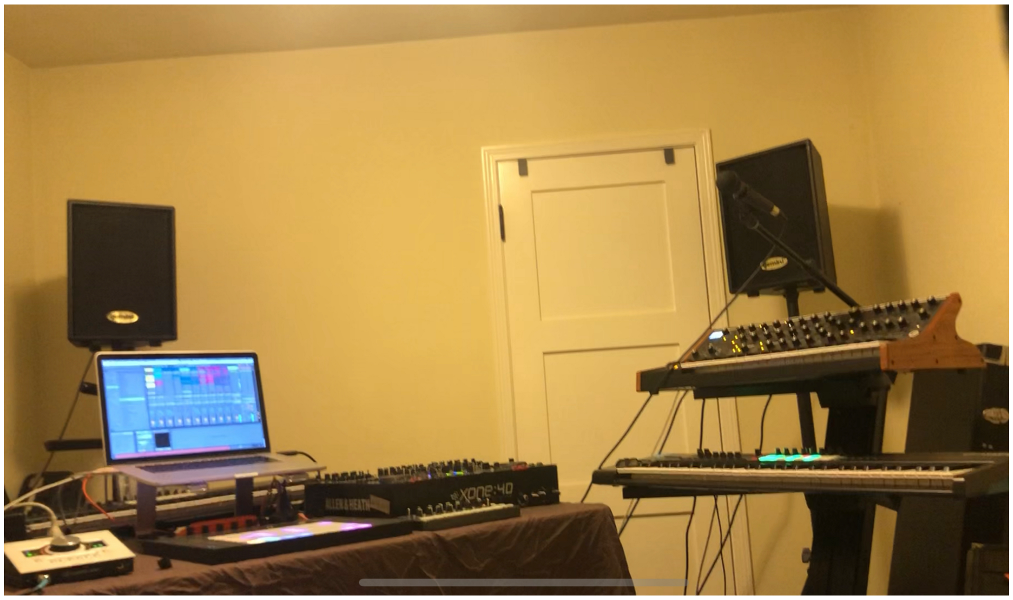


Figure 4.8 Image of initial setup

My Mid-Res recital was the culmination of years of experience, and two semesters of school. The idea behind it started long before even attending Cal Arts, but the application of it was a collaboration between myself and Kai Luen.

The structure I wanted was simple, an immersive musical experience curated by myself. I wanted the listeners to be transported from the ROD to places across the world, but kind of sci-fi style. During my first semester at Cal Arts, I had met Kai in Sara Reid's Electroacoustic Ensemble and Machine Orchestra. His input was not only instrumental to the mid-res recital, but also to my journey at Cal Arts. He helped lead me in the direction of which classes to take, pre-requisites and more. My respect for his skills in visual programming and our friendship led me to ask him to collaborate on the recital.

Giving Kai the wiggle room he needed, I put together a layout to showcase his video. There would be projectors in operation, back lighting two screens arranged in a V shape at the back of the stage. It would complement the V shape of the equipment set up, lead the eyes of the audience towards the performer and immerse the whole room in a visual performance in addition to the musical one. So now it was time to tackle the specifics.

First and foremost, I needed to be in control of the mix. Not just because I wanted to, but because of the nature of the performance and having only two XLR outs from my main source. This meant that before starting any kind of rehearsing with Kai, I would have to make sure that each song's volumes and mixes were sounding appropriate. This was challenging in that, one third of the performance were nicely mixed originals I would break down into performable sections, another third was covers such as *Enjoy the Silence* by Depeche Mode, and the final third were improv sections that almost never truly turned out the same. All I can say is, thank you to whoever invented low latency adaptive limiting.

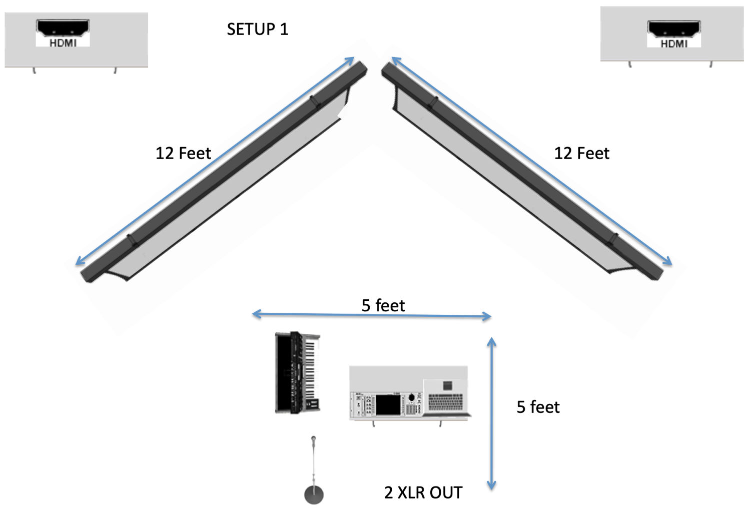


Figure 4.9 Layout of Stage Plot

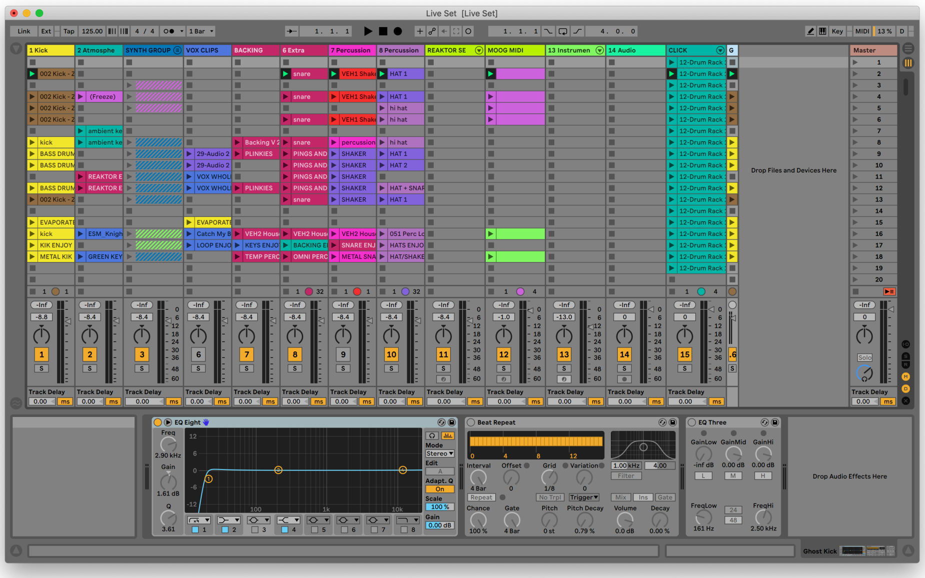


Figure 4.10 Overview of the Ableton Live Layout

The main driver behind this performance was Ableton Live. The first eight tracks were routed to the Stereo outs Channel ½ and sent to the corresponding audio card on the Xone:4D. Additionally each track was assigned its own column of parameters on the Xone 4D. They included in order, one rotary encoder, two knobs, one fader and three buttons. The faders were used specifically for volume modulation, while the knobs buttons and encoders were used to switch between various effects such as beat repeat, grain delay and so on.

Inside this specific eight track setup I included a synth group. The synth group was comprised of a MIDI and Audio track that were bussed together. The audio track housed backing synthesized sounds that would play during sections where I was too preoccupied to modulate parameters. The MIDI track contained an instrument rack that housed a few separate VST synth plug ins that midi clips would feed while I controlled parameters such as cutoff and resonance.

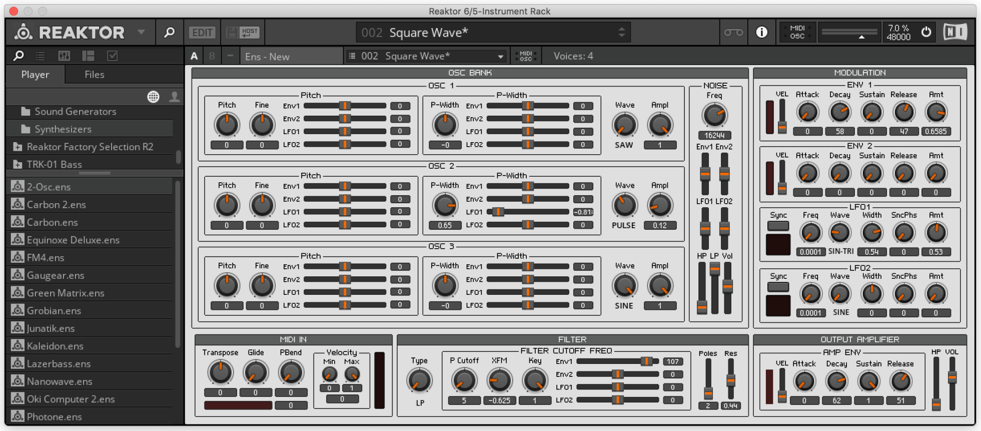


Figure 4.11 Reaktor Patch Built for Synth Group Instrument Rack

Instead of swiping through scenes, I used mutes and volume automations to control which instruments played at specific times. This meant that my own brain's processing power could be better suited to composing and playing, rather than remembering when to press a button to switch between sections. The implementation of this first section of eight tracks was simple as it didn't require any external routing apart from sending the audio directly to the Xone:4Ds first audio channel.

The ninth channel was built specifically for on-the-fly improv sections that were sectioned out as transitions between songs. This channel was driven solely by a Reaktor patch that used randomized tables for note velocity, ADSR for cutoff and pitch. It was very squelchy and could fill up a whole section very easily on its own. This allowed me to use vocal samples of my own or import them to do live remixing or just play aggressive techno riffs. The parameters for this channel were exclusively controlled by a Novation Launch Control. The eight buttons allowed me to scroll through patch options, while the knobs allowed me to adjust the synthesizers parameters subtly to craft the instrument for the song.

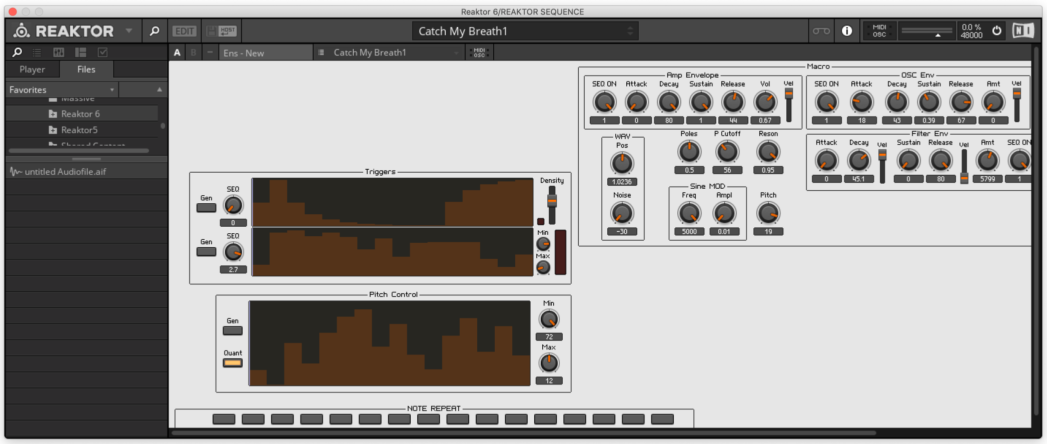


Figure 4.12 Randomizable Reaktor Patch Built for Channel Nine

While this Reaktor patch was routed directly out to Outputs five and six, corresponding to channel three on the Xone:4D, volume was easily modulated from directly within the plugin itself. Additionally, the live audio fader on the Xone:4D could be used to do quick chops and slow volume changes without requiring MIDI value changes. This channel also housed the routed output for Channel 13. 13 Instrument contained another instrument rack that included synths such as Reaktor, Massive X and other that were played directly by my Novation MIDI keyboard during more melodic pieces.

The Novation MIDI keyboard had knobs that allowed me to control swells of cutoff and delays on the soft synths as well as a pedal to control the sustain. These patches were all selected specifically as virtual synthesizers and not samplers since large sample libraries used up valuable processing power when performing in low latency. The low latency was necessary since I was manually playing out chord progressions for certain songs and while a couple milliseconds of delay was alright, anything more would be too much. Because of the 'humanness' of these two channels, they received their own output just in case they needed to be quickly muted or phased out from human error.

Outputs three and four, channel two on the Xone:4D was a bit of a different beast. This channel was specifically used for audio data. The audio provided was the direct outputs of my UAD Apollo Twin. Channel one was an SM57 microphone and channel two was output of a Moog Sub 37. This channel sent TRS from the Apollo straight to RCA and int the Xone:4D inputs never touching the inside of a DAW. This allowed the data to be processed in as close to real time as possible, reducing any chances of delay caused by processing. The UAD console, while limited, does have a lot of processing options which were instrumental in implementing this section of the set up.

The first issue was getting the vocals working correctly. Although I warmed up as much as possible and my pitch is better than decent, I knew that I had much more to focus on than just my vocal performance. On top of that, modern music expects a sort of digital feel for processed vocals. The first audio plugin I put on my vocal chain inside the UAD console was Antares Auto-Tune. While this did alleviate the issues I found myself having with my pitch, while multi-tasking it actually added an additional issue I had to plan around. When setting up for future songs with vocal performance I needed to dedicate time to switch from Ableton view to UAD console view and manually change the key with a mouse inside the plugin. Looking back had I used something like the open source UAD MIDI Control I could have avoided this pitfall.

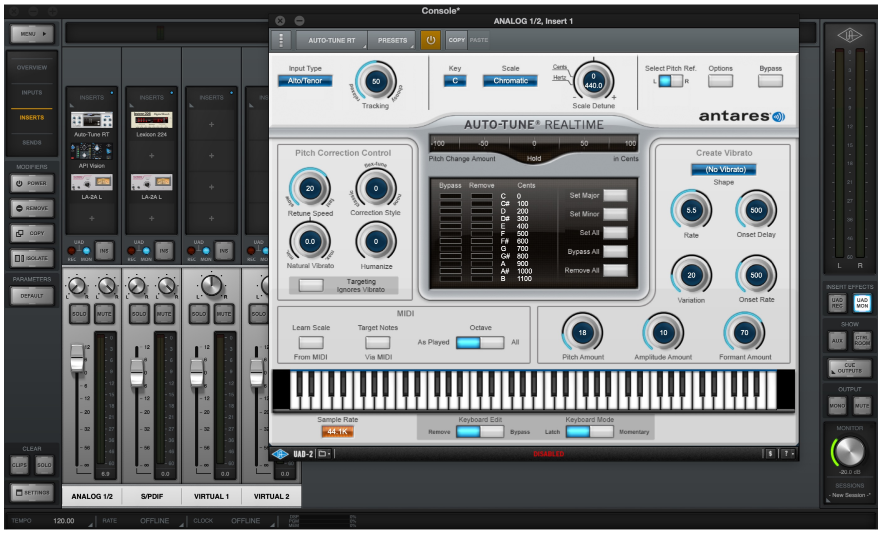


Figure 4.13 Recreation of UAD Console Layout for Performance

Secondly, most popular music’s vocal tone is soft and raspy. Big loud prominent vocals don’t have the same kind of emotional value as something quiet and emotional, at least in my opinion. Singing quietly in a large room presented its own challenge due to microphone feedback. One of the first ways I addressed this was by using zero monitor speakers on stage. Since the performance was loosely based on a DJ format, I used headphones to monitor the whole performance from inception to finale. However, the second solution was found through troubleshooting the set up for a different instrument.

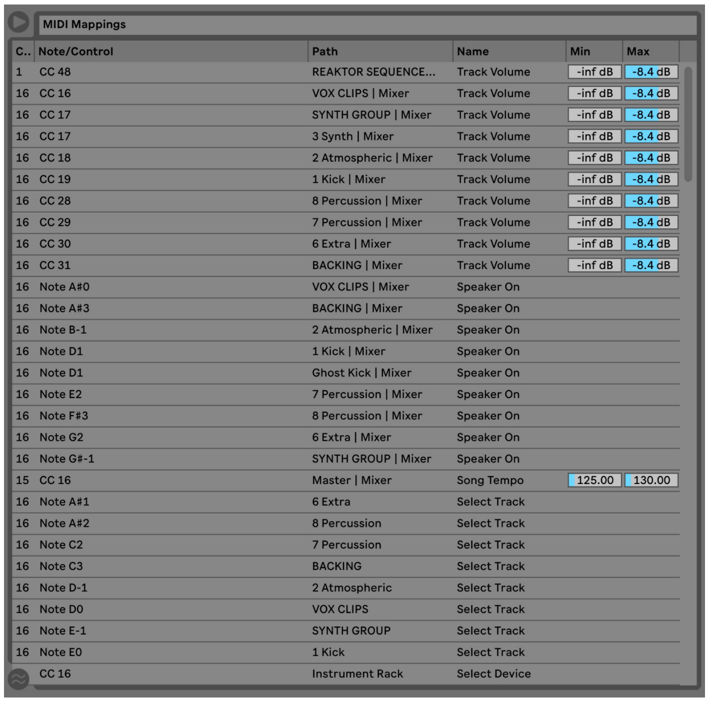
The second input on the Apollo Twin was for the Moog Sub 37 which I had used in my Cal Arts audition. In the initial set up with the Moog it was overpowering and while it looked cool and made some great sounds, it didn’t quite fit with the overall musical theme. Techno music relies heavily on side-chain compression where the kick is routed directly to the compressor of the synthesizers creating a pumping sound. This was easy to implement on the soft synths but required a bit of creative thinking for the Moog.

The Xone:4D, while technically only having four fade able output channels, it included a fifth, outputs 9 / 10, in the form of an optical channel for sends to video synthesis applications. The channel furthest to the right, the ghost kick drum, could be routed directly to this optical output. Then via a digital audio converter I turned the signal back into audio and fed it directly into a dbx 266 compressor as a side chain signal. This helped in two ways, the dbx was a compressor and a noise gate and thus I could kill one bird with two stones. For the Moog, without having to sacrifice valuable latency by running it through the DAW, I could feed it through sidechain compression in the dbx creating the pumping sound necessary for it to fit within the composition. And secondly, I was able to use the gate function on the additional channel to harshly gate my vocals so that a feedback loop would be avoided whenever the microphone was not in use. This was the backbone that made the audio channel sound like it was part of the composition overall, and not just two separate channels superimposed on top of an Ableton Live Set.

While most of the tracks maintained a form with strong rhythm sections, there were a few tracks that had long sections with no percussion. These were mainly atmospheric and melodic songs that had singing and pads with long swells. It was imperative that I had a functional click track. The Xone:4D isn't a traditional live mixing device for bands, like the Presonus, which would have built in headphone monitors. My headphone monitor was the headphone output from the front of DJ mixer, and it was identical to the stereo output feeding the front of house. So, to have a click track that was hidden from the front of house, available for me to listen to, I had to get a little bit creative.

Part of the reason the channels were grouped the way they were was because Outputs seven and eight, or channel four on the Xone:4D was dedicated solely for the click track. The crossfader could be pushed fully in the direction opposite channel four, but I could have mix monitoring on it, as if it was being treated like a muted turntable for beat matching purposes. In this way I was able to have a click track in my headphones that was not blasting out to the rest of the audience.

The final piece of the puzzle was how to navigate throughout the set. I also had more parameters than could be handled on the Launch Control and the Xone:4Ds surface mapping. In the past, I've used the Xone:4D's switch mode, where the face can be swapped for a fresh set of MIDI parameters by holding down a knob. All the parameters could technically be mapped a second time, double the number of useable features. However, the knob had to be held down for two whole seconds before the switch, which is an eternity when you're juggling a live set such as this.


MIDI Mappings

C..	Note/Control	Path	Name	Min	Max
1	CC 48	REAKTOR SEQUENCE...	Track Volume	-inf dB	-8.4 dB
16	CC 16	VOX CLIPS | Mixer	Track Volume	-inf dB	-8.4 dB
16	CC 17	SYNTH GROUP | Mixer	Track Volume	-inf dB	-8.4 dB
16	CC 17	3 Synth | Mixer	Track Volume	-inf dB	-8.4 dB
16	CC 18	2 Atmospheric | Mixer	Track Volume	-inf dB	-8.4 dB
16	CC 19	1 Kick | Mixer	Track Volume	-inf dB	-8.4 dB
16	CC 28	8 Percussion | Mixer	Track Volume	-inf dB	-8.4 dB
16	CC 29	7 Percussion | Mixer	Track Volume	-inf dB	-8.4 dB
16	CC 30	6 Extra | Mixer	Track Volume	-inf dB	-8.4 dB
16	CC 31	BACKING | Mixer	Track Volume	-inf dB	-8.4 dB
16	Note A#0	VOX CLIPS | Mixer	Speaker On		
16	Note A#3	BACKING | Mixer	Speaker On		
16	Note B-1	2 Atmospheric | Mixer	Speaker On		
16	Note D1	1 Kick | Mixer	Speaker On		
16	Note D1	Ghost Kick | Mixer	Speaker On		
16	Note E2	7 Percussion | Mixer	Speaker On		
16	Note F#3	8 Percussion | Mixer	Speaker On		
16	Note G2	6 Extra | Mixer	Speaker On		
16	Note G#-1	SYNTH GROUP | Mixer	Speaker On		
15	CC 16	Master | Mixer	Song Tempo	125.00	130.00
16	Note A#1	6 Extra	Select Track		
16	Note A#2	8 Percussion	Select Track		
16	Note C2	7 Percussion	Select Track		
16	Note C3	BACKING	Select Track		
16	Note D-1	2 Atmospheric	Select Track		
16	Note D0	VOX CLIPS	Select Track		
16	Note E-1	SYNTH GROUP	Select Track		
16	Note E0	1 Kick	Select Track		
	CC 16	Instrument Rack	Select Device		

Figure 4.14 An example of some of the MIDI mapped Parameters

While I did have minor mouse navigations in the form of swapping to the UAD Console to change auto-tune keys, it was important to me that I wouldn't be hunched over my laptop. The set up should feel like one big instrument, not a DAW. My initial inclination was to use the Ableton Push 2. However, this device has native mappings and menus for

Ableton and serves a very particular studio function. While it is great for launching clips, it is natively assigned to parameters for all the buttons. The solution was to use a regular Launchpad in a custom mode. I could create two scenes, that could be switched between instantaneously, one for launching clips and one for handling on/off values. This was the backbone that moved the set along as well as allowed me to select instruments and control the excess parameters. All in all, I was happy with how the set-up had turned out.

4.2.3 Early Conceptions

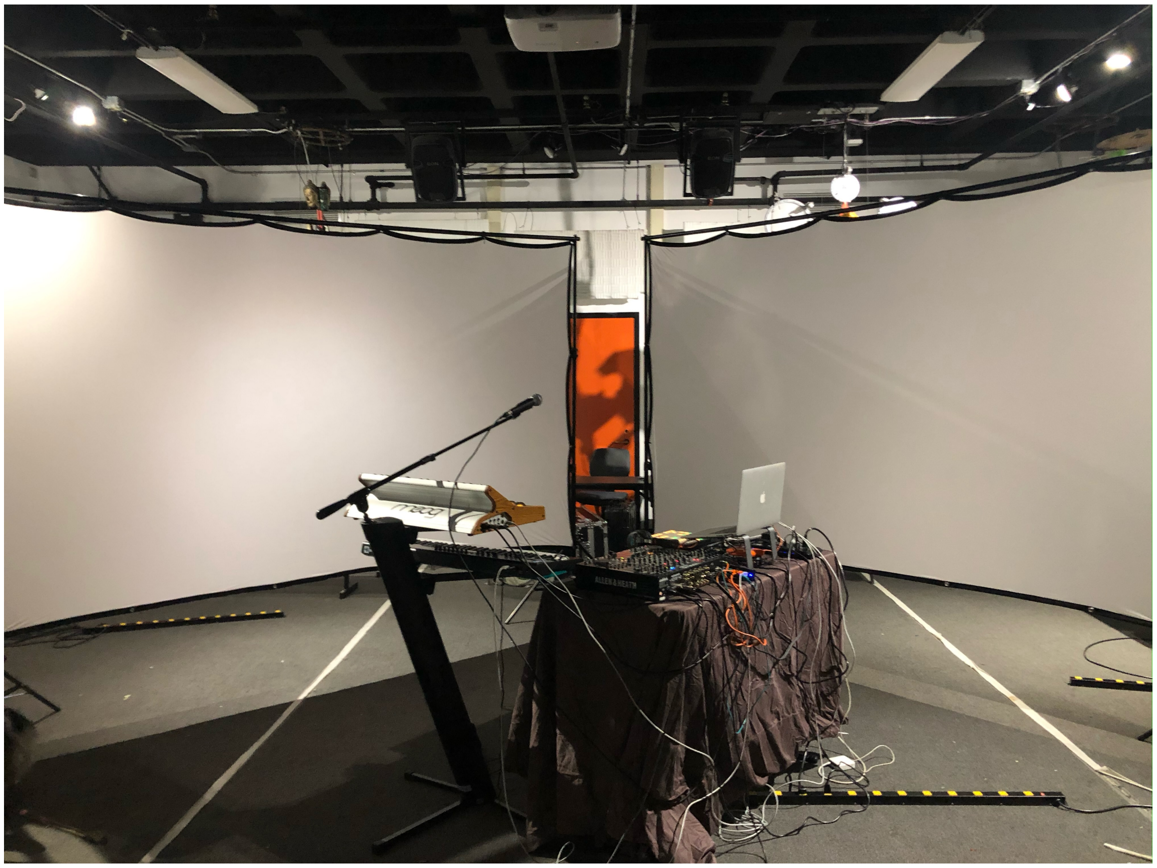


Figure 4.15 Rehearsal in the Machine Lab

Rehearsal was a bit of a different beast. I hadn't fully tuned my final set up and a lot of my first rehearsals at home were hours of me scratching my head in attempts to figure out why something wasn't working. Like much of what we do in Music Tech, there's no tutorial for custom set ups. But quickly, levels were adjusted, effects were swapped out and parameters were set up and moved around. Soon all that was left was just nailing down my own timing.

Figuring out where I had to be and when, was a large portion of this set up. I was juggling more than I ever had and it was as challenging as it was fun. Because I wanted as much physical control as possible to make choices on the fly, I also wasn't using the DAW as something to lean back on. I used very few MIDI messages outside of basic played note values for the VSTs. This meant that when I muted a track, I needed to turn it back on before it would work again. If I slammed a parameter in a certain direction, it wouldn't snap back the moment I switched to a different scene. This set was different than any I had played before, where maybe at certain points my timing had been critical for a few moments to minutes, but in this arrangement my timing was crucial for the whole hour and a half.

The performance was a dance between what changes were critical and what adjustments I could play around with for fun. The month of January in 2020 was spent fine tuning my important timings and learning where I could lean into elements to play around with. I had gotten it as close to perfect as I could on my own. It was time to move onto phase two.

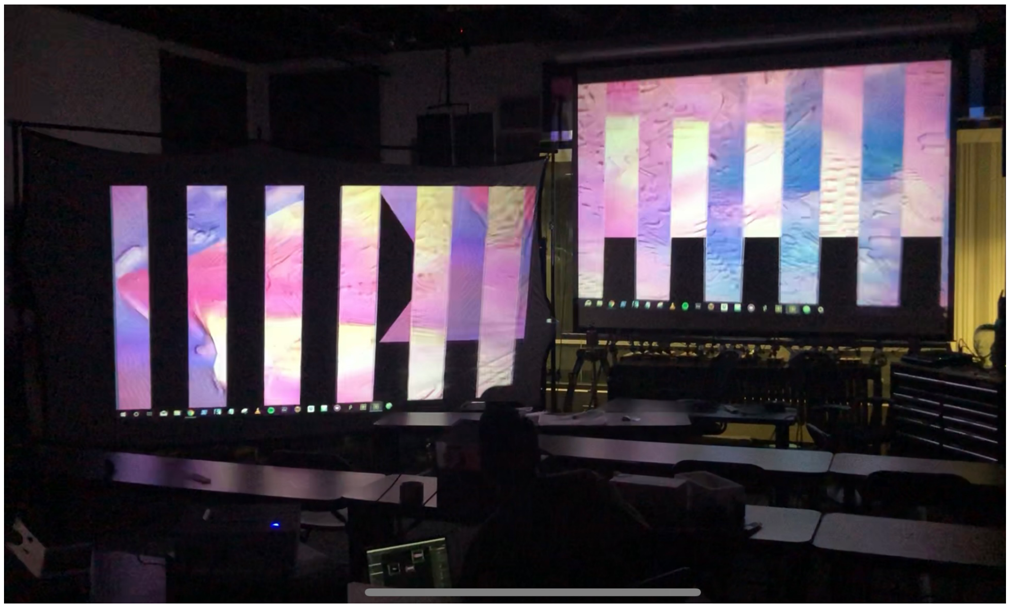


Figure 4.16 Previewing the Visuals Behind Mid-Res

During my rehearsal time, I had been sending Kai finished versions of songs, or clips and bits of sections so that he would have an idea of what he would be working with. The whole time he had been working behind the scenes on his own, honing his own skills as a visual programmer. All the while he had also been working on his own Wave Cave Installation. So, when I sat down to watch these visuals with him on a weeknight in the Machine Lab. I was absolutely blown away.

So, a few days later I crammed everything in my car and drove to school for one of the first of a handful of dress rehearsals. Kai and I set up the screens and projectors and I put together my musical set up while he calibrated the visuals. Then we settled in for a few hours of work, fun work, but work.

We started out by running through the entire set, timing it down, first so I would know how long it was for technical purposes and so that Kai would know where his cues were. Afterwards, we would work on through the performance section by section and often times students who had been hanging out on campus late would come in to watch. The students who came to watch the rehearsals, since I was facing away from the visuals and had *no clue* what was happening behind me, would help Kai with constructive criticism. I couldn't provide any feedback to Kai, yet he could provide helpful feedback to me. Having multiple sets of eyes made a huge difference in getting this project from its beginnings to the polished set that it was.

What made the collaboration with Kai particularly meaningful to me was that at no point did I send him data for his visual performance. He was entirely performing his visuals as an instrument, jamming with me. He would watch me, feel out the music and work out his timing based on our rehearsals, not OSC or MIDI or anything of that sort. It was truly showcasing not just my musical work, but also Kai's visual work in congruence. Something about the fact that we were two musicians playing together felt more special than just having designed visuals playing behind my music.

Our rehearsals would run late into the night, many times ending after three in the morning. But by the time it was a week away from the recital, we were stoked. Kai had helped me fine tune certain elements in my set as well as his own performance and we were ready to bring our work to the ROD. I celebrated my birthday rehearsing on March 3rd and was ready to share the performance.

4.2.4 The Performance

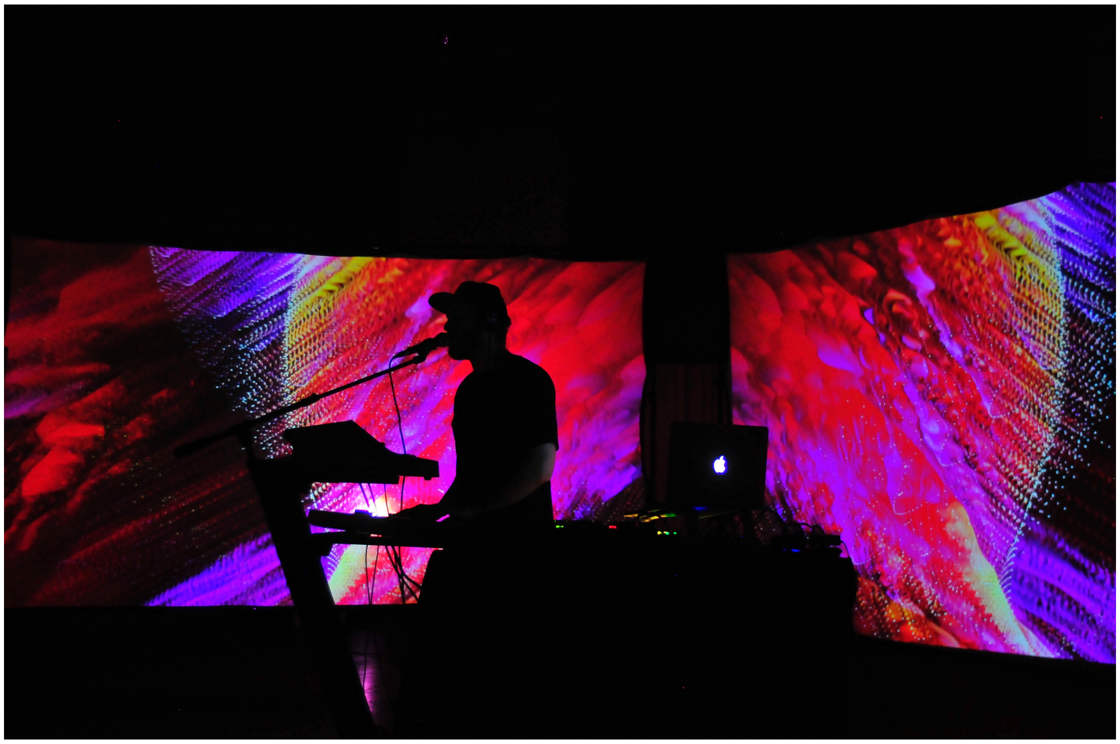


Figure 4.17 Photo of Mid-Res Recital

March 7th, 2020, I woke up with one of the worst sore throats *in my life*. How could this have happened? I had been so careful, taking vitamins, washing my hands. A novel new virus was turning up on the other side of the planet, could it be that? I told myself I wouldn't think about it until I finished my performance.

I arrived and with some help from the ROD crew unloaded gear from the parking lot to the venue. Set up was uneventful, everything worked minus a couple tweaks here and there. A couple changes to the gating, some changes in the Eq and levels while my parents set up snacks and drinks for the attendees. It was time for all the work we've done to come to fruition, we hoped.

The room was pitch black and we were ready to go. After all that work, I was a little tense, hoping it would turn out exactly how I wanted. Of course, it didn't. My sore throat was pitchy, and I found myself constantly out of breath. However, after the performance I got nothing but praise for my singing… thank God for auto-tune. I made mistakes in my timing, but since it was formatted similarly to a DJ set… nobody really noticed.

Of course, some of the levels were off. Of course, the audience being seated didn't let them fully engage with the some of the dance themed music. Of course, some of my patches even after checking them decided to flip out. But what I gained the most from this performance is a deeper understanding of what I wanted in performance.

If I walked away from this performance with anything, it was a lot of new information I hadn't had before. Primarily, I learned my limitations and the limitations of performances of this nature. The mistakes I made were of course a product of my own nerves, but also of the creation that I had put together. Perhaps I was a little bit over eager to do too much.

The room being completely dark made connecting with audience a challenge. If I had had a larger budget, I would have loved LED screens or stronger projectors, but the ones we had served their purpose. The visuals looked stunning, and the silhouette effect was strong, but the ROD is darker than the Machine Lab and I found myself stumbling to find some of my parameters.

Simplicity is something this rig didn't have. I didn't have anyone to assist with changes or adjustments. It was all on me. The positive response from the audience afterwards, reminded me of one super important thing. Nobody knew even a fraction of what I was doing up there. They knew I was playing piano and singing. They knew I was adjusting some synthesizer stuff. That was the extent of it. So many of the parameters in the performance weren't important. They didn't do anything that engaged the audience, they were for me not them.

Reactions came from big changes in small moments, a knob would turn and have a great effect. But muting and unmuting instruments in a compositional way had little to no effect on the enjoyment of the audience. I could have automated a gigantic chunk of this performance, and no one would have known any different, and I would have made less mistakes. It would have also helped me connect with the audience better. However, all the same it went well and was received positively.

The conclusion of this performance was met with the end of the life my old MacBook Pro and was the final performance for my Xone 4D, due to it being phased out of use in newer operating systems beyond Mac OS Mojave…

It was also the end of life as we knew it!

4.3 COVID

On March 10th, three days after my recital. The whole world entered a lockdown to try and prevent the spread of COVID. And a little over a month later, my father passed away from a sudden accident. With the uncertainty of the world and my own personal life, I deemed it was important for me to take some time off if I wasn't going to be able to give my all to my Performer-Composer degree.

4.3.1 A year of leave, but not leaving

During this time, my partner Anais, and I had begun putting out music under the moniker, tau0n. We were invited to play live streams on channels run by large event companies such as Insomniac and Desert Hearts. Live music was all but a thing of the past. Live streams were how we enjoyed performances, and they grew more and more immersive. Some artists dove into the realm of virtual reality, others added interactive features. But what was clear was that to be an artist, you had to know how to use streaming platforms.

We ran our own streams, learning how to operate YouTube and Twitch, OBS and stream-keys. It was totally new territory, but at least we could still play music. As time wore on, I found myself working more as an engineer, mixing songs for friend's record labels. I found myself running live streams. The live set I had spent so much time fine tuning was shelved. Instead, we were releasing music on other people labels, and had started releasing other people's music on our own labels.

As time wore on, as people grew less scared. The work that we had put into tau0n put us in a great position to start playing shows. In 2021 we played shows across California, Utah, Colorado, and Vegas. We even played a few with a lightweight portable version of the mid-res live set, on a working computer of course.

Figure 4.18 Compilation of images of tau0n performances

By the middle of the year, I wasn't sure I would be returning to Cal Arts unless I had a reason. What I remembered about the music program was that there was something different about Music Tech. It wasn't heavy academia about old dead people and abstract art (pardon me). It was hands on and cutting edge. The stuff that students like Kai had been working on was all stuff that was applicable to what I was doing at that time.

Would learning how to make my own visuals help me? Absolutely. What about building small interfaces that I could bring on the road? One hundred percent. How about helping me find solutions for questions I didn't even know I had yet? That's a little philosophical, but sure. I was already composing and getting paid for it. I was performing on the stages that I wanted to be.

So, after 3 semesters off, I changed my major from Performer-Composer to Music Technology and returned to Cal Arts.

Chapter 5
Where Art Intersects Science

My first semester back at Cal Arts, I was a little lost. I started in spring, so I didn't know my classmates. I wasn't sure what classes I should be taking and I was entirely out of my element. I didn't know what I wanted fully out of my degree, and if it wasn't for branching out and taking classes that I really wasn't sure about, I don't think I would have figured it out.

5.1 Surrounding myself.

I have always been curious about surround sound, ever since I fell off a fifteen-foot ladder installing a Dolby 5.1 speaker system with my dad in his living room. When I was introduced to 4DSOUND via Max Cooper, I was enamored by it. However, Cooper had a PhD in computational biology, and I barely scraped my way through my undergrad in business. It was out of reach. I could never do it. The applications seemed so complex and grand. The performances I saw in 4DSOUND seemed so calibrated and well, scientific. Was there any place for me in this?

5.1.1 Early Conceptions

My earliest desires to perform in surround sound came long before Cal Arts. When I performed with bands, I always considered what the effect of spatializing sounds would create during a performance. As music producers were often focused on widening sounds and creating spatialized simulations in stereo, and as a DJ and performing electronic artist I had access to hundreds of parameters, but nothing for any type of even the simplest spatialized sound.

The spark that started the concept behind the application/plugin I am lovingly calling the *Surround Sounder* started after my re-starting of Cal Arts. While taking Cristian Amigo's Research Based Sound Design, and James Wiley's Immersive Experience Design courses my spring semester of 2022, I was reintroduced into concepts that had left me while DJing with tau0n. Sound, while yes, it is a commodity, comes in many different forms of consumption. During RBSD (Research Based Sound Design), Amigo covered topics relating to the human relationship of sound from listening on headphones to the design of temples that induce sub-sonic frequencies that cause different states in our consciousness. However, one assignment had us bird watching.

It wasn't exactly bird "watching," but more bird "listening." Using an application called Audubon we were tasked to go out into nature and record and listen to bird calls and recognize them using the app. The assignment wasn't really about the birds, it was about introducing a new concept, and in this case the concept was a new way of listening. Listening actively, not passively. The terms are self-definable, Active Listening is listening by choice, focusing in on the sounds around you. Passive listening is listening not by choice, instead choosing to let the signals of sound you hear just happen, possibly because you are preoccupied with something else. However, something about this assignment reminded me the importance of active listening, as well as pulled something up from deep inside of me that was curious again in the effect of sounds and their positions in the space around us.

Tasked with coming up with different ideas for immersive theater designs as a final project, I instantly gravitated toward the idea of a digital sound bath. Sound baths are used for both recreational and therapeutic purposes to influence the mind. However, the typical sound bath is made up of simple live instruments such as gongs, chimes, singing bowls, percussion and sometimes maybe guitars or other performative instruments.

In some instances of a sound bath, the leader of the group will sit in the center with their instruments, but often these are small intimate experiences that are for only a few individuals. The larger, group experiences often have the instruments arranged in an array along the outside of a circle and are played by a group of performers, sometimes asynchronously or at the behest of a conductor. Many times, due to the nature of instruments being physical, an instrument can be picked up and walked along the outside of the circle, changing its position in space, and adjusting the listening experience.

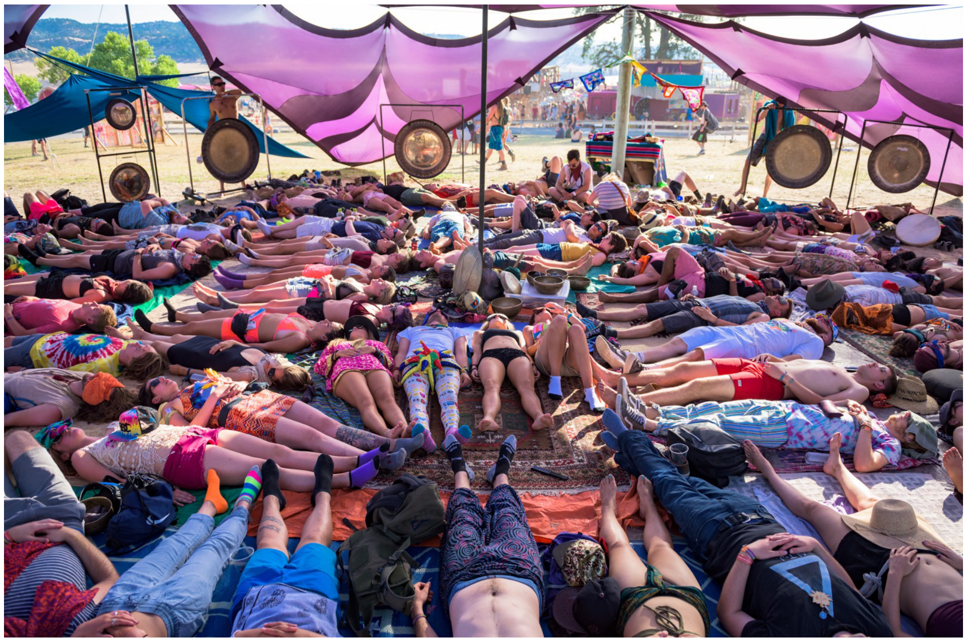


Figure 5.1 Example of a Sound Bath Experience

The idea behind the *Digital Sound Bath* is that instead of the instruments being physical objects in space, they are output channels from a DAW, in my case I believed I would use Ableton Live. An array of speakers would then be set up along the outside of the circle. Using an audio interface with multiple output busses, I would connect each speaker to a corresponding bus. The busses would then be configured in order inside of Ableton Live. While it wasn't necessary to put the performance on, I had begun attempting to figure out locations and who I could borrow speakers from to attempt an initial set up as it seemed like something I was interested in pursuing.

At the time I was unaware of the class of plugins, called Surround Panners. My only foray into Surround Mixing was with Logic's surround feature. And when I googled Surround Sound plugins, for the most part sound channels were treated like objects on a two-dimensional plane. The X axis was left and right and Y axis was front to back. To move sound amongst an array of speakers, every sound object would require two parameters, a knob for X and a knob for Y.

As a performer, this was unacceptable for live usage. Human beings only have two hands, so this meant that only one sound could be "panned" at a time using this technique, and it would also have a hefty mental bandwidth when adding more variables into the object position of (X, Y). Many plugins had multiple parameters per speaker, and the simplest had just X and Y faders. It needed to be much easier.

The format had to be scalable. With as many audio samples, MIDI instruments, or even live instruments as I wanted to be introduced into the structure. And in the vein of Jordan Belson and his *Vortex* knob, it was crucial to be able to control the position of each instrument and channel within the array of speakers using physical knobs that adjusted parameters using MIDI, like all my previous electronic performances. There was just once crucial flaw, *I had zero clue on how to create this effect.*

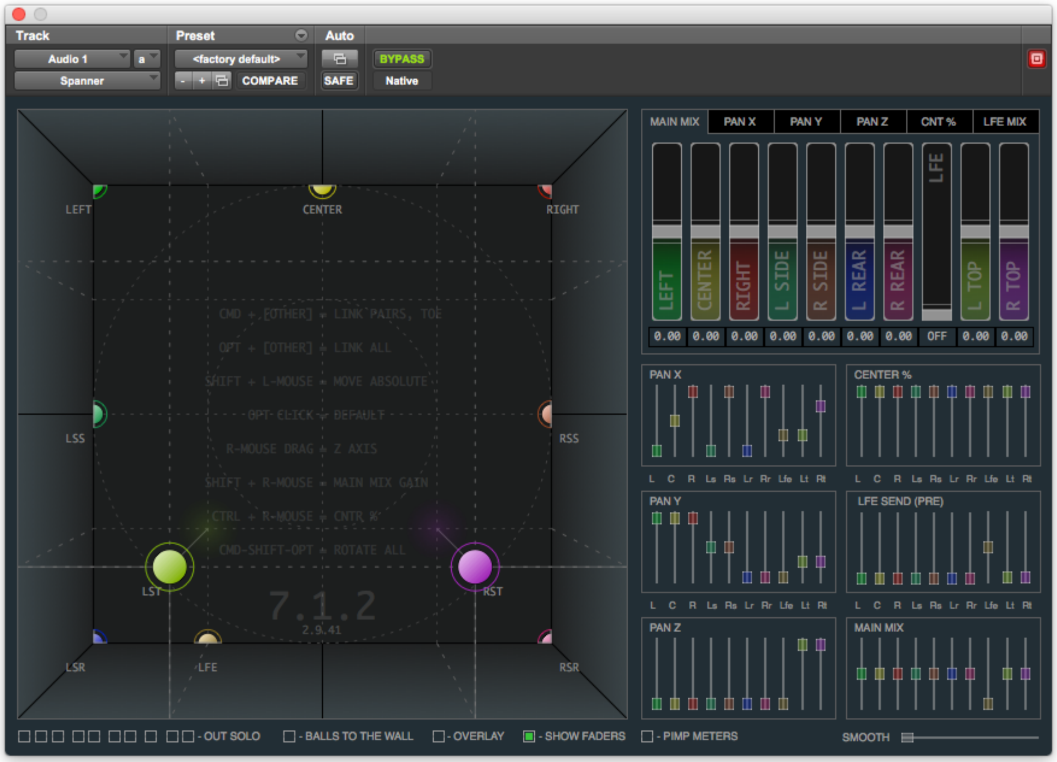


Figure 5.2 Example of a Complex Surround Sound Plugin

I came up with some concepts at first that were band aid fixes. First, I considered setting the ranges of volumes/bus sends within the MIDI mapping settings within Ableton.

This would address the problem, but it wouldn't be scalable. The math would have to be reapplied for every new speaker. Additionally, MIDI has limited resolution, so the more speakers I would add, the less smooth my panning adjustments would be. The second solution was just to mute and unmute bus sends for each channel, which would be much easier to scale. But it wouldn't give the feeling of panning.

I recalled a distant memory, showing up early to class one day in the Machine Lab, back before COVID and accidentally stumbled into the end of a class that had been using C++ to teach Music Tech students how to make a volume knob. Now that I had changed my major, I decided that the best course of action would be to develop this plugin for myself. How hard could it be?

5.1.2 Breaking Down the Surround Panner

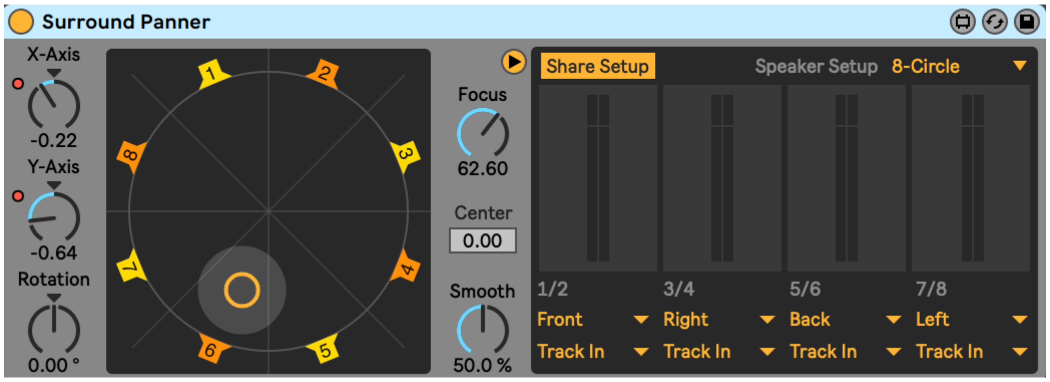


Figure 5.3 Ableton Live's Surround Panner

Ableton Live provides a Max for Live Addon plugin called Surround Panner that can be downloaded from their site. When I learned about this plugin, I almost stopped work on my application. It seemed to solve my problems. It was the simplest implementation of a Surround Panner that I had found. While a channel was treated like a sound object, moved along an X and Y plane, it also had a rotation knob that functioned similarly to how I wanted.

The rotate parameter would slide the sound object, the yellow circle, along the diameter of a simulated ellipse that was the distance of the sound object from the center of the circle. This would give me the single knob control effect of panning the sound along the array of speakers. However, Surround Panner has a long list of limitations that I wanted to eliminate from my plugin.

First off, this Surround Panner is only useable inside of Ableton Live and only useable in its own specific configuration within the Ableton Architecture. Also, there are only a few layouts. It is useable in four, six and eight channel output set ups. Also, it treats the stereo channels as mono, compressing them and then iterating them along the array of speakers. An issue with this would be, what if there was a song that was mixed with widely panned instruments? The original stereo signal would be lost and turned into mono. Sounds that are explicitly mono could be played and panned along nicely, but not stereo mixes. The plugin does function well as a feature to be automated within Ableton Live. But as a performance tool, it was lacking.

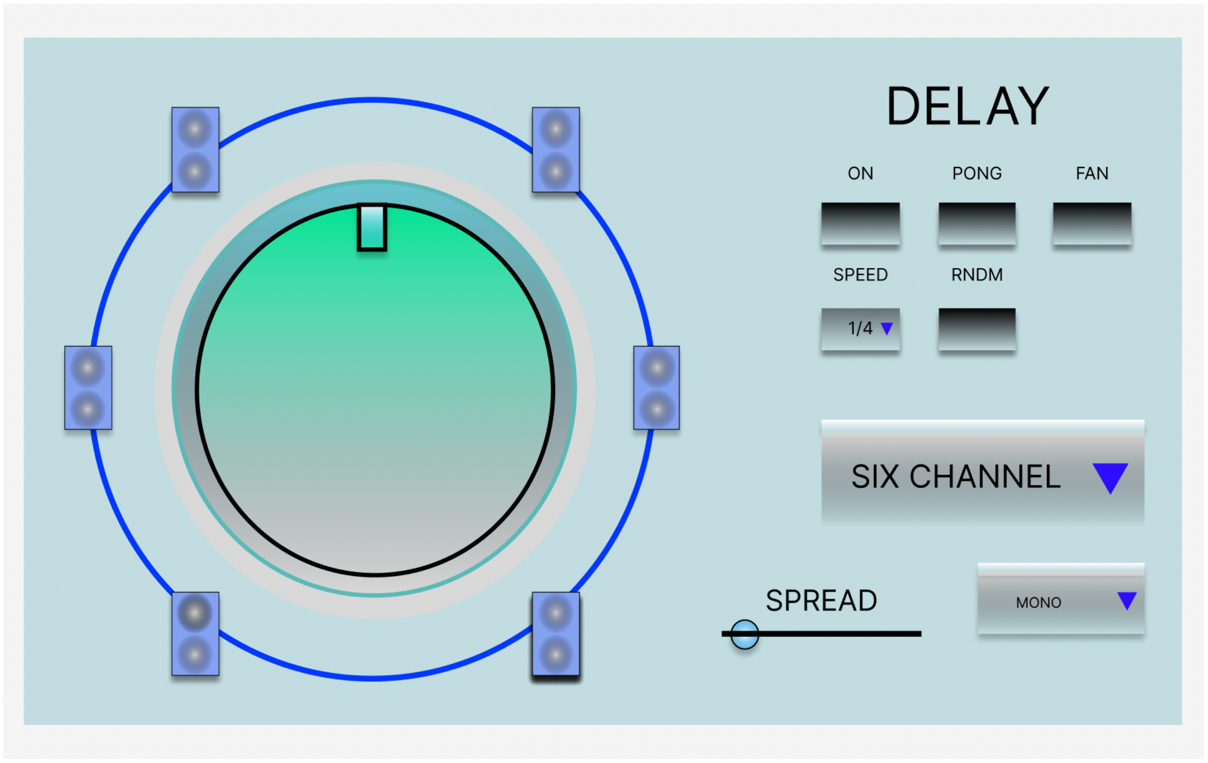


Figure 5.4 Visual Prototype of The Surround Sounder

My first inclination when conceptualizing the Surround Sounder was to build just a single knob, to pan the sound. However, since this did already exist in some form, I decided it would be better to expand upon the features that I, as a performer, would like when playing a set or providing a digital sound bath. In terms of mono and stereo, there should be a feature that allows the plugin to understand whether the signal is stereo. If the signal is highly stereo, it should be broken down into two signals, a left signal, and a right signal. They would be treated as two separate sound objects along the path, modulating accordingly.

Additionally, while Ableton Live's Surround Panner had numbered speakers around the outside of the X and Y grid, those numbers were purely ornamental in terms of interaction. The sound object's relationship can be seen when it moves along the plane, and the sound object can be moved along the plane when it is interacted with by the mouse. The numbers themselves are not mappable parameters. Snapping an object to a specific speaker requires automation of the X and Y knobs. It was important to me that each speaker location on the Surround Sounder could be assigned mappable parameters so that sounds could easily jump from speaker to speaker without the need for a mouse or prior automation.

Ableton Live's Surround Panner is useable on any instrument channel or send bus. As a matter of fact, it comes with a preset Live Set that includes a reverb send. The reverb send can then be automated, or manually panned along the speaker Array. What this doesn't allow for though is natural sounding decay on each channel as the Reverb iterates along the speaker array. The sound moves from one speaker to another as desired but doesn't linger. The volume is either up or down. This would be an easy fix, just add a reverb to each channel so that it is triggered as sound input enters and exits. But the layout of the Surround Panner treats the channels as groups. Speakers one and two are grouped into channel one which shows outputs 1 / 2. So, adding stereo effects such as delay or chorus, can only effect two channels at a time. In the case of a delay, eight channels are combined into the four busses on which you can apply a stereo delay (such as a ping pong) which means that the busses will effectively function as a stereo pair and not as either individual speakers or as a greater part of the speaker array.

This led to the concept of the Simple Surround Delay. The delay itself would operate as a basic single channel delay. Without adding any other parameters, a sound could play in its place on the array and have a delay effect. Changing the parameter to fan would cause the delay to fan out in two directions from the single source. Pong would create a ping pong effect delay coming from the individual source and random would randomly send the delay effect to different points upon the speaker array. This way instead of needing multiple stereo effects, the plugin itself would be a surround delay, allowing for more performance options.

As stated before, the Surround Panner isn't very modular since it only allows for layouts of four, six and eight. In many cases, especially for personal use, this number of speakers is more than adequate. For the application to be scalable for professional uses as well as personal, such as in cases like the multiple hundred speaker array that was presented at the World's Fair at the Phillips Pavilion, the available outputs should be based on the limitation of the DAW, not the plugin. The plugin should be able to configure for as many outputs as needed, even number or odd, that allow for the sound to be panned across.

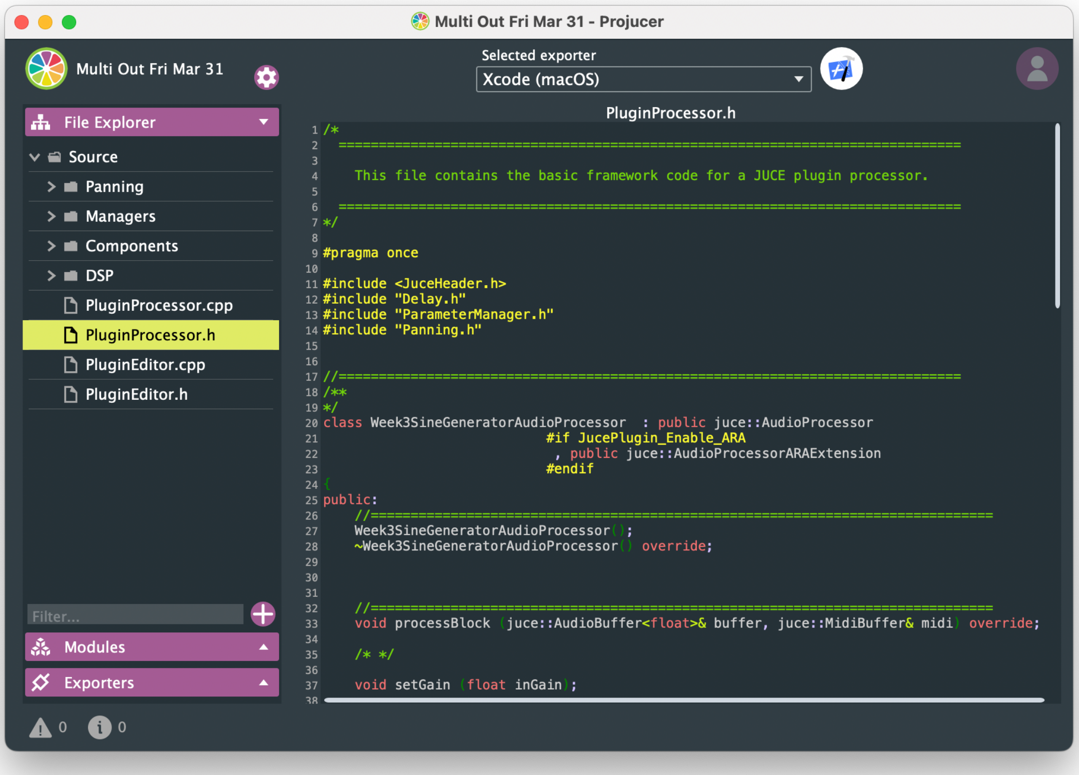
Finally, I wanted a different effect when it came to volume and spread. The Ableton Live Surround Panner operates as an X and Y grid, which gives the user the ability to move the sound object towards the middle of the circle. While some people might like this added parameter, it also leaves open space for mistakes. After getting a proper mix of a live performance, if the value of either X or Y changes, the volume of the sound object can change. The closer the sound object goes to the center of the circle, the quieter it gets. This is a redundant function as a parameter mapped to a volume fader inside the DAW could achieve the same effect without having any correlation to any pan position values.

The spread effect, however, should function similarly. Increasing the size of the sound object. In the Surround Panner, the pinpoint of the sound object is adjusted and as the object grows, the sound bleeds further out into more speakers. In the Surround Sounder, the object will grow and bleed out to more speakers in the array so that a single point will fill all the speakers so that the sound is coming from all directions. This feature also acts as a basic volume automation, since a sound playing out of more speakers naturally becomes louder. There are also plenty of solutions to account for this change in volume, if necessary, that don't require X and Y axis.

5.1.3 Early Attempts at Programming

Coming from a background of performance and composition, I had little to no idea where to begin. My inclinations were either to start in Reaktor, which I had been comfortable with a couple years prior but hadn't used in a while, or to try and learn Max. After consulting some professors, I was led to my solution, the JUCE framework. At the time, learning Chuck and Python, I figured that C++ would be no big deal as the other two languages had some degree of a relationship. I underestimated the fact that having taken p5js nearly two years prior left me out of my element entirely since I had spent barely any time learning coding environments. How difficult could it be to make a simple knob? Oh well let me tell you.

There have been few times in my life where I have felt utterly and absolutely stumped. This is one of them. The only easy part was the process of installing JUCE, after that it was chaos. With an elementary understanding of statements and functions, I googled how to properly set up Xcode and watched internet clips on basic JUCE. With no understanding of the JUCE modules or the JUCE source code I spent way too long just trying draw a fader.



```
/*
  ==============================================================================

    This file contains the basic framework code for a JUCE plugin processor.

  ==============================================================================
*/

#pragma once

#include <JuceHeader.h>
#include "Delay.h"
#include "ParameterManager.h"
#include "Panning.h"


//==============================================================================
/**
*/
class Week3SineGeneratorAudioProcessor  : public juce::AudioProcessor
                            #if JucePlugin_Enable_ARA
                             , public juce::AudioProcessorARAExtension
                            #endif
{
public:
    //==============================================================================
    Week3SineGeneratorAudioProcessor();
    ~Week3SineGeneratorAudioProcessor() override;



    //==============================================================================
    void processBlock (juce::AudioBuffer<float>& buffer, juce::MidiBuffer& midi) override;

    /* */

    void setGain (float inGain);
```

Figure 5.5 JUCE's Projucer Application

Without really knowing where I was going, I stumbled through terminal commands on a new MacBook Pro to properly allow Xcode to run code directly through Ableton Live. I spent way too long trying to operate the JUCE audio plugin host, before later being told that it wasn't even really a necessary application to learn at all. But each step of the way I gained something new, some semblance of a piece of a puzzle that was being compiled in my own brain. Nearing the end of fall semester and knowing that upon the start of the new year I would be shifting into high gear, I decided to take additional online courses in C++ and thankfully I was able to build a fader.

The joy that I felt when that little fader box appeared on screen was more than I had felt in a long time. I spent more time looking at a single webpage, than I ever had before [65] and the concepts behind it were so foreign to me I didn't think I'd ever even start to

[65] "Ecosystem | JUCE C++ Library."

understand it. But with help from some classmates and my Mentor, Michael Leisz, things were starting to take shape.

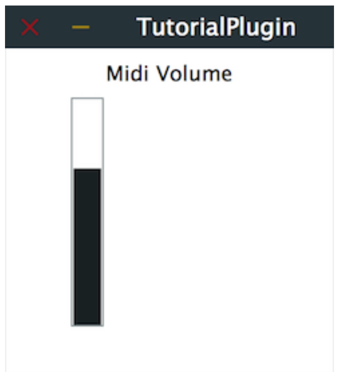


Figure 5.6 The Tutorial Plugin That Took Two Months to Make

5.2 Where Do I Plug This Thing In?

5.2.1 Basic Math

Had I started learning C++ before I began attempting to follow the tutorials, and had a stronger understanding of namespaces and variable types I would have had a far easier time getting started in coding this project. Despite having followed all the code almost perfectly in a blank project, I was still unable to get a working application. I was being held back by my understanding of basic concepts such public and private variables, as well as the need to use namespaces. As a matter of fact, I wasn't quite sure what the difference between a PluginEditor and PluginProcessor, despite having had it explained in JUCE's tutorials.

So, with my Mentor we began to conceptualize what the Math behind the application might be. Firstly, what was the expected behavior? And secondly, what would it take to make that happen?

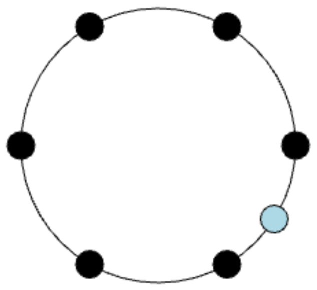
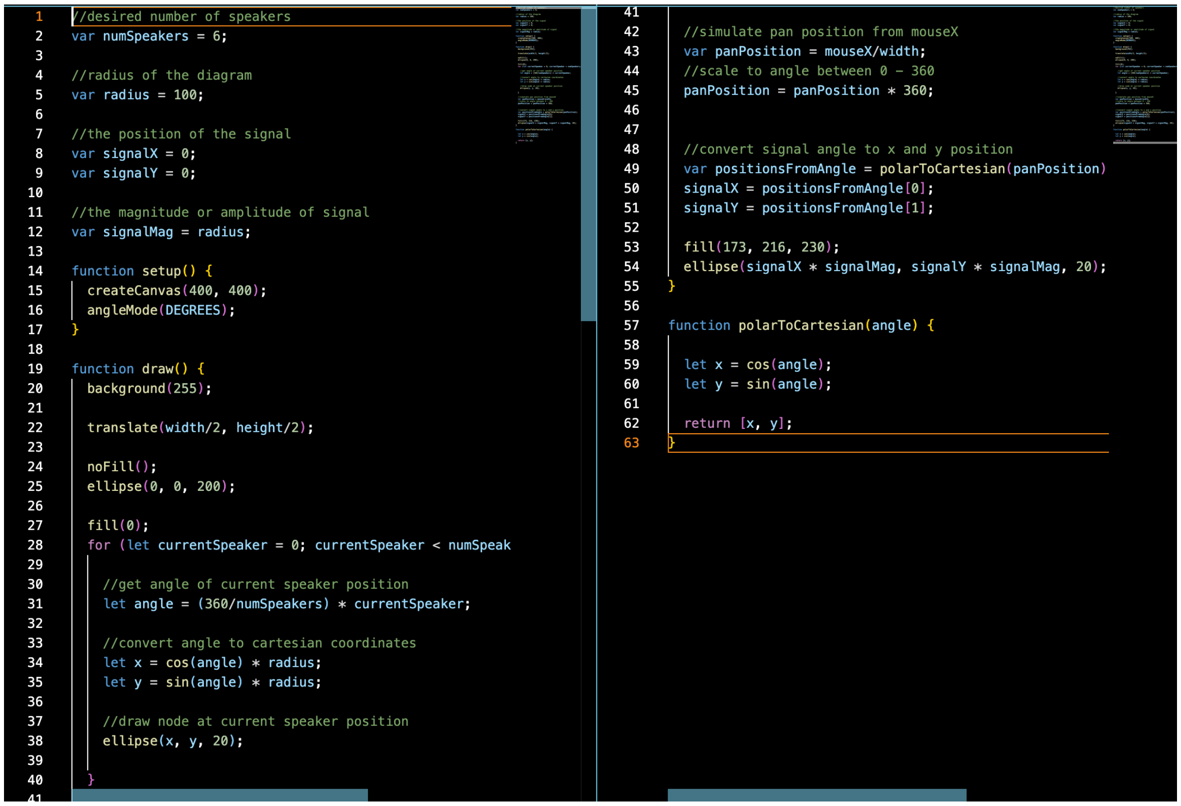


Figure 5.7 Demonstration of Code Behavior in Visual Studio Code



```
1   //desired number of speakers
2   var numSpeakers = 6;
3
4   //radius of the diagram
5   var radius = 100;
6
7   //the position of the signal
8   var signalX = 0;
9   var signalY = 0;
10
11  //the magnitude or amplitude of signal
12  var signalMag = radius;
13
14  function setup() {
15    createCanvas(400, 400);
16    angleMode(DEGREES);
17  }
18
19  function draw() {
20    background(255);
21
22    translate(width/2, height/2);
23
24    noFill();
25    ellipse(0, 0, 200);
26
27    fill(0);
28    for (let currentSpeaker = 0; currentSpeaker < numSpeak
29
30      //get angle of current speaker position
31      let angle = (360/numSpeakers) * currentSpeaker;
32
33      //convert angle to cartesian coordinates
34      let x = cos(angle) * radius;
35      let y = sin(angle) * radius;
36
37      //draw node at current speaker position
38      ellipse(x, y, 20);
39
40    }
41
42    //simulate pan position from mouseX
43    var panPosition = mouseX/width;
44    //scale to angle between 0 - 360
45    panPosition = panPosition * 360;
46
47
48    //convert signal angle to x and y position
49    var positionsFromAngle = polarToCartesian(panPosition)
50    signalX = positionsFromAngle[0];
51    signalY = positionsFromAngle[1];
52
53    fill(173, 216, 230);
54    ellipse(signalX * signalMag, signalY * signalMag, 20);
55  }
56
57  function polarToCartesian(angle) {
58
59    let x = cos(angle);
60    let y = sin(angle);
61
62    return [x, y];
63  }
```

Figure 5.8 The code that drew Figure 5.7

Figure 5.6 shows a finalized version of the application behavior. After a few different variations started by Michael Leisz, this final version properly defines the behavior that the basic panning knob in the Surround Sounder should accomplish.

We declare the number of speakers to add to an array, this can always be changed either manually or through a separate function. Then we define a variable for the size of the knob and variables for the starting positions of the signal before defining the amplitude of the signal, which would remain the same as the ellipses. After which we set up a canvas that is a small square and set our angle mode to degrees instead of radians or floats.

Inside of a draw function we set the background to white and create an ellipse with no color inside of it that sits directly in the center of the canvas. Then inside a for loop we create another ellipse that acts as a pointer to an evenly laid out array of speakers along the outside of this ellipse calling the angle function set up starting on line 57. This function will be crucial in building the layouts of the Surround Sounder and setting up the snap to speaker parameters. After this for loop we grab the position of the mouse on its X axis and treat it like a vertical slider for panPosition. We create signals for the X and Y axis and correlate them to the angle and draw an ellipse that follows along the outside of the circle.

In its simplest form, this is the idea behind the Surround Sounder.

5.2.2 What Route?

The concept of gain staging is simple and while it doesn't directly apply to this application, it does apply in the sense that there is a multiplicity of ways to adjust volume automation. Gain staging, or the process of adding (and subtracting) volume through multiple sources, Adjusting the volume of a signal to the desired rate can be done in almost any available plugin. You'll likely see a volume or gain knob on most if not all your favorite digital audio plugins. [66] But what does this have to do with panning audio you ask?

As we've seen in chapters one and two, the simplest stereo panning algorithm was at its core just a volume automation relationship between two speakers. In more complex systems, such as Dolby Atmos, the panning systems became highly advanced relational volume automation systems. While the brain might be thinking about sound as an object in relationship to space, that's not the case. The sound inputs are static, the panning system is just adjusting the volume feed to different speakers. Ableton live in its raw performance mode, can adjust the volume in four ways. Three times before reaching before the audio reaches a bus output and the total maximum volume can be adjusted on the master fader afterwards (but this wouldn't be relevant for my application).

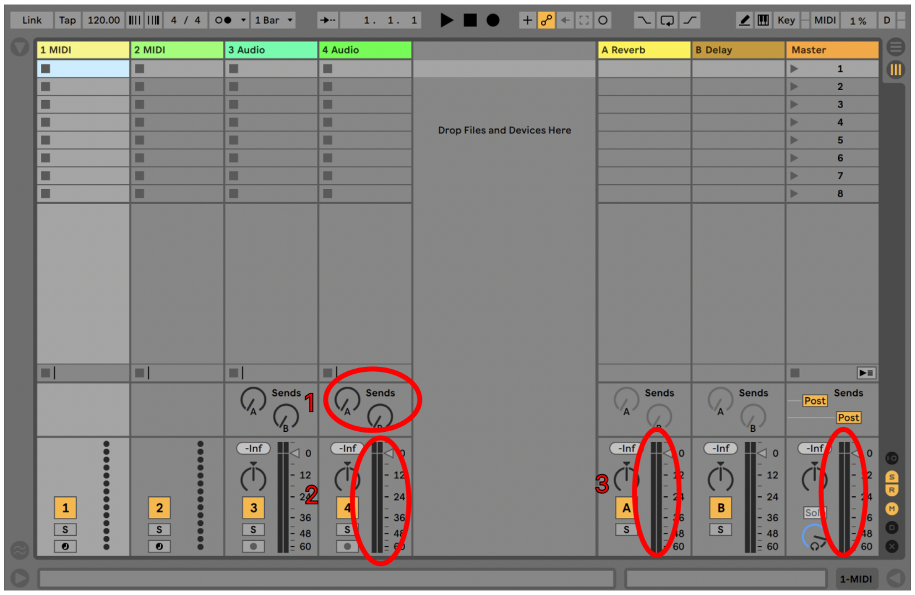


Figure 5.9 Raw Ableton Volume Parameters

Automating directly inside of live seemed like the simplest option as there are many ways to adjust parameters inside of live using just MIDI mappable devices. The concept was to control the return sends, treating the returns as individual busses. A single knob would send MIDI values based on the code in Figure 5.7 that would be applied as volume automation to the bus sends. The created return tracks wouldn't be reverb or delay, they would be speaker one, two, three and so on. The panning effect would be applied by adjusting the amount of send to the individual busses via these MIDI mapped parameters. However, this went back to my first concept of setting MIDI mappable parameters that

[66] "Gain Staging."

would have taken me back to the issue relating to templates and having to apply MIDI mapping and parameter adjustments every time a change was made to the Live Set.

The solution that I conceptualized for this problem was to use OSC (Open Sound Control). OSC was introduced as an alternative to MIDI, providing higher bit resolution packets for more precise parameter control. Also, OSC is sent over network such as UDP/IP or Ethernet, and while it can be sent via WIFI that's never a great idea. OSC would solve my problem relating to the parameter set up inside of Ableton and would be able to achieve the same effect as manually inputting the MIDI parameters.

Except it wouldn't. First, although Ableton Live is beautifully and intuitively configured to use MIDI, Live has no native configuration in *any way* with OSC. To account for the fact that there are producers and performers who do use OSC however, Live provides a max addon called the Connection Kit. Connection Kit comes with a variety of plugins that allow for addons such as LEGO Mindstorm, Arduino and even a webcam can be configured as parameter control devices. There are also a few OSC helpers available. This was the bridge that I was looking for.

Except it wasn't. Connection Kit provided a basic OSC sender, but no receiver. Ableton Live could be easily configured to send OSC data to outside devices, but the native configuration in Live was as I early stated, was not designed to receive any sort of OSC message. There was a Touch OSC receiver however which seemed promising. My familiarity with Touch OSC came from the days before iPad when electronic artists such as Daft Punk and deadmau5 used a device called the Lemur in the mid-2000s. While Touch OSC came slightly later, and with the evolution of more processor strong iPads, Lemurs were phased out. However, the design behind custom touchscreen interface controls were always appealing. [67]

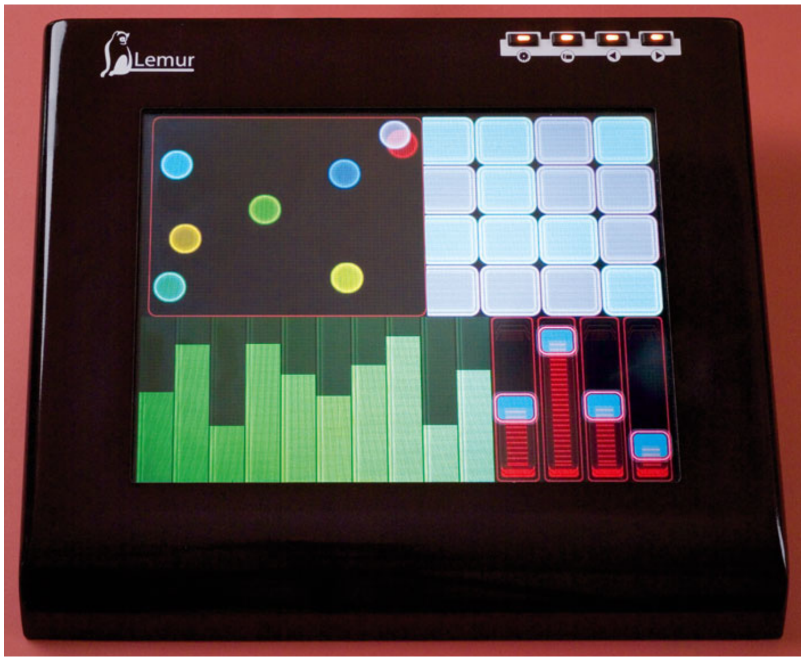


Figure 5.10 The Jazzmutant Lemur (A thing of the past)

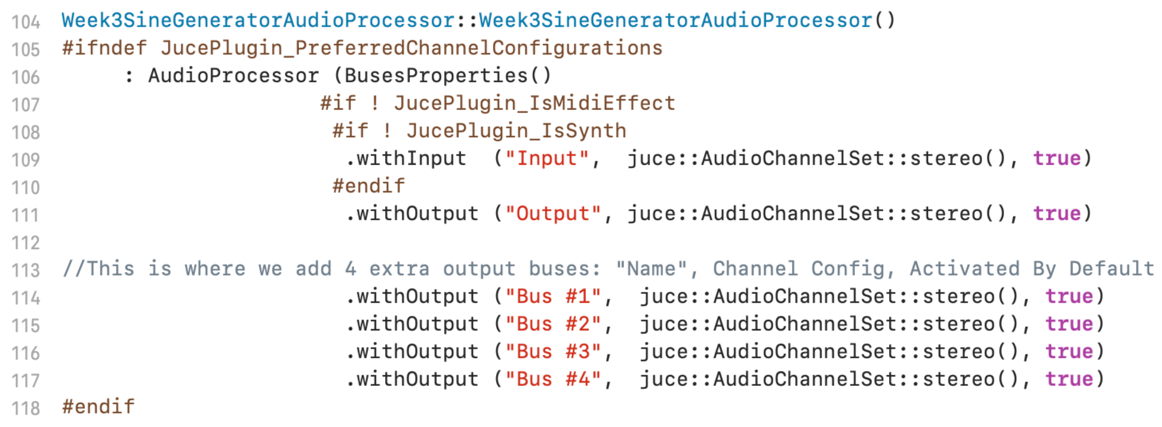
[67] "Jazzmutant Lemur."

However, while it seemed like building custom interface in Touch OSC would be possible for the concept I was working on it would have to remain open source and wouldn't fulfill the primary goal of keeping things simple. I wanted anyone to be able to use this plugin. Having it be a requirement to understand Touch OSC as well as Ableton Live's bridge, the Connection Kit, in order to use this plugin wasn't lowering the barrier of entry it was raising it.

After this I briefly toyed with the idea of building a plugin that could wirelessly, or locally networked, send RTP or UDP packets of MIDI. My thought process had continued to follow the same road of needing send control signals to the DAW, when had I understood the JUCE framework a little better I would have known there was a far simpler solution. There was a way to control everything within an application itself.

After many frantic emails on my part sent to an f extremely patient Professor Jacob Penn and the TA Zane Golas, I had my understanding of how powerful the JUCE engine was flipped on its head. While I had done some research on bussing and multiple outputs in JUCE, I hadn't been able to understand what their tutorials or site was describing about the framework. Since I was also building a rather niche application as well, there were no tutorials or even really a place for me to begin.

Busses were referred to as channels inside the of the plugin when I read the JUCE tutorial for bus layouts. The way that I understood that was in the simplest sense, routing. Signal could be routed from one module to another or sent to an additional effect such as reverb or delay when building a synthesizer. Well, I couldn't have possibly be more wrong. The JUCE framework had accounted for the desire for a plugin to have multiple outputs as well as inputs, and provided a way for signal path to flow cleanly through the plug in and out to the DAW.



```
Week3SineGeneratorAudioProcessor::Week3SineGeneratorAudioProcessor()
#ifndef JucePlugin_PreferredChannelConfigurations
     : AudioProcessor (BusesProperties()
                     #if ! JucePlugin_IsMidiEffect
                      #if ! JucePlugin_IsSynth
                       .withInput  ("Input",  juce::AudioChannelSet::stereo(), true)
                      #endif
                       .withOutput ("Output", juce::AudioChannelSet::stereo(), true)

//This is where we add 4 extra output buses: "Name", Channel Config, Activated By Default
                       .withOutput ("Bus #1",  juce::AudioChannelSet::stereo(), true)
                       .withOutput ("Bus #2",  juce::AudioChannelSet::stereo(), true)
                       .withOutput ("Bus #3",  juce::AudioChannelSet::stereo(), true)
                       .withOutput ("Bus #4",  juce::AudioChannelSet::stereo(), true)
#endif
```

Figure 5.11 Audio Processor Constructor Code for Multi-Channel Out

Right there inside the special member class constructor, we can add additional outputs. These outputs are sends directly from the application itself. These busses can have audio routed directly to them, volume automation applied to them internally and finally be passed directly to the DAW. A simple solution for a problem I had unintentionally made complex, it passed with flying colors. The routing inside Ableton Live was minimal, with only a few extra steps.

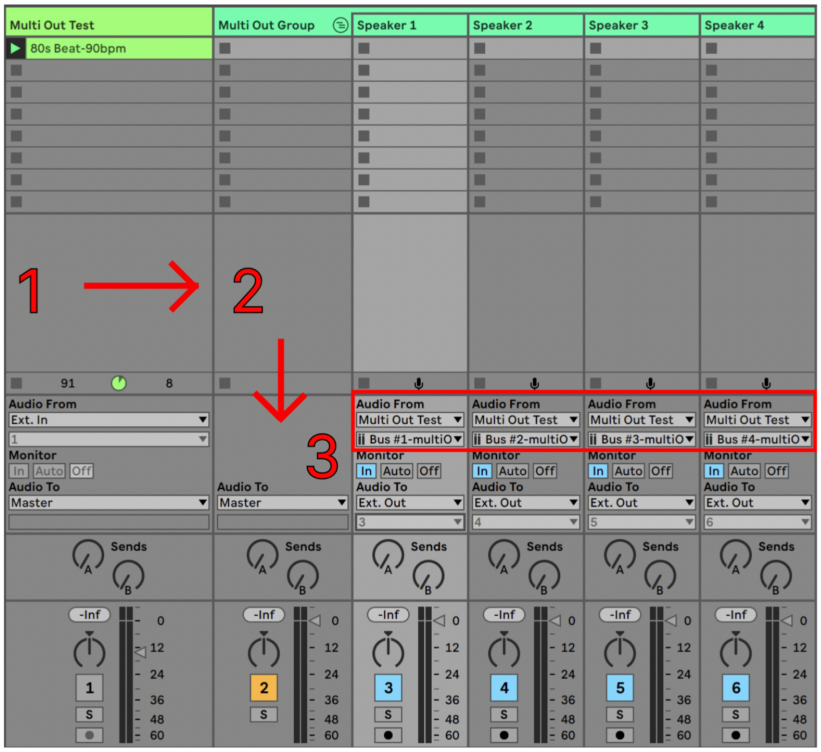


Figure 5.12 Simple Layout for Four Channel Output

The first step would be to create an instrument channel. Whether it was a synthesizer, sampler, or just plain audio in wouldn't matter. But on this track, we would place the Surround Sounder, which would grab input from this channel. Second, each individual instrument track would require the set-up of a surround group. In the case of Figure 5.11, there are four outputs so the I created four audio tracks and named them to correspond with their relevant loudspeaker. These tracks could then be grouped together inside of the DAW and collapsed in the performance view to conserve space if necessary.

Third, and last, the user would then configure their bus settings. This was the most complicated step, but the simplest of all my possible solutions. In the red box, Ableton Live provides an "Audio From" menu. Instead of taking audio from the channel itself, the user would configure the channel to receive audio input first from the name of the Audio Channel it was looking to receive audio data from. In the case above, it is "Multi Out Test." Then below, the user can choose which output bus to receive data from, in a basic set up the number of the bus and the number of the channel would be the same, for example Bus #1 and Speaker 1.

It's important to note that for the channels to work, they need to be set up for constant monitoring. If the channel is not set up for constant monitor, it will only provide audio output when it is armed to record. Since we aren't recording, nor are we technically using these channels as monitors, it is important that the value of Monitor is changed to 'in' so that the channel is constantly playing whatever audio is coming through the channel. The last step is to configure the Audio To-s. In this case, my studio is already using Stereo 1 and 2 for audio output to monitor, so I had configured the channels to 3, 4, 5 and 6 in order. These outputs are dependent upon the user's audio interface's layout and the number and order of speakers.

What makes this set up so convenient is, not only does it allow for panning along an array of speakers, but it also allows for modular set up of routing inside of the DAW.

For example, there are three simple ways to route audio to a reverb when looking at Figure 5.11, but each way functions EXTREMELY different. In both cases, the first step would be to create a return track and add a reverb effect like normal. However, the second step is where things get interesting. A single parameter could either be mapped to the "Multi Out Test" primary channel's return knob and the sound could passed directly to the reverb bus. In this case we might configure the reverb bus to also have an instance of Surround Sounder on it set with a low spread. The reverb itself would become its own sound object and could be panned along the speaker array through its own grouping of speaker sends just like Ableton's Surround Panner. Secondly, a single physical MIDI knob could be mapped to the all the send knobs on all four speaker channels, or even just one send, and sent directly to a reverb send. This send could then be configured either to go out to all channels, or only a couple at a time creating a widening or jumping effect.

The third option is slightly different and doesn't require the use of sends. In this case we would create a reverb preset and clone it onto each of the four speaker channels. A physical knob could then be mapped as a MIDI parameter to Dry/Wet or Mix functions. When the audio would pass through each channel, the reverb would be triggered and remain as a point in space, providing the most fluidly simulated spatialized effect. While all these different routing concepts are essentially a small variation of a very simple thing, their effects would be drastically different.

5.2.3 Starting to connect the Dots.

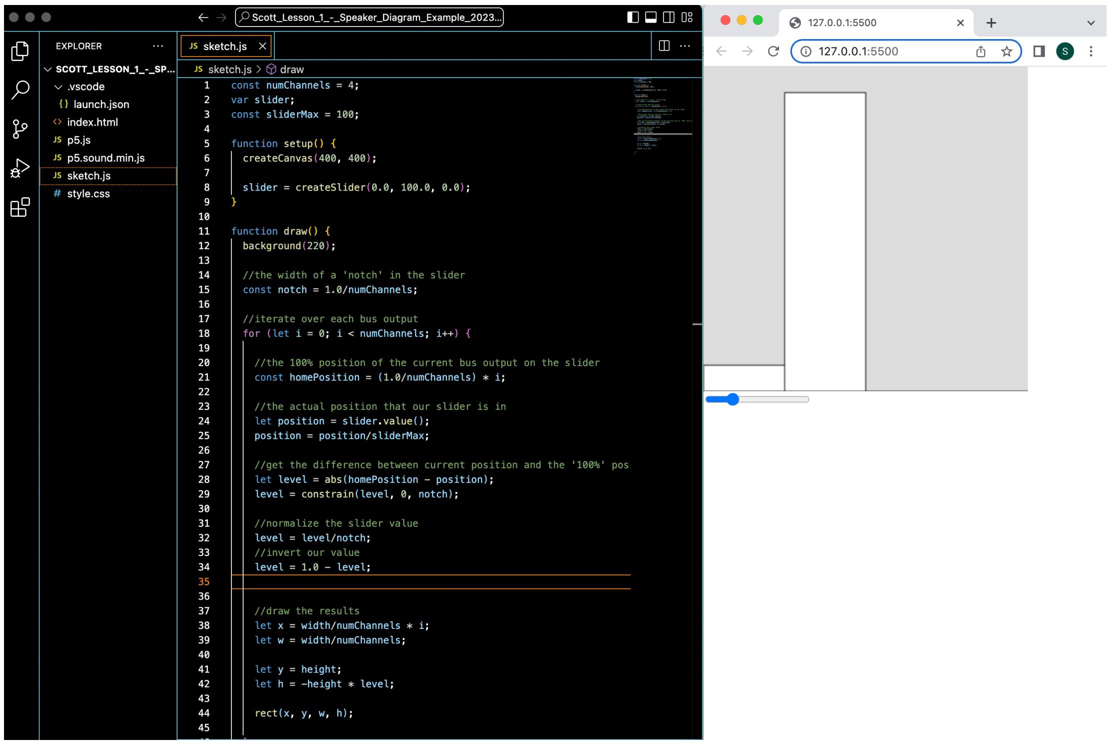
Coming up with the behavior for the volume automation was the next task. In this instance I turned again to my Mentor. Since the position inside of an application would be controlled by a slider, and volume portrayed as vertical amount we started from there. The task was to build a slider that would adjust the height of rectangles based on the corresponding distance of the slider pointer from a home position. The rectangles would represent the amount of gain applied to the sample buffer inside of the audio plugin.

In this version of the code, we created a canvas and a slider and then defined what a notch would be. If a slider is represented as one, and there are four speakers, a notch would be equal to 0.25. The notch represents the equal space between each of the speakers in the array, represented on a slider. We wrote a for loop where we would define a variable as zero and that it will remain less than the number of channels. In here we define the behavior of the slider and the rectangles.

First, we set the home positions of the rectangles along a value of zero to one. This is so that we can define them as a point along the slider. In this case with four rectangles, the home positions are 0.25, 0.5, 0.75 and 1. Then we define a position which is our slider value and normalize it between a range of zero and 1 which can correspond to our home positions.

Defining the level is where things got a little bit tricky. First, we define our level as our home position subtracting our sliders position. However, for the effect to happen in both directions we have to do this in an absolute value otherwise we could get both positive and negative values. In this case, the only time level is ever equal to zero is when home position is equal to position. Then, we constrain our level variable between zero and notch, in the case of four channels that is 0.25. If the position is further away than 0.25 our level will not go greater than 0.25. This piece of code isn't entirely necessary, but it is a safety check in case we start getting some wild values for level. For instance, if we start introducing a spread or bleed effect.

Then we normalize our level back to a value of one, by multiplying it by the notch value. The final piece to our level variable is inverting the normalized version. In this case we take our normalized value of 1 and subtract level from it. This means that level is always zero unless we move into our notch, in which case it will be equal to one when the home position is equal to the value of the slider position. Below this we draw some rectangles using the level to control their height. In an audio processor, we would multiply the audio by the level variable. But this is where things started to get a little tricky.



```javascript
const numChannels = 4;
var slider;
const sliderMax = 100;

function setup() {
  createCanvas(400, 400);

  slider = createSlider(0.0, 100.0, 0.0);
}

function draw() {
  background(220);

  //the width of a 'notch' in the slider
  const notch = 1.0/numChannels;

  //iterate over each bus output
  for (let i = 0; i < numChannels; i++) {

    //the 100% position of the current bus output on the slider
    const homePosition = (1.0/numChannels) * i;

    //the actual position that our slider is in
    let position = slider.value();
    position = position/sliderMax;

    //get the difference between current position and the '100%' pos
    let level = abs(homePosition - position);
    level = constrain(level, 0, notch);

    //normalize the slider value
    level = level/notch;
    //invert our value
    level = 1.0 - level;


    //draw the results
    let x = width/numChannels * i;
    let w = width/numChannels;

    let y = height;
    let h = -height * level;

    rect(x, y, w, h);
```

Figure 5.13 Simple Layout for Four Channel Output

A computer can only process so much data at a time, this includes audio. Audio when passed through an application or plugin, which is basically just lines of code, isn't handled analog. It's handled digitally. This means that audio is broken down into bits, like the black squares on a piece of graph paper. Most audio producers and audiophiles are familiar with the concept of a sample rate. This is the number of samples captured per second when audio is being processed, whether coming through an interface or being affected inside of the DAW.

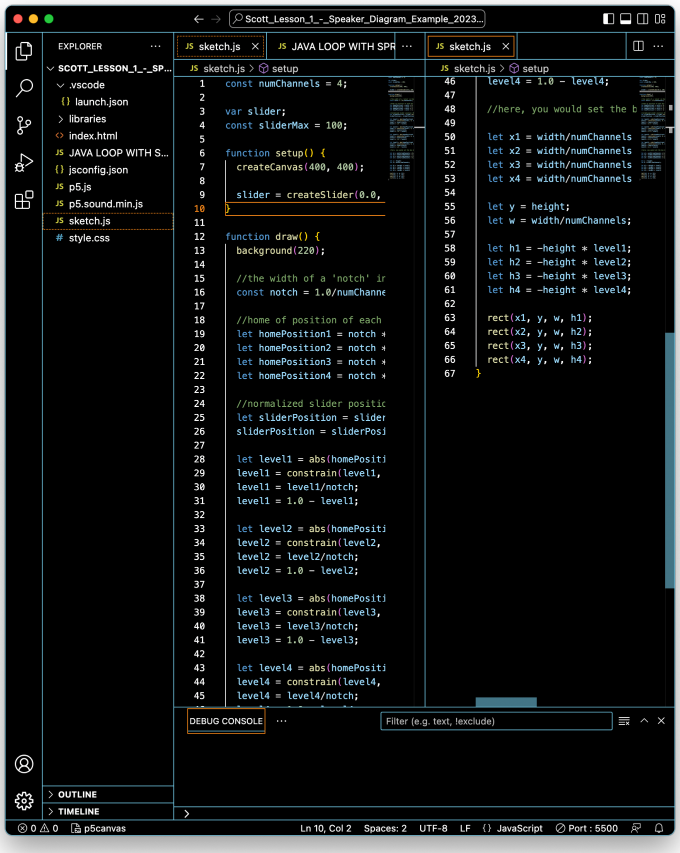
Sample size is determined by word size, which is called bit depth in Ableton live and can be either 16, 24, or 32 bits. This means that each sample is comprised of a corresponding number of bits. For example, if the DAW was set for a sample rate of 44.1khz with a word size of 16, we could say that every second of audio is broken into 44,100 * 16 = 705,600 bits. That's a lot of data to be processing every millisecond.

The way I visualize it is as a plate of Onigiri. Think of the bits as grains of rice, they're too small to eat by themselves. So those grains of rice are put together into Onigiri, a sample, and we serve the Onigiri to the processor on a plate, the buffer size. The buffer size is the number of samples that we provide to the processor at a time. The lower the buffer size, the fewer samples get passed through at a time, meaning that there is more processing required per second. The larger the buffer size, the less buffers processed per second. However, because it takes more time for an audio processor to physically process each byte when the buffer size is larger, it introduces a delay.

Figure 5.14 Image of Onigiri

So, what does this have to do with the Surround Sounder? A JUCE application isn't just painting a circle on the screen and repainting the position of a sound, like that circle whenever it receives the mouse position. The application is performing thousands of tasks per second, processing hundreds of thousands of bits. Understanding the architecture of how this happens would be crucial if I was going to build the Surround Sounder.

If the audio processor was going to be creating busses in a for loop repeatedly within JUCE's process block, thousands of times a second. We would probably start having issues. So, we adjusted the math, to allow for the static creation of the bus buffers. In this case we removed the for loop that controlled the levels as well as the creation of the rectangles in cases we needed call this function without looping.



```
const numChannels = 4;

var slider;
const sliderMax = 100;

function setup() {
  createCanvas(400, 400);

  slider = createSlider(0.0,
}

function draw() {
  background(220);

  //the width of a 'notch' ir
  const notch = 1.0/numChanne

  //home of position of each
  let homePosition1 = notch *
  let homePosition2 = notch *
  let homePosition3 = notch *
  let homePosition4 = notch *

  //normalized slider positic
  let sliderPosition = slider
  sliderPosition = sliderPosi

  let level1 = abs(homePositi
  level1 = constrain(level1,
  level1 = level1/notch;
  level1 = 1.0 - level1;

  let level2 = abs(homePositi
  level2 = constrain(level2,
  level2 = level2/notch;
  level2 = 1.0 - level2;

  let level3 = abs(homePositi
  level3 = constrain(level3,
  level3 = level3/notch;
  level3 = 1.0 - level3;

  let level4 = abs(homePositi
  level4 = constrain(level4,
  level4 = level4/notch;
```

```
  level4 = 1.0 - level4;

  //here, you would set the b

  let x1 = width/numChannels
  let x2 = width/numChannels
  let x3 = width/numChannels
  let x4 = width/numChannels

  let y = height;
  let w = width/numChannels;

  let h1 = -height * level1;
  let h2 = -height * level2;
  let h3 = -height * level3;
  let h4 = -height * level4;

  rect(x1, y, w, h1);
  rect(x2, y, w, h2);
  rect(x3, y, w, h3);
  rect(x4, y, w, h4);
}
```

Figure 5.15 Code Without for Loop

At this point I was able to begin implementing the code inside of Xcode. Some adjustments had to be made, calling variables required different language in C++ and there were different namespace specific commands that worked better with the JUCE framework. But I got it to compile. I'm pretty sure I scared my neighbors when I celebrated seeing the audio pan across the four output channels.

Finally, Surround Sounder might actually work.

5.3 Drink the JUCE

5.3.1 When the GUI gets tough the tough get GUI

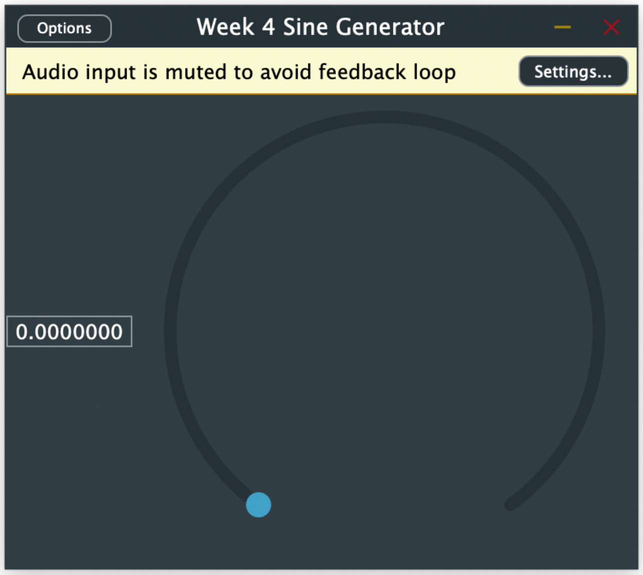


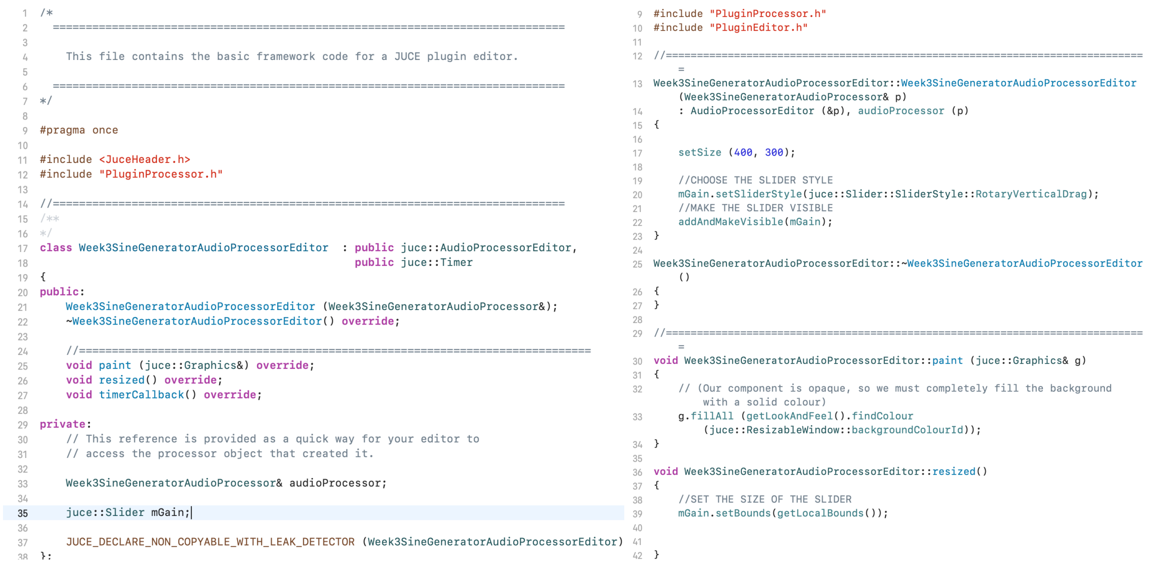
Figure 5.16 Let's just say I was getting real sick of staring at this knob.

As any person who has spent time learning to code, different languages have different applications. In the same way the p5js is simple, it is also limited in its uses. On the other hand, in the way the C++ is powerful and robust, it's complex and can be confusing to learn, especially to a new programmer… like myself. In java, adding a slider is a simple as slider = createslider(min, max, value, step); but to add a visible slider to a JUCE application there are a handful of different steps.

First, a programmer must understand the difference between the plugin editor and plugin processor. The plugin processor is self-explanatory, it fulfills the processes necessary for the application to function. The editor, on the other hand requires a little bit more looking at to realize that its purpose is to primarily control the aspects of the plugins GUI (graphical user interface). So, let's go over the steps to add a rotary slider to JUCE that was then used to control a simple panning algorithm.

Before we get started, if I was building this application in C++ from scratch the steps would be very different. Not only would I have to build the slider from scratch, but I would also have to define all the steps to paint a canvas and the form that it should take. Unlike p5js that assumes that we are drawing something, C++ assumes nothing. You give it some values and it gives you the behavior. The JUCE framework contains many already defined classes and functions that allow objects such as a rotary slider to be added relatively simply.

First things first, a slider provides a variable, a numerical output. So, we must declare the object, just as we would in p5js. A couple things to note, variables inside of C++ classes naturally default to private, so unless you are specifically providing them for the compiler to view publicly, they will be private. Also, when we are looking at our headers and cpp files inside of JUCE, we aren't just looking to add lines of code, we need it to be defined properly within the classes. In this case, the AudioProcessorEditor. In here we define public functions, which are viewable and accessible outside of the AudioProcessorEditor class, and below we define our private variables and objects which are specific to the class itself. These private variables are only viewable and storable within the class itself. So, in our case, we want to declare our slider as a private object in our plugin editor since we don't need our processor to know how to draw it. In this case we call it mGain.



```cpp
/*
  ==============================================================================

    This file contains the basic framework code for a JUCE plugin editor.

  ==============================================================================
*/

#pragma once

#include <JuceHeader.h>
#include "PluginProcessor.h"

//==============================================================================
/**
*/
class Week3SineGeneratorAudioProcessorEditor  : public juce::AudioProcessorEditor,
                                                public juce::Timer
{
public:
    Week3SineGeneratorAudioProcessorEditor (Week3SineGeneratorAudioProcessor&);
    ~Week3SineGeneratorAudioProcessorEditor() override;

    //==============================================================================
    void paint (juce::Graphics&) override;
    void resized() override;
    void timerCallback() override;

private:
    // This reference is provided as a quick way for your editor to
    // access the processor object that created it.

    Week3SineGeneratorAudioProcessor& audioProcessor;

    juce::Slider mGain;

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (Week3SineGeneratorAudioProcessorEditor)
};
```

```cpp
#include "PluginProcessor.h"
#include "PluginEditor.h"

//==============================================================================
    =
Week3SineGeneratorAudioProcessorEditor::Week3SineGeneratorAudioProcessorEditor
    (Week3SineGeneratorAudioProcessor& p)
    : AudioProcessorEditor (&p), audioProcessor (p)
{

    setSize (400, 300);

    //CHOOSE THE SLIDER STYLE
    mGain.setSliderStyle(juce::Slider::SliderStyle::RotaryVerticalDrag);
    //MAKE THE SLIDER VISIBLE
    addAndMakeVisible(mGain);
}

Week3SineGeneratorAudioProcessorEditor::~Week3SineGeneratorAudioProcessorEditor
    ()
{
}

//==============================================================================
    =
void Week3SineGeneratorAudioProcessorEditor::paint (juce::Graphics& g)
{
    // (Our component is opaque, so we must completely fill the background
        with a solid colour)
    g.fillAll (getLookAndFeel().findColour
        (juce::ResizableWindow::backgroundColourId));
}

void Week3SineGeneratorAudioProcessorEditor::resized()
{
    //SET THE SIZE OF THE SLIDER
    mGain.setBounds(getLocalBounds());


}
```

Figure 5.17 AudioProcessorEditor Class

The declaration itself doesn't do anything except let us know that here we are able to defining a slider based on the JUCE framework's native Slider Class. To add that we must do a little bit of extra code. First inside of the AudioProcessorEditor constructor class inside of the constructor we must choose our slider style, in this case it is a rotary slider. And then we must addAndMakeVisible, which will add it to our canvas. Finally, in the resize function we must set the actual size, and in this case, we're setting it to the size of the canvas.

Now, this might seem simple enough. C++ is specific, just like C and requires very precise instructions to the compiler to the code to compile. Of course, you must set the variable, set the design, make it visible and set the size. It shouldn't just pop up like it does in p5js, that's the point. I'm trying to illustrate here that simple functions such as the one in Figure 5.12 don't just happen to work correctly inside of C++. There is an applicable number of extra steps to get it working.

Continuing along the road of the GUI let's look at the application in an early form. If something like this was built in a language like p5js, it would take only a few minutes and some simple drawing. However, for the GUI in this JUCE application to compile the draw functions as well as the processing functions it again takes quite a few extra steps. Getting to this point as a new coder was particularly challenging to me as I was learning basic C++ concepts at the same time as the JUCE framework.

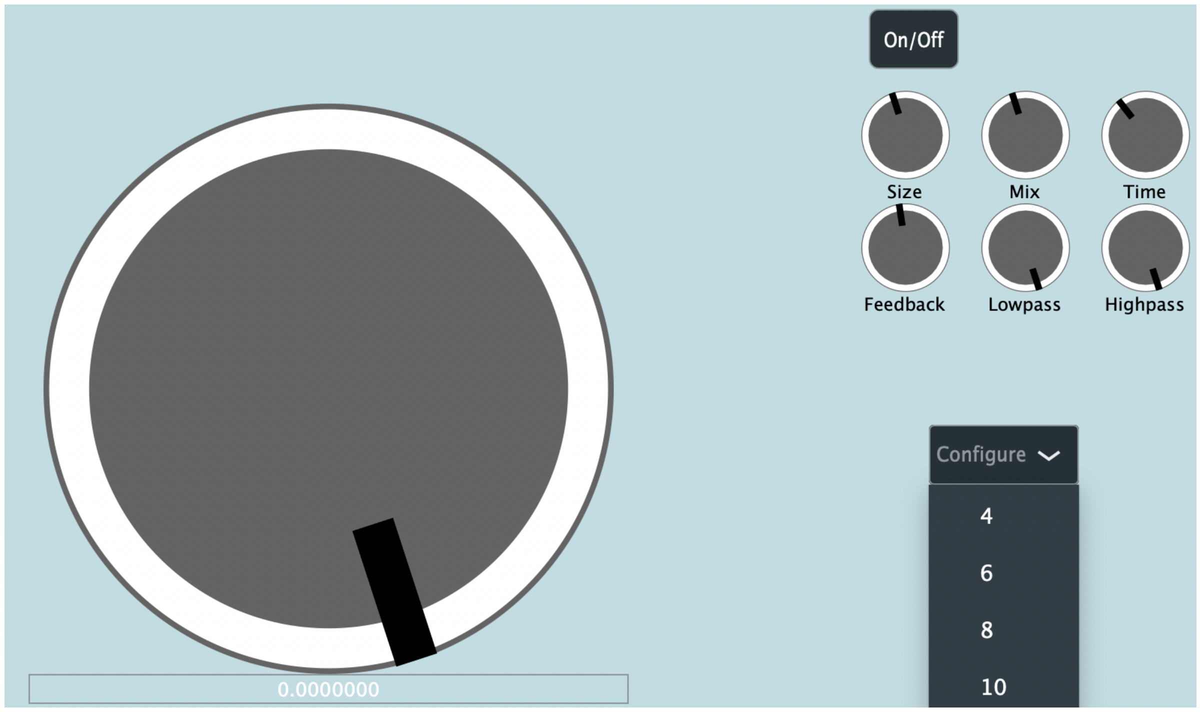
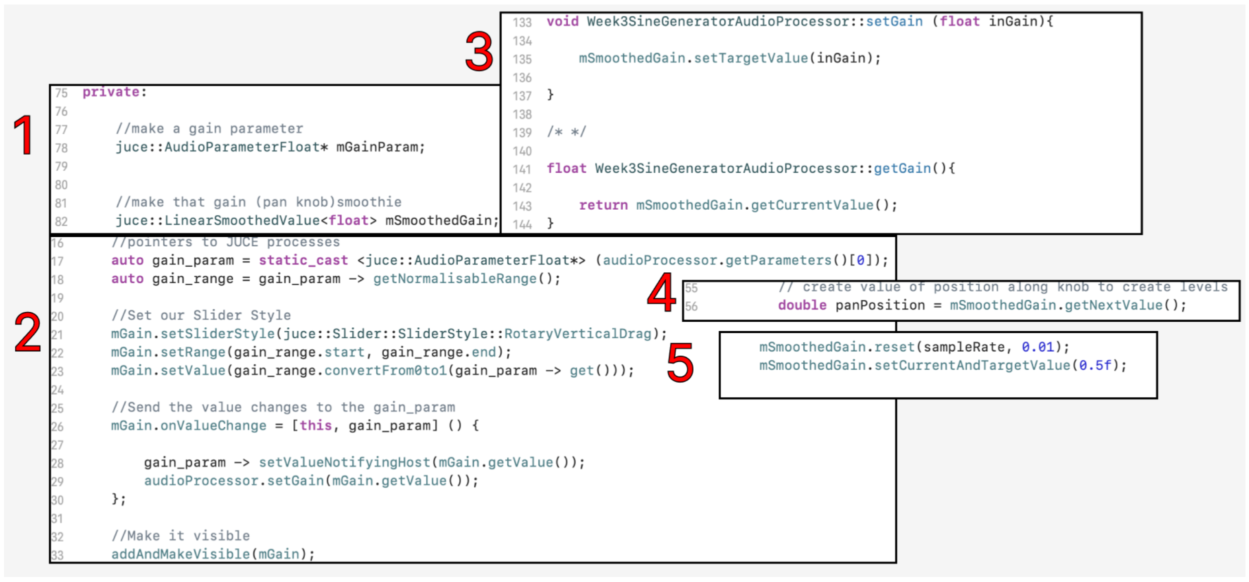


Figure 5.18 Early Stages of the Surround Sounder GUI

As you can see, there's a strong difference between Figure 5.4 and Figure 5.16. Not just in the complexity, but also in changes to the layout. Some of this is due to the actual practicality of using slider parameters instead of buttons or combo-boxes as well as the fact that much of it was because I was learning new aspects of the JUCE code. There's no such thing as coding inside a vacuum when building an application in C++, everything must be precise and every class you create will have influence over another.

After learning the basic set up of parameters as individual objects, it was easy to move knobs and buttons around the screen. You add and make your parameter visible in the editor. Then using a pointer, you would create the corresponding parameter variable inside of your editor where you send the value to your audio processor. The parameter does some function, and you apply this to your audio block, or your virtual synthesis and send it to your outputs. Easy enough.

Let's look at a practical example from an early version of my code in Figure 5.17, with the mGain slider having already been declared in in Figure 5. 16.. In box 1, inside the Audio Processor Class we declare new object mSmoothedGain. In box 2, we create our slider, setting its range to be the same as a gain parameter to be defined by, gain_range. Then we apply smoothing function to the mGain knob directly in lambda function on line 26. In box 3, we create two void functions that utilize JUCE framework smoothed value operation. We smooth the value of this slider because otherwise we get clicking artifacts. The functions first get the gain from the value of the mGain knob, and then we provide a function that sets the new smoothed gain value. The setGain function is applied directly back to mGain in that lambda in box 2. In box 4, we set the value of mSmoothedGain to our panPosition so that it can be used in a variation of our p5js code above. Then finally in box 5, we set the amount of smoothing for our smoothed parameter.



```
private:

    //make a gain parameter
    juce::AudioParameterFloat* mGainParam;


    //make that gain (pan knob)smoothie
    juce::LinearSmoothedValue<float> mSmoothedGain;
```

```
    //pointers to JUCE processes
    auto gain_param = static_cast <juce::AudioParameterFloat*> (audioProcessor.getParameters()[0]);
    auto gain_range = gain_param -> getNormalisableRange();

    //Set our Slider Style
    mGain.setSliderStyle(juce::Slider::SliderStyle::RotaryVerticalDrag);
    mGain.setRange(gain_range.start, gain_range.end);
    mGain.setValue(gain_range.convertFrom0to1(gain_param -> get()));

    //Send the value changes to the gain_param
    mGain.onValueChange = [this, gain_param] () {

        gain_param -> setValueNotifyingHost(mGain.getValue());
        audioProcessor.setGain(mGain.getValue());
    };

    //Make it visible
    addAndMakeVisible(mGain);
```

```
void Week3SineGeneratorAudioProcessor::setGain (float inGain){

    mSmoothedGain.setTargetValue(inGain);

}

/* */

float Week3SineGeneratorAudioProcessor::getGain(){

    return mSmoothedGain.getCurrentValue();
}
```

```
        // create value of position along knob to create levels
        double panPosition = mSmoothedGain.getNextValue();
```

```
    mSmoothedGain.reset(sampleRate, 0.01);
    mSmoothedGain.setCurrentAndTargetValue(0.5f);
```

Figure 5.19 Example of a Smoothing Algorithm

This is a simple task that a user normally wouldn’t consider when using an app. A knob is a knob, what average music producer would know or care about whether a parameter was smoothed unless he could hear the difference. If this was all the app was doing this implementation would be fine if the plugin remained exceedingly simple. The code would be relatively normal to sift through if there were only a couple parameters. However, this isn’t normally the case, Audio Plugins are normally robust with many parameters and interactable features. So, in class we began to cover the concept of data structures to store data related to our parameters, we can refer to them as container items in this paper for our purposes.

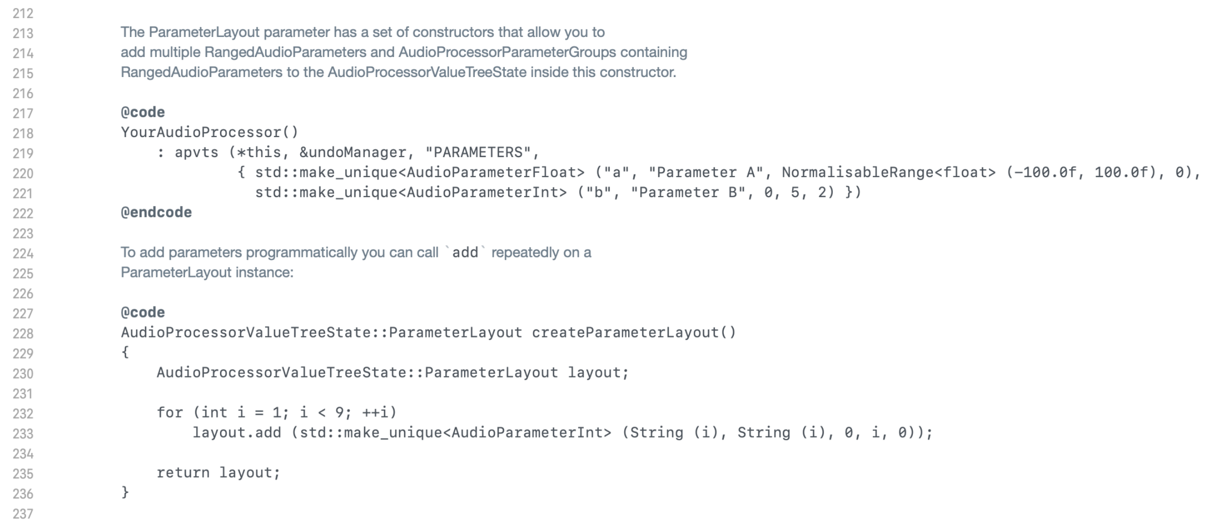
Creating new classes in C++ generally requires creating new headers and cpp files, which is also good organizational practice. So, following at the direction of my professor, Jacob Penn, I created a set of container items.

5.3.2 The Container Store

In this chapter I’ll cover how container items allow for more streamlined code, but also create their own set of issues for a new coder. In Figure 5.16 you notice a group of knobs in the top right corner. The knobs could have been created individually and positioned by a resize function. But as we’ve seen, creating, and setting functions for parameters can lead to long and messy lines of code. This can lead to difficulty reading as well as higher CPU usage.

JUCE provides us with a data storage structure called a Value Tree. A Value Tree in its simplest form is a container to store parameter data. You define a list of parameters either individually or as a list and use the Value Tree to grab the data from the parameters and store it as a tree. In my case, the six knobs in the top right side of Figure 5.16 are the parameters I chose to store in a value tree. There are multiple ways to store data into a value tree.

Using a for loop we can iterate over the parameter data that has been created, storing it in the Value Tree for our plugin to find. A value tree for the audio processor requires the name of the parameter, the name of the parameter for its label, it's minimum value and its maximum value as well as it's starting value.



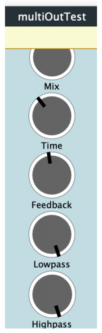
```
    The ParameterLayout parameter has a set of constructors that allow you to
    add multiple RangedAudioParameters and AudioProcessorParameterGroups containing
    RangedAudioParameters to the AudioProcessorValueTreeState inside this constructor.

    @code
    YourAudioProcessor()
        : apvts (*this, &undoManager, "PARAMETERS",
                 { std::make_unique<AudioParameterFloat> ("a", "Parameter A", NormalisableRange<float> (-100.0f, 100.0f), 0),
                   std::make_unique<AudioParameterInt> ("b", "Parameter B", 0, 5, 2) })
    @endcode

    To add parameters programmatically you can call `add` repeatedly on a
    ParameterLayout instance:

    @code
    AudioProcessorValueTreeState::ParameterLayout createParameterLayout()
    {
        AudioProcessorValueTreeState::ParameterLayout layout;

        for (int i = 1; i < 9; ++i)
            layout.add (std::make_unique<AudioParameterInt> (String (i), String (i), 0, i, 0));

        return layout;
    }
```

Figure 5.20 How JUCE Defines a Value Tree

The next item we will talk about is the Slider Container. Using Slider Container allows you to utilize the Value Tree state to create the physical parameters in your editor. Inside the Slider Container you will set the sizes of the knobs and their styles and where the labels will sit. If your plugin has a separate look and feel defined you can apply this to your Slider Container, but we will cover this later.

So, at this point, we have stored some parameters into a Value Tree, created the parameter's sliders accordingly in the Slider Container. Great, we should now have knobs added in a nice box, right? Well not exactly. The containers that we just created using a parameter Value Tree, the values of parameters we want to store, and the Slider Container, painting the sliders we want to use in relationship to those parameters, are creating virtual items. This is C++ though, so unless we tell these objects where to go, they're just going to follow the directions of the code. In my case the sliders themselves are painted vertically starting from the top down. So, they'll just populate the full length of the canvas directly down from the top.

We need to create a physical container to house these parameters, that are made visible by this slider container. There are a few options that C++ provides us with to do this, and JUCE also gives us a couple options as well. The most dynamic one that I found served my purposes was the Flexbox.



```
/**/
void SliderContainer::resized()
{

    auto bounds = getLocalBounds();
    int knobSize = mLookAndFeel.getKnobSize();

    mSlider.setBounds(bounds.removeFromTop(getHeight()*.8f));
    mSlider.setTextBoxStyle(juce::Slider::TextBoxBelow, true, knobSize, mSlider.getHeight() * 0.03f);

    mLabel.setBounds(bounds);
    mLabel.setFont(Font(14.0f));
    mLabel.setColour(Label::textColourId, Colours::black);


}
```

Figure 5.21 Knobs Painted from the Top Down

A Flexbox is a dynamic visual container item that allows for the storage of Flex Items. More simply, put your knobs in a Flexbox and then set its size and position and layout options and the items will adjust themselves to fit in it accordingly. A Flexbox is handled pretty much the same way a slider is. You create a Flexbox object, declaring it as a private variable inside of the Plugin Editor header. Then you can set the parameters of your flexbox. My settings for the Flexbox that I used for the knobs in Figure 5.16 are below.

```
    auto bounds = getLocalBounds();
    auto flexbox_width = bounds.getWidth() * 0.3;
    auto flexbox_bounds = bounds.removeFromRight(flexbox_width).withY(-100);


    mFlexbox.flexDirection = FlexBox::Direction::row;
    mFlexbox.flexWrap = juce::FlexBox::Wrap::wrap;
    mFlexbox.justifyContent = juce::FlexBox::JustifyContent::center;
    mFlexbox.alignContent = FlexBox::AlignContent::center;
    Array<FlexItem> itemArray;

    for (auto* i : mSliderContainers)
    {
        mFlexbox.items.add (juce::FlexItem (*i).withMinHeight (75.0f).withMinWidth (75.0f).withFlex (1));

    juce::FlexBox fb;
                fb.flexDirection = juce::FlexBox::Direction::column;

        fb.items.add (juce::FlexItem (mFlexbox).withFlex (2.5));

                fb.performLayout (flexbox_bounds.toFloat());
            }
```

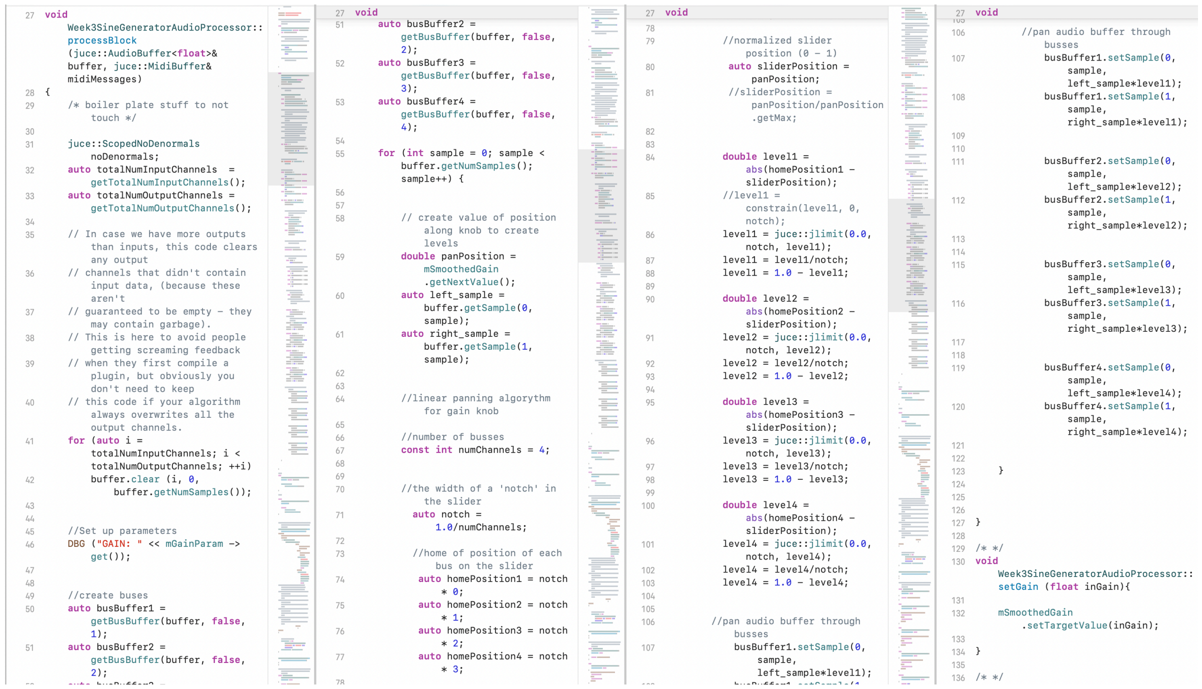
Figure 5.22 Flexbox Settings

I'm not a very visual person, so this took me a while to figure out. First, we declare the bounds as the size of the canvas. Then we set the width of the flexbox, which will be thirty percent of the size of our canvas. Then we set the position of the flexbox, using the right side and the width of the flexbox as the X position, and then pulling it down one hundred pixels from the top. Afterwards, we tell the flexbox to position the items in a row, have them wrap if they aren't able to all fit on the same line and then center the items. Then we make an Array to store the items and iterate over them and add them into the Flexbox using a for loop. This is what I would think of if you asked me what a container item is, but inside of JUCE there are many ways of storing data structures.

5.3.3 Panning

So, as we’ve seen, moving audio around an array of outputs isn’t quite as simple as moving an ellipse along the circumference of a circle in p5js. However, if I considered the behavior of the circle to be the like the behavior of the audio buffer then I started seeing some of what I was looking for. There were some hiccups when converting p5js code into C++ and those took a little while to move past.

First, the syntax is slightly different. For example, the command let in p5js is most closely related to the declaration of auto in C++. Additionally, there isn’t a basic constrain function in C++, it either needs to be defined or pulled from a separate namespace. In our case we do have the JUCE namespace, which offers us the jlimit function. It operates in an almost identical way to p5js’s constrain. Finally, our sliders require a different kind of declaration which we covered, so they need to be interfaced into the code to work.



```
void
    Week3SineGeneratorAudioProcessor::
    processBlock
    (juce::AudioBuffer<float>&
    buffer, juce::MidiBuffer&
    midiMessages)
{
    /* boiler plate stuff to not
        touch */

    juce::ScopedNoDenormals
        noDenormals;
    auto totalNumInputChannels  =
        getTotalNumInputChannels();
    auto totalNumOutputChannels =
        getTotalNumOutputChannels();

    // In case we have more outputs
        than inputs, this code clears
        any output
    // channels that didn't contain
        input data, (because these
        aren't
    // guaranteed to be empty - they
        may contain garbage).
    // This is here to avoid people
        getting screaming feedback
    // when they first compile a
        plugin, but obviously you
        don't need to keep
    // this code if your algorithm
        always overwrites all the
        output channels.
    for (auto i =
        totalNumInputChannels; i <
        totalNumOutputChannels; ++i)
        buffer.clear (i, 0,
            buffer.getNumSamples());


    //Set up parameters
    DBG ("GAIN: " << mGainParam ->
        get());


    //create buses
    auto busBuffer1 =
        getBusBuffer(buffer, false,
        1);
    auto busBuffer2 =
        getBusBuffer(buffer, false,
        2);
```

```
void
    auto busBuffer2 =
        getBusBuffer(buffer, false,
        2);
    auto busBuffer3 =
        getBusBuffer(buffer, false,
        3);
    auto busBuffer4 =
        getBusBuffer(buffer, false,
        4);

    for (int sample = 0; sample <
        buffer.getNumSamples();
        sample++) {


        // create value of position
            along knob to create
            levels
        double panPosition =
            mSmoothedGain
            .getNextValue();
        auto left_sample =
            buffer.getSample(0,
            sample);
        auto right_sample =
            buffer.getSample(1,
            sample);


        //linear panning algorythm
            for gain knob

        //number of busses
        const int numChannels = 4;


        //the width of a 'notch' in
            the slider
          auto notch =
              1.0/numChannels;

          //home of position of each
              bus on the slider
           auto homePosition1 = notch
               * 0;
           auto homePosition2 = notch
               * 1;
           auto homePosition3 = notch
               * 2;
           auto homePosition4 = notch
               * 3;
```

```
void

          //normalized slider
              position (0 - 1)
           auto sliderPosition =
               panPosition;
           //sliderPosition =
               panPosition/panPosition
               .getMax;


          double level1 =
              abs(homePosition1 -
              sliderPosition);
          //level1 =
              constrain(level1, 0,
              notch);
          level1 = juce::jlimit(0.0,
              notch, level1);
          level1 = level1/notch;
          level1 = 1.0 - level1;

          double level2 =
              abs(homePosition2 -
              sliderPosition);
          level2 = juce::jlimit(0.0,
              notch, level2);
          level2 = level2/notch;
          level2 = 1.0 - level2;

          double level3 =
              abs(homePosition3 -
              sliderPosition);
          level3 = juce::jlimit(0.0,
              notch, level3);
          level3 = level3/notch;
          level3 = 1.0 - level3;

          double level4 =
              abs(homePosition4 -
              sliderPosition);
          level4 = juce::jlimit(0.0,
              notch, level4);
          level4 = level4/notch;
          level4 = 1.0 - level4;


        //pan audio buffer through
            busses
            busBuffer1.setSample(0,
                sample,
                left_sample*level1);
```

```
void
        //pan audio buffer through
            busses
            busBuffer1.setSample(0,
                sample,
                left_sample*level1);
            busBuffer1.setSample(1,
                sample,
                right_sample*level1);


            busBuffer2.setSample(0,
                sample,
                left_sample*level2);
            busBuffer2.setSample(1,
                sample,
                right_sample*level2);


            busBuffer3.setSample(0,
                sample,
                left_sample*level3);
            busBuffer3.setSample(1,
                sample,
                right_sample*level3);


            busBuffer4.setSample(0,
                sample,
                left_sample*level4);
            busBuffer4.setSample(1,
                sample,
                right_sample*level4);


    }



}

/* */
void
    Week3SineGeneratorAudioProcessor::
    setGain (float inGain){

    mSmoothedGain
        .setTargetValue(inGain);

}

/* */
```

Figure 5.23 Implementation of the Panning Algorithm

Figure 5.20 shows the first working version of the audio panning algorithm in JUCE’s process block. It functions very similarly to the code in psj5 that didn’t use the for loop. First, we create us for loop which clears our buffer whenever it is fed a new buffer. Then we create a set of bus buffers. We pass in the audio as the audio buffer, since we’ve already cleared the buffer we tell the bus that it’s not required to clear and then we assign each bus buffer to its corresponding bus out of the four all inside the getBusBuffer(). Next, we create our panPosition variable using the value of our gain knob but using its smoothed value mSmoothedGain before defining our left and right samples.

Next, we define the number of channels we want as four, since that’s all we’ve prepared for in the constructor and start the application of the p5js code that did not use a for loop. We create our notch value and use it to create individual levels for each of the busses, you can see here the jlimit function is used instead of constraining as well as the use of a double variable to store integers with long strings of numbers.

After this we set the samples to our bus buffers. First, we pass in the value of zero or one to declare whether it is the left side or the right side. Then we pass in our sample variable. After this we apply the level changes to our left_sample and right_sample to adjust the gain as our knob passed along the notches.

When I first implemented this code, I wasn't sure which of those three arguments to apply the level to so I figured it must be sample. What was fascinating about this was that applying gain directly to the sample caused a distorted destruction-like effect on the audio buffer as it was passed through the gain function. But after swapping the level value to the respective sample values instead of the samples themselves it functioned as a device that panned linearly over the course of the four busses.

Unfortunately, though, this code was incomplete in a multitude of ways. First, a constant of four output busses isn't modular, adjustable, or even more output busses than Ableton Live's Surround Panner plugin. Also, this code was all done directly inside of the process block of the audio processor, so if it was developed into more complex directions, it might get messy or too long. The code didn't use for loops, so the levels were all being created manually with each channel, instead of dynamically so that it could be scaled. The code also didn't have the variability to store parameters within a value tree. And most importantly, the rotary sliders in JUCE don't wrap. So, when the slider reached its highest value, the panning stopped and could only be turned around instead of continuing past one back to zero.

The fact that it worked wasn't lost on me. I was exuberant. However, there was a long way to go if this application was going to go from a single rotary slider to a fully functioning surround panning application with integrated delay.

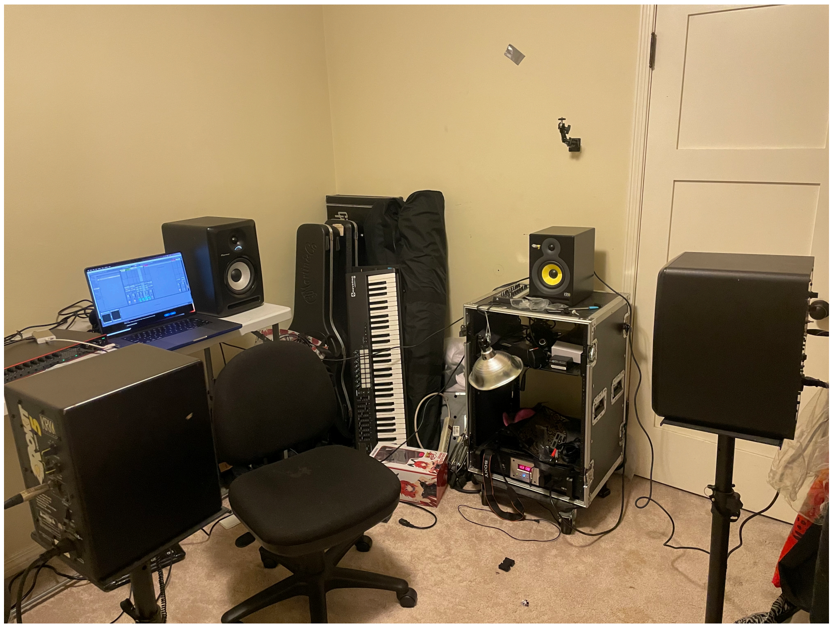


Figure 5.24 My Surround "Testing Environment"

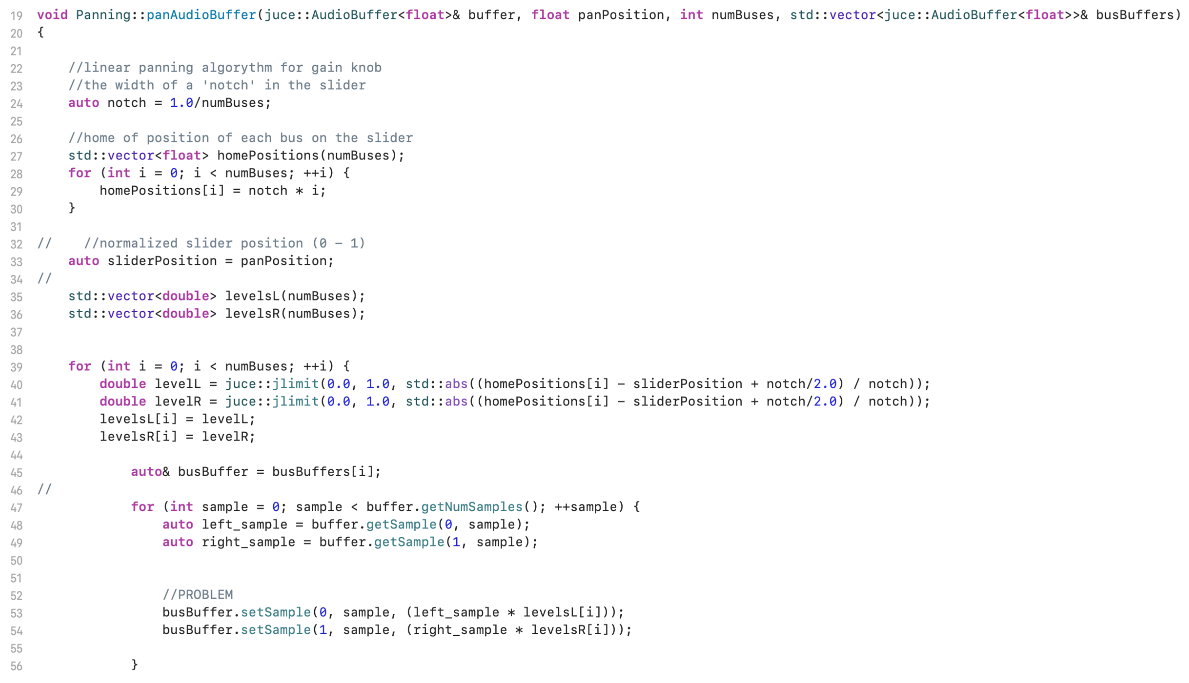
5.4 Refining The Build

5.4.1 Finding a Starting Point

The code was operating, so it was time to break it. At this point I knew I had to create a new class to store my panning algorithm. So, I made new header and cpp files and began to chip away at turning my code into functions that could be called. This proved significantly more challenging than any other step of this process, apart from the initial set up of JUCE and understanding the basic architecture of a plugin.

The other thing about C++ and JUCE is that while your code may compile, it might not actually work in the way you intended. This will pop what is known as a jassert inside of the JUCE framework and in most cases, crash your plugin when running it in debug mode. These jasserts are built in safety mechanisms to keep you from undefined behavior. Additionally, JUCE allows you to check for undefined behavior with the jassert command.

So, the first task I set out to do was create the levels and assign them to the bus buffers, however instead of handling this manually it was important that it was possible to do within a loop. This would allow me to change the number of busses via either a combo box or slider as well as just clean up the messy code I had put together in my Audio Processor. So, I went back to our first version of the p5js code and started from there.



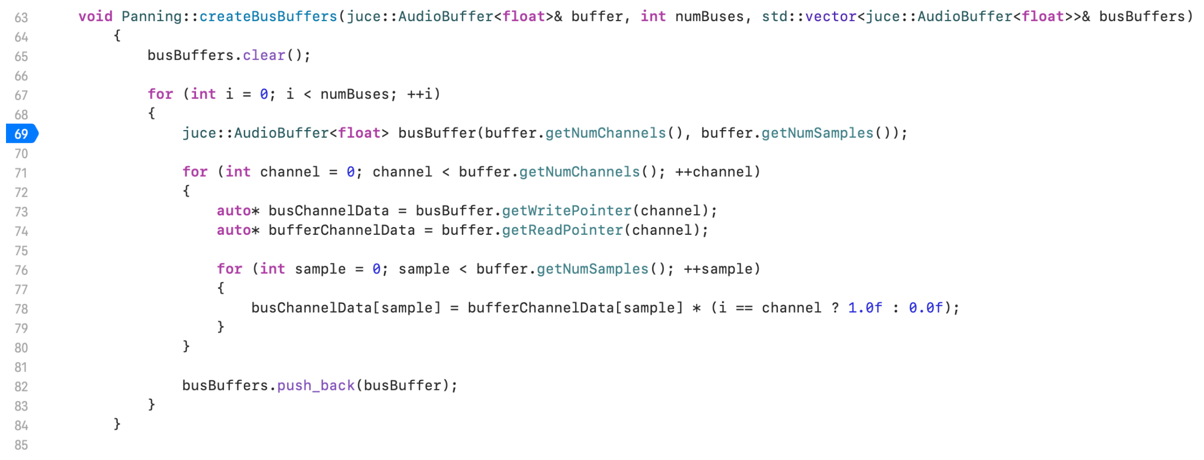
```
void Panning::panAudioBuffer(juce::AudioBuffer<float>& buffer, float panPosition, int numBuses, std::vector<juce::AudioBuffer<float>>& busBuffers)
{

    //linear panning algorythm for gain knob
    //the width of a 'notch' in the slider
    auto notch = 1.0/numBuses;

    //home of position of each bus on the slider
    std::vector<float> homePositions(numBuses);
    for (int i = 0; i < numBuses; ++i) {
        homePositions[i] = notch * i;
    }

//    //normalized slider position (0 - 1)
    auto sliderPosition = panPosition;
//
    std::vector<double> levelsL(numBuses);
    std::vector<double> levelsR(numBuses);


    for (int i = 0; i < numBuses; ++i) {
        double levelL = juce::jlimit(0.0, 1.0, std::abs((homePositions[i] - sliderPosition + notch/2.0) / notch));
        double levelR = juce::jlimit(0.0, 1.0, std::abs((homePositions[i] - sliderPosition + notch/2.0) / notch));
        levelsL[i] = levelL;
        levelsR[i] = levelR;

            auto& busBuffer = busBuffers[i];
//
            for (int sample = 0; sample < buffer.getNumSamples(); ++sample) {
                auto left_sample = buffer.getSample(0, sample);
                auto right_sample = buffer.getSample(1, sample);


                //PROBLEM
                busBuffer.setSample(0, sample, (left_sample * levelsL[i]));
                busBuffer.setSample(1, sample, (right_sample * levelsR[i]));

            }
```

Figure 5.25 Function for Creating Levels

In this code we again create our notches based on the number of our busses. However, this time we define our home positions within a vector where i is the number of home positions inside the array based on the number of busses and the width of each notch. This array is used to try and keep the function as being separate processes rather than one gigantic for loop. Afterwards, we set our slider position to our pan position value and create vectors for the levels of our left and right channels.

The next step was a bit of a tricky situation for me, the for loop needed a bit of trimming and experimenting before it would compile but theoretically it would create the values of the left and right levels and then store them into individual vectors for both left and right. While in this set up it wasn't treating them as separate channels, it could be done later. An idea that was floated to me by my mentor was to start the loop at 1 and have levelL's homePosition[i] – 1 to start that at zero and the right side at 1.

After this we made a pointer to our busBuffers, these were created in a different function that I will enumerate over next. And finally, we iterated over the samples inside a for loop that added our left and right samples to our busBuffer and multiplied them by the corresponding level. At this point I wasn't getting any syntax errors.



```
void Panning::createBusBuffers(juce::AudioBuffer<float>& buffer, int numBuses, std::vector<juce::AudioBuffer<float>>& busBuffers)
    {
        busBuffers.clear();

        for (int i = 0; i < numBuses; ++i)
        {
            juce::AudioBuffer<float> busBuffer(buffer.getNumChannels(), buffer.getNumSamples());

            for (int channel = 0; channel < buffer.getNumChannels(); ++channel)
            {
                auto* busChannelData = busBuffer.getWritePointer(channel);
                auto* bufferChannelData = buffer.getReadPointer(channel);

                for (int sample = 0; sample < buffer.getNumSamples(); ++sample)
                {
                    busChannelData[sample] = bufferChannelData[sample] * (i == channel ? 1.0f : 0.0f);
                }
            }

            busBuffers.push_back(busBuffer);
        }
    }
```

Figure 5.26 Function for Creating Bus Buffers

The bus buffers were being created in their own function within the Panning class. First the buffers would be cleared before their initialization and then the compiler would enter a for loop. Inside the for loop, we first would create an array to store our bus buffers before entering an additional for loop. Inside the first section of this statement, we initialize busChannelData to write to the busBuffer and the memory block of samples and then we initialize the channel data variable to point at the sample information.

Finally, we enter that last for loop, creating a loop that looks at every sample in the buffer. We access the current sample in the bus buffer that we want to write to and then have it equal to the sample that we want to read from. Afterwards we make sure that this matches the current sample index and then multiplies it against the position in the buffer.

The code would compile! But it gave me a nasty jassert. Pretty quickly you can see why this doesn't work since it's applying the panning function twice. The create bus buffer's function does create the bus buffers but as a side effect is processing the sample rate a second time. So, I made it simpler.

```cpp
void Panning::createBusBuffers(juce::AudioBuffer<float>& buffer, int numBuses, std::vector<juce::AudioBuffer<float>>& busBuffers)
{
    // Create bus buffers with the same number of channels as the input buffer (assuming stereo)
    for (int i = 0; i < numBuses; ++i) {
        busBuffers.push_back(juce::AudioBuffer<float>(buffer.getNumChannels(), buffer.getNumSamples()));
    }
}
```

Figure 5.27 Function for Creating Bus Buffers (Simple)

However, at this point no matter how hard I tried. I kept getting a jassert that was stating that my number of channels and the channel data just weren't lining up. If I called the create bus buffer function in my process block but didn't apply the panAudioPosition my faders were smashed with invisible audio. But at least now there was audio appearing instead of nothing at all, but if I tried to call the panAudio position, the code would fail to compile.

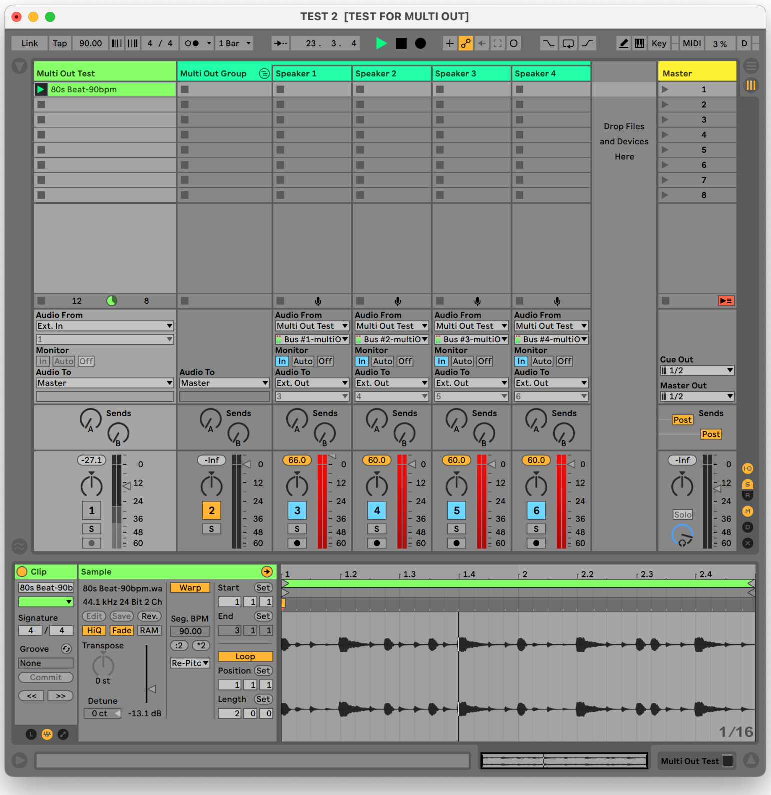


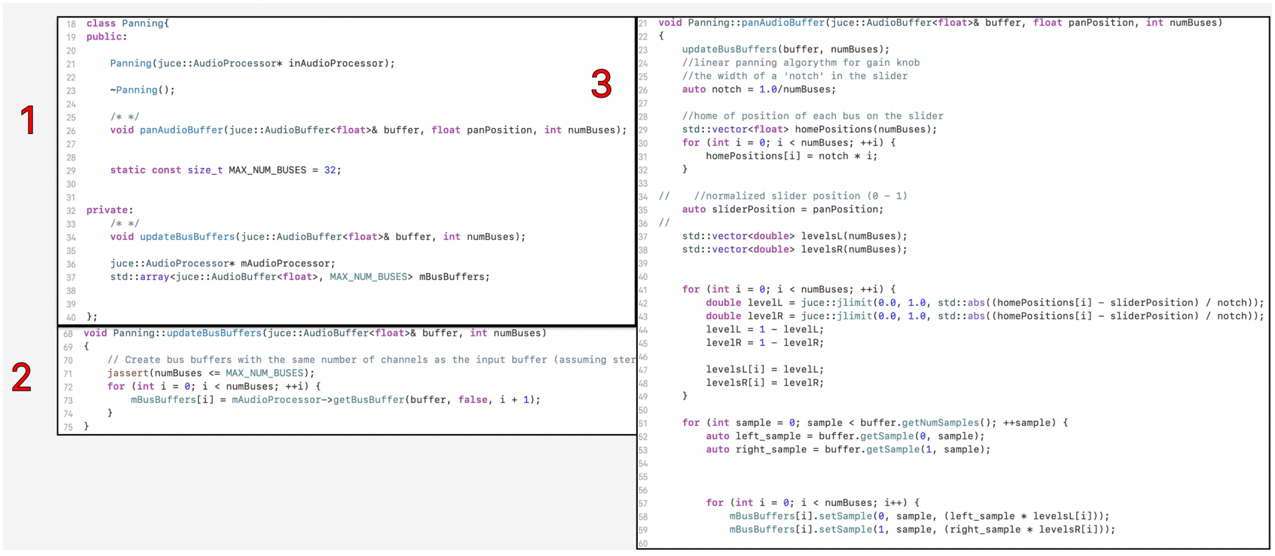
Figure 5.28 Bad Behavior

So what do we know about C++. C++ uses classes. C++ is fast. C++ is non-concurrent, and C++ is an imperative programming language. So while the directions I was giving to the compiler could theoretically work, JUCE was telling me that there was some undefined behavior that was causing a critical issue. Clearly, something strange was happening inside of the code that I had built.

One of the first things I noticed was that while I was properly declaring the audio buffer into my panning class, it was missing two critical things. I wasn't defining the bus buffers. While the code was creating a vector for the bus buffers, it wasn't creating the bus buffer objects. This is done using an important piece of code you can see if you look at

Figure 5.23, the audio processor function of getBusBuffer. So createBusBuffers wasn't creating bus buffers and filling them at all, I was just creating a vector in the style of a bus buffer.

Another thing to consider when creating these bus buffers is the processor power that it takes to ask the computer to create these objects at the sample rate, tens of thousands of times a second. Not defining some level of upper limit was causing JUCE to not only see that I was creating audio and panning it without sending it to any outputs, I was also providing the engine with undefined behavior. Where was the size of the bus buffer array being set? The computer had nothing to cycle through and had to figure it out every time it ran. This was problematic.



```
class Panning{
public:

    Panning(juce::AudioProcessor* inAudioProcessor);

    ~Panning();

    /* */
    void panAudioBuffer(juce::AudioBuffer<float>& buffer, float panPosition, int numBuses);


    static const size_t MAX_NUM_BUSES = 32;


private:
    /* */
    void updateBusBuffers(juce::AudioBuffer<float>& buffer, int numBuses);

    juce::AudioProcessor* mAudioProcessor;
    std::array<juce::AudioBuffer<float>, MAX_NUM_BUSES> mBusBuffers;


};
```

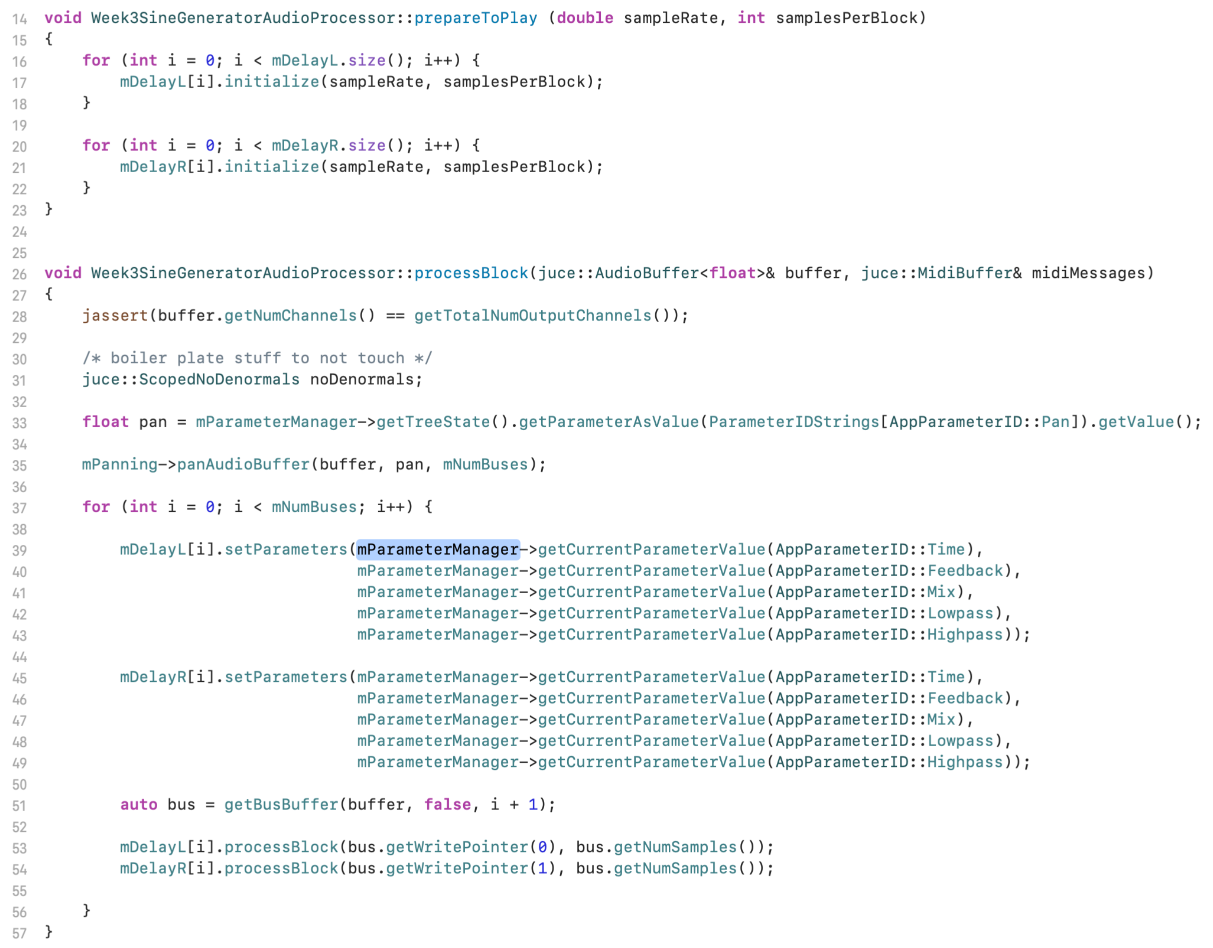
```
void Panning::updateBusBuffers(juce::AudioBuffer<float>& buffer, int numBuses)
{
    // Create bus buffers with the same number of channels as the input buffer (assuming ster
    jassert(numBuses <= MAX_NUM_BUSES);
    for (int i = 0; i < numBuses; ++i) {
        mBusBuffers[i] = mAudioProcessor->getBusBuffer(buffer, false, i + 1);
    }
}
```

```
void Panning::panAudioBuffer(juce::AudioBuffer<float>& buffer, float panPosition, int numBuses)
{
    updateBusBuffers(buffer, numBuses);
    //linear panning algorythm for gain knob
    //the width of a 'notch' in the slider
    auto notch = 1.0/numBuses;

    //home of position of each bus on the slider
    std::vector<float> homePositions(numBuses);
    for (int i = 0; i < numBuses; ++i) {
        homePositions[i] = notch * i;
    }

//    //normalized slider position (0 - 1)
    auto sliderPosition = panPosition;
//
    std::vector<double> levelsL(numBuses);
    std::vector<double> levelsR(numBuses);


    for (int i = 0; i < numBuses; ++i) {
        double levelL = juce::jlimit(0.0, 1.0, std::abs((homePositions[i] - sliderPosition) / notch));
        double levelR = juce::jlimit(0.0, 1.0, std::abs((homePositions[i] - sliderPosition) / notch));
        levelL = 1 - levelL;
        levelR = 1 - levelR;

        levelsL[i] = levelL;
        levelsR[i] = levelR;
    }

    for (int sample = 0; sample < buffer.getNumSamples(); ++sample) {
        auto left_sample = buffer.getSample(0, sample);
        auto right_sample = buffer.getSample(1, sample);



        for (int i = 0; i < numBuses; i++) {
            mBusBuffers[i].setSample(0, sample, (left_sample * levelsL[i]));
            mBusBuffers[i].setSample(1, sample, (right_sample * levelsR[i]));

```

Figure 5.29 The Solution to the jassert

In box 1, we declare a maximum number of busses for our array. Then, in box 2, instead of creating the bus buffer and adding them via pushback to the array, like in figure. 5.29 we created the bus buffers using getBusBuffer function from the audio processor. Not doing this was what was causing our undefined behavior, the program could add them in the wrong order or add too many. The bus buffers themselves were not defined before compiling so by defining them and then cycling through them in order using the getBusBuffer function we had fixed the issue with the jassert.

In box 3, we then call the updateBusBuffer function and go back through the panning function again. I was also able to see that some of the changes I made weren't working correctly. For instance, I had added notch/2 to our notch values. This was an attempt to smooth our algorithm and start exploring some of the adjustments that could be made to the linear panning function. I was trying to move the notch position from the beginning of the notch to the middle, but it didn't do that, it just made it so audio was always playing. Secondly, I had forgotten to invert our level values, so the audio was always playing. However, upon making these changes the plugin compiled and began to pan the audio buffer.



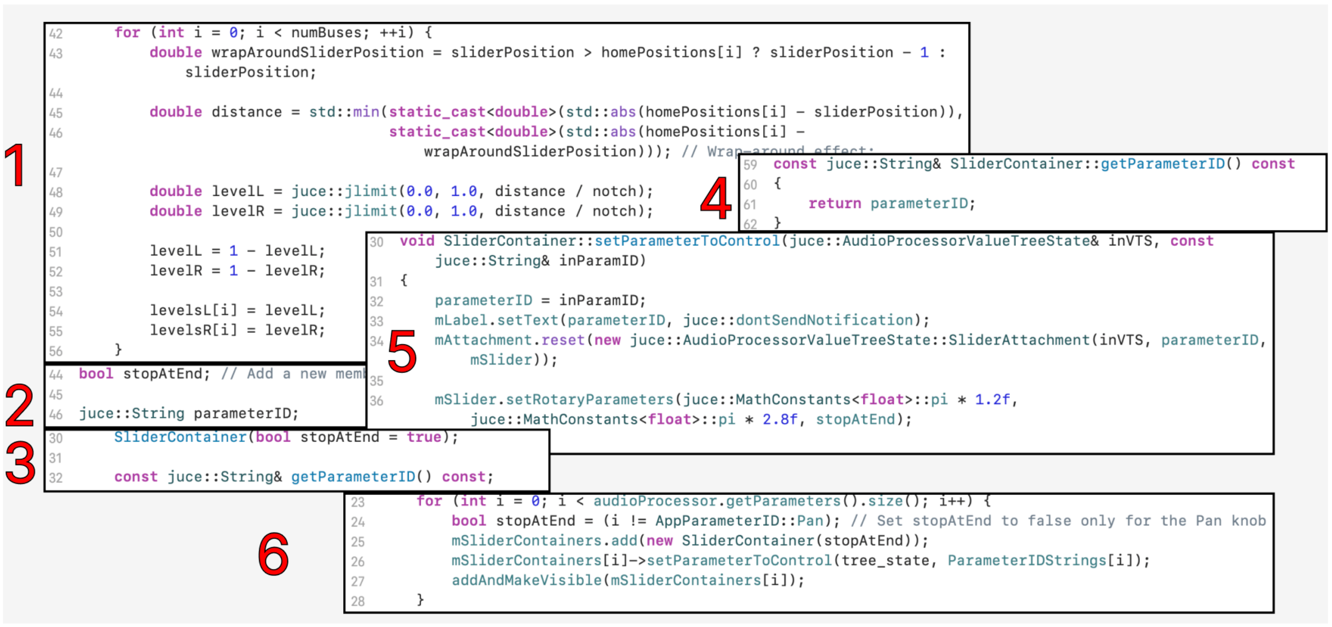
```cpp
void Week3SineGeneratorAudioProcessor::prepareToPlay (double sampleRate, int samplesPerBlock)
{
    for (int i = 0; i < mDelayL.size(); i++) {
        mDelayL[i].initialize(sampleRate, samplesPerBlock);
    }

    for (int i = 0; i < mDelayR.size(); i++) {
        mDelayR[i].initialize(sampleRate, samplesPerBlock);
    }
}


void Week3SineGeneratorAudioProcessor::processBlock(juce::AudioBuffer<float>& buffer, juce::MidiBuffer& midiMessages)
{
    jassert(buffer.getNumChannels() == getTotalNumOutputChannels());

    /* boiler plate stuff to not touch */
    juce::ScopedNoDenormals noDenormals;

    float pan = mParameterManager->getTreeState().getParameterAsValue(ParameterIDStrings[AppParameterID::Pan]).getValue();

    mPanning->panAudioBuffer(buffer, pan, mNumBuses);

    for (int i = 0; i < mNumBuses; i++) {

        mDelayL[i].setParameters(mParameterManager->getCurrentParameterValue(AppParameterID::Time),
                                 mParameterManager->getCurrentParameterValue(AppParameterID::Feedback),
                                 mParameterManager->getCurrentParameterValue(AppParameterID::Mix),
                                 mParameterManager->getCurrentParameterValue(AppParameterID::Lowpass),
                                 mParameterManager->getCurrentParameterValue(AppParameterID::Highpass));

        mDelayR[i].setParameters(mParameterManager->getCurrentParameterValue(AppParameterID::Time),
                                 mParameterManager->getCurrentParameterValue(AppParameterID::Feedback),
                                 mParameterManager->getCurrentParameterValue(AppParameterID::Mix),
                                 mParameterManager->getCurrentParameterValue(AppParameterID::Lowpass),
                                 mParameterManager->getCurrentParameterValue(AppParameterID::Highpass));

        auto bus = getBusBuffer(buffer, false, i + 1);

        mDelayL[i].processBlock(bus.getWritePointer(0), bus.getNumSamples());
        mDelayR[i].processBlock(bus.getWritePointer(1), bus.getNumSamples());

    }
}
```

Figure 5.30 A Mostly Finished prepareToPlay and ProcessBlock

New ideas for features and concepts began to materialize in my mind as the days went on, but I reminded myself it was important to tackle one small problem at a time. The final element to tackle before the end of my time at Cal Arts was *The endless knob.*

JUCE's rotary slider class naturally has a start point and an end point, and in most cases this is necessary. For example, when dealing with audio buffer inside of a plugin we use smoothing algorithms, such as JUCE's linear smoothing class, used for the panning knob in my plugin. When volume is adjusted on a single channel from quiet to loud, smoothed audio flows in one direction. Now, say audio was attached a smoothed volume control with values from zero to one that continued past the value of one, wrapping back to zero. What would happen to the audio?

Digital audio buffer doesn't move cyclically, the audio in our example would correct itself at the speed of the smoothing algorithm, from the one value back down to zero. While this isn't a huge problem when dealing with volume automation on a single channel, there is the potential for a huge problem when dealing with audio buffer position on many channels. If my pan knob wrapped back around from one to zero the audio buffer would quickly, and audibly, snap back through all the audio buffers to try and normalize its position with the slider position. The obvious solution would be to interpolate between an x and y value, applying vector-based gain the way that most surround audio plugins would.


1

```
    for (int i = 0; i < numBuses; ++i) {
        double wrapAroundSliderPosition = sliderPosition > homePositions[i] ? sliderPosition - 1 :
            sliderPosition;

        double distance = std::min(static_cast<double>(std::abs(homePositions[i] - sliderPosition)),
                                   static_cast<double>(std::abs(homePositions[i] -
                                       wrapAroundSliderPosition))); // Wrap-around effect:

        double levelL = juce::jlimit(0.0, 1.0, distance / notch);
        double levelR = juce::jlimit(0.0, 1.0, distance / notch);

        levelL = 1 - levelL;
        levelR = 1 - levelR;

        levelsL[i] = levelL;
        levelsR[i] = levelR;
    }
```

2

```
bool stopAtEnd; // Add a new mem

juce::String parameterID;
```

3

```
    SliderContainer(bool stopAtEnd = true);

    const juce::String& getParameterID() const;
```

4

```
const juce::String& SliderContainer::getParameterID() const
{
    return parameterID;
}
```

5

```
void SliderContainer::setParameterToControl(juce::AudioProcessorValueTreeState& inVTS, const
    juce::String& inParamID)
{
    parameterID = inParamID;
    mLabel.setText(parameterID, juce::dontSendNotification);
    mAttachment.reset(new juce::AudioProcessorValueTreeState::SliderAttachment(inVTS, parameterID,
        mSlider));

    mSlider.setRotaryParameters(juce::MathConstants<float>::pi * 1.2f,
        juce::MathConstants<float>::pi * 2.8f, stopAtEnd);
```

6

```
    for (int i = 0; i < audioProcessor.getParameters().size(); i++) {
        bool stopAtEnd = (i != AppParameterID::Pan); // Set stopAtEnd to false only for the Pan knob
        mSliderContainers.add(new SliderContainer(stopAtEnd));
        mSliderContainers[i]->setParameterToControl(tree_state, ParameterIDStrings[i]);
        addAndMakeVisible(mSliderContainers[i]);
    }
```

Figure 5.31 Code for Wrapping Linear Panning Algorithm

I started to make some headway into a new panning algorithm at this point. However, a small issue that I had with my initial linear panning code lead me to an unexpected answer to my problem. There was a problem with the initial code that had been developed in p5js; when the rotary slider approached the value of one, all the bus channels would be empty. The notch for the first bus position was zero, and there was no notch value for the slider position of one. This would technically be an additional position if the program operated linearly. Example: The positions of four busses would be 0, 0.25, 0.5 and .075. So, to counter this, we needed to first modify our pan algorithm. I knew I needed to let the compiler know that the panning behavior would change when the slider position moved beyond the highest home position. So, I created a conditional statement for a variable wrapAroundSliderPosition, where when the slider position was greater than the home position, the value would be the slider position minus one. This would start pointing the value to our zero-position bus.

Secondly, instead of just calculating our LevelsL and LevelsR directly within the jlimit condition, I would create a distance variable. This variable would point to an absolute minimum value of either the home position minus the slider position, as our levels did before, or the home position minus the wrapAroundSliderPosition. Technically, we're taking the same code that we used previously in part three of figure 5.29, except that when the value of the slider position went above our highest home position, the minimum statement would kick in and the smaller distance value would keep the level variable within the range of our first bus position. Now as the knob approached its highest point the first bus would fill again, reaching full volume at the position of one, fixing the problem of the empty buffers.

As I continued to attempt creating a new vector based panning algorithm, I realized I still faced the problem of creating an endless knob. While there isn't much hype or activity relating to this in the forums or just generally online, JUCE had recently added documentation for this function into the rotary parameters structure class with the stopAtEnd function. So, inside my slider container class, I added two new variables. First, I added a boolean called stopAtEnd and then I created a member variable of JUCE's string type to store a parameter ID. I knew I would have to pull my pan parameter out of the slider container's layout, which used our parameter managers value tree, to apply the wrap function to the pan knob alone and remove it from any flexboxes or other JUCE framework functions that would call parameters from our value tree.

Third, instead of an empty constructor we pass in a true value of stop at end so that all our knobs will naturally stop when they reach their minimum or maximum value. Then we declare our getParameterID function. After that, we create our function, which was a simple return of our parameter ID so we can call it later. In the fifth section of figure 5.31, at the very end of our setRotaryParameters function we pass in our stopAtEnd variable allowing us to adjust the knobs stop at end parameter. Finally, inside of our plugin editor, we create our slider containers. Since we had already set stop at end to true for our sliders, we don't need to specify anything for the majority of them. However, we do set our stop at end to false for our pan knob specifically. This allows for the pan knob to spin endlessly inside of our plugin.

At this point, I was afraid that I was going to see my audio buffer quickly pass backwards through all the busses as it tried to normalize itself to the position of the slider. However, because of the new wrapping function, the audio buffer was already being passed directly into the first bus and wouldn't normalize as it was already in the correct position. While this behavior was unintended, all my issues related to the endless wrapping knob had been solved. I no longer had to work on vector-based panning as my knob was functioning correctly.

This might not be the most important aspect of the plugin to many performers, as many MIDI controllers use potentiometers with hard starts and stops. Additionally, inside of the DAW the parameter would operate as a linear slider with a true minimum and maximum. However, with some additional set up I could now use a rotary encoder to pan my audio in endless loops just like the Ableton Surround Panner. However, in my version, as the audio passed through each bus the delay lingers allowing for a more analog sounding audio effect.

Now the plugin was functioning in its most basic form. In our prepareToPlay, we initialized our delay. Instead of initializing it as just a variable, we had declared it as a two-dimensional array, passing in our delay class and the maximum number of busses. This would allow us to pan the delay across our output busses. We then defined the variable pan, calling its value from the corresponding parameter that was adjusted by its slider attachment. After that we sent our audio buffer through our panAudioBuffer function, passing in the audio buffer, the pan position, and the number of busses. Finally, inside of a for loop, we called the functions of our delay parameters, filled them into the bus buffers and then loaded them into the corresponding delay array.

At this point the plug-in, while not being finished, fulfilled the initial requirements. It was scalable to as many outputs as a DAW would allow. It was simple and easy to use. It contained a delay that would iterate along the multiple buses in a natural way as it endlessly panned back and forth along an array of speakers. Now, it was time to start diving deeper into panning algorithms and adding the functions that would increase the performability of the application.

Chapter 6
In Conclusion

Surround Sound is a complex beast. The world of parameter-based performance has yet to be fully explored. Digital audio processing has almost limitless possibilities, but the introduction of bits to analog sound makes the syntax long and sometimes confusing. Einstein said it best, "If you can't explain it simply, you don't understand it well enough." This applies doubly when you're attempting to learn how to code as well as the context behind the concept you're coding.

The idea of simply creating an application that would allow me to perform a digital sound bath is long gone. The mechanisms behind it, I'm continuing to learn. However, the concepts remain the same. For almost as long as our earth has existed, sound has existed as the vibrations in the airwaves. It has remained ripples in space that we perceive with our ears in only specific necessary frequencies that we adapted to hear for our survival. Sound is the space that we live in, we don't live in a vacuum we are surrounded by air. The air that we breathe to survive that is constantly filled with sound.

It's been only a blip in the span of the universe that humankind has had electrified noise. But since the beginnings of amplified sound our species has done it's best to try and recreate the effect of the natural state of the earth. Recreating sound in space. With the advent of technological evolution, the applications of speaker systems went from weak metal telephones to violently loud and overly complex arrays of vast networks of arrays in the case of wave field synthesis.

In the twentieth century, computational devices grew stronger and stronger allowing for more complex problems to be addressed allowing for the growth of beefier algorithms and faster processing. Soon analog concepts became digital and advanced processes became simplified, no longer just high academic research projects but applications for the average joe. Does everyone need to understand the abstract concepts that lead us to the place that we are now?

The short answer is no. In the case of surround sound, not every mixing engineer and musician needs to understand the history of surround sound from Theatrephone to 4DSOUND. They also don't need to know C++ and the rules that allow them to recreate their guitar pedals inside of their computer. All they need to know is that there is something for them to use out there. Something for them to create with. The idea behind Surround Sounder is the idea that anyone should be able to boot up a simple plug in, borrow some speakers from their friends, plug into an interface with multiple outfits and start to perform in a spatialized format.

It doesn't matter whether they use it to automate sounds in an installation, provide therapeutic sound baths or perform epic live concerts with the plugin what matters is that they can. Maybe they'll grow tired of it, or maybe they'll find it interesting. Maybe someone just might find it interesting enough that they start work on new projects. New projects that involve things like wave field synthesis and 4DSOUND. They start to investigate the relationship of sound in the space that we perceive. And maybe down the line they invent something new, something better that hasn't been done before.

When computers first appeared, it was hard for people to adopt them, the barrier of entry was so high to learn. These days anyone can pick up a smart phone and intuitively send data that was too high of a resolution to even be stored on the biggest hard drives. Surround Sound has been treated as an art of the elite for a long time. Dolby saw to that. Even though we have been using widely since the 1970s, it hasn't grown in popularity and the idea that it takes so much more talent to mix in surround sound is wrong.

When we reduce the barrier of entry to using technology, more people can use it and more people can understand and experiment with it. This doesn't make the technology less valuable it makes is more valuable. Creating a simple plugin that I could set up and perform a digital sound bath was a concept I came up with after my experience performing with parameters. Had I not spent as much time troubleshooting and contemplating how to do it, I wouldn't have been able to conceive the idea of treating a sound object as an ellipse along an array, a position upon a knob. What Ableton Live's Surround Panner viewed as an afterthought; I viewed as a way to evolve my live electronic performance.

As a musician with hardly any experience coding, I was able to begin to learn JUCE's framework to build an application. The process of building an application that was done from scratch as recently as 2004. Had I had this idea back then, I would have had nowhere to begin besides learning how to code from scratch. The idea would never have existed in the first place because many of the plugins that I used on my own performances were built in JUCE!

So, if I can help someone, besides myself, who might be interested in using surround sound but felt like the applications are all too complex, I've achieved my goal. Making things simple for someone who has an idea but hasn't been able to bring it to life yet, that's what this is about. Take something that has been turned into a skill that only the elite can attain and make it something that anyone can use it. That was the real reason people like Disney wanted to make *Fantasia,* to create something everybody would understand and enjoy.

I spent more time at Cal Arts than many graduate students, but less in my program than most. The time that I spent was primarily focused on conceptualizing and implementing an application that was far out of my usual knowledge base. In the future, I hope to use it. To perform with it and experiment with set ups and sounds and collaborate with artists like Kai Luen to build set ups with visuals that follow the objects around spaces. I didn't build this thing to just look at it and let other people use it. I built it because I loved the idea of playing music in surround sound for more than just myself.

Imagine you're in the nightclub, the whole room is dark. But from the back right corner a little blip starts to play. And then from the left, another blip joins the fray. They blip back and forth getting faster and faster before slowly moving from the back to the front. And a light starts to shine, bright as day. It moves and it grows before suddenly all that sound goes away and the band starts to play. That's it, and that's all that I'll say.

Thank You for Reading

Bibliography

“4DSOUND.” Accessed March 27, 2023. https://4dsound.net.

“80 Years On & Counting: Progress In ‘Getting It Right’ With Speech Reinforcement - ProSoundWeb.” Accessed February 2, 2023. https://www.prosoundweb.com/80-years-on-counting-progress-in-getting-it-right-with-speech-reinforcement/.

“A Silver Screen Special Edition: Ladies and Gentlemen: This Is Cinerama.” Accessed February 21, 2023. https://www.blu-ray.com/news/?id=9433.

Abbey Road. “Abbey Road Studios Celebrate Alan Blumlein’s Birthday.” Accessed February 2, 2023. http://www.abbeyroad.com/news/abbey-road-studios-celebrate-alan-blumleins-birthday-2213.

“About — SSI.” Accessed March 1, 2023. https://spatialsoundinstitute.com/about.

Achinstein, Peter. “Waves and Scientific Method.” *PSA: Proceedings of the Biennial Meeting of the Philosophy of Science Association* 1992 (1992): 193–204.

“Adding the Sound to Cinerama.” Accessed February 21, 2023. https://www.in70mm.com/cinerama/archive/sound/index.htm.

“AFI|Catalog - This Is Cinerama.” Accessed February 21, 2023. https://catalog.afi.com/Film/50687-THIS-ISCINERAMA.

“ANALOG Arts » Recreating the Philips Pavilion.” Accessed March 15, 2023. http://analogarts.org/anablog/pavilion/.

Black Mountain College Museum + Arts Center. “LISTENING SESSION: Williams Mix by John Cage,” December 10, 2016. https://www.blackmountaincollege.org/listening-session-williams-mix-john-cage/.

Bornoff, Jack. “Hermann Scherchen - Youth and Music / Hermann Scherchen — Jeunesse Et Musique / Hermann Scherchen — Jugend Und Musik.” *The World of Music* 1, no. 3 (1959): 42–45.

Cantrell, Olivia. “The Making of Fantasia: Disney’s Most Ambitious Masterpiece.” *Houston Symphony* (blog), October 26, 2022. https://houstonsymphony.org/disney-fantasia/.

Dannenberg, Roger B. “A Perspective on Computer Music.” *Computer Music Journal* 20, no. 1 (1996): 52–56. https://doi.org/10.2307/3681271.

“DDSP-VST.” C++. 2022. Reprint, Magenta, March 27, 2023. https://github.com/magenta/ddsp-vst/blob/cf5d3c803ba2b26e444e179b855e99c74d48320b/README.md.

Dienstfrey, Eric. “The Myth of the Speakers: A Critical Reexamination of Dolby History.” *Film History* 28, no. 1 (2016): 167–93. https://doi.org/10.2979/filmhistory.28.1.06.

Digital Cinema Report. “Dolby Atmos: Past, Present and Future,” June 25, 2019. https://digitalcinemareport.com/article/dolby-atmos-past-present-and-future.

DJ TechTools. “The 4D Soundsystem: Max Cooper Interview,” October 25, 2013. https://djtechtools.com/2013/10/24/the-4d-soundsystem-max-cooper-interview/.

Eck, Cathy van. “An Active Loudspeaker by Hermann Scherchen.” *Between Air and Electricity* (blog), March 10, 2017.

https://microphonesandloudspeakers.com/2017/03/10/active-loudspeaker-hermann-scherchen/.

"Ecosystem | JUCE C++ Library," August 7, 2013. https://web.archive.org/web/20130807023518/http://www.juce.com/ecosystem.

"Electrophone in England (1901)." Accessed January 29, 2023. https://earlyradiohistory.us/1901elec.htm.

Endpoint Audio Labs. "Optical Sound Examples." Accessed May 1, 2023. https://www.endpointaudio.com/optical-sound-detail.

"Espacenet – Search Results." Accessed January 29, 2023. https://worldwide.espacenet.com/patent/search/family/010368359/publication/GB394325A?q=pn%3DGB394325.

ETHW. "Fantasound," April 12, 2017. https://ethw.org/Fantasound.

"Fantasia Movie Poster (#4 of 9)." Accessed February 21, 2023. http://www.impawards.com/1940/fantasia_ver4.html.

"'Fantasound.'" Accessed February 21, 2023. http://www.r-type.org/articles/art-299.htm.

"FANTASOUND*." Accessed February 21, 2023. http://www.widescreenmuseum.com/sound/fantasound1.htm.

Frankel, Eugene. "Corpuscular Optics and the Wave Theory of Light: The Science and Politics of a Revolution in Physics." *Social Studies of Science* 6, no. 2 (1976): 141–84.

"Harvey Fletcher – the Father of Stereophonic Sound | SciHi Blog." Accessed February 2, 2023. http://scihi.org/harvey-fletcher/.

Hidden City Philadelphia. "At The Academy, The Birth Of Wire Transmission Of Music," April 30, 2013. https://hiddencityphila.org/2013/04/at-the-academy-the-birth-of-wire-transmission-of-music/.

"History and Types of Speakers." Accessed January 29, 2023. https://edisontechcenter.org/speakers.html.

histv. "Du Moncel - Système Ader." Accessed January 29, 2023. https://www.histv.net/du-moncel-systeme-ader.

IMDb. "Apocalypse Now (1979)." Accessed March 21, 2023. https://www.imdb.com/title/tt0078788/mediaviewer/rm938805761/?ref_=tt_ov_i.

iZotope. "Gain Staging: What It Is and How to Do It." Accessed April 4, 2023. https://www.izotope.com/en/learn/gain-staging-what-it-is-and-how-to-do-it.html.

"Jazzmutant Lemur." Accessed April 4, 2023. https://www.soundonsound.com/reviews/jazzmutant-lemur.

Jiang, Zhiping, Xuefeng Tang, Simin Wang, Ge Gao, Dajiang Jin, Jianfeng Wang, and Minggang Si. "Progresses of Pilot Tone Based Optical Performance Monitoring in Coherent Systems." *Journal of Lightwave Technology* 40, no. 10 (May 2022): 3128–36. https://doi.org/10.1109/JLT.2022.3146232.

"JUCE Announces Acquisition by PACE | JUCE," April 19, 2020. https://web.archive.org/web/20200419092952/https://juce.com/discover/stories/juce-announces-acquisition-by-pace.

Keefer, Cindy. "'RAUMLICHTMUSIK' - EARLY 20TH CENTURY ABSTRACT CINEMA IMMERSIVE ENVIRONMENTS," n.d.

"Loudspeaker History." Accessed January 29, 2023. https://www.aes-media.org/historical/html/recording.technology.history/loudspeaker.html.

Loy, Gareth. “Musicians Make a Standard: The MIDI Phenomenon.” *Computer Music Journal* 9, no. 4 (1985): 8–26. https://doi.org/10.2307/3679619.

“Max Cooper on the 4DSOUND System (2013) — SSI.” Accessed March 27, 2023. https://spatialsoundinstitute.com/Max-Cooper-on-the-4DSOUND-System-2013.

McDermott, Zach. “What Is the Blumlein Technique?” AEA Ribbon Mics & Preamps, February 4, 2018. https://www.aearibbonmics.com/what-is-blumlein-technique/.

Morrison, Geoffrey. “Surround All around: Dolby Atmos Explained.” CNET, January 19, 2022. https://www.cnet.com/tech/home-entertainment/dolby-atmos-what-you-need-to-know-about-the-spatial-audio-format/.

Nordost. “The Evolution of HiFi: The Shearer Horn.” *Nordost Blog* (blog), August 7, 2018. https://nordost.com/blog/the-evolution-of-hifi-the-shearer-horn/.

Oellers, Helmut. “Wave Field Synthesis.” Accessed March 27, 2023. http://www.syntheticwave.de/Wavefieldsynthesis.htm.

“P_Sound Holograms — SSI.” Accessed March 27, 2023. https://spatialsoundinstitute.com/P_Sound-Holograms.

“Ray Dolby: Inventing the Future of Entertainment.” Accessed March 21, 2023. https://www.dolby.com/about/leadership/ray-dolby/.

ResearchGate. “Fig. 1: Coordinates Used in the Kirchhoff-Helmholtz Integral Equation.” Accessed March 27, 2023. https://www.researchgate.net/figure/Coordinates-used-in-the-Kirchhoff-Helmholtz-integral-equation_fig3_263013469.

Scientific American 1925-09: Vol 133 Iss 3. Nature America, Inc., 1925. http://archive.org/details/sim_scientific-american_1925-09_133_3.

Sennheiser.com/Bluestage. “ONES TO WATCH: Spatial Sound.” Accessed March 27, 2023. http://en-us.sennheiser.com/shape-the-future-of-audio-paul-oomen.

“SHEARER HORN.” Accessed January 29, 2023. https://www.audioheritage.org/html/profiles/lmco/shearer.htm.

“Software — 4DSOUND.” Accessed March 27, 2023. https://4dsound.net/software.

Solomon, Charles. “Fantastic ’Fantasia’ : Disney Channel Takes a Look at Walt’s Great Experiment in Animation.” Los Angeles Times, August 26, 1990. https://www.latimes.com/archives/la-xpm-1990-08-26-tv-552-story.html.

“Stereo_shuffling_A4.Pdf.” Accessed January 29, 2023. https://www.audiosignal.co.uk/Resources/Stereo_shuffling_A4.pdf.

Stroustrup, Bjarne. “A History of C++: 1979--1991.” In *History of Programming Languages---II*, edited by Thomas J. Bergin and Richard G. Gibson, 699–769. New York, NY, USA: ACM, 1996. https://doi.org/10.1145/234286.1057836.

Suber, Peter. “What Is Software?” *The Journal of Speculative Philosophy* 2, no. 2 (1988): 89–119.

Sundius, Michael. “Weekend Rewind: Deadmau5 Explains His Live Setup from 2008.” Dancing Astronaut, February 8, 2015. https://dancingastronaut.com/2015/02/weekend-rewind-deadmau5-explains-live-setup-2008/.

“The Auxetophone.” Accessed January 29, 2023. http://www.douglas-self.com/MUSEUM/COMMS/auxetophone/auxetoph.htm.

“The Doctrine of Description: Gustav Kirchhoff, Classical Physics, and the ‘Purpose of All Science’ in 19th-Century Germany - ProQuest.” Accessed March 27, 2023.

https://www.proquest.com/openview/0193bd2732814c7e6d9a16c12cf752f5/1?pq-origsite=gscholar&cbl=18750.

“The Electrophone (1893).” Accessed January 29, 2023. https://earlyradiohistory.us/1893elec.htm.

“The Forgotten Johann Philipp Reis - Integrated Network Cable - National or Local IT Rollouts and Installation Servcies.” Accessed January 29, 2023. https://web.archive.org/web/20150612190406/https://www.integratednetworkcable.com/technology/the-forgotten-johann-philipp-reis.

The History of Magnavox. “Timeline.” Accessed January 29, 2023. https://magnavoxhistory.com/timeline/.

“The Queen and the Electrophone (1899).” Accessed January 29, 2023. https://earlyradiohistory.us/1899elec.htm.

“The Todd-AO Process.” Accessed February 21, 2023. https://www.in70mm.com/newsletter/2002/67/what_is/index.htm.

Tygel, M, L T Santos, J Schleicher, and P Hubral. “The Kirchhoff-Helmholtz Integral Pair,” n.d.

Valiquet, Patrick. “The Spatialisation of Stereophony: Taking Positions in Post-War Electroacoustic Music.” *International Review of the Aesthetics and Sociology of Music* 43, no. 2 (2012): 403–21.