SIN(language)

Visualizing Acoustics During COVID-19 With Spatial Signal Processing

New Mexico
Supercomputing Challenge
Final Report
April 6, 2021

Team 13
New Mexico School for the Arts

Team Members:
- Rowan Jansens
- Madelyn Kingston
- Brandon Morrison

Teacher:
- Mohit Dubey

Project Mentor:
- Stephen Guerin

Table of Contents

1. Executive Summary

Waves are among the most foundational concepts in physics today. They govern every aspect of our world from the very small--such as wave functions in quantum mechanics--to the very large--such as gravitational waves rippling through spacetime. In this project, we are looking specifically at acoustic sound waves as we explore three questions: What is the nature of sound waves? How can they be modeled? And what are the applications of a visual model of sound in the real world?

We began by delving into the mechanics of modeling sound waves, first by looking at the necessary and applicable equations and then by experimenting with how to implement these equations in NetLogo. Once we developed a computational model that gave us a good visualization of how sound waves behave, we branched out and explored two separate applications of our model in the real world.

The first application was focused on Spatial Audio Rendering. In this application, we used the NetLogo model as a tool to transform a raw sound file input into an output that incorporates the effects that a certain space has on a sound.

The second application focused on creating a Sonic Proximity System that calculates the relative distance between any two smartphones using ultrasonic chirps. The logic and feasibility behind this system was analyzed in the NetLogo model while a real version of the system was developed in JavaScript.

2. Introduction

a. Project Background

The global COVID-19 pandemic introduced a number of challenges that we have never faced before. With our interest in waves, the pandemic provided us with inspiration for the many applications of our model. At its core, our project is about modeling spatial acoustic dynamics. Our most significant achievement has been developing a technique for rendering audio performances by simulating the acoustics within a modeled space.

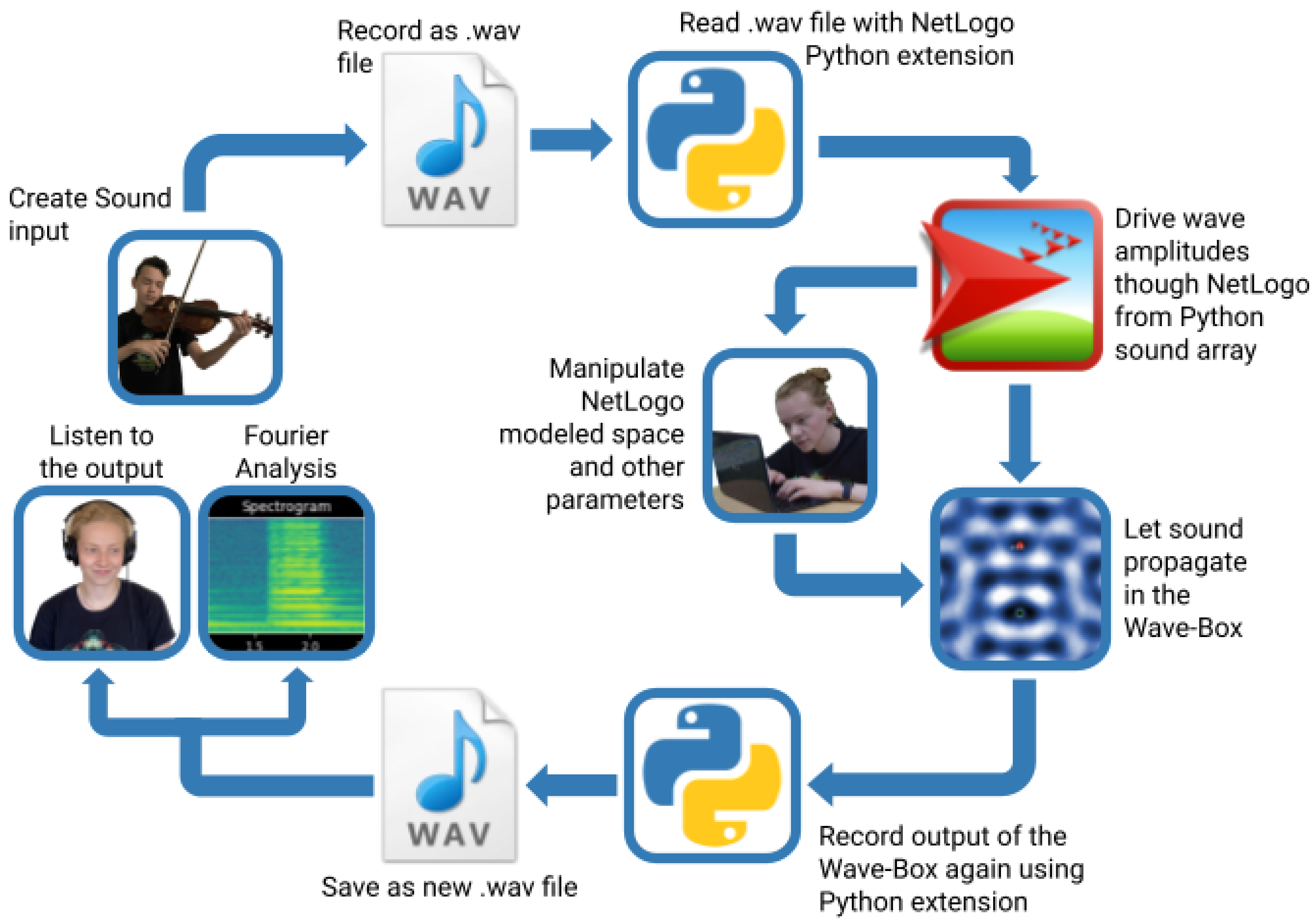
This interest in audio rendering emerged from the limitations that COVID-19 put on musicians playing their instruments together in the same space. With many of the students at the New Mexico School for the Arts studying music, this question seemed fitting for us to study. Our idea was that with an understanding of spatial acoustics, we could mix several different audio tracks together to create a musical performance that sounded like it was recorded in person and in a real concert hall.

COVID-19 also led us to study the use of sound as a tool to measure distance in a Sonic Proximity System. The question was how do we use smartphones to monitor social distancing while maintaining privacy and simplicity.

b. Project Outline

The purpose of this project is to create a tool that we can use to study various acoustic systems such as a Spatial Audio Rendering System or a Sonic Proximity System. In the next sections, we explain the equations that aided in our study of sound waves and discuss how we developed a sand-box for researching waves in NetLogo, which we have named the

"Wave-Box." We will then discuss the Spatial Audio Rendering system, the steps taken that led to its success, and the notable results of this system (Figure 1).



Create Sound input
Record as .wav file
WAV
Read .wav file with NetLogo Python extension
Drive wave amplitudes though NetLogo from Python sound array
Manipulate NetLogo modeled space and other parameters
Let sound propagate in the Wave-Box
Record output of the Wave-Box again using Python extension
Save as new .wav file
WAV
Listen to the output
Fourier Analysis
Spectrogram

Figure 1. Flow Chart outlining the steps for our exploration of Spatial Audio Rendering, showing the complete path a sound clip takes when it is recorded, rendered, and then returned with spatial effects.

Next, we will describe our attempts at developing and testing the Sonic Proximity System as well as our experiences with this application. Finally, we will discuss the major takeaways from the project as well as recommendations for future work.

3. NetLogo Wave-Box Simulation

a. Shallow Water Wave Equations

To begin our study of acoustic waves, we first took time to better understand the mathematical methods used to model any wave in general. The first scenario we considered was a model of longitudinal waves along a length of rope. In such a scenario, the change in the upward and downward velocity of a particular node (turtle) in the rope is influenced by the mean heights of its neighboring nodes relative to its own height. In other words, the height of a node, y, is a function of both position and time, $y = f(x, t)$. When we moved to 2D wave equations, our nodes became patches instead of turtles.

We modeled this behavior using the following equation which relates the velocity of a patch to its position and time.

$$\frac{\partial y}{\partial t} = \sum f(n, t) - (f(x, t) * k)$$

Where

- n is the relative position of the neighboring patches
- k is the number of neighbors. In the case of a rope, $k = 2$

This equation provided us with a method for updating the vertical velocity of a patch with respect to the position of its neighbors. The equation could then be applied to higher dimensions to model waves in a plane or even a field in R^3. For our 2D model, it made computational sense to chose $k = 4$ to represent the patches in the 4 cardinal directions to the patch being calculated by the model. Figure 3 shows that even with $k = 4$, the model produced distinctly curved wavefronts.

b. Tuning the Model

Although these wave equations were initially used to understand water waves, they can be used to model the behavior of sound as pressure waves in air. Instead of modeling the velocity of patches in an incompressible medium such as water, we modeled the pressure of patches in a compressible medium such as air. At this point, we chose to alter the equation slightly to make it more representative of sound waves. We did this by incorporating two variables that we could tune in order to make the model match some predefined quality such as the inverse square law. The variables were named “Surface Tension” and “Sustain” due to the visual effect they had on the model.

We ran a behavior space, sweeping through the two variables to explore the “phase space” of the model and measure the amplitudes of the waves at different distances. Then, we calculated a score that represents how closely the model obeyed the inverse square law using the following formula: $\text{score} = |0.25a_1 - a_2|$

In this equation, a_1 is the amplitude at some distance from the wave source and a_2 is the amplitude exactly twice as far away from the source as a_1. A score of 0 indicates a perfect inverse square attenuation.

c. Results

After running the behavior space consisting of over 400 combinations of variables, we analyzed the data to determine the pair of variables that scored lowest. The results are displayed in Figures 2.1 and 2.2.

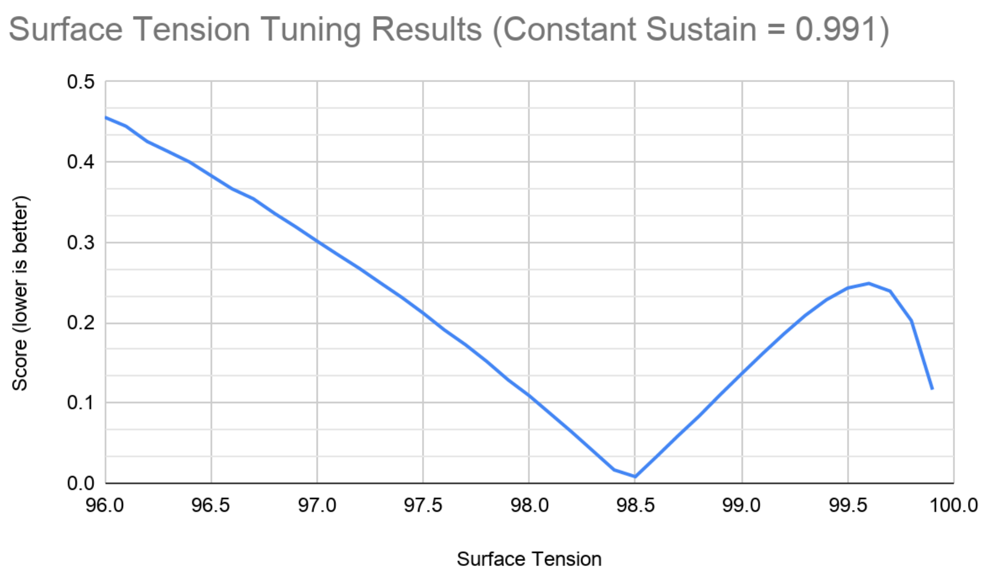


Figure 2.1. "Surface Tension" Variable Behavior Space Results. Score is minimized when Surface Tension = 98.5

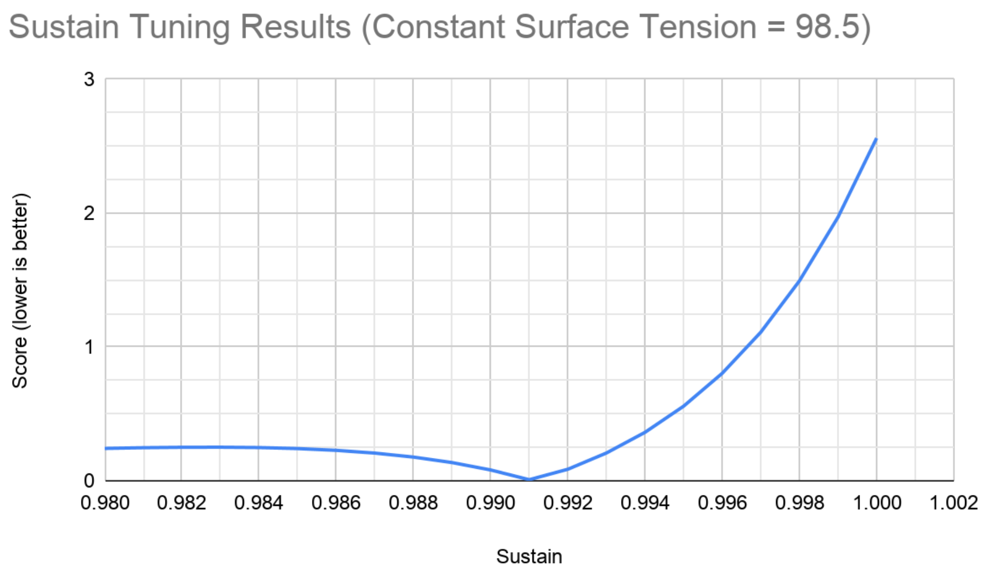


Figure 2.2. "Sustain" Variable Behavior Space Results. Score is minimized when Sustain = 0.991

The inflection points at the x-axis in both graphs are a result of the absolute value term in our score formula.

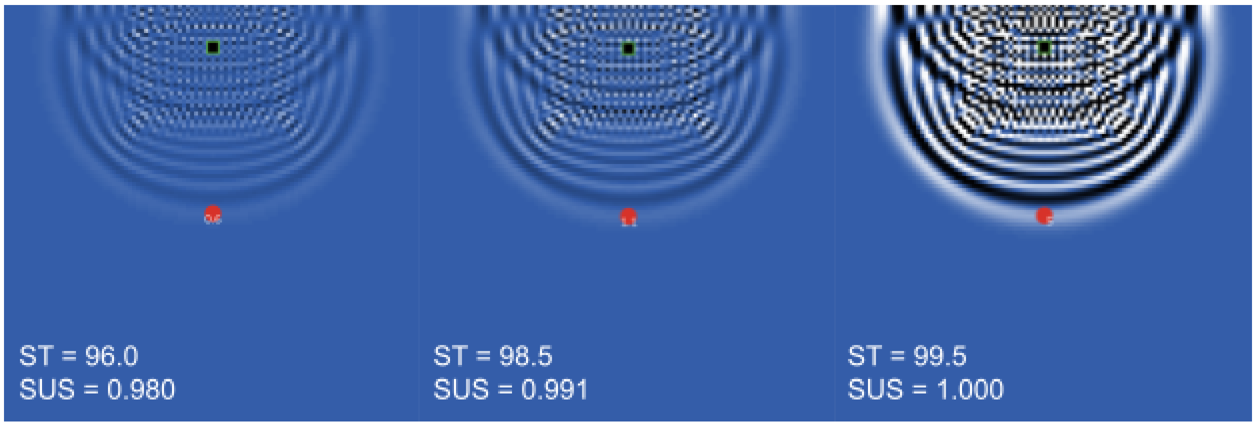


Figure 3. Comparison of different Surface Tension (ST) and Sustain (SUS) variables. The center image is generated using the variable chosen to best match the inverse square while the two outer images are generated using the minimum and maximum values of the tested variables.

With this combination of optimized variables, the model matches the inverse square law with a margin of error of less than one percent.

4. Spatial Audio Rendering

a. Background

From here, we applied our model of how sound waves behave to our main focus in acoustics: Spatial Audio Rendering.

Audio rendering, in the context of music, film, and even video games, is the process of combining many sound input recordings into a single output recording. With COVID-19 preventing many musicians from playing music together in the same space, the specific question we asked is how do we use Spatial Audio Rendering to combine multiple recordings (or sound files) to simulate musicians playing in the same space?

Rendering an acoustically detailed and accurate scene involves understanding how a certain space changes the way things sound. For example, when sound waves originate from an instrument in any room, they travel through the air in all directions, bouncing off of walls or being absorbed by other surfaces in the room. This intricate combination of reverberations is what distinguishes a concert played in a large concert hall from one played in a bedroom. Through Spatial Audio Rendering, it is possible to simulate the natural effects a space has on sound waves, which is necessary to make a rendered sound be as realistic as possible.

There are two common ways to incorporate the effects of a space into a track through audio rendering, both of which are mathematically based: algorithmic reverb and convolution reverb. Algorithmic reverb is the product of synthesized computer manipulation of the waveform of a sound and is used to imitate a space or form an imagined effect (a common example is spring reverb). Convolution reverb is the product of recording an impulse response file and then mathematically combining it with a sound file (MusicTech). The impulse response file is created by initiating a trigger sound (usually a clap, gunshot, or other sound with a high initial volume peak) and then recording the space-specific decay profile of the sound.

While convolution reverb creates a more accurate representation of a space, this is impractical for spaces that have not been previously recorded and are not accessible in a digital sound library. Additionally, it is often the case that convolution and algorithmic reverb libraries require purchase for use with a Digital Audio Workstation (DAW). Furthermore, for video games, where the scene exists only as a digital 3D model, neither convolution nor algorithmic reverb will accurately represent the way sound would interact with the space[1].

Our goal was to investigate a new method for Spatial Audio Rendering that relies entirely on simulating the sound in a space using the Wave-Box. With the Wave-Box, the rendered output of a sound comes directly from the simulated movement of the sound within a space. This

[1] One other potential way to render sound is to use ray tracing or even bi-directoinal ray tracing which is often used to render images in video games and movies. However, this method would stray too far from the functionality of the Wave-Box.

eliminates the need for complex algorithms and/or impulse response files. Although this method had not previously been used for Spatial Audio Rendering and high-level sound/music production, it presents a more intuitive approach to rendering sound than that in a DAW.

b. Development

The first step in preparing the Wave-Box for audio rendering was to incorporate a way to play a .wav file in the model space. A .wav file is one of the simplest ways to encode sound in a digital file as it is just an uncompressed[2] list of time-domain amplitudes. With the help of the Python extension and soundfile library, NetLogo can read this data directly. We then built a speaker and microphone object within the NetLogo environment. This allowed us to play an input .wav file through a simulated speaker by continuously[3] setting the amplitude (pressure) of the field at the location of the speaker to the value specified in the .wav file. We then recorded the output of the sound with the microphone object by continuously writing the amplitude of the field to a new file.

Through this method we were able to render sound by emitting it from one location, letting it propagate in the room, and then recording it at a different location. This allowed the space to affect the sound by infusing it with natural echoes, reverberation, and distortion before it was picked up by the microphone.

For our rendering tests, we recorded a 3.5-second clip of two notes being played on a violin. The model runs slowly, so the clip was recorded at an 8khz sample rate rather than the traditional 48khz used in most recording scenarios. This allows us to render the clip 6 times as

[2] Compressed audio file types, such as MP3 and AAC, would require us to first uncompress the file to access the raw data. This is what made uncompressed .wav files the preferred format for our recordings and renderings.
[3] Each tick in the model represents the next sample in the .wav file so it would take 48 thousand ticks to play one second of audio recorded at 48khz

fast with the only drawback being a removal of the higher overtones above 4khz. However, even with these optimizations it still takes several minutes to render one second of audio (although a faster computer would shorten this duration). While this technique is slow to render in Netlogo, porting the model to Agentscript (Javascript in the Browser) should allow us to speed up rendering time by a factor of 20.

It should be noted that for all of the following tests, the amplitude of the rendered output sound was normalized so that it could be audible regardless of the input levels. Therefore, the main result was the change in amplitude and frequency structure rather than volume alone.

c. Results

The results[4] of these simulations were rather surprising. In the first render test, the Wave-Box was configured with the optimized Surface Tension and Sustain variables introduced in the development of the Wave-Box. The microphone and speaker were placed near one another as shown in figure 4 and the clip was rendered. However, the output was extremely degraded as is evident in figure 5.

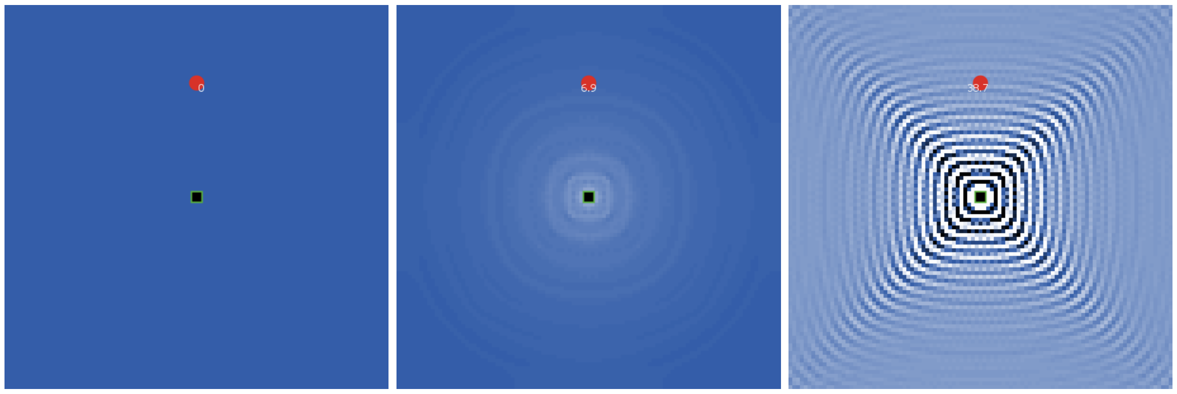


Figure 4. First render test. The black square is the speaker and the red dot is the microphone. Surface Tension = 98.5 Sustain = 0.991

[4] If viewing from a computer, the audio clips can be accessed using this google-drive link

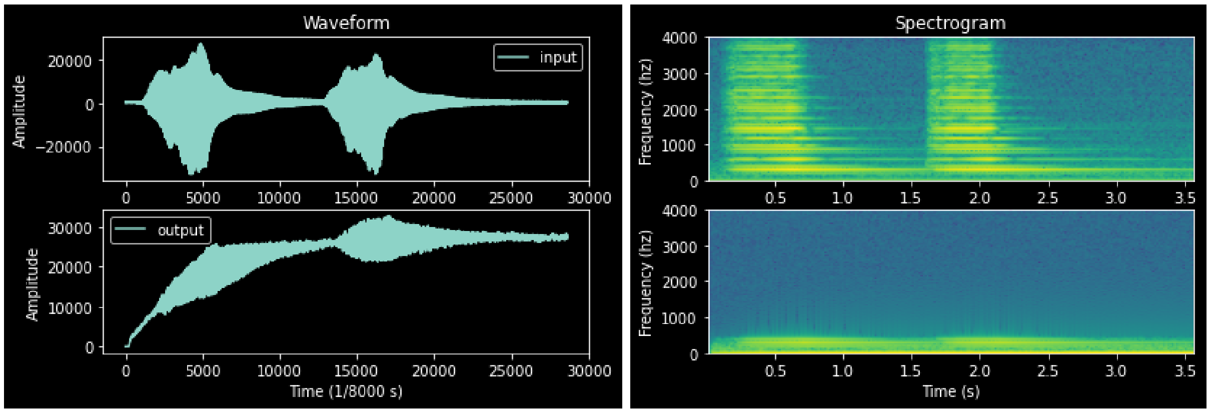


Figure 5. Waveform and spectrogram data from the first render test. The top graphs show the audio input and the bottom shows the rendered output. In the output, note the loss of frequency data (overtones in the original violin recording) above 500hz. Surface Tension = 98.5 Sustain = 0.991

In this first experiment, almost all of the high-frequency data was missing, which led to a distorted result. However, we observed that by decreasing the Surface Tension variable, the rendering improves dramatically, which we did not anticipate. In fact, as the Surface Tension decreases, the cut-off in the data for higher frequencies that occurs at around 500hz in figure 5 continues to rise, including more of the overtone series. With Surface Tension set to 50 (the minimum value before the model breaks down) and with Sustain left unchanged at 0.991, the frequency data cut-off was practically non-existent resulting in an output with no audible frequency degradation. With these settings, the output was nearly identical to the input, which is what we had expected for the scenario when the microphone is placed directly next to the speaker. The added presence of echo and reverberation accounts for the increased smoothness in the waveform as well as increased noise in the spectrogram. These results are displayed in figure 6.

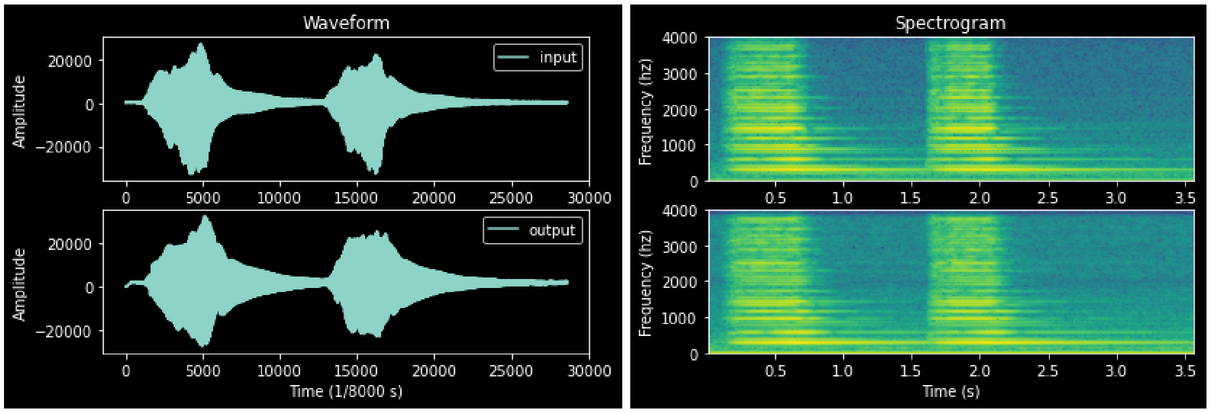


Figure 6. Waveform and spectrogram data from the second render test. On top is the audio input while on bottom is the rendered output. Surface Tension = 50.0 Sustain = 0.991

We also experimented with changing the Sustain variable and observed that this variable affects how much reverberation there is in the final result. Choosing a lower Sustain added less reverb to the output (making it resemble the original recording more closely) while choosing a higher Sustain increased the amount of reverb on the output (making it more noisy/echoey). By coincidence, it is common to also have a "sustain" variable within a DAW to adjust convolution and analytical reverb. It was shocking for us to discover that our use of this homonymous variable exactly mirrors the acoustic effects of increasing/decreasing this variable within a DAW. Figure 7 shows a rendered comparison of choosing low and high Sustain values.

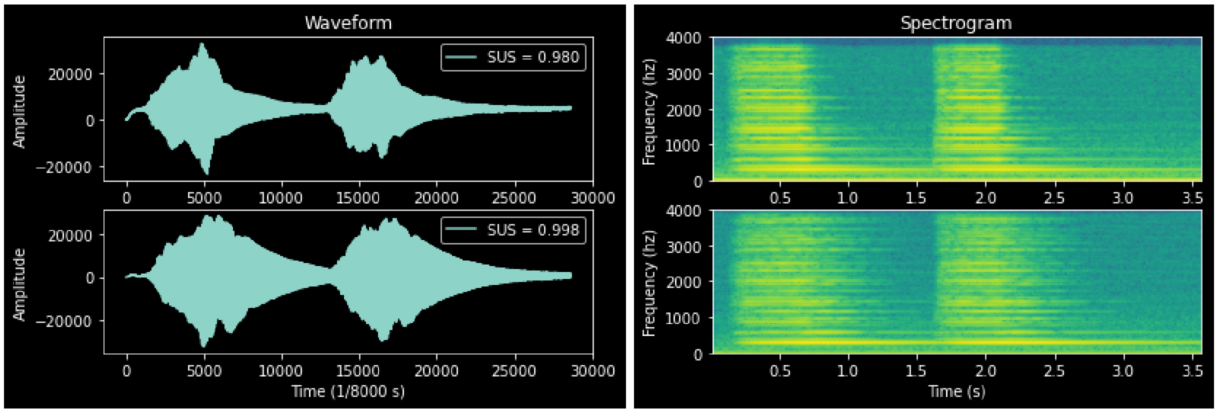


Figure 7. Waveform and spectrogram data from the third render test. The top is the rendered output with Sustain = 0.980 and the bottom is the rendered output with Sustain = 0.998. Surface Tension = 50.0

The next experiment was to add a sound impermeable barrier in the space and see how this affected the output. The speaker was on one side while the microphone was on the other. The setup and results are shown in Figures 8 and 9.

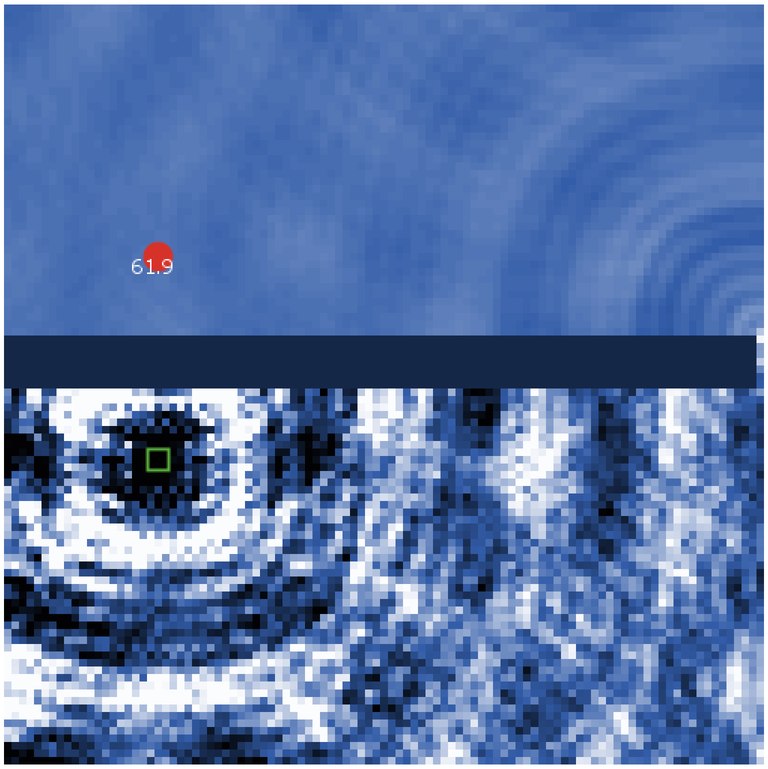


Figure 8. Fourth render test. Dark blue "sound-proof" wall in the center with a small gap on the right side. Surface Tension = 50.0 Sustain = 0.991

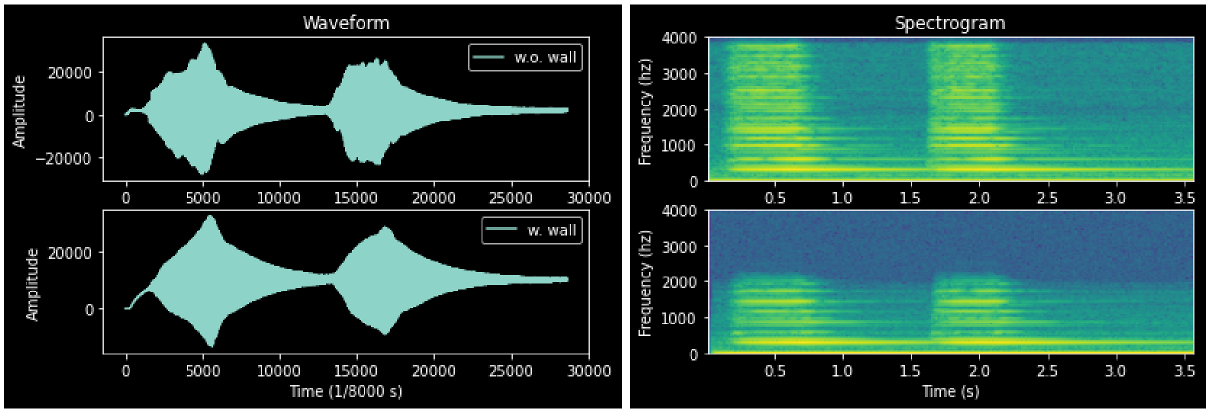


Figure 9. Waveform and spectrogram data from the fourth render test. On top is the rendered output without the wall while on bottom is the rendered output with the wall. Surface Tension = 50.0 Sustain = 0.991

These results clearly show that the space did have an impact on the rendered sound. The attenuation of the higher frequencies, shown in figure 9, gave the effect of a muffled sound which is what we had expected for the scenario where sound is only coming through a small gap in a sound-proof wall.

To increase our mapping to real space, it was useful to get an idea of the physical size of the space we are modeling. We measured that it takes a wave 60 ticks to move a distance of 40 patches. Each tick is one eight thousandth of a second, defined by the sample rate of the input clip. Then, using the speed of sound, we calculated the physical distance traveled. Dividing this by the number of patches gave us a good idea of how big the modeled space really is. With Surface Tension[5] set to 50 and Sustain at 0.991, we calculated that each patch is approximately 6.4 centimeters squared, making the 100 by 100 plane about 6.4 meters squared (the size of a small stage). This information could be useful in determining the placement of musicians in the space when rendering a larger performance.

This concludes our research into simulation-based Spatial Audio Rendering. The result is a logical and hands-on approach to rendering that allows an audio engineer to visualize sound and how it changes with space.

[5] It is important to note that the Surface Tension variable changes how fast the waves appear to move (in patches per tick) and so a larger Surface Tension will result in a larger modeled space. Sustain, however, has a negligible effect on the "speed" of the waves.

5. Sonic Proximity System

a. Background

In addition to Spatial Audio Rendering, we also looked at using the Wave-Box to prototype a Sonic Proximity System. Our initial ideas for this system came from a public health question of how to help individuals maintain 6-foot social distancing during COVID-19.

When it comes to measuring distance and position, the most common method is to use GPS[6]. However, this system has several limitations that make it ineffective in this application. First, the accuracy of this system has been calculated to be about ±3 meters which is larger than the 2 meters (~6 feet) distance that is going to be measured. Secondly, GPS signal reception is easily interrupted by overhead obstacles, which make its functionality indoors or in big cities with tall buildings unreliable. Thirdly, the system is unable to take into account physical barriers separating people which would be an important feature for enforcing social distancing.

A Sonic Proximity System (SPS) could solve these problems. We envisioned SPS to be a short-range, private, serverless way to compute relative positions using ultrasonic[7] sound waves and smartphones. SPS would operate by measuring the time it takes for an ultrasonic chirp to move between two phones and then calculate the distance by multiplying the measured time by the speed of sound. Even with just millisecond timing resolution (which we believed would be easily achieved on a smartphone), the system would have an accuracy of about 34 cm, nearly a 9 times increase in accuracy compared to GPS. In addition to this, SPS could work indoors and could prevent false proximity alerts when there is a physical barrier separating two individuals.

[6] Global Positioning System

[7] By using ultrasonic frequencies of sound above the human threshold of hearing, the system can operate without causing a disturbance.

b. Development

To begin our exploration of this interesting application of sound to SPS, we first looked at the practical limits of such a system. After some initial research, we came up with the following approach. A phone will start by sending out a ping and record the time the ping was sent. A second phone will be listening for the ping and when it receives/"hears" it, it will send a return ping. When the first phone receives the return ping, it will record the time again. Using these two measurements, the first phone can calculate the distance between the phones. This proposal was ideal because it removed the need for synchronizing the clocks on all of the phones, something that the earlier designs were limited by.

The second feasibility test we conducted was that of ultrasonic sound range. To test this, we analyzed volume drop-off for different ultrasonic frequencies. It is well known that higher frequencies on any wave cannot penetrate mediums as easily as lower frequencies. Light, for example, can not pass through an opaque wall but radio waves can because they have a much lower frequency and are less likely to get scattered by the particles in the wall. The results of this test are displayed in figure 10.

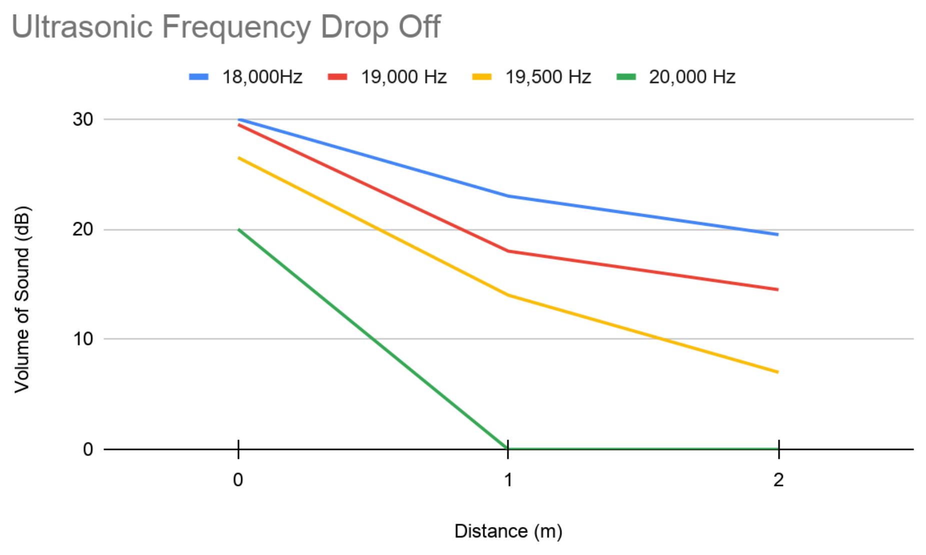


Figure 10. Graph made by playing ultrasonic frequencies in a microphone at increasing distances.

With this experiment, we found that using a ping with a frequency around 19,500 Hz would be optimal, as the microphone on a phone was able to pick up the ping over a distance of 2 meters while still being outside the typical human hearing threshold[8].

c. NetLogo Model

Using the NetLogo Wave-Box that we developed at the start of the project, the SPS could be visually modeled and logically analyzed. Figure 11 shows the progression of the system as the first phone sends out a ping and waits for its return.

[8] The upper threshold for infants and children can be upwards of 20khz but continues to decrease with age. For the average adult, the upper threshold is around 17khz. (Purves)

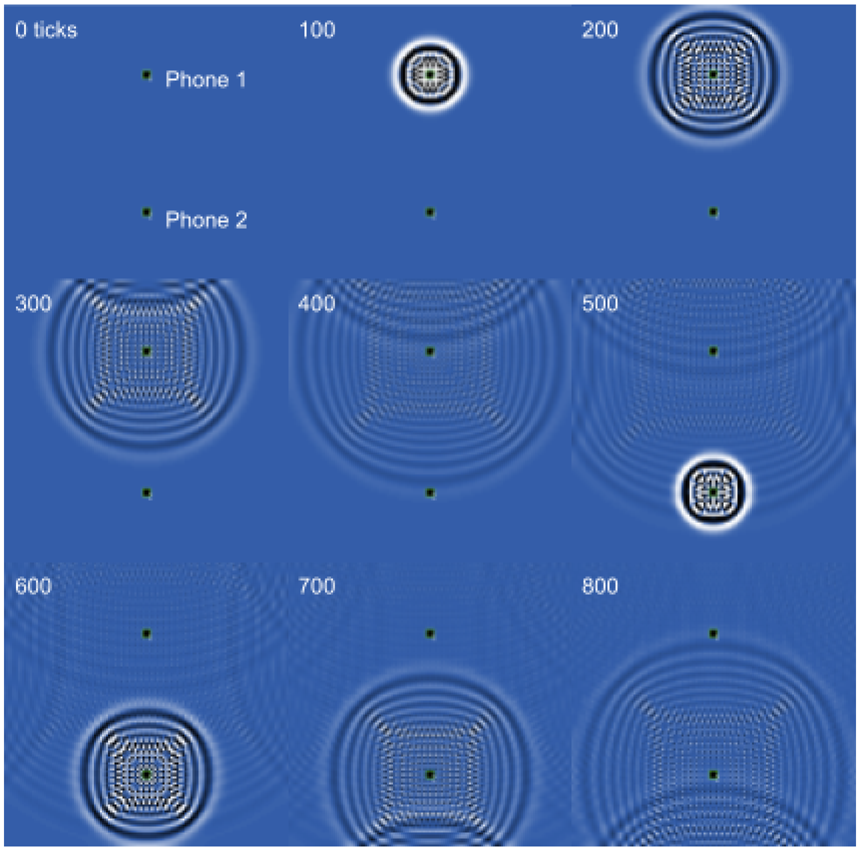


Figure 11. Progression of SPS model in NetLogo. The two black squares represent the two phones.

By measuring the number of ticks that it took for the waves to go between phones, the distance could be calculated. What was challenging about this process was finding ways for the phones to detect only the initial ping rather than the echoes and general noise in the system. In the full-blown SPS, we envisioned that each phone would transmit pings at slightly different frequencies in the ultrasonic range. This way, each phone could be "listening" for a specific frequency and background noise or echoes could not be mistaken for the ping. This led us to investigate the use of fast Fourier transforms (FFT) to decode the frequencies in a certain ping.

The principal behind a Fourier series is that any function can be written as an infinite sum of sine and cosine terms. The FFT is an algorithm optimized to quickly generate what is known as the Power Spectral Density (PSD) of the data which determines the coefficients of these sine and cosine terms. FFT operates by analyzing a chunk of time-domain data (such as the raw amplitude data coming from a microphone), and in our case, the outputs of the specific frequencies of sound that make up the data.

One thing we had to consider when working with FFT was the “bin size” of each analysis. With small bin sizes, you can get high accuracy in the time domain but it comes at the cost of low accuracy in the frequency domain. The reverse is true with larger bins. For this application, measuring frequency accurately was important but having high timing resolution was even more important. With bins 1 ms long and with a microphone sample rate of 48khz, we could theoretically achieve the necessary locational accuracy of ±34 cm while still managing to have enough data to determine the frequency. Figure 12 shows the FFT on a 1ms sample of a pure tone of 20khz.

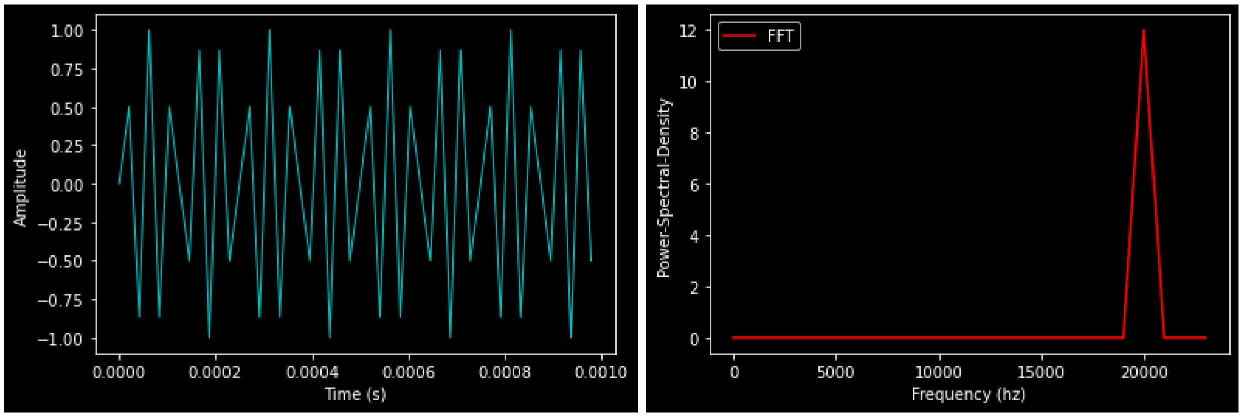


Figure 12. FFT Results. Left: 1ms of raw amplitude data of a 20khz pure tone. Right: PSD showing most prominent frequencies. With a larger bin, the spike at 20khz would become narrower (i.e. less uncertainty).

Even if we added background noise to our data, we saw that the FFT was still able to decipher the most prominent signal. This made the FFT a good solution for analyzing incoming signals and then processing out the raw frequencies to determine if a real ping is present amongst any background noise.

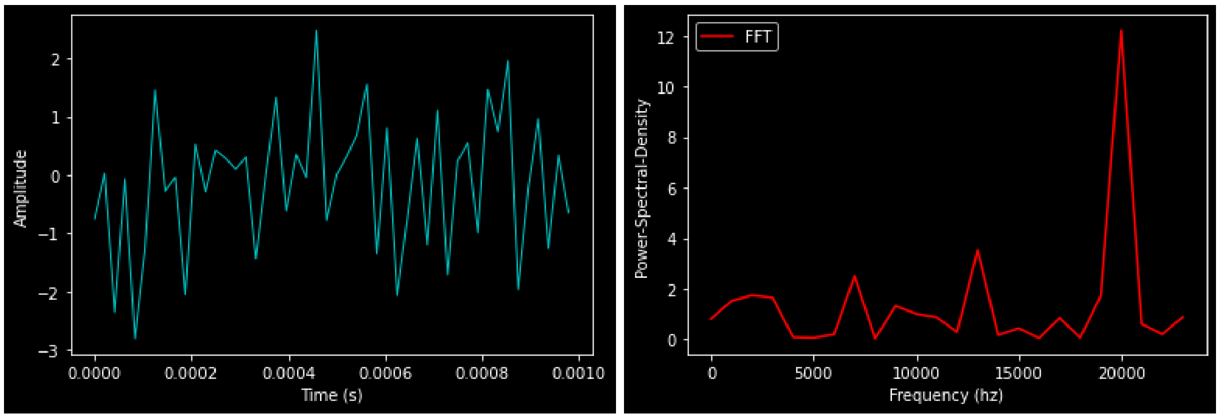


Figure 13. FFT of 20khz tone with Gaussian noise. Left: raw amplitude data. Right: PSD showing most prominent frequencies.

While it would be possible to implement this FFT analysis in NetLogo (again relying on the Python extension) to improve the robustness of our signal detection, it made more sense to implement this within JavaScript and take advantage of its Web Audio API which includes an FFT library.

d. JavaScript Model

To begin the development of the full-blown SPS in JavaScript, we first learned to work with the Web Audio API to detect and emit pings of various frequencies. Then, using some demo Web Audio API code, we implemented our approach to calculating distance using ping timing with the following equation:

$$d = c\frac{(t_2 - t_1) - \delta}{2}$$

where

- c is the speed of sound
- t_1 is the time, recorded by the first phone, when it sends the ping
- t_2 is the time, recorded by the first phone, when it receives the return ping.
- δ is the system calibration constant.

We expected that there would be some delay in the system, particularly when the second phone had been triggered to send a return ping, and predicted that this delay would be uniform across different trials. This led us to include a calibration constant. The long-term goal was that after the system started working reliably, the calibration constant could be calculated explicitly by equating the previous expression to the true distance and solving for δ as is shown in the following equation where d_{true} is the true distance between the two phones.

$$\delta = (t_2 - t_1) - \frac{2d_{\text{true}}}{c}$$

While we were able to make some progress in building this system, we experienced significant obstacles. In the Web Audio API, we ran into the limitation that although the microphone of the device can sample at up to 48khz, it appeared this raw data was only accessible in JS via the API at intervals of 60hz. With only 60hz (17 ms) timing resolution, the accuracy of the system drops to ±6m. These results are displayed in figure 14.

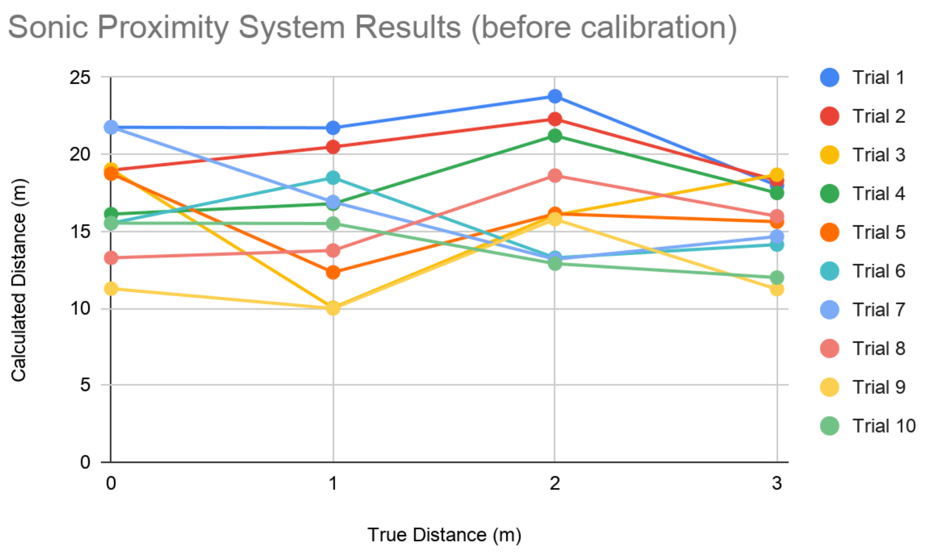


Figure 14. Comparison of the True Distance vs. the Distance Calculated by SPS over 10 trials. Although this is before the calibration constant was evaluated, it is clear that even with it, there will still be an unacceptable level of inconsistency. The calibration would simply translate the data points down to the x-axis. Notice how the range of the data at each distance is consistent with an accuracy of ±6m.

These limitations on timing resolution meant that the system did not meet the necessary accuracy to be useful in a social distancing application. With these results, this branch of development came to a close.

e. Results

While it would be possible to create this system with a more extensive knowledge and utilization of the JS libraries and Web Audio API, this application of sound waves was diverging too far from the functionality of the Wave-Box. Upon further research, we became familiar with the NOVID app which also uses ultrasonic approaches to track viral contact (Payne, 2020). This gives credence to the idea that our concept of SPS could be achieved with the right tools.

Development of SPS, while incomplete, proved to be an excellent learning experience. We were able to use the NetLogo Wave-Box to experiment with the logic behind SPS. We discovered the importance of signal processing which led us to learn about FFT and its uses. Furthermore, we gained valuable exposure by working with JavaScript and the Web Audio API.

6. Conclusion

a. Takeaways

In the end, we were able to develop the Wave-Box simulation into a fully functioning 2D model that uses the shallow water wave equation to simulate how sound behaves in acoustic spaces. This allowed us to use the Wave-Box to perform Spatial Audio Rendering. The rendered outputs closely matched the hypothesized outputs for many of our tested scenarios. Furthermore, the resulting acoustic effects resemble many of the effects commonly available in a DAW. We were also able to explore the effects of different obstacles on a sound.

Additionally, we used the Wave-Box to visualize the functionality of a Sonic Proximity System while also making an attempt at building SPS in JavaScript.

b. Limitations and Errors

One of the large sources of error in this project came from the use of shallow water equations to model sound in air. Despite our best efforts to tune our model to be representative of sound, water waves and sound waves are fundamentally different. This means that there were certain limitations that arose when we modeled them as if they were the same.

Although the use of the Wave-Box for rendering proved to be effective, what remains unclear is why the model performed so well using the alternate Surface Tension value rather than the one identified at the start of this project.

Unfortunately, we were unable to develop SPS to its full potential which was the result of technological limitations of processing speed in the Web Audio API.

c. Future Work

The Wave-Box could be improved by adding a third dimension in order to get a realistic idea of sound waves in a 3D space rather than in just a plane. Although this would be easy to code, it would drastically increase computational time and certain optimizations would be needed to make this feasible.

To continue research in Spatial Audio Rendering using only simulation, an interesting addition would be to incorporate different materials with different levels of sound absorption, reflection, and permeability. For example, sound hitting a carpeted obstacle might bounce back or reverberate less than a wooden one. This would open up a range of possibilities for spatial composition and rendering. For instance, we could build up a more detailed modeled space in the NetLogo environment such as an auditorium, and see how the sound changes. Additionally, more sound inputs could be added at different locations in a space representing the different members of an orchestra, band, or choir spread out on a stage. By adding a second microphone offset slightly from the first, a stereo output could be generated to further increase the immersiveness of the sound.

The next steps to be taken to make SPS a reality would be to find a way to bypass the Web Audio API data recall limit in order to arrive at the desired timing resolution in the system. The system could then be scaled up to use more devices, and eventually, absolute position could be calculated using a trilateration[9] algorithm.

[9] Trilateration is the same method that the GPS system uses to calculate absolute position. It works by finding a solution to a set of equations defining 4 or more spheres of known radii. Each GPS satellite is located at the center of a sphere and the position of the object in question lies at the intersection of the spheres. In the case of SPS, the spheres would be centered around each phone and the radii would be calculated using the distance between any two phones using the method described above (Geography).

7. Acknowledgments

We would like to express our gratitude towards Stephen Guerin and Mohit Dubey. They both have helped us immensely with consolidating and clarifying our project, teaching us about shallow water equations, helping us debug our code, and providing guidance and support when our project became unfocused. We would also like to thank the organizers of the 2021 Supercomputing Challenge for nurturing our curiosity in STEM and providing us with an amazing opportunity to learn about computer science!

8. References

Brunton, Steve. “Denoising Data with FFT [Python].” *YouTube*, YouTube, 7 Apr. 2020, www.youtube.com/watch?v=s2K1JfNR7Sc&t=123s.

Geography, GIS. “How GPS Receivers Work - Trilateration vs Triangulation.” *GIS Geography*, 27 Feb. 2021, gisgeography.com/trilateration-triangulation-gps/.

Kleusberg, Alfred, and Richard B. Langley. *The Limitation of GPS*. GPS World, gauss.gge.unb.ca/gpsworld/EarlyInnovationColumns/Innov.1990.03-04.pdf.

MusicTech. “Ableton Live Tutorial - Reverb and Convolution Reverb.” *YouTube*, 6 May 2020, youtu.be/sK11zQKLmus.

Payne, Emily. *NOVID Is the Most Accurate App for Contact Tracing - News - Carnegie Mellon University*, Carnegie Mellon University, 30 June 2020, www.cmu.edu/news/stories/archives/2020/june/novid-update.html.

Purves, Dale. “The Audible Spectrum.” *Neuroscience. 2nd Edition.*, U.S. National Library of Medicine, 1 Jan. 1970, www.ncbi.nlm.nih.gov/books/NBK10924/#:~:text=Humans%20can%20detect%20soun s%20in,to%2015%E2%80%9317%20kHz.).

Rypuła, R. (2021). Robertrypula/AudioNetwork [JavaScript]. https://github.com/robertrypula/AudioNetwork (Original work published 2016)

“Shallow Water Equations.” *Wikipedia*, Wikimedia Foundation, 14 Jan. 2021, en.wikipedia.org/wiki/Shallow_water_equations.

Wilson, Chris. “Live Web Audio Input Enabled! | Google Developers.” *Google*, Google, developers.google.com/web/updates/2012/09/Live-Web-Audio-Input-Enabled.