Sound Propagation Simulation in Video Game

SeoKyeong Toby Kim

Maynooth University

Sound Propagation Simulation in Video Game

Abstract In this research, I aspire to find both realistic and efficient ways to achieve spatial audio, and this research is an ongoing effort to bridge this gap. In this manner, I am developing and testing methods that aim to balance realism with computational efficiency in spatial audio simulation. My immediate focus is on enhancing the accuracy of sound localization in complex environments, while also considering the computational constraints in a game environment. By investigating algorithms and refining existing techniques, this research seeks to provide a more comprehensive solution for dynamic sound propagation in interactive virtual environments. As this project progresses, I hope to uncover new insights and advance the field of audio simulation in video games.

1.1 Introduction

3D audio simulation in computer is well studied and sounds realistic in many use cases like audio plugins or video game integration. Technologies like Ambisonics and Dolby-Atmos have been widely integrated into video games as well as game engines like Unity and Unreal Engine's 3D audio libraries themselves. However, simulating sound propagation in video games, especially in the presence of obstacles can still be very CPU intensive and often neglected in development because it is seen less significant than graphics. People often use various strategies to optimize the process. For example, if there's a wall between audio source and a listener, we can simply use ray-tracing to check if there's any obstacle in between, and then apply low-pass filter to simulate occlusion. While it could sound realistic enough, for games where directivity of sound is crucial, the lack of localization of sound could be a problem since in reality the sound would travel around

the wall, sound as if it is coming from different direction. Moreover, we need to take diffusion into account as the sound might have travelled such a long way by bouncing off the walls many times. It would affect the quality of the sound. For that reason, we can use multiple recursive rays until it finds the listener. And depends on the number of possibilities of rays' reaching the listener, we can apply reverb to simulate diffusion, but the problem is that the number of rays could be too high to even compute. And it's hard to decide how deep one ray's recursion should be.

1.2 Directivity

First, I thought of ways to get the direction of which the sound is coming from. As I mentioned above, if we were to use ray-tracing, we would not know how many rays we must use to get the optimal result. As you can see in **Fig.1,** the rays cannot cover the whole screen.

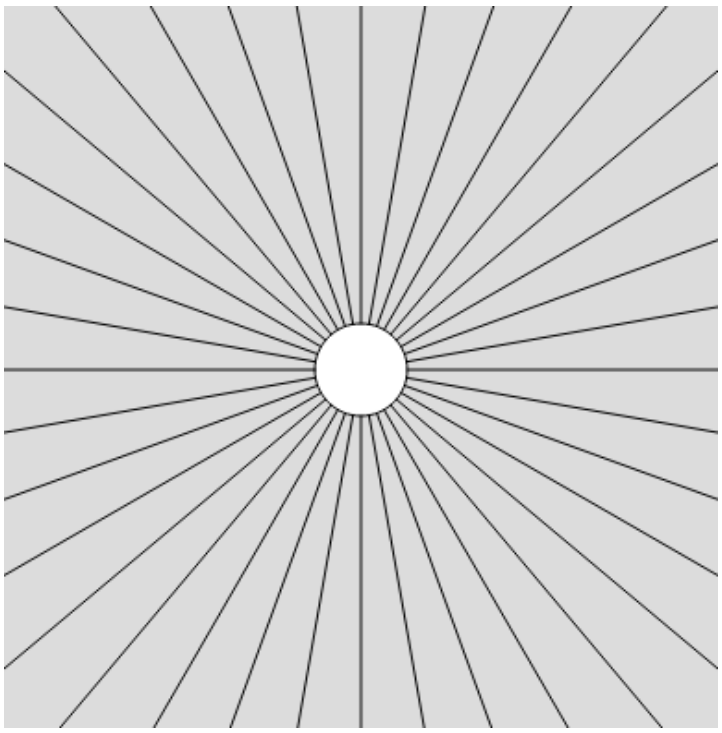


Figure 1. Sound source is emitting rays to find listener.

As might be expected, we can make it denser by increasing the number of rays, yet still if the distance between audio source and the listener is far away from each other, there are chances that rays cannot find the listener and same goes with the reflections of the rays.

1.2.1 Wave Equation

As sound propagates all 360 degrees, I thought I could possibly use water ripple effect to simulate air molecule behavior because fluids and gases are similar in the context of wave propagation, as both can transmit energy through processes of compression and rarefaction. I found wave equation[1] that I could use, and **Fig.2** shows how propagation works with obstacles in 1D (imagine wave bounces off the wall at both left and right).

The Equation can be discretized like this.

$$\delta_i^{n+1} = 2\delta_i^n - \delta_i^{n-1} + \left[c\frac{\Delta t}{\Delta x}\right]^2 (\delta_{i-1}^n - 2\delta_i^n + \delta_{i+1}^n)$$

Figure 2. Simple 1D wave Equation Visual (Moving image attached)

[1] 'Lab12_1: Wave Equation 1D', Haroon Sahotra, uploaded 22 April 2016. 'Lecture video on wave Equation' < https://www.youtube.com/watch?v=MqlDQzw3x6Q&t=4s> [accessed 1 October 2023].

Where ‘n’ represents coordinates in time and ‘i’ represents coordinates in space.

And for 2D (the way I implemented), It can be like this.

$$\delta_{ij}^{n+1} = 2\delta_{ij}^{n} - \delta_{ij}^{n-1} + \left[c\frac{\Delta t}{\Delta x}\right]^2 \left(\delta_{i-1j}^{n} - 4\delta_{i}^{n} + \delta_{i+1j}^{n} + \delta_{ij+1}^{n} + \delta_{ij-1}^{n}\right)$$

Where ‘i’ represents x-coordinates and ‘y’ represents y-coordinates in 2D space. And by making current amplitude of the edges fixed to 0, it can flip the polarity and bounce off the wall.

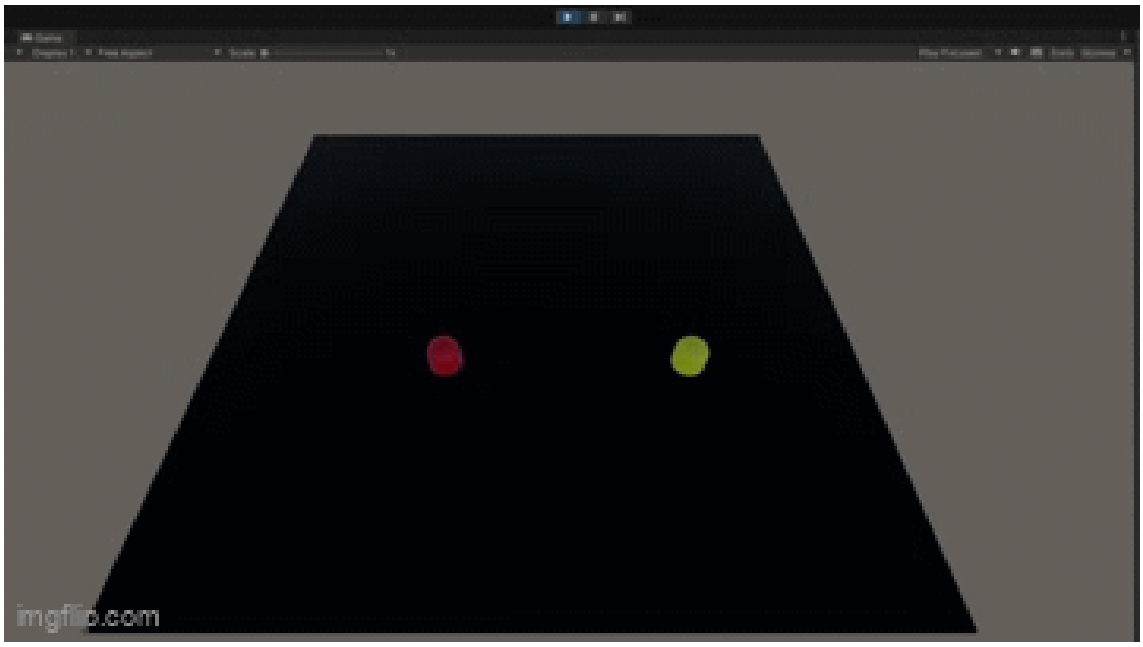


Figure 3. Unity implementation with wave equation with compute shader(Moving image attached)

Now, I can cover all 360 degrees without any gaps, and I don’t need to worry about recursive computation required for reflection from the ray-tracing method. In **Fig.3,** I am using Unity’s compute shader for computation. The way it works is that I set a certain resolution for the computation (in this case 512 * 512), and I calculate each pixel’s amplitude by the wave equation

one by one. The red thing from **Fig.3** is the sound source and the yellow one is the listener. Since it runs on GPU side, the speed of calculation is well optimized.

Figure 4 Using texture as a record for meaningful information for DSP.

Now, I still need to obtain data for sound propagation. The communication between CPU and GPU can slow down the process as CPU has to read through all the pixel data. So, instead of reading through all the pixels every frame, I let CPU to send current time information to GPU every frame. And then From GPU side, I let it store meaningful information as a texture for 10 seconds (**Fig.4**) so that CPU can read the whole pixels only once. I checked amplitude of listener position coordinate at every frame. And when it's not nearly 0, I calculated the direction of which the sound is coming from by adding all the surrounding velocity and normalized it. With that, I could get all instances of reverberation of the sound with their direction (**Fig.5**).

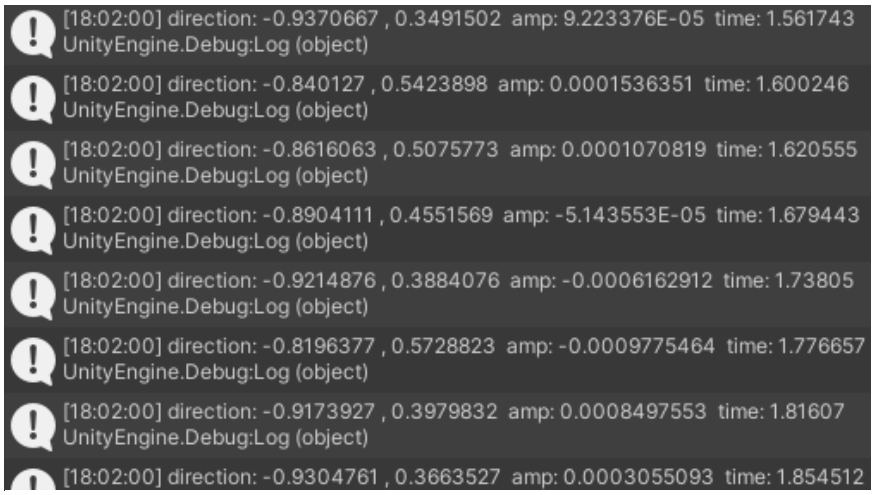


Figure 5. Record achieved for DSP. On average, I get 150 instances.

Ideal way to simulate this would be making sounds of all those instances. If I didn't have to do spatialization for each voice, it would be doable. However, the computational cost is too high. Instead, I picked one voice with the loudest amplitude.

1.2.2 Interaural Timing Difference

The first DSP process is to do ITD (interaural Timing Difference). If you think about our head (**Fig.6**), when the sound source is at one's right, there must be a slight time difference between

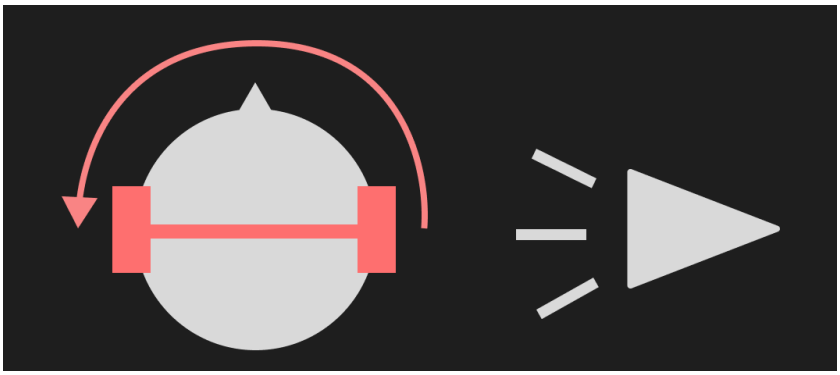


Figure 6. Timing Difference that occurs due to the position of the sound source.

the left ear and the right ear. Albeit subtle, our brain is able to recognize it. Even with the same amplitude, it would sound like the sound is coming from one direction with this slight time difference. The equation for ITD works like this. I used my normalized vector with this equation,

$$ITD = \frac{r(\theta + sin\theta)}{c}$$

To get the theta,

$$atan2(y,x)$$

$$= \begin{cases} arctan(\frac{y}{x}) & if\ x > 0 \\ arctan(\frac{y}{x}) + \pi & if\ x < 0\ and\ y \geq 0 \\ arctan(\frac{y}{x}) - \pi & if\ x < 0\ and\ y < 0 \end{cases} \begin{cases} +\frac{\pi}{2} & if\ x = 0\ and\ y > 0 \\ -\frac{\pi}{2} & if\ x = 0\ and\ y < 0 \\ undefined & if\ x = 0\ and\ y = 0 \end{cases}$$

but I had to clamp my radian in between 0 and half pi. Because when the theta becomes larger than the half pi, this would act like the sound would travel all the way through the left side as shown in the **Fig.7.** So, I made it increase until it gets and half pi and then start to decrease until it gets 0 again by subtract the difference between current theta to half pi when the current theta becomes greater than half pi, same thing for the other side but with flipped polarity. As you can hear with **Aud.1,** even only with timing difference, you can sense the position of the sound.

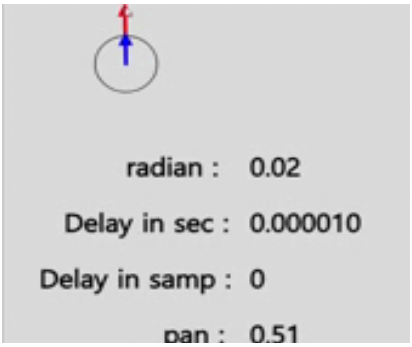


Figure 7. Radian Manipulation
(Moving image attached)

1.2.3 Inter-Channel Level Difference Panning

Second, I implemented Inter-Channel Level Difference panning. I also modified the amplitude of each audio channel based on the sound's direction. This adjustment in loudness between channels helps to more accurately pinpoint the sound's location.

Since dB is,

$$dB = 20\log_{10}(\frac{A2}{A1})$$

$$\frac{A2}{A1} = 10^{\frac{dB}{20}}$$

I linearly interpreted the clamped radian from the normalized vector from -1 to 1. Polarity means the direction and the absolute value means the difference in amplitude. And I interpreted this absolute value from 0dB to 12dB. When the sound source is at your left ear, it decreases the right channel by 12dB. So, when the pan value is greater than 0, I apply,

$$Left\ Channel = Left\ Channel * 1$$

$$Right\ Channel = Right\ Channel * 10^{\frac{-dB}{20}}$$

And when the pan value is less than 0, I apply,

$$Right\ Channel = Right\ Channel * 1$$

$$Left\ Channel = Left\ Channel * 10^{\frac{-dB}{20}}$$

You can listen to this stage of process with **Aud.2.**

1.2.4 Pinnae Shadowing

Third, I also implemented pinnae shadowing effect by applying two instances of low-pass filter at both left and right channels. When the sound travels from your back, your ear pinnae (rear helix) would act like a barrier shadowing high frequency above 10kHz. Since current model only can differentiate between left and right, I wanted it to be able to simulate sound coming from back and front as well. For this, I disregard directivity in left and right. I only took front and back into account. So, I needed to get the unclamped radian from the normalized vector. When the absolute value of it becomes greater than half pi, it starts to be at the back of our head. And then I linearly interpret this value from 4.0 to 4.3 as 10^4 is 10k and 10^4.3 is roughly about 20k. And use this value as a cut-off frequency for the filter. The spectrum analyzer of this is following (**Fig.8**).

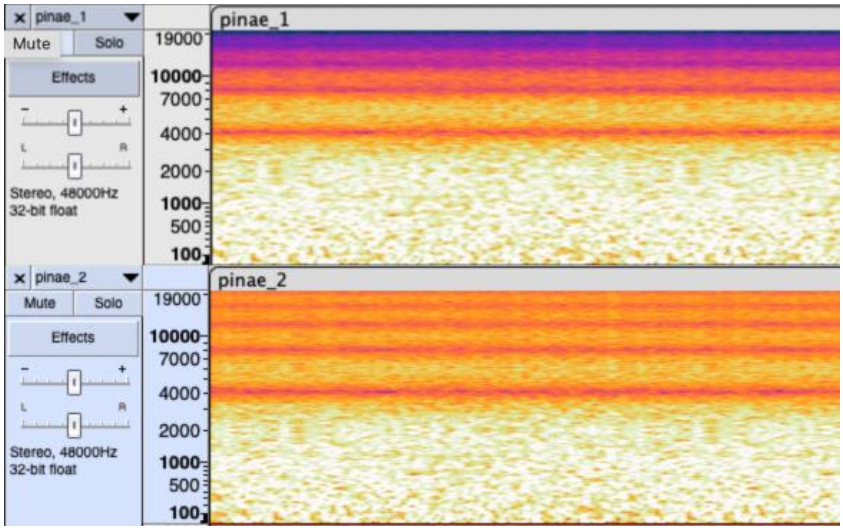


Figure 8. Pinnae Shadowing Simulation Spectrogram

You can listen to this stage of process with **Aud.3**.

1.3 Distance

1.3.1 Amplitude Adjustment

Now, I'd like to introduce distance to the sound. There can be three ways to simulate distance in sound. One way is to adjust the amplitude of the sound. We do have amplitude from

the texture record. We can certainly use this. However, the model we used is not ideal representation of sound propagation. What I think is the more meaningful value in this context is time that has passed since the burst of sound. Though it captures propagation for 10 seconds for visual effect, sound propagate much faster than that in real life. So, imagine our 10 seconds are actually 0.1 second. That means if it took 1 second to get the loudest instance of reverberation, the distance between sound source and the listener is.

$$Distance = 343m/s * 0.1s$$

$$Distance = 3.43\,m$$

Then we can get the amplitude of the sound more reliably

$$Intensity\ factor = 1 / Distance^2$$

$$Amplitude\ factor = 1 / Distance$$

$$Sample = Sample * Amplitude\ factor$$

When Distance is greater than 1. If not, I consider that the Distance is 1.

1.3.2 Amplitude Adjustment per frequency band

I should also apply another low-pass filter too as higher frequencies tend to be absorbed more quickly in the air than the lower frequencies. However, I begun to question if another low-pass filter is the right way to simulate this behavior. According to 'Absorption and Attenuation of Sound in Air'[2] by Dr. Daniel A. Russell from the Pennsylvania State University, there are two main

[2] Daniel A. Russell, 'Absorption and Attenuation of Sound in Air', The Pennsylvania State University, 2 December 2013 <https://www.acs.psu.edu/drussell/Demos/Absorption/Absorption.html> [accessed 1 January 2024].

reasons why sound energy is absorbed by air. First, classical absorption. As molecules like nitrogen, oxygen, argon, carbon dioxide collides with each other propagating the energy, it results in the viscous losses. This thermal-viscous classical absorption has two main factors, the square root of temperature and the square root of the frequency. Second, relaxation processes. As both nitrogen and oxygen is the two primary molecules that make up 99% of air, this article focused on kinetic energy absorption that occurs when some of the energy carried by the sound wave causes these molecules to vibrate and rotate. Using these factors, the researchers provided the absorption calculator I can you. As you can see at the calculator, frequency response of the attenuation in decibel is different. So, I used this calculator with my FFT (Fast Fourier transform) to simulate sound absorption through the air. FFT is an algorithm that converts time-amplitude domain data into amplitude-phase domain so that we can see the representation in the frequency domain as well as edit it. As this project's sampling rate is 48KHz and the FFT table size is 1024, the frequency resolution of each bin is this.

$$i * \frac{48{,}000\ (Sampling\ Rate)}{1024\ (FFT\ table\ size)}$$

When $1 \leq i \leq 1024$.

To be more specific, since the second half of the FFT table is mirror image of its first half due to the Hermitian symmetry, I ignored 'i' that is greater than 512 for the efficiency (also it is above Nyquist Frequency). Then I plugged each bin's frequency into the calculator to get the attenuation values at 100m. I linearly interpreted the distance that it took for the sound to reach the listener and then used that to adjust the amplitude of each bin. Before I could turn it back to time-amplitude domain, I have to adjust phase to avoid abrupt changes in phase since changes in each bin's amplitude can result phase shift. To do that, I stored last phase information for both input & output

from every bin. And then calculated the next phase increment depends on the center frequency of each bin. Then change both real and imaginary number like this,

$$Complex(x + yi) = Amplitude * Cos(newPhase) + Amplitude * Sin(newPhase)i$$

And made the conjugate of the first half at second half of the bins because It's crucial to maintain the conjugate symmetry in the FFT bins, especially after modifying them, to ensure the IFFT results in time-amplitude domain. Then I applied IFFT (Inverse Fast Fourier Transform) to turn the frequency domain representation into the time-amplitude domain again for playback.

1.3.3 Reverberation (Diffusion)

Then I applied reverb depends on the time it took to get to the listener and the number of reverberations it captured within 10 seconds. As I mentioned above, the ideal way would be to process all the instances but now you can see how intense the computation would be to simulate the whole process that I have covered so far for every voices. Instead, I used Schroeder Reverb to simulate reverb and changed its parameter depending on the data that I retrieved. Schroeder Reverb, discovered by Manfred Schroeder in the early 1960s, has four serial all-pass filters and four or more parallel comb-filters (delay lines with feedback) of unique length of delay times. Usual Schroeder reverb model would process four comb-filters first but the specific implementation[3] that I found from Stanford University CCRMA (Center for Computer Research in Music and Acoustics) article (called JCREV developed by Prof. John Chowning, founding director of CCRMA) had four all-pass filter first and then applied four comb-filters later and I followed that model. For the all-pass filter, I used first order IIR all-pass filter which contains two comb-filters (feedforward and

[3] Julius O. Smith III, `Physical Audio Signal Processing', 2010 <https://ccrma.stanford.edu/~jos/pasp/Schroeder_Reverberators.html> [accessed 1 January 2024].

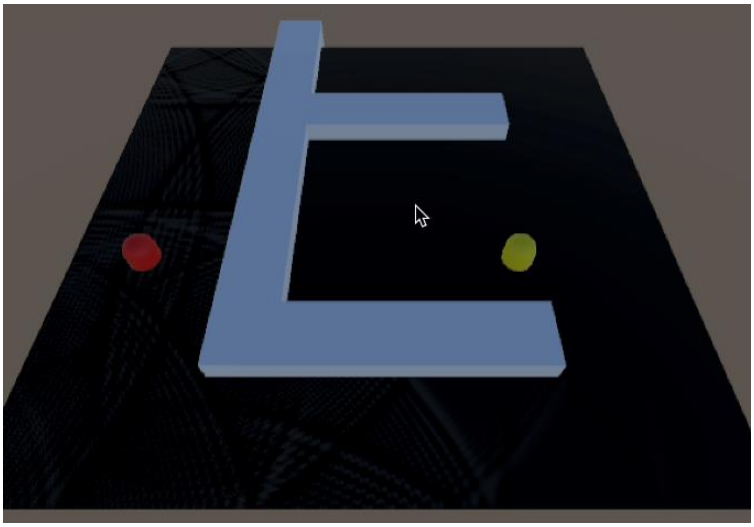
feedback) in it. For the delay time coefficients for each filter, I refer to diagram from CCRMA article with a slight change. And for the feedback coefficients, I made it larger than the article because I wanted to make it large by default so that I can attenuate them depending on the number of reverberation instances. As these coefficients set the tone of the place, it could later be variable depends on a place, material of virtual walls and architectures. And for the 'Wet' parameter for the reverb is affected by the time that it took for the sound to reach the listener. As if it is a free-field and the sound is far away from the listener, the sound still gets diffused through the air and since the number of reflections is not high, there wouldn't be high feedback. On the other hand, if it is closed environment with immediate contact between audio source and the listener, the feedback will be higher, but the wet parameter would be small so that not much reverb is audible. Only when it's closed environment and the source and the listener is far away from each other, reverb would sound significantly.

1.4 Test Cases (Video Attached)

1.4.1 At Left, No Obstacle

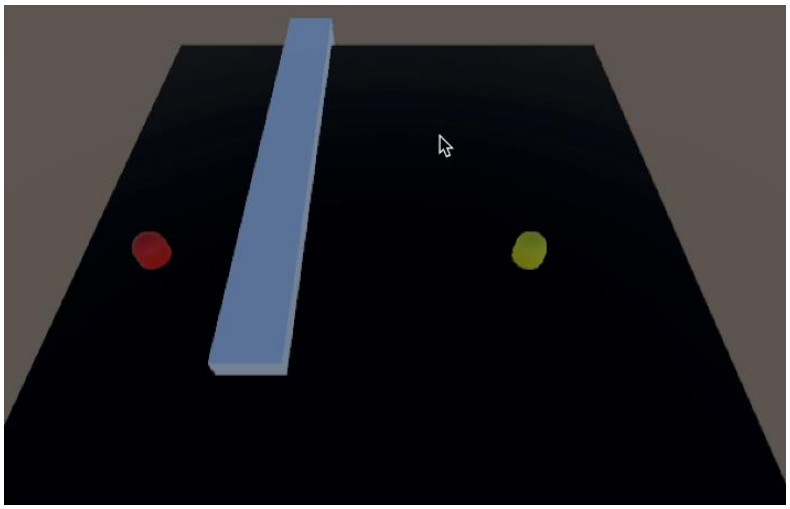
Since there is no obstacle, the distance it took to get to the listener is medium therefore a little bit of reverb is audible. Pan towards left side. The difference between left amplitude and the right amplitude is about 12dB as left amplitude is 0 and right amplitude is 0.25.

1.4.2 At Left, With Obstacle



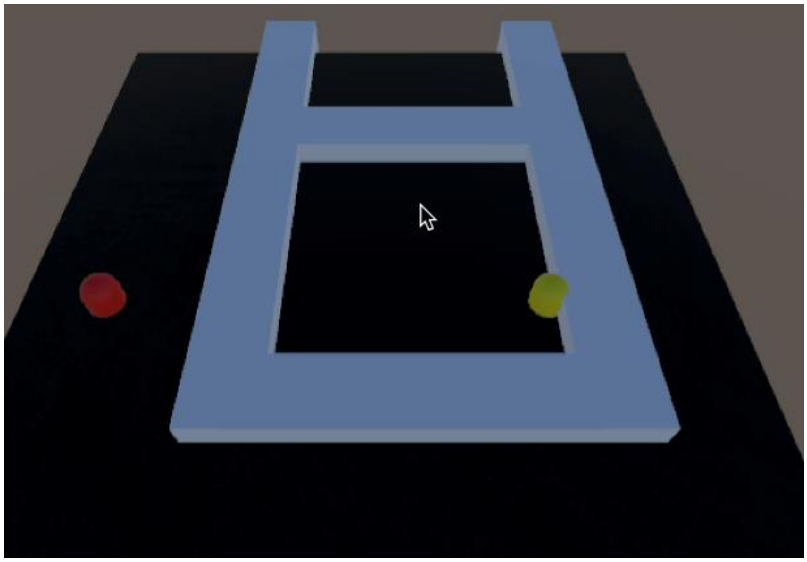
Since the sound would have to travel all the way through the bottom, the distance it took to reach the listener was longer therefore a lot of reverbs is audible and the amplitude is low. The sound reaches the listener from its right side, and we can hear that. However, unlike white noise, the human voice does not have a lot of energy at every frequency so, the attenuation depends on the frequency band is not prominent.

1.4.2 At Left, With Obstacle 2



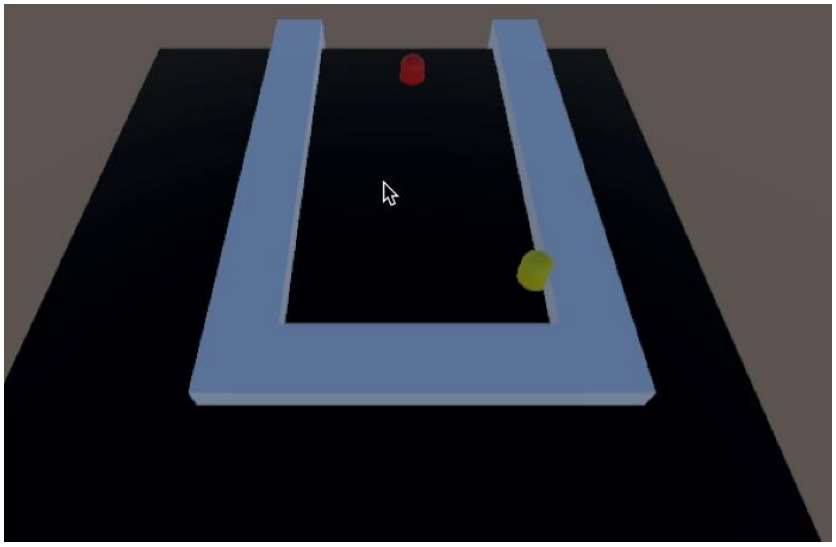
This time, the obstacle only makes the sound to reach the listener from the left back. We can hear the sound is coming from the left side. And then the attenuations in amplitude and a lot of diffusion from the reverb as well.

1.4.3 At Left, With Obstacle 3



This time, the sound is not able to reach the listener, therefore it prints “no entries has been made.”

1.4.4 At Front left, With Obstacle (closed)



Since it is a more closed environment, it captured more reverberation (although due to the size of the environment the difference is not significant). Therefore, more feedback is audible. As the sound is coming from the front, higher frequency is more boosted resulting in clearer sound (not dull).

1.5 Conclusion

This research project aims at finding both realistic and efficient ways to achieve spatial audio in a complex environment. By delving into known acoustics knowledge, algorithms, and techniques, I have tried formalizing a set of system that could simulate the sound propagation in a complex digital environment. I have discovered a way of working with GPU to enhance the speed of calculation for sound propagation. Despite its significant limitations such as latency (as it has to wait for 10 seconds to process data), it enlightens the possibility of avoiding bottlenecks for complex audio propagation process. The system could cover broad range of aspects of acoustics in real life as well as some DSP techniques like filtering, reverb and FFT. As some of the methods could be more tested out and refined, for example, the coefficients in reverb and how it would

react to the reverberation instances, there remains more works to be done. However, it is evident that I have made significant understand in acoustics and DSP. This research serves a role as a steppingstone, highlighting need for more investigation.