Research Project

Adaptive Reverberation in Unity Games

Albert Madrenys

Acoustics & Psychoacoustics 2023-2024

Index

1. Introduction

In the world of video games, sound design often takes a back seat. Sound design is a very important aspect, but it doesn't always need to be hyper-realistic. This is a practical choice given that video games demand substantial computational power, forcing developers to simplify various systems to conserve resources, even if it means sacrificing some realism. In most cases, a cheap acoustic approach is enough to achieve the desired results.

Because of these constraints, a lot of game engines don't usually incorporate sophisticated sound engines. This leads audio programmers or game developers that are creating videogames that require a more realistic approach, to rely on external tools.

In video games, reverb filters typically rely on developers to freely adjust their values, allowing for flexibility. However, having a tool that enables an audio source to dynamically modify its filters, like the reverb filter, by scanning and analysing its surroundings would be highly beneficial. This approach would facilitate smooth, real-time adjustments of parameters for an audio source moving across various acoustic settings, eliminating the need for developers to fine-tune every possible outcome.

This research aims to bridge the gap between realistic audio reverb and real-time performance within Unity Engine, offering game developers a great tool to enhance the immersive auditory experience in their creations.

The tool's functionality revolves around an audio source in an environment, analysing its surroundings and dynamically adjusting reverberation parameters. To accomplish this analysis, a ray tracing approach will be employed. While numerous plugins exist to enhance Unity's basic reverb model, none leverage a Raycast approach.

2. Raytracing in audio

Ray tracing is a rendering technique used in computer graphics to simulate the way light interacts with objects in a scene. It works by tracing the path of light rays as they travel through a 3D scene. These rays are traced from the eye of the viewer (or the camera) and are followed as they bounce off objects, interact with surfaces, and eventually reach a light source or contribute to the final image.

Each ray, as it travels through the scene, may undergo reflection, refraction, or absorption based on the properties of the objects it encounters. This simulation of light behaviour helps create highly realistic and detailed images with accurate lighting, shadows, reflections, and other visual effects.

While historically used predominantly for graphic rendering, ray tracing's inherent capabilities extend well to simulating audio. Over the past five years, various approaches have emerged, exploring different aspects of acoustics, including propagation and absorption.

Although conceptually intuitive, ray tracing is extremely computationally expensive. To grasp its computational intensity, it's noteworthy that Albrecht Dürer proposed ray tracing as early as 1525[1], but only in recent years have we managed to recreate it in real-time on computers, thanks to the increased computational power at our disposal. As such, this research project will not only focus on a ray tracing approach to

[1] Hofmann, G.R. ‘Who invented ray tracing?’ The Visual Computer 6, 120–124 (1990). <https://doi.org/10.1007/BF01911003> [accessed 19 October 2023], p. 1.

the reverb effect but also on optimizing the process to ensure efficiency and simplifying the model to achieve real-time results.

3. Unity engine and Raycast

Unity is a very popular and user-friendly game engine, designed for small to medium-sized projects and with low-powered devices in mind, such as modest PCs and mobile devices. Because of that, to achieve the reverb effect simply offers a 3D zone called a 'ReverbZone'. When a player is inside this zone, reverb is audible, and as the player exits the zone, the reverb gradually diminishes.[2] The zone must be placed manually by the developers.

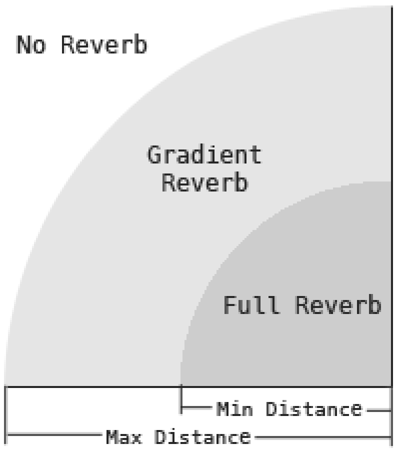


Figure 1: Gradient reverb using Unity Reverb Zones, 2024

While this approach is cost-effective and suitable for most developers, it lacks the ability to realistically emulate how the reverb effect works in a realistic way.

[2] Unity Technologies, 'Unity – Manual: Reverb Zones' 16 October 2023, <https://docs.unity3d.com/Manual/class-AudioReverbZone.html> [accessed 19 October 2023].

Unity Raycasts are a versatile tool. According to Unity's manual, ‘A Raycast casts a ray, from an origin point, in a certain direction and length, against all colliders in the Scene.’[3] Raycasts are very useful and have many applications, as the ray will detect collisions and extract information of each object, as seen in Figure 2.

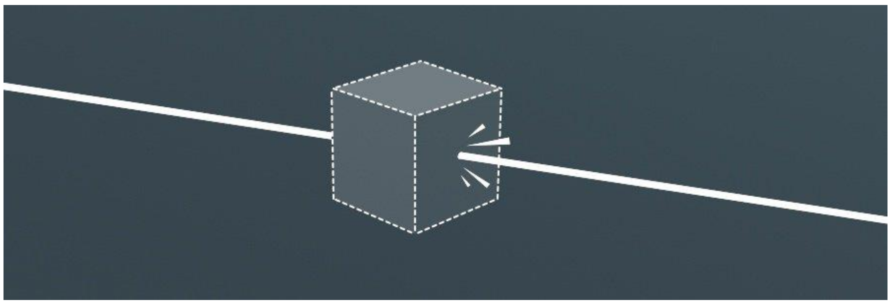


Figure 2: Raycast example. John French, 18 June 2021

With that in mind, an audio source can be thought of as an object that emits an infinite number of rays, each one in a different direction. When a ray collides with another object, it is reflected, thus creating a new Raycast. This process continues indefinitely, only stopped when the ray collides with the listener, or the intensity of the sound associated with the ray is negligible.

While Raycasts may appear to offer an ideal solution, a notable factor to consider is their computation on the CPU, which differs from the typical empowerment of ray tracing by the GPU. Because of this, caution is necessary regarding the volume of rays permitted for audio source casting, as surpassing this threshold risks game performance issues like stuttering and lag. Fortunately, obtaining the necessary information to adjust reverberation does not demand an extensive number of rays. This stands in contrast to

[3] Unity Technologies, ‘Unity – Scripting API: Physics.Raycast’ 16 October 2023, <https://docs.unity3d.com/ScriptReference/Physics.Raycast.html> [accessed 19 October 2023].

ray tracing for graphic computation, where each screen pixel typically needs at least one ray per frame.

4. Basic movement implementation

Though not the primary focus of the project, implementing basic player movement is essential to navigate and experience the virtual sound space. This project will adopt a first-person view, where the scene is observed directly from the character's perspective, enhancing realism and immersion. This stands in contrast to the third-person view, where an external camera shows both the scene and the character.

For input controls, utilizing the conventional scheme used in such games proves effective. This involves using the 'WASD' keys or the arrow keys on the keyboard for forward, backward, left, or right movements, respectively. Notably, the 'WASD' keys are positioned in a cross-like shape. The mouse will be used to look up and down and to rotate the character to the left or right.

In addition, the 'E' key will be used to interact with the environment, making some parts of the scene change through the players will. This feature will allow the player to perceive the audio reverb changes more easily. The space bar will be utilized for jumping.

5. Ray tracing implementation

Within Unity, scenes are comprised of various objects, each serving a distinct purpose. Objects include the player, individual audio sources, floors, rooms, and ceilings. Additionally, objects can be part of each other in a hierarchical scheme and possess various components. For instance, an object designated as an audio source requires the

Unity AudioSource component. Furthermore, introducing a filter involves adding a component. Filters, as components, intercept audio data from the source and apply modifications to it.

The proposed action involves augmenting an audio source object by incorporating a new component called ‘AudioRaycaster’. This additional component's primary function is to intermittently cast rays, capturing relevant spatial information. Subsequently, it will use this data to adjust the parameters of the reverb filter component.

Every 5 seconds, the AudioRaycaster script initiates a new set of rays. Through experimentation, it was determined that using 14 rays strikes a balance. This quantity provides satisfactory results without being overly computationally expensive. These rays cover the full range of possible directions uniformly, as illustrated in Figure 3.

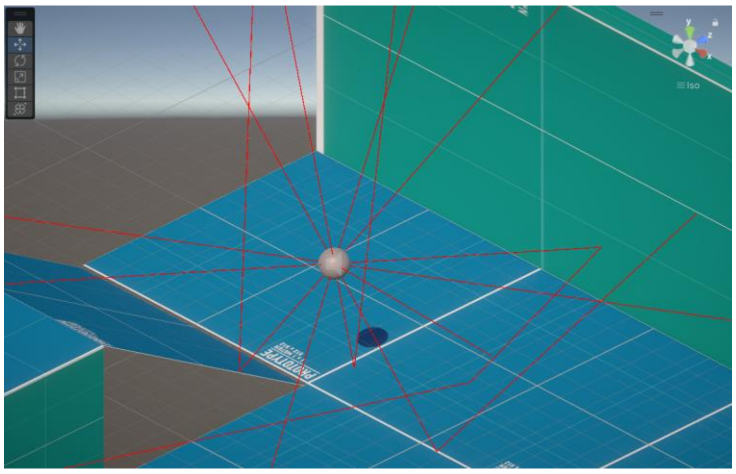


Figure 3: AudioRaycaster launching rays in different directions, Albert Madrenys, 2024

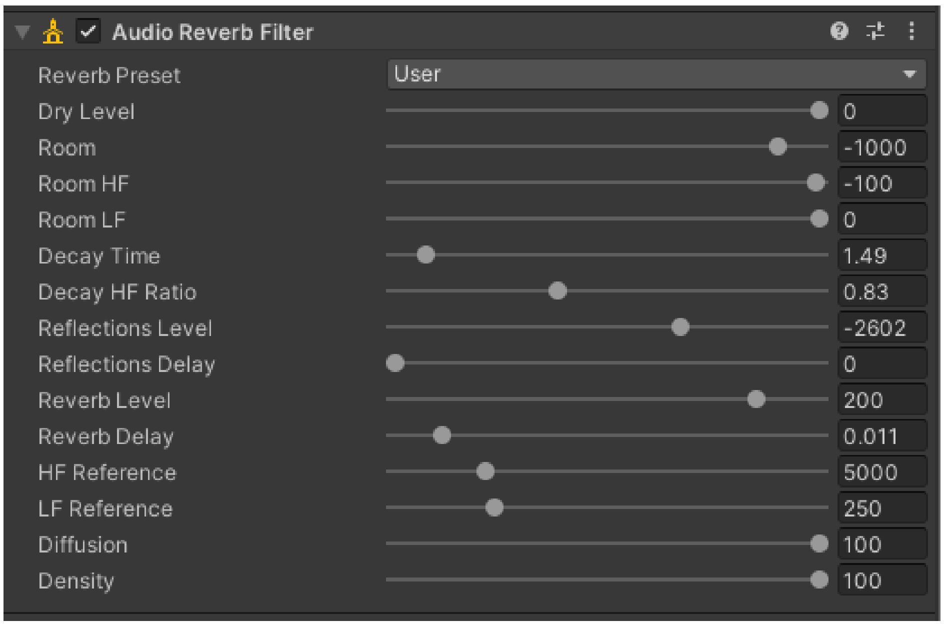
Each ray represents an instance of the ‘AudioSignal’ script, which contains a method called ‘Launch()''. This ‘Launch()’ method generates a Raycast. If this ray hits an obstacle, it collects and stores information about the obstacle. Specifically, if the obstacle is associated with a ‘AudioObstacle’ component, it saves details related to the obstacle's absorb coefficient.

This information, compiled into a structured class named ‘AudioSignalData’, is gathered by the ‘Launch()’ method. This method calls itself recursively until the maximum number of hits isn't reached. When it recalls itself, the new ray direction aligns with the reflection of the initial direction on the surface's normal. Below is the code of the ‘AudioSignal’ class showcasing the ‘Launch()’ method.

```
public class AudioSignal
{
    private int m_MaxCollisions = 3;
    private float m_MaxDistance = 100f;
    private LayerMask m_CollisionLayerMask;
    private AudioSignalData m_SignalData;

    public AudioSignalData Launch(Ray ray, LayerMask collisionLayerMask)
    {
        m_CollisionLayerMask = collisionLayerMask;
        m_SignalData = new AudioSignalData().Initialize(ray.direction);

        Launch_Internal(ray);
        return m_SignalData; // Return collected data
    }

    private void Launch_Internal(Ray ray)
    {
        float distanceTravelled = m_MaxDistance;
        if (Physics.Raycast(ray, out RaycastHit hit, m_MaxDistance,
m_CollisionLayerMask))
        {
            distanceTravelled = hit.distance;
            m_SignalData.AddNewHitInfo(hit); // Add new hit data

            if(m_SignalData.DetectedCollisions < m_MaxCollisions)
            {
                Ray nextRay = new Ray(hit.point,
Vector3.Reflect(ray.direction, hit.normal));
                // Recursive call with new position and direction
                Launch_Internal(nextRay);
            }
        }
        else
        {
            m_SignalData.AddDistanceWithoutHit(distanceTravelled);
             // Add no hit data
        }
    }
}
```

6. Unity reverb filter failed implementation

At the project's start, the exploration centred on Unity's default reverberation filter, known for its efficiency. Although Unity's documentation lacked specifics, it appeared to function as a convolution reverb filter. Yet, an early obstacle emerged: too many parameters.



Reverb Preset	User
Dry Level	0
Room	-1000
Room HF	-100
Room LF	0
Decay Time	1.49
Decay HF Ratio	0.83
Reflections Level	-2602
Reflections Delay	0
Reverb Level	200
Reverb Delay	0.011
HF Reference	5000
LF Reference	250
Diffusion	100
Density	100

Figure 4: Audio reverb filter, Unity Technologies, 2024

Attempting to comprehensively map these parameters seemed impractical given the project's scope. Even focusing on specific parameters failed to yield satisfactory and realistic outcomes. This may be attributed to the filter's primary design for optimized performance. Additionally, its effectiveness might be limited to static settings. While selecting a preset, like ‘cave’, and manually tweaking parameters could yield desired results, dynamically adjusting parameters during runtime (like when an audio source exits a cave) could easily result in inaccuracies and unrealistic effects. This discrepancy arises because although the reverb filter features parameter names related to acoustic properties, their translation into these concepts may not align seamlessly.

However, these observations remain speculative and cautionary. It's plausible that the filter lacks these limitations, hindered by time constraints for thorough investigation. Despite potential resolution with more time, exploring alternative approaches becomes crucial for project completion.

Efforts were made to modify certain parameters to reflect realistic acoustic properties. Some of these parameters include:

6.1. Early reflection delay

The early reflection delay refers to the time between the initial direct sound and the first early reflection. When considering a single ray, this delay can be calculated as the path length divided by the speed of air. However, as we're unable to predict which ray precisely represents the theoretical path from the audio source to the player (due to the audio source being unaware of the player's location in our current setup) and considering the player's frequent movement requiring quick updates, an average of all the rays is utilized.

6.2. Early reflection level

The early reflection level can be described as the presence, gain or power of the sound of the early reflections in relation from the direct sound. When a signal collides with a surface, this level varies depending on the surface's absorption coefficient and the path length:

$$W_{er} = W_{1\,m} - 20\log(Path\ lenght) + 10\log(1-\alpha)$$

where W_{er} = the early reflection sound power (in W)

$W_{1\,m}$ = the sound power of the source in 1 metre of distance (in W)

and α = absorption coefficient

Utilizing this equation enables the concatenation of multiple hits by incorporating additional terms. To accommodate more paths, we introduce terms associated with path length, while for additional collisions, terms related to the absorption coefficient are included. Each ray accumulates hit information along with its corresponding path and absorption term. Subsequently, an average across all rays is computed.

The following code segment describes the intensity difference, similar to the contrast between the final power or intensity compared to the original direct sound. This comparison arises because the original direct sound isn't accessible within this script. The primary goal of this script is to dynamically adjust the parameters of the reverb filter, rather than act as the filter itself. Each ray segment, identified by collisions, accumulates the specified terms from the preceding equation.

```
public float IntensityDifference
{
    get
    {
        float i = 0;
        foreach(float l in PathLengths) i += -20f*Mathf.Log10(l);
        foreach (float c in AbsorbCoeffitients) i +=
10f*Mathf.Log10(1-c);
        return i;
    }
}
```

7. STK and NRev

The previous unsuccessful approach required an alternative solution. Upon investigation, another reverb filter with simplicity and effective results using fewer parameters emerged: the STK library. The Synthesis ToolKit is a set of open source audio signal processing and algorithmic synthesis classes written in C++ originally developed by Perry R. Cook and Gary P. Scavone. Is open source and made primarily for academic purposes[4].

While STK is a C++ library and Unity operates in C#, Keijiro Takahashi has developed a GitHub repository that ports some STK filters to Unity[5]. Leveraging this prior work allows the author to utilize one of the available reverb filters effectively. After evaluation, the NRev was opted due to its ability to maintain natural and pleasing sounds even with extreme settings. This choice eliminates the need for cautious parameter adjustments, unlike the PRCRev or JCRev.

The NRev is based on the use of networks of simple all-pass and comb delay filters. It consists of 6 comb filters in parallel, followed by 3 all-pass filters, a lowpass filter, and another all-pass in series, followed by two all-pass filters in parallel with corresponding right and left outputs[6]. (see Figure 5).

[4] Perry R. Cook & Gary P. Scavone, ‘The Synthesis ToolKit in C++ (STK)’ 4 August 2023, <https://ccrma.stanford.edu/software/stk/faq.html> [accessed 22 December 2023].

[5] Keijiro Takahashi, ‘Unity audio filters’ 20 July 2013, <https://github.com/keijiro/unity-audio-filters> [accessed 20 December 2023].

[6] Perry R. Cook & Gary P. Scavone, ‘NRev Class Reference’ 4 August 2023, <https://ccrma.stanford.edu/software/stk/classstk_1_1NRev.html> [accessed 22 December 2023].

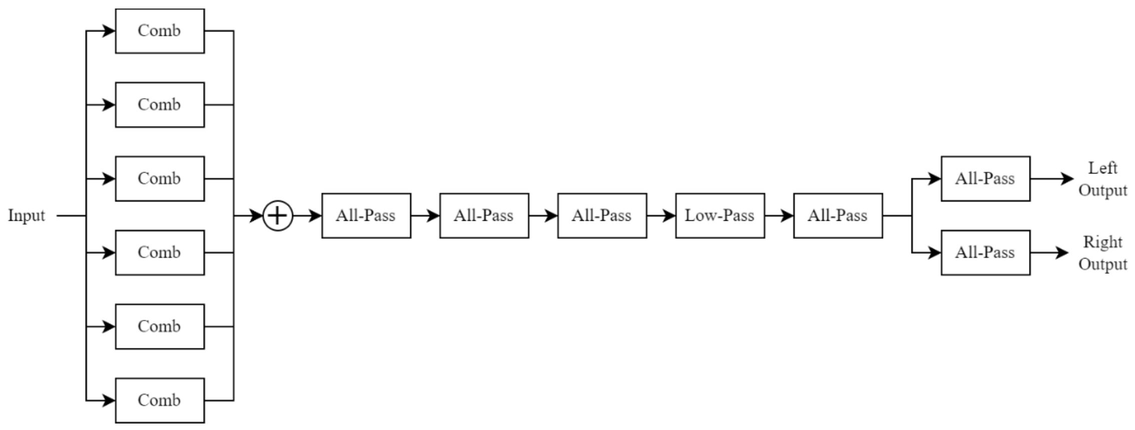


Figure 5: NRev filter network configuration, Albert Madrenys, 2024

The NRev has two parameters: decay time and send level.

8. Decay time

Decay time refers to the duration, measured in seconds, during which the reverb persists until the sound completely decays. This parameter closely resembles an acoustic aspect known as reverberation time. The prediction of reverberation time often involves using the 'Norris–Eyring reverberation formula':

$$T_{60} = \frac{-0.161\,V}{S\ln(1-\alpha)}$$

where T_{60} = the 60dB reverberation time (in s)
S = surface area of the room (in m^2)
V = volume of the room (in m^3)
and α = average absorption coefficient

Obtaining the average absorption coefficient proves straightforward by computing the mean of all coefficients encountered by the rays. However, determining the surface area and volume of the room poses a significant challenge with the available information. In this context, the concept of mean free path comes into play.

The mean free path of the room is a measure of the average distances between surfaces, assuming all possible angles of incidence and position[7]. It serves as a measure for the average duration that a sound wave travels between interactions with the room's surfaces. For an approximately rectangular room, the mean free path is calculated using the following equation:

$$MFP = \frac{4\,V}{S}$$

where MFP = the mean free path (in m)
S = surface area of the room (in m^2)
and V = volume of the room (in m^3)

Approximating the mean free path involves calculating the average length of segments between hits from all the rays. With this approximation in hand, a new equation can be created to determine the reverberation time.

$$T_{60} = \frac{-0.161\,V}{S\ln(1-\alpha)} = \frac{4\,V}{S} * \frac{-0.161}{4\,\ln(1-\alpha)} = \frac{-0.04025\,MFP}{\ln(1-\alpha)}$$

Using this formula allows for an estimation of the reverberation time based on the MFP and the average absorption coefficient. This value can then be directly applied to adjust the reverb filter.

[7] Howard D. and Angus J., *Acoustics and Psychoacoustics*, 4th edn (Elsevier Ltd, 2009), p. 300

9. Send level

The send level of the NRev describes the relative presence, power, or gain of the reverberation concerning its initial signal. This property signifies the level of the reverberant sound during its steady state. The steady state of the reverberation is the state in time where the sound level is higher. The reverb level can be described with the following equation:

$$W_{reverb} = W_{source}\, 4\, \frac{1-\alpha}{S\,\alpha}$$

where W_{reverb} = the reverberant sound power (in W)
W_{source} = the power of the source (in W)
S = surface area of the room (in m^2)
and α = average absorption coefficient

As previously discussed, the average absorption coefficient can be derived by averaging the encountered walls' coefficients within the path of rays. Calculating the source level isn't necessary, as the reverb filter internally adjusts the send level property with the source power. Being a filter, it alters the source signal accordingly.

However, a significant challenge lies in determining the surface area, much like the issue encountered when calculating the decay time due to the inherent limitations of the raytracing approach and available data. Obtaining the room's surface area becomes a complex task given the available information.

To address this challenge, the equation will be modified by substituting the surface area with the mean free path, which introduces another complication, requiring knowledge of the room's volume. The discussion on volume will be explored in the following chapter. For now, let's consider the final formula that will be utilized.

$$W_{reverb} = 4\,\frac{1-\alpha}{S\,\alpha} = 4\,\frac{1-\alpha}{\frac{4\,V}{MFP}\alpha} = \frac{MFP\,(1-\alpha)}{V\,\alpha}$$

where W_{reverb} = the reverberant sound power (in W)
MFP = the mean free path (in m)
S = surface area of the room (in m^2)
V = volume of the room (in m^3)
and α = average absorption coefficient

The last piece of this puzzle involves mapping the W_{reverb} value onto the NRev send level property, which typically spans from 0 to 1. However, when applied in Unity's virtual space using the formula, the values tend to range approximately between 1E-06 to 0.08. To accommodate this, the latter value needs to be mapped within the range of 0 to 1.

10. Volume estimation

As previously discussed, determining the volume of the room, or at least an approximation, is crucial for calculating the reverberation level. One potential approach involves gathering the mesh that comprises all the spatial positions of the first hit for each ray, followed by calculating the volume of that mesh. However, this method presents a challenge due to its computational complexity.

Virtual meshes conventionally consist of points interconnected by segments, often forming triangles. While we have access to the points, establishing relationships between these points proves challenging. Algorithms that extract the convex hull of a point set exist, but their computational demands are excessive for real-time implementation in a gaming environment. Therefore, an alternative approach is required.

Similar to our assumption made for the mean free path equation, we'll consider the room to be rectangular in this context as well. The volume of a rectangular room can be determined using the formula:

$$V = \ length * width * height$$

In our model, as the sound source is positioned at the centre of the virtual space or room, the height is calculated as the sum of the distance between the source and the floor, along with the distance between the source and the ceiling. Similar principles apply to determine the length and width, corresponding to each wall.

To establish an audio source's specific point in space, six values are required, describing the distances between the source and surfaces in six given directions: up, down, left, right, forward, and backward.

An initial approach to obtain these values would be to extract the distance of the first ray hit within each direction. However, this approach holds inherent risks. For instance, consider an audio source positioned in a forest; the forward ray might collide with a nearby tree despite the forest's larger dimensions. To mitigate this, a weighted arithmetic mean is calculated for each direction. Each mean is computed using the distances of the first ray hit, with the weight of each element determined by the magnitude of the normalized vector projection within the desired direction.

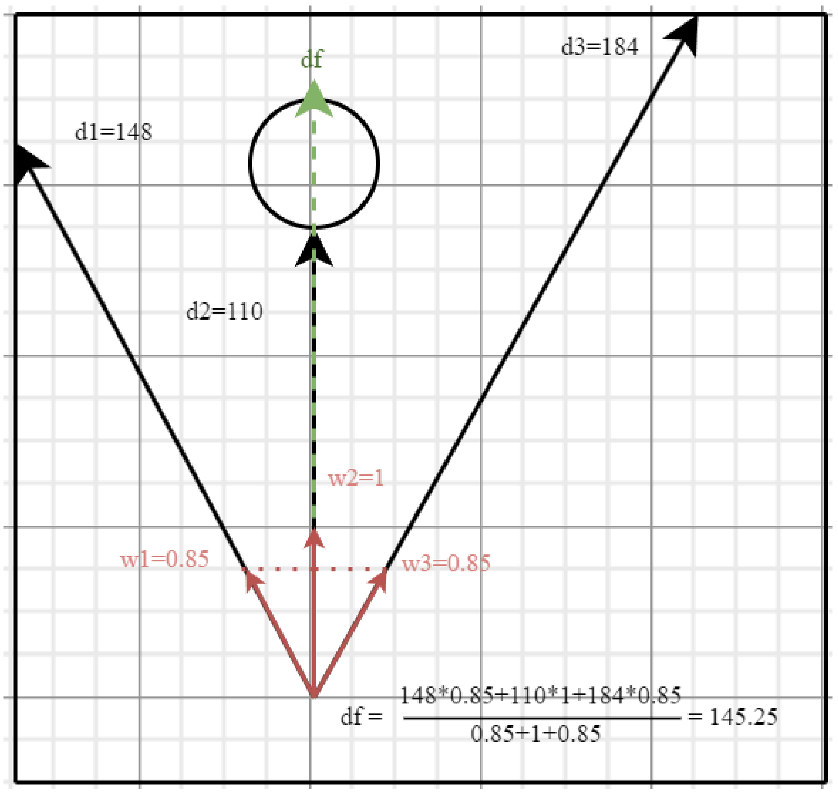


Figure 6: Distance calculation using a weighted arithmetic mean, Albert Madrenys, 2024

In Figure 6, an illustrative example showcases three rays, each associated with a distance and weight. The second ray, despite aligning with the desired direction for calculation, encounters an obstacle, distorting the representation of the room's true dimension. However, considering the additional rays, each weighted based on the similarity between their direction and the intended calculation direction, helps alleviate the impact of the obstacle. This adjustment is evident in the final direction, denoted as ‘df’ in Figure 6.

This approach gives good results for the level of accuracy needed. When translating this method into Unity, efficiency becomes crucial, necessitating a reduction in operations. Consequently, encountering a negative weight signifies a positive weight in the opposite direction. For instance, when computing the forward direction, a negative weight indicates that the corresponding ray should contribute to the backward direction, yet with its weight adjusted to a positive value. This approach optimizes calculations by reassigning negative-weighted rays to their appropriate opposite direction, minimizing computational load.

11. Sound materials

Sound material, within this context, refers to how material properties affect audio signals upon collision with an object. In this implementation, a sound material will be characterized solely by its absorption coefficient. The coefficients provided below are extracted from real-world material tables and adjusted to suit video game applications[8].

Material Name	Absorption Coefficient
asphalt	0,60
body / flesh	0,70
brick	0,35
carpet / cloth / fabric	0,65
ceramic / marble	0,01
composite / plastic	0,30
concrete	0,05
cut stone	0,10
dirt / soil	0,60
foliage / grass / moss / plant / vegetation	0,75
glass	0,10
gravel	0,40
ice	0,20
iron / metal / steel	0,10
rock	0,15
sand	0,55
snow	0,80
water	0,01
wood	0,25

[8] James Boer 'Automatic Generation of Environmental Reverb Data' 24 September 2020 <https://github.com/JamesBoer/ReverbGen/tree/main/Docs> [accessed 2 January 2024], p. 8

In the Unity implementation, each audio material will exist as an independent scriptable object. These scriptable objects, belonging to the ‘AudioMaterial’ type, will be distinct assets stored within the project's asset folder. Subsequently, every audio obstacle will feature a basic component containing a reference to the corresponding material. When a ray intersects with an object, the ray will access the material's scriptable object and retrieve its data through the obstacle's component. This mechanism enables the retrieval and utilization of material information by the ray during collisions.

12. Results

To assess the effectiveness of the approach, a Unity scene has been constructed, incorporating multiple audio sources in distinct settings. Each of these sources uniformly plays the same sample: the sound of clapping hands. This choice stems from the unique attributes of clapping sounds: a brief, impulse-like nature coupled with a wide frequency range. These characteristics make clapping sounds ideal for discerning subtle reverb variations across different environments. Utilizing this sample within diverse settings in the Unity scene is expected to yield insights into the comparative reverberation effects across these varied settings.

In the described scene, three distinct sections can be described. The initial section comprises the main square, featuring a basic platforming layout with a wall, ramp, and level floor (refer to Figure 3). Notably, the wall's AudioMaterial is brick, while the floor material is asphalt. An important element of this section is the player's interaction, triggered by pressing the ‘E’ key, which prompts an additional iron box to envelop the audio source entirely. Given iron's minimal absorption coefficient and the resulting low mean free path, the reverberation effect becomes notably pronounced in this setting.

Moving to the second segment, the scene transitions to the chapel, a construction predominantly made of stone, designed to loosely resemble real chapels. Because of the stone's relatively low absorption coefficient and the chapel's confined space, the reverb effect is perceptible, albeit not as pronounced as in the main square, particularly with the iron box. In Figure 7, the rays observed within the chapel are depicted in red, showcasing their distinct paths and interactions. The volume distances, determined using the previously described approach, are denoted in yellow, providing a visual representation of the calculated distances within this space.

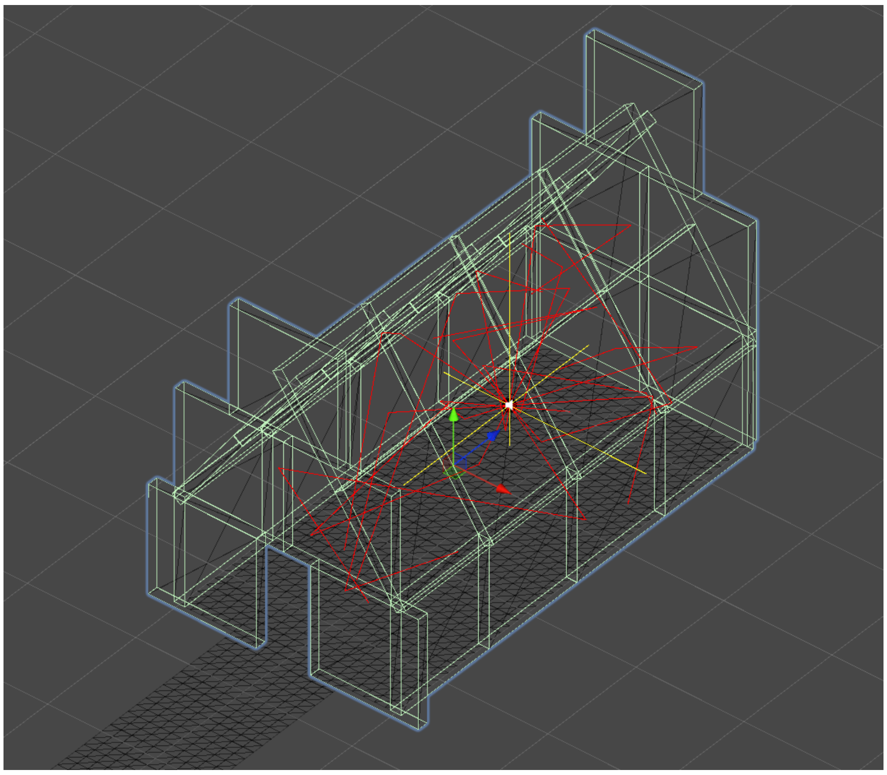


Figure 7: Chapel section ray tracing and volume distances, Albert Madrenys 2024

The final section depicts a forested area, with tree trunks constructed from wood and the ground and foliage constructed from the vegetation AudioMaterial. Despite being relatively open, the forest presents numerous obstacles. This setting serves as a valuable test to assess the effectiveness of volume calculations within a challenging and intricate configuration.

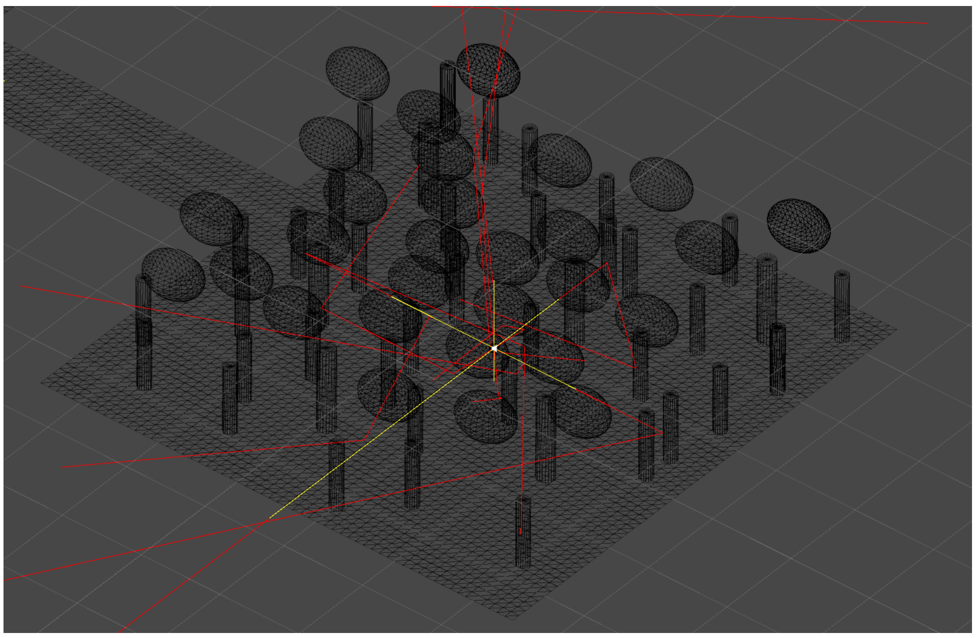


Figure 8: Forest section ray tracing and volume distances, Albert Madrenys 2024

In Figure 8, the accuracy of volume calculations for the vertical directions—down (representing the floor) and up (indicating the height of tree foliage)—is notably precise. However, the horizontal directions exhibit varying degrees of accuracy, with only one extending significantly far. Despite inherent limitations and unavoidable inaccuracies, it's evident that the system works better than a simplistic approach that merely acquires the distance of the Raycast within those directions.

13. Conclusion

By employing the sequence of operations detailed earlier, a concise demo was developed to assess the tool's functionality. Impressively, the outcomes were notably satisfactory. Despite operating with just two parameters, the NRev STK filter demonstrated exceptional capabilities, accurately reproducing natural-sounding reverberations even under extreme parameter settings.

Additionally, the tool's user-friendliness for external developers is noteworthy. Integration is effortless—simply appending the ‘AudioRaycaster’ component to the audio source objects and attaching an ‘AudioMaterial’ to each wall.

Moving forward, the logical progression involves refining the tool further. Smooth transitions among NRev parameters post-ray launch would mitigate abrupt, artificial changes, enhancing the overall experience. Exploring mechanisms to detect environmental changes and subsequently trigger ray launches could optimize resource utilization by preventing unnecessary launches in unchanged environments, currently set at intervals of 5 seconds.

Furthermore, exploring this approach in other sound phenomena stands as a logical extension. Revisiting the use of Unity's default reverb filter presents opportunities, as this filter manipulates early reflections and diverse frequency ranges, unlike the NRev filter, which focuses solely on reverberation.

Despite these potential enhancements, the existing results, though straightforward, remain satisfactory and user-friendly. They form a solid foundation for further development, promising an even more refined and effective tool in the future.

14. Bibliography

[1] Hofmann, G.R. '*Who invented ray tracing?*' The Visual Computer 6, 120–124 (1990). <doi:10.1007/BF01911003>.

[2] French, J. '*Raycasts in Unity, made easy - Game Dev Beginner*' 18 June 2021, <https://gamedevbeginner.com/raycasts-in-unity-made-easy/> [accessed 19 October 2023].

[3] Unity Technologies, '*Unity – Scripting API: Physics.Raycast*' 16 October 2023, <https://docs.unity3d.com/ScriptReference/Physics.Raycast.html> [accessed 19 October 2023].

[4] Unity Technologies, '*Unity – Manual: Reverb Zones*' 16 October 2023, <https://docs.unity3d.com/Manual/class-AudioReverbZone.html> [accessed 19 October 2023].

[5] David M. Howard & Jamie A. S. Angus, '*Acoustics and Psychoacoustics*' 4th edn (Elsevier Ltd, 2009).

[6] Bilkent Samsurya, '*Realtime Audio Raytracing and Occlusion in Csound and Unity*' 6 November 2022, <https://csound.com/icsc2022/proceedings/Realtime%20Audio%20Raytracing%20and%20Occlusion%20in%20Csound%20and%20Unity.pdf> [accessed 26 December 2023].

[7] Unity Technologies, '*Unity – Manual: Audio Reverb Filter'* 7 November 2023 <https://docs.unity3d.com/ScriptReference/AudioReverbFilter.html> [accessed 7 November 2023].

[8] Perry R. Cook & Gary P. Scavone, '*The Synthesis ToolKit in C++ (STK)*' 4 August 2023, <https://ccrma.stanford.edu/software/stk/faq.html> [accessed 22 December 2023].

[9] Perry R. Cook & Gary P. Scavone, '*NRev Class Reference*' 4 August 2023, <https://ccrma.stanford.edu/software/stk/classstk_1_1NRev.html> [accessed 22 December 2023].

[10] Keijiro Takahashi, '*Unity audio filters*' 20 July 2013, <https://github.com/keijiro/unity-audio-filters> [accessed 20 December 2023].

[11] Higini Arau-Puchades, '*Sound Pressure Levels in Rooms: A Study of Steady State Intensity, Total Sound Level, Reverberation Distance, a New Discussion of Steady State Intensity and Other Experimental Formulae*' (Building Acoustics, 2012) <doi:10.1260/1351-010X.19.3.205>.

[12] James Boer '*Automatic Generation of Environmental Reverb Data*' 24 September 2020 <https://github.com/JamesBoer/ReverbGen/tree/main/Docs> [accessed 2 January 2024].

[13] Niklas Röber, Ulrich Kaminski and Maic Masuch, '*Ray Acoustics Using Computer Graphics Technology*', September 2007 (Proceedings of the 10th International Conference on Digital Audio Effects)

15. Annex: Unity reverb filter presets

During the period dedicated to configuring the Unity reverb filter, the author encountered a challenge: the unavailability of presets for simultaneous comparison. This lack of access to these presets within Unity, documentation, or any online resources prompted the individual to compile the presets independently. As a result, a comprehensive compilation has been assembled for reference purposes:

	Dry Level	Room	Room HF	Room LF	Decay Time	Decay HF Ratio	Reflections Level
Generic	0	-1000	-100	0	1.49	0.83	-2602
Padded Cell	0	-1000	-6000	0	0.17	0.1	-1204
Room	0	-1000	-454	0	0.4	0.83	-1646
Bathroom	0	-1000	-1200	0	1.49	0.54	-370
Livingroom	0	-1000	-6000	0	0.5	0.1	-1376
Stoneroom	0	-1000	-300	0	2.31	0.64	-711
Auditorium	0	-1000	-476	0	4.32	0.59	-789
Concert hall	0	-1000	-500	0	3.92	0.7	-1230
Cave	0	-1000	0	0	2.91	1.3	-602
Arena	0	-1000	-698	0	7.24	0.33	-1166
Hangar	0	-1000	-1000	0	10.05	0.23	-602
Carpeted Hallway	0	-1000	-4000	0	0.3	0.1	-1831
Hallway	0	-1000	-300	0	1.49	0.59	-1219
Stone Corridor	0	-1000	-237	0	2.7	0.79	-1214
Alley	0	-1000	-270	0	1.49	0.86	-1204
Forest	0	-1000	-3300	0	1.49	0.54	-2560
City	0	-1000	-800	0	1.49	0.67	-2273
Mountains	0	-1000	-2500	0	1.49	0.21	-2780
Quarry	0	-1000	-1000	0	1. 49	0.83	-10000
Plain	0	-1000	-2000	0	1.49	0.5	-2466
Parking Lot	0	-1000	0	0	1.65	1.5	-1363
Sewer Pipe	0	-1000	-1000	0	2.81	0.14	429
Underwater	0	-1000	-4000	0	1.49	0.1	-449
Drugged	0	-1000	0	0	8.39	1.39	-115
Dizzy	0	-1000	-400	0	17.23	0.56	-1713
Psychotic	0	-1000	-151	0	7.56	0.91	-626

	Reflections Delay	Reverb Level	Reverb Delay	HF Reference	LF Reference	Diffusion	Density
Generic	0	200	0.011	5000	250	100	100
Padded Cell	0	207	0.002	5000	250	100	100
Room	0	53	0.003	5000	250	100	100
Bathroom	0	1030	0.011	5000	250	100	60
Livingroom	0	-1104	0.004	5000	250	100	100
Stoneroom	0	83	0.017	5000	250	100	100
Auditorium	0	-289	0.03	5000	250	100	100
Concert hall	0	-2	0.029	5000	250	100	100
Cave	0	-302	0.022	5000	250	100	100
Arena	0	16	0.03	5000	250	100	100
Hangar	0	198	0.03	5000	250	100	100
Carpeted Hallway	0	-1630	0.03	5000	250	100	100
Hallway	0	441	0.011	5000	250	100	100
Stone Corridor	0	395	0.02	5000	250	100	100
Alley	0	-4	0.011	5000	250	100	100
Forest	0	-229	0.088	5000	250	79	100
City	0	-1691	0.011	5000	250	50	100
Mountains	0	-1434	0.1	5000	250	27	100
Quarry	0	500	0,025	5000	250	100	100
Plain	0	-1926	0,1	5000	250	21	100
Parking Lot	0	-1153	0.012	5000	250	100	100
Sewer Pipe	0	1023	0.021	5000	250	80	60
Underwater	0	1700	0.011	5000	250	100	100
Drugged	0	985	0.03	5000	250	100	100
Dizzy	0	-613	0.03	5000	250	100	100
Psychotic	0	774	0.03	5000	250	100	100