Realtime Audio Raytracing and Occlusion in Csound and Unity

Bilkent Samsurya[1]

Berklee College Of Music
bilkentcw@gmail.com

Abstract. Extensive research has gone into the development of interactive virtual environment tools but its studies on audio technology are limited. There are many acoustical modeling tools which can be applied to audio processing for virtual environments and the integration of Csound and Unity presents an opportunity to explore this area of research. This thesis describes an implementation of a realtime ray tracing and sound occlusion system to model acoustics in a virtual environment. The game engine Unity is chosen for the environment and Csound is the sound system chosen for the audio processing component. In the following paper, the mechanisms for which the process is outlined and the workflow is presented. A test to evaluate the effectiveness of the system is conducted. This is followed by a description of its limitations and proposal for a revised model.

Keywords: Csound, Unity, Raytracing, Occlusion

1 The Problem and Initial Description

Research into the development of audio implementation tools has been lagging behind rendering technology in the application of virtual simulations. Audio interfaces have been increasingly overshadowed and its outdated interfaces and subsequent unintuitive workflow only serves to amplify the problem. But to create a more seamless and immersive experience, there needs to be a greater emphasis in the development of audio tools so that more realistic audio cues can be produced to support better interactions between the user and the virtual environment [1].

The integration of Csound and Unity presents an opportunity for the development of such technologies. The use of Csound as an audio engine can facilitate the incorporation of realistic sound modeling methods that can bridge the gap between sound and light behavioral simulations in virtual environments.

Audio ray tracing is a technology that has been extensively used to replicate sound propagation in acoustical softwares. The inclusion of this sound modelling method in game development can go a long way in designing a rich and responsive acoustical environment. It has the potential to reinvent the way sound designers interface with game engines and effectively allow for a more efficient mode of

implementation. In this paper, a method of using audio ray tracing to stochastically produce sound reverberations and occlusion effects is demonstrated and evaluated in the Unity environment, using Csound as the audio processing unit.

2 Program Architecture

The delegation of work between the programs is established in this section.

The virtual environment is built and run in the Unity game engine. The ray generation, detection process and the data required to perform sound alteration processes is stored and updated in Unity. When the appropriate conditions are met, the data and sound file is sent from Unity to Csound for signal processing (filtering, eqs, delays).

Csound alters the sound based on the data sent from Unity and the treated sound is sent back to Unity for playback. This process is done in run time allowing for users to be able to modify that data and hear the evolved audio signal in real time. The implementation of this project can be conceptualized as having two components, one to handle the reflections and the other to account for sound occlusion. A high level depiction of the pipeline is illustrated in figure 2.

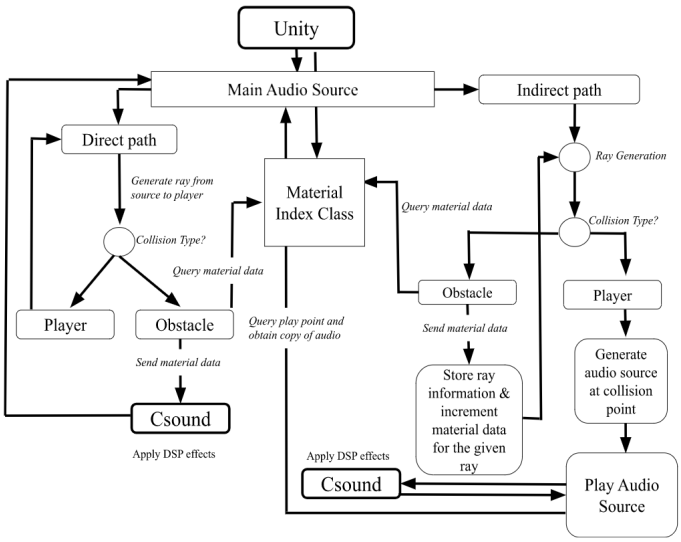

Figure 2 : Illustration of the program pipeline

3 The Ray Generation and Detection Process

A number of rays (a variable that can be determined by the user) are generated in randomly set directions (sound source polarity assumed to be omnidirectional) from the position of the audio source. If a ray collides with an object that is not the player, the data associated with the material of the object, length of the ray

and angle between the normal of the object's surface and the ray is queried and stored. A new ray is generated from the point of collision at an angle from the normal of the surface of the collision. The angle of the reflected ray is equivalent to the angle of the previous ray, ensuring that the reflection is specular[6]. The process for generating rays is illustrated in Figure 3.

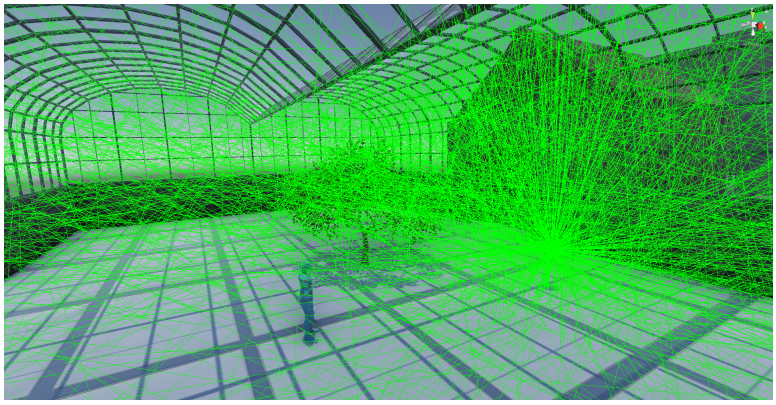

Figure 3 : The green lines represent the sound propogation paths from a point source to the listener in a glass dome.

The ray will continue to propagate until one of three conditions is satisfied:

1. The number of reflections has been exceeded. The number of reflections represents the order of reflections the system takes into account. This variable, although customizable by the player, should be kept low (4 or below). This is to ensure that Unity is able to run at acceptable frame rates ($\geq$30fps) and latency is kept low.

2. The total distance traveled by the ray has exceeded a maximum threshold. This threshold is to ensure that the sound path does not propagate indefinitely. It is a user-modifiable variable so as to allow users to match the propagation paths to take into account enclosures of varying sizes and geometries.

3. The ray collides with the player. The method for detecting ray collision in Unity is employed by placing a collider around the player. The most common implementation of this form of detection is to use a spherical collider to ensure that there is an equal likelihood of collision from all directions [2]. An important consideration is the size of the sphere. This is because the size of the collider will affect the probability of collisions. If the collider is too large it results in the oversampling of rays which can result in a false acoustical image of the space. Similarly, a collider that is too small may result in the undersampling of rays which leads to an incomplete acoustical image of the space. It is shown in [2] that the minimum radius of the receiver required can be determined by the following equation :

$$r = \sqrt{\frac{V}{\pi c \Delta t N}}$$

where r is the radius of the listener, V is the volume of the environment, c is the speed of sound, Δt is the audio sampling period and N is the number of rays. When a collision is detected, a secondary audio source is generated at the point of collision and triggered to play the reflected sound.

4 Processing in Csound

In this section we follow the sound reverberation generation process in Csound.

The first task of Csound is to break the sound into 9 discrete frequency bands: 63Hz, 125Hz, 250Hz, 500Hz, 1KHz, 2KHz, 4KHz, 8KHz and 16kHz so that each bandwidth can be processed separately. This is done by applying bandpass filters, using the butterbp opcode. Each of these signals is then attenuated based on the absorption coefficient values associated with the frequency band. The pareq opcode is used for the attenuation. To establish a relationship between the absorption coefficient and attenuation factor, the following equation was used to translate absorption coefficient values to dB values.

$$\Delta dB = 20log_{10}(1-\alpha)$$

where Δ dB is the change in decibel values and α is the absorption ceofficient.

In obtaining the Δ dB value, it can be converted to an amplitude value using the ampdb opcode, which can then be used as input to attenuate or boost parameters for the paraeq opcode. A positive α would imply attenuation while a negative would imply a boost. A constraint is that if values of $\alpha \geq 1$, it would have to be truncated to 0.99 to avoid a math error. This is congruent with how sound absorption works in real world acoustics as surfaces with an absorption coefficient of 1 or greater would imply a 100 % absorption of sound.

Early conceptions of electronically generated reverbs were lacking in two areas: low echo densities which resulted in an unnatural fluttering effect and a non flat frequency responsive which adds unpleasant coloration to the reverberated sound [3]. These limitations resulted in the reverb sounding unnatural and artificial. Schroeder's reverb design introduced the use of allpass filters and feedback delay lines which helped to substantially increase echo densities and allowed for a flat frequency response [3]. Taking these factors into consideration, the reverb design for this program was built around the use of allpass filters. The alpass opcode in Csound has two parameters which are important in determining the reverb amount and tonality: reverb times (i.e RT60) and echo densities. In acoustics, the room's reverb time (RT60) can be approximated using Sabine's equation [4].

$$\text{RT60} = \frac{0.049V}{S\alpha}$$

where V is the volume of the room, S is the total surface area of the room and α is the total absorption coefficients of the room. It is clear from Sabine's equation that the room size V, is a significant variable in determining the RT60 time. In ray tracing reverb however, the room size is stochastic as the size and surface area of the room contributing to the reverb generation varies based on the listener's position within the space. This difference can be accounted for by using the distance traveled by each contributing ray in place of the volume of the room. The RT60 formula can then be reconfigured to:

$$\text{RT60} = \frac{d}{(1+\zeta)(1+\alpha)}$$

where d is the distance travelled by the ray , ζ is the damping factor and α is the total contibuting absorption coefficients. For the purpose of giving customizability to the

user, a damping factor is introduced to the equation to modify the decay amount and allow users to more accurately fine tune the parameter. Aside from the reverb decay time, the order of reflections also determines the tonality and clarity of the reflected sound. As the order of reflections increases, the more smeared and complex the sound becomes [6]. To emulate this effect, a feedback loop through the allpass filters can be used. More specifically, the order of reflections determine the number of times the signal is looped through the allpass filter. After the signal has been allpass filterered, the signal is then delayed by a time period which is derived by dividing the distance traveled and the speed of sound (343 m/s) [1]. The implementation of the sound processing for one contributing ray can be summed up in the following diagram:

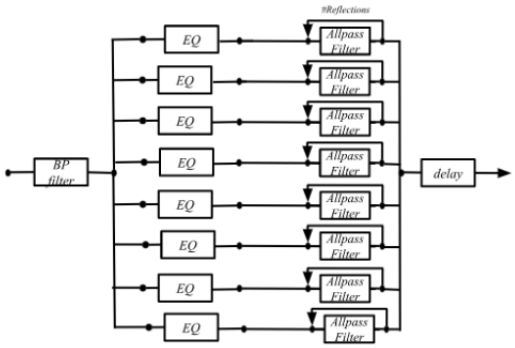

Figure 4 : Illustratation of the reverb signal flow

5 Evaluating Impulse Responses

The impulse response of a room is a good measure in determining the acoustical properties of the room. A common practice in acoustics is to use exponential sine sweeps to measure the impulse response [5]. Four environments were modeled in Unity, consisting of various sizes and materials. An exponential sine sweep with a 2s duration is generated from Matlab and triggered in these environments. It is assumed that the sound source and the receiver are in the same location when the tests are performed. The impulse respoonses of the four rooms are tabulated and depicted in the graph below.

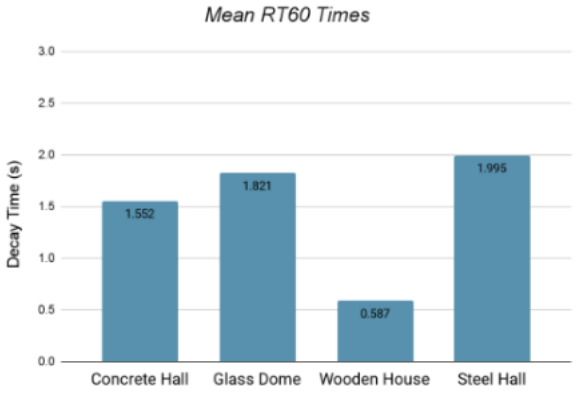

Figure 5 : The results of the impulse responses of 4 envrionemnts : Concrete hall, Glass dome, Wooden house, Steel hall

6 Constraints and Possible Improvements

The current model relies on the Unity audio engine to play the generated reverb effect. A constraint from this method of implementation is that the number of active audio sources can grow very large depending on the number of rays that collide with the player. The unity audio engine can realistically support 32 real voices (channels) and the number of contributing rays can quickly surpass this limit. A more elegant approach would be to use the coordinates of collisions between contributing rays and player to translate and pass the hrtf information to Csound. In doing so, Csound would be the program responsible for the sound generation, requring only two active channels to generate the reverb effect.

7 Conclusions

The current state is a working model. Future work on the project will consist of research into more complex reverb designs and efficient raytracing algorithms. This project demonstrates the capabilites of Csound as an audio processing tool in virtual simulations applications and further exploration in this area holds the potential for revitalizing the way audio tools can be conceptualized and conceived.

References

1. Funkhouser, Thomas & Tsingos, Nicolas & Jot, Jean-Marc. (2003). Survey of Methods for Modeling Sound Propagation in Interactive Virtual Environment Systems.

2. Ray Tracer. Wayverb - Ray tracer. (n.d.). Retrieved July 14, 2022, from https://reuk.github.io/wayverb/ray_tracer.html

3. MA. R.. Schroeder, "Natural Sounding Artificial Reverberation," J. Audio Eng. Soc., vol. 10, no. 3, pp. 219-223, (1962 July.). doi:

4. Leo L. Beranek, Tim J. Mellow, Chapter 10 - Sound in enclosures, Editor(s): Leo L. Beranek, Tim J. Mellow,Acoustics: Sound Fields and Transducers,Academic Press,2012, Pages 449-479,ISBN 9780123914217, https://doi.org/10.1016/B978-0-12-391421-7.00010-5

5. P. Guidorzi, L. Barbaresi, D. D'Orazio, M. Garai, Impulse Responses Measured with MLS or Swept-Sine Signals Applied to Architectural Acoustics: An In-depth Analysis of the Two Methods and Some Case Studies of Measurements Inside Theaters, Energy Procedia, Volume 78, 2015, Pages 1611-1616, ISSN 1876-6102, https://doi.org/10.1016/j.egypro.2015.11.236.

6. Rathsam, Jonathan & Wang, Lily. (2006). A Review of Diffuse Reflections in Architectural Acoustics. 10.1061/40798(190)23.