Interactive Geometry-Based Acoustics for Virtual Environments

Using the Graphics Card

by

R. Baksteen

4017781

Media & Knowledge Engineering
Specialization: Computer Graphics

This thesis was submitted for
the degree of Master of Science in the

Computer Graphics & Visualization Group,
Department of Intelligent Systems

at the Delft University of Technology, faculty EEMCS,
to be defended publicly on Monday October 30, 2017 at 10:00 AM.

Supervisor:	Prof. dr. E. Eisemann,	TU Delft
Thesis committee:	Prof. dr. E. Eisemann,	TU Delft
	Dr. ir. R. Heusdens,	TU Delft
	Dr. ir. R. Bidarra,	TU Delft

Delft
University of
Technology

Preface

Thesis Topic

This thesis was submitted for the degree of Master of Science in the computer graphics & visualization group, department of intelligent systems at the Delft university of technology. It aims to make a contribution in the field of acoustics modeling in virtual environments. The challenge in this field is to accurately auralize virtual sound sources in a virtual space, which has applications in architectural acoustics, games and virtual reality.

Abstract

The acoustics of a space contribute much to the perceived size and atmosphere of the environment. Modeling these acoustics in virtual environments can make a big improvement to the immersion of the player. There are very accurate models that can reproduce realistic acoustics in virtual spaces, but these are computationally very expensive. Cheaper algorithms are either very inaccurate or impose very limiting constraints on the environment. This thesis aims to introduce an acoustics model that can run in real time, while still providing a reasonable degree of accuracy. The perceptually most important parts of the reverberation are modeled accurately in real time, allowing for excellent sound source localization. The remaining part of the reverberation is generated using an artificial reverberator whose input parameters are derived from the geometry, again in real time. This system is intended to give players a convincing sound in any environment without heavily taxing the available processing power.

Thesis Committee

Prof. dr. E. Eisemann,	TU Delft
Dr. ir. R. Heusdens,	TU Delft
Dr. ir. R. Bidarra,	TU Delft

Acknowledgements

First and foremost I want to thank my supervisor Elmar Eisemann for his continued support and enthusiasm towards my thesis project. His insights, experience and critical thinking have really enabled me to push for better solutions and a better final product. I could not have wished for a better supervisor. I also want to thank Richard Heusdens for his contributions in our meetings and his valid criticisms during the course of this project. I want to thank Timothy Kol for his assistance with OpenGL programming and helping me understand graphics programming further. I want to thank all the other employees and students of both the computer graphics group and the audio processing group who attended my mid-term presentations for providing valuable feedback and asking great questions that helped me think critically. Thank you to my girlfriend Martine van den Berg for all her love and support. Finally, I want to thank all my family and friends for their continued support and interest in my project. A special mention goes to my two grandmothers who both took great interest and always believed in me, but who both did not make it to see the end result of this project. Above all, I want to thank God for everything He has given me and everything He continues to give.

R. Baksteen
Delft, October 2017

Contents

1 Introduction

1.1. Main Research Themes

The main goal of this project is to design an acoustics model for virtual environments that can run in real time. To enable full freedom for both the player and the level designer, no pre-processing and minimal input from the environment artist are required to arrive at a convincing sound. This allows for truly interactive acoustics at any location that can handle unforeseen changes in geometry, while providing natural sounds for any environment. Additionally, sound source localization is possible because the most important propagation paths are determined accurately using the geometric model. This system will model the direct signal path and first order diffraction as rays and first order specular reflections as a combination of rays and beams. Higher order reflections are generated using an algorithmic reverberator whose input parameters are derived from the geometry during runtime. The system handles sound absorption in three frequency bands, allowing the environment artist a lot of freedom in designing acoustically distinct environments. The main contributions are a novel way of finding 0^{th} and 1^{st} order propagation paths efficiently and a fast method of extracting late reverberation parameters from the geometry in real time. Limitations are the very basic sound source occlusion and the computationally expensive auralization. There are also some geometry requirements that are necessary to prevent invalid propagation paths. Since the late reverberation is modeled by an algorithmic reverberator, the system is not usable if high accuracy is required. Only the perceptually most important parts of the reverberation are modeled accurately.

1.2. Overview Of The Report

After this introduction, the report starts with a chapter that gives background information on the research topic, to help the reader understand some of the techniques and terminology used in this field. Next, some relevant previous publications on this subject are presented, which gives the work in this report some context and shows in what way its contributions are valuable to the field. Then the system itself is introduced starting with a complete overview, followed by a chapter on how all the information that is required for auralizing a sound source is gathered from the geometry. The next chapter then discusses the actual auralization, with all the signal processing that this entails. After that, the results are presented, showing both the accuracy and the performance of the system. The final two chapters discuss these results and show areas for future work.

2 Background

The work that is presented in this report combines techniques from two different fields, namely acoustics and computer graphics. This chapter aims to introduce some terminology and techniques from each of these fields that is required to understand the underlying principles of the remainder of this report.

2.1. Acoustics

The field of acoustics is concerned with explaining the behaviour of sound waves. Sound waves are pressure waves that travel from a sound source to the eardrum in a human ear, which allows us to hear. The human ear picks up the sound waves as they travel through the air at speeds of around 330m/s. Adults can hear frequencies of roughly 20Hz up to 16kHz, which are perceived as very low and very high pitches respectively. The propagation of a sound wave through the air can be described by the wave equation, a second order partial differential equation. Given some boundary conditions and an excitement or initial displacement, it can model sound propagation quite accurately. Solving the wave equation results in a continuous pressure field, where at any location the pressure is known. That this pressure field can become very complex very quickly can be seen in Figure 2.1, which shows a sound source emitting a single frequency wave at a wall. The reflected sound creates a complex interference pattern in the pressure field. The fact that sound waves reflect off of surfaces this way is what makes acoustics interesting.

2.1.1. Room Acoustics

When a sound source is in an enclosed space, the walls will redirect the sound energy back into the room. When listening at a fixed position in that room, this means that the original wave emitted from the sound source will reach the listener first. All the reflected waves arrive later, because they have to travel further. With every reflection, some of the sound energy is absorbed. As a result the energy in the waves slowly decays, until they are no longer audible. Reflections are usually referred to using reflection orders. A first order reflection is a wave that reflected of a wall once, a second order reflection has reflected twice etc. In a room with little absorption, the reflection order can go over one hundred before the signal dies out. The time between the first wavefront arriving and the reflections becoming inaudible is an important acoustic characteristic of a room, usually referred to as the T_{60}, the time it takes for the signal level to drop by 60dB. The T_{60} of a room is primarily influenced by two things: its size and the acoustics properties of the surfaces. A bigger room means that the waves can travel longer without interacting with the walls, which results in a longer T_{60}. How much sound energy is absorbed by the wall materials negatively influences the T_{60}. All of this is assuming that the room is completely closed. In practice, an open door or window will allow the sound energy to bleed away over time, decreasing the T_{60} further. The absorption of sound energy by walls is often very frequency dependent. As a result, the materials in the room will start to color the sound noticeably at higher reflection orders, since some frequencies will last much longer than others. In general, lower frequencies are more difficult to absorb.

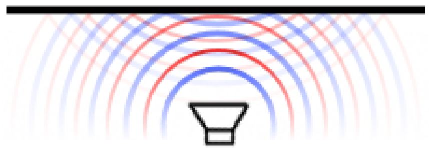


Figure 2.1: A sound source emitting a pressure wave which is being reflected off the wall.

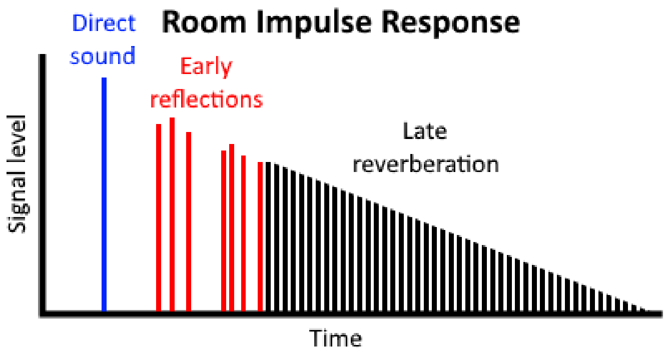


Figure 2.2: An example of a room impulse response, showing the temporal distinction between the early and late reflections.

Impulse Response

The acoustic properties of a setup with a fixed sound source and a fixed receiver in a room can be captured entirely in what is called a room impulse response (RIR or IR). Although measuring an accurate RIR in a real room is difficult to do well, the concept is straight forward. Imagine playing a very short click (the impulse) from the sound source. At the receiver, every time this click is heard, the signal level of that click is plotted against the time. The result is an IR that can look something like Figure 2.2. It shows the signal that arrived directly from the sound source first, and an increasing amount of reflections arriving later, with decreasing signal level. What makes an IR useful is the fact that it can be convolved with any sound signal. The result will sound like that sound source was in the room the IR was recorded in. Figure 2.2 makes a distinction between early and late reverberation. That is because the early reflections and late reverberation have some distinct properties. The early reflections are relatively far apart in time, which means they can individually be perceived by the human ear. As the reverberation develops, the reflection density becomes so high that the result sounds more like a wash of reflections instead of individual echos. This wash is often referred to as reverberation or reverb.

2.1.2. Localization

The human ear is capable of localizing the origin of a sound quite well. Both the direction and the distance of a sound source can be estimated in the right circumstances. To determine the direction of a sound source, the brain processes the differences between the left and right ear. Both the difference in time and the difference in pressure level give an indication of whether the sound source is to your left or to your right. The signal from a sound source to your right will arrive at your right ear a little bit earlier than to your left ear. Although the timing difference between the two ears is less than a millisecond, the human brain can distinguish that and use it. With all the reflections coming in at nearly the same time, that would be too much information to process and make sense of. For that reason, the human brain is trained to base its decision mostly on the first arriving wavefront. That makes sense, because in most cases the first wave to arrive is the direct sound, which comes from the direction of the sound source. This is referred to as the Haas-effect [1]. The contributions of the later reflections to localization decrease with the increasing echo density. The reliance on the first wavefront becomes very clear in a situation with very distinct long range echos, which can have one look in the wrong direction to find the sound source.

Apart from the timing, the pressure level and frequency content of the signal in both ears contains directional information. The head will obstruct some of the signal in the left ear in the case of a sound source on one's right, creating a difference in the pressure level between the ears. Finally, the outer ear filters high frequencies when a sound source is behind you, so that every direction sounds slightly different. It is very difficult to hear these things consciously, but the human brain is capable of using all this information to effectively localize a sound source.

For determining the distance to a sound source, the main indicator is pressure level. Consider a unidirectional sound source, meaning it emits sound in all directions equally. The sound energy close to the source will be high. However, the sound wave expands like a sphere around the sound source, which means the energy has to be distributed across the surface of that sphere. This surface area increases with the square of the distance, which is why a sound source becomes louder when one moves towards it. Estimating the distance requires some reference material, because once the brain recognizes the sound source, it references it to past experiences. We basically learn the correlation between the distance and pressure level for specific sounds. In the case of speech for instance, one knows the expected volume when someone stands right next to you, or much further away. Combined with the difference in sound when someone raises his voice, this gives a good indication of the distance. However, the human brain can also use some other information to determine the distance. The timing of the early reflections also gives an indication. Imagine standing in the middle of a large room, with a sound source close by. The direct signal will arrive very quickly, but the first order reflections have to travel very far before they get back to you. This timing difference tells you that the sound source is much closer to you than the walls. If the sound source is far away instead, the first order reflections will not be far behind by the time the direct signal arrives.

2.1.3. The Geometric Model

The oldest application of acoustics in virtual environments is architectural acoustics, where architects needed ways to predict the acoustic properties of their design before constructing it, and adjusting their design accordingly. The acoustics model would estimate the acoustic properties of the design based on a virtual representation of the room or building. Although the wave equation would be able to model the acoustics in a given virtual environment very accurately, it appears to be extremely difficult to solve. It would require a lot of work to set up all the boundary conditions, and then the computation would take many days even on modern CPUs, which is unacceptable. Instead of dealing with the computational challenges of the wave equation, solutions rely on a different model of sound propagation: geometric acoustics. The geometric model uses a simplified representation of wave propagation, by borrowing concepts from geometrical optics. The main simplification comes from the fact that sound is no longer represented by a continuous 3D pressure field, but by rays which are emitted from a sound source and are propagated through the environment. By tracing these rays through the environment, one gets an idea of how the sound can reach any location. Geometric acoustics does not model the actual sound wave, but it aims to find propagation paths from a sound source to the listener through the environment, and then later tries to auralize these propagation paths. The same setup as in Figure 2.1 is depicted in Figure 2.3, this time modeled using rays. The main advantage of this model is that it is easier to compute. Although there are many interactions between rays and the environment, each interaction itself is very easy to handle. Since this model is borrowed from the field of geometric optics, a lot of techniques can be taken from the work done in computer graphics. Particularly the fact that rays all need to be propagated through the environment independently makes the geometric model attractive, because tracing rays can be implemented very efficiently on the graphics card.

Although lighting is modeled similarly, there are some things that do not transfer directly from optics to acoustics. The main differences between acoustics and optics are in absorption and propagation speed. In real life, a lot of light is absorbed when it hits a surface, and only some of it is reflected. For that reason, when tracing rays in a lighting model, one can usually stop after a few interactions because the remaining energy in the ray is minimal. With sound that is not the case at all. Very little energy is absorbed by walls, which means propagation paths can still be relevant even after a hundred interactions. Secondly, the speed of light is so high that in almost all applications it is perfectly valid to just assume a light ray arrives at its destination instantly. The speed of sound in air is roughly 330m/s. That sounds fast, but if one were to assume all sound rays reach their destination instantly, all sounds would arrive at the listener at the same time and there would be no perceived reverberation. The

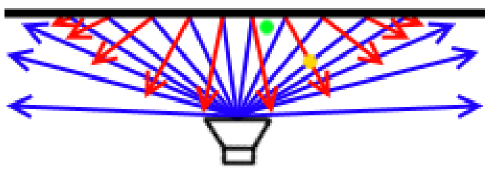


Figure 2.3: The same setup as in Figure 2.1, this time as it is modeled in geometric acoustics.

techniques from geometric optics can thus be a good start, but they need to be adapted to account for the increased number of interactions and the fact that sound takes time to travel somewhere.

Assumptions

Solving the latter is very easy if a constant speed of sound is assumed. As long as the path length is known, the delay between a sound being emitted from a source and receiving the sound at the listener position can be computed. Practically all geometric acoustics models make that assumption, disregarding the fact that the speed of sound in air is actually dependent on frequency, pressure, temperature and humidity. This travel time does mean that once a propagation path has been found, its audio cannot be played back yet. It has to be buffered to account for the time taken for the sound propagation. Secondly, the assumption of sound traveling in straight lines is only really valid for high frequencies. As soon as the wavelength of the sound is the same order of magnitude as objects in the environment, geometric acoustics cannot model the correct behaviour. It has to be noted that the same assumptions apply in optics, but they are more valid there because the wavelength of visible light is so short. Thirdly, the geometric model can only be used for point sources. The only way to model a surface emitting sound (or light) would be to create a lot of point sources on the surface. Since each source emits a lot of rays and each individual ray has to be traced, it is undesirable to use more than one point source to model a sound source. If required the sources can be made directional by limiting the directions in which rays can be cast, or by changing the initial energy in each ray depending on its direction.

Finally, a big disadvantage of using rays is undersampling. A continuous pressure field can not be completely modeled by infinitely small rays, no matter how many rays are used. There will always be parts where a propagation path should be found, but it is missed because there was no ray emitted in exactly the right direction. When working with rays, one has to accept the fact that some propagation paths will be missed. Since the probability of an infinitesimal ray hitting a point in space is zero, the listener has to be defined as a volume. To find propagation paths, each ray is checked for intersections with the listener volume. Figure 2.3 shows two listener volumes. The orange listener volume receives both the direct signal and a reflection off the wall. The green volume however receives nothing, caused purely by undersampling of the rays. There are some alternatives to rays that try to solve this issue which will be discussed briefly at the end of Section 3.3.3

Sound Wave Interactions

When a sound wave hits an object, the object can redirect the wave in several ways. These effects can all be modeled by the wave equation, but since this report will deal with these interactions using the geometric model, they will be explained in terms of their geometric description. The four main types of interactions are explained below, illustrations of these effects can be found in Figure 2.4.

Specular Reflections The most basic result of a sound wave hitting geometry is the fact that the wave is reflected off the surface, like light hitting a mirror. This is called specular reflection 2.4a. Nearly every geometric acoustics model uses specular reflections, some even rely on no other interactions at all while still obtaining usable results.

Diffraction Diffraction allows a sound wave to propagate around a corner. Low frequencies can travel around a corner more easily than higher frequencies. Since a ray hitting a corner can go anywhere it is necessary to spawn a number of new rays to cover the newly accessible area, as demonstrated in 2.4b. Creating many new rays each time is expensive though. Alternatively, diffracted propagation paths can

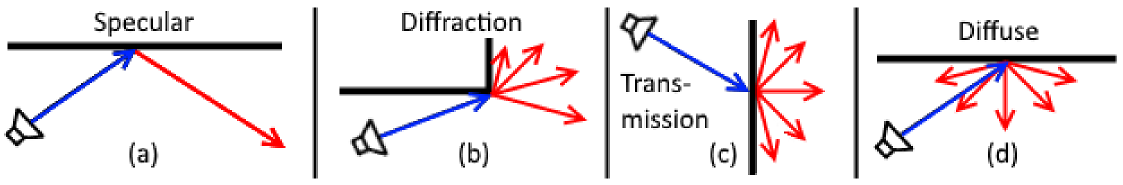


Figure 2.4: Ray interactions in the geometric model.

be calculated explicitly by finding the shortest path from the sound source to the listener along all the edges. This technique is referred to as the 'Uniform Theory of Diffraction' (UTD) [2]. If a model relies just on the 'bounces' of specular reflections, some areas might be very hard to reach for a wave in a complex environment, resulting in a discontinuous sound field. Diffracted propagation paths are a valuable addition in such a case because they allow a wave much easier access to certain areas. If there is no direct line of sight between the listener position and the sound source, the shortest propagation path will almost always be a diffracted path, not a reflected path. For that reason, diffraction is also valuable for sound source localization.

Transmission When a sound wave hits a wall, said wall can start to vibrate and transmit the sound to the other side of the wall. This allows a listener to hear a sound source behind a wall. In practice, high frequencies do not transmit very well because of their low energy and because the natural frequencies of walls are much lower. For that reason, transmitted sounds are often very muffled. Unfortunately, to geometrically model this properly a new source of rays should be created on the opposite side of the wall. Since the entire wall surface is vibrating, there should ideally even be a number of new sources on both sides of the wall because a point source is not enough. This makes transmission expensive to model using classic ray tracing techniques.

Diffuse Reflections The last type of wave redirection is diffuse reflections. Diffuse reflections allow a sound wave to reflect in all possible directions after hitting a wall. Each single ray has a very low energy, but these types of reflections do contribute noticeably to the reverberation. Because there are so many of them, they create a very high echo density which is generally perceived as pleasing. The naive way of implementing diffusion in the geometric model is to create a new source of rays for every single interaction. That is simply not doable in real time for anything more than very low order diffuse reflections, which is why they are never modeled entirely in real time scenarios. Most systems either do not model them, greatly reduce the number of new rays, interpolate them from the other propagation paths or generate them artificially.

Absorption Apart from just redirecting the sound, the environment can also absorb the sound. Different materials can have different absorption characteristics, often very frequency dependent. For a sound wave that hits multiple surfaces, these effects accumulate with each interaction.

2.2. Computer Graphics

The field of computer graphics is concerned with displaying 2D or 3D information. This is generally done on the graphics processing unit (GPU), which is built to output 2D arrays of color values, known as pixels. Drawing a virtual scene is called rendering. However, a GPU does not have to output its results to the computer monitor, it can also store results which is why it can also be used for other tasks. The GPU is designed for performing the same calculation many times, since it has many processing cores which all run in parallel. As mentioned in Section 2.1.3 already, this is what makes it useful for tasks like tracing rays through the environment. Each ray has to undergo the exact same process, so all shader cores are assigned a ray and perform the same calculations simultaneously.

2.2.1. Geometry And The Graphics Pipeline

The term geometry usually refers to the branch of mathematics that is concerned with describing and working with shapes. In computer graphics, the term geometry is used to describe all the objects in the virtual environment. This geometry is made up of interconnected points in space. These points are referred to as vertices. Three vertices are combined to a triangles and triangles are combined to make up a surface that describe an object. The graphics card uses a vertex shader to move the vertices that make up the entire virtual world, which allows for camera positioning and perspective. The scene is then rasterized to fragments, and in a fragment shader each fragment is assigned a color based on the color of the triangle, the lighting model and any other influences.

2.2.2. Occlusion

The term occlusion is often used in computer graphics. It refers to the fact that some objects may block the line of sight to other objects. When drawing objects on the screen, they need to be drawn

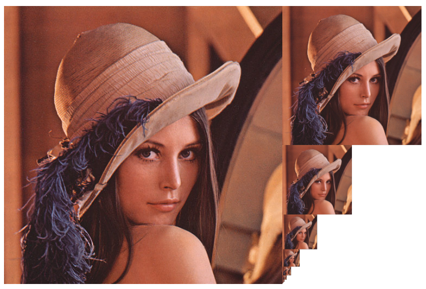


Figure 2.5: An example of a mipmap.

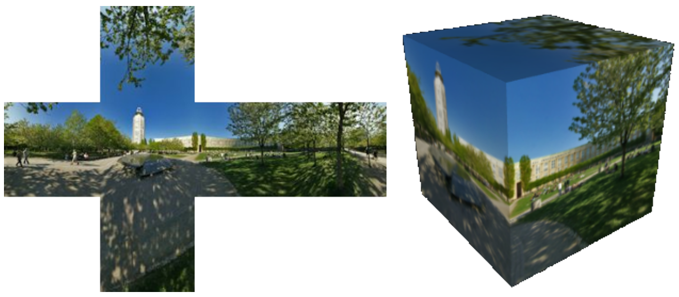


Figure 2.6: An example of a cubemap from Emil Persson's collection [3]

back to front so that the objects in the back cannot overlap objects in the front. This is done by storing the depth of each fragment, so they can be sorted when drawing the final image. However, the term occlusion is also relevant in acoustics, as objects can also block the sound propagation from a source to the listener. Just like with computer graphics, where partly transparent objects are possible, some objects may also partially let through sound which makes things even more complex. In the context of this report, the term 'occluded' simply means 'not visible or audible because something else is in front'.

2.2.3. Useful Techniques

There are two particular techniques in the field of computer graphics which will turn out to be very useful, namely mipmaps and cubemaps. To make their application easier to understand in later chapters, their concepts are explained here.

Mipmap

A graphics card has its own memory where it can store arrays of pixels, known as textures. These textures are basically digital images that the graphics card can read and write individual pixel values from or to. A mipmap of a texture is a collection of smaller versions of said texture. They are all combined into a single image as illustrated in Figure 2.5. The graphics card can generate these mipmaps very quickly, by simply recursively taking the average of groups of 4 pixels until the last mipmap level contains just a single pixel, which contains the average color of the entire texture. The advantage of this is that one can request a particular mipmap level from the GPU, which means a very small texture can be returned if a high resolution is not required. The entire mipmap structure only takes up 50% more storage space than the original texture, since each mipmap level is only a quarter of the size of its parent.

Cubemap

A Cubemap is a combination of 6 square textures that together form a cube. When these textures align nicely, they form a continuous image when viewed from inside the cube. These are often used for skyboxes for instance, because when viewed from the inside they can be made to look like a sky dome. The image in Figure 2.6 shows an example. Remember that textures are not just for reading, the GPU can also write to textures. That means that it is also possible to render to a cubemap. By positioning the camera somewhere in the virtual scene and rendering six square viewpoints with a $90°$ field of view, a cubemap can be built that contains everything that is visible from that location in every direction.

3 Previous Work

3.1. The Field

The field of acoustics is a long-studied subject, with reports on architectural acoustics originating from the era before Christ. Many famous scientists made contributions to the field including Pythagoras, Aristotle and Galilei. However, real advances in the physical understanding of acoustics and mathematical models only started in the 19th century. Modern architectural acoustics started with the work of W.C. Sabine, who through countless measurements derived an equation that could estimate the reverberation time of a room based on its volume and absorption coefficient. Currently, the behaviour of sound waves is fairly well understood, and the acoustic wave equation can model it very accurately. However, it remains notoriously difficult to solve so its applications are limited to research problems where very high accuracy is required. Since the wave equation is of little practical use in acoustics modeling situations where speed is a concern, a lot of work has gone into simplified versions of it which are easier to compute, as well as alternative ways of capturing the behaviour of sound.

In architectural acoustics real time performance is not a necessity. Providing an auralization in a couple of fixed positions in the environment after some computation time is generally sufficient. However, this thesis is concerned with acoustic models that can provide acoustics in real time in virtual environments. The main application for real time acoustics models is in computer games, where the player is free to move anywhere at any time. The acoustics model should be able to respond to any changes in the geometry or player position very quickly in order to provide a natural sound. To achieve real time performance either the accuracy or the interactivity has to suffer. High accuracy is not necessary in games though, since a convincing sound is good enough. Because games are often pushing the boundaries of what they can deliver in terms of visual quality, there is very little computational power reserved for audio processing. Even though there are powerful geometric models available that can deliver accurate acoustics in real time, these models are usually not employed in modern games. There is a discrepancy between what techniques are available and what is actually being used, mainly because of the little processing power available for sound and because more accurate sound is not considered worth the cost. That is why the currently employed techniques are discussed separately from the state of the art techniques.

3.2. Currently Employed Techniques

There are some games that employ smart techniques for their acoustics, but by far most games use one of the following three simple techniques. In increasing order of versatility they are baked reverb, convolution reverb and algorithmic reverb.

3.2.1. Baked Reverb

Baked reverberation is by far the easiest and simplest technique to use. Reverb is generated beforehand, using some type of reverberator, and the result is baked into the audio files. The advantage is that a high fidelity reverb generator can be used and that no computation is required while running the game. There are some significant disadvantages though. This reverb is completely static, in that it will not change depending on the environment or listener position. Sound source localization, if present at all, is usually achieved through panning the entire audio signal in the stereo field to the required position, but this also pans the reverberation tail which is not what would happen in real life. In short, baked reverb is very cheap and easy to implement, but offers no versatility or realism at all.

3.2.2. Convolution Reverb

Convolution reverbs use pre-recorded impulse responses (IRs) and convolve those with the audio in the game. IRs can be seen as a long list of incoming reflections, each of which has a different delay, phase and signal level. A binaural IR can be used which has a different IR for each ear so that a stereo effect is achieved. Impulse responses can be measured in real environments, which can capture the fine details of the reverberation in a specific position in that environment very accurately. That means the resulting reverb is very detailed and accurate, but only in a virtual recreation of said environment, and only at the location where the IR was measured, looking in the same direction. In practice games only use at most a handful of IRs to reproduce some different acoustics in different areas. These are very general IRs which are not related to the actual geometry of the environment at all, but are just selected for sounding good and conveying a certain atmosphere. Because of that, the resulting reverb is completely inaccurate and does not help with source localization. Of the three simple techniques, convolution is computationally the most expensive. It can be made cheaper by transforming the audio and the IRs to the frequency domain, where convolution is reduced to a pointwise multiplication instead. Despite being faster, the forward and inverse fast Fourier transforms (FFT) are still relatively expensive operations, and the cost increases for longer reverberation times.

3.2.3. Parametric Reverb

Parametric reverberators are algorithms that generate reflections that are not based on actual propagation paths. They usually employ some sort of audio feedback network where the input audio is delayed and added back onto itself. With a good network topology and careful tweaking the result will sound like reverberation, providing an increasing echo density and a natural decay curve. However, since the reflections are generated, there is no accuracy whatsoever when it comes to directionality. It is computationally very cheap though, and it has a lot of flexibility because the delays and feedback gain can be adjusted in real time to achieve different reverberation times from the same network. This is usually combined with trigger volumes, where the environment artist sets the reverberator parameters for each area in the environment, and the game applies them depending on the player’s location. This requires some work by the artist though, especially since changes in the environment might also ask for changes in the reverberator parameters.

3.2.4. Occlusion

Sound source occlusion remains a difficult problem to solve. Determining how much of a sound source should be audible when the environment blocks the direct signal is not trivial. A pole blocking the direct line of sight would not affect the sound all that much, since the sound wave will be able to propagate around it, especially at lower frequencies. A thin wall might block all the sound on the other side, or it could let some through, let alone the fact that there might be a path around the wall that makes a sound source audible on the other side. Most sound engines avoid this problem by simply dividing the space up in predefined volumes, where they specify for each volume from which other volumes sounds should be audible. With careful tweaking this will create acceptable results, but it is not exactly accurate and also not easily capable of handling unforeseen changes in the environment.

3.2.5. Localization

For sound source localization, most games rely on a simple stereo or surround mix. A surround mix allows for reasonably accurate localization based on the direction and volume, a stereo mix is more limited. The tiny timing differences between ears as well as all the filtering from different directions are generally neglected. For a surround speaker setup that is not necessary, but more often than not games are played on a stereo speaker setup or headphones which would greatly benefit from these things, but they are generally not found in modern games.

3.2.6. Setting The Bar

The most advanced technique applied in a popular modern game that I could find would be Rainbow Six Siege, where a combination of baked reverb and convolution reverb is used. These are accompanied by a node grid that allows the game to find the shortest diffracted propagation path from a sound source to the player in real time. That enables the game to find the direction from which a sound should be heard when it is occluded, so players can follow their ears to find a sound source. The game can also propagate sounds through walls, with the expected muffling effect. There are destructible environments, so the grid is updated in real time to adapt to changes in sound source occlusion [4]. The solution provides very convincing sound source occlusion effects, but the reflections inside a room are not modeled at all. The convolution reverb they use for the reverberation is not based on IRs computed from the actual geometry, and it is only applied to some sound sources. All others use baked reverb because the convolution is too expensive to use on all sources.

3.3. State Of The Art Techniques

In terms of accuracy and localization, the results achieved by the previously discussed models leave a lot of room for improvement. However, much better acoustics would be possible if more computational power was reserved for audio. The techniques discussed below can all run in real-time if the required processing power is made available.

3.3.1. Pre-Computed Impulse Responses

A rather obvious extension to the convolution reverb would be to use a high quality acoustics model to generate IRs based on the actual geometry. These IRs are computed in many different locations and stored in a grid. When playing the game, the most appropriate IR is selected based on the listener position, as proposed by Astheimer [5]. That way the reverb is based on the actual geometry, and when the player is on a grid node, it is as accurate as the selected model. Positions between grid nodes could for instance use interpolated IRs. However, this technique requires an IR grid for every sound source. To achieve some level of accuracy, the IR grid needs to be dense and contain binaural IRs in many directions at every grid node. A dense 3D grid of nodes, with at every node multiple IRs for each ear in different directions for each sound source would inevitably require a lot of memory. A major limitation is that these sound sources cannot move, nor can the geometry change. All the propagation paths would change, invalidating the pre-computed IRs. Additionally, convolution is computationally relatively expensive, especially for longer reverberation times. Although the results are very good for static scenes, the memory and processing power requirements of such a system are problematic.

3.3.2. Wave Equation Models

As mentioned before, the wave equation is a 3D second order partial differential equation that describes the pressure field. Solving it numerically takes days, so it is far from usable in real time applications. There are however some solutions that are based on extreme simplifications of the wave equation which can produce results in real time. A major advantage of wave equation solutions over the previously discussed IR grid and the geometric models that will be considered in section 3.3.3, is that its result is a continuous pressure field. All other solutions compute the acoustics just for a single location and orientation. This allows multiple players to move freely inside the 'solved space', because a result is available everywhere simultaneously.

Simplifying The Wave Equation

The first step towards approximating the wave equation is discretizing the space. Instead of trying to find a continuous pressure field for the entire room, a much simpler equation is solved for small communicating volumes, usually referred to as waveguides. In each waveguide, a time domain difference model is solved. The resolution of the grid allows for a variable trade-off between accuracy and cost. Figure 3.1 shows a horizontal slice of waveguides in a 3-room setup, processing an excitation by a single pulse. The fact that waveguides can be solved in parallel on the graphics card allows for real time implementations, as shown by Röber et al [6]. Their ten year old hardware managed to achieve real time results with a reasonable grid resolution for low frequencies. The issue with this model is that the update frequency needs to be high enough to model high frequencies, which requires a rather large increase in computational power if the entire audible spectrum has to be modeled. Furthermore, unlike

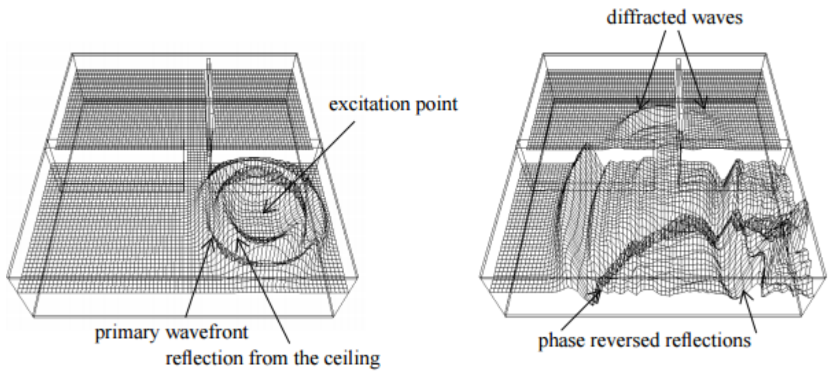


Figure 3.1: Illustration of a horizontal layer of waveguides in a 3D waveguide setup.

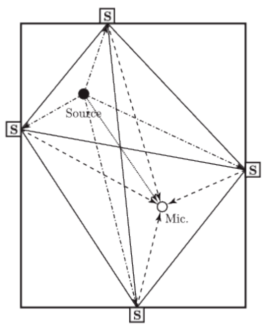


Figure 3.2: The scattering delay network setup, using a single waveguide per wall.

geometric models, larger environments increase the computational cost significantly because the grid size increases with the volume of the environment. Changes in the environment would be difficult to handle, as the waveguide grid would have to be extended with a new grid for newly accessible areas. Although promising and very accurate, this model requires a lot of processing power to compute only the lower end of the frequency range, which would be inadequate for practical use by itself.

The waveguide setup can be simplified further though, as is shown by De Sena et al [7]. Instead of filling the entire space with a grid of waveguides, they place only a single waveguide on each wall. These waveguides communicate, allowing sounds to reflect from each wall to the next. The setup is depicted in Figure 3.2. Propagating sound through just a few waveguides is manageable in real time. A sound source and listener can move through the room and simply act as moving waveguides that communicate with all other waveguides to respectively send and receive sound. Instead of modeling the actual wave propagation, this system approximates the possible propagation paths of all the reflections, where the resulting path lengths and directions are roughly correct. This makes this system a hybrid between a wave equation model and a geometric model. By moving the waveguides along the wall (using the image source method, see section 3.3.3) so that they represent first order specular reflections correctly at all times, the early reflections are still quite accurate, and the higher order reflections increasingly inaccurate. The resulting audio is moderately accurate, but still very much defined by the geometry of the enclosure. They claim that perceptually the quality is similar to the image method, while being one to two orders of magnitude faster to compute. Like the image method though, the system is limited to enclosed areas and cannot account for other objects in the room or changes in the room geometry.

3.3.3. Geometric Models

There are a plethora of geometric solutions for acoustics available. They all differ mainly in which effects they model and what clever optimizations they introduce. There is however no system that can model all kinds of interactions in real time for long propagation paths. The most famous geometric acoustics technique is called the image source method [8]. Although not efficient enough to run in real time, it is the basis for many other techniques, like the one discussed previously in the wave equation section. The image method only models specular reflections. Instead of reflecting rays off of walls, it mirrors the sound source in walls. When drawing a straight line from the mirror images to the listener position, each propagation path will have the correct length and direction. The mirror images can be mirrored again to find higher order reflections. This process is illustrated in Figure 3.3 for two first and second order reflections. Unfortunately, not every mirror image leads to a valid propagation path, which means each result needs to be checked. This performance of this check scales rather poorly with the reflection order, which is why the image method is not great for long reverberation times. The biggest limitation though is that it only works for completely closed rectangular rooms, which is very restricting.

Ray Tracing Solutions

Classic ray tracing works as follows: First, a big number of rays are emitted in random directions from the sound source. Each ray is checked against each triangle in the scene to see if they intersect. Of

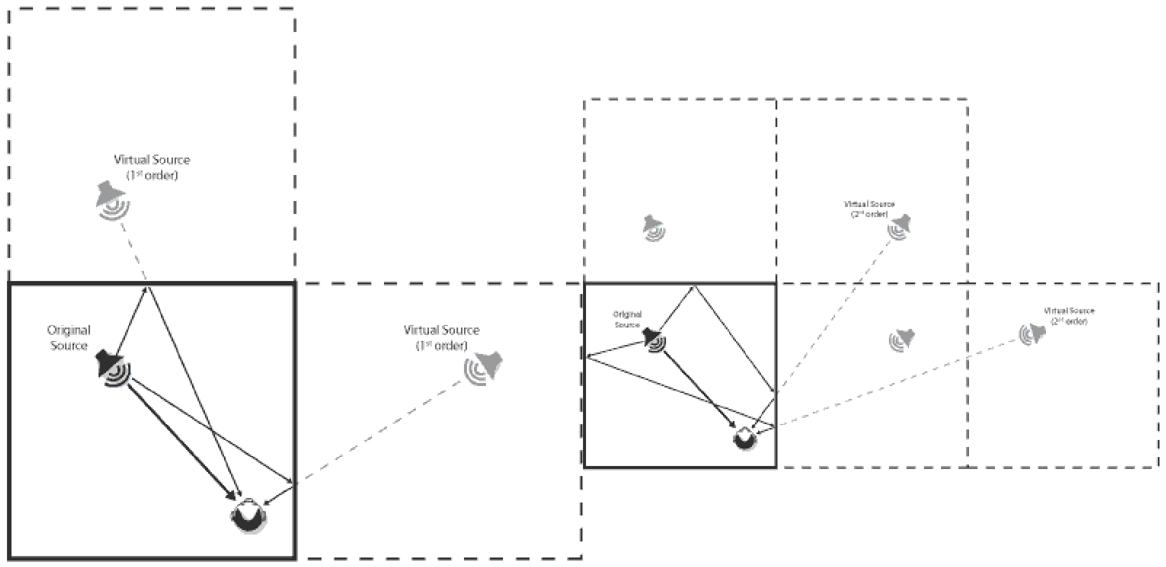

Virtual Source (1st order)
Original Source
Virtual Source (1st order)
Original Source
Virtual Source (2nd order)
Virtual Source (2nd order)

Figure 3.3: Example of two first and second order reflections using the image method, taken from Qeiroz et al [9].

all the intersected triangles, the one closest to the ray origin is selected. Depending on which types of interactions are being modeled, one or more new rays are spawned at the intersection location. The ancestors of each new ray are stored with the ray, so the entire path can be rebuilt later. Additionally, every ray has some energy associated with it, which is influenced by the distance traveled and the absorption of the surfaces that have been hit. This process repeats itself until some stop condition is met, like a minimum energy level in all the rays. Every ray is also checked for intersection with the listener volume, in order to find which paths eventually reach the listener position. This process works well, but it is computationally very expensive for complex environments and high ray densities, especially when each interaction can spawn hundreds of new rays. Classic ray tracing is thus not viable as a real time solution. The techniques presented below are optimizations or simplifications of ray tracing that allow for faster results.

There are many solutions that try to offload the real time computation with a precomputation. For instance Funkhauser et al [10] compute all the propagation paths beforehand and store them in a lookup tree. During runtime, the player location is used to find possibly relevant propagation paths in the tree. A spatial subdivision is used to facilitate fast lookup times. Although this lookup tree requires not nearly as much memory as an IR grid, it suffers from the limitation that all precomputations have, which is that they cannot easily facilitate moving sound sources or changes in the geometry.

A notable ray tracing solution is proposed by Taylor et al [11]. They suggest a system that models specular, diffuse and diffracted propagation paths for early reflections. They use UTD for diffraction and a stochastic ray tracer for diffuse reflections, which creates just a few new rays in random directions for each interaction. In order to achieve the desired real time performance, they process the rays a couple of interactions at the time. By doing this, they can incrementally update parts of the impulse response, reusing the previous IR for the other parts. This works well, but it cannot account for fast changes in the IR because it takes two seconds before the whole IR is updated. Despite these optimizations, the entire reverberation tail is still to complex to compute entirely, which is why late reverberation is provided by an algorithmic reverberator. By fitting a curve to the energy levels of the discovered early reflections, a reverberation time estimate is found which is used to tune the reverberator.

Another way to speed up ray tracing is through guided ray tracing, also proposed by Taylor [12]. The idea is to use a very large listener volume and trace a small number of rays to find potential propagation paths. The ray tracing is then repeated for a smaller listener volume, with rays emitted in the general direction of the rays that found propagation paths in the previous run. The result is relatively fast because in total less rays are emitted. However, some propagation paths are missed because the ray density in the first step is low. Such a system only works well for geometry with large surfaces.

One can also decide to use fairly few rays, and fill in the missing information by generating artificial reflections. Schimmel et al [13] propose to use the image method to find some specular reflections, and enrich the sound by constructing additional diffuse paths. These paths are not based on true propagation paths, but because they are generated based on real propagation paths, the final reverberation still sounds reasonable accurate. Most people in the target audience for games would not be able to tell the difference, which makes it an attractive option. This technique is usually referred to as diffuse rain, and can be applied to enrich the results of any system with interpolated diffuse reflections. The solution proposed by Schimmel suffers from the same limitations as the image method though.

A creative and quite different approach to the geometric model is introduced by Kapralos et al [14]. They call it sonel mapping, a derivative of photon mapping. Instead of tracing rays, they trace 'sonels'. These behave like rays, but for every interaction with the geometry, they 'splat' some energy onto that geometry. This sonel map is then used in a Monte-Carlo ray tracing step from the listener position, where the splatted energy is recalled and used to estimate the energy of the diffuse soundfield. The sonel map is created for several frequency bands, so frequency-specific absorption can be modeled. The paths of the sonels themselves also contribute propagation paths to the auralization. By randomly selecting different effects for each interaction, the computational cost is kept low while still enabling long propagation paths. Unfortunately, the performance of the whole system leaves some room for improvement, because their implementation struggles to deliver real time results for anything more complex than a small room.

A very powerful and complete solution is introduced by Schissler et al [15]. They propose a system that can handle large numbers of sound sources by using backwards ray tracing and clustering nearby sound sources. By emitting rays from the listener position instead, propagation paths to any number of sound sources can be found. Additionally, by clustering soundsources that are close together and far away from the listener, the audio processing is reduced. Such sources would have very similar propagation paths anyway, so processing them as one saves having to deal with them individually. Finally, they also manage to model Doppler shifting to some extent, an effect that is normally never associated with geometric models. Their performance is impressive, with long reverberation times computed explicitly, even for large scenes with many sound sources. This system is a good example of what is possible in real time if all the computational power of a decent computer is made available for acoustics in virtual environments.

Ray Alternatives

There are alternatives to using rays that can overcome the problem of undersampling. Instead of casting infinitely small rays, one can also cast volumes. For acoustics, beam tracing [10] and frustum tracing [11] can be used. The idea is similar in both cases. From the sound source, neatly aligned beams are cast in all directions. This covers all possible space, so there is no undersampling. The disadvantage is in processing the interactions. When a beam hits a surface, it has to be recast, just like a ray. However, often a beam will hit only part of a surface, or multiple surfaces. In that case, the beam has to be split up and several parts will have to be traced further or recast. This is a bit more complex than with rays. Frustum tracing tries to find the middle ground between rays and beams by casting triangular beams and simplifying the interactions. Instead of splitting the frustum up, it will simply continue as one frustum according to what the most influential surface in the interaction would dictate. That sacrifices some propagation paths for easier computation.

4

System Overview

4.1. Goal of This Thesis

What the previous chapter shows is that there are some really accurate models that are either computationally too expensive or too limited to use in interactive applications. There are also some techniques that are very cheap but they are wildly inaccurate. There will always be a trade-off between accuracy and computational power. However, the goal of this thesis is to design an acoustics model that has the most perceivable advantages of an accurate model while minimizing the additional computational cost. These advantages are:

1. Improved sound source localization.
2. Truly interactive acoustics which:
   - Adapts in real time to changes in sound source or listener position.
   - Handles changes in geometry instantly, even completely destructible environments.
3. Automatically realistic acoustics in any environment, because the acoustic parameters are modeled from the geometry in real time.

Interactive acoustics means a pre-computation is out of the question, so everything has to happen in real time. All of this has to run on consumer hardware, leaving some computational power to do other things like rendering graphics and handling game logic.

4.2. Building Blocks

Apart from being easier to compute, the geometric model has another advantage. Since it does not make any use of the actual audio, the search for propagation paths and their auralization can be completely disconnected from each other. That means that these two processes can run asynchronously, giving the designer full freedom to decide on how often the propagation paths need to be updated. It also means that finding propagation paths and auralizing them are two completely separate problems. This fact allows the system to be split up into separate components. This section will give an overview of the main components of the proposed system, detailing their functions and how they are connected.

4.2.1. Prerequisites for Source Spatialization

The final goal of this entire system is to output a convincing sound reproduction in a given virtual environment with a given sound source and listener position. As discussed in Chapter 2.1, this means finding out how a sound wave can propagate from the sound source to the listener. To auralize a propagation path, it is required to know the total distance, the direction from which it arrives at the listener and how the interactions with the environment absorb the sound. The distance is necessary for calculating the time it takes for the wave to arrive, and for calculating how much sound energy will reach the listener. The direction is required for localization. The absorption is necessary for controlling the energy level of the wave. Since this absorption is frequency dependent, it is also used for coloring the sound.

What Will Be Modeled

A key observation that forms the underlying foundation of the proposed system is that the acoustic response of an environment can be split up into a number of separate parts, as was already shown in Figure 2.2. Each part makes its own contribution to the perception of the acoustics. For that reason, each part will be modeled separately, so that the main characteristics of each part are captured properly, without spending unnecessary computational power on any of them. The reverberation of an environment will be split up in a direct signal, the early reflections and the late reverberation.

Direct Signal

The direct signal has an important role in sound source localization because, if a direct signal path is possible, it will be the first sound that arrives at the listener position. It is thus crucial to model the direct signal's distance and direction accurately.

Early Reflections

The early reflections are those propagation paths that arrive at the listener shortly after the direct signal. They only interact with the environment a couple of times and do not cover a very long distance. Their main contribution is a sense of distance, as well as enabling even more detailed sound source localization. The transition between early reflections and late reverberation is not clearly defined, but for performance reasons this system only considers first order propagation paths as early reflections. They perceptually contribute the most and higher orders would require much more computation for relatively little gain. These early reflections will be based on first order specular reflections and first order diffraction.

Late Reverberation

The late reverberation finally is made up of all the complex and long propagation paths that sound waves can take through an environment. These propagation paths go through a large number of interactions with the environment before reaching the listener. They are characterized by a high and increasing temporal density and an exponentially decaying signal level. This is the 'reverberation tail' of a room, and it is most noticeable in large indoor areas like cathedrals. The main contribution of the late reverberation to the perception of acoustics is the sense of size of a room and its absorptive qualities. Carpet and curtains for instance have a noticeable effect on the late reverberation, because they absorb high frequencies. The number of reflections (and thus the computational cost) in the reverberation tail is very high, while their individual contribution to the sound is hardly perceivable. For that reason, the accuracy of the individual late reflections is less important. Instead of finding propagation paths for them, the late reverberation is generated artificially. Although individual paths are not modeled, the reverberation tail as a whole still needs to sound as expected given the environment. The most important characteristics of this tail are its length and its frequency response. Chapter 3 already mentioned Sabine's equation, which stated that these characteristics are controlled mostly by the absorption of the environment and the size of the enclosure.

4.2.2. Preparation

For all of this to work, the absorption characteristics of the environment need to be added to the geometry somehow. Most 3D modeling applications allow a designer to assign material properties to surfaces. Originally, this is intended for indicating how each surface responds to incoming light. This material definition can however easily be extended to include sound absorption parameters. When processing the geometry, these surface characteristics are attached to each vertex of a surface as vertex attributes. For sound absorption, this information is supplied as a RGB color value, which can store absorption coefficients for three different frequency bands.

4.2.3. Summary

In summary, the system is thus split up into two distinct parts: A geometry processing stage which is in charge of finding propagation paths and late reverberation parameters, and an auralization stage which is in charge of processing the source audio according to the results of the geometry processing. The geometry processing stage gathers all its information as often as is deemed necessary. Whenever changes in the listener position, a sound source position or the geometry occur the geometry processing could be redone. Again, one is entirely free to specify when and how often the information from the geometry stage is updated.

In order to auralize a sound source in a virtual environment, the information that is required from the geometry processing is:

- A possible direct signal
  - Path length
  - Direction of the incoming wave
- All possible first order specular reflections
  - Path lengths
  - Direction of the incoming waves
  - Absorption of the materials at the reflection points
- All possible first order diffraction paths
  - Path lengths
  - Direction of the incoming waves
- Late reverberation parameters
  - Absorption of the materials in the environment
  - Volume of the enclosure

These elements will all be discussed in Chapter 5. The auralization stage is running continuously, providing an uninterrupted audio output signal using the latest available propagation path information. To spatialize the sound source according to the information from the geometry processing, the audio processing needs:

- The source audio
- A way to control the signal level of each reflection
- A way to delay each reflection so it arrives at the right time
- A way to filter each reflection based on the absorption
- A way to give a sense of direction to each reflection
- A way to generate fitting late reverberation
- A way to mix all these audio streams together

All the audio processing that is required for the auralization will be discussed in Chapter 6.

5 Geometry Processing

5.1. Basic System Setup

The challenge in this chapter is to extract all the required information that is required for auralizing a sound source from the environment. For finding propagation paths, the geometric model will be used. This comes with the assumption that sound travels as rays, which means it cannot propagate around objects, nor does it propagate differently at different frequencies. Acoustics systems using the geometric model are often implemented on the graphics card because ray tracing is easily parallelizable. The system proposed here will also use the graphics card, but not for classic ray tracing. Instead, a 'line of sight' approach is used, where the main idea is that if an object is visible from a certain viewpoint, a sound wave must be able to travel from that object to the viewpoint. the graphics card is used to render the scene and derive the acoustic properties of the environment from the geometry as seen from the viewpoints of the sound source and listener position. That makes this system quite different from existing geometric models. This chapter will discuss the different elements separately, and then conclude by showing how they all come together.

5.2. Direct Signal

As mentioned in the introduction already, the first arriving wavefront plays an important role in localizing a sound source. If there is a clear line of sight between the sound source and the listener position, the first arriving wavefront will be the direct signal that reaches the listener without any geometry interactions. For that reason, the direct signal is important to get right, but also relatively easy to get right because there are no interactions involved. The main challenge is thus to find out whether or not there is a clear line of sight between the listener and a sound source. Once that has been determined, the problem is reduced to auralizing a sound source at a given distance and direction from the listener.

5.2.1. Line Of Sight

Finding out if a real life object is visible or occluded in a real life scenario is very easy. If one simply looks around in all directions, one can quickly determine if said object is visible or not. The approach used in the virtual case is exactly the same. By rendering the environment and all sound sources to a cubemap (see Section 2.2.3) from the listener position, everything that is visible from the listener perspective is drawn to the cubemap. If there is a clear line of sight from the listener to a sound source, said sound source has to be visible in the cubemap.

When using a graphics language like OpenGL this solution is very easy to implement. For every object that is being drawn, one can request how many fragments of every object ended up as pixels in the cubemap. One or more pixels means that the source is visible and that there should be an audible direct signal. This is already the solution to the occlusion question. What makes this technique very flexible is the fact that any object can be drawn as a sound source. If the sound source is for instance a vehicle, one could draw the entire vehicle and use that for the occlusion testing. However, it is also possible to use for instance a cube of any size, which could potentially save a lot of triangles being

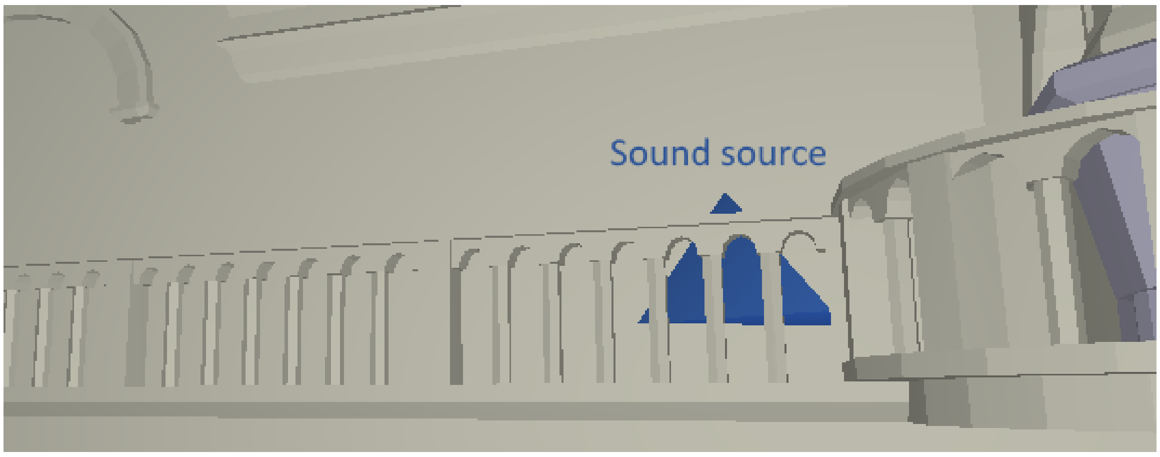


Figure 5.1: A sound source being tested for occlusion.

drawn. The object used for the sound source occlusion testing can be picked based on the size of the source, the desired accuracy and the complexity of the environment. Figure 5.1 shows an example where a sound source is rendered as a simple blue pyramid. This pyramid consists of just 4 triangles so it is very cheap to draw. It is partly occluded by the balcony bars, but because the pyramid is big enough its line of sight will always be detected when it is moving along the balcony. A much smaller pyramid could also be used, if one wanted the direct signal to be intermittently blocked by the balcony bars if the sound source were to move along the balcony. In this case, the pyramid's size was decided on so that a continuous direct signal would be heard.

5.2.2. Preparing For Auralization

Rendering a single cubemap is enough to determine the line of sight to all sound sources simultaneously, which is convenient. For every sound source, a simple yes or no flag can be stored for the existence of a direct signal. The only other information that is required for auralizing the direct signal is the path length and the direction of the incoming wave. These are readily available since the source and listener positions are known, and the propagation path is a straight line between the two.

Note that the problem of finding the direct propagation path is actually performed in the 'wrong' direction. Sound travels from a source to the listener, but in this case the propagation path is found by looking from the listener to the source. The technique could be reversed, but that would require a cubemap render from every sound source which is evidently more computation with no added benefit. Separating the actual audio from the search for propagation paths allows for these kinds of optimizations.

5.3. Early Reflections: Specular

With the direct signal, there was no interaction between the geometry and the sound. However, all other possible propagation paths involve some sort of interaction at the surfaces in the geometry. The first one that will be considered is specular reflection, where the sound bounces off the surface it hits.

5.3.1. How it Works

This time a propagation path consists of two parts, the part from the sound source to the wall and the part from the wall to the listener. Note that the number of specular propagation paths per sound source can be much higher than just the single direct signal path. To find the specular reflections, first all the surfaces that will reflect the sound wave are found and then each reflection is traced to see whether or not it will reach the listener.

The first step is easily solved. When the environment is rendered to a cubemap from the sound source position, this cubemap contains all the surfaces that will receive the sound waves from the source. All the surfaces in this cubemap will thus generate a specular reflection. The question is now whether or not each individual reflection will reaches the listener. This question consists of two parts:

1. Is there an unobstructed path from the reflection point to the listener? (the occlusion problem)
2. Does the reflection go in the direction of the listener? (the direction problem)

These two questions will be answered in the following two sections.

5.3.2. The Occlusion Problem

So far, cubemap renders have been a great help in solving occlusion problems, and they will be used again here. The question of whether there is an unobstructed path from the reflection point to the listener is the same as asking if there is a clear line of sight between the two. The easiest solution would be to take each reflection point in the sound source cubemap and render the scene from there to see if the listener is visible, just like with the direct signal. However, rendering the entire scene again and again for each pixel in the sound source cubemap is a lot of work. Just like in the direct signal case, it is much more efficient to perform this check in the opposite direction instead. The question is then changed to 'which of the reflection points in the sound source cubemap are visible from the listener?' That question can be answered from just a single cubemap render from the listener position. The visibility information in the two cubemaps now somehow has to be combined, to find out which parts of the environment can both receive the sound wave and reflect it towards the listener. The left image in Figure 5.2 shows an example setting. Everything that is visible in the sound source cubemap is red, and everything that is visible in the listener cubemap is blue. The surfaces that are purple are the ones that are actually relevant and worth investigating. Notice how a reflection hitting a surface that is only visible from the sound source can reflect in the direction of the listener, yet never reach it.

Finding Relevant Geometry

Conveniently, the problem of finding geometry that is visible from two different viewing positions is not new. In computer graphics, this is a very common issue when considering lighting. Finding out whether or not some part of the geometry is illuminated by a light source or in shadow is exactly the same challenge. When viewing the geometry as seen from the light source, every visible surface must be illuminated and every occluded surface must be in shadow. The challenge is now to transfer this information to the camera viewpoint. The solution that is used is called shadow mapping [16]. Shadow mapping will be used here to find the relevant parts of geometry that could contribute a specular reflection.

Shadow Mapping

Shadow mapping works by measuring distances and comparing them. For each bit of geometry that is visible in the sound source cubemap render, its location is stored. The Euclidean distance between this location and the listener is computed. Next, the scene is viewed from the listener in the direction of the stored location. The listener will see a surface there, and the distance to that surface is computed. If the two distances are the same, then the sound source and the listener are looking at the same surface. If they are not, then the surface that the sound source could see is occluded when viewed from the listener. The image on the right side of Figure 5.2 shows the principle. The distance between

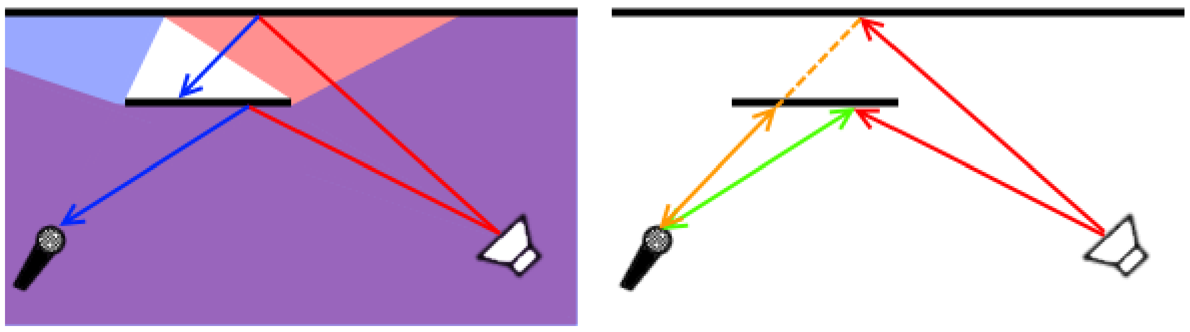


Figure 5.2: Shadow mapping

the lower reflection point and the listener is equal to the length of the green arrow. But the distance between the upper reflection point and the listener is much larger than the length of the orange arrow. The conclusion is that the listener can see the lower reflection point, but not the upper.

Implementation

To implement shadow mapping, all this distance information has to be available somehow. Recall from Section 2.2.2 that the graphics card computes depth values for each fragment, which describe the distances between the camera plane and the viewed fragments. These will be used here. To start, a cubemap is rendered from the listener position. Instead of drawing the colours to the cubemap, the depth is stored in each pixel. The scene is then rendered from the sound source perspective, which has access to this depth cubemap as well as the location of the listener. For each fragment it draws, the shader computes the distance between the fragment and the listener. Next, it reads the depth value from the listener cubemap in the direction of this fragment. After transforming this depth value to a distance, it can compare the two distances to know if the current fragment is visible by the listener. In principle the first question is now answered. However, in practice shadow mapping is not without faults, so some additional measures are necessary.

Shadow Mapping Issues

Comparing the distances directly does not work in all cases. The distance taken from the listener cubemap is not the distance to the exact location of the fragment, but the distance to the pixel in the cubemap that is closest to that position. That means the resulting value might be slightly higher or lower than the actual distance. The result is a lot of noise in the shadow map, because some distances will be slightly different. This phenomena is called shadow acne. To counter this, the two values are not required to be exactly equal. Instead, the fragment is considered visible if its distance from the depth map with a small bias added is larger than the actual distance. A bigger bias gives better results, up to the point where it allows the listener to see through walls. The bias value thus has to be tweaked according to the dimensions of the environment. Shadow acne cannot be completely prevented. When the resolution of the depth map is very low at a location close to the sound source, this can still cause some artifacts. In the worst case scenario, this will mean a specular reflection will be missed.

A second problem arises from the fact that the triangles in the geometry have no thickness, they are infinitely thin. If the source and listener both see a different side of the same triangle, shadow mapping will indicate that they both see the same surface. Similarly, if walls have no thickness, reflections can pass through corners as is shown in Figure 5.3. The little green line extending the blue arrow indicates the bias. The bias prevents shadow acne but at the same time causes these potential problems. The best solution for this is to make sure that it is impossible to see a surface from both sides at once. Infinitely thin walls do not exist in real life, so as long as all walls in the virtual world also have two separate sides this problem does not occur. That does unfortunately mean that the number of triangles that make up the environment goes up. Another slightly less elegant option is to check the orientation of the surface from both viewpoints to make sure that they both see the same side of the triangle. It

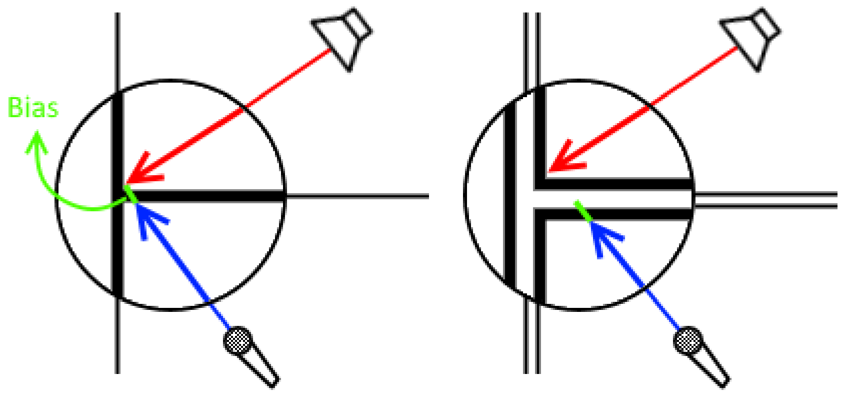


Figure 5.3: Single walled geometry allows specular reflections to pass through walls at corners.

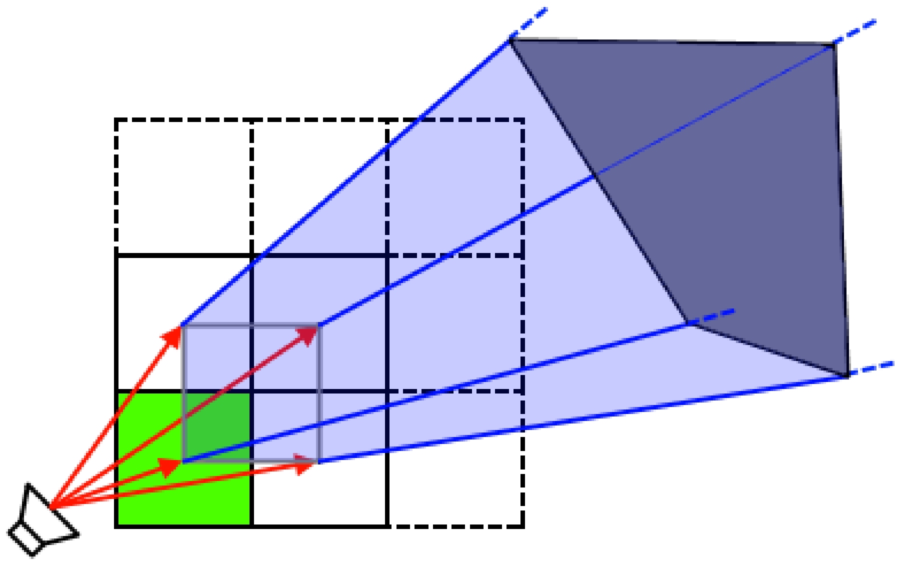


Figure 5.4: Reflecting a beam off of four pixels.

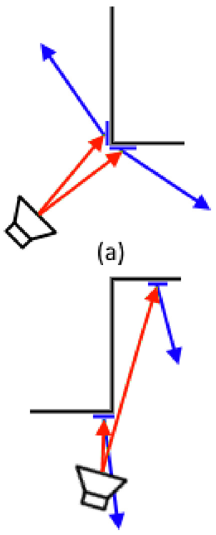


Figure 5.5: 2D illustration of corner cases where beams should be discarded

is less elegant because it results in similar artifacts as shadow acne at the triangle edges, which can not be prevented with a bias this time.

5.3.3. The Direction Problem

It is now known which reflections can potentially reach the listener. However, it is still unknown if the reflection is actually going in the right direction. Finding the direction of the outgoing reflection vector is very easy if the incoming direction vector V and the surface normal N are known. The reflection vector R is found from equation (5.1):

$$R = V - 2N(V \cdot N) \tag{5.1}$$

So how does one determine if this infinitesimal ray hits the infinitesimal point in space that is the listener position? In classic ray tracing this was solved using a listener volume instead of a point. A different solution is used here. As was shown at the end of Chapter 3, there are alternatives to casting rays that do not suffer from undersampling. Instead of casting a reflection ray, a reflection beam is cast. This beam is a volume, and for each beam it is checked if it contains the listener position.

Beam Casting

It would be possible to cast a beam for every pixel in the sound source render that passed the shadow mapping check. However, that could result in both overlapping beams as well as volumes of space that are not covered by a beam, because different pixels can have different surface normals. Instead, beams are cast from the centers of groups of four pixels. The size of such a beam is still a single pixel, but now neighbouring beams align perfectly with each other. Figure 5.4 shows what this looks like. The green pixel is the pixel that is currently considered. The red arrows are the viewing vectors from the sound source to each pixel. The blue lines are reflected vectors off of each pixel. The beam is constructed from these four reflection vectors. In order to do this, the world positions and normals of neighbouring pixels need to be available. The information of neighbouring fragments is not accessible in a fragment shader, so before this render process a preparation render pass is done that stores the fragment normals and positions to a texture. The reflection vectors in each of the four pixels are found from Equation 5.1, using the vector from the sound source position to each fragment as the viewing vectors. The resulting four reflection vectors are used to construct the reflection beam.

The final step is to find out whether or not the sound source is inside the four faces of the reflection beam. To do this, the beam is not defined by four faces, but as four planes. The listener's position is checked against each plane. If it is located on the 'inside' of each plane, which is the side that contains the beam, then it must be inside the beam. Each plane is defined by the normal vector of two reflection vectors. First the angle between the two reflection vectors is computed, because a positive or negative angle would change the direction of the normal. This angle determines the order of the two reflection vectors in the cross product, which results in the desired normal vector. If the location check against all four planes is positive, a specular reflection has finally been found.

Beam Casting Issues

In principle, a beam is cast for every pixel that is marked as relevant in the shadow mapping stage. However, there are some cases where reflecting a beam is undesired. Figure 5.5a shows how looking at an outer corner would produce a very wide beam. Similarly, if the sound source were looking at the inside of a corner, the resulting beam would immediately fold onto itself and produce a wide inverted beam. Neither of these describe a proper first order specular reflection. To remove the wide beams that occur when the surface normals of neighbouring pixels vary too much, the angles between the normals are computed as $\theta = \arccos(N1 \cdot N2)$. If said angle exceeds $\frac{\pi}{8}$, the beam casting for this pixel is aborted. Figure 5.5b shows how edges in the scene could also invalidate neighbouring pixels. To find these edges in the geometry, the depth values of each fragment are compared. If they differ more than some threshold, the beam is also discarded. This threshold depends on the resolution of the cubemap, since the depth differences between neighbouring pixels increase with a decrease in resolution. This can result in false edge detection for walls that are almost parallel to the viewing vector. The pseudo-code for the entire process from shadow mapping to beam casting for a single fragment is available in Appendix A.

An example of the output of the shadow mapping stage and edge and angle detection is shown in Figure 5.6. The right image shows the scene as seen from the listener position. The cubemap is rendered from the sound source, the listener position would be roughly in the middle of the left most face of the cubemap. Notice how the balcony casts clear shadows onto the ground and wall behind it. The conclusion is that the parts that are in shadow are not visible from the listener position. For all the white pixels in the cubemap a reflection beam will be cast.

5.3.4. Preparing For Auralization

The information that is required to auralize a reflection is the path length, the direction from which it approaches the listener and the absorption coefficients of the reflection point. The path length and direction are trivial since the sound source, listener and reflection point position are all known. The absorption at the reflection point is taken from the geometry.

5.4. Early Reflections: Diffraction

Diffraction allows a sound wave to propagate around a corner. Again, the propagation path consists of two parts. From the sound source to the corner, and from the corner to the listener. The way this will be modeled is very similar to the specular reflections case. Corners that are visible from the sound source will diffract the wave, and corners that are visible from the listener can send a diffracted wave

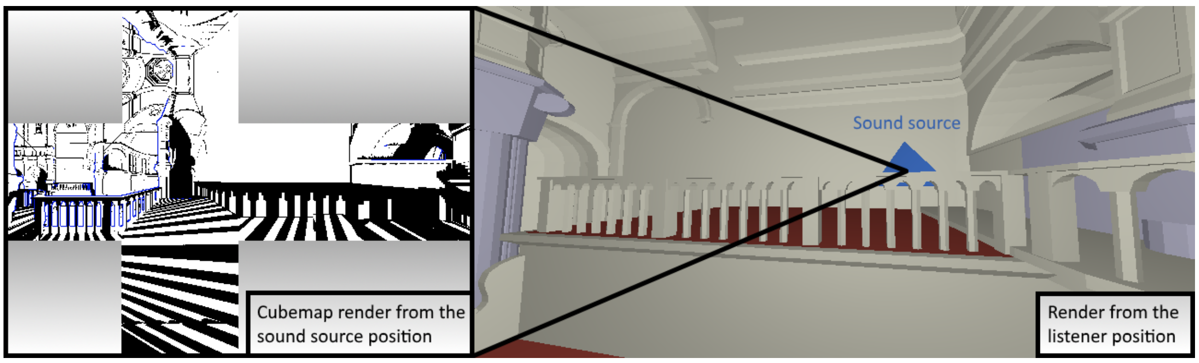

Cubemap render from the sound source position
Sound source
Render from the listener position

Figure 5.6: Result of a shadow map with edge detection, showing the diffraction edges in blue.

to the listener. The key problem is again an issue of visibility: Which corners are visible from both the listener and the sound source? Because the sound wave can take any direction after hitting the corner, there is no need to trace the path from the corner to the listener to see if it goes in the right direction. The answer is always yes provided that the listener can see said corner.

5.4.1. Finding Corners

Finding corners is in fact very easy. Recall how edge detection was used in the beam casting stage. The same method will be used here. When rendering the environment from the sound source position, the depth values of each fragment are available. Finding an edge is as simple as defining a threshold between neighbouring depth values. The depth of the current fragment is compared to the fragment above it and to its right. If either one exceeds the threshold, a diffraction edge is found. The depth information of neighbouring fragments is again not available in a fragment shader, so that needs to be prepared beforehand. That is however all there is to finding corners. The two red arrows in Figure 5.7 show the big depth difference at an edge which allows the system to detect the corner.

5.4.2. The Occlusion Problem

The question that remains is which corners are visible by both the source and the listener. This can be solved using shadow mapping, by exploiting a limitation of the technique. Consider Figure 5.7 again: the sound source sees the fragment right at the edge of the wall. The listener obviously cannot see that surface. However, if the sound source requests this visibility information from the listener depthmap using shadow mapping, it turns out that the listener can apparently see that fragment. This happens because the bias that is used to counter shadow acne actually allows the listener to see around the corner slightly. This shadow mapping error turns out to be very useful, since it allows the system to solve the occlusion problem even though technically the listener cannot see the surface that is being considered. First order diffraction is thus fairly trivial to accomplish using just a simple edge detection algorithm and shadow mapping to see if a detected edge is visible from both the source and the listener. In fact, the solution is very similar to the initial setup of specular reflections. Consider Appendix A again. Instead of aborting on line 9, a diffraction path is found. That shows both how simple the diffraction path detection is and how well it integrates with the specular reflection detection. The blue pixels in the cubemap in Figure 5.6 show detected corners which will diffract the signal towards the listener.

5.4.3. Preparing For Auralization

When sound is diffracted around a corner, the absorptive qualities of the surface do not really affect it. However, low frequencies diffract easier than high frequencies. That means that one would hear the lower frequencies around a corner more than the higher frequencies. Instead of using the absorption coefficients of the surface, absorption values that achieve this behaviour can be used. Values of 0.0, 0.085 and 0.38 were used for low, medium and high frequencies respectively to model this effect. The only other required information is the path length and the direction of the incoming wave, which are easily available since the sound source, listener and corner locations are known.

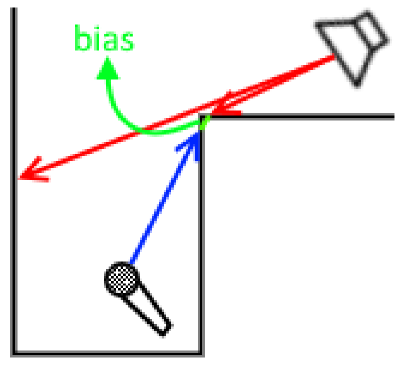


Figure 5.7: Edge detection and occlusion check through the wall for diffraction

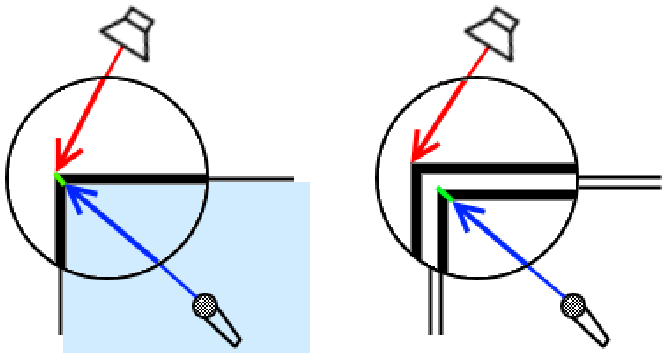


Figure 5.8: Single walled geometry allows diffracted propagation paths to pass through walls at corners on the wrong side.

5.4.4. Diffraction Issues

The process of finding diffraction edges results in a lot of propagation paths, many of which are very similar in length and direction. Imagine an L-shaped room with the source in one extremity of the room and the listener in the other. A couple of specular reflections might be found there. However, in the concave corner of the room, a vertical row of diffracted paths is found, which could easily contain over 100 pixels at a 128x128 cubemap resolution. These 100 propagation paths all have to be individually auralized which would completely drown the specular paths. Additionally, this auralization would take up a lot of computation with hardly any gain, since the path length and direction of all these propagation paths is nearly the same. For that reason, only the shortest diffracted propagation path is considered, all the others are simply discarded. This still allows for a sound source to be audible around a corner allowing the player to find the shortest path to the source, without spending a lot of processing power on the auralization.

Another problem with this solution to diffraction stems from the infinitely thin geometry. Recall that for specular reflections, this could in some cases lead to a reflection making it through a wall. The same thing happens for diffraction, but much more frequently. The problem is that the shadow mapping exploit used for the occlusion problem also works from the wrong side of the wall. Figure 5.8 illustrates the problem. Anywhere in the blue area the listener would receive a diffraction signal, which is clearly unintended. The solution to this is again to use double-walled geometry, to make sure that a surface is never visible from both sides.

5.5. Late Reverberation

So far, the system has computed exact propagation paths for a direct signal, first order specular reflections, and first order diffraction. To achieve a complete reverberation tail, the higher order reflections will also need to be modeled in some way. Unfortunately, finding exact propagation paths for higher order reflections becomes expensive very quickly. Since the amount of propagation paths increases tremendously, treating them all separately would put too much strain on the auralization stage as well.

To overcome these limitations, the late reverberation will not be modeled by finding the individual propagation paths. Instead, the reverberation will be generated so that the reverb tail has the desired properties. The human ear cannot distinguish the individual reflections in a dense reverberation tail so as long as the main characteristics of the late reverberation are right, the resulting reverberation is convincing [17]. The goal is thus to determine these characteristics and model late reflections accordingly.

5.5.1. Late Reverberation Characteristics

The most important characteristics of a reverberation tail are its length and its frequency content. Both of these depend on the environment. According to Sabine's equation, the length of the tail is proportional to the volume of the space and inversely proportional to the absorptive qualities of the materials in the environment. The equation results in a T_{60} value, which denotes the time between the arrival of the direct signal and the reverberation becoming inaudible. To calculate the expected reverb tail length, the volume and the absorption of the enclosure are thus necessary. The frequency content of the reverberation tail also depends on the absorption of the environment. The contribution of the absorption accumulates with every reflection, and in enclosed areas the reflection order can reach values of 100 and higher. For that reason, distinctive frequency-dependent absorption is most audible in the late reflections. The absorption coefficient of the environment will need different values for different frequencies.

The late reverberation consists of very long propagation paths that undergo lots of interactions with the environment. Since these propagation paths themselves will not be modeled, a decision has to be made on which surfaces will influence the late reverberation and which will not. Without knowing the propagation paths, there is no definitive answer to this question. However, it is fairly safe to assume that a sound wave that reaches the listener will also reach all the surfaces close to the listener. These surfaces will thus be considered for both the absorption and the volume parameters.

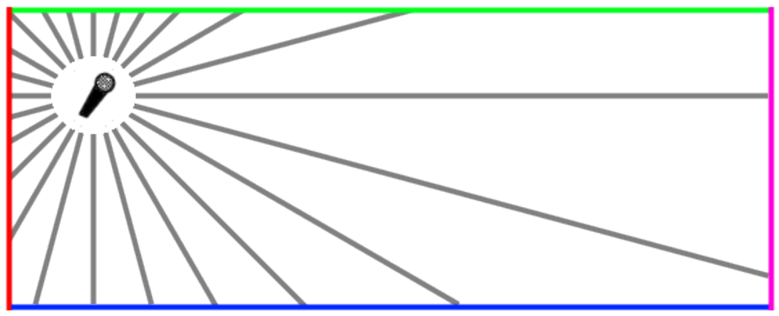


Figure 5.9: Sampling the absorption coefficient from the listener position.

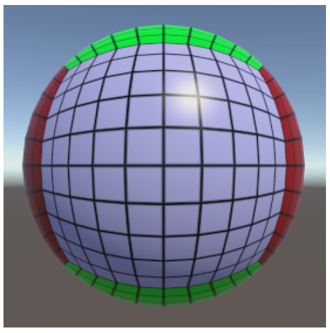


Figure 5.10: Mapping a cubemap to a sphere. [18]

5.5.2. Finding The Absorption Coefficient

As was proposed in Section 4.2.2, the absorption coefficients of each surface are stored with the geometry as RGB color values, where each color channel represents a frequency band. This solution will make finding the absorption coefficient very simple. The absorption values of the environment as seen from the listener position can be made visible by rendering a cubemap again, but this time drawing the absorption coefficients of each surface instead of the actual colors. The result is an absorption map of the environment around the listener. Large surfaces will make a large contribution and small surfaces a small contribution, exactly as expected. However, the reverberator requires a single absorption value for each frequency band, so absorption cubemap has to be processed. The easiest way of arriving at a single value is by simply taking the average of all the pixels in the cubemap. Incidentally, there is a very convenient way of finding the average color of a texture: a mipmap (See Chapter 2.2.3). By requesting the highest mipmap level of the cubemap, a single pixel is returned which has the average color of the entire cubemap. The red, green and blue values of this pixel are the absorption coefficients of the low, middle and high frequency bands.

Corrections

There are two issues with this technique. The first is caused by the fact that surfaces far away contribute far less than surfaces nearby. Although that sounds reasonable at first glance, it is not really desirable. Very high order reflections will probably hit each surface multiple times, which means that a surface that is further away can contribute to the absorption coefficient just as much as an equally large surface nearby. Consider the example in Figure 5.9. It shows the listener in a room with four different walls. The gray lines represent pixels in the cubemap. The red and pink walls are equally large, yet the red wall contributes 8 pixels to the absorption coefficient and the pink wall just two. If the absorption coefficients of these two walls is very different, the final absorption coefficient estimate would vary wildly based on the listener position, whereas the expected result is that the reverberation tail is mostly the same within a single room, regardless of the listener position. Luckily, it is fairly easy to compensate for this fact. Each pixel represents a surface area. As is evident from the figure, pixels on a wall nearby represent a small surface area, and pixels on a wall far away a much larger area. Each pixel will thus be weighted based on the distance. The surface area of a pixel actually grows with the square of the distance. By multiplying each absorption value in the cubemap with the square of the distance, this is solved. However, the final averaging is still based on the sum of all the pixel values divided by the number of pixels. That no longer results in the average, since the value should also be divided by the sum of all the weights. To solve this, the weights of each pixel are also drawn in the cubemap. When requesting the mipmap, the average weighted absorption is returned, as well as the average weight. Dividing the former by the latter results in the desired absorption coefficient.

The second issue is caused by the fact that a cubemap does not cover each direction equally accurately. Ideally one would map the scene as seen from the listener to a sphere. Instead of rendering a cubemap, a spherical projection could be used which would solve that issue. However, textures have to be square and a spherical projection has a circular result. Taking the mipmap of a square texture containing a circle does not result in the average value of the circle, but of the entire square which is not the desired result. Spherical projection is thus out of the question. Instead, the cubemap will be used, but it will be mapped to a sphere. Figure 5.10 shows what happens when mapping a cubemap to a sphere. Notice how the pixels in the center of the face of a cube contribute more than the pixels in the corners of the faces. This effect has to be compensated for to correctly regard each viewing direction equally. This is done by weighting each pixel with the cosine of the angle between the viewing vector to this pixel in the cubemap and the vector to the center of the face it is on. Since these weights are constants as long as the resolution does not change, they can be precomputed and stored in a texture for quick access.

The two weights introduced in the paragraphs above are combined, so that the final absorption coefficient is derived as follows:

$$\overline{a} = \frac{\sum(a \cdot d^2 \cdot cos\theta)}{n \cdot \sum(d^2 \cdot cos\theta)}$$

With a being the three frequency band absorption values in each pixel, d the distance, theta the angle between the current pixel and the center of its cube face and n the total amount of pixels.

5.5.3. Estimating The Volume Of The Environment

Finding the volume of the room as seen from the listener position is actually a very similar issue as the absorption coefficient. Every pixel in a cubemap represents a beam from the listener to the environment. A beam has a volume, which means it is possible to accumulate the volumes of all the beams and arrive at a volume estimate. This is very crude, since a small object nearby can significantly reduce the volume estimate of the entire room. However, without predefining the volume or making things significantly more complex, this method provides usable results.

The volume estimation works as follows: Every pixel in a listener cubemap render represents a surface area. The center of this pixel has a known distance from the environment to the listener. By drawing lines from the four corners of a pixel to the camera, an obtuse pyramid is created. The volume of such a pyramid can be calculated as $\frac{A \cdot h}{3}$, where A is the surface area of the base of the pyramid and h its height. Finding h is trivial, since this is the distance from the camera plane to the pixel. The surface area of the base depends on the distance, much like in the absorption case. Figure 5.11 shows this construction in for the pink pixel. Assuming the pixels are square (which means the vertical and horizontal resolutions of the cubemap are the same), the volume of this beam is $\frac{w^2 \cdot h}{3}$. To find w, the angle theta between the pyramid edges is used. This angle is constant for all pixels, as it is the $90°$ field of view divided by the horizontal resolution. When assuming the pyramid base is perpendicular to the camera, this angle can be used to find the length of the base of the pyramid as $2 \cdot tan\theta \cdot h$. This assumption is not entirely correct since the pyramid is now no longer obtuse. However, it does not change the volume estimate of the beam since the equation for the volume is the same for regular and obtuse pyramids.

The Final Volume Estimation

This technique suffers from the same problem as the absorption, where a cube is being mapped to a sphere. For that reason, the same correction is applied using the cosine of the angle between the viewing vector to each pixel in the cubemap and the vector to the center of the face it is on. To find the final volume estimate, the highest mipmap level of this cubemap is requested again, which contains a single pixel with the average volume of all the beams in the cubemap. Multiplying this value by the resolution of the cubemap returns the final volume estimate.

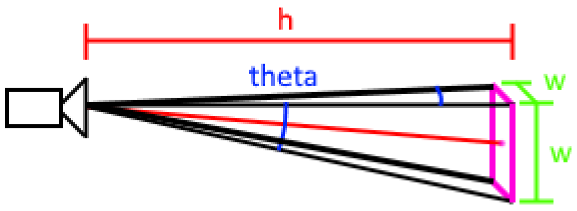


Figure 5.11: Constructing a beam volume for a pixel

5.5.4. Using Sabine's Equation Outdoor

Sabine's equation is being used for finding the T_{60} of a room, which is proportional to its volume and inversely proportional to some absorption coefficient. However, this equation is only valid for enclosed spaces. An outdoor area would have an enormous volume and very little materials that could absorb sound energy, which would indicate a very long reverberation time, whereas in reality the T_{60} of outdoor areas is very short. To make Sabine's equation viable for use in any environment, the dissipation of sound energy to an open sky has to be captured in the absorption coefficient somehow. The more of the sky is visible, the shorter the T_{60} of the reverberation tail will be. Since a sound wave that travels towards the open sky will never be reflected back, the effect of open sky on the reverberation time is quite high. It can be modeled as infinite absorption.

Luckily, detecting open sky is not so difficult. Open sky simply means that there is no surface being drawn somewhere. The graphics card will assign a depth of 1, the maximum depth, for areas where nothing is drawn. By assigning maximum absorption to these pixels, the absorption estimate increases for each pixel of open sky.

5.5.5. Finding The Reverberation Time

The final reverberation time is found from a slightly modified version of Sabine's equation:

$$T_{60} = 0.005 \cdot \frac{\sqrt{V}}{a + l}$$

Where V is the volume estimate, a the average absorption of the three frequency bands and l the loss factor for the open sky. In the original equation, the absorption coefficient is supplied in Sabins, which is not a very convenient unit. That is why the equation is modified to work well with the absorption coefficients that are extracted from the geometry. By squaring the volume, the resulting T_{60} scales very well in different environments. High absorption values will always drop the reverberation time to nearly zero, just like being outside. The reverberation time also scales very nicely with the volume estimate, providing long reverb tails in large halls and short tails in small rooms.

5.5.6. Late Reverberation Signal Level

The transition between the early reflections and the late reverberation needs to be smooth. The smoothest solution would be to take the volume of the last early reflection and use that as input for the reverberator. However, with all the work done thus far, it is actually possible to do something better, that does not rely on the early reflections that were found. There are possible setups where no propagation paths are found, yet one would still expect to hear late reverberation. Using first order specular reflections and diffraction is fairly limited in terms of solving sound source occlusion problems. Higher order reflections would be very useful in that regard, but they are expensive to model and thus not available. What has not been considered yet though is first order diffuse reflections. A diffuse reflection models the behaviour of a wave that hits a wall and then reflects back in all directions. It is much like a specular reflection, but without the direction constraint. That means that taking the shadow mapping step that was used for specular reflections contains all the surfaces that the sound wave can use to propagate diffusely from the source to the listener. Auralizing all these diffuse reflections is out of the question simply because of their sheer numbers, but this information can still be used. It is suggested to use the shadow map to control the input gain of the signal of each sound source to the late reflections reverberator. That way late reflections for a sound source are heard even if no propagation paths were found at all, as long as there is enough overlapping geometry in the shadow map. Once again, taking the highest mipmap level of the shadow map cubemap results in value between zero and one, which is an indication of the amount of diffuse reflections that reach the listener. This value can be used to control the level of the signal going into the reverberator.

However, the transition still needs to be smooth. In addition to this diffuse model, the input gain is further determined by the distance between the source and the listener. For a propagation path the signal level was reduced by $\frac{1}{d}$. Using that value for the late reverberation would result in a lower than expected output level, since that equation is valid for an ever expanding sphere. In an enclosed room, the sound energy is not dissipating like a sphere since much of the energy is reflected back into the room. For that reason, a value of $\frac{1}{\sqrt{d}}$ is used, which sounds more like one would expect. Finally, the input gain is further reduced by an energy loss factor. When rendering the sound source cubemap, all pixels that see no geometry reach the far plane and are considered as open sky. A sound source that can see a lot of open sky will not contribute much sound energy to the late reverberation because most of it will have dissipated. That means that even though the listener may be in a highly reverberant room, the sound source just outside will not be very audible in the reverb tail compared to sources inside. Although only a single reverberator is used to model all the late reverberation, the input gain is calculated per sound source, so that each transitions nicely from their early reflections and each of them benefit from the diffuse model. The final input level to the reverberator is calculated as: $\frac{1}{distance} \cdot diffuse \cdot (1 - opensky)$, where each of the variables are values between zero and one.

5.6. Putting It All Together

A lot of steps were taken to model three kinds of propagation paths, as well as derive the parameters for the late reverberation. There is much overlap between the different steps, which is why they can be combined in just a few render calls. This section will describe how it all comes together. The output of each render stage is summarized in Table 5.1

5.6.1. The Listener Render

When rendering from the listener position, the desired information is:

- Direct signal occlusion
- Depth map for shadow mapping
- Absorption estimate
- Volume estimate
- Visible open sky

To solve the direct signal occlusion problem, an occlusion query is used. This draws the entire scene including the sound sources and returns its result directly to the CPU. However, when using placeholders for the sound sources, it is important to use a write mask when drawing them. These placeholders should not appear in the render as they can block propagation paths or they may not have absorption coefficients assigned to them.

The data for the other four items are only available in the graphics memory. The depth map can stay there, it is only used in the sound source render later. The absorption and volume data has to be made available to the CPU somehow. The absorption can be stored in a high precision RGBA cubemap texture. The RGB channels contain the weighted absorption coefficients in the three frequency bands, the Alpha channel contains just the weights. Requesting the highest mipmap level of that texture on the CPU is relatively fast since it only returns a single pixel. By dividing the RGB values of this pixel by the Alpha value, the absorption coefficients are found. A second texture is used for the volume estimate and the open sky. This texture has two color channels, which again require higher precision because the volume values can be higher than a regular texture can store. To find the volume estimate, again only the highest mipmap level of this texture is necessary. This single value is multiplied by the cubemap resolution to find the final volume estimate. To find the amount of open sky, the blue channel in this texture receives a value of one for each pixel that sees geometry. The clear color is zero, so open sky pixels will have a value of zero there. Taking the highest mipmap level of the blue channel returns the fraction of pixels that were not open sky. 1 minus that value gives a factor between zero and one that indicates how much open sky is visible from the listener.

5.6.2. The Sound Source Render

The sound source render consists of two stages. The first one is just to prepare the required information for the second stage. The first stage outputs two cubemaps, one containing the world positions and depths of each fragment, to be used for edge detection and beam casting. The second contains the surface normals of each fragment, also to be used for beam casting, as well as an open sky factor again.

These two textures will be used in the second rendering stage, so they stay in the graphics memory. To save memory, the world positions and normals can be normalized so they fit in a regular color texture. This does however result in some loss of precision when computing the final propagation paths lengths later. In practice that will mean that some reflections will arrive at exactly the same time, where they should have been slightly apart. For the final auralization of the late reverberation, a loss factor was necessary that compensates for the open sky visible from the sound source position. This information is stored in the alpha channel of the normals texture. When clearing the texture beforehand an alpha value of 1 is used. In the shader every fragment that sees a triangle sets the alpha value to zero. When taking the highest mipmap level of this texture's alpha channel, it returns the amount of visible open sky as a value between zero and one. The second stage is used for:

- Finding diffuse reflections (shadow mapping)
- Finding specular reflections
- Finding diffraction

Since shadow mapping is required for both the specular reflections and diffraction, this is the first step in this shader. Then the edge detection step is done, since that too is required for both. For every detected edge, the diffraction information is stored. For every non-edge, the surface normals are compared to determine if a pixel can reflect a valid beam. If so, the beam casting process is started to determine the specular reflections.

The output of the second stage is stored in three textures. The first one has 3 color channels. Each pixel that passes the shadow mapping test is assigned a white color. Next the edge detection is performed after which the diffraction edges can be determined. Each diffraction edge pixel gets a value of (0.0f, 0.085f, 0.38f) in the first texture, which is the absorption coefficient used for diffraction. Similarly, for each specular reflection the absorption value of the reflection point is stored in the same texture. All other pixels in the first texture are black. The second and third textures are mostly empty, they only contain data in the pixels where specular or diffracted propagation paths were found. One of them is high precision and has just a single color channel that contains the path lengths of each propagation path. If however low precision world coordinates were used, the path lengths can also be stored in the alpha channel of the third texture. The third texture has three color channels which contain the normalized direction from which the sound wave will reach the listener, again only in the pixels where a propagation path is found.

The diffuse reflections are used for the input gain in the reverberator. This value is made available to the CPU by requesting the highest mipmap level of just the red color channel of the first texture. Since this texture is mostly black and white from the shadow mapping stage, this is a good indication of the number of diffuse reflections. Getting the specular and diffracted propagation paths to the CPU is a little more challenging. They are stored in specific pixels in the textures, but the CPU does not know where exactly. The simple solution is to transfer all three textures to main memory and use the CPU to scan them completely to find the pixels that contain propagation path information. Both the transfer and the scanning are relatively expensive though, depending on the cubemap resolutions. A slightly cheaper solution could be to transfer just one of the three textures and scan that, and for each

	Output Textures	Texture Type
Listener render	Depthmap	1 channel UB
	Weighted absorption + weights	4 channel FP
	Corrected volume estimates + open sky	2 channel FP
Source render 1	World positions + depth	4 channel UB or FP
	Surface normals + open sky	4 channel UB
Source render 2	Shadow mapping + PP absorption	3 channel UB
	Path lengths	1 channel UB or FP
	Path directions	3 channel UB

Table 5.1: The three render stages and their output textures. UB = Unsigned Byte, FP = Floating Point, PP = Propagation Path

pixel that contains a propagation path request the corresponding pixels from the other two textures in the graphics memory. The best solution though is to write a third shader that can process the output textures and write a much smaller texture that contains just the propagation path information. This would be possible using an atomic counter to append the output buffer of this shader.

When extracting the propagation paths from the textures, the diffraction paths are recognized by their distinctive absorption coefficients, which is the only thing that can be used to identify them at this stage. This is important, since there are too many diffraction paths to auralize all of them. The path lengths of all diffracted propagation paths are compared to find the shortest one. All the others can be discarded.

5.7. Conclusion

In conclusion, the entire system needs just a single render pass from the listener and two from each sound source to find all the information it needs to auralize a direct signal, first order specular reflections and first order diffraction, as well as to model late reverberation based on the environment around the listener.

6 Audio Processing

All the computations thus far have resulted in propagation paths for each sound source and a number of parameters for the late reverberation. With this information, the sound sources can now be auralized. This chapter discusses all the steps that deal with the audio signals, from the sound source to the stereo output of the computer sound card.

The goal of the auralization is to spatialize all the sound sources. What that means is that their output signal has to be processed in order to make them sound like they are in the virtual space. To achieve this, their output signal has to be played back for every propagation path. Each of these signals will be delayed, filtered, attenuated and directionalized before they are send to the sound card.

It is necessary to stress again that the audio processing is completely independent from the geometry processing. Since the audio stream has to continue at all times, it runs in a completely separate thread from the geometry processing, game logic and rendering stages. This ensures that the audio stream will keep going regardless of the complexity of the game logic or the environment. For more complex environments, the propagation paths will simply be updated less often, but the sound output will be uninterrupted.

6.1. Audio Throughput

Before looking at the auralization, it is important to have some understanding of how an audio engine works under the hood. A sound source plays back a predefined audio file. In this case, this audio file will be mono, because the stereo information will come from the spatialization. The sample values that make up the audio signal are processed and then sent to the sound card for playback. However, processing individual samples is not very efficient. Instead, samples are processed in batches. The reason for this is that other programs also request CPU time. Preparing a big number of samples in advance allows the CPU to perform other tasks while those processed samples are playing. This batch of samples is referred to as an audio buffer. Larger buffers are better because they allow the CPU more time to juggle between its tasks, but the disadvantage is that a larger buffer results in a longer output latency.

6.1.1. Mixing Sound Sources

Every sound source will produce a number of audio streams, one for each propagation path. All these audio signals have to be mixed together. For that purpose a mixer has to synchronize all the sound sources and combine their output to a single stereo signal. For playback, each sound source simply takes the next buffer from its input audio, and plays it back for each propagation path. Each path requires different filtering, volume and direction, which they are processed for. Since propagation paths also have different lengths, these signals should be played back to the listener at different times. To facilitate this, the mixer maintains a long output buffer. Each sound source provides its audio streams to the mixer, and for each audio stream it will indicate how much that signal needs to be delayed. The mixer can then place each audio stream in the output buffer, shifting the signal backwards by the

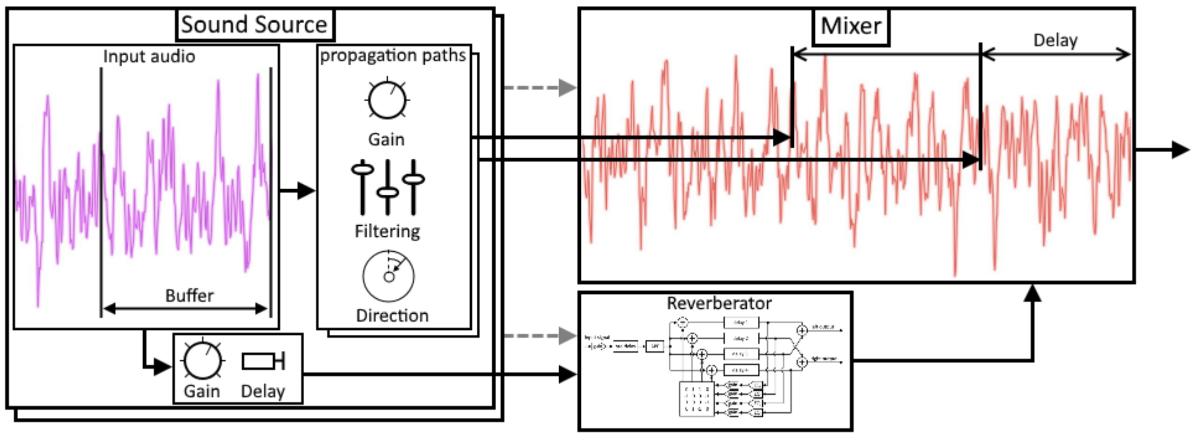


Figure 6.1: Signal Chain

number of samples it needs to be delayed for. Finally, the sound source also sends its input audio to the reverberator, after delaying and attenuating it to make it blend in with the early reflections smoothly. That is in short what the signal flow will look like. This process is illustrated in Figure 6.1.

6.1.2. Denormalization

A word of warning for anyone building their own audio engine: Even though input and output audio formats are usually 16-bit fixed-point samples, all the processing happens on 32-bit floats. Floating point values provide a lot of headroom because of their huge range, which makes it virtually impossible to clip the signal during processing. However, special care must be taken when dealing with floating point values in real-time audio applications. Floating point values have a mechanism where their representation changes when they become smaller than a specific value, in order to achieve improved accuracy for very small numbers. This is called de-normal representation. The problem is that de-normal values can take significantly longer to process on the CPU, up to 100 times more than normal floating point values. Since the threshold for the de-normal representation is so low that audio would not be heard at that level anyway, it is advised to make sure de-normal values can not propagate through the entire system. For that reason, all sample values are denormalized as soon as they are read from the input files. Additionally, the denormalization is repeated after each equalization or attenuation. This is done by simply adding the de-normal threshold value to every sample value, forcing it to be in normal representation.

6.1.3. Processing

For the processing of the audio there are many possible options. What will be presented here are the basic options that were used for this project, but different effects can be used in most cases. The processing that is required for each propagation path is:

1. Attenuating the signal
2. Filtering the signal
3. Directionalizing the signal
4. Delaying the signal

Attenuation

The signal attenuation depends on the path length. The sound energy of a wave is inversely proportional to the square of the distance. However, the system is processing sample values which are the amplitude of the signal. The amplitude of a signal is proportional to the distance, which means the fall-off is as easy as dividing each sample value by the distance in meters.

Absorption

The frequency content of specular reflections are influenced by the materials of the surface they hit. The absorption in three different frequency bands has been stored with this propagation path, which are now used to control a 3-band equalizer. By using the absorption values from the render, the audio can now be filtered to account for the absorption. For the purpose of this project, the equalizer curves are implemented as a simple time-domain filter, the design of which was taken from the 'Audio EQ Cookbook' [19].

Directional Sound

The propagation paths now have to go from a monophonic audio stream to a stereo signal. Going to stereo enables the option to properly give a sense of direction to the sound. First, the relative horizontal angle between the viewing direction of the listener and the direction of each propagation path is computed. A simple and computationally cheap solution would be to simply use this angle to play back the incoming audio at different levels in each ear. Although that will result in a reasonable distinction between left and right, it gives no indication of front or back and up or down. A much better sense of direction can be achieved by using a head-related transfer function (HRTF). A HRTF is a lookup table of impulse responses of the human head for each ear, measured for a lot of different directions. By supplying the audio and a vertical and horizontal angle, the audio signal can be convolved with the corresponding impulse responses. The result is a much improved directionality of the sound. It reproduces all the effects mentioned in the Localization section, Section 2.1.2. The HRTF used here for this project was kindly supplied by Hendrik Kayser [20]. From this collection of HRTFs, the 300cm anechoic inner ear IRs were selected, because most sounds would probably come from relatively far away. Room reflections from the HRTF would be undesired as they are already computed in this system, which is why the anechoic HRTF is used. This HRTF was measured on a dummy head which gives good results for an 'average' head, but exactly how well it works varies from person to person. The whole HRTF lookup table only takes up 5MB worth of memory. The anechoic HRTFs are very short, only 4800 samples per impulse response, which keeps the computational cost relatively low. To further speed up the process, the HRTFs can be loaded and transformed to the frequency domain before runtime. During runtime, incoming audio is also transformed to the frequency domain, which changes the convolution to a pointwise multiplication of the audio and IRs. The resulting signal is transformed back again using the inverse FFT for playback.

Delay

The delay is found by dividing the path length by the speed of sound: 330m/s. Because a delay in seconds is not very practical, it is converted to the number of samples by dividing it by the samplerate. The sound source can use this number to write the output for a specific propagation path to the mixer with an offset of a number of samples which delays the sound.

Windowing

Time-domain filters like the ones used for the absorption rely on their history to do the filtering. That has the unfortunate side effect that they do not filter a new signal straight away, they need to process some samples to 'warm up'. Because the filter is processing buffers, it has to warm up for each buffer again, which results in audible artifacts. To overcome these, a small windowing function is necessary. Each buffer is slightly extended to produce some overlap between buffers. Cross fading this overlap gets rid of most of the artifacts. The cross fading also helps to fill the gaps when delays change. When the geometry processing has completed and new delay times are available, this results in a discontinuity between the previous buffer and the next. Having some overlap between buffers reduces this effect. The size of the window depends on how big the delay changes get, which depends on how often the geometry processing runs and how fast things change in the environment.

6.2. Generating The Late Reverberation

Most games these days use an artificial reverberator for all their reverberation. Here, it is only used to generate a reverberation tail. Most game engines come with very usable reverberators already. These reverberators usually have an adjustable reverberation time and some filtering options, so they can be integrated into this system without much effort. The volume and absorption parameters found in the geometry processing stage can be used to calculate the reverberation time by using Sabine's equation.

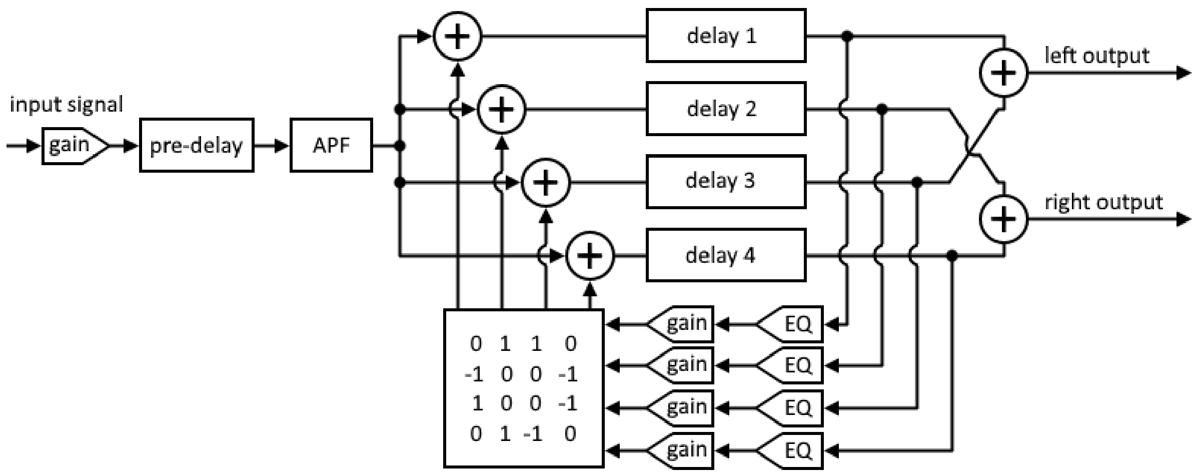


Figure 6.2: Feedback delay network topology

How exactly the filtering on the reverberation is applied depends on the available parameters of the reverberator. For the purpose of this project, a basic feedback delay network (FDN) was designed that can produce a reverberation tail that responds very nicely to changes in the T_{60} and the absorption. The schematic of this network is shown in Figure 6.2. Changing the gain in the network changes the reverberation time. The EQ is used in the feedback loop to iteratively color the sound based on the absorption, just like would happen in real life.

6.2.1. Blending Early And Late Reflections

The late reverberation has to be played back at the right time and volume so that the transition from early to late reflections is natural. To ensure a continuous signal without overlapping the early and late reverb, the input to the reverberator is delayed by the time it takes for the longest propagation path to arrive at the listener. This is controlled per sound source, so they all transition smoothly. The reverberator input level for each sound source is controlled by the shadow map, the distance and the losses from the sound source point of view.

6.3. Dynamics

As a final step in the signal chain, some dynamics processing is applied to keep the output from clipping. To achieve this, a basic dynamics compressor can be used that reduces sample values that exceed a threshold value by some ratio. A fairly mild dynamics compressor is advised on the FDN output, with a threshold of 0.7 and a ratio of 6. On the master output, a much more aggressive brick-wall limiter setting is advised of threshold 0.95, ratio 200. This soft-clips peaks in the signal which prevents the DA converter from clipping. This is necessary because stacking propagation paths together can sometimes lead to unexpected peaks in the signal when their delays are equal.

7 Results

7.1. Quantitative Results

Commenting on the sound quality of any system is subjective in many ways. The subjective experience is discussed briefly in Chapter 8. However, some metrics can be devised that describe parts of the subjective experience and allow for a quantitative comparison. This section will analyze the accuracy and the measurable perceived realism of this system.

7.1.1. Early Reflections

The best way of determining the accuracy of the early reflections is by comparing them to a method that will always find all of them. The direction from which these reflections arrive at the listener position and the time they take to get there are the most important characteristics here.

Specular Reflections

For specular reflections in a rectangular room, this is very easily done using the image method [8]. The MatLab application 'RoomSim' [21] was used to generate an impulse response for a 'shoebox' room of 4 by 5 meters, and 3 meters height. The sound source was placed at a height of 1m, 0.6m from the back wall and 0.7m from the left wall. The listener was positioned at a height of 2.6m, 0.55m from the front wall and 1.4m from the right wall. The setup is depicted in Figure 7.1. This figure also shows the direct signal and all first and second order reflections, except those that reflect off the floor or ceiling. The right side of Figure 7.1 shows the resulting impulse response, this time including all the reflections up to 20.5ms. To find these results, the air temperature is set to 0 °C, humidity to 50%, distance attenuation is set to on and the HPF, air absorption and smoothing are set to off. All walls are assigned the painted concrete material, which has a relatively low absorption. Because RoomSim uses a different HRTF, a single omnidirectional receiver is used and the HRTF is disabled in both systems.

When comparing the first order reflections from RoomSim to the reflections found in this system, they align nearly perfectly. The timing differences between the two systems are <0.1ms for the direct signal and the first order reflections. When measuring the angles of the incoming reflections, their accuracy is higher than the HRTF could reproduce. The signal level of each reflection also drops off with distance as expected. Irrespective of the sound source and listener position, the system appears to find all six reflections and the direct path in this simple setup. To achieve this, a cubemap resolution of 128x128 pixels is easily sufficient. Going down to 32x32 also works, provided the shadow mapping bias and edge detection threshold in the source fragment shader are adjusted accordingly. At a low resolution the timing of the reflections will be slightly off, since the reflection point will be less accurate. Of all the reflections in the IR plot in Figure 7.1, only the blue and red ones are modeled in this system. Higher order reflections are only modeled after the last first order reflection has arrived, which means that the six second order reflections that arrive before the last red reflection are missed.

In a more complex environment it is more difficult to find exact results to compare to. However, the system seems to find the expected first order reflections off of small and big surfaces alike. For

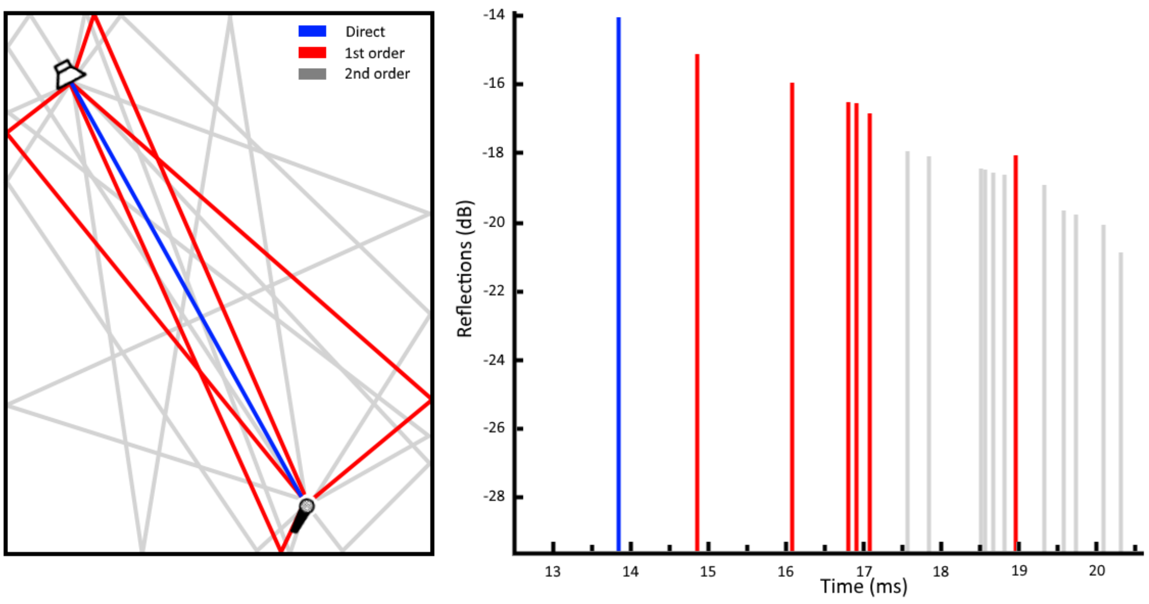
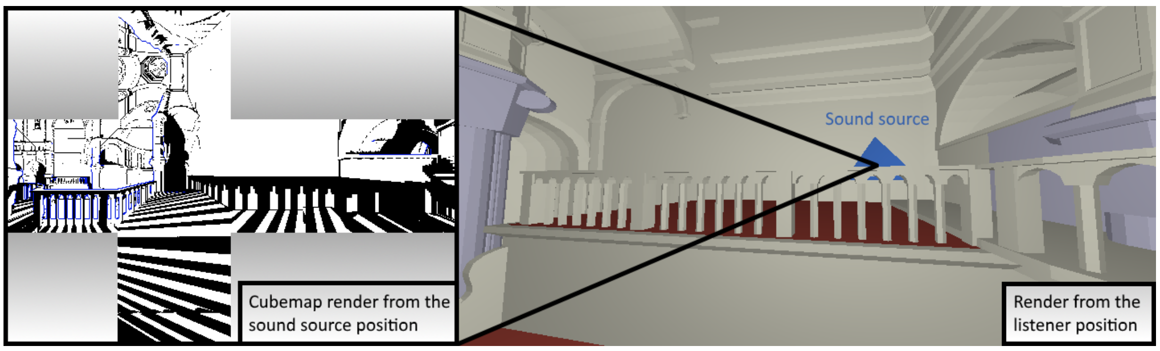


Figure 7.1: Impulse response of the early reflections in a rectangular room



Sound source
Cubemap render from the sound source position
Render from the listener position

Figure 7.2: Result of a shadow map with edge detection, showing the diffraction edges in blue.

instance placing a source and the listener in the focal point of the dome in the Sibenik cathedral [22] results in a very high number of specular reflections from all sides.

Diffraction

There is no diffraction when inside a shoebox since there are no edges there, and the image method cannot model diffraction anyway. Another technique is thus necessary to determine the accuracy of the diffraction model. There is however no easily available technique of finding all the diffracted propagation paths to compare to. However, only a single diffracted propagation path is being auralized in the end (See section 7.2.2), so making sure every single one is found is less important. Figure 5.6 is repeated in Figure 7.2. It shows the cubemap render from the sound source position, with a screenshot from the listener position looking in the direction of the sound source (the blue pyramid). All the blue pixels in the cubemap are diffraction paths. As expected, the balcony enables a lot of different diffracted propagation paths, easily allowing for a continuous signal for a moving listener, even if the direct signal is blocked regularly by one of the balcony bars. This cubemap render has faces of 128x128, however even with faces of 32x32 there are still plenty of diffraction paths found.

In short, this technique will find enough of the most important diffracted propagation paths to provide a continuous signal to the listener, even in occluded areas. When a less costly auralization technique is available, the accuracy and detail of the diffracted propagation paths will only increase since most of them are currently discarded.

7.1.2. Late Reverberation

The volume and absorption estimations used for modeling the late reverberation are very crude. As mentioned, both are easily influenced by small objects nearby. However, within the constraints of this limitation both estimates actually work really well. When moving through an empty room, both estimates are very stable. The absorption estimate changes slightly when moving towards a more absorbing material, but the difference between the far end of the room and standing right next to this wall is only about 10% which is perfectly acceptable. Similarly for the volume estimate. When moving through various rooms of different shapes and sizes, the estimate would vary being between 0 and 15% off, being consistently just under the actual value. This is perfectly acceptable since this variation is not noticeable in the final reverberation time. Should some objects skew the estimates too much, one can opt to simply not render them during the geometry processing stage. The takeaway message is that the estimates used here provide a very solid basis for controlling a reverberation generator. They will consistently deliver high reverberation times for larger rooms and the absorption estimate will also vary nicely with changes in the geometry, resulting in a very dynamic experience.

7.2. Computational Cost

In this section, the computational performance of the system is considered. The goal was to design a system that could easily model the acoustics in real time, leaving computational power for other things like rendering the game to the screen and running the game logic. The performance of the geometry processing and the auralization will be discussed separately since they are asynchronous.

7.2.1. Geometry Processing

The Paradigm

Chapter 5 started with noting how this system is quite different from classic ray tracing. Instead of creating actual rays and tracing them through the environment, the system renders the environment, treating each pixel in the render as a ray. Classic ray tracing requires every triangle to be checked against every ray, which is what makes it expensive. By using this rendering solution instead, all the triangles are drawn only once, providing a solution for all the pixels (rays) simultaneously. This only works because all the rays have the same starting position: the sound source. First order propagation paths are possible because all relevant rays have the same destination: the listener. Higher order reflections would not be nearly as efficient, since then a render call would be necessary for each individual ray, which brings performance back on par with classic ray tracing. It is the huge performance boost in the first order reflections that makes this system desirable.

Performance Measurements

The measurements presented in Table 7.1 give an indication of the computational cost of all the main parts of the system that concern the render thread. They are averaged timings from the java OpenGL code running on an Intel i5 4670K at 3.4GHz, 8GB ram and a Zotac Amp! GeForce 980Ti. Cube-map resolutions of 6x128x128 were used. These performance tests were conducted in two different environments: A simple shoebox consisting of 36 vertices, and the more complex Sibenik Cathedral, consisting of 225849 vertices.

The components in the table are structured as follows:

- Listener render: includes the cubemap render from the listener point of view and performing the sound source occlusion query.
- Listener data transfer: the time taken to create mipmaps and read the results to the CPU.
- Source render 1: the first render pass.
- Source render 2: the second render pass.
- Source data transfer: transferring the 3 source output textures to the CPU.
- Source texture scanning: scanning the output textures on the CPU for propagation paths.

	Shoebox	Sibenik Cathedral
Listener render	32μs	1240μs
Listener data transfer	566μs	580μs
Source render 1	18μs	1239μs
Source render 2	51μs	1287 μs
Source data transfer	3138μs	3029μs
Source texture scanning	356μs	430μs

Table 7.1: Computation time per component in two environments of different complexity

	Shoebox 1	Shoebox 5	Sibenik 1	Sibenik 5
Listener overall	796μs	860μs	2296μs	1665μs
Source overall	3462μs	3271μs	5379μs	4591μs

Table 7.2: Computational cost scaling with the number of sound sources, showing cost per source

The results are mostly as expected. The more detailed environment heavily hits the render stages, but not the data transfers since the size of the output data remains the same. Despite source render 2 having a seemingly complex shader, it hardly influences the render time. The obvious bottleneck is the source data transfer. Loading three large textures to the CPU is evidently quite costly. However, the total rendering time for a complex environment is 3ms per source, which means there is still processing time left to render the environment with a lot of graphical effects to the screen.

Table 7.2 shows how the performance scales with the number of sound sources. The times shown are measured on the CPU from the start of the listener geometry processing to the very end. The sound source data is measured per source. Measuring CPU timings means the results are fairly noisy, because the graphics card performs its computations asynchronously. By forcing timing results at the end of a cycle the process is effectively synchronized which means the CPU will have to wait for results from the GPU sometimes. The individual measurements are quite noisy because of that. However, these results were averaged over a large number of runs, so the measurements in the table give a good indication of the expected performance. Interestingly, adding more sources in the complex scene improves the performance, while it decreases the performance in the simple environment. Other than that, the results are again as expected and align nicely with the results from Table 7.1

7.2.2. Audio Processing

The audio processing is not really affected by the complexity of the geometry per se. However, the number of found propagation paths does affect the computational cost of the audio processing a lot. For that reason, the times presented in Table 7.3 are per propagation path. Interestingly enough, complex geometry does not necessarily mean more propagation paths are found. Inside an empty shoebox, the system will always find 7 (One for each surface and the direct signal). In a more complex environment like the Sibenik Cathedral, the number of propagation paths will vary between 0 and 12 per sound source in most places. In some cases, when for instance attaching a sound source to the player and positioning the listener in the focal point of one of the domes, it will find as many as 74 reflections. In general though, for indoor scenes a maximum of 12 propagation paths can be expected. The components in Table 7.3 are measured for 128 sample buffers and are measured per reflection except for the reverberator, which is just measured per buffer.

Component:	Time:
Initial processing	4μs
HRTF FFT forward	45μs
HRTF processing	8μs
HRTF FFT inverse	90μs
Reverberator	20μs

Table 7.3: Computational cost of the audio processing stages, measured per propagation path

From the results it is clear that the computational complexity of the initial EQing, crossfading and de-normalization is negligible compared to the HRTF. Finding the required angles and distances that are used for selecting the IR and the distance attenuation are not even included in the table as they consistently measured less than a microsecond. However, despite using the fastest FFT library available for Java (JTransforms [23]), the Fourier transforms for the HRTF take up the bulk of the computation time. The forward transform takes about 45μs, the inverse transform takes double that because the signal is now stereo. The actual IR lookup and multiplication only takes around 8μs for both ears combined. The entire reverberator feedback loop, including EQing, redistributing signals, dynamics compression etc takes roughly 20us, regardless of the reverberation time.

That means the total audio processing adds up to 147μs per reflection + 20μs. For a setting of 2 sources with 10 reflections each that adds up to 3.1ms processing time for a 128 sample buffer at 44.1kHz samplerate. On a single core that would be the limit, and that core would just be processing audio. However, the sound sources run in separate threads so they can be distributed over several cores, allowing for more sources and reflections. Still, the number of propagation paths is clearly a bottleneck. This is also the main reason why auralizing all the discovered diffracted propagation paths is not doable. It shows that convolution is simply a relatively expensive process, even when taken to the frequency domain and when the HRTF IRs are only 4800 samples long. For reference, doing this in the time domain would result in a convolution that takes more than a millisecond, so despite being expensive, the FFTs are still definitely worth doing.

Chapter 6 already referred to the output buffer and the fact that it introduces latency. Unfortunately, the optimal setting greatly depends on the computer. Real time performance is only partly reliant on pure processing power. Dropouts can occur way before the CPU runs out of available clock cycles on a badly configured system. Device drivers or even power saving options in the OS can greatly affect the real time performance of a system. It is thus very difficult to give any meaningful hardware requirements for a real time audio system. However, on the test system a stable output is achieved with an output latency that is roughly the same as a single frame at 60Hz, which is 17ms. That means there is no visual clue that can make a player notice such latency.

8 Discussion

8.1. Qualitative Conclusions

Overall, the results are very promising. Reverberation can be modeled in a variety of environments with accurate sound source localization, while all the required information is taken directly from the geometry. The results from the geometry processing stage are very good. The early reflections are modeled very accurately and the late reverberation parameter estimation provides very usable results. The fact that the estimates are not perfect is not really a problem, since small variations only lead to small changes in the length of the reverberation tail, which is hardly noticeable even for a trained ear. If the resulting parameters are not as desired, this can quickly be achieved by adjusting the absorption parameters in the materials of the geometry.

The results from the auralization are also good. When comparing the full system to using just a fixed-parameter reverberator the difference is night and day. Sound source localization is much better and the reverberation responds very nicely to changes in the environment. Even if early reflections are skipped and just the reverberator is used with the parameters taken from the environment, the resulting reverberation is more engaging and dynamic. The biggest perceived difference is in the early reflections though, because the HRTF combined with the first order reflections results in excellent imaging. It has to be noted that the effectiveness of the HRTF varies from person to person, which means that the localization may work very well for some and not so well for others. When using this system in a commercial product, it is probably best to supply a number of different HRTFs so users can pick the one that works best for themselves.

In most cases, the auralization sounds smooth. The changes in path lengths and HRTF IRs produce some artifacts, but they are usually masked by the sounds themselves and the late reverberation. However, musical sounds with long constant notes with very few harmonics have very little content that can mask these artifacts, so in that case they can be audible. Such a setting would require a more intelligent way of transitioning between buffers and IRs if the artifacts are unacceptable.

8.2. Performance bottlenecks

A good sounding system is important, but there are plenty of good sounding systems available. The challenge was to design one that does not require as much computation as the others. This section discusses the computational performance of the system.

8.2.1. Geometry Processing Optimizations

As was mentioned in Chapter 7, the computational load on the graphics card is very reasonable. The complexity of the shaders is low, which means the performance really only depends on the cubemap resolution and the complexity of the geometry. The GPU render time of 3ms per source in a complex environment and at a high cubemap resolution is acceptable, depending on the amount of sources and the update frequency. Exactly how many updates are necessary for a smooth experience depends on the setting and requires some experimenting, but in general it seems like 30Hz would easily suit most applications.

The biggest bottleneck in the geometry processing stage is the texture transfer from the graphics memory to the CPU and scanning them. Nearly half of the computation time is spent on these two steps. Since better solutions are available as mentioned in Chapter 5, it would be interesting to see how much further the system can be optimized. A GPU texture scanning solution would require a little bit of extra render time, but it saves a lot of memory transfers and frees up some extra CPU time because the resulting output texture is much faster to process.

8.2.2. Audio Processing Optimizations

The biggest bottleneck of when using this system comes from the HRTF processing. A shorter HRTF or no HRTF would allow for many more sound sources or propagation paths. Using a bigger buffer size than 128 would also help, as that reduces the amount of forward and inverse Fourier transforms. As long as the buffer size stays below 4800 samples (the HRTF IR length) that would not introduce any additional computation at all, while significantly reducing the amount of times the HRTF has to process audio. A bigger buffer does increase the output latency though, so one has to find a balance.

An alternative to using a HRTF, would be to use a surround playback system instead. That would require the system to provide samples to a lot more output channels, but panning sound sources in surround is much cheaper than putting them through a HRTF. An additional advantage is that surround systems will work for everyone equally well, whereas a general purpose HRTF may perform underwhelmingly for some people. However, the spatial resolution of a HRTF is higher than that of a surround system. More importantly, many people do not have a surround setup, whereas a HRTF is designed for stereo headphone use.

Furthermore, not all types of sound sources benefit from spatialisation. Resources can be saved by carefully selecting which sources to process. For instance, body sounds like the beating of the player's heart or breathing do not need to be spatialized. The same applies to static sounds like menu interactions or background music. Finally, very low frequencies are very difficult to spatialize using a HRTF, so machines that emit a low drone for instance could benefit more from a good stereo sample and if necessary just the late reflections modeling. Making smart choices of which sources to spatialize can thus expand the number of simultaneous sources.

8.3. Limitations

The system is designed to work equally well in any environment. There are however some restrictions that apply to make it work properly.

8.3.1. Geometry Requirements

As mentioned twice in Chapter 5, there is a need for double walled geometry to make both the specular reflections and diffraction work. Another way to put it is that any surface in the environment can only ever be seen from one side. If that is not the case, specular reflections and diffraction can occur through a wall in corners. Double walled geometry that leaves enough room between walls to compensate for this bias will completely solve this problem.

Another small but important limitation on the geometry is that the absorption coefficient [0.0, 0.085, 0.38] cannot be used in any of the materials. The system will incorrectly treat any specular reflection of such a surface as diffraction, which means it will likely be discarded since only a single diffracted propagation path is auralized in the end.

8.3.2. Occlusion Limitations

By using only first order specular and diffuse reflections and first order diffraction, there are some limitations on when sound sources are audible. In this system, an audio source is inaudible if there is no geometry that is visible from both the listener and source position. This is no problem in a simple room,

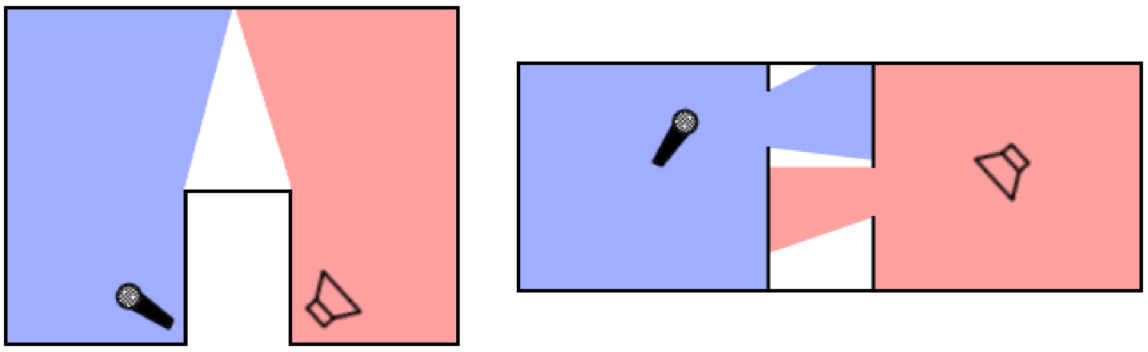


Figure 8.1: Occlusion limitations for complex or adjacent rooms

but it could be a problem for multiple connected rooms or complex room shapes. Figure 8.1 shows examples of these two cases. In both cases, a second order diffuse reflection or second order diffraction would solve this problem. However, second order diffuse reflections would be very undesirable to implement in the current system since it would require a new render for every visible pixel in the sound source cubemap. Second order diffraction could be an option if a technique like UTD [2] was added. A cheaper solution would be to enforce a minimum input gain to the reverberator for each sound source inside a predefined area. That will ensure occluded sound sources are always audible in the late reverberation if the listener and source are in the same area. This does however require some additional input from a level designer in the form of a predefined spatial subdivision. Predefining anything in a dynamic environment does have its drawbacks though, since this spatial subdivision does not adapt to changes in the environment.

8.3.3. LR Limitations

The late reverberation currently starts right after the last first order reflection. However, there are often second order reflections that arrive before that time. Since they are not modeled now, the last part of the early reflections may be less dense than it should be. This could be compensated for by shifting the late reverberation a bit forward in time. However, the amount of shifting that is required to achieve a result that is similar to the actual IR is very difficult to find without modeling these paths explicitly.

Another limitation stems from the fact that only a single reverberator is used to model all the late reverberation. This provides a good result for sound sources that are in the same room or an acoustically similar room as the listener. However, if the sound source is in a highly reverberant room and the listener is not, one would expect to hear a long reverb tail coming from the direction of the sound source, which will not happen. This is only really noticeable in extreme situations, where the listener is for instance standing outside a big cathedral and a sound source inside is audible through the open door. This sound source will sound dry, as the reverberator provides very little reverberation for an outside environment, while in reality the sound from the door opening would have the long reverb tail that comes from the cathedral. A possible solution would be to use a reverberator per sound source, whose input parameters are derived from the sound source position. The output of this reverberator is then fed into the listener reverberator, whose parameters are derived from the listener position. That allows a sound source in a reverberant room to always have a reverberation tail, regardless of the acoustics at the listener position. Since a reverberator is fairly cheap to run, this solution could work very well provided the number of sound sources remains manageable. An additional advantage is that the reverberator output of the sound sources can now be given a direction using the HRTF or regular panning. That would allow the listener to locate the source of that long reverberation tail even if he is not inside the reverberant room himself.

9 Future Work

9.1. Improvements

The system can be improved upon in a number of ways. More time could be spent looking at replacing the cubemap implementation with a projection that requires less than six render calls, like a spherical or parabolic projection. Challenges here are the increased distortion, and the fact that both of these projections result in two circular images, where the four corners of each image overlap with the other image. That means that using mipmaps to get average values is not valid.

The current volume estimation technique is quite limited and sensitive to nearby objects as they block much of the visible volume of the room. An alternative way of estimating the volume could be considered that has more information to work with than just line of sight from the listener. For instance a voxelization of the environment on the graphics card combined with an algorithm that determines which voxels should contribute could produce more reliable results.

Additionally, the current system only uses a single diffracted propagation path even though many more are found. A smart way to filter these propagation paths could decide which of these are a valuable addition to the overall sound. The system could be made even more efficient with clever sound source management. If there is a way to determine whether or not a sound source is audible at all, processing could be reduced by disabling inaudible sources for as long as they are inaudible. More research can also be done into the perceived effects of lower propagation path update frequencies.

If graphics performance is a concern, the system could be accelerated by using simplified geometry for the acoustics model. Siltanen et al [24] have introduced an automatic method for simplifying geometry while keeping the acoustical properties mostly intact. Finally, finding the propagation paths happens by scanning the output texture on the CPU. This is an expensive task that scales poorly with resolution. A GPU solution for getting the propagation paths available to the CPU efficiently would alleviate this task from the CPU.

9.2. New Features

In terms of features, many directions are possible. Transmission has not been considered at all, but it would be a welcome addition to the model. The line of sight approach would be challenging though, since double-walled geometry is required to make the other propagation paths work. Second order specular reflections or diffraction might also be possible with just a single additional render call if a binary voxel grid is used to test occlusion in the propagation path between the first and second reflection point. Even higher order diffraction could be achieved with UTD, as mentioned in Chapter 8. The idea of using multiple reverberators was also mentioned there, which would allow more flexibility.

Even with a single reverberator though, there is a lot of spatial information in the sound source renders that could be used to enhance the volume and absorption estimates for the late reverberation, which is currently not exploited. Characteristics of the environment that are not visible from the listener position could be taken from the sound source positions by somehow combining these cubemaps, weighting the contribution of each sound source by a value that indicates if they are in the same room or not. Said value could for instance be based on the shadow map.

Finally, some parametric reverberators offer additional parameters. For instance, the initial echo density could be tweaked depending on the volume estimation and source proximity, which allows for a more accurate transition between early and late reflections.

A

Specular Reflection Pseudo-Code

```
//shadow mapping
actualDistance = length(listenerPosition - fragmentPosition);
depth = listenerCubemap(direction(fragmentPosition - listenerPosition));
measuredDistance = length(depth);
if(actualDistance < (measuredDistance + bias)){

  //check for edges
  if(abs(pixel[1].distance - pixel[2].distance) > 1.2){
      abort;
  } ... // repeat for pixels 1 and 3, 2 and 4, 3 and 4.

  //check for big angles
  if(abs(arccos(dot(fragmentNormal[1], fragmentNormal[2]))) > Pi/8){
      abort;
  }
  if((abs(arccos(dot(fragmentNormal[1], fragmentNormal[3]))) > Pi/8){
      abort;
  } ... // repeat for pixels 2 and 4, 3 and 4.

  //create reflection vectors
  for(pixel i=1:4){
      viewVector[i] = fragmentPosition[i] - soundsourcePosition;
      reflectionVector[i] = viewVector[i] -
              2*fragmentNormal[i]*dot(viewVector[i], fragmentNormal[i]);
  }

  //construct beam plane 1 (bottom plane)
  if(dot(reflectionVector[1], reflectionVector[2])>0){
      planeNormal[1] = cross(reflectionVector[1], reflectionVector[2]);
  }else{
      planeNormal[1] = cross(reflectionVector[2], reflectionVector[1]);
  }

  //check listener position with respect to plane 1
  if(dot((listenerPosition - fragmentPosition[1]),
          planeNormal[1]) > 0){
```

```
     //construct beam plane 2 (left plane)
     if(dot(reflectionVector[1], reflectionVector[3])<0){
         planeNormal[2] =
                 cross(reflectionVector[1], reflectionVector[3]);
     }else{
         planeNormal[2] =
                 cross(reflectionVector[3], reflectionVector[1]);
     }

     //check listener position with respect to plane 2
     if(dot((listenerPosition - fragmentPosition[1]),
             planeNormal[2]) > 0){

         //construct beam plane 3
         if(dot(reflectionVector[2], reflectionVector[4])>0){
             planeNormal[3] =
                     cross(reflectionVector[2], reflectionVector[4]);
         }else{
             planeNormal[3] =
                     cross(reflectionVector[4], reflectionVector[2]);
         }

         //check listener position with respect to plane 3
         if(dot((listenerPosition - fragmentPosition[2]),
                 planeNormal[3]) > 0){

             //construct beam plane 4
             if(dot(reflectionVector[3], reflectionVector[4])<0){
                 planeNormal[4] =
                     cross(reflectionVector[3], reflectionVector[4]);
             }else{
                 planeNormal[4] =
                     cross(reflectionVector[4], reflectionVector[3]);
             }

             //check listener position with respect to plane 4
             if(dot((listenerPosition - fragmentPosition[3]),
                     planeNormal[4]) > 0){
                     //Specular reflection found!
             }
         }
      }
   }
}
```

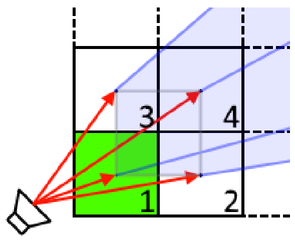


Figure A.1: Pixel numbering used

Bibliography

[1] H. Haas, *The influence of a single echo on the audibility of speech,* J. Audio Eng. Soc **20**, 146 (1972).

[2] N. Tsingos, T. Funkhouser, A. Ngan, and I. Carlbom, *Modeling acoustics in virtual environments using the uniform theory of diffraction,* in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (ACM, 2001) pp. 545–552.

[3] E. Persson, *Fatbursparken cubemap,* (2008).

[4] Gamasutra and L. P. Dion, *Game design deep dive: Dynamic audio in destructible levels in rainbow six: Siege,* (2017).

[5] P. Astheimer, *What you see is what you hear-acoustics applied in virtual worlds,* in *Proceedings of 1993 IEEE Research Properties in Virtual Reality Symposium* (1993) pp. 100–107.

[6] N. Röber, M. Spindler, and M. Masuch, *Waveguide-based room acoustics through graphics hardware.* in *ICMC* (2006).

[7] E. De Sena, H. Hacıhabiboğlu, Z. Cvetković, and J. O. Smith, *Efficient synthesis of room acoustics via scattering delay networks,* IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP) **23**, 1478 (2015).

[8] J. B. Allen and D. A. Berkley, *Image method for efficiently simulating small-room acoustics,* The Journal of the Acoustical Society of America **65**, 943 (1979).

[9] M. Queiroz, F. Iazzetta, F. Kon, M. A. H. A. Gomes, F. A. L. Figueiredo, B. Masiero, L. K. Ueda, L. Dias, M. A. H. C. Torres, and L. F. Thomaz, *AcMus: an open, integrated platform for room acoustics research,* Journal of the Brazilian Computer Society **14**, 87 (2008).

[10] T. Funkhouser, N. Tsingos, I. Carlbom, G. Elko, M. Sondhi, J. E. West, G. Pingali, P. Min, and A. Ngan, *A beam tracing method for interactive architectural acoustics,* The Journal of the acoustical society of America **115**, 739 (2004).

[11] M. T. Taylor, A. Chandak, L. Antani, and D. Manocha, *Resound: Interactive sound rendering for dynamic virtual environments,* in *Proceedings of the 17th ACM international conference on Multimedia* (ACM, 2009) pp. 271–280.

[12] M. Taylor, A. Chandak, Q. Mo, C. Lauterbach, C. Schissler, and D. Manocha, *Guided multiview ray tracing for fast auralization,* IEEE Transactions on Visualization and Computer Graphics **18**, 1797 (2012).

[13] S. M. Schimmel, M. F. Muller, and N. Dillier, *A fast and accurate "shoebox" room acoustics simulator,* in *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on* (IEEE, 2009) pp. 241–244.

[14] B. Kapralos, M. Jenkin, and E. Milios, *Sonel mapping: A probabilistic acoustical modeling method,* Building Acoustics **15**, 289 (2008).

[15] C. Schissler and D. Manocha, *Interactive sound propagation and rendering for large multi-source scenes,* ACM Trans. Graph. **36**, 2:1 (2016).

[16] L. Williams, *Casting curved shadows on curved surfaces,* SIGGRAPH Comput. Graph. **12**, 270 (1978).

[17] J. Ahrens, *Analytic Methods of Sound Field Synthesis* (Springer Berlin Heidelberg, 2012) pp. 263–264.

[18] J. Flick, *Cube sphere, a unity c# tutorial,* (2015).

[19] R. Bristow-Johnson, *Cookbook formulae for audio eq biquad filter coefficients,* (2006).

[20] H. Kayser, S. D. Ewert, J. Anemüller, T. Rohdenburg, V. Hohmann, and B. Kollmeier, *Database of multichannel in-ear and behind-the-ear head-related and binaural room impulse responses,* EURASIP Journal on Advances in Signal Processing **2009**, 10 (2009).

[21] D. Campbell, K. Palomaki, and G. Brown, *A matlab simulation of "shoebox" room acoustics for use in research and teaching,* Computing and Information Systems **9**, 48 (2005).

[22] M. Dabrovic, *Sibenik cathedral 3d model,* (2001).

[23] P. Wendykier, *Jtransforms,* (2017).

[24] S. Siltanen, T. Lokki, L. Savioja, and C. Lynge Christensen, *Geometry reduction in room acoustics modeling,* Acta Acustica united with Acustica **94**, 410 (2008).