INTERACTIVE SOUND PROPAGATION FOR MASSIVE MULTI-USER AND DYNAMIC VIRTUAL ENVIRONMENTS

Micah Taylor

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2014

Approved by:

Dinesh Manocha

Gary Bishop

Ming Lin

Russ Taylor

Nicolas Tsingos

ABSTRACT

Micah Taylor: Interactive Sound Propagation for
Massive Multi-user and Dynamic Virtual Environments
(Under the direction of Dinesh Manocha)

Hearing is an important sense and it is known that rendering sound effects can enhance the level of immersion in virtual environments. Modeling sound waves is a complex problem, requiring vast computing resources to solve accurately. Prior methods are restricted to static scenes or limited acoustic effects. In this thesis, we present methods to improve the quality and performance of interactive geometric sound propagation in dynamic scenes and precomputation algorithms for acoustic propagation in enormous multi-user virtual environments.

We present a method for finding edge diffraction propagation paths on arbitrary 3D scenes for dynamic sources and receivers. Using this algorithm, we present a unified framework for interactive simulation of specular reflections, diffuse reflections, diffraction scattering, and reverberation effects. We also define a guidance algorithm for ray tracing that responds to dynamic environments and reorders queries to minimize simulation time. Our approach works well on modern GPUs and can achieve more than an order of magnitude performance improvement over prior methods.

Modern multi-user virtual environments support many types of client devices, and current phones and mobile devices may lack the resources to run acoustic simulations. To provide such devices the benefits of sound simulation, we have developed a precomputation algorithm that efficiently computes and stores acoustic data on a server in the cloud. Using novel algorithms, the server can render enhanced spatial audio in scenes spanning several square kilometers for hundreds of clients in realtime. Our method provides the benefits of immersive audio to collaborative telephony, video games, and multi-user virtual environments.

Dedicated to Christine, Charlotte, and Thomas.

ACKNOWLEDGEMENTS

I want to thank the members of my committee for their feedback and advice: Gary Bishop, Russ Taylor, and Nicolas Tsingos. I want to thank Dinesh Manocha, Ming Lin, and the members of the GAMMA group for all their support, criticism, and insight over the years. I have enjoyed working with all of you.

I especially want to thank Zhimin Ren, Qi Mo, and Lakulish Antani for their support. I also thank my colleagues at Rose-Hulman, who supported and trusted me during the last years of this work. Each of you helped me grow as a scientist and a friend.

There were many people who helped me see the joy in learning, long before I understood their generosity: James Frazier, Ken Holmes, Mark Ardis, and J. P. Mellor. My family back in Indiana also gave me tremendous support before and during this work; I cannot thank them enough for their love and encouragement.

I am very grateful for my friend Anish Chandak, without whom I would never have completed this work. I could always go to him for valuable advice and careful consideration, no matter the difficulty of the problems facing me.

Finally, I want to thank my lovely wife, Christine, for holding my hand during this adventure. Her endless love and wisdom led me to where I am today.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Our ability to hear sound waves is one of our most important senses. Along with sight, it is the only sense that can detect remote objects with high-fidelity [Blauert, 1997]. Unlike sight, our hearing is not limited to a small field of view in front of us; hearing allows us to sense the world in any direction. Without turning to direct our focus, our sense of hearing can distinguish sounds that come from in front from those that come form behind. Our ability to emit sound complements our hearing, allowing communication through sound. Interaction with sound waves is shared by all humans and other animals. As a society, we have invested a great deal of effort in harnessing sound for communication and entertainment.

Like all waves, sound has complicated interactions with the environment. Sound waves begin at some vibrating source and propagate out through a medium. The waves may reflect and scatter as they encounter objects in the medium. These interactions alter the wave in ways that are understood. Our perception of sound is adapted to recognizing these changes in the wave, and we can derive information about the source and environment from the modified wave patterns.

1.1 Physical properties of sound

Sound begins as a mechanical vibration (itself a wave) of an object. The vibrations may be transmitted into the object's containing medium and longitudinal waves propagate forth [Kinsler et al., 1999]. These waves are compressions and rarefactions of the medium.

The waves may vary in amplitude and frequency; our ears can perceive and distinguish both. The source of the mechanical vibration determines the wave properties. A high frequency vibration results in a high frequency wave. As the vibration is transmitted to the medium, the medium directly influences the speed of the wave, and consequently, its wavelength.

As the wave propagates through the medium, it may encounter objects or variations in the medium, for example density. These may reflect or scatter the wave in various ways. Sound

reflections result in the echoes heard in canyons or large empty rooms. The wave may also transfer some of its energy when it encounters objects, causing the object to vibrate. The object's vibration may cause another sound wave to propagate. This is how sound travels through apartment walls. Some of these wave effects vary based on the wavelength of the wave. Since wavelength and frequency have an inverse relationship with wave speed, throughout the text we will often use wavelength and frequency interchangeably.

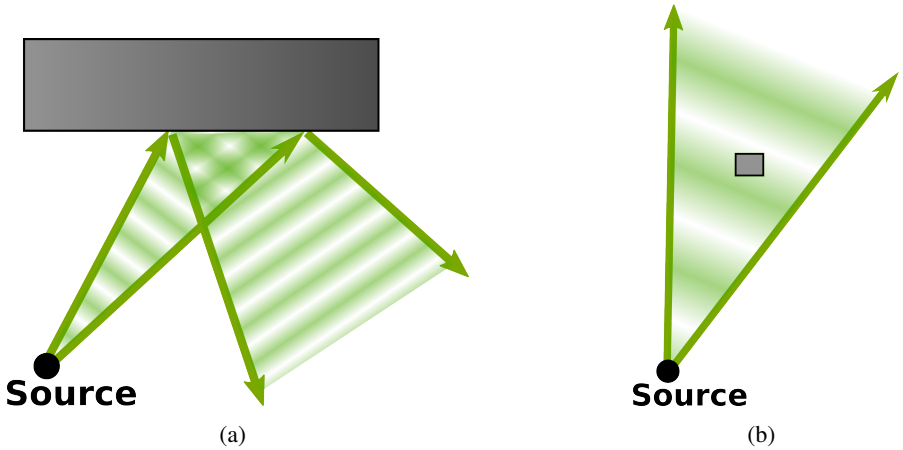


Figure 1.1: **Wavelength and object size:** Objects and details much larger than the wavelength (a) cause the wave to reflect in a specular manner, while objects much smaller than the wavelength (b) have little influence on the wave.

If the object's surface is smooth relative to the wavelength, the wave can be reflected specularly (Figure 1.1a). This kind of reflection is similar to how light reflects in a mirror: the angle of incidence to the surface is the same as the exit angle for any portion of the wavefront. Similarly, if the entire object is much smaller than the wavelength, the wave continues propagation with change (Figure 1.1b).

However, for objects that have details that are similar in scale to the wavelength size, diffraction occurs (Figure 1.2). For example, when a wave encounters an edge, some of the energy in the wave bends around the edge. Since the bending is related to wavelength, some frequencies may bend more than others. This can lead to audible sounds even when the sound source is out of sight of receiver. This is an important effect for humans, since it allows us to detect sound sources even when they are visually occluded.

If the object's surface has surface complexity in similar scale to the wavelength of the sound wave, diffraction occurs due to the surface complexity and the resulting wave is scatters in a complex

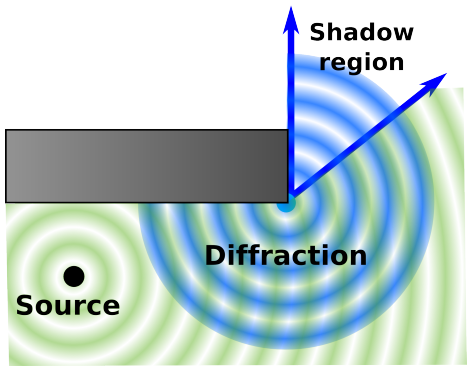
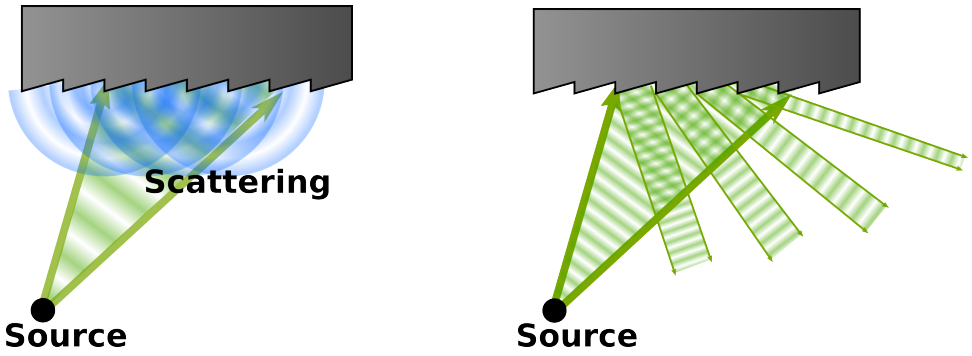


Figure 1.2: **Diffraction:** Edge diffraction occurs when a wave encounters an object edge and some of the wave propagates in the *shadow region*.



(a) (b)

Figure 1.3: **Wave scattering:** Diffuse reflections (a) are a result of many diffraction interactions when surface details match wavelength; if wavelength is much smaller than the details (b) specular reflection occurs.

pattern (Figure 1.3a). This interaction reflects the wave in a diffuse manner; the exit angle has a complex relationship to the incident angle. However, if the surface complexity is much larger than the wavelength of the sound wave, the wave reflects specularly off the individual surface facets (Figure 1.3b).

As the wave moves through the environment, some energy is lost due to propagation through the medium. Additionally, each encounter with an object or medium change alters the waveform and may absorb energy. Since the speed of sound is hundreds of meters per second and typical indoor spaces are only a few meters big, the wave may undergo hundreds of interactions in just a few seconds. As the wavefront scatters, different portions are scattered and reflected along different paths through the environment, causing the wave's energy to be spread out over time and frequency, and the wave amplitude to vary.

Eventually, some portion of the wave energy may arrive at a receiver. Human ears have organs that can sense the wave as changes in pressure. The changes in pressure in our ear canal are transmitted to our brain as neural signals, which are perceived as sounds. Since our ears are asymmetrical in the horizontal plane and the vertical plane, waves arriving from different directions diffract around our bodies in unique ways. Additionally, the wave must diffract around the head to reach both ears. This can delay the arrival of sound to one ear. All of these properties allow our brains to localize the sound source, that is, determine the approximate direction of the source.

1.2 Multi-user voice communication

Sound has been subject to much use and investigation over the centuries. Speech, the act of emitting purposeful sounds, is humanity's fundamental method of communication.

As science established the properties of sound waves, humans have sought to harness sound for our advancement. Telephones and recording tools are the most obvious recent advancements. The invention of the telephone was humanities first tool that allowed natural two-way communication over vast distances. Telephones leverage our already existing biological tools in a natural way, unlike any technology heretofore.

Our technology continues to advance and humanity's knowledge is now stored in more permanent and convenient written form. Even so, audio communication remains important, as written storage requires training and tools to use. Indeed, audio communication is so basic, that realtime voice communication between any two places in the world is almost taken for granted.

With the advent of the Internet, voice communication became extremely affordable and reliable. The flexibility of modern computers has allowed voice communication over vast distances to be trivially accessible and collaboration between large groups easily attained. Voice over Internet Protocol (VoIP) is the process of using the internet to enable voice communication. Voice data is recorded and compressed on one computer, then transmitted to a remote computer for reconstruction. The remote system attempts to recreate the input sound using rendering software and an output speaker system.

In modern VoIP systems, many people may be communicating in a single conversation. Since each person wishes to hear all other speakers, all voice data must be streamed to each client computer.

A *peer-to-peer* arrangement is when each clients sends all recorded voice data to each other client, but this may require significant network resources. An alternative is a *client-server* arrangement, where the voice data is sent to an intermediary, which aggregates the data and sends the result to each client. This moves the network and compute burden to the intermediary server.

1.3 Information in sound

Given the physical properties of sound waves and the way humans use sound to communicate, it is clear that waves can carry explicit information from their source as well as implicit information due to their propagation. From birth, our sense of hearing continually saturates us in sound information and our brains adapt to what we hear. We grow to be able to infer a great deal from the sounds that we hear and become naturally trained at coupling hearing with our other senses.

As discussed in Section 1.1, human ears allow localization of sound sources. There is also a discrepancy between the perceptions of sight and sound. We spend most of our lives in an air medium where light travels very quickly and sound relatively slowly. We learn to recognize that when an accompanying sound response is delayed from a visual response, that the action that propagated the light and sound wave is some distance away. Children use this to determine the distance of thunderstorms by counting the seconds between a lightning strike and the corresponding thunder sound. Relative to our perception, light travels practically instantaneously, while sound travels at approximately 343 meters per second. Thus, every three seconds between lightning strike and thunder indicates that the storm is one kilometer away.

Each reflection of the sound wave sends energy along a new path through the environment. Some sound energy may arrive at the receiver along a path with no reflections, and is only delayed by the distance between source and receiver based on the speed of sound. Other energy arrives by way of several reflections. This energy must travel a greater distance and more energy is lost to the medium and the reflection, and is thus delayed with a lower amplitude. This results in echoes and reverberation. Humans come to expect sounds in large enclosed spaces to produce such reflected effects. Additionally, we can infer the size of the enclosed space based on the density and length of the echoes and reverberation. Cathedrals and canyons will have lots of echo and reverberations, living rooms will not.

Since we hear sounds our entire lives, we come to expect certain actions to have certain sounds. If the sound source is well known, we can detect changes in frequency and amplitude from the expected sound. Someone speaking from around a corner will be slightly muffled; music heard through a wall will be very muffled; a voice from far away will be quieter. The effect of the environment on the wave conveys information about the environment and the location of the source to us.

1.4 Sound simulation

Given how pervasive sound is in the physical world, it is desirable to be able to simulate the properties of sound. It is convenient to divide sound simulation into three parts: input, propagation, and output. Each part can be combined with others to form a complete sound rendering pipeline.

There are many applications for sound simulation. Architectural acoustics is the study of sound in man-made structures. Buildings have been designed with acoustics in mind for centuries. Many modern auditoriums have been built without regard to acoustic properties and suffer from intelligibility issues for both speech and music. Acoustic simulations can provide valuable insight into acoustic problems with existing structures as well as guide design of future structures . The propagation of sound waves through the architectural environment is the most important process that is modeled for this application. Most commercial acoustic simulators, such as CATT[1], EASE[2], and ODEON[3], are designed for architectural acoustics.

Many entertainment mediums rely on high quality visuals and audio. In film, *Foley artists* have long designed the sound effects that accompany on screen actions. Various physical objects are used to create artificial sounds that are expected by viewers, for example banging coconuts for horse foot steps. Many film visual effects are simulated in virtual environments for cost and safety reasons. It is reasonable that sound simulation could assist Foley artists in the creation of sound. For this application, the creation of sounds by physical contact would be important to simulate.

Video games are another medium that could benefit from sound simulation. Just as with film, three types of simulation could be useful: sound synthesis from contacts, sound propagation through environments, and sound output from speakers. An additional requirement is that all these simulations

[1] http://www.catt.se

[2] http://ease.afmg.eu

[3] http://odeon.dk

must run interactively meaning that the simulation must match the visual simulation, often within a few milliseconds. Many games feature highly dynamic environments, where doors open and close, buildings collapse, and vehicles move around. The acoustic simulation must be able to respond to these dynamic events.

Virtual training simulations are interactive virtual environments similar to video games, but with clearly defined goals of enhancing the participant's ability in specific areas. Possible simulations could be street crossings for blind people, battle training for soldiers, and emergency situations for medical personal. These applications require interactive and realistic visual and audio simulations to avoid the possibility of negative training.

Multimodal visualization is the use of several senses to convey data relationships. Visual displays are the most common way to visualize data, but sound can be used also. Voice communication systems with many participants can place each participant in a virtual auditory space to allow the listener's natural localization to determine which participant is speaking. Simple auditory displays include how many operating systems alert users when long operations complete with an audible alert (a 'ding' or such). In more realistic virtual environments, non-physical auditory cues can aid the participant in understand the virtual environment. With realistic propagation simulations, a participant could be alerted to activity in other parts of the virtual environment. Simplified versions of this exist in some video games in the form of 'map pings' where an audible noise is made to draw attention to the environment. Auditory displays are not restricted to virtual environments.

Simulations can also be used to design better transmitters and receivers for very complex audio problems such as medical ultrasounds and underwater sonars. These require propagation simulations to account varying media in addition to the effects described above. Density changes result in changes in the speed of sound and make predicting wave propagation difficult.

1.4.1 Input

All sound simulations begin with some input sound signal. This can come from a recording of a real-world sound or synthesized using a simulation. Most common input sounds come from physical vibrations of materials in the environment captured with a recording device. Without careful planning, the specific radiation pattern of the physical event is not captured. Indeed, most recordings capture a single sound channel and this is insufficient to fully represent the input signal in a virtual environment.

A *beamforming* technique can use an array of recording devices to measure the radiation field more accurately.

An alternative to measuring physical sounds is to create the sounds synthetically. *Sound synthesis* is the process of creating sound signals algorithmically. Frequency modulations uses combinations of pure tones to form useful signals for music. Physical contact sounds can also be modeled by simulating material vibrations in contacting virtual objects.

1.4.2 Propagation

Once a sound wave is transferred to the environment's medium, it begins propagating through the environment. Simulate of wave propagation must balance accuracy and speed. Humans can perceive a wide range of frequencies: 20 hertz to 20 kilohertz. With a typical speed of 343 meters per second in air, sound waves have a wavelength in the range of 17 meters to 1.7 centimeters. Most objects built and used by humans (furniture, office doors, cups, etc.) have similar scale to sound waves. This means sound has wave-like interactions with these objects. An acoustic wave equation predicts the propagation of sound waves. The details of this are discussed in Section 2.2.

For visual simulations of light propagation, most wave effects can be ignored since the wavelength of light is on the order of hundreds of nanometers and wave effects on that scale are hard for humans to observe visually. Moreover, light travels much faster than sound, so only the steady state needs to be computed. Most light simulations model the light wave as a wave front of particles and ignore the time component.

Sound simulation is much more challenging as compared to visual simulation. Wave effects, like diffraction and interference are prominent and easily audible to humans. Sound also travels much slower and humans can easily detect delays, echoes, and reverberation effects caused by interactions from the environment.

Accurate simulations should be able to compute a wide range of frequency inputs and outputs, handle wave effects, and output correct time domain values. Solving all of these effects in a single simulation is difficult. The acoustic wave equation predicts all these effects, but is very time consuming to solve, as it scales with the fourth power of the maximum simulation frequency. Dynamic scenes further complicate simulation, since objects may shift during the simulation step

and frequency shifts from the Doppler effect are audible. Another consideration is environments where the medium density varies and the medium is in motion, such as atmosphere or ocean currents.

1.4.3 Output

Synthesis and propagation simulations are of no use without a means to render audible sounds for the listener. Since humans can hear frequencies up to 20 kilohertz, accurately reconstructing sound signals requires sampling rate of about 40 kilohertz, due to the Nyquist limit. Humans can perceive a wide range of amplitudes, from approximately one ten-thousandth of a pascal of pressure to tens of pascals of pressure.

Further complicating matters is the fact that sound waves arrive at a receiver from some direction, allowing the receiver to spatialize the source direction. Humans can spatialize in three dimensions since our ears are asymmetrical vertically and horizontally. The asymmetrical shape means that a wave will scatter differently based on arrival direction. This effect requires at least two output channels for realistic spatialization cues to be reconstructed (i.e. binaural audio). The scattering effects of ear, head, and shoulder shape is usually encoded in special Head Related Transfer Functions (HRTF). In cases of more complex output, such as moving sources and receivers or high numbers of output channels, more advanced reconstruction techniques are required.

If an unit impulse response is used as the input signal to an acoustic simulator, an impulse response is generated. This response represents how the environment modifies the input signal. The impulse response can then be convolved with any signal to auralize output. Depending on the type of simulator, the impulse response can measure the pressure response or the energy response of the environment.

1.5 Thesis statement

Prior methods are restricted to only specular reflection and diffraction on dynamic scenes. Wave based solvers can simulate all wave effects, but are too slow for any kind of dynamic scenes. Even when precomputing the propagation, current methods are restricted to scenes of tens of meters and a few sources.

Geometrical acoustics does not fully model the wave equation and needs additional effects to simulate realistic sound propagation. Such simulations should harness the parallel capabilities of modern many-core CPUs and GPUs. Additionally, low-power mobile devices must also be considered when developing propagation algorithms, especially as virtual environments increase in size and number of users. Our thesis solves these problems:

> Using parallel ray tracing methods and precomputation algorithms, realistic interactive geometrical sound propagation can be performed on dynamic scenes and massive multi-user virtual environments.

1.6 Challenges

Interactive sound propagation is a challenging problem and potential solutions must match both the goal application and the type of target hardware. For example, some applications (e.g. games) may require low latency simulations of sound propagation, others (architectural acoustics) may require high accuracy, while other may have combinations of multiple requirements. Some hardware may support high single thread performance, some other hardware may support hundreds of low performance threads, while another architecture is heterogeneous. Given the variation in requirements, it is unlikely that a single algorithm will satisfy all needs for several decades.

When developing our algorithms, we considered current and likely future hardware trends. Compute hardware is increasingly moving away from fast single threaded models to wider parallel configurations. This trend is seen in CPU designs and GPU designs. Intel's most recent server architecture [4] supports 120 threads on 60 cores; the most recent GPUs from AMD [5] and NVIDIA [6] support more than 5,600 threads. Clearly, it is desirable to have algorithms that parallelize across CPUs and GPUs.

Another important trend in computing is the widespread use of mobile devices or thin-clients. Phones, media players, and other pocket computers are in widespread use. These devices also show a trend towards parallel architectures, but with much greater restrictions on power use. Often, mobile

[4] Intel Xeon Processor E7-4890 v2, http://ark.intel.com/products/75251/

[5] AMD Radeon R9 295X2, http://www.amd.com/en-us/products/graphics/desktop/r9/295x2

[6] NVIDIA GeForce GTX TITAN Z, http://blogs.nvidia.com/blog/2014/03/25/titan-z/

devices rely on networked servers (colloquially 'the cloud') to spend compute power on their behalf, then retrieve compute results over the network.

It can be difficult to design propagation algorithms that parallelize well on modern hardware. It requires that the global propagation solution be decomposed into a very high number of independent steps with similar workloads. Even highly parallel problems like ray tracing are non-trivial when implemented on actual hardware. Often, memory access patterns and cache issues become the limiting factors in such parallel algorithms.

Mobile implementations of sound propagation add further complexities. The mobile client often does not have the compute power or the battery power to simulate many propagation effects. If a backing server computes the propagation results, it must be able to handle many client renders in order to be effective. This means the propagation simulation must be formatted in a way to minimize the per-client compute cost.

Specific applications of sound propagation often require certain properties for plausible rendering. This can make simulation for some applications difficult. For example, if the source or receiver is fixed, optimizations can be employed in the simulation algorithm. However, many interactive applications, like video games, require that the source and receiver be allowed to move freely. Additionally, video games may require the entire scene to be dynamically updated when the simulation is running. Since many rendering methods assume a static scene, this presents significant difficulties.

Diffraction is another important property that is practically required in all propagation simulations that are used for auralization (or audio outout). Diffraction is especially difficult to model in geometrical acoustic methods since it results in a scattering effect. This can lead to very high compute costs when multiple diffracting edges interact. Diffraction is important because we rely on sound bending around corners to hear before we can see. Sound propagation without diffraction has unnatural discontinuities and shadow regions, where even nearby sound sources cannot be heard.

1.7 Contributions

In this thesis, we present algorithms for fast, accurate simulation of sound propagation in a medium of constant density. Moreover, these algorithms are designed to work well on commodity processors and scale to be capable to rendering acoustic effects in massive environments. We first

present a method for rendering diffraction effects using the Unified Theory of Diffraction (UTD) . We then use our diffraction algorithm to develop RESound, a CPU based unified ray engine that support specular reflections, diffraction, diffuse reflections, and reverberation effects. For GPUs, we present a multiview visibility algorithm that adapts to changing environments. Finally, we design a precomputation algorithm that can render hundreds of sources in parallel on massive multi-user environments. The details and contributions of each algorithm are shown below.

1.7.1 Diffraction modeling

Diffraction is one of the most important wave effects sound undergoes. When the sound wave encounters a boundary, the wave is reflected. If the boundary has discontinuities, the wave scatters based on wavelength. This effect is prominent at edges. For example, sounds propagate around open door ways, allowing people in a room to hear approaching footsteps before the person walking is visible.

Some propagation simulations directly simulate wave propagation and can render this effect without any special handling. However, other simulators only model high frequency effects and treat sound as linear rays. These simulators are called Geometrical Acoustic (GA) simulators since they primarily consider the bounding geometry and not the actual wave front. These simulators are often very fast, but cannot render important effects like diffraction without special additions.

We have designed efficient ways to augment GA simulators with diffraction effects. The diffraction rendering is based on the Unified Theory of Diffraction (UTD). This is the first algorithm capable of rendering diffraction effects in dynamic scenes at interactive rates.

1. **Interactive dynamic scenes:** Our algorithm can find diffraction paths at interactive rates for moving objects in dynamic scenes.

2. **Accurate simulation:** A subdivision process allows performance versus accuracy adjustment at runtime. We show that our algorithm approaches the accuracy of state-of-the-art GA methods for high subdivision levels.

1.7.2 RESound: unified propagation

While diffraction is a very important effect, there are still other important wave effects that must be handled. If a wave reflects off a surface with many discontinuities on the same scale as the wavelength, the wave will experience many diffraction effects. The wave will then scatter off the surface in many directions. This is called diffuse reflection.

Since modeling these very small scale diffractions is difficult, diffuse reflection is often considered a separate effect in GA simulation. We have developed an algorithm that can handle specular reflection, diffuse reflection, and diffraction in a single framework. All effects are supported on by a single Bounding Volume Hierarchy (BVH) acceleration data structure to reduce precomputation time cost and runtime memory cost. Moreover, this algorithm supports interactive scene dynamism of any type simultaneously: sources may move, receivers may move, and scene boundaries may transform.

1. **Unified ray model:** Using a single ray acceleration structure, we support specular reflections, diffuse reflections, and diffraction effects This allows most major acoustic effects to be simulated with a single method.

2. **Fully dynamic scenes:** We use recent BVH algorithms [Lauterbach et al., 2009] to quickly build and modify our ray acceleration structure to support moving sources, receivers, and objects. Our method is the first to support specular reflections, diffuse reflections, and diffraction in fully dynamic environments.

3. **Robust acoustic signals:** Propagation simulation provides the signals for the early acoustic response. We complement the early signal with a reverberation estimation to provide a full acoustic signal.

1.7.3 Guided visibility

As processor design shifts from fast single cores to many parallel cores, appropriate parallel algorithms must be developed. Some of the techniques in the unified algorithm described above map very well to CPU designs, but not to modern GPUs due to memory access and branch restrictions. We design a multiview algorithm that allows portions of the unified framework to effectively use the parallel computing power of GPUs.

In all GA methods, there is time cost in finding possible sound paths and verifying the paths as valid. The parallel simulation we designed is very flexible and can vary the time spent on finding paths versus validating them. With an easy mechanism to vary time cost, we develop a guidance algorithm that can adjust to find local minima in rendering time cost interactively with no loss in accuracy.

1. **Fully dynamic scenes:** Supports moving sources, receivers, and objects.

2. **Guided visibility and validation:** We present a novel algorithm to reduce the cost of the visibility and validation steps. Using simple algorithms, the cost of both operations can often be reduced while retaining an accurate set of sound propagation paths.

3. **Multi-viewpoint ray casting:** We describe a ray casting algorithm that performs approximate visible surface computations from multiple viewpoints in parallel. We use this to accelerate specular reflection calculations on GPUs.

4. **Diffraction computation by barycentric coordinates:** To enhance our implementation, we have developed a low cost method of detecting rays near diffracting edges. Using the barycentric coordinate of ray intersections, we can create an origin for diffraction propagation.

5. **Interactive auralization:** Using the above algorithms, we implemented a GPU based system to demonstrate the method.

1.7.4 Rendering massive multi-user environments

Given the above methods to quickly compute realistic sound propagation on CPUs and GPUs, it is natural to use them in interactive virtual environments. However, many modern clients are mobile devices and thin-clients that lack significant compute resources. For such devices, the propagation results can be precomputed. However, even with fast propagation simulation, computing propagation effects on large scene is challenging due to the time and space costs.

We present an algorithm that can select a small number of sample points in large scenes so as to minimize error while maintaining reasonable time and space costs. This algorithm performs low order sampling of the scene to discover the most critical sample points, then forms enclosing regions

where the sound field is likely to experience minimal change. A single sample is used for each region, reducing both time and storage costs.

We combine this reduction with efficient simulation, storage, and rendering techniques. Our algorithms support diffraction effects, multiple frequency bands, and full surround sound capabilities. We show that implementations of these algorithms can render hundreds of sources in scenes spanning tens of square kilometers in size.

1. **Geometrical acoustic similarity measure:** We introduce a geometric measure based on the properties that influence the acoustic field. The measure can be computed quickly using the local neighborhood of a given point location in the environment. This enables us to perform scene sampling in $O(n)$ time for n sample points (section 6.1).

2. **Scene subdivision:** We use the similarity measure to sample the virtual environment and then segment the scene into regions based on the measure. The full acoustic response is only sampled at the center of each region, resulting in a reduction of both the precomputation time and the storage overhead, and thereby enables us to handle very large scenes which span kilometers in virtual space.

3. **Efficient response storage:** We present an efficient approach that scales in both time and space complexity to accommodate large acoustic scenes. Our storage algorithm compresses redundant data while supporting fast inserts and constant average time retrieval. This enables efficient storage of the tens of billions of acoustic responses needed for kilometer-sized scenes.

1.8 Organization

The contributions of this thesis are divided into two main parts. We first discuss interactive GA simulations on dynamic scenes. Chapter 3 details how to improve the realism of geometrical acoustic methods by adding support for diffraction effects. We then cover a unified framework for multicore CPUs in Chapter 4 and a guided multiview algorithm for GPUs in Chapter 5.

We then discuss precomputation methods for mobile devices and thin-clients. Using variations of our interactive GA methods, we present an algorithm for simulation of sound propagation in massive multi-user virtual environments in Chapter 6.

CHAPTER 2: RELATED WORK

In this section, we give a brief overview of prior work in acoustic simulation. Acoustic simulation for virtual environment can be divided into three main components: sound synthesis, sound propagation, and audio rendering. In this dissertation, we focus on sound propagation and the necessary audio rendering. Modeling the creation of sound is only briefly discussed. Our work is based in GA methods, and previous methods are detailed. Audio output is also covered, since it is necessary to render the signal for the user to hear.

In many algorithms, the input signal can be model separately from the propagation. A response signal can be convolved with an input signal to output a modified version of the input signal. If the response signal is the unit impulse, the convolved output will be the same as the input signal. If propagation of a unit impulse is simulated, the output response signal represents the effect the environment had on the unit impulse. By the distributive property, the propagated unit impulse can be convolved with an input signal to produce a signal as if the input signal had been used in the simulation. This allows propagation to be decoupled from the input signal.

2.1 Sound synthesis

Sound synthesis generates audio signals based on interactions between the objects in a virtual environment. Synthesis techniques often rely on physical simulators to generate the forces and object interactions [Cook, 2002, O'Brien et al., 2002]. Many approaches have been proposed to synthesize sound from object interaction using offline [O'Brien et al., 2002] and online [Raghuvanshi and Lin, 2006, van den Doel, 1998, van den Doel et al., 2001] computations. Anechoic signals in a sound propagation engine can be replaced by synthetically generated audio signal as an input. Thus, these approaches are complementary to the presented work and could be combined with most propagation simulations for an immersive experience.

2.2 Sound propagation

Sound propagation deals with modeling how sound waves propagate through a medium. Wave effects such as reflections, transmission, and diffraction are the important components. The acoustic wave equation (AWE) 2.1 is a partial differential equation that predicts how sound waves behave in a medium with obstructions.

$$\frac{\partial^2 p}{\partial t^2} - c^2 \nabla^2 p = F(\mathbf{x}, t) \tag{2.1}$$

where $\mathbf{x}$ is the position, t is the time, p is the pressure as a function of position and time, c is the speed of sound, F is a forcing term representing sound sources, and ∇ is the Laplacian in 3D. This equation describes how the pressure p changes over time in response to dispersion $\nabla^2 p$ and the input source terms F.

The wave equation also has a frequency domain representation and the time domain and frequency domain representations can be solved by standard numerical techniques. Such simulations are quite accurate and discussed below. These simulations are unfortunately quite costly in terms of compute time and memory space. Geometrical Acoustics are approximate methods that model the wave front as particles. This is typically very fast to simulate, but is unacceptably inaccurate due to missing wave effects, notably diffraction. GA simulation methods and attempts at improving their accuracy are also discussed below.

2.2.1 Numerical solutions

The wave equation can be solved directly by numerical techniques, such as the Boundary Element Method (BEM), the Finite Element Method (FEM), the Finite-Difference Time-Domain method (FDTD), and Digital Waveguide Meshes (DWM), and others [Kleiner et al., 1991].

Each method discritizes space and solves the wave equation across the elements. For the BEM, the boundary elements are discritized, while for the FEM, space is divided into tetrahedra. FDTD methods [Botteldooren, 1994] divide space into a grid of cells and are the most common method of solving wave equations [Shlager and Schneider, 1995]. The DWM method [Duyne and Smith, 1993] is very similar to FDTDs.

For frequency f, compute costs for solving the AWE scales with f^4. Similarly, memory cost scales with scene volume. Thus, for 3D simulations, memory cost is the product of scene dimensions x, y, and z. These scaling issues limit the utility of wave based simulation methods.

There have been recent advances in making these simulations tractable on modern desktop computers. Adaptive Rectangular Decomposition (ARD) [Raghuvanshi et al., 2008] uses analytical solutions to the wave equations for carefully defined rectangular regions and has been used on larger scenes and higher frequencies than previous methods. FDTD [Savioja, 2010] and ARD [Mehra et al., 2012] can be implemented in a parallel efficient manner, leveraging modern parallel hardware like GPUs.

Even with these advances, large scenes and high frequencies remain a problem. Solutions have been proposed [Mehra et al., 2013, Raghuvanshi et al., 2010], but these still have very high compute costs over large scenes.

2.2.2 Geometrical methods

The most widely used methods for interactive sound propagation in virtual environments are based on geometrical acoustics. GA methods are so named because they compute sound propagation only accounting for the geometry which describes the scene. This is a high frequency approximation and essentially models sound waves as particles emitted from a source.

2.2.2.1 Image source

All GA methods are variations of the image source method. The image source method [Allen and Berkley, 1979, Borish, 1984] assumes a small wavelength relative to the objects in the scene and models only specular reflections. The goal of the image source method is to enumerate all valid specular reflection paths between a source and a receiver in an acoustic scene composed of planar walls. The general algorithm is to recursively reflect the source point about all of the geometry in the scene to find specular reflection paths.

The image source method works on 3D scenes composed of walls as planar surfaces. Given a sound source position, the source is reflected over all walls forming a reflected image source for each wall. These image sources represent a single level of reflection. For each subsequent level of reflection, each image source must be reflected over all walls. High orders of reflection are thus

extremely expensive, with w^r images for w walls and r orders of reflection. The basic image source method is tractable on small scenes and low orders of reflection.

The images represent every possible reflection sequence for the specified reflection order. There are likely to be many physically invalid sequences, so an image validation process is used to cull invalid sequences. Given a receiver position in the scene, line-of-sight between the receiver and the highest order of reflection images is verified to pass through the wall that caused the reflection image. Such a case represents an unoccluded and valid reflection path segment. This validation process repeats all the way to the source. Any sequences that are completely validated represent valid specular reflection paths between the source and the receiver. See Figure 2.1 for details.

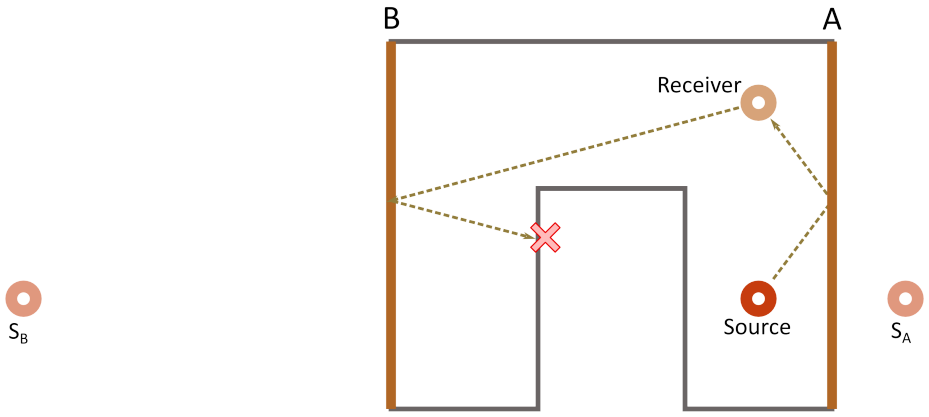


Figure 2.1: **Image source:** From the $Source$, image sources S_A and S_B are created over walls A and B respectively. Only S_A is occlusion free; a reflection sequence using S_B is physically impossible.

These paths represent the specular reflection that sound will follow when propagated from the source. Sound that propagates on these paths is attenuated by the medium and the reflections with the walls. Not all of the paths are the same total distance, so the signal from the source will be distributed over time when it arrives at the receiver.

2.2.2.2 Accelerated image source methods

The high cost of the image source method can be reduced by avoiding calculating images that are known to produce invalid paths. This is typically done using visibility queries. A visibility query is a geometric query that can tell if an object is in line-of-sight to a point or region. The validation step of the image source method uses a simple visibility query to verify that an image's reflecting wall is

unoccluded to the receiver. Visibility queries can also indicate if any portion of a wall is visible to a point. This can be used to avoid the creation of images that cannot lead to physical reflection paths. Consider if a wall is not visible to a source point. It is impossible for the sound to specularly reflect off a wall that is hidden from the source, so an image source should not be created for such a wall. Binary Spatial Partition (BSP) trees have been used to directly reduce the number of invalid sources created [Schröder and Lentz, 2006] in the image source algorithm.

Beam tracing: Many techniques designed for accelerating visual rendering are applicable to accelerating GA methods. Beam tracing was one of the earliest methods used, first described for visual rendering [Heckbert and Hanrahan, 1984] and later adapted for use in acoustics [Funkhouser et al., 1998]. In beam tracing, a view volume is projected out from the source and repeatedly clipped against the nearest polygons. The polygons that clip the initial volume represent surfaces that are visible for the first reflection order. View volumes can be reflected off the polygon surfaces to form reflection beams for the next order of reflection.

Computing all the beams is costly and can take several minutes to an hour to compute. Once computed, the visibility information can be stored in a visibility tree structure for fast queries. A validation step still takes place to compute the final path segments and measure the paths for signal output. The validation step is fast enough that interactive rendering of the propagation signal is possible.

Beam tracing was initially an expensive operation, requiring complex clips against all the objects in the space. Recently, Binary Spatial Partitioning (BSP) trees have been used to accelerate the clipping operations [Laine et al., 2009], resulting in beam tracers that can render 6 orders of reflections in a few minutes. Memory costs are quite high and scenes with high numbers of polygons are not viable.

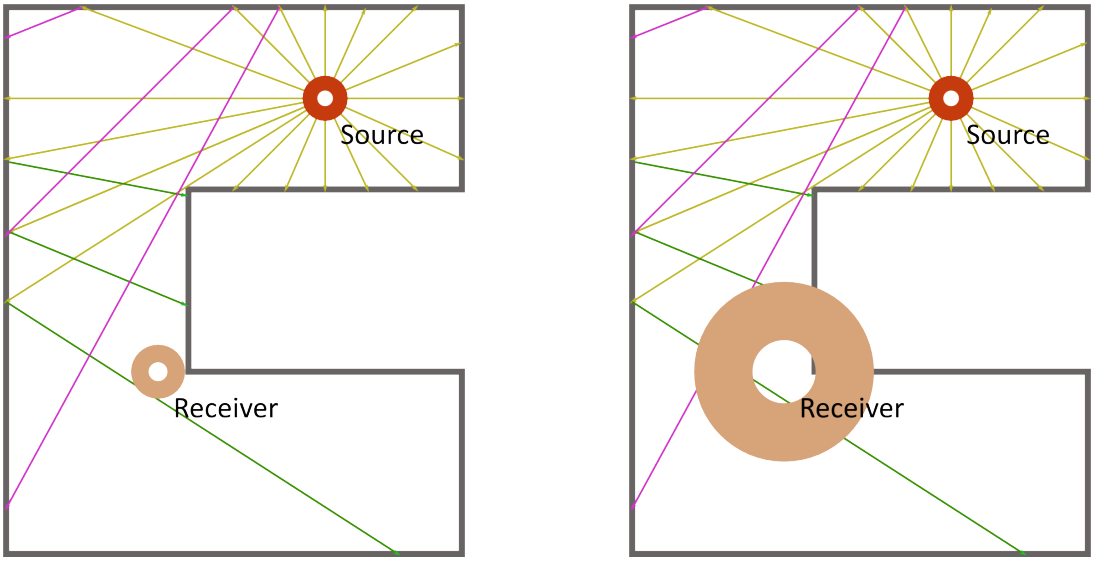
Ray tracing: The beam tracing method uses *object space* visibility, which means the visibility results are analytically calculated from scene data. Such GA methods thus compute the same answer as the brute force image-source algorithm. Other methods use less accurate visibility queries for large improvements in speed and capabilities, at the possible expense of accuracy.

Ray tracing is a type of visibility query that can test a ray for intersection against objects in a scene. If enough rays are cast into the scene, a rough estimate of the visible objects can be determined.

This is a *sample space* visibility query, since the visibility results are the result of repeated samplings of the scene. If the sampling density is too low, some visible objects may not be reported.

One of the earliest uses of ray tracing [Krokstad et al., 1968] precedes even the image source method. In the ray tracing method, many rays are traced from the source and reflected in the scene. The rays eventually intersect a receiver and their visibility information recorded.

Since intersecting an arbitrary ray with a receiver point is unlikely, a sphere is often used as a detector to collect the rays [Ondet and Barbry, 1989]. The size of the sphere is related to the number of rays collected, as well as the accuracy of the simulation. A large sphere size can lead to incorrect sound paths being detected [Lenhert, 1993]. Different methods have been developed to select an appropriate sphere size, usually based on the number of rays traced or distance between source and receiver [Lenhert, 1993, Xiangyang et al., 2003]. The methods for selecting an appropriate ray count are based the assumption that all surfaces are visible to any one source position [Lenhert, 1993]. In scenes where most paths occur after at least one reflection or the scene configuration changes, it is difficult to reliably predict an appropriate sampling density.



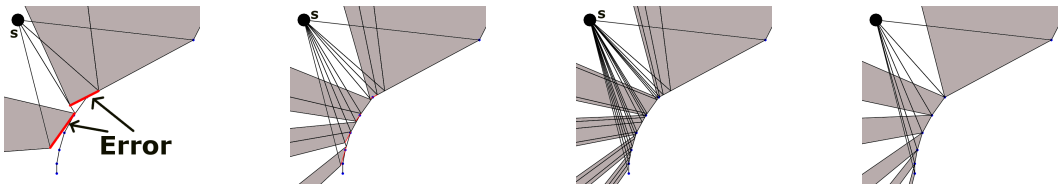
(a) (b)

Figure 2.2: **Receiver size:** For the subset of first order reflection rays shown, smaller receivers (a) result in fewer visibility samples needing to be culled, but may have aliasing artifacts if the scene changes slightly. Larger receivers (b) will have more sequences that must be validated and discarded, but fewer aliasing artifacts.

The visibility paths can be validated [Bertram et al., 2005, Svensson, 2008, Vorländer, 1989] as occlusion free. Simulations that use sample based visibility may not produce the same results as the image source method, especially if the sampling is sparse. However, if enough initial samples are used, the complete set of acoustic paths can be found and the output matches the image source method.

Radiance transport: Radiosity based methods are similar to BEM methods in that they discritize the scene surface into patches. A visibility factor is computed between all patches and is used to compute attenuate between any two points in the scene [Siltanen et al., 2007, 2009]. Similarly, the acoustic radiance of a scene can be computed and stored efficiently for interactive auralization [Antani et al., 2012a,b]. Radiance transfer methods improve on radiosity by allowing endpoint motion, while radiosity requires the source to be fixed.

Discrete volumes: Ray tracing techniques are fast, but can suffer from aliasing or sampling errors. Beam tracing is a volumetric technique and performs accurate geometric propagation, but relies on elaborate clipping algorithms and acceleration structures that are limited to static scenes. Frustum tracing [Lauterbach et al., 2007a] attempts to strike a balance between ray tracing and beam tracing by performing discrete clipping along with volumetric tracing. This greatly reduces sampling issues and uses hierarchical acceleration structures to handle complex, dynamic scenes. Frustum tracing traces convex frustum primitives through the scene.



(a) (b) (c) (d)

Figure 2.3: **Adaptive Frustum Tracing [Chandak et al., 2008]:** Adaptive Frustum Tracing traces frusta primitive from the source S (a). As subdivision increases (b,c), frustum tracing approaches the ideal solution (d).

AD-Frustum [Chandak et al., 2008] is an adaptive subdivision algorithm for frustum tracing. This algorithm automatically increases detail level at object edges (Figure 2.3) by subdividing large frusta into smaller frusta. This increases accuracy at boundaries, while maintaining high performance by using large frusta when intersecting object interiors. FastV [Chandak et al., 2009] uses similar

visibility primitives, but computes conservative visibility results. This means that the visibility queries may include walls that are not necessarily visible to the source. While this increases the number of validation tests needed, the final validated results match the analytical solution. Additionally, the validation can be completed at interactive rates and the initial tree computation is much faster than beam tracing.

2.2.2.3 Additional wave effects

The GA methods above mainly support specular reflection. However, sound waves may undergo many other types of interactions with the environment. The wave could reflected in a scattered manner or the wave could bend around corners.

Diffuse reflections: The methods thus discussed compute pressure values and supporting wave interference. For diffuse reflections, the reflection waves scatter and interference effects become less prominent. If interference simulation is not needed, energy based methods can be used. Energy based methods act as if each ray carried part of the acoustic energy and some of that energy is collected at the receiver and integrated. Such methods can avoid the validation step, since unique reflection paths are not needed. However, in this type of model, the receiver size must be set correctly to avoid integrating too much energy. Diffuse reflections have been shown to be important for modeling sound propagation [Dalenbäck et al., 1994] and are often modeled with such simulators, since interference effects are minor. Cone tracing [Dalenbäck, 1996], variations of ray tracing [Kapralos et al., 2004], and similar methods have been used to compute diffuse reflections.

Diffraction: There has also been work on complementing GA methods with edge diffraction effects. Diffraction effects are very noticeable at corners, as the diffraction causes the sound wave to propagate in regions that are not directly visible to the sound source. The two primary diffraction models used in geometrical simulations are the Uniform Theory of Diffraction (UTD) [Kouyoumjian and Pathak, 1974] and the Biot-Tolstoy-Medwin (BTM) [Biot and Tolstoy, 1957, Medwin, 1981] method. A time domain formulation of BTM [Svensson et al., 1999] is very useful in acoustics. These methods are widely used since they describe the diffraction of a ray path incident on an edge. The BTM method is considered more accurate than UTD and can be formulated for use with finite edges. However, the BTM method is compute intensive, which has led to different techniques to improve its performance [Antani et al., 2012c, Calamia and Svensson, 2005, 2007].

Due to its high cost, the BTM methods has only recently been used in interactive simulation [Antani et al., 2012c, Schröder and Pohl, 2009]. The UTD, however, has been adapted for use in several interactive simulations [Antonacci et al., 2004, Cowan and Kapralos, 2011, Schissler and Manocha, 2011, Taylor et al., 2009b, Tsingos et al., 2001]. The UTD assumes all edges are infinite, which results in low computational requirements. Even with this restrictive assumption, the UTD has been used to calculate diffraction coefficients for several interactive simulations, based on beam tracing [Tsingos et al., 2001] and 2D visibility diagrams [Antonacci et al., 2004]. Such simulations compare well to measured results [Tsingos et al., 2002]. In practice, these approaches have been mainly limited to static scenes.

Reverberation: Sound wave can propagate for many hundreds of reflections. After so many reflections, an individual wavefront may not have enough energy to be audible. However, constructive interference can allow low energy waves to form audible signals. In some enclosed spaces like cathedrals and music halls, these late reflections have a significant effect. Along with the early reflections, the later acoustic response must also be calculated [Hodgson, 1990]. This is often done through statistical methods [Eyring, 1930] or ray tracing [Embrechts, 2000].

2.3 Audio rendering

Audio rendering generates the final audio signal which can be heard by a listener over the headphones or speakers [Kuttruff, 2007]. In the context of geometrical sound propagation, it involves convolving the impulse response computed by the propagation algorithm with an anechoic input audio signal and introducing 3D cues in the final audio signal to simulate the direction of incoming sound waves.

In many interactive virtual environments (e.g. video games), the environment responses are not computed based on the actual scene. Instead, reverberation is either directly pre-rendered into the sound effects or implemented at run-time using dynamic artificial reverberation filters [Jot, 1999]. Parameters of reverberation decay can be directly manipulated by the sound designer to achieve a desired effect without requiring any geometrical modeling [3D Working Group of Interactive 3D Audio SIG, 1999]. While simplifying the authoring process, traditional artificial reverberators suffer from a number of issues. They impose a "single room" model [Eyring, 1930] and constrain

the shape of the decay profile (e.g., exponential). Because of their limited use of scene geometric representation, these methods also fail to convincingly model coupled or outdoor spaces. In [Bailey and Brumitt, 2010], distance histograms extracted from a cube map rendered at a position of interest are used to select the parameters for such an artificial reverberator. However, distances to the camera origin alone fail to capture local variations of the surfaces, which have a strong influence on the scattering properties [Tsingos et al., 2007].

Moving sound sources, receivers, and scene objects can cause variations in the impulse response from source to receiver and could lead to artifacts in the final audio output. Several methods have been proposed to reduce the artifacts in scenes with moving sources and receivers, including motion prediction [Tsingos, 2001], simple interpolation and windowing techniques [Savioja et al., 2002, Siltanen et al., 2009, Wenzel et al., 2000], and imposing restrictions on source and receiver motion [Taylor et al., 2009a]. Furthermore, many techniques have been proposed to reduce the runtime computational cost of 3D audio in scenarios with large number of sound sources (including virtual sources) based on clustering [Tsingos et al., 2004, Wand and Straßer, 2004] and perceptual methods [Moeck et al., 2007].

Simple binaural output (stereo output) is easy to implement and very common, but cannot represent sound sources in full 3D [3D Working Group of Interactive 3D Audio SIG, 1998] space. Introducing 3D cues in the final audio signals requires convolution of an incoming sound wave with a Head Related Impulse Response (HRIR) [Algazi et al., 2001, Larcher et al., 2000]. This can only be performed for a few sound sources in real-time. Recent approaches based on audio perception [Moeck et al., 2007, Tsingos et al., 2004] and sampling of sound sources [Wand and Straßer, 2004] can handle 3D sound for thousands of sound sources.

It is often desirable to reproduce sounds on multichannel surround output systems. Wave field synthesis [Berkhout et al., 1993] is based on Huygens's superposition principle and uses an array of output speakers to create virtual sound sources in an environment. Ambisonics [Gerzon, 1973] is a method of encoding the sound field using a spherical harmonic basis. Ambisonics has robust multichannel support, since output can be decoded per channel. Spatial Impulse Response Rendering (SIRR) [Merimaa and Pulkki, 2005] is another method for rendering multichannel output. By storing time domain direction and diffuseness information for several frequency bands, SIRR rendering algorithms can reproduce spatial audio over any type of multichannel system.

2.4 Voice communication

For networked virtual environments, such as social communities or massively multiplayer online (MMO) games, meaningful interaction through voice conversation with other participants is a valuable feature [Sallnäs, 2005, Wadley et al., 2007, Williams et al., 2007]. First adopted through side-clients that enabled telephone-quality, walkie-talkie style communication, voice services are becoming more integrated and are now connecting hundreds of millions of users on PCs, game consoles and cell phones. For instance, group voice chat is integral to gaming services such as Microsoft Xbox LIVE, Sony Playstation Network, and Valve Steam, and is also directly integrated into such games as Blizzard's World of Warcraft, CCP Games's EVE Online, Electronic Arts's Need for Speed World, and Linden Lab's Second Life.

There has been much work on voice communication over the internet, typically called VoIP (Voice over Internet Protocol) [Goode, 2002]. Some of the key issues include latency, voice coding efficiency, network error resilience, and endpoint voice cleaning and processing [Benesty et al., 2000, Markopoulou et al., 2002]. While most work in scalable VoIP focuses on the infrastructure system, there has also been significant research on improving the immersive effects of the voice communication, typically through spatialized rendering [Hollier et al., 1997, West et al., 1992]. Studies have shown [Gibbs et al., 2006, Halloran, 2009] that, while voice communication helps users coordinate in virtual environments, a lack of environmental effects can make it difficult for users identify sound sources.

Modeling the effects of sound propagation, such as occlusion and echoes, can help convey scenes where participants communicate from different rooms or areas. For example, the direction of the direct sound path and early sound reflections can help a user spatialize the sound source position, while the time it takes for the late reverberation relfections to decay conveys the scale of the environment and the materials present. However, these acoustic effects are difficult to implement in a large-scale VoIP system. Since the voice mixing is generally performed on a remote server, network delivery cost restricts the amount of data that can be transferred. Moreover, a typical multi-core server platform must handle thousands of remote clients simultaneously [Dolby, 2012], strongly limiting the processing capabilities. Peer-to-peer VoIP systems remove the need for a server system to process all client audio, but push the communication and simulation cost to the clients. In these

systems, implementing spatialized audio is possible [Zimmermann and Liang, 2008], but that only supports distance attenuation.

Even VoIP systems that include effects beyond simple direct-path distance attenuation [Boustead and Safaei, 2004, Radenkovic et al., 2002, Safaei, 2005] only support direct line-of-sight occlusion modeling and simplified diffraction effects, which result in unrealistic proximity cues. This is because the storage or compute cost of propagation simulations (discussed below) is quite high. This load must be either borne by the clients (handling 3-4 simulations in real-time) or the server (handling hundreds of simulations in real-time).

Client-server solutions have been proposed to dynamically compute sound propagation paths using the actual scene representation [Funkhouser et al., 1999]. But even the most recent geometrical acoustic (GA) approaches, which can model dynamic sound reflection and diffraction interactively [Taylor et al., 2012], cannot scale to large environments with a high number of sources. Other geometrical acoustic methods [Chandak et al., 2009, Funkhouser et al., 1998] take advantage of the static nature of the scene to precompute a visibility tree. However, these methods assume that either the source or the receiver position is fixed.

One practical approach to simulating the acoustics of virtual environments is to pre-compute the acoustical impulse response (IR) at several locations throughout the environment in an off-line process; the results can then be efficiently re-used to process the audio signals at run-time by querying the response database and reconstructing a reasonable solution for a given source and microphone pair [Antani et al., 2012a, Mehra et al., 2013, Pope et al., 1999, Raghuvanshi et al., 2010, Siltanen et al., 2009, Tsingos, 2009]. The main benefit of the off-line computation is that both early and high-order sound scattering (reflection/diffraction) can be simulated, providing improved proximity cues and distance perception. However, most of these techniques have been designed for small indoor or outdoor acoustic spaces with only a few sources or objects and may not scale to large virtual worlds with a high number of sources and receivers.

CHAPTER 3: FRUSTUM DIFFRACTION

Ray-frustum tracing has been shown to be an efficient algorithm for GA simulation [Lauterbach et al., 2007b]. However, it is limited to specular reflections and suffers from sharp discontinuities in the sound field without diffraction effects.

The frustum tracing algorithm is described in Section 2.2.2. The underlying algorithm generates 4-sided frusta based on specular reflections and edge diffractions and intersects the frusta with the scene primitives. After intersection, if a frustum is not fully contained within a scene triangle, part of the frustum must lie outside the triangle edges. To find the region outside the triangle, the frustum is sub-divided into sub-frusta using a quad-tree structure. These sub-frusta are then intersected with the triangle and the process repeats to a user defined limit. This sub-division allows a frustum to more accurately represent the shape of the scene primitives encountered. For more details, we refer to [Chandak et al., 2008, Lauterbach et al., 2007b].

In this chapter, we present our algorithm for support ray-frustum diffraction in detail and address the issues that arise in terms of incorporating edge diffraction in ray-frustum tracing.

3.1 Algorithm

In this section, we present our algorithm for diffraction in detail and address the issues that arise in terms of incorporating edge diffraction in ray-frustum tracing. The frustum tracing algorithm is described in Section 2.2.2. The underlying algorithm generates 4-sided frusta based on specular reflections and edge diffractions and intersects the frusta with the scene primitives. After intersection, if a frustum is not fully contained within a scene triangle, part of the frustum must lie outside the triangle edges. To find the region outside the triangle, the frustum is sub-divided into sub-frusta using a quad-tree structure. These sub-frusta are then intersected with the triangle and the process repeats to a user defined limit. This sub-division allows a frustum to more accurately represent the shape of the scene primitives encountered.

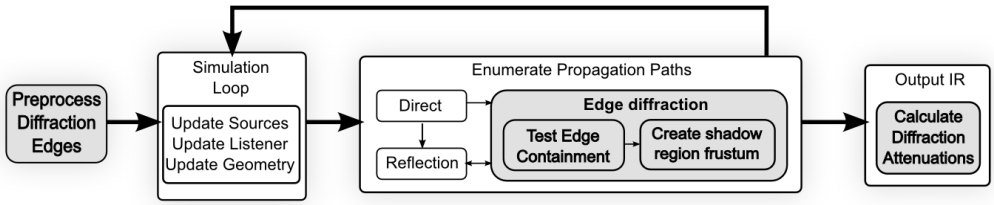


Figure 3.1: **Overview of our edge diffraction algorithm:** Possible diffracting edges are detected and marked as a preprocess. During the simulation, frusta are checked for diffracting edge containment. If so, a new diffraction frustum is created. After the propagation is complete, the diffraction paths are attenuated by the UTD coefficients.

In order to reduce the runtime overhead of checking if a triangle edge can diffract, our algorithm precomputes all of the diffraction edges in the scene. When a sub-frustum is found to contain a potentially diffracting edge, a new frustum is created to contain the possible diffraction contributions. After all propagation paths are found, they are attenuated based on the path characterization and scene primitives as described in Section 3.1.5. Figure 3.1 shows the various steps of our algorithm. While we are immediately concerned with the direct, specular, and diffraction components, our algorithm could also be combined with diffuse and late reverberation calculations for more accurate simulations.

3.1.1 Preprocess

Since the UTD is formulated for long edges, we allow the user to set a minimum diffracting edge length. Rather than explicitly testing whether an edge is a candidate for diffraction at runtime, we find all possible diffraction edges as part of a preprocess by analyzing the scene and marking possible diffracting edges. Specifically, we use a data structure that links each edge to its incident triangles. The edges of each triangle are classified based on the surface normals of the incident faces (see Figure 3.2).

If the triangles incident to an edge have similar normals, they are considered almost planar and the resulting edge is not considered as a candidate for diffraction (see Figure 3.2(a)). If the normals are exterior and point away from one another, the edge is part of a diffracting wedge (Figure 3.2(b)). The two surface normals are used to compute the wedge angle that is later used for calculating the diffraction coefficients. There are two other remaining cases that can be marked as diffracting or

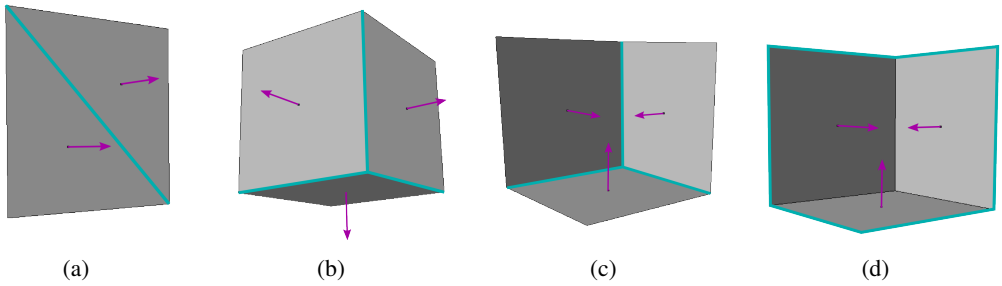


Figure 3.2: **Preprocessed edge types:** (a) Planar edges that never diffract; (b) exterior edges that always diffract; (c) interior edges and (d) disconnected edges that can be configured by user choice to diffract.

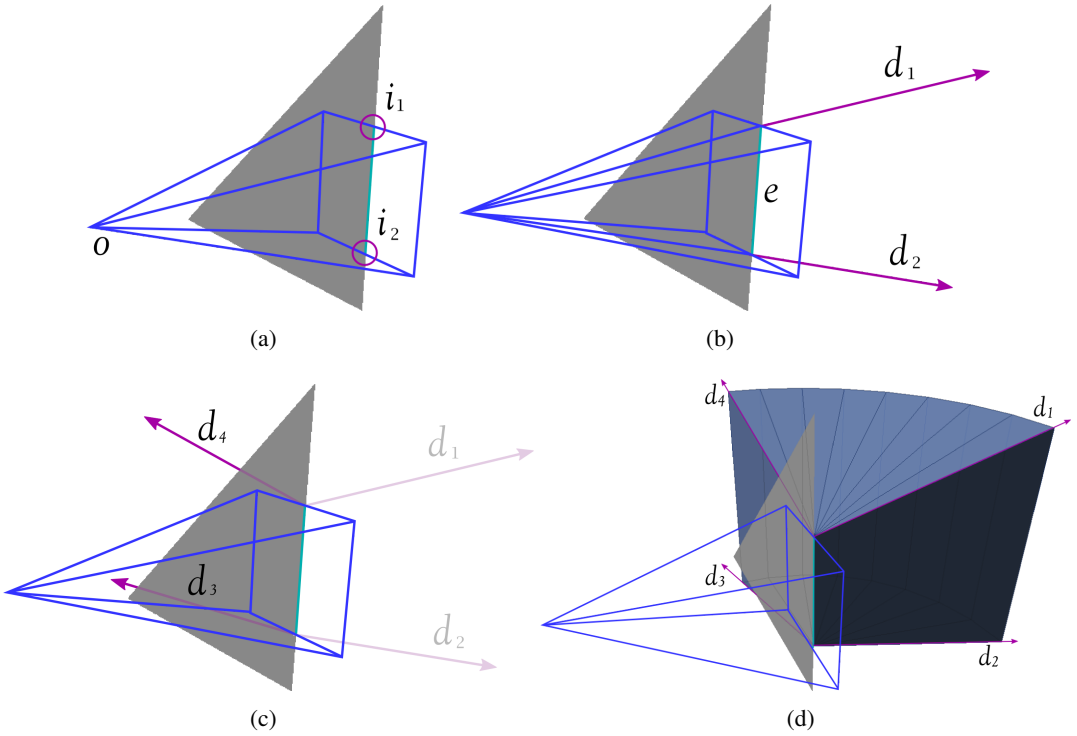


Figure 3.3: **Diffraction frustum creation:** (a) Given a frustum's origin o and its edge intersection points i_1 and i_2, (b) the edge axis e and the initial diffraction vectors d_1 and d_2 are created. (c) Rotating d_1 and d_2 about the edge axis towards the far side of the diffracting wedge sweeps a diffraction cone in the shadow region bounded by the final vectors d_3 and d_4. (d) We create complete the frustum volume.

non-diffracting depending on the scene design. In scenes where triangles form both the interior and exterior sides of a wall, the user can elect to have triangles with normals facing inwards (Figure 3.2(c)) marked as diffracting edges from the backface. Similarly, disconnected edge (i.e. triangles without neighbors, see Figure 3.2(d)) can be marked as diffracting edge if the user desires. If marked,

each disconnected edge would have a wedge angle of 2π radians. Edges may also be marked by other general criteria, such as direct user selection or minimum and maximum length.

3.1.2 Edge containment

During scene traversal, it is necessary to identify the diffracting edges that are contained within a propagating frustum. Consider the case where a frustum intersects a triangle and is not fully contained within the triangle. In this case, at least one of the corners of the frustum face lies outside of the triangle edges (Figure 3.4(a)). After many iterations of the adaptive frustum subdivision, the subdivision limit is reached and the edge is approximated by many sub-frusta, as shown in Figure 3.4(b). Some of these sub-frusta must contain the edge that caused the initial subdivision.

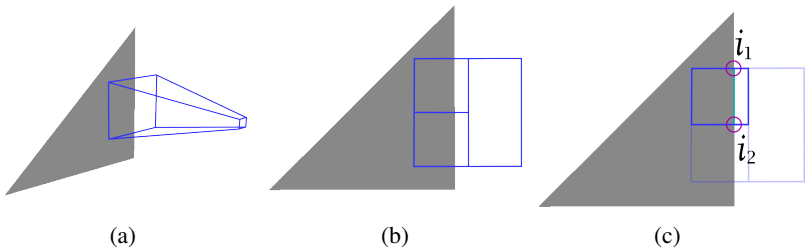


Figure 3.4: **Edge containment check:** After the frustum encounters a triangle (a), its face is projected into the triangle plane (b). Each diffracting edge is then checked for intersection with the face (c) to find the intersection points i_1 and i_2.

A series of tests determine whether a diffracting edge is contained within a frustum and thus whether we need to compute a diffraction frustum. Since the preprocessing step has assigned each of the triangle's edges a type, if none are marked as a diffracting edge, the test terminates. However, if any of the edges are diffracting, it is necessary to find the portion of each edge that is exposed to the sound field. This is tested by performing intersection between the edge and the four lines that form the frustum face boundaries in the plane of the triangle. The diffracting edge is checked for intersection against the boundaries of the frustum face. If the edge does not cross the bounds of the sub-frustum, the diffracting edge must not be within this sub-frustum and the test is repeated with the next sub-frustum.

In the case that the frustum boundaries intersect the diffracting edge, the exact orientation of the edge within the frustum needs to be determined. This is performed by completing the intersection

calculation and finding the two intersection points i_1, i_2 (see Figure 3.4 (c)), of the edge on the frustum boundary. These intersection points are used in the construction of the diffraction frustum.

3.1.3 Diffraction frustum construction

When a diffracting edge is found within a frustum, a diffraction frustum is created and propagated through the scene. This diffraction frustum should encompass the shadow region that is hidden from the direct contribution or specular contribution. We will now detail the calculations used during frustum creation (see Figure 3.3). Since most diffracting edges are located at wedges formed where two triangles meet, we will differentiate between the two triangles as the *source* side and the *receiver* side of the wedge. The source side is the side that is exposed to the original propagation path; the receiver side is the side where the new diffracted field will propagate in the shadow region.

In order to create a diffraction frustum, given the diffracting edge, the region of the edge that is contained within the initial frustum must be known. The intersection points from the edge containment test describe this portion of the edge. Using these points i_1 and i_2 on the edge and the origin of the original frustum o, two new vectors d_1 and d_2 are defined as $d_1 = i_1 - o$ and $d_2 = i_2 - o$. These vectors describe the side of the new diffraction frustum that borders the transition from line-of-sight contribution to shadow contribution.

Next, we construct the vectors that are used to represent the far plane of the diffraction frustum. This far plane will border the face of the triangle on the receiver side of the diffracting edge, and combined with the first set of vectors, bounds a portion of the shadow region. We begin the computation by defining an edge axis vector $e = i_1 - i_2$. There is a vector f_r which is perpendicular to the diffracting edge and lies in the plane of the triangle that represents the receiver side of the diffracting edge. This receiver face vector is defined as $f_r = e \times n_r$, where n_r is the normal of the receiver side triangle. We also compute the vector d_{perp} by projecting d_1 onto the plane perpendicular to e. Once these vectors are computed, we find the angle between them, and rotate d_1 and d_2 about e towards f_r by this angle. Beginning at d_1 and d_2, at intervals along the rotation, new frusta are created to approximate the diffraction cone, with the rotation ending at the vectors d_3 and d_4, which lie in the plane of the triangle corresponding to the receiver side.

In order to create the full diffraction region about the edge (not just the shadow region), the vectors d_3 and d_4 can be created efficiently as:

$$d_f = \begin{cases} (d_s \cdot e)e + (d_s \cdot f_r)f_r & \text{if } d_s \cdot f_r < 0 \\ (d_s \cdot e)e - (d_s \cdot f_r)f_r & \text{otherwise} \end{cases}$$

Where d_f is the resulting vector in the plane of the receiver triangle and d_s is a vector that borders the shadow region. Similar to the example in Figure 3.3, d_4 results from d_1 and d_3 results from d_2. The vectors d_4 and d_3 are then rotated about e towards the triangle face on the source side of the wedge. At intervals along this swept region, new frusta are created to approximate the diffraction region.

3.1.4 Path generation

The frustum tracing algorithm generates new reflection and edge diffraction frusta based on the intersections with scene primitives. As each frustum traverses the scene, the data needed to attenuate its contribution is pushed on a stack. This includes the data that describes the direction and location of the frustum and the geometric primitives encountered. The reflected frusta have the material attenuation values pushed, while diffraction frusta have the wedge angle and triangle data pushed onto the stack. This data is later used to create the contribution paths used in generating an IR (Impulse Response).

As each frustum is propagated through the scene, it is checked for containment of the receiver. If the receiver is contained in the frustum, there is some reflection or diffraction path from the source to the receiver. We wish to compute the path segment found inside the receiver containing frustum as well as the segments inside each parent frustum that was propagated up to the containing frustum. These path segments are computed from a linear combination of the four the rays that form the frustum edges. Together, these path segments represent the entire contribution path through the scene.

This is quick to compute and accurately represents the contribution path for specular reflection. However, for diffraction, there is slight error in the path vectors since the UTD cone is being approximated by a finite number of frusta. The details of this error are outlined in Section 3.2.2.

3.1.5 Attenuation

After all the frusta are computed, the contribution paths are used to compute an impulse response for the scene. Algorithms to compute the attenuation values for reflection have been described previously [Lauterbach et al., 2007b]. In this section, we highlight the calculation of the diffraction attenuation coefficients.

The UTD was chosen as the method to generate the diffraction coefficients for the paths since it works well with the discrete intersection tests performed for ray-frustum culling. The UTD assumes that an edge has infinite length and the actual length or subset of the edge that is exposed to sound energy is not used. Specifically, a single frustum can be checked if it contains a valid diffraction path completely independently of all other frusta.

Each path is attenuated using the UTD calculations for a user defined number of frequency bands (see Figure 3.5). Since only paths in the shadow region are attenuated, there will be a discontinuity in the field at the boundary of the shadow region. Near this boundary, approximate attenuation values are found by normalization [Tsingos et al., 2001]. This allows a smooth transition at the shadow boundary.

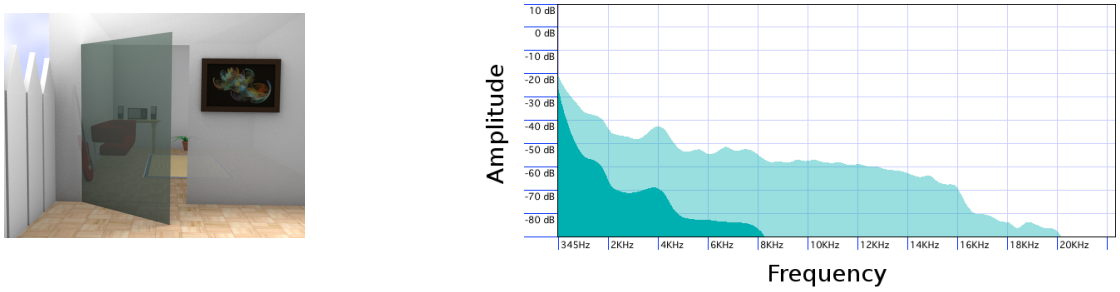


Figure 3.5: **UTD attenuation:** A radio is playing behind the door. The light green region shows the spectrum for the direct path when the door is open. The dark green region shows the spectrum of strongest diffraction path as the door closes.

3.2 Accuracy

In this section we validate the results computed by our edge diffraction algorithm against results from the well known Bell Lab Box benchmark [Tsingos et al., 2002]. We compare the number of paths our algorithm finds for various subdivision levels to the number of paths that the beam

tracing algorithm finds in the Bell Lab Box benchmark. In addition, we discuss the frustum tracing approximation of diffraction cones and detail the limitations of our approach.

Throughout this section and the next, we will refer to the subdivision level chosen for the frustum tracing system. Given a maximum subdivision level of x, a frustum may be recursively split up to 2^x times.

3.2.1 Bell Lab Box comparison

The "Bell Lab Box" is an acoustic benchmark. The Bell Lab Box (Figure 3.6) is a large closed box which contains a sound source and receiver. An optional baffle may be inserted that obstructs the visibility between portions of the box. This Bell Lab Box was used to conduct a controlled study [Tsingos et al., 2002] of audio diffraction. Knowing the scene dimensions and layout, physical measurements were compared to a beam tracing simulation of a similar virtual scene. The resulting output compares well with the physical measurements from the Bell Lab Box.

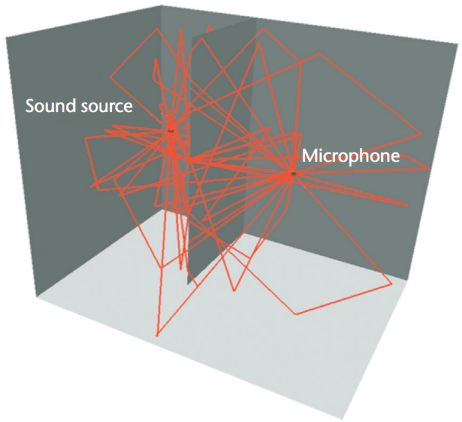


Figure 3.6: **Bell Lab Box [Tsingos et al., 2002]:** The Bell Lab Box is a simple room divided by a diffracting baffle. The image shows the 45 paths resulting from two orders of specular reflection and one order of diffraction.

The referenced paper provides the earliest 60 geometric paths traced from a source position to a receiver out of line-of-sight. We match the path sequences from their highly accurate geometric simulation to sequences generated by our diffraction frustum propagation. For these comparisons, we compute frusta for the full diffraction region about the edge as predicted by the UTD, instead of using the shadow region approximation.

In the Bell Lab Box path data there are a large number of paths that encounter the diffracting edge, and depart the edge traveling parallel along the diffracting wedge. Due to its basis on ray tracing methods, frustum tracing may not find these paths that are parallel to and travel along the plane of the diffracting wedge. Consider a frustum with corner rays that travel parallel to the wedge face plane. Even if there are more diffracting edges in this plane, they will not be found, since such edges are not contained within the frustum shape. This is similar to intersecting a ray with a triangle oriented such that the triangle normal is perpendicular to the ray direction.

We compare the number of paths found by each simulation in Figure 3.7. As the subdivision level of frustum tracing increases, more paths are found and the accuracy increases. For clarity, we include results from a hypothetical frustum tracing simulator that correctly calculates all paths except those that are parallel to the diffracting surface. As shown in the figure, the number of paths found by high subdivision frustum tracing compares very favorably with this ideal frustum tracer.

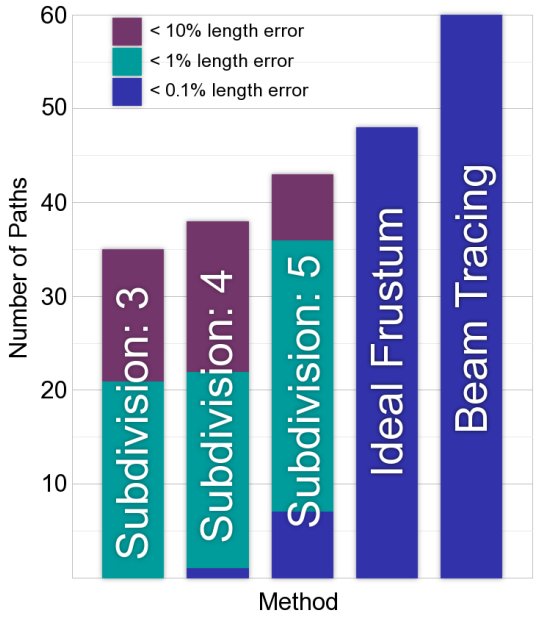


Figure 3.7: **Path length:** As subdivision level increases, more paths are found and the error decreases.

Since our algorithm has a time to accuracy tradeoff, we have measured the cost of various subdivision levels compared to beam tracing results on the Bell Lab Box (Table 3.1). All simulations compute 4 orders of specular reflection and 2 orders of diffraction. We compare the accuracy results by percentage of the first 60 paths found. Please note that the beam tracing time was measured in 2002. We estimate that CPU single thread performance has increased 3-4 times since then.

Subdivision	% paths found	Time cost (s)
3	58	2.5
4	63	11
5	73	47
beam	100	610

Table 3.1: **Time/accuracy cost:** We compare various subdivision levels to a beam tracing solution on the Bell Lab Box. Our method

3.2.2 Accuracy of diffraction frustum

Frustum tracing is an approximate method and can achieve high update rates by reducing simulation accuracy. Conversely, higher accuracy can be achieved by reducing the simulation update rate. The creation of diffraction frusta follows this same property. Diffraction frusta are initially subdivided based on the subdivision level chosen by the user. Since each diffraction frustum is bounded above and below by an approximate diffraction cone, it is helpful to evaluate the difference in volume between a subdivided approximation and a perfect diffracting cone. Figure 3.8 and Table 3.2 shows that as the subdivision level increases, the diffraction frustum quickly converges to the ideal volume.

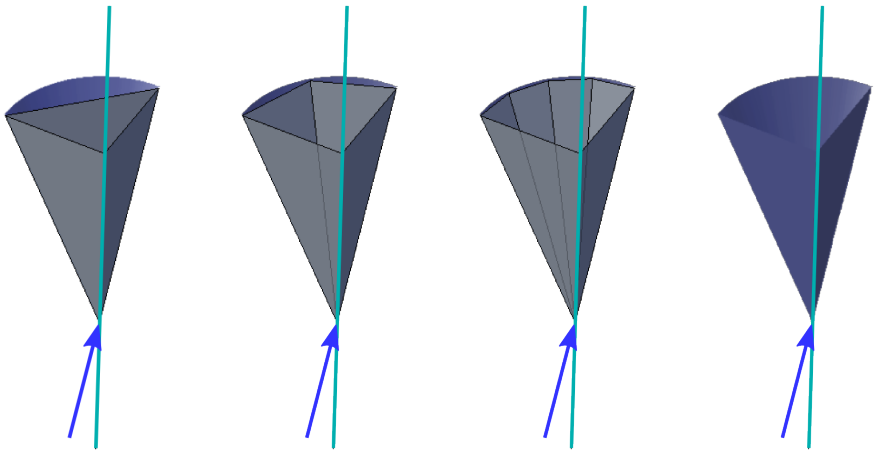


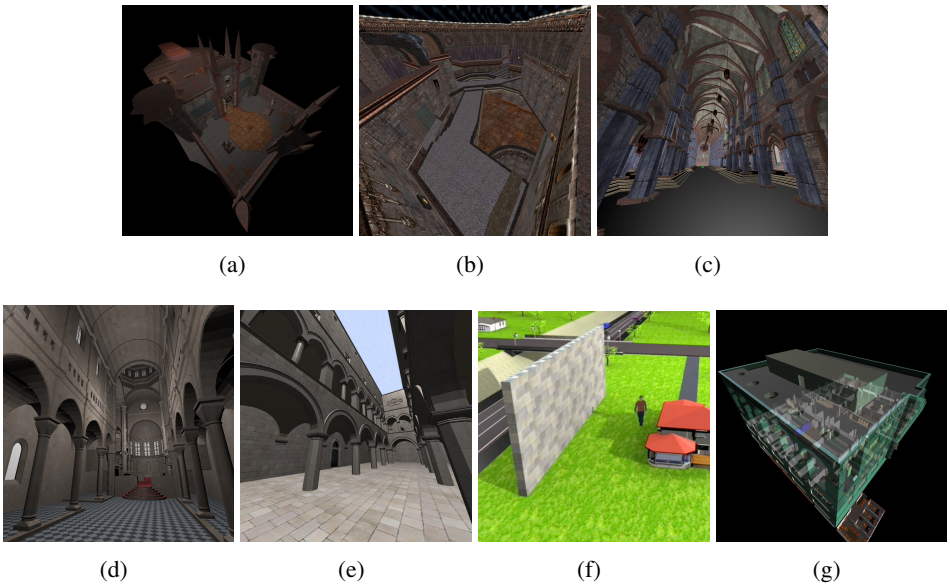
Figure 3.8: **Frustum subdivision accuracy:** the resulting diffraction cone with a subdivision of 0 (a), subdivision of 1 (b), and subdivision of 2 (c). The diffraction frustum approximates the ideal cone (d).

Subdivision level	Volume error
0	36.34%
1	9.97%
2	2.55%
Cone	0.0%

Table 3.2: **Volume error:** As the subdivision level increases, the error in the volume of the diffraction cone decreases

3.3 Performance

In this section we evaluate the performance of our method with various scenes and settings. Unless otherwise noted, all simulations are rendered with 8 frequency bands and 4 threads on a modern 2.66 Ghz multi-core machine. The scenes used are detailed in Table 3.3 and Figure 3.9. We use a maximum subdivision level of 3 for all benchmarks. Also, unlike in the Bell Lab Box comparison, we only propagate diffraction frusta in the shadow region.



(a) (b) (c)

(d) (e) (f) (g)

Figure 3.9: **Evaluation scenes:** (a) Q3dm1, (b) Atrocity, (c) Chartres, (d) Sibinek, (e) Sponza, (f) Highway, (g) Sodahall.

3.3.1 Diffraction cost and benefit

Generating the diffraction frusta during simulation incurs a time cost in addition to direct contributions and specular reflections. The time cost needed to propagate the diffraction frusta varies greatly from scene to scene due to the number of triangles and edges encountered. The benefit of

Scene	Type	Triangles	Diffracting edges
Q3dm1	Closed	14k	4032
Atrocity	Closed	12k	1531
Chartres	Open	192k	40489
Sibinek	Open	76k	1358
Sponza	Open	66k	1021
Highway	Open	350k	1248
Sodahall	Closed	1510k	9457

Table 3.3: **Scene overview:** Data on the scenes used for the performance results. Some scenes are very open with much geometry visible from any given point. Others are closed, with short visibility distances.

using diffraction also varies; depending on the scene layout and source/receiver position, very few valid diffraction paths may be found. Table 3.4 shows the added cost and benefit of using diffraction in various scenes with 3 orders of recursion. To highlight the effects of diffraction, in each scene we chose the source and receiver positions such that few specular paths are found. For example, in the highway scene, the receiver is placed behind an occluder that blocks all direct and specular contributions.

Scene	Diffraction	# Frusta	Time	Paths found
Q3dm1	Off	80844	219 ms	3
	On	114372	338 ms	5
Atrocity	Off	114183	282 ms	4
	On	140454	370 ms	7
Chartres	Off	219865	1306 ms	2
	On	292256	2078 ms	8
Sibinek	Off	370594	1614 ms	12
	On	377521	1636 ms	15
Sponza	Off	198022	861 ms	2
	On	209737	921 ms	11
Highway	Off	21178	62 ms	0
	On	23553	84 ms	5
Sodahall	Off	81269	436 ms	0
	On	91879	510 ms	3

Table 3.4: **Diffraction benefit:** Diffraction incurs a slight performance decrease, but often finds more propagation paths.

CHAPTER 4: RESOUND: A UNIFIED RAY FRAMEWORK

Diffraction is a form of wave scattering that occurs at discontinuities and results in the acoustic field having smooth transitions as sources and receivers move out of line-of-sight. In addition to the scattering is that caused by edge diffraction, variations in an object's material may cause incoming waves to reflect in a scattered manner. These *diffuse reflection* are an important acoustic effect when modeling environments, since they influences the overall acoustic field.

In this chapter, we introduce a unified simulation method called *RESound* which can simulate specular reflections, diffuse reflections, diffraction effects. These effects are complemented with statistical reverberation estimation, for a full and robust acoustic signal. Our system is the first that can render these effects in general dynamic scenes on a modern PC.

4.1 System overview

In this section, we give an overview of our approach and highlight the main components. RESound simulates the sound field in a scene using geometric acoustics (GA) methods.

4.1.1 Acoustic modeling

All GA techniques deal with finding propagation paths between each source and the listener. The sound waves travel from a source (e.g. a speaker) and arrive at a listener (e.g. a user) by traveling along multiple propagation paths representing different sequences of reflections, diffraction, and refractions at the surfaces of the environment. Figure 4.2 shows an example of such paths. In this chapter, we limit ourselves to reflections and diffraction paths. The overall effect of these propagation paths is to add reverberation (e.g. echoes) to the input sound signal. Geometric propagation algorithms need to account for different wave effects (described below) that directly influence the response generated at the listener.

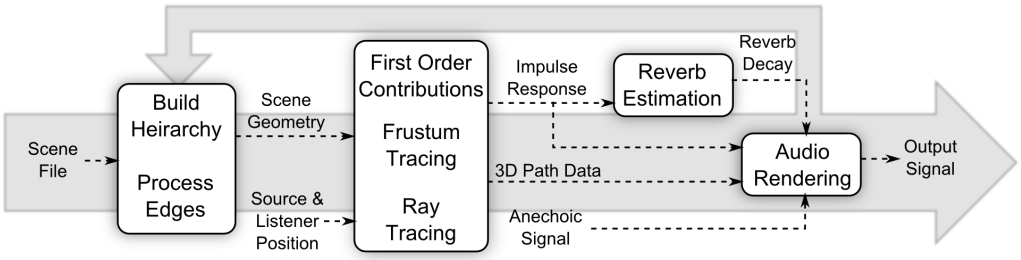


Figure 4.1: **The main components of RESound:** scene preprocessing; geometric propagation for specular, diffuse, and diffraction components; estimation of reverberation from impulse response; and final audio rendering.

(a) (b) (c)

Figure 4.2: **Example propagation paths:** This scene shows (a) specular, (b) diffraction, and (c) diffuse propagation paths.

When a small, point like, sound source generates non-directional sound, the pressure wave expands out in a spherical shape. Due to the spreading of the field, the amplitude at the listener is attenuated. The corresponding GA component is a direct path from the source to the listener. This path represents the sound field that is diminished by distance attenuation.

As the sound field propagates, it is likely that it will encounter objects in the scene. These objects may reflect or otherwise scatter the waves. If the object is large relative to the field's wavelength, the field is reflected specularly, as a mirror does for light waves. In GA, these paths are computed by enumerating all possible reflection paths from the source to the listener, which can be a very costly operation. There has been much research focused on reducing the cost of this calculation [Funkhouser et al., 2003], as most earlier methods were limited to static scenes with fixed sources. The delay and attenuation of these contributions helps the listener estimate the size of the propagation space and provides important directional cues about the environment.

Objects that are similar in size to the wavelength may also be encountered. When a sound wave encounters such an object, the wave is influenced by the object. We focus on two such scattering effects: edge diffraction and diffuse reflection.

Diffraction effects occur at the edges of objects and cause the sound field to be scattered around the edge. This scattering results in a smooth transition as a listener moves around edges. Most notably, diffraction produces a smooth transition when the line-of-sight between the source and listener is obstructed. The region behind an edge in which the diffraction field propagates is called the *shadow region*.

Surfaces that have fine details or roughness of the same order as the wavelength can diffusely reflect the sound wave. This means that the wave is not specularly reflected, but reflected in a *Lambertian* manner, such that the reflected direction is isotropic. These diffuse reflections complement the specular components [Dalenbäck et al., 1994].

As the sound field continues to propagate, the number of reflections and scattering components increases and the amplitude of these components decreases. The initial orders (e.g. up to four or six) of reflection are termed *early reflections*. These components have the greatest effect on a listener's ability to spatialize the sound. However, the early components are not sufficient to provide an accurate acoustic response for any given scene. The later reverberation effects are a function of the scene size and material [Eyring, 1930] and convey an important sense of space. For example, large cathedrals are often known for their long decay times. Without late reverberation, these decay times cannot be simulated.

4.1.2 Ray-based path tracing

RESound uses a unified ray representation for specular reflections, diffuse reflections, and diffraction path computations. The underlying framework exploits recent advances in interactive ray tracing in computer graphics literature. We compute diffuse reflections using a discrete ray representation [Lauterbach et al., 2006, Wald, 2004] and specular reflections and diffraction using a ray-frustum representation [Chandak et al., 2008, Lauterbach et al., 2007b]. A frustum is a convex combination of four corner rays. We use fast ray tracing algorithms to perform intersection tests for the discrete rays as well as volumetric frusta.

We assume that the scene is composed of triangles and is represented using a bounding volume hierarchy (BVH) of axis-aligned bounding boxes. A BVH can be used to handle dynamic scenes efficiently [Lauterbach et al., 2006]. The same underlying hierarchy is used for both discrete rays and ray-frusta as part of our unified representation. Rays are shot as ray packets [Lauterbach et al.,

2006] and efficient frustum culling is used for fast intersection of ray packets and frusta with the BVH. In order to perform fast intersection tests with scene triangles the frustum representation uses Pücker coordinates [Shoemake, 1998].

4.1.3 RESound components

Our system consists of three main processing steps. These are outlined in Figure 4.1.

Preprocessing: As part of preprocessing, a scene bounding volume hierarchy is created. This is a hierarchy of axis-aligned bounding boxes and is updated when the objects in the scene move. This hierarchy is used to perform fast intersection tests for discrete ray and frustum tracing. The edges of objects in the scene are also analyzed to determine appropriate edges for diffraction.

Interactive Sound Propagation: This stage computes the paths between the source and the listener. The direct path is quickly found by checking for obstruction between the source and listener. A volumetric frustum tracer is used to find the specular and edge diffraction paths. A stochastic ray tracer is used to compute the diffuse paths. These paths are adjusted for frequency band attenuation and converted to appropriate pressure components.

Audio Rendering: After the paths are computed, they need to be auralized. A statistical reverberation filter is estimated using the path data. Using the paths and the estimated filter as input, the waveform is attenuated by the auralization system. The resulting signal represents the acoustic response and is output to the system speakers.

4.2 Interactive sound propagation

In this section, we give an overview of our sound propagation algorithm. Propagation is the most expensive step in the overall sound rendering pipeline. This large computational cost arises from the calculation of the acoustic paths that sound takes as it is reflected or scattered by the objects in the scene. Under the assumption of geometric acoustics, this is primarily a visibility calculation. Thus, we have chosen rays as our propagation primitive. For example, the direct sound contribution is easily modeled by casting a ray between the source and listener. If the path is not obstructed, there is a direct contribution from the source to the listener. The other propagation components are more expensive to compute, but rely on similar visibility computations.

When computing the propagation components, many intersection computations between the scene triangles and the ray primitives are performed. In order to reduce the computation time, we minimize the cost of intersection tests using a BVH acceleration structure. Since this structure can be updated with refitting algorithms [Lauterbach et al., 2006], it allows elements of the scene to be dynamically moved and repositioned with little cost. Also, we mark all possible diffraction edges. This allows the diffraction propagation to abort early if the scene edge is not marked as a diffracting edge.

4.2.1 Specular paths

We use volumetric frustum tracing [Lauterbach et al., 2007b] to calculate the specular paths between the source and listener. From our basic ray primitive, we form a convex volume bounded by 4 rays. In order to model a uniform point sound source, we cast many of these frustum primitives such that all the space around the source is covered. For each frustum, the bounding rays of the volume are intersected with the scene primitives. After the rays have hit the geometric primitives, they are specularly reflected. This gives rise to another frustum that is recursively propagated. This continues until a specified order of reflection is achieved.

However, it is possible that the 4 bounding rays of the frustum did not all hit the same object in the scene. In this case, it cannot be guaranteed that the resulting specular frustum correctly contains the reflection volume. As such, we employ an adaptive subdivision strategy [Chandak et al., 2008] to reduce the error in the volume. If it is found that the 4 rays do not intersect the same geometric primitive, that is, the frustum face is not fully contained within the bounds of the geometric primitive, the frustum is subdivided using a quad-tree like structure into 4 sub-frusta. The sub-frusta are then intersected with the scene and the subdivision process continues until a user-defined subdivision level is reached. When the subdivision is complete, any ambiguous intersections are resolved by choosing the closest intersected object and reflecting the subdivided frustum's rays against it. This process results in a simulation that becomes more accurate as more time is spent subdividing.

Given any propagation frusta, if the listener is contained within the volume, there exists some sound path from the source to the listener. This path is verified by casting a ray from the listener towards the frustum origin. If the ray intersection point is contained in the frustum origin face on the triangle, the path segment is valid. This validation process is repeated using the computed

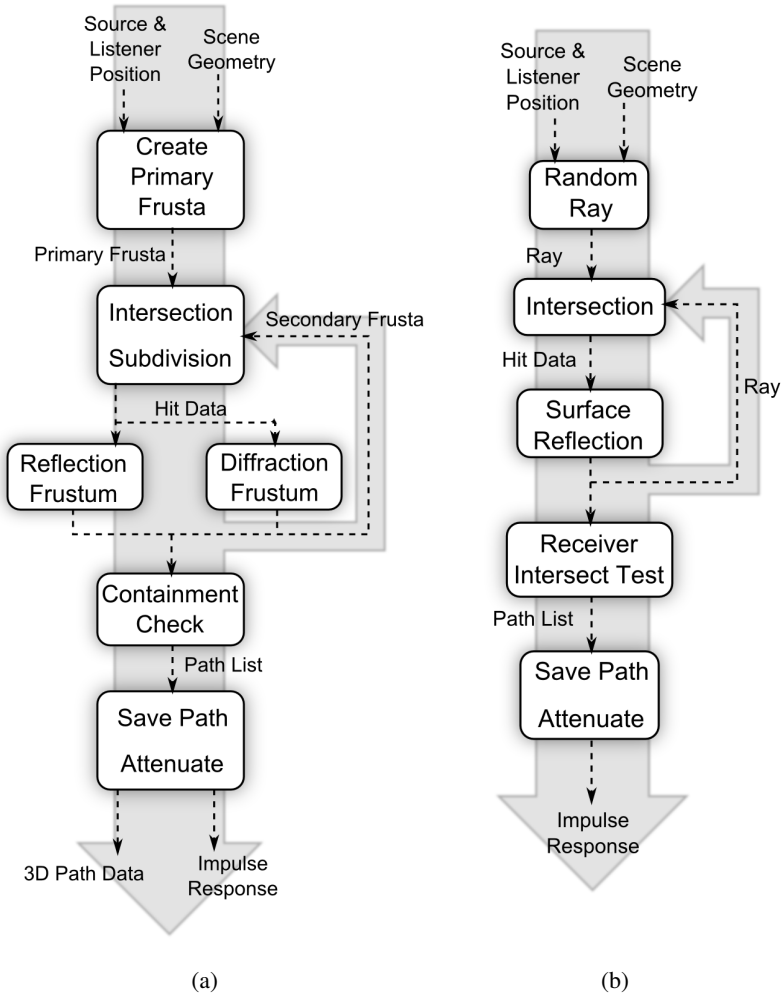


Figure 4.3: **Unified ray engine:** Both (a) frustum tracing and (b) ray tracing share a similar rendering pipeline.

intersection point to the origin of the previous frustum. If the entire path is valid, the path distance and attenuation are recorded. Figure 4.3(a) shows an overview of the frustum engine.

4.2.2 Edge diffraction paths

Frustum tracing can be modified to account for diffraction contributions [Taylor et al., 2009b] using the Uniform Theory of Diffraction (UTD) as described in Chapter 3. The UTD can be used to calculate the diffraction attenuation for ray paths used in GA. When a sound ray encounters an edge, the ray is scattered about the edge. In the UTD formulation, the region covered by the diffraction contribution is defined by the angle of the entrance ray. If a ray hits the edge with an angle of θ, the ray is scattered about the edge in a cone shape where the cone makes an angle θ with the edge.

As the frusta intersect the scene triangles, the triangle edges are checked to see if they are marked as diffracting edges. If the triangle has diffracting edges, and the edges are contained within the frustum face, a new diffraction frustum is created. Similar to other approaches [Taylor et al., 2009b, Tsingos et al., 2001], we compute the diffraction component only in the shadow region. As such, the frustum is bounded by the line-of-sight from the frustum origin and the far side of the triangle. This frustum then propagates through the scene as normal.

The final sound path is verified using the same process described for specular paths. However, for diffraction sequences, the path is attenuated using the UTD equation [Kouyoumjian and Pathak, 1974]. The UTD equation is in the frequency domain, and is thus computed for a number of frequency bands. The resulting UTD coefficients are combined with the attenuation for the other path segments to create the final path attenuation.

4.2.3 Diffuse component

In order to compute sound reflected off diffuse materials, we use a stochastic ray tracer (Figure 4.3(b)). Rays are propagated from the sound source in all directions. When a ray encounters a triangle it is reflected and tracing continues. The reflection direction is determined by the surface material. Some surfaces reflect mostly specularly, while others scatter diffusely. The listener is modeled by a sphere that approximates the listener's head. As the rays propagate, we check for intersections with this sphere. If there is an intersection, the path distance and the surfaces encountered are recorded for the audio rendering step.

The scattering coefficient for surface materials varies for different sound frequencies. Thus, for one frequency, the incoming rays may be heavily scattered, while for another frequency the reflection is mostly specular. Since intersecting rays with the objects in the scene is a costly operation, we wish to trace rays only once for all the frequencies. As such, for each ray intersection, we randomly select between diffuse and specular reflection [Embrechts, 2000]. The ratio of diffuse to specular paths is based on the acoustic material properties of the scene.

If a ray hits the listener, the ray's contribution must be computed for all frequency bands. We scale the energy for each frequency band appropriately based on the material properties and type of reflections selected [Embrechts, 2000]. For example, if a ray arrives at the receiver from an entirely diffuse path, the energy contribution for frequency bands that required specular reflection will be

heavily attenuated. If a path is found to be composed entirely of specular reflections, it is discarded as such paths are found in the frustum tracing step.

Once all paths have been computed and attenuated, each ray's resulting energy values must be integrated into a single response. Each path's time delay is quantized, so contributions with similar delays are summed into a single, larger contribution. The rays that contribute to this energy response represent a subset of the total energy emitted by the source. Thus, the energy in the response is scaled based on the number of rays that have been propagated from the source. Since the response is an energy (or sound power) value, the square root of the entire response is taken to compute a final pressure response.

4.3 Reverberation estimation

The propagation paths computed by the frustum tracer and stochastic ray tracer described in Section 4.2 are used only for the early reflections that reach the listener. While they provide important perceptual cues for spatial localization of the source, capturing late reflections (reverberation) contributes significantly to the perceived realism of the sound simulation. Reverberation is necessary to simulate the long decay in cathedrals or flutter echoes in tiled bathrooms. These effects are the results of hundreds of wave reflections and are prominent examples of late reverberation effects.

We use well-known statistical acoustics models to estimate the reverberant tail of the energy impulse response (IR). The Eyring model [Eyring, 1930] is one such model that describes the energy decay within a single room as a function of time:

$$E(t) = E_0 e^{\frac{cS}{4V} t \log(1-\alpha)} \tag{4.1}$$

where E_0 is the initial energy, c is the speed of sound, S is the total absorbing surface area of the room, V is the volume of the room and α is the average absorption coefficient of the surfaces in the room.

Given the energy IR computed using GA, we perform a linear least-squares fit to the IR in $\log$-space [ISO 3382, 2009]. This gives us an exponential curve which fits the IR and can be extrapolated to generate the reverberation tail. From the curve, we are most interested in estimating the RT_{60}, which is defined as the time required for the energy to decay by 60 dB. Given the slope computed by

the least-squares fit of the IR data, it is a simple matter to estimate the value of RT_{60}. This value is used in the audio rendering step to estimate late reverberation effects.

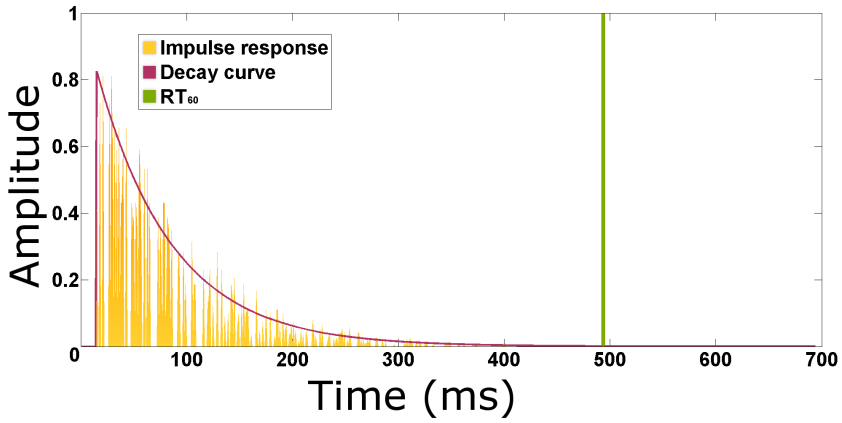


Figure 4.4: **Extrapolating the IR to estimate late reverberation:** The red curve is obtained from a least-squares fit (in log-space) of the energy IR. The green vertical line is the RT60 mark where the signal has decayed by 60 dB.

Note that Equation (4.1) is for a single-room model, and is not as accurate for scenes with multiple rooms (by "rooms" we mean regions of the scene which are separated by distinct apertures, such as doors or windows). The single-room model is a good approximation for large interior spaces and many outdoor scenes. Other models exist for coupled rooms [Summers et al., 2004], but they require fitting multiple curves to the impulse response IR, and the number of curves to fit would depend on the number of rooms in the scene. In the interests of speed and simplicity, we have chosen to use a single-room model.

4.4 Audio rendering

Audio rendering is the process of generating an audio signal which can be heard by a listener using headphones or speakers. In this section, we provide details on the real-time audio rendering pipeline implemented in our interactive sound propagation system. Our audio rendering pipeline is implemented using XAudio2[1], a cross-platform audio library for Windows and Xbox 360.

Our sound propagation algorithm generates a list of specular, diffuse, and diffracted paths from each source to the listener. These paths are accessed asynchronously by the audio rendering pipeline

[1] http://msdn.microsoft.com/en-us/library/bb694503(VS.85).aspx

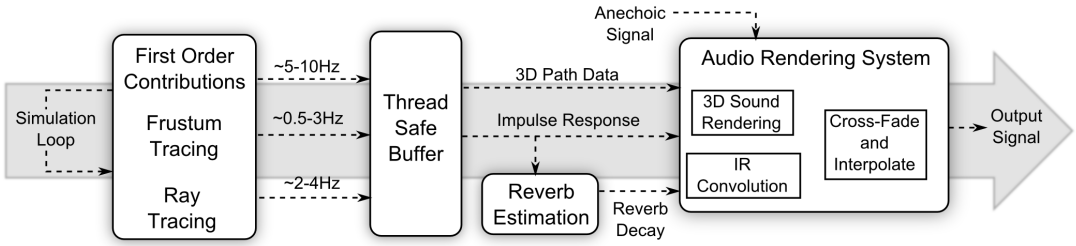


Figure 4.5: **Algorithm overview:** An overview of the integration of audio rendering system with the sound propagation engine. Sound propagation engine updates the computed paths in a thread safe buffer. The direct path and first order reflection paths are updated at higher frequency. The audio rendering system queries the buffer and performs 3D audio for direct and first order paths and convolution for higher order paths. The cross-fading and interpolation components smooth the final audio output signal.

as shown in Figure 4.5 at different rates. Furthermore, each path can be represented as a virtual source with some attenuation, distance from the listener, and the incoming direction relative to the listener. The direction of a virtual source relative to the listener is simulated by introducing 3D sound cues in the final audio. Additionally, the source, listener, and scene objects can move dynamically. In such cases, the impulse response (IR) computed during the sound propagation step can vary significantly from one frame to another. Thus, our approach mitigates the occurrence of artifacts by various means described in Section 4.4.2. Our system also uses the previously described reverberation data to construct the appropriate sound filters, as described in Section 4.4.4.

4.4.1 Integration with sound propagation

The paths computed by the sound propagation algorithm in Section 4.2 are updated at different rates for different orders of reflection (see Table 4.2). These paths are queried by the audio rendering system in a thread safe manner. To achieve a high quality final audio signal, the audio rendering system needs to query at the sampling rate of the input audio signal (44.1 KHz). Our output audio is computed in fixed sized blocks of data called audio frames. We have found frames containing 10ms worth of audio samples suitable gives a good tradeoff between latency and compute cost. Various user studies support that a lower update rate [Savioja et al., 1999] can be used without any perceptual difference (about 100ms). It should be noted that the direct sound component and the early reflection components are very fast to compute. Thus, we update the direct contribution and first order reflections at a higher rate than the other components. For the direct and first order reflection

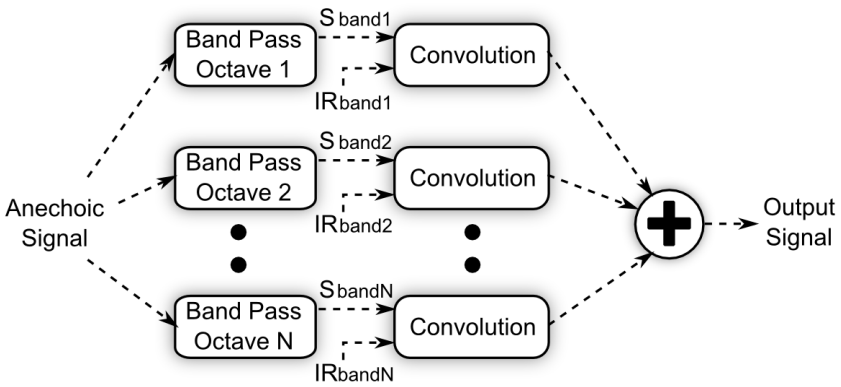


Figure 4.6: **IR convolution:** The input audio signal S is band passed into N octave bands which are convolved with the IR of the corresponding band.

paths, we also introduce 3D sound cues in the final audio signal. To produce the final audio we band pass the input signal into eight octave bands. For each octave band we compute an impulse response, which is convolved with the band pass input audio to compute final audio as shown in Figure 4.6. The details on computing an impulse response using the paths from the sound propagation engine are below.

Specular and Diffraction IR: The specular reflections and diffraction are formulated as a function of the sound pressure, as described in the previous sections. Thus, any path reaching from a source to the listener has a delay computed as d/C where d is the distance traveled, and C is the speed of sound. Each impulse is attenuated based on frequency dependent wall absorption coefficients and the distance traveled. For all the paths reaching from a source to the listener, a value with attenuation A_{path} is inserted at time index d/C in the impulse response. One such impulse response is computed for all different octave bands for a source-listener pair.

Diffuse IR: The diffuse reflections are formulated as a function of the energy of the sound waves. Using the paths collected at the listener, an energy IR is constructed for all the reflection paths reaching the listener. This energy IR is converted into pressure IR for audio rendering. We take the square root of energy response to create a pressure IR for each frequency band. This IR is combined with specular and diffraction IRs to produce the final IR used in the audio rendering.

4.4.2 Issues with dynamic scenes

Our sound propagation system is general and can handle moving sources, moving listener, and dynamic geometric primitives. This introduces a unique set of challenges for our real-time audio rendering system. Due to the motion of the sources, listener, and scene objects, the computed paths from successive simulation frames could change dramatically, so producing artifact-free audio rendering can be challenging. Therefore, we impose physical restrictions on the motion of sources, listener, and the geometric primitives to produce artifact-free audio rendering. We restrict source and receiver motion to one meter per second and object motion to 10 meters per second. If these limits are exceeded, there may be sharp transitions in the output signal, resulting in audible clicks. Some scenes may require different limits based on achievable update rates.

To further mitigate the effects of the changing IRs, we convolve each audio frame with the current and the previous IRs and crossfade them to produce the final audio signal. The window of cross-fading can be adjusted to minimize artifacts due to motion. Other more sophisticated approaches like predicting the positions and velocities of source or the listener can also be used [Tsingos, 2001, Wenzel et al., 2000].

4.4.3 3D sound rendering

In a typical sound simulation, many sound waves reach the listener from different directions. These waves diffract around the listener's head and provide cues regarding the direction of the incoming wave. This diffraction effect can be encoded in a Head-Related Impulse Response (HRIR) [Algazi et al., 2001]. Thus, to produce a realistic 3D sound rendering effect, each incoming path to the listener can be convolved with an HRIR. However, for large numbers of contributions this computation can quickly become expensive and it may not be possible to perform audio rendering in real time. Thus, only direct and first order reflections are convolved with a normalized HRIR [Algazi et al., 2001]. Some recent approaches have been proposed to handle audio rendering of large numbers of sound sources [Tsingos et al., 2003, Wand and Straßer, 2004]. These approaches can also be integrated with our system.

4.4.4 Adding late reverberation

XAudio2 supports the use of user-defined filters and other audio processing components through the XAPO interface. One of the built-in filters is an artificial reverberation filter, which can add late decay effects to a sound signal. This filter can be attached to the XAudio2 pipeline (one filter per band) to add late reverberation in a simple manner.

The reverberation filter has several configurable parameters, one of which is the RT_{60} for the room. In Section 4.3, we described a method for estimating this value. The reverberation filter is then updated with the estimate. This approach provides a simple, efficient way of complementing the computed IRs with late reverberation effects.

4.5 Performance

Our system makes use of several levels of parallel algorithms to accelerate the computation. Ray tracing is known to be a highly parallelizable algorithm and our system uses threads to take advantage of multi-core computers. Also, frustum tracing uses vector instructions to perform operations on a frustum's corner rays in parallel. Using these optimizations, our system achieves interactive performance on common multi-core PCs.

In this section, we detail the performance of RESound. We highlight each subsystem's performance on a varying set of scenes. The details of the scenes and system performance are presented in Table 4.2, and the scenes are visually shown in Figure 4.7. In all benchmarks, we run RESound using a machine with multiple Intel Xeon 5355s at 2.66Ghz; the number of threads per component is described in each section. Thread scaling is described in Table 4.1.

The time to preprocess the scenes was about 4 seconds for the largest (Conference). We note that scenes with moving objects require varying time costs. We found that our test scenes could undergo rigid transforms and typical movements like moving doors and cars with time cost on the order of 20ms per frame. Very large non-linear transforms could require up to the full preprocessing time; refer to [Lauterbach et al., 2006] for details.

(a) (b) (c) (d)

Figure 4.7: **Test scenes used:** (a) Room, (b) Conference, (c) Sibenik, and (d) Sponza.

Method	1 thread	2 threads	4 threads
Frustum tracing	1.0X	2.0X	3.2X
Ray tracing	1.0X	1.9X	3.2X

Table 4.1: **Performance scaling:** We show the performance scaling of our frustum tracing and ray tracing implementations.

		Specular + diffraction (3 orders)			Specular + diffraction (1st order)			Diffuse (3 orders)	
Scene	Triangles	Time	Frusta	Paths	Time	Frusta	Paths	Time	Paths
Room	6k	359ms	278k	4	77ms	7k	3	274ms	228
Conference	282k	1137ms	320k	7	157ms	5k	2	323ms	318
Sibenik	76k	2810ms	900k	14	460ms	10k	5	437ms	26
Sponza	66k	1304ms	598k	8	260ms	10k	3	516ms	120

Table 4.2: **Performance:** Test scene details and the performance of the RESound components.

Specular and Diffraction: We generate two separate IRs using frustum tracing. One IR includes only the first order specular and diffraction contributions. Since these paths are fast to compute, we devote one thread to this task. The other IR we generate includes the contributions for 3 orders of reflection and 2 orders of diffraction. This is done using 7 threads. The first order paths are removed from the higher order results since they are captured by the first order simulation. The performance details for both simulations cycles are described in Table 4.2.

Diffuse tracing: Our diffuse tracer stochastically samples the scene space during propagation. As such, the rays are largely incoherent and it is difficult to use ray packets. Nonetheless, even when tracing individual rays, RESound can render at interactive rates as shown in the performance table. The timings are for 200k rays with 3 reflections using 7 threads.

Late reverberation: We measured the time taken by our implementation to perform the least-squares fitting while estimating late reverberation. The execution time was measured by averaging

# Impulses	Compute time (ms)
10	0.026
50	0.111
100	0.425
1000	37.805
5000	1161.449

Table 4.3: **Reverberation timings:** The time cost to estimate the reverberation decay is quite small compared to propagation times.

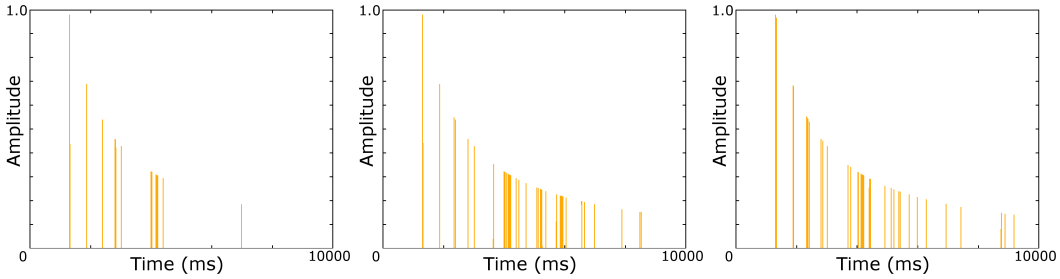


Figure 4.8: **Specular paths:** With a subdivision (a) level of 2, frustum tracing finds 13 paths. A subdivision (b) level of 5 finds 40 paths. The (c) image-source solution has 44 paths.

over 10 frames. During testing, we vary the density of the impulse response. The reverberation calculation is not threaded due to its minimal time cost. The results are summarized in Table 4.3.

4.6 Quality

The algorithms used in RESound are based on the physical properties of high frequency acoustic waves. We discuss the output quality of each component in the RESound system and compare against the more accurate simulations. We also note the benefits that RESound offers over simpler audio rendering systems. The underlying limitations of the methods used are also discussed.

4.6.1 Quality

Since adaptive frustum tracing approximates the image source reflection model, its accuracy has been compared to image-source methods [Chandak et al., 2008]. It was found that as the subdivision level increases, the number of contributions found by the frustum simulation approach the number found by the image-method. Moreover, the attenuation of the resulting impulse response from frustum tracing is similar to that found by image-source (Figure 4.8).

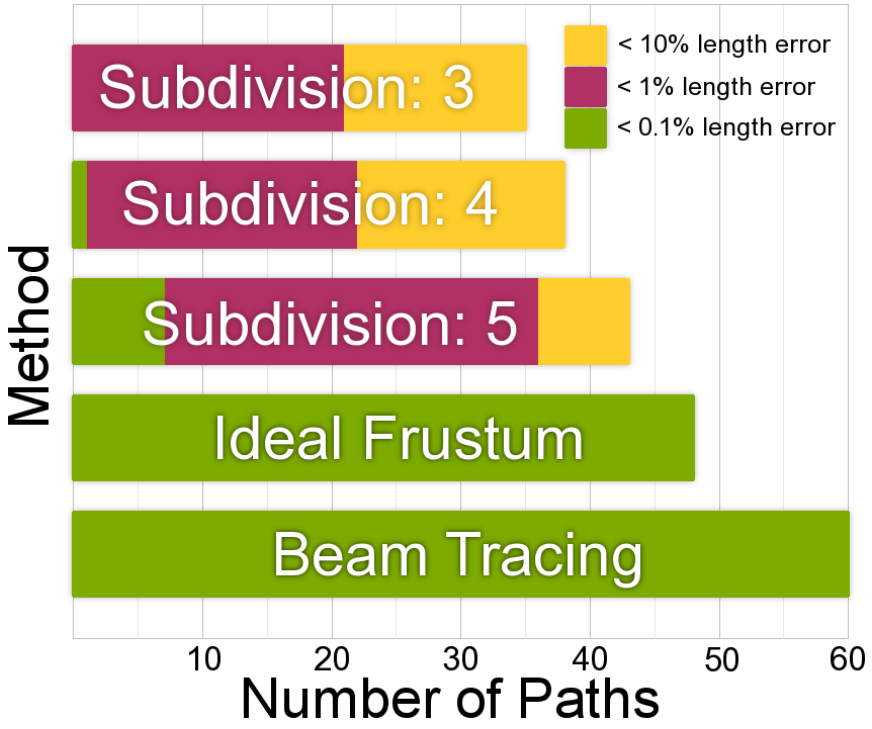


Figure 4.9: **Diffraction paths:** Increasing the frustum subdivision improves the diffraction accuracy.

Similarly, the validity of diffraction using frustum tracing has also been compared to an accurate beam tracing system with diffraction [Taylor et al., 2009b]. Due to limitations of frustum engines, it was found that certain types of diffraction paths could not be enumerated. When a diffraction frustum was created, one plane of the frustum was parallel to the wall on the shadow region of the diffracting edge. Since this side of the frustum was parallel to the wall, it could not diffract off any edges that lied in the plane. However, as the frustum subdivision level was increased, the number of diffraction paths found approached an ideal solution (Figure 4.9) and the paths accurately matched the reference solution.

The diffuse IR in RESound is generated by stochastic ray tracing. The sampling and attenuation model RESound uses has previously been shown to be statistically valid with sufficient sampling. Detailed analysis and validation has been presented by Embrechts [Embrechts, 2000].

We compare our reverberation decay times to statistically estimated times in two simple scenes. Similar scenes are described in other work [Hodgson, 1990, Kapralos et al., 2004]. The results are presented in Table 4.4.

Room size (m)	Absorption	Predicted	RESound
4x4x4	0.1	1030 ms	1170 ms
27.5x27.5x27.5	0.0	8890 ms	7930 ms

Table 4.4: **Reverberation decay times** Statistical predicted times compared to RESound measured times for two models.

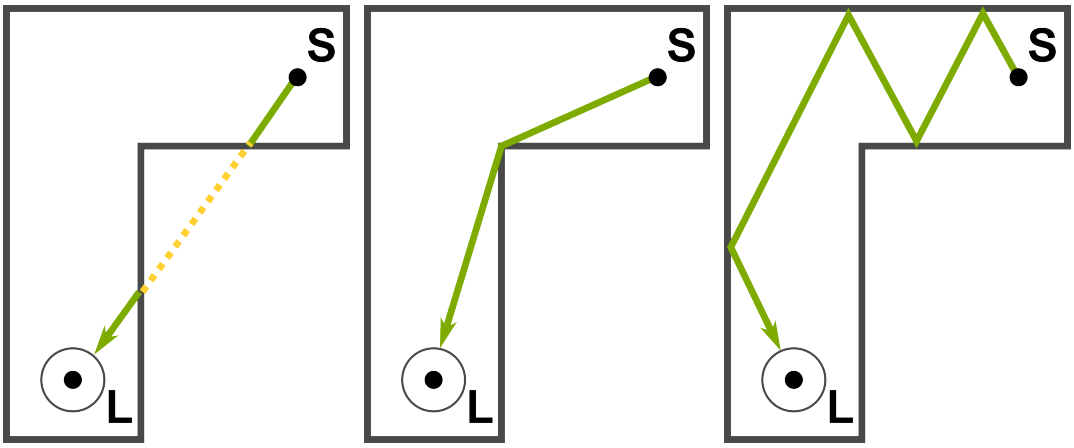
Most other interactive acoustic simulations cannot render diffuse reflections [Chandak et al., 2008] and some have restrictions on source movement [Tsingos et al., 2001]. Others are restricted to simple scenes of a few thousand triangles [Savioja et al., 2002]. Since our system has a unified ray model based on advanced ray tracing acceleration structures, it can handle large and complex scenes while providing diffraction, specular reflections, and diffuse reflections.

4.6.2 Benefits

Interactive audio simulations used in current applications are often very simple and use pre-computed reverberation effects and arbitrary attenuations. In RESound, the delays and attenuations for both reflection and diffraction are based on physical approximations. This allows RESound to generate acoustic responses that are expected given scene materials and layout.

In addition to calculating physically based attenuations and delays, RESound also provides accurate acoustic spatialization. Simple binaural rendering often only uses the direct path, which may not be valid. With RESound, the reflection and diffraction path directions are included. Consider a situation when the sound source is hidden from the listener's view (Figure 4.10). In this case, without reflection and diffraction, the directional component of the sound field appears to pass through the occluder. However, propagation paths generated by RESound arrive at the listener with a physically accurate directional component.

Simple binaural rendering is very cheap to compute, but computes responses that do not included acoustic effects and spatial queues that can be physically incorrect. RESound has a greater compute cost, but for applications where plausible acoustics are needed at interactive rates, it represents the first system to handle several major acoustic effects in general scenes.



(a) (b) (c)

Figure 4.10: **Path direction:** From source *S* to listener *L*: (a) The simple direct path is physically impossible, but (b) diffraction and (c) reflection paths direct the listener as physically expected.

CHAPTER 5: GUIDED MULTIVIEW TRACING

Modern programable GPUs have become very useful in solving many computationally intensive problems. Unfortunately, the method introduced in the previous chapter does not map well to modern GPUs due to the locally adaptive frustum tracing process. In this chapter, we describe algorithms to efficiently simulation GA on a GPU. Since our method uses ray samples, we introduce a method to dynamically adjust sampling parameters to minimize time cost while preserving accuracy. In addition, we present an algorithm for computing ray diffraction.

5.1 Guided propagation

In this section we describe a cost function for specular GA propagation. We then describe our approach to reducing the number of tests conducted during propagation.

Some GA methods, such as beam tracing, compute a very accurate, minimal visibility tree. Since the reflection data in the tree is very detailed, the complete set of valid paths can be created very quickly [Funkhouser et al., 1998]. However, generating such an accurate tree is costly. Other methods, such as conservative frustum culling [Chandak et al., 2009], compute accurate trees that may be overly conservative. The visibility tree can be generated faster, but the path creation time may increase, since some of the paths will be occluded and must be discarded. This concept is similar for other GA based methods, leading to a simple cost function for specular propagation:

$$T = \sum_{i=0}^{n} S_i + \sum_{i=0}^{n} \sum_{j=0}^{m} R_{i,j}$$

Where T is the total cost of specular propagation, S_i is the cost of generating the visibility tree for source i, and $R_{i,j}$ is cost of path creation for source i to receiver j. Each source requires separate visibility tree and path calculations. It should be noted that propagation paths are reciprocal when ignoring source and receiver directivity, so the endpoint types can be swapped if it minimizes propagation cost.

5.1.1 Ray traced propagation cost

Sample based visibility methods like ray tracing are not guaranteed to generate an accurate visibility tree. This is because some triangles that are visible to the source may be missed by the samples and incorrectly excluded from the visibility tree. However, ray tracing is still used in sound propagation because of its high performance and ease of implementation. Since our system uses ray based propagation, we focus our discussion on the cost of ray traced propagation.

The general form of ray traced propagation is to trace a distribution of visibility rays from the source into the scene, reflecting the rays to the desired order of recursion. A sphere of some radius is set at each receiver location and some of the visibility rays may hit this detector sphere. The visibility rays that strike the sphere represent likely propagation paths and should be validated to be occlusion free.

For ray traced sound propagation systems with visibility and validation, S and R can then be expanded as:

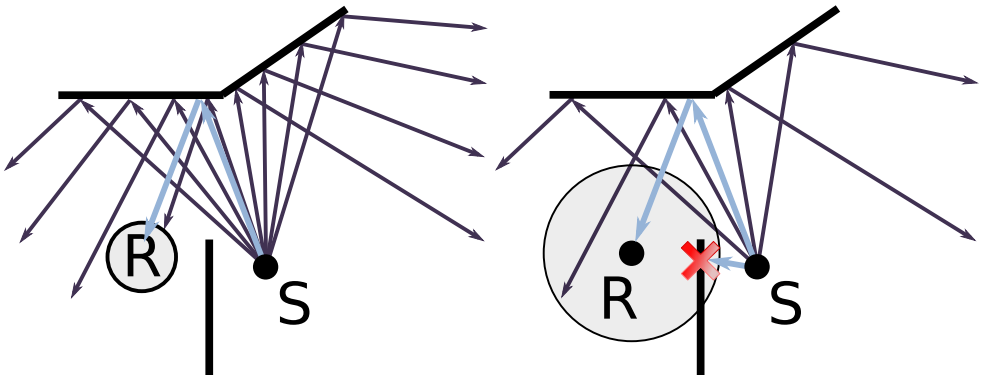
$$T = \sum_{i=0}^{n} N_i V_i + \sum_{i=0}^{n} \sum_{j=0}^{m} N_i P_{i,j} L_{i,j}$$

where T is the total time cost of the simulation, N_i is the number of visibility rays cast for source i, V_i is the cost of propagating a visibility ray for source i, $P_{i,j}$ is the probability that a visibility ray from source i strikes receiver j and must be validated, and $L_{i,j}$ is the cost of validating this propagation path.

The cost of visibility and path creation are scene and position dependent and can be minimized by efficient ray tracing techniques or reducing the number of tests in the propagation step. We discuss efficient tracing techniques in Section 5.2. To minimize the number of tests, we decouple visibility and validation and use a guidance algorithm to minimize the cost of each independently.

5.1.2 Guidance algorithm

In most simulations, it is desirable to reduce the time cost T while maintaining simulation accuracy. This can be accomplished by controlling the terms in the cost equation. Given a minimum number of propagation paths to be discovered, there exists some efficient values for N and the receiver radius (and thus P) that find the specified number of paths while minimizing T.



(a) (b)

Figure 5.1: **Sample-based visibility:** Visibility rays are traced from source S into the scene. Paths that strike receiver R are then validated. (a) A small receiver requires dense visibility sampling to find the propagation path. (b) Using a larger receiver allows sparse sampling resulting in fewer visibility tests, however more validation tests are need to remove invalid path sequences.

Conducting a large number of visibility tests N with a high probability P of discovering many paths to validate (by way of a large receiver size) will likely find the wanted paths, but with high time cost. If N is reduced, there will be fewer rays that encode each unique path sequence. Accuracy can be maintained if the probability of these rays being detected is increased by increasing receiver size. Similarly, with a high N, the receiver size can likely be decreased (reducing P) while still detecting the necessary path sequences (see Figure 6.4).

The base cost values of visibility rays V, and of path validation L, vary vastly based on the underlying algorithms and implementations. Also, when the source or receiver move, or objects in the scene move, V and L can change, altering the total cost function. Indeed, since the optimal values may change throughout the course of the simulation, it is difficult to find the best values to reduce the time cost.

Instead of attempting to find the optimal values, our guidance algorithm seeks to independently minimize the number of visibility and validation tests used in propagation. It works by adjusting the number of visibility rays cast and the size of the detection sphere. These two factors correspond to N and P, respectively. Figure 5.2 shows an example cost function in terms of the minimal number of tests needed to find a specific percentage of the total paths in a scene.

Our algorithm monitors the count of unique contribution paths found during a single simulation frame. The goal on subsequent frames is to find an equal or greater number of paths to this maximum recorded path count, while using a minimal number of visibility and validation tests. The algorithm

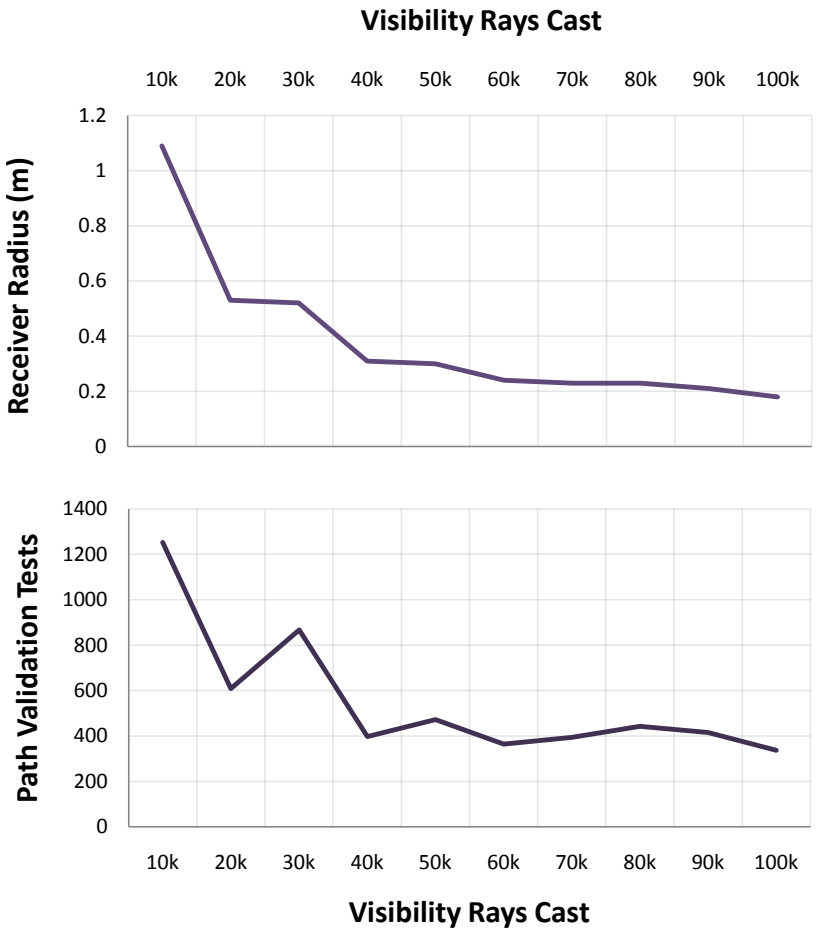


Figure 5.2: **Propagation test count:** With a goal of finding 90% of the total paths in the scene, an increasing number of visibility rays are traced and the minimum required size of the receiver sphere changes accordingly. With sparse visibility sampling, a large sphere is required, resulting in many validation tests. With dense sampling, the sphere size can be reduced. For specific cost values for visibility and path validation tests, some minimal total cost exists.

achieves this by reducing the number of rays traced and the size of the detection sphere. If at any time the path count decreases (i.e. a path is lost), the algorithm responds by increasing the number of rays and receiver size until the path is recovered. If the path cannot be recovered after aggressive adjustment, the lower path count is selected as the maximum known path count. If at any time the current path count exceeds the recorded maximum count, the maximum count is updated to the new higher count. This allows our method to respond conservatively to scene changes.

On startup, the algorithm begins by tracing a user specified number of rays and with a user specified receiver sphere size. We use 50k rays and a sphere radius of $\frac{1}{4}$ the length of the maximal scene axis in our tests. From this point, the number of rays and sphere size are reduced to find local

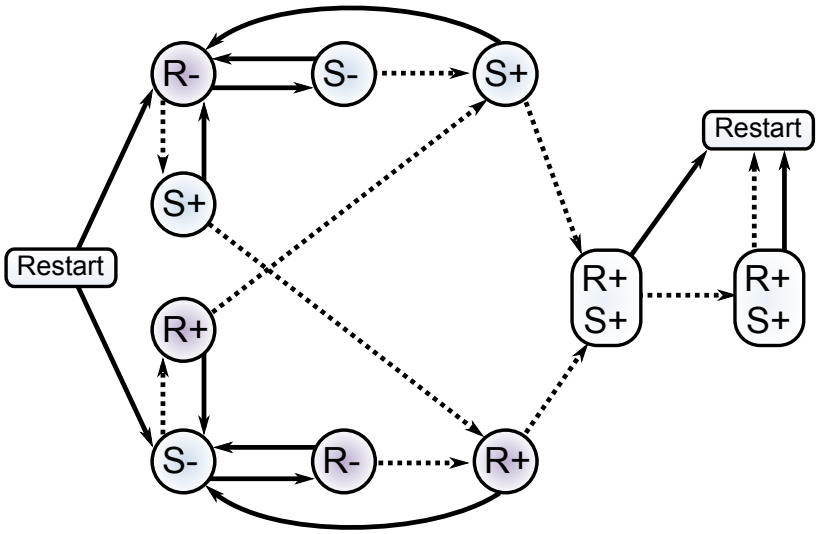


Figure 5.3: **Guiding state machine:** This state machine tracks the number of unique contribution paths found. Solid lines are followed if the current path count matches the recorded maximum count, dashed lines are followed if the path count is less than the recorded maximum. States marked $R+$ and $S+$ increase the ray count and sphere size, while states marked $R-$ and $S-$ decrease the ray count and sphere size, respectively. At the *Restart* state, the maximum paths count is set to the current count. The $(R+, S+)$ states attempt to recover lost paths before recording a new count. The main top and bottom arms focus on reducing rays and receiver size respectively.

minima of the total cost function without decreasing accuracy. A small initial number of visibility rays can lead to sampling errors that are further discussed in Section 6.4.

Our guiding algorithm is easily represented as a state machine. Figure 5.3 shows the details of the state machine. After each simulation cycle a new state is found and the propagation parameters are adjusted. This process continuously adjusts the number of rays traced and the size of the receiver spheres. Each receiver sphere is adjusted independently; if the state machine enters a state where sphere size is increased, but no paths have been missed to a certain receiver, that specific receiver sphere is not increased. The accuracy and performance of the algorithm is discussed in Section 6.4.

5.2 Multi-view GPU ray tracing

We use a high performance GPU ray tracer to conduct the visibility and validation tests needed during sound propagation. To further improve performance, we attempt to process each specular view in parallel independently using a multi-view tracing approach. We describe our basic GPU ray tracer, the multi-view tracing process, and our diffraction and validation approaches.

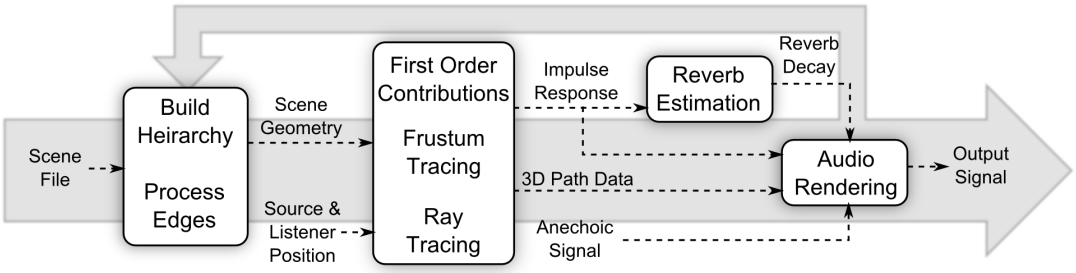


Figure 5.4: **Implementation overview:** All scene processing and propagation takes place on the GPU: hierarchy construction, visibility computations, specular and edge diffraction. The sound paths computed using GPU processing are returned to the host for guidance analysis and audio processing. The guidance results are used to direct the next propagation cycle.

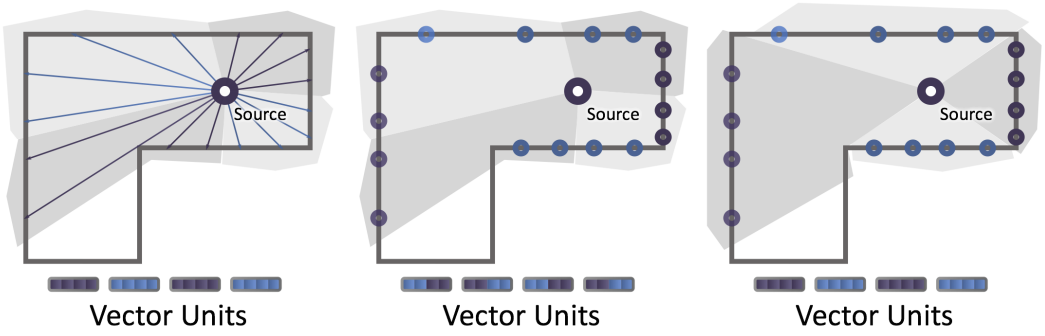
5.2.1 GPU propagation

We divide the processing work between the host and GPU device. The host handles all audio processing, while the GPU device computes the propagation results. Figure 5.4 shows the overall details. Our propagation algorithm traces visibility rays through the scene, intersects them with a receiver sphere, and validates the possible propagation paths to be occlusion free.

For general ray tracing, previous approaches have investigated efficient methods to implement ray tracing on massively parallel architectures such as GPUs, which have a high number of cores as well as wide vector units on each core. Current methods for GPU-based ray tracing mainly differ in the choice of acceleration structure such as kd-trees [Zhou et al., 2008] or BVHs and the parallelization of ray traversal step on each GPU core. For example, rays can be traced as a packet similar to CPU SIMD ray tracing [Popov et al., 2007] and some recent approaches can evaluate them independently [Aila and Laine, 2009], as more memory is available for local computation on current GPUs.

We build on the bounding volume hierarchy (BVH) ray tracing ideas in [Lauterbach et al., 2009] and implement our multi-view ray casting system in CUDA. This allows us to render scenes with dynamic geometry, as the BVH can be refit or rebuilt as needed. While NVIDIA provides a ray tracing system [Parker et al., 2010] for use on CUDA hardware, we use our own fast ray tracer due to its flexibility.

Rays are bundled into packets that are executed on each core while scheduling each ray on a lane in the vector unit. The rays are then traversed through the BVH and intersected against the triangles. For primary visibility samples, we use a simple ray tracing kernel that exploits the common



(a) (b) (c)

Figure 5.5: **Multiview tracing:** (a) From the source, rays are grouped into packets that can be efficiently processed on the vector units. (b) However, a single packet may hit multiple surfaces, resulting in reflection packets that are inefficient. (c) We reorder packets so that each reflection view can be traced efficiently.

ray sources for efficiency. Reflections are handled by a secondary kernel which loads the previous hit data and traces a reflection ray. To decouple the number of visibility samples from the number of threads allocated for processing, we iteratively process visibility samples in small thread blocks until all samples have been traced. As rays exit the scene, they are removed from the work queue and no longer processed. The algorithm ends when no more active samples exist or the maximum recursion depth is reached in terms of reflections. At any point during tracing, if the ray coherence is reduced past a user specified threshold, multi-view tracing is employed.

Once the visibility computations have been performed up to a specified order of reflection, the visibility data is tested against the receiver spheres. Each ray is tested against each receiver sphere for intersection and is marked if it hits the sphere and needs to be included in the path validation tests.

As a final step, once the receiver intersect tests are complete, we compute the valid contribution paths to the receiver. For each valid path, image source and triangle data is retrieved. A test checks if the line connecting the source image point to the receiver passes through the associated triangle. This test immediately discards most invalid paths. Then, for each receiver, a ray is constructed from the receiver towards each image point and traced through the scene. From the resulting hit point, a new ray is traced to the parent image, continuing back to the initial source point. If the entire path is unoccluded, there is a contribution.

5.2.2 Multi-view tracing

The underlying formulation of the image source method is such that each reflection path can be evaluated independently. When using ray tracing visibility with the image source method, all visibility and validation tests are also independent. As such it is possible to evaluate queries in parallel. For example, if there are multiple sound image sources, we may perform visibility computations from each of them in parallel. Our multi-view algorithm exploits this: in order to achieve high performance, we process all independent visibility and validation tests simultaneously.

When considering specular reflection rays, it is helpful to view rays as visibility queries that accelerate the image source process. From the source point, the ray visibility query returns the set of triangles visible to the source (subject to sampling error). From this set of visible triangles, image sources can be created by reflecting the source point over each triangle face. New samples can then be generated on each triangle face, forming a reflection visibility query with the image point at the ray origin. This process repeats to the recursion limit.

In our case, it is natural to bundle all the reflected visibility samples from one origin together in ray packets. Since the main factor determining performance in our packet based ray tracer is ray coherence, such bundling allows efficient use of memory bandwidth and SIMD vector units. As described in the previous section, primary visibility rays are easy to group into coherent packets. However, as the rays are reflected, it is likely that the rays in the packet will hit different triangles, and thus be reflected in different directions with different ray origins. As a result, the packets are less coherent and may require multiple queries to the BVH, thus wasting computational resources.

On the GPU, each thread block is treated as if it is running on independent hardware from all other blocks. Our ray packets are formed with a ray for each thread in the thread block. When all the rays in a packet share a common origin, the packet represents a single ray traced view that can be traced very efficiently. The goal of our multi-view system is to achieve this as often as possible. Our system detects when packets become incoherent and restructures all rays into more efficient packets.

Given the results of visibility ray casting as a list of triangle IDs, we perform a data-parallel bitonic sort using the vector unit. Using the list sorted by ID, it is trivial to find out for each hit or intersection point whether the ID is a duplicate (i.e. hit by multiple ray samples) by comparing against its successor. If all IDs are duplicates, all rays in the packet hit the same triangle and reflection

Figure 5.6: **Multiview performance**: Multi-view ray tracing out performs standard ray tracing for scenes (80k triangle scene shown) with many specular views. The multi-view timings include the time cost of all necessary sorting and reordering.

rays are likely to share the same origin (at the image source) and direction. Such a packet is likely to be coherent and efficient. However, if the sort reveals multiple triangle IDs, the reflection rays will likely not share a common origin, and are probably incoherent.

After each trace recursion, the coherency test is applied to each ray packet. The number of packets that are likely to be incoherent is then recorded. The percent of packets that are incoherent is compared to a user specified limit (we use 80% in all our tests). If the threshold is exceeded, the ray packets are reordered into more efficient views (see Figure 5.5).

This is done by performing a parallel radix sort on triangle ID across all hit data. The hit and ray data is reordered according to the sort results. Since each ray's index is no longer indicative of its parent ray, an index table is also created to find parent hit and ray data. As a result of this view reordering process, our multi-view tracing algorithm performs high specular reflection orders faster than standard ray tracing (see Figure 5.6).

5.2.3 Diffraction

We use an approximate edge diffraction algorithm based on the UTD formulation.

Similar to other approaches, only certain edge types [Taylor et al., 2009a] are considered to cause diffraction. We select two types of edges as diffracting edges: disconnected edges that are only shared by one triangle and edges shared by triangles with normals that differ by $> \frac{1}{8}\pi$ radians. For each diffracting edge, we store the indices of the triangles that share the edge. This edge data

is precomputed before simulation. As part of the ray casting algorithm, we compute barycentric coordinates of each intersection point on the triangle face [Arenberg, 1988]. These coordinates represent how far an intersection point in the interior of a triangle is from the triangle vertices; in the intersection routine, the barycentric coordinates are used to detect if the ray hit point lies within the triangle boundaries. We reuse the barycentric coordinates when detecting if diffraction rays need to be traced. If a hit point's barycentric coordinates show that the hit point is within 10% of a diffracting edge, as measured along the triangle surface, we consider the ray close enough to the edge to continue diffraction propagation. Using the barycentric coordinates of the hit point, we project the hit point on to the diffracting edge. This point on the edge becomes the origin from which diffraction propagation takes place.

Given the barycentric coordinates of the hit point: λ_1, λ_2, λ_3 and a triangle with edges e_1, e_2, e_3, a diffraction origin o can be created on e_3 from a hit point that hit near e_3 as follows. Figure 5.7 shows the arrangement visually.

$$s = \lambda_1 + \lambda_2$$
$$\alpha = \frac{\lambda_2}{s}$$
$$d = \alpha e_3$$

Rays are then traced from the diffracting origin according to the UTD: the outgoing diffracting rays have the same angle relative to the diffracting edge as the incident ray. However, we only trace diffraction rays in the shadow region, not the full diffraction cone, as described by the UTD. The diffracting edge is shared by the triangle that was hit by the ray and an occluded triangle (possibly the backface of the hit triangle). These two triangle faces form a diffracting wedge as describe by the UTD. In order to create diffraction rays for the shadow region, we create a vector based at the origin that was previously found and with the incident ray's direction. This vector is rotated towards the occluded triangle face, sweeping out only the part of the diffraction cone that is in the shadow region. Since we only conduct first order diffraction in our simulation, we only trace rays that pass near the receiver spheres. This greatly reduces the required computation. It is also possible to trace higher order diffraction rays by sampling the swept surface of the cone, discretizing the region. See Figure 5.8 for details.

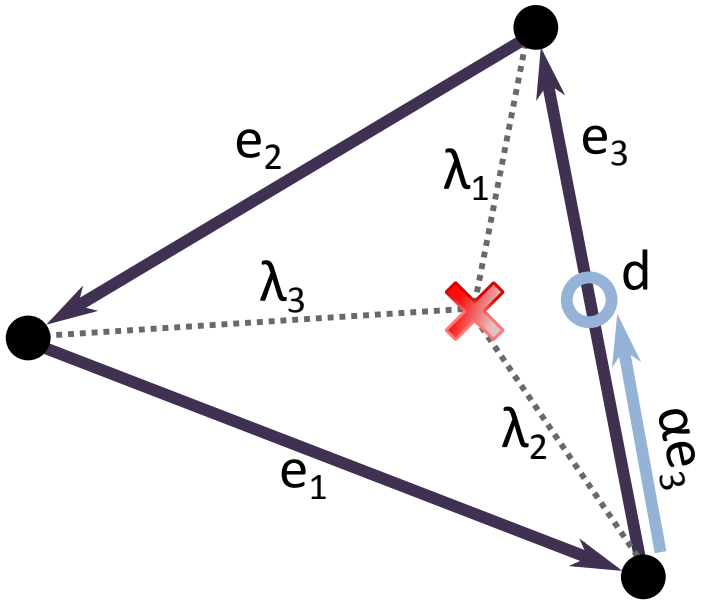
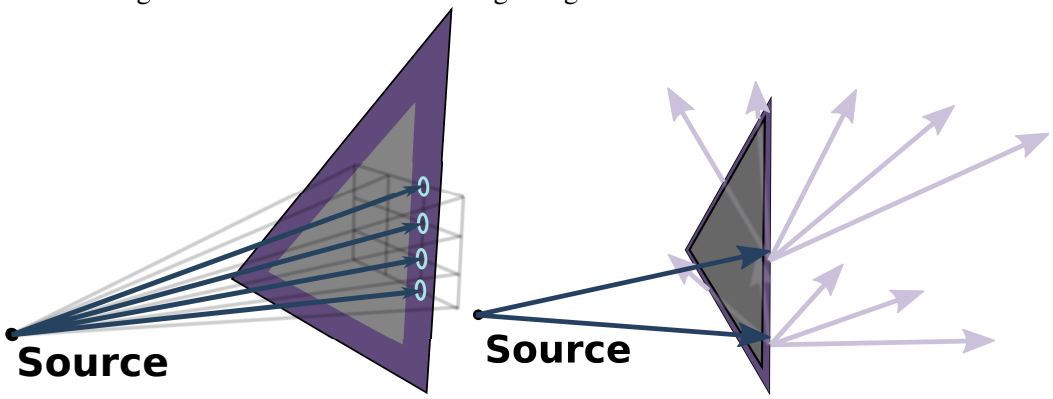


Figure 5.7: **Barycentric diffraction hit points:** Using the barycentric coordinates of a ray hitpoint, a diffraction origin d can be found on the triangle edge.



(a) (b)

Figure 5.8: **Edge diffraction**: (a) Rays near the edge are detected for resampling. (b) Diffraction samples are cast through the shadow region, bounded by the adjacent triangle.

Even low order diffraction can branch rapidly [Calamia et al., 2008]. Since our ray tracer is implemented on a GPU and such branching would require expensive dynamic allocation and kernel management, our system only conducts one order of diffraction. As each ray intersects a triangle, the hit point is checked for nearness to a diffracting edge. If the hit point is within 10% of the edge in triangle space, a diffraction ray is traced to the receiver point. Only diffracting rays where the incident and outgoing rays differ by $< 10^{\circ}$ are retained. These paths are sent back to the host after tracing and the shortest (i.e., least error) paths are kept for attenuation and output.

If a diffraction path is found to reach the receiver, the acoustic signal must be attenuated based on the diffraction interaction. Our system uses the UTD attenuation function [Kouyoumjian and Pathak, 1974], adjusted for smoothness at the shadow region. This function gives the band attenuation based on the properties of the diffracting edge and the ray geometry. We apply the function for each of the frequency bands that the user has selected, resulting in appropriate attenuation of high frequencies when the path is diffracted.

5.2.4 Path creation

During the ray tracing visibility step, all visibility information is recorded in GPU memory. This data is used in the path creation stage to determine which paths are occlusion free between the source and receiver.

When using ray tracing to determine visibility, we recognize that there will be many visibility rays that record duplicate sequences of triangle IDs. In the path creation stage, it would be most efficient to perform a validation test for unique sequences only, not for each individual visibility ray. However, it can be difficult to remove duplicate visibility sequences.

Our initial attempts at efficient path validation removed duplicate paths by creating visibility hashes for each sequence. Visibility rays were sorted by sequence hash, then the unique visibility sequences were found, and a single validation ray was cast for each unique sequence. This resulted in very low cost visibility tests for path creation: a single ray for each sequence. However, the cost of the required sorting and scans to arrange the sequences was very expensive.

Our final path validation method is less elegant, but much simpler to perform. For each visibility ray, a validation test is performed. This results in many duplicate validation tests, but these can be efficiently performed on parallel hardware. Each path is validated to be occlusion free in reverse, from receiver to source. Path data is returned to the host, where duplicate removal takes place.

The visibility ray order is not changed before validation; this results in the *first* order of validation reflection rays being as coherent as the last order of visibility reflection rays. The directional coherence of the zeroth order of validation rays is not guaranteed, but all zeroth order validation rays share the receiver as a common origin.

Unlike specular paths, our diffraction validation is approximate. Rather than creating each optimally short UTD path, we select the most optimal path from the unmodified visibility samples.

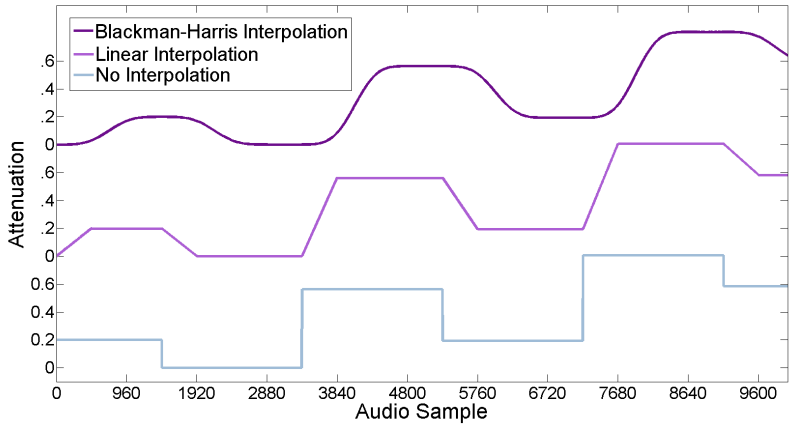


Figure 5.9: **Interpolation schemes:** Different attenuation schemes applied for attenuation interpolation. Discontinuity in attenuation between two audio frames interpolated with linear interpolation and Blackman-Harris interpolation. Delay interpolation is performed using a linear interpolation. Variable fractional delays due to linear delay interpolation are handled by applying low order Lagrange fractional delay filter on a supersampled input audio signal during the audio processing step.

On the host, the diffraction path are sorted by a visibility sequence hash. For each unique diffraction sequence, the shortest path is selected for audio output. This path has a possible error of $< 10^{\circ}$ in edge angle and $< 10\%$ error spatially on the triangle surface.

5.3 Audio processing

Sound propagation from a source to a receiver computes an impulse response (IR) based on the length of the propagation paths and attenuation along the path due to spreading, reflection, diffraction, and medium absorption. The IR is convolved with the anechoic source audio to compute the final audio. Such a convolution based audio processing works well for static scenes, i.e., static sound sources, receiver, and scene geometry. However, dynamic and interactive scenes introduce variations in the paths reaching from a source to a receiver between two consecutive simulation frames and can lead to artifacts in final audio due to the variations in the IR between frames. In this section, we present our method to process audio in dynamic scenes and to minimize artifacts in final audio output.

5.3.1 Dynamic scenes

In many interactive applications, the source and receiver movements could be quite large. This can lead to sudden changes in the propagation paths (i.e. delay and attenuation) from a source to a

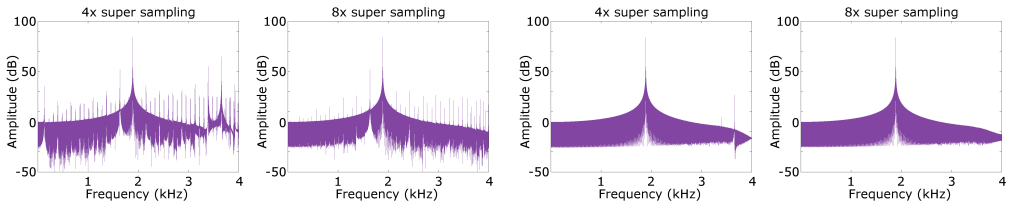
receiver. New paths may suddenly appear when a receiver comes out of a shadow region or due to the movement of scene geometry. Existing paths may disappear due to occlusion or sampling errors. To handle such scenarios, we track the paths and interpolate the changes in the paths to produce artifact-free audio output. Our approach combines parameter interpolation [Savioja et al., 1999, 2002, Wenzel et al., 2000] and windowing based schemes [Siltanen et al., 2009] to reduce the audio artifacts.

5.3.2 Parameter interpolation

In interactive applications, audio is typically processed in chunks of audio samples, called audio frames. For example, an audio signal sampled at 48 KHz could be processed at 100 audio frames per second, i.e. each audio frame has 480 samples. Between two such adjacent audio frames the propagation paths from a source to a receiver may vary, leading to a jump in attenuation and delay values per sample at the audio frame boundaries (see Figure 5.9). We track propagation paths and interpolate the delay and attenuation for a path per audio sample to reduce the artifacts due to changes in the path. To track propagation paths, each path is assigned a unique identifier.

We perform parameter interpolation of propagation paths for audio processing to achieve artifact-free final audio. It is equivalent to computing an IR per sample based on the parameter interpolation per path. Such an approach is physically intuitive and different interpolation schemes can be applied [Tsingos, 2001, Tsingos et al., 2004]. We treat each path from a source to a receiver as a parameter and represent it as an equivalent image source, i.e. delay, attenuation, and direction in 3D space relative to the receiver. Each image source is treated as an independent audio source during audio processing. Thus, changes in the paths are equivalent to changes in the corresponding image sources.

As an image source changes, its attenuation, delay, or direction relative to receiver may change. We perform attenuation interpolation between audio frames by applying a windowing function (Blackman-Harris) to smoothly interpolate attenuation at the audio frame boundary. This interpolation is performed on a per sample basis and leads to smooth transition across the audio frame boundary. To interpolate delay, we perform linear interpolation between audio frames. Linear delay interpolation augmented with supersampling and low order fractional delay lines work well to reduce the artifacts due to delay discontinuities between audio frames (see Section 5.3.3). Figure 5.9 shows interpolated attenuation per sample for an image source with attenuation discontinuities.



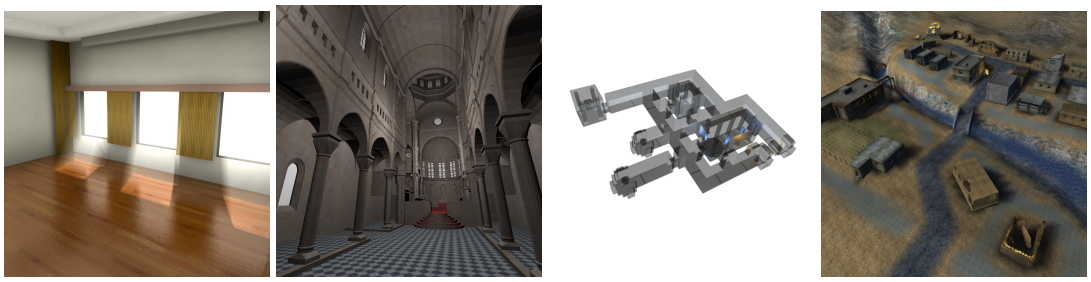
(a) Lagrange Filter Order = 0 (b) Lagrange Filter Order = 3

Figure 5.10: **Fractional delay:** Applying fractional delay filter and supersampling input signal to get accurate Doppler effect for a sound source (2 KHz sine wave) moving away from the receiver at 20 m/s. The sampling rate of the input audio is 8 KHz. The supersampling factors are 4x and 8x for left and right figures respectively. Zeroth order and third order Lagrange filters are applied.

5.3.3 Variable fractional delay

Fractional delay filters have been applied to speech coding, speech synthesis, sampling rate conversion, and other related areas [Välimäki, 1995]. In our application, we apply fractional delay filters to handle interpolated delays as sources (or image sources) and receivers move in a virtual environment. Rounding off the interpolated delays to nearest integer as sources and receivers move can lead to noticeable artifacts in the final audio (see Figure 5.10). Thus, we require efficient *variable* fractional delay filter that can provide fidelity and speed required in virtual environments. A good good survey of FIR and IIR filter design for fractional delay filter is provided in [Laakso et al., 1996].

We use a Lagrange interpolation filter due to explicit formulas to a construct fractional delay filter and flat-frequency response for low-frequencies. Combined with supersampled input audio signal, we can model fractional delay accurately. Variable delay can be easily modeled by using a different filter computed explicitly per audio sample. To compute an order N Lagrange filter, the traditional methods [Välimäki, 1995] require $\Theta(N^2)$ time and $\Theta(1)$ space. However, the same computation can be reduced to $\Theta(N)$ time and $\Theta(N)$ space complexity [Franck, 2008]. Many applications requiring variable fractional delay oversample the input with a high-order interpolator and use a low-order variable fractional delay interpolator [Wise and Bristow-Johnson, 1999] to avoid computing a high-order variable delay filter during run time. Wise and Bristow-Johnson [Wise and Bristow-Johnson, 1999] analyze the signal-to-noise-ratio (SNR) for various low-order polynomial interpolators in the presence of oversampled input. Thus, for a given SNR requirement, optimal supersampled input signal and low-order polynomial interpolator can be chosen to minimize computational and space complexity. Ideally, a highly oversampled input signal is required (see



(a) (b) (c) (d)

Figure 5.11: **Example scenes:** The scenes used to test the performance of our implementation: (a) Music hall model; (b) Sibenik cathedral; (c) Indoor scene; (d) desert scene. While the music hall scene is not often used for low order acoustic simulation, we selected it to show the animation sequence in Figure 5.13. Sibenik cathedral was selected as a very challenging visibility test case.

Figure 5.10) to achieve 60 dB or more SNR for a low-order Lagrange interpolation filter, but it might be possible to use low oversampling to minimize artifacts in final audio output [Savioja et al., 2002].

5.4 Analysis

In this section, we analyze the performance of our algorithm, highlight the error sources, and compare it with prior methods.

5.4.1 Performance

We have used our algorithms on several different scenarios and scenes. The complexity of these scenes is similar to those used in current games with tens of thousands of triangles for audio processing. We test the performance of the system on a multi-core PC with NVIDIA GTX 480 GPU and use a single CPU core (2.8 GHz Pentium) for audio processing. We used some common scenes to evaluate the performance of our system (Figure 5.11). Results for static source and receiver positions are shown in Table 5.1. We also show the cost of conducting higher order recursion in Table 5.2.

In addition to static scenes, we show results with dynamic movement in the Music hall scene using a 500 frame sequence. For this test, we use the coordinates defined in the round robin III dataset [1]. The source and receiver begin at coordinates S1 and R1, respectively. Over frames 100-200, the source moves linearly from S1 to S2. Over frames 300-400, the receiver moves linearly from R1 to R2. We compare our guidance method to other receiver size models. In each method, r is the

[1] http://www.ptb.de/en/org/1/16/163/roundrobin/roundrobin.htm

predicted receiver radius, N is the ray count, V is the scene volume, ℓ is the ray length, and d is the distance between source and receiver.

- **Lehnert model**: This model increases the receiver radius for rays that travel farther, adjusting the radius as rays spread out [Lenhert, 1993].
  $r = \ell\sqrt{\frac{2\pi}{N}}$
- **NORMAL model**: Originally a method for predicting the number of rays needed based on scene volume [Dance and Shield, 1994, Yang and Shield, 2000], this algorithm has been adapted as a receiver size model [Xiangyang et al., 2003].
  $r = \sqrt[3]{\frac{15V}{2\pi N}}$
- **Xiangyang model**: This model accounts for the minimal sphere receiver size needed for detection and adjusts the radius based on scene volume [Xiangyang et al., 2003].
  $r = \log_{10}(V)d\sqrt{\frac{4}{N}}$

It should be noted that these receiver models are intended for simulations that include high orders of diffuse reflections and are not necessarily optimal for low orders of specular reflections. Each of these receiver models have been implemented in a parallel efficient manner and integrated into our simulation. All simulations begin with 50,000 rays. Figure 5.13 shows detailed data for the animation sequence.

Model	#Tri	Bounces	#Paths	PT (ms)	AT (ms)
Desert	35k	3R+1D	15	53	3
Indoor scene	1.5k	3R+1D	27	62	5
Music Hall	0.2k	3R	62	23	7
Sibenik	80k	2R	11	90	3

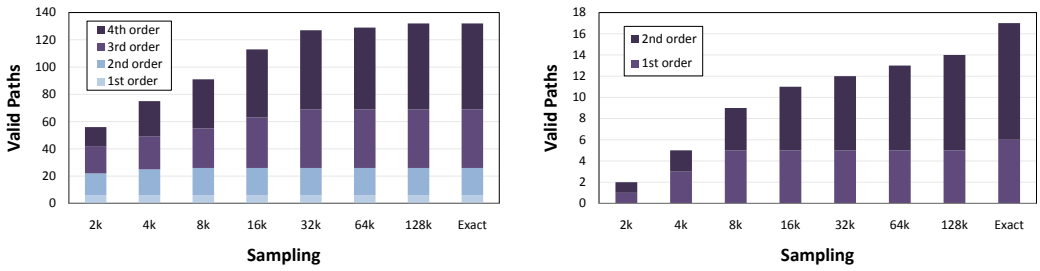
Table 5.1: **Performance in static scenes:** The top two represent simple indoor and outdoor scenes. The third one is a well known acoustic benchmark and the fourth one is the model of Sibenik Cathedral. The number of reflections (R) and edge diffraction (D) are given in the second column. The time spent in computing propagation paths (on GPU) is shown in the PT column and audio processing (on CPU) is shown in the AT column. The simulation begins with 50k visibility samples; we measure the performance after 50 frames.

5.4.2 Audio processing limitations

Our interpolation scheme presented in Section 5.3.2 produces smooth audio. It could be improved by interpolating image sources by predicting their new position based on their current positions and

Model	1R	2R	3R	4R
Desert	30	41	53	57
Sibenik	51	90	153	226

Table 5.2: **Performance per recursion:** Average performance (in ms) of our GPU-based path computation algorithm as a function of number of reflections performed. The Desert scene also includes edge diffraction. 50k visibility samples were used.



(a) Music Hall

(b) Sibenik Cathedral

Figure 5.12: **Recursion path count:** These figures show the number of paths found for varying visibility rays. The receiver size is fixed at 1 meter. As visibility ray count increases, low triangle count scenes like the Music hall (a) are quickly saturated. However, in complex scenes like Sibenik cathedral (b), higher visibility ray counts are required to explore the scene.

velocities [Tsingos, 2001]. Additionally, in cases where the number of image sources (or paths) is large, it is possible to apply clustering and perceptual acceleration [Moeck et al., 2007, Tsingos et al., 2004] for efficient audio processing. Currently, our audio processing step does not interpolate direction of an image source relative to the receiver but we encode it by computing delays and attenuation for left and right ears for 3D audio.

5.4.3 Comparisons

We compare our system with other general GA methods and specific rendering systems.

Ray tracing algorithms: Previous ray-shooting based propagation algorithms [Krokstad et al., 1968, Vorländer, 1989] trace each ray or ray packets independently to compute paths from the sources to the receiver. These methods model the receiver as an object of some size to determine when a discrete ray is close enough to the receiver to be considered a valid contribution path. This can lead to missed contributions, duplicate contributions (see Figure 6.4), or statistical errors [Lenhert, 1993]. Since we can adjust the visibility sampling and detection sphere size, our method can achieve better performance than prior ray tracing methods.

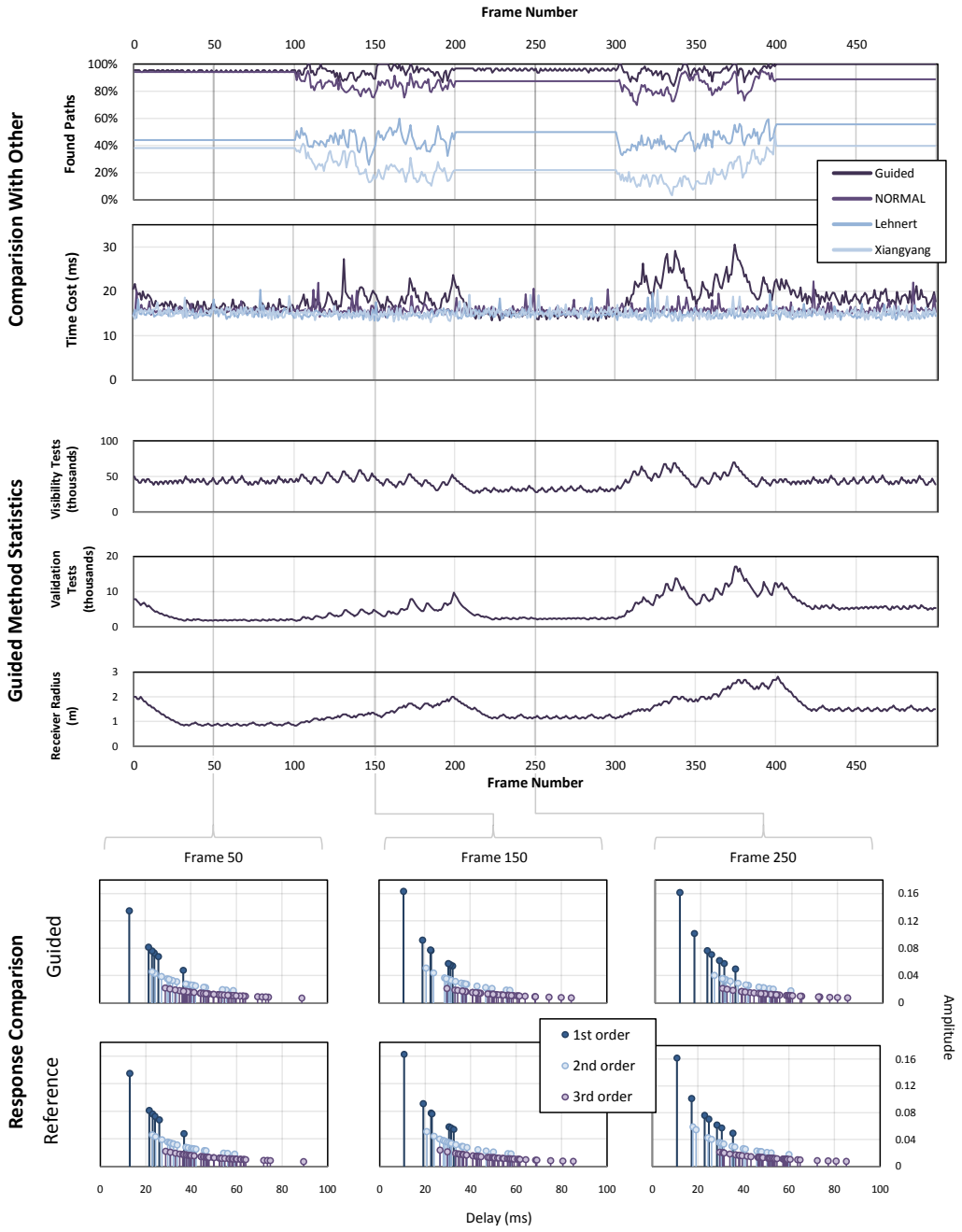


Figure 5.13: **Music Hall animation:** This figure compares various receiver size models to our guided method. During frames 100-199, the source moves to a new position; during frames 300-399, the receiver moves to a new position. The top charts show the accuracy and the time cost over the animation sequence. The three middle charts show number of validation and visibility tests conducted by our guided method, in addition to the radius of the receiver. The bottom charts show the impulse response of our method compared to an accurate image source simulation for frames 50, 150, and 250. Our method is more accurate than the others, while incurring a small additional time cost.

Exact GA algorithms: Exact GA algorithms are based on beam tracing [Laine et al., 2009] and conservative ray-frustum tracing [Chandak et al., 2009]. These methods can accurately compute all the specular reflection and edge diffraction paths. However, frustum tracing methods [Chandak et al., 2009] can take a $6-8$ seconds on simple models composed of a few thousand triangles with three orders of reflections on a single core and beam tracing algorithms are almost an order of magnitude slower than frustum tracing.

Ray-frustum tracing: These methods trace frusta and use a combination of exact intersection tests and discrete clipping. Overall, their accuracy lies between discrete ray tracing and beam tracing methods. However, current implementations can compute the propagation paths with specular reflection and edge diffraction at $2-3$ fps on a 7-core PC. In our tests, our system running on a single GPU is about an order of magnitude faster than ray-frustum tracing.

Other systems: ODEON is a popular acoustics software which can compute specular reflections, diffuse reflections, and diffraction [Christensen, 2009] and is perhaps the most widely used commercial system for architectural acoustics. ODEON performs early specular reflections and diffraction using a combination of ray tracing and image source method [Christensen, 2009]. For diffraction, ODEON computes at most one diffraction path from a source to a receiver [Pierce, 1974]. CATT-Acoustic [CATT, 2002] is another popular room acoustic software which performs specular and diffuse reflections using a combination of image source and ray tracing methods, along with randomized tail-corrected cone tracing [Dalenbäck, 1996]. RAMSETE [RAMSETE, 1995] is a GA based prototype acoustic system. It performs indoor and outdoor sound propagation using pyramid tracing [Farina, 1995]. It can perform specular reflections, diffuse reflections, and multiple orders of diffraction using the Kurze-Anderson [Kurze, 1974] formula for free edges. RAVEN at RWTH Aachen University is a framework for real-time auralization of virtual environments [Lentz et al., 2007, Schröder and Lentz, 2006]. RAVEN uses spatial hierarchies to render specular reflection and simplified diffraction in dynamic scenes with hundreds of triangles. Another prototype system for real-time auralization is based on beam tracing [Funkhouser et al., 1998, Tsingos et al., 2004]. It can perform specular reflections and diffraction using beam tracing. A beam tree is constructed in an offline step which limits the system to either a static source or receiver position. RESound [Taylor et al., 2009a] is also a real-time auralization system. It is based on a combination of frustum tracing

and ray tracing to handle specular reflections, diffuse reflections, and UTD diffraction in dynamic scenes.

CHAPTER 6: RENDERING MASSIVE MULTI-USER ENVIRONMENTS

The methods presented thus far have high computational requirements. Many modern mobile devices lack such capabilities. It is possible to precompute acoustic responses for scenes using GA methods like the ones described in the previous chapters. In this chapter we discuss how to efficiently precompute, store, and render audio for thin mobile clients using a server in the cloud.

We introduce a similarity metric for reducing the amount of GA simulation samples needed in the scene. We also discuss how the simulation results can be compressed and stored efficiently, while still having fast rendering capabilities for thin clients.

6.1 Geometric acoustic similarity measure

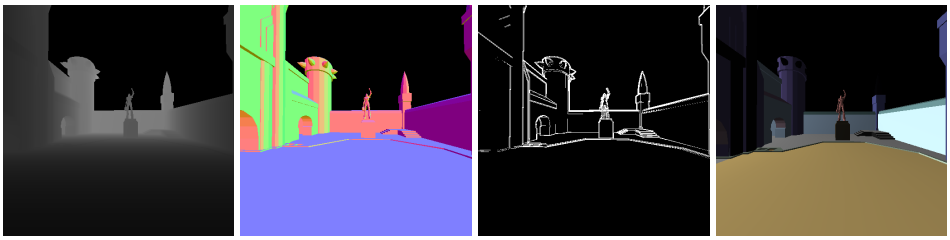
In this section, we introduce our metric and present a GPU-based algorithm for fast computation. In Section 3, we use this metric to decompose the scene into regions.

Our goal is to sample the acoustic field and form a database of acoustic responses, using as few samples as possible, while still generating a reasonable set of responses. The final output of any acoustic simulation is derived from the source/receiver configuration, the location of obstacles and the material properties of the scene. This can be expressed using the room acoustic rendering equation (RARE) [Siltanen et al., 2007], which describes the acoustic radiance from a point in the scene as

$$\ell(x', \Omega) = \ell_0(x', \Omega) + \int_G R(x, x', \Omega)\ell(x, \Gamma)dx, \tag{6.1}$$

where the outgoing acoustic radiance ℓ for point x' and angle Ω is a sum of the emissive radiance $\ell_0(x', \Omega)$ and the integral of all reflected radiance from point x. The reflected radiance is scaled by the function R

$$R(x, x', \Omega) = V(x, x')\rho(x', \Theta, \Omega)g(x, x'), \tag{6.2}$$



(a) (b) (c) (d)

Figure 6.1: **Sample signature for FPS scene in Figure 6.8(b):** The components of the similarity measure: (a) distance with black being near and white far, (b) direction with vectors shown as RGB components, (c) discontinuities in depth and direction, and (d) materials with three frequency bands shown in RGB components.

which accounts for the visibility V between points, the reflectance function ρ, and the distance and orientation of the points g. The function $g()$ can be expressed as:

$$g(x, x') = \left[\mathbf{n}(x) \cdot \frac{x' - x}{|x' - x|}\right] \left[\mathbf{n}(x') \cdot \frac{x - x'}{|x - x'|}\right] \frac{S_{|x-x'|}}{|x - x'|^2}, \tag{6.3}$$

where $\mathbf{n}$ represents the surface normal at a point and S represents the effects of propagation over a distance, i.e. delay and attenuation.

Our goal is to quickly sample the scene and find locations where the acoustic field has a high gradient to guide the sampling. One possibility is to use the RARE to sample the scene, but this can be rather expensive for a large environment. Instead, we compute a first-order approximation that can be evaluated quickly using standard rasterization algorithms such as *cube mapping* [Greene, 1986]. Using axis-aligned cube maps, we sample geometric and spatial data from the scene. From the cube map data, we extract several values related to the physical properties that influence the propagation of sound waves in the scene: surface distance, surface orientation, surface discontinuities, and surface absorption. Figure 6.1 shows the visual data that is sampled.This spatial data corresponds to some of the physical properties that influence first-order reflections: path length, reflection direction, surface reflectance properties, and diffusion.

6.1.1 Distance

Acoustic waves that arrive at the receiver as reflections must first bounce off an object in the scene. The path the sound travels results in delay, creating temporal effects that humans use to determine environment properties. In media that absorbs sound, the distance the sound travels

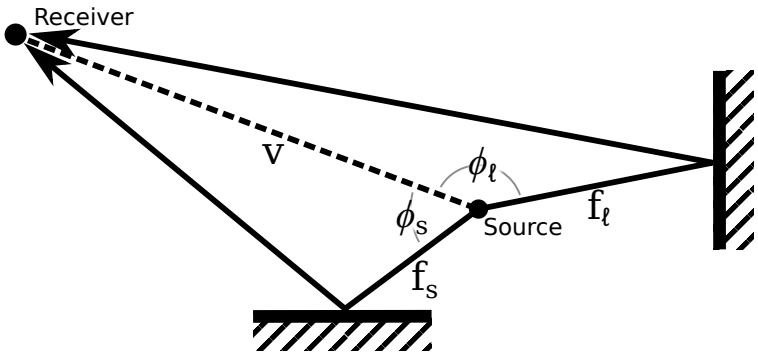


Figure 6.2: **Surface distance:** The first-order propagation distance p is directly related to nearby reflectors. v is the direct path to the receiver. f_s represents the shortest first-order reflection path as ϕ_s goes to 0. f_ℓ represents the longest first-order reflection path as ϕ_ℓ goes to π. Our algorithm measures the f terms of p.

modifies the signal further, as some energy is lost to absorption. The S term in RARE (Equation 6.3) accounts for effects that are related to the propagation distance. However, in order to measure this effect exactly, one must perform an acoustic simulation that computes both early reflections and late reverberations between the source and the receiver. It can be expensive to perform this computation for each source-receiver pair. Instead, we use a simpler approximation, and only sample first-order geometric properties to compute the reflection paths.

We note, using the law of cosines, that the length of the reflection path, p, can be computed by forming a triangle between the source, the reflector, and the receiver (Fig. 6.2). This is given as

$$p = f + \sqrt{f^2 + v^2 - 2fv\cos\phi},$$

where f represents the path to the reflector, v is the direct path to the receiver position, and ϕ is the angle between these vectors. As ϕ approaches zero, the path approaches the shortest possible reflection path between the source and the receiver; when ϕ is π, the reflection path is the longest possible path between the source and the receiver. Figure 6.2 illustrates the shortest path as f_s and the longest path as f_ℓ. The value of ϕ, and consequently p, are only known at runtime, so we sample the f component and use it to approximate the distance in the final acoustic response.

6.1.2 Surface orientation

The direction of earliest incoming sound paths to a receiver is highly indicative of the direction of the sound source, and is a key element in sound localization. The $g()$ function in Equation 6.3

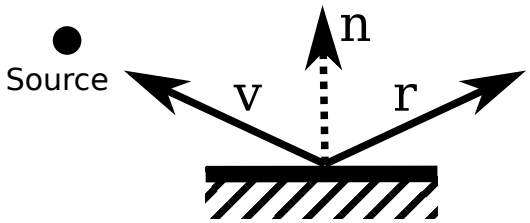


Figure 6.3: **Surface orientation:** Reflection direction r varies as a property of the incoming vector v and the surface normal orientation n.

describes how the direction of an object's surface normal directly influences the direction of any reflected sound paths off the surface. As in our handling of arrival delay, we avoid computing the full acoustic simulation by sampling only the first-order geometric properties.

Reflection direction $\mathbf{r}$ can be determined based on the view direction $\mathbf{v}$ and the surface normal direction $\mathbf{n}$ (Fig. 6.3). We note that $\mathbf{v}$ is fixed by the cube map sample location and the reflection direction is a function of $\mathbf{n}$, given as:

$$\mathbf{r} = 2(\mathbf{v} \cdot \mathbf{n})\mathbf{n} - \mathbf{v}.$$

Since the surface normal directly influences the direction of first-order reflection, computing the difference in first-order normals between two scene locations provides an approximation of the difference in reflection direction, and the actual difference in early onset direction.

6.1.3 Surface discontinuities

The amount of diffraction and diffuse scattering from surfaces can significantly influence the final acoustic response at the receiver's location. The RARE does not directly account for edge diffraction effects, but the reflectance function $\rho()$ (Equation 6.2) can model such scattering. Scattering occurs at the boundaries, which correspond to edges in 3D scenes. Simulating scattering effects using geometric acoustics is time-consuming and challenging, as compared to reflection effects [Calamia and Svensson, 2007, Taylor et al., 2012], and may require an accurate wave-based solver [Raghuvanshi et al., 2010]. Instead, we use an approximation based on first-order geometric properties that influences these effects, including depth and orientation discontinuity.

We measure the surface gradient with respect to surface normal and depth to estimate diffraction and diffuse properties. When sampling our cube maps, we record the depth values and surface

normals. Next, we use a series of 2D operators over the cube face to find discontinuities in depth gradient and normal gradient. These discontinuities represent potential regions where diffraction and diffusion effects are significant. First, a 2D gradient is computed from the depth information:

$$\nabla_x = [1, -1]; \quad \nabla_y = \begin{bmatrix} 1 \\ -1 \end{bmatrix}.$$

This results in a two-component image that represents depth changes in x and y directions. We look for discontinuities in this image to estimate where scattering is likely to originate from. An edge detection kernel is applied to the components of the gradient:

$$k = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

The maximum value, u, of the two components is retained and clamped to the range $[0, 1]$, resulting in a black and white map used to detect depth and normal discontinuities. Since the depth and normal discontinuities influence the scattering of sound in the scene, we use these discontinuities as a component in our measure.

6.1.4 Surface material

The material properties govern the absorption and scattering effects. The RARE accounts for materials in the bidirectional reflectance distribution function (BRDF) term, $\rho(\Omega_i, \Omega_e; x')$ (Equation 6.2). A commonly used BRDF segments the reflectance into pure absorption, diffuse reflection, and specular reflection components using an absorption factor α and a diffusion factor δ. These factors are defined over several frequency bands to define absorption, diffuse, and specular coefficients:

$$\underbrace{\alpha}_{absorption} + \underbrace{\delta(1-\alpha)}_{diffusion} + \underbrace{(1-\delta)(1-\alpha)}_{specular} = 1.$$

We directly sample the material of the first-order surface and use that to approximate the resulting effect. Note that α is a vector over several frequency bands.

6.1.5 Overall similarity measure

The components of the measure are stored on each face of the cube map and constitute a first-order response measure related to the surfaces near the sampling point. Storing all cube-face images for each sample point requires large amounts of storage. In order to reduce the memory overhead, we perform an integration step to compute the mean of the geometric properties on each cube face. Given a geometric value q (representing one of the four data values) across s cube map samples, we compute the mean $q_{avg} = \sum_{i=0}^{s} \frac{q_i}{s}$.

After the integration step, the cube map data for each axis is reduced to a four tuple of mean surface distance f_{avg}, mean surface orientation $\mathbf{r}_{avg}$, mean surface absorption α_{avg}, and mean number of pixels that lie on possible scattering edges u_{avg}. The final similarity measure $\mathbf{G}$ is a six row matrix composed of this four tuple for each of the six axis-aligned sample directions. Each direction is referred to as a superscript (shown as x below):

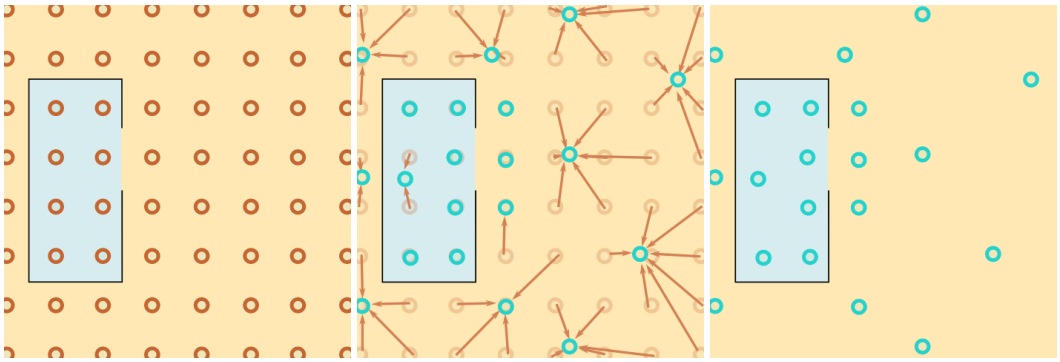
$$\mathbf{G} = [f^x_{avg}, \mathbf{r}^x_{avg}, \alpha^x_{avg}, u^x_{avg}].$$

6.2 Scene decomposition and sampling

In this section, we present our scene-decomposition algorithm, which greatly reduces the time and storage cost of the acoustic precomputation while incurring only small error in response accuracy. The similarity measure, $\mathbf{G}$, described above is simple and fast to compute. As a result, our algorithm first evaluates the similarity measure on a dense grid of sample points over the scene. Once the surface properties near each sample point have been measured, sample points with similar properties are merged, which reduces the total sample point count in the scene. We first discuss the method to compute the difference between two samples, followed by the techniques used to segment the scene and compute acoustic responses.

6.2.1 Similarity comparison

In the previous sections, we described our method for extracting geometric properties that correspond to expected reflection-path length, expected reflection direction, and expected diffuse



(a) (b) (c)

Figure 6.4: **Adaptive sampling:** (a) Regular grid sampling creates a very high number of samples; (b) we remove redundant samples; (c) adaptive sampling of the scene with fewer samples.

energy. We assume that if two sample points have similar values for these parameters, it is likely that the acoustic field measured around the sample positions will be similar. The difference $\mathbf{S}$ between sample points $\mathbf{a}$ and $\mathbf{b}$ can be computed by taking the component-wise difference of the quantities described above:

$$\mathbf{S} = |\mathbf{G}_a - \mathbf{G}_b|.$$

When calculating the difference in the mean surface distance, f_{avg}, the percent difference distances is used. This results in a difference value in the range $[0, 1]$, which can account for changes in both short and long distances. Similarly, for direction, the dot product is taken and scaled to the range $[0, 1]$. Since the absorption vector, α, is already bounded $[0, 1]$, we use the maximum scalar absorption difference for any sub band. The discontinuity measure is also naturally bounded $[0, 1]$. The result is that $\mathbf{S}$ is a 6×4 matrix with the components being scalars in the range $[0, 1]$.

$\mathbf{S}$ will be used to determine if two locations in the scene are similar. In addition to the $\mathbf{S}$ parameters, line of sight is also used to restrict similarity; we assume that there is a line of sight (LOS) between the sample points considered similar. This forms the *basis* of our similarity metric: sample points are likely to have a similar acoustic response if the early response (i.e., cube map measured data) is similar and if the points are visible to each other (LOS restriction).

6.2.2 Sample merging

Once the similarity properties between all neighboring sample points have been measured, the scene-decomposition algorithm is used to compute adaptive sampling. We note that the similarity

measure $\mathbf{S}$ is useful for evaluating whether two sample positions are similar, but not directly useful for eliminating sample positions. If a sample point is removed, the nearby acoustic field will be sampled sparsely, which may lead to more error at reconstruction when a receiver is placed near that sample point.

The sample points are merged in an adaptive quad-tree like manner. This is done by recursively subdividing the sample points into regions. The corners of regions are evaluated for similarity. Most quad trees divide the working set by four at each level. Our reduction algorithm also attempts to subdivide the four half regions (two along each dimension). This reduces the number of subdivision steps by a small amount.

The maximum norm is taken for each property in $\mathbf{S}$ across all dimensions (e.g. column vectors), and the result is used to form a weighted average error, s_p. This error is compared against the similarity threshold s_{thr}:

$$s_p = \frac{1}{4}(||\mathbf{S}_0||_\infty + ||\mathbf{S}_1||_\infty + ||\mathbf{S}_2||_\infty + ||\mathbf{S}_3||_\infty),$$

$$s_p < s_{thr}.$$

If the error criterion is satisfied, all points bounded by the corners may be combined into a single region. If the corners do not satisfy the metric, the region is further subdivided and the tests are repeated for all sub-regions. Each resulting region then contains similar geometric properties based to the degree specified in s_{thr}. The details of the adaptively merging algorithm is given in Algorithm 1.

6.2.3 Acoustic regions

The output of the refinement stage is an irregular arrangement of merged samples (see Figure 6.4). The process of sample merging also gives us acoustic regions: if two sample points are merged, they were classified to be similar by the error criteria, and their original positions in the 2D grid are marked as belonging to the same region. As points are merged, the acoustic regions grow until the edges at the boundary of the region no longer satisfy the error criteria. At runtime, the acoustic

Algorithm 1 Adaptively merge regions with similar signatures. Each region can be subdivided into four subregions. Neighboring subregions could form larger rectangular regions. In this algorithm, the ordering of merges ensures that the largest possible regions are merged. That is, if there is a possible merge of a rectangular region, it is used. The $similar()$ function tests if the signatures in the region satisfy the LOS and $s_p < s_{the}$ constraints.

function MERGE(corner, size)
  $c \leftarrow corner$
  $s \leftarrow size$
  $h \leftarrow size/2$
  $entireRegion \leftarrow (c, c), (s, s)$
  $topRect \leftarrow (c, c), (s, h)$
  $botRect \leftarrow (c, c + h), (s, h)$
  $lefRect \leftarrow (c, c), (h, s)$
  $rigRect \leftarrow (c + h, c), (h, s)$
  $topLef \leftarrow (c, c), (h, h)$
  $topRig \leftarrow (c + h, c), (h, h)$
  $botLef \leftarrow (c, c + h), (h, h)$
  $botRig \leftarrow (c + h, c + h), (h, h)$

  if similar($entireRegion$) **then**
    merge($entireRegion$)
  else if similar($topRect$) and similar($botRect$) **then**
    merge($topRect$); merge($botRect$)
  else if similar($lefRect$) and similar($rigRect$) **then**
    merge($lefRect$); merge($rigRect$)
  else if similar($topRect$) **then**
    merge($topRect$); merge($botLef$); merge($botRig$)
  else if similar($botRect$) **then**
    merge($botRect$); merge($topLef$); merge($topRig$)
  else if similar($lefRect$) **then**
    merge($lefRect$); merge($topRig$); merge($botRig$)
  else if similar($rigRect$) **then**
    merge($rigRect$); merge($topLef$); merge($botLef$)
  else
    merge($topLef$); merge($topRig$); merge($botLef$); merge($botRig$)
  end if
end function

region of the source point and the receiver point must be selected based on the spatial position of the source and the receiver.

Since the source and the receiver will not lie exactly on the sample points used during precomputation, the appropriate acoustic region is computed by checking the four nearest grid locations. The simplest case is when the four nearest grid locations belong to a single acoustic region. This region can be immediately identified. If not, LOS queries with $O(\log n)$ time cost are performed from the four points against the objects in the scene. The closest point with a clear LOS is selected as the closest grid node.

6.2.4 Simulation of sound propagation

Our method treats sound propagation simulation as a black box, and its only requirement is with respect to the input: the scene representation, along with the positions of source and receiver. From this input, it computes an impulse responses as the output. We use geometric acoustics (GA) simulation in order to compute responses on large scenes, since the complexity of wave-based simulators is a linear function of the scene volume. Our simulator traces specular rays and diffraction rays [Taylor et al., 2012, Vorländer, 1989] and computes paths between a source-receiver pair; it then outputs an acoustic impulse response that represents environmental effects on sound waves traveling from the source to the receiver. This impulse response can be convolved with any input sound signal to render an output signal with the appropriate effects.

We identify the center of each region obtained from the decomposition process; this point becomes a sample point for the propagation simulation. Each sample is modeled as a source from which rays are traced into the scene. Every sample is also considered a receiver, and propagation paths are recorded at each sample. Even with region segmentation, there are still a high number of source-receiver pairs. Since each ray would require intersection with n receivers at the sample points, and each sample point also acts as a source, this results in $O(n^2)$ ray tests using naive methods. We include receiver locations in a standard ray-tracing acceleration structure to reduce the receiver intersection cost to $O(n \log n)$.

6.3 Storage and reconstruction

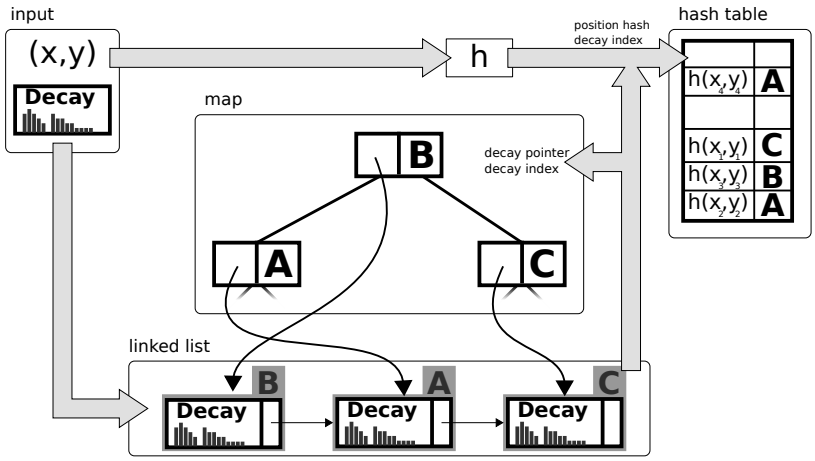
This section describes our storage algorithm. We have designed a method to efficiently compress, store, and retrieve audio-response data. Since our method is designed for large scenes, our storage database must scale in time and space. An acoustic response is a signal over time or frequency. If spatialization effects are desired, direction data must also accompany the signal. For our purposes, three portions of the response data are recorded: decay profile, incident direction, and diffusion. However, the methods described below can encode any type of data.

6.3.1 Response representation

The paths computed in the trace step form the acoustic response between the two samples and represent the environment's filter of the acoustic signal between the two locations. Such filters can be represented compactly by sampling the energy decay profile through time [Merimaa and Pulkki, 2004, Tsingos, 2009]. The decay profiles are built by integrating the energy in the impulse response over small time-steps and a number of frequency sub-bands. In this way, both temporal and frequency resolution can be controlled by the user.

The response is stored in three parts: pressure values, a direction value in 2D, and a diffuse coefficient to indicate how strong the directionality is. Similarly to other methods [Tsingos, 2009], we store pressure values for several spectral sub-bands quantized in the time domain. An energy-weighted average direction of incidence is stored at time-step resolution; all pressure frequency bands share the same direction. Similarly, a directional-to-diffuse energy value is also computed [Merimaa and Pulkki, 2004].

In the theoretical worst case, n^2 responses could be generated (n sources to n receivers), this does not happen in practice. Since there exists a maximum sound energy level for each source (e.g. human voices), there must be a bound on the number of responses generated. Moreover, the energy emitted is diminished by air absorption as it travels. For a given grid of n sample points, only k samples will have audible response to the human voice. The k factor is based on the distance between the sampling points, dissipation by the air attenuation model, the maximum pressure level that is simulated, and the minimum pressure level that can be sensed.



(a)

Figure 6.5: **Data insertion:** The decay data A, B, C is appended to a linked list and the length of the list is the data index. The index is stored with a pointer to the decay data as a pair in a map. The index is paired with a position hash and stored in a hash table. The insertion can be performed in average $O(\log n)$ time. After all the data is stored, the linked list is converted to a linear array, for a total time complexity of $O(n)$ and average storage cost of $O(n)$.

This bounds the number of responses that can be be inserted in the database, i.e. kn. This also bounds the time cost of the simulation, since the simulation can be confined to a region large enough to enclose the k sample points. As the scene size increases, the fact that k is constant allows the time and space required by the simulation to be bounded.

6.3.2 Storage data structures

Since there is response data for each source-receiver pair, it is logical to store the data in a linear vector with an index table to query by source-receiver. This approach allows $O(1)$ insert and retrieval time. However, since the number of source-receiver pairs can be very large, the $O(n^2)$ scaling of the table index dominates the storage cost. For example, a scene of 35K sample points would result in nearly 5GB of storage for the table itself. On very large scenes, the size of the index can exceed the storage needed for the acoustic data. It is thus desirable to reduce the size of the actual data stored and the size of the index needed to query the data.

6.3.2.1 Storing acoustic data

Based on acoustic reciprocity, we assume that acoustic responses from swapping source position and receiver position should be similar. We capitalize on this factor to reduce the amount of acoustic

data stored. To perform this step, we detect if an insertion will result in duplicate data. We can impose strict weak ordering on all the acoustic data, so a map offers fast insertion and query time. However, the ordering of map nodes is not fixed until all the insertions are completed. Since the index must be built at the same time, we use a linked list to store the actual acoustic data, and the map references this data. The ordering of this list is fixed through the insertion process, and the index is built against the list. This allows insertions to take place in $O(1)$ time for the list and $O(\log n)$ time for the map. Moreover, space complexity for both structures is $O(n)$. Refer to Algorithm 2 for details.

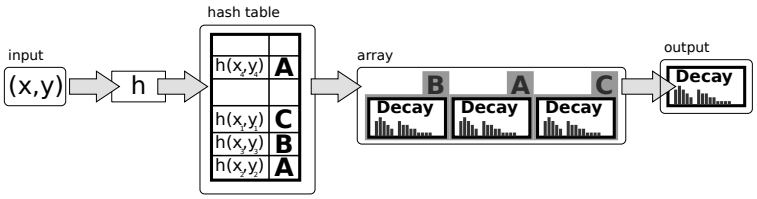
In our algorithm, we store decay profile, direction, and diffusion separately, and store only unique responses to take advantage of reciprocity. This allows very efficient duplicate storage, and, most importantly, sparse storage of the empty response. Since we are simulating very large scenes, the most common output from the simulator is an empty acoustic response.

Algorithm 2 Insert response data. All the insertions are performed during simulation phase. We give details of our insertion algorithm based on the hash table representation.

function INSERTRESPONSEDATA(data, location)
  if $data = emptyData$ **then**
    return
  end if
  $ptrD \leftarrow pointer(data)$
  $id \leftarrow map.Query(ptrD)$
  if $id = emptyID$ **then**
    $list.Insert(data)$
    $ptrD \leftarrow pointer(list.End)$
    $map.Insert(ptrD)$
    $pair(location, list.Size)$
    $hashmap.Insert(pair)$
  else
    $pair(location, id)$
    $index.Insert(pair)$
  end if
end function

6.3.2.2 Efficient data indexing

For the acoustic data, both insertions and queries are performed during the simulation. At runtime, only the acoustic data is queried. Thus, when all insertions are complete, the map is discarded and the list is converted to a vector for efficient storage. The separate index is used to



(a)

Figure 6.6: **Data access:** For runtime access, the query position is hashed and the decay index is found in average $O(1)$ time. The data array is then queried for the final decay data.

query acoustic data from this vector. Since the vector representation supports random access in $O(1)$ time, the cost of querying acoustic data is thus dominated by index lookup.

The data stored in the index is a key-value pairing of a hash with the index to the acoustic data in the previously discussed list. When the list is converted to a vector, the ordering is preserved. Combining the source-receiver indices results in a good hash function; it can be represented as a single integer that is guaranteed to not collide with any other hashed values. We also note that we expect many responses to be empty and thereby, represent such a quiet response (i.e. no sound). Based on these assumptions, we do not store keys that reference empty acoustic data. Instead, if a key is not found at query time, it is assumed that the data for the key is the empty response. This significantly reduces the size of the index.

This data is queried often at run time to lookup the acoustic data vector locations. Therefore, during we construct a hash map from the unordered vector to represent the final index for $O(1)$ query time and $O(n)$ storage cost on average. We also found that sorted vectors performed equally well, even with $O(\log n)$ query time due to the constant values associated with each data structure. The responses can be accessed in constant time using our data structure (see Figure 6.6 and Algorithm 3). The source-receiver hash map can be queried in $O(1)$ time on average.

Algorithm 3 Get response data. This query is performed at runtime to lookup the acoustic data vector locations.

function GETRESPONSEDATA(location)
  $id \leftarrow index.Query(location)$
  if $id = emptyID$ **then**
    return $emptyData$
  else
    return $data[id]$
  end if
end function

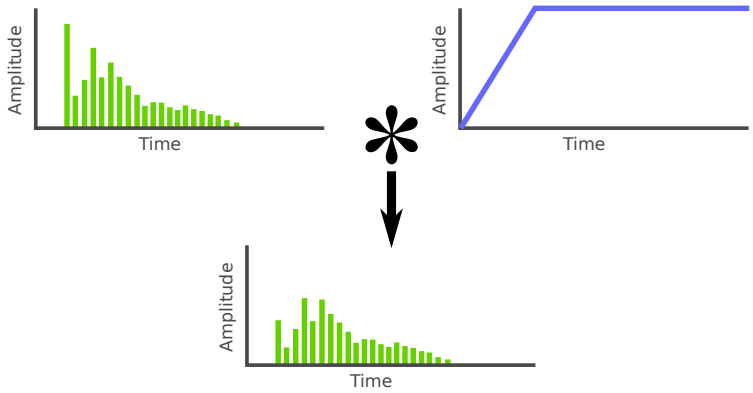
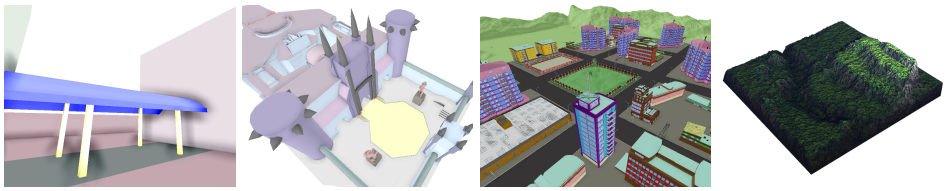


Figure 6.7: **Early response attenuation:** The early response pressure is attenuated for source/receiver pairs in the same region.

6.3.3 Response reconstruction and rendering

When sampling an acoustic region, the source and receiver positions for response sampling are centered in the region. However, at runtime, the source and the receiver may be in the same region separated by the full width of the the region. Since our algorithm has only computed a single response for the entire region, we approximate the distance attenuation by scaling the early portion of the decay response. For example, we define the early portion as the first 140ms of the response. We scale each pressure value in the early portion by $\frac{1}{d}$, where d is the distance that leads to a response during that time step. This attenuation factor is linearly reduced for each consecutive time step, so that no attenuation is applied to the final early time sample. This process allows the early field to be attenuated without altering the standing late reverberation field (see Fig. 6.7). The direct path and early reflection rendering is performed on the server using a block-based convolution in the Modified Cosine Transform Domain [Tsingos et al., 2011]. This is efficient as audio is often coded in that same domain. The direct path and decay data are interpolated over time as the user moves through the scene to avoid audible discontinuities in the output.

In many situations, the server can process the reverberation in realtime and operates in full mixing mode. This means all the processing is done on the server and stereo or multichannel results are generated and sent back to the client. Thanks to our representation of reverberation filters, it is also possible to send only a mono mixture and energy-weighted direction and direct/diffuse metadata to further reduce bandwidth and enable flexible spatialization on the client. No matter the rendering arrangement, all head-related filtering is performed at the client-side.



(a) (b) (c) (d)

Figure 6.8: **Example scenes:** Our algorithm can generate environmental acoustic effects in large virtual worlds and games. We show different benchmarks with their dimensions in meters: (a) simple outdoor scene ($33 \times 33 \times 10$); (b) first person shooter (FPS) game scene ($30 \times 60 \times 20$); (c) city scene ($600 \times 980 \times 33$); (d) canyon model ($4000 \times 4000 \times 100$). Our approach scales with the size of these models and can handle a large number of sources and receivers in multi-player enviroments at interactive rates.

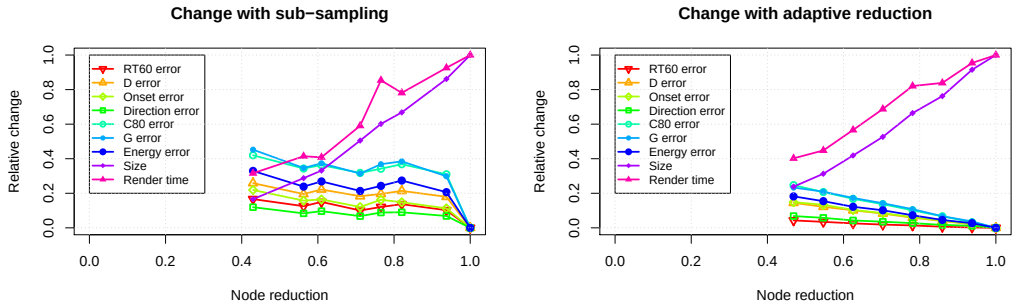
For environments with very long reverberations, which cannot be completely processed on the server, a further optimization would be to process only the direct + early part of the response (up to some number of blocks) on the server. An additional dry mono mixture and representative late reverb metadata would also be sent to the client so that it can provide local late reverb processing but this is not demonstrated in our video or results. This necessary metadata is the local late decay rate in the vicinity of the listener and an average late decay rate for all the sources.

6.4 Validation

In this section, we evaluate the accuracy of our approach using various perceptual metrics used in room-acoustic analysis. We used several example scenes, shown in Figure 6.8. These selected test scenes represent the likely use cases, and comprise both indoor and outdoor scenes, corresponding to game maps and virtual worlds. The details about these models and the underlying grid resolution are given in Table 6.1.

Scene	# Triangle	Size (m)	Grid spacing (m)	Sample count
Simple outdoor	2k	33 x 33 x 10	4	81
FPS game	14k	30 x 60 x 20	4	128
Small city	2k	100 x 100 x 33	4	625
Large city	2k	247 x 168 x 33	4	4.9k
Canyon	540k	4k x 4k x 100	4	1,000k

Table 6.1: **Example scenes:** Physical sizes for the indoor and outdoor scenes are given in meters (m). The sample count is for a regular grid at the given resolution.



(a) (b)

Figure 6.9: **Sampling accuracy vs. error and cost:** Naive subsampling (a) is the most common way of reducing time and storage cost. As the threshold error in our adaptive sampling algorithm changes (b), the overall error in the acoustic evaluation metrics increases, while overall storage, precomputation time, and number of samples decrease (FPS game scene).

6.4.1 Error computation

We have measured various errors related to acoustic responses generated by our algorithms, including the similarity measure and the scene decomposition. We measured these errors in the reconstructed results based on certain properties of the impulse responses. In particular, we used some well-known *acoustic evaluation metrics* related to evaluating acoustic responses: onset time delay, initial onset direction, reverberation time in the form of RT60, and signal definition in the form of D [ISO 3382, 2009, Kuttruff, 2007]. We performed GA simulation using our precomputation algorithm, computed impulse responses, and compared our results to a ground truth dataset. The ground-truth data is the full set of responses from the GA simulator with the source and the receiver positions at the same grid size, but without any sample reduction. We used a dense, uniformly-sampled grid to generate the ground-truth, as shown in Table 6.1. In both simulations, we traced 50K rays from each sample point, performing 50 orders of specular and diffuse reflections and 5 orders of edge diffraction. The relative error is calculated by dividing the absolute error by the maximum possible error for each metric. Below we give details related to error computation and evaluation and illustrate many error maps (as shown in Figure 6.10 and Figure 6.11). The low error in these maps validates our approach with respect to various acoustic evaluation metrics.

6.4.2 Similarity measure thresholds and validation

During the precomputation step, our method computes an acoustic similarity measure for positions in the environment. An error threshold is used to select similar regions during the scene decomposition. When merging the sample points and performing scene decomposition, the error between two sample points ranges in the interval $(0, 1]$. For the results presented in this thesis, we set our error threshold s_{thr} to such that 75% of the original nodes are removed.

Since our approach reduces the number of acoustic responses based on our similarity measure, it is possible that the final acoustic map may not accurately represent the responses of the original acoustic environment. In order to show the accuracy of our decomposition algorithm, we compare the properties of a full ground-truth simulation to properties generated from our method in two scenes with the initial grid size set to $4m \times 4m$.

By adjusting the error threshold in the region-segmentation step, higher accuracy in acoustic properties can be achieved. However, this increases the number of acoustic regions that need to be stored. We expect simulations with fewer regions to have greatly reduced computational overhead, but to also have a small increase in errors in the impulse response. Figures 6.9 highlights the relative variation in error and computation cost as the region merge threshold is adjusted. The errors in this figure are computed by taking the average of the error for every possible position of the receiver, similar to the error maps used in Figure 6.10.

6.4.3 Acoustic property error calculation

In order to compare our approach to a ground-truth simulation, we consider four acoustic evaluation metrics described below. More details about these metrics can be found in [ISO 3382, 2009, Kuttruff, 2007]. We give details on how we compare the values of these metrics to our simulation results with the ground-truth. Due to our adaptive decomposition, some positions in our precomputed solution may be associated with empty responses, while the corresponding solution in the ground-truth data may have a non-zero response (and vice versa). In this case, the error at those positions is undefined and we use a worst case error value at these locations for our error metrics.

Onset delay (Onset): This is the time it takes for the earliest first-order path (reflection or diffraction) to reach the receiver. This is very similar to the Initial Time Delay Gap [Andō, 1998],

but we cannot guarantee the presence of a direct contribution; so we measure the delay from the initial simulation time. We compute this parameter by finding the delay of the first impulse in the acoustic response. Moreover, we assume that a missing value indicates that the onset occurred outside the measurement range (4 seconds in our implementation). In these cases we set the error to the difference between the maximum measurable onset and the known delay. Otherwise, this error is computed as a time difference between the two response onset delays.

Onset direction (Dir): This is the average direction of all contributions for the initial impulse. It is the first directional sound that a listener hears in the absence of direct sound and is important for localization. We compute this parameter by averaging the directions of all contributing paths that contribute the first impulse. This error is computed as the dot product of the normalized onset directions of the responses. We scale the error over the $[0, 1]$ interval, where an error of 1 represents the maximum error of $90°$. Since highly diffuse signals will have nearly random direction, the direction error is scaled by the strength of the spatial response. This means that highly diffuse signals will produce low directional error, while signals with less diffusion could produce more error.

Reverberation (RT60): This corresponds to the time it takes for sound waves to decay past a certain threshold. In particular, RT60 is the time needed for the sound to decay -60dB. The reverberation decay helps the listener to determine the size of the space. We compute this using the Schroeder method [ISO 3382, 2009, Schroeder, 1965] by matching a least-squares fit of the pressure decay in log space. We compute the time to decay to -60 dB by fitting a least square line to both the decay responses; we report the error as the time difference in decay time. For reverberation, we assume that a missing value indicates that the signal decayed immediately, so the error is the decay time from the other signal.

Definition (D) This value represents the ratio between energy levels in the early and late portions of the acoustic response. We use D, the ratio of the integral of the first 50 milliseconds of energy to the total energy in the response, as defined in [ISO 3382, 2009, Kuttruff, 2007]. D represents the intelligibility of speech signals. If both responses have a value, the error is the difference between the measured D values. If the response is empty, we assume it has a D value of 0.

Clarity (C80) This value is similar to D, except it is the ratio between the energy in the first 80 milliseconds of the response to the rest of the energy [Kuttruff, 2007]. In short responses, this can lead to a division by 0 if there is no energy past the initial 80 milliseconds. We compute error in

this value as a percent difference in decibel values. If there is no energy in the later portion of the response, we avoid dividing by 0 and assign a default value of 20 decibels.

Strength (G) This value represents the strength of the response as a ratio of response energy to the direct path at 10m [Kuttruff, 2007]. We compute error in this value as a percent difference in decibel values.

Total energy (Energy) This value represents the sum of the sound pressure levels over the length of the response. Before the sum, the frequency band values are averaged into a single time domain response. We compute relative error in this value as a percent difference in decibel values.

6.4.4 Error map calculation

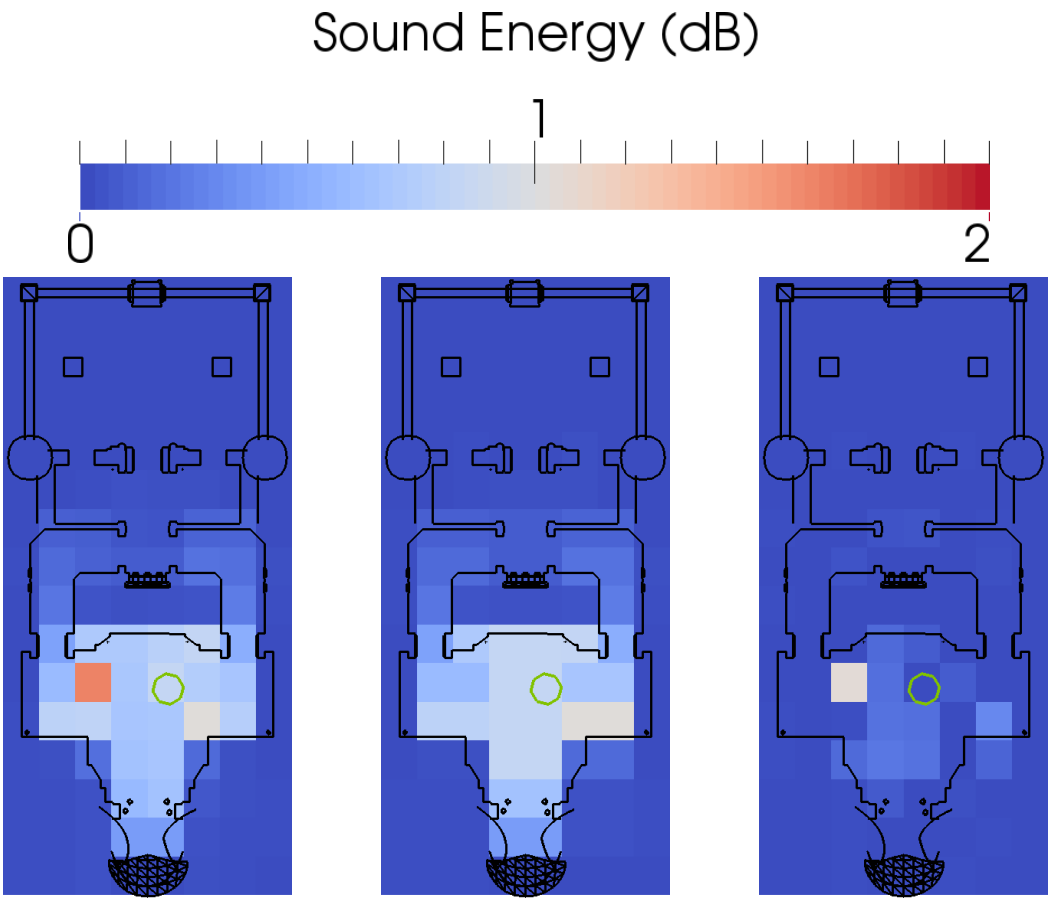
Since it is difficult to report the thousands of error values from a single scene, we compute various maps that average the error values over the entire scene. All of these calculations are performed on the acoustic evaluation metrics defined above. The most intuitive error is the difference between a property f from a ground-truth acoustic response and a property c from our region-segmented acoustic response. For a given source position, we measure the acoustic property difference of all receiver positions over the entire map as $|f - c|$. This differencing process is shown in Figure 6.10.

We weight the acoustic property error relative to audibility a, which we define as the maximum decibel value in the acoustic response. The error for a response is scaled over the range 0 to 2 for audibility of -60 to 0 dB. Thus, the error in quiet responses is weighted less, while error in loud responses is weighted more.

Using these difference maps, we compute the relative error in acoustic evaluation metric over the range of that metric (g) and scale it by the audibility factor a as $\frac{|f-c|}{g}a$. The average relative error A for a single source position s to any of n receiver positions can be given as

$$A = \frac{\sum_{s=0}^{n} \frac{|f_s - c_s|}{g} a_s}{\sum_{s=0}^{n} a_s}$$

Figure 6.11 shows the value A visually. The average relative error M is computed for all positions: $\sum_{s=0}^{n} A_s/n$. This is the value shown in Figure 6.9. We note that for large maps, many responses are silent, as the sound decays before reaching distant receivers. These responses are trivially equal (i.e. empty) and are not included in any error calculations.



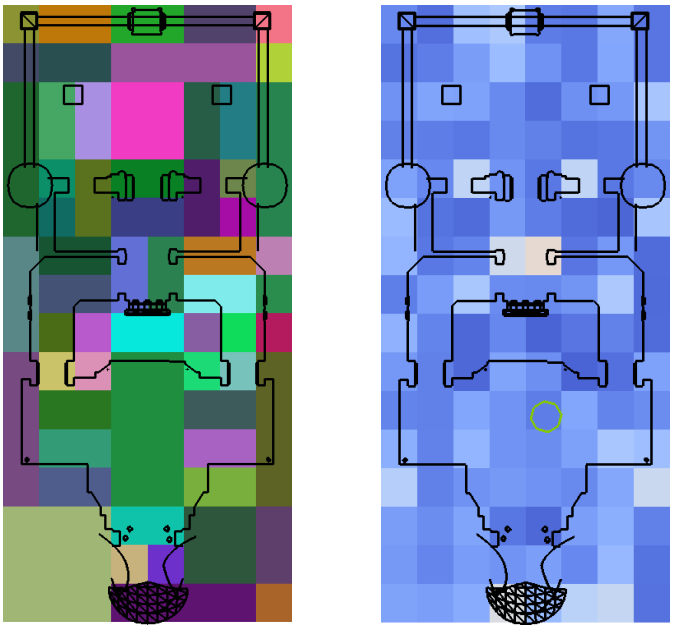
(a) (b) (c)

Figure 6.10: **Error maps:** We compute acoustic evaluation properties in the FPS scene for (a) full dataset and (b) our reduced dataset. The details of these datasets are given in Table 6.2. The difference between these datasets represents the error in our solution (c). The total energy values for the source position outlined in green are shown.

We have computed the ground-truth for the FPS Game scene using our simulator with no segmentation. The results in Figure 6.12 are the relative error for an initial sampling of 4 meters with a merge threshold S_{thr} set such that the simulated node count is 75% of the original grid sample count.

6.4.5 Error analysis for reduction algorithms

We investigated other possible reduction algorithms: naive sub-sampling, greedy flood fill reduction, and greedy sorted merge, and compared them with our adaptive schemes. Subsampling is placing the sample points in a regular grid at lower resolution sampling rates. This is the most natural way to reduce sampling in a scene. Flood fill is performed by sorting edges according to the



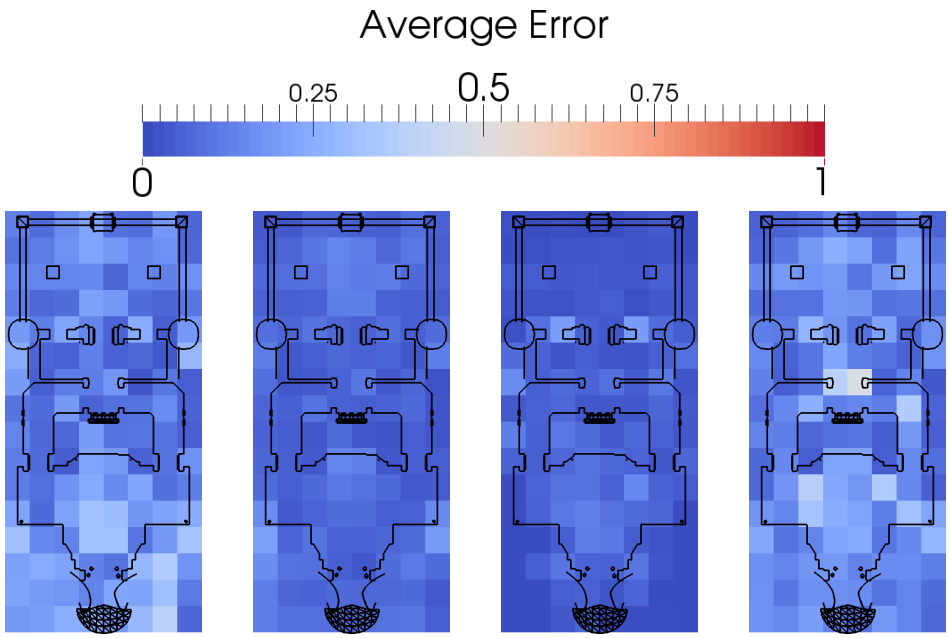
(a) (b)

Figure 6.11: **Segmentation error:** The segmentation map for the FPS scene is shown in (a), where each unique color is an acoustic region. The total energy relative error resulting from this segmentation is shown in (b) We show the source position sampled in Figure 6.10 as a green circle. The legend for (b) is the same as the legend in Figure 6.12.

minimum error, then greedily merging the lowest error nodes to form a region. Any nodes connected to the region are merged, provided the resulting error is low enough. In greedy sort merge, the edges are sorted according to the minimum error. Edges are merged in a greedy fashion until all possible edges are merged. We investigated all these methods and observed that adaptive reduction is the most effective (see Figure 6.13).

6.4.6 Metric analysis

We use four metrics in our similarity measure: distance, direction, diffuseness, and material. We weight the metrics identically since it is difficult to know which metric will perform best for a given scene. For example, the material metric may be very useful in an office environment with many different materials, but less useful in a stone cathedral where all materials share similar properties. In Figure 6.14 we show the results of reduction with some metrics disabled.



(a) (b) (c) (d)

Figure 6.12: **Error maps:** We compute the relative error in FPS scene with respect to different evaluation metric: (a) onset delay; (b) onset wave direction; (c) RT60; (d) and definition D. A wireframe of the scene is overlaid on the error maps. Red areas indicate high error. In most regions the errors in terms of onset delay, RT60 and definition are low. A few locations result in high values of the onset direction relative error.

6.5 Implementation and performance

We describe our implementation and highlight the performance of our algorithm on game scenes and virtual worlds. We have implemented our precompute system on a PC with an NVIDIA GPU with 480 threads and 1.5 GB video memory. The precompute system CPU is an Intel Xeon with 48 threads and 256 GB main memory. The runtime system is a common Intel Core i7 CPU at 3.3 GHz with 16GB memory. The cube sampler and similarity metrics were implemented in OpenGL with GLSL shaders to compute the diffusion and integration steps. The precomputation algorithm is implemented in C++ with OpenMP threading.

6.5.1 Similarity and reduction cost

Our similarity metrics are computed on cube faces using the GPU. We used GLSL shaders to sample the surface distance and normals. The distance is computed in the object space and recorded along with the normals. The diffusion metric is computed in one pass based on the gradient for each query using the edge-convolution kernel. The integration step is performed using a simple texture

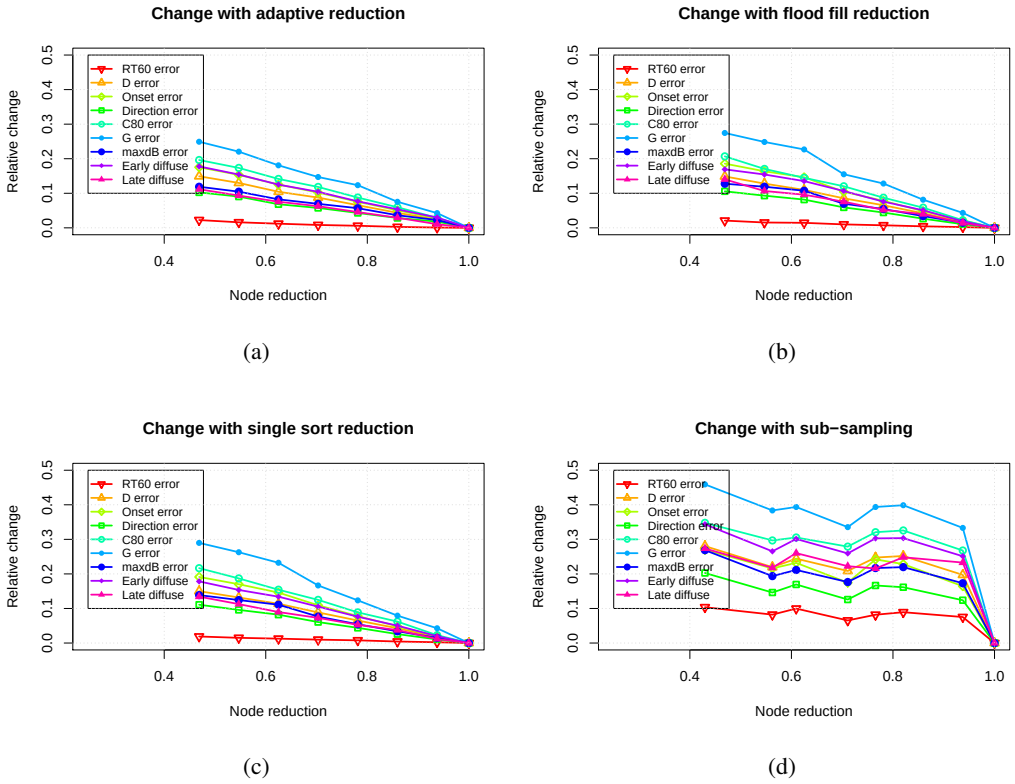


Figure 6.13: **Error values for different reduction algorithms corresponding to different acoustic evaluation metrics:** The (a) adaptive algorithm performs better than any other; the other node placement algorithms guided by our signature, (b) flood fill and (c) sorted merge, perform much better than naive (d) subsampling. These error plots demonstrate the benefit of using our geometric acoustic similarity criteria along with the adaptive scheme as compared to other approaches. For example, the error reduction over sub-sampling algorithms can be large, as compared to that over flood-fill and single sort reduction. Due to high compute cost, these results only include specular and diffraction responses.

value summation kernel and scaled by the kernel size. In our benchmarks, we found that a kernel size of 4 pixels gives the best performance on our GPU. In the reduction and similarity comparison step, LOS queries are performed using a fast CPU-based BVH ray tracer. All edge-reduction and region computations are performed using custom data structures backed by STL containers. The reduction process has a low computational overhead. Given the appropriate error thresholds, we perform sampling and segmentation on our benchmark scenes. The time to compute the similarity measure at each sample point as well as the time cost to eliminate similar sample points are shown in Table 6.2. The results in this section are for specular and diffuse responses only. Diffuse reflection

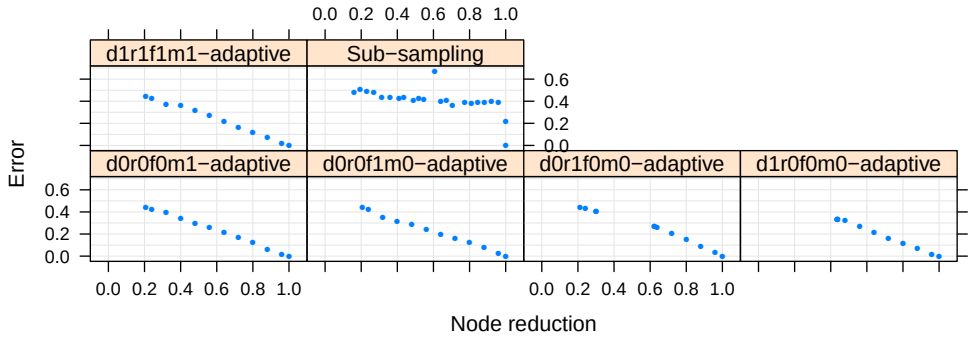


Figure 6.14: **Individual metric results:** We show the reduction results when only a single metric is enabled on the Small City scene. The chart title indicates which metrics are enabled with a 0 or 1: distance (d), direction (r), diffuseness (f), and material (m). The top row shows the results with all metrics enabled and the results from naive subsampling. The vertical axis is maximum error of any measured acoustic property, while the horizontal axis is the nodes remaining from the original count. Due to high compute cost, these results only include specular and diffraction responses.

requires four times as many rays to be traced in order to model per frequency effects. Table 6.3 shows the cost of simulating diffuse reflections.

Scene	Full grid	Reduced			Time improvement
		Cubemap	Segmentation	Trace	
Simple outdoor	15.14s	1.3s	1ms	6.4s	2.0x
FPS game	141.0s	2.0s	1ms	41.6s	3.2x
Small city	8.8m	8.6s	3ms	4.6m	1.9x
Large city	68m*	26s	69ms	14.4m	4.7x
Canyon	52d*	1.24h	3s	174h	7.17x

Table 6.2: **Precomputation time cost:** Region segmentation using cube maps allows a significant reduction in precomputation time. The full grid data is generated based on the grid size given in Table 1 for each benchmark. Due to the high time and space cost, times marked with an * are based on partial simulation.

6.5.2 Precomputation: time and storage

By segmenting the scene and reducing the number of sample points during precomputation, the time and storage costs can be considerably reduced. Our sound propagation simulator is based on discrete ray tracing, accelerated by efficient BVH trees. The ray tracer is heavily multi-threaded, but not SIMD optimized, as high-order reflections are very incoherent. The simulator computes the response at each receiver in the scene from a single source in one simulation cycle. This results in n simulation cycles for n acoustic regions or source positions.

	With diffuse		Without	
	full	reduced	full	reduced
FPS time	554s	223s	141s	42s
FPS space	2.0MB	0.5MB	1.3MB	0.3MB
Small city time	33.5m	14.9m	8.8m	4.6m
Small city space	39MB	3.3MB	39MB	3MB

Table 6.3: **Diffuse reflection cost:** Diffuse reflections requires more simulation time and slightly more storage space. Reduction results are for 75% node reduction.

We store our acoustic data as quantized decay filters. A four second reverberation decay profile can be efficiently encoded using 200 blocks, each containing 6 bytes of information, for a total of 1.2 KB. Our reduced sample count and efficient storage structure significantly reduce the storage overhead, as compared to grid based methods. Table 6.4 compares our storage overhead ("Reduced") to standard grid based methods ("Full").

We design our storage algorithms to reduce both time and space costs. For n insertions, the time cost is $O(n \log n)$, which is dominated by map operations and sorting the index vector. Additionally, data is stored sparsely when possible. Sparse data, response quantization, and node reduction result in highly compressed acoustic databases (Table 6.5).

	Full grid		Reduced		Storage
Scene	inserts	storage	inserts	storage	improvement
Simple outdoor	5.5k	1.6MB	1k	0.3MB	5.3x
FPS game	7.3k	1.3MB	1.6k	0.3MB	4.3x
Small city	229k	39MB	7.3k	3MB	13x
Large city	*	*	722k	173MB	*
Canyon	*	*	11.6B	20.8GB	*

Table 6.4: **Storage cost:** We compare storage cost of our reduction algorithm to a full grid, both stored in our efficient sparse data structure.We observe significant improvement for large scenes. Due to the high time and space cost, times marked with an * could not be computed.

Method	Error	Space savings
Value quantization	∼2%	75%
No duplicates	None	∼50%
Node reduction	∼20%	∼80%
Total	∼ 22%	∼99.99%

Table 6.5: **Compression:** We combine several algorithms to produce highly compressed scenes

6.5.3 Runtime cost

The direct sound and early reflection computations are performed on the server for a viable VoIP application. A single server would be expected to support thousands of clients (e.g. Axon supports

8K clients on an 8-core server [Dolby, 2012]). The results for the maximum number of streams that can be mixed on a single core in realtime are given in Table 6.6. By implementing our runtime mixing using efficient GPU kernels, we can further scale the approach to thousands of streams in realtime using high-end GPUs [Tsingos et al., 2011].

Scene	# Mixes	Convolve (ms)	Setup (ms)
Walkway	764	16.2	5.0
Game FPS	731	13.7	4.8
Small city	800	15	5.6
Large city	692	14.8	5.2
Canyon	684	14.2	5.8

Table 6.6: **Max realtime mixes per core:** This table shows mixing costs for 500ms of decay data per stream. The setup time includes data structure access and LOS traces. The realtime mixing step in our system is performed in less than 20ms.

In traditional spatial VoIP systems, occluded voices that are not in direct LOS would fade into silence, and therefore would not be considered for mixing; this indirectly improves the computation and bandwidth scalability of the system. In contrast, we take into account occluded voices for clustering and mixing as they might be audible due to early scattering and reverberation effects. This can decrease the forwarding-to-mixing ratio and requires more runtime processing.

We measure the performance of our algorithm by changing the number of audio streams that can be rendered simultaneously. For our purposes, an audio stream is the data that must be mixed for each source to each receiver. For example, if a single source is within the range of three receivers, three streams would need to be mixed. If source or receiver is moving, the decays must be interpolated at region boundaries. This results in an extra multipy-add for each impulse, decreasing performance by a slight degree. All numbers shown in Table 6.6 are for moving sources and receivers.

In this section, we compare our approach with other precomputation methods and highlight some limitations.

6.5.4 Comparison

Many precomputation techniques have been proposed for interactive sound propagation and rendering. We compare some features of our approach to other precomputation methods in Table 6.7.

Numerical propagation: Wave-based precomputation approaches [Mehra et al., 2013, Raghuvanshi et al., 2010] are more accurate than GA methods and can model higher order diffraction and scattering effects. However, the complexity of this simulation increases as a fourth power of

Algorithm	Ours	PART	Wave-grid	IS-gradient	DP-Cache	Reverb-graph
Mem. use	<Low	Low	High	Low	Medium	Low
Convolution	>Realtime	Interactive	Realtime	>Realtime	Realtime	Realtime
Regions	Acoustic	Spatial	None	Cell+Portal	Cell+Portal	Cell+Portal
Directionality	All samples	1st order	1st order	All samples	1st order	1st order
Decomposition	Automatic	Automatic	Automatic	Manual	Manual	Manual

Table 6.7: **Comparison:** We compare some features of our approach with other precomputation methods, including PART [Siltanen et al., 2009], Wave-grid [Raghuvanshi et al., 2010], IS gradient [Tsingos, 2009], DP Cache [Foale and Vamplew, 2007], and Reverb graph [Stavrakis et al., 2008].

frequency and is a linear function of the volume of the virtual world. As a result, these techniques are currently limited to small indoor scenes and are used only for low frequencies (e.g. less than 1000 Hz) . Although our current implementation is based on GA, we believe that our similarity-measure and decomposition method could also be combined with wave-based solvers [Raghuvanshi et al., 2010].

Cell and portals: Many games and interactive applications use cell-and-portal scene decompositions, which can be utilized to precompute higher-order reflections of sound between moving sources and listeners using ray tracing [Foale and Vamplew, 2007, Stavrakis et al., 2008, Tsingos, 2009]. These approaches typically store IRs sampled at a single position for each cell and/or portal encountered along the paths between the source and the receiver. However, these approaches require significant manual intervention to define regions and portals, making them impractical for large-scale environments. Defining cells and portals suitable for acoustic rendering can often be unintuitive, especially for large outdoor scenes. In contrast, our approach can automatically partition any large environment into acoustic regions.

Frequency and time decomposition: Some techniques perform frequency-domain precomputation based on the acoustic rendering equation [Siltanen et al., 2009], which limits them to static sources. Recently, [Antani et al., 2012a] extended the approach by precomputing acoustic transfer operators. However, this approach can only handle a few moving sources, since it performs ray tracing to compute early reflections at runtime. As a result, the runtime overhead can be very high for a large number of sources and receivers. Furthermore, its storage overhead is about $50 - 100$X higher than our approach.

CHAPTER 7: CONCLUSION

We have presented novel algorithms for interactive sound propagation. Our GA algorithms all support diffraction simulation by the UTD. This allows sound to propagation out of line-of-sight as expected. Our methods map well to parallel architectures and mobile devices.

RESound combines frustum tracing, ray tracing, and statistic reverberation for a simulation that supports early specular reflection, early diffraction, early diffuse reflection, and statistical late field. Tracing random rays is more difficult on GPUs, so our guided algorithm focuses on fast specular simulation on GPUs. Both algorithms support fully dynamic scenes with moving sources, receivers, and objects.

These algorithms are far too compute intensive to be interactive on mobile devices. For such devices, we rely on a precompute scheme. Our method samples the scene, finds the most important acoustic propagation points, then simulates and stores the related data. At runtime, a database of response signals can be queried and sent to the client device. This places the compute burden on the network server, while only requiring the client to receive the data over the network.

In this section, we discuss the limitations of our methods and possible future work.

7.1 Diffraction

The presented edge diffraction method enhances frustum tracing by allowing diffraction contribution paths that can be computed and auralized. Our resulting system computes direct contributions, specular reflections and edge diffraction using ray-frustum tracing. To the best of our knowledge, this is the first edge diffraction method that results in near-interactive performance in complex scenes with dynamic objects with reasonable GA accuracy. The overall performance of the system increases as the order of reflections and diffraction increases. We have observed comparable results with the beam tracing method on the Bell Labs Box, and our algorithm can generate plausible acoustic simulation results on complex benchmarks.

7.1.1 Limitations

Since the UTD is used to calculate the diffraction coefficients, the underlying restrictions of the UTD model naturally apply to our algorithm. UTD is a high frequency approximation, and is not very accurate for low frequency diffraction. Moreover, UTD assumes that the diffracting edge is of infinite length and the source and receiver are far from the edge (relative to wavelength). These restrictions are discussed in further detail in [Kouyoumjian and Pathak, 1974].

Frustum tracing can be regarded as an approximation of beam tracing and introduces some additional limitations. While it is a volumetric technique, some paths are missed since the frusta cannot be subdivided infinitely to represent the scene primitives. The level of subdivision is controlled either using a uniform global parameter [Lauterbach et al., 2007b] or an adaptive subdivision scheme [Chandak et al., 2008]. This error can be avoided entirely by extending frustum tracing to compute accurate object-space visibility and perform accurate geometric acoustics [Chandak et al., 2009].

Since our method approximates the diffraction cone with many linear frusta, this subdivision can result in over-estimation or under-estimation in the final frustum volume. This results in the back projected path having a slightly over or under estimation of the path length. As discussed in 3.2.2, this error is reduced as the subdivision limit is increased.

Due to the approximation of computing only shadow region diffraction, a discontinuity exists at the shadow boundary. While this can certainly be resolved by computing the entire diffraction region exactly, this greatly increases the number of frusta that must be propagated. As such, we resolve the discontinuity by using approximate normalized attenuation values, resulting in a few dB of signal error near the boundary.

As previously mentioned, frusta may have difficulty finding paths that lie parallel to the corner ray of the frustum. This may cause some important contribution paths to be unaccounted for. These paths could be found by including impostor structures at edges such that impostor will be encountered during propagation. Other potential approaches would be 2D ray intersection of the scene primitives in the plane of the wedge or propagating a special frustum along the plane that contains the region of the plane.

7.2 RESound

We have presented an interactive sound rendering system for dynamic virtual environments. RESound uses GA methods to compute the propagation paths. We use a ray-based underlying representation that is used to compute specular reflections, diffuse reflections, and edge diffraction. We also use statistical late reverberation estimation techniques and present an interactive audio rendering algorithm for dynamic virtual environments. We believe RESound is the first interactive system that can generate plausible sound rendering in complex, dynamic virtual environments.

7.2.1 Limitations

RESound has several limitations. The accuracy of our algorithm is limited by the use of underlying GA algorithms. In practice, GA is only accurate for higher frequencies. Moreover, the accuracy of our frustum-tracing reflection and diffraction varies as a function of maximum subdivision. Our diffraction formulation is based on the UTD and assumes that the edge lengths are significantly larger than the wavelength. Also, frustum tracing based diffraction also is limited in the types of diffraction paths that can be found. Our approach for computing the diffuse IR is subject to statistical error [Embrechts, 2000] that must be overcome with dense sampling. In terms of audio rendering, we impose physical restrictions on the motion of the source, listener, and scene objects to generate an artifact free rendering.

7.3 Guided visibility

We have presented a new auralization algorithm for interactive scenes. Our guidance algorithm reduces visibility and path cost while maintaining accuracy. Moreover, we exploit the computational power of GPUs to perform the visibility computations in parallel and achieve significant performance improvement over prior GA methods for the same number of contributions. In practice, we are able to compute most of the contribution paths to the receiver in game like scenes with thousands of triangles. Overall, we are able to generate plausible audio rendering in dynamic game-like scenes at $8 - 30$ fps on current PCs. Moreover, our approach aligns well with the current technology trends and its accuracy and performance would improve with the increased parallelism available in the GPUs.

7.3.1 Limitations

Overall, our approach is designed to exploit the computational power of GPUs to perform interactive visibility queries. The overall goal is accurate auralization, but our approach can result in the following errors:

1. Visibility errors: The accuracy of the visible surface or secondary image source computation algorithm is governed by the number of ray samples and relative configuration of the image sources. Our algorithm can miss some secondary sources or propagation paths and is more accurate for the first few orders of reflections and diffraction. Figure 5.12 compares the found paths on two scenes of varying complexity.

2. Limited path count: Our algorithm uses the number of valid paths found as input to the guiding algorithm. If the initial visibility sampling is too low, it is possible that some paths will not be found. Since the guiding algorithm cannot account for these unknown paths, it cannot increase sampling density to find them.

3. Diffraction errors: Our formulation is a variation of the UTD method and its application to finite edges can result in errors. Moreover, our system only simulates one order of approximate diffraction paths and it is possible that we miss some of the diffraction contributions due to sampling errors. It is also possible that the found paths will have slight geometric error.

4. Acoustic response errors: The overall GA method is a high frequency approximation and may not be applicable to scenes with very small and detailed features. Furthermore, our system does not model diffuse reflections or high order specular reflection and diffraction. Other complementary algorithms [Antani et al., 2012b] could allow diffuse responses to be included with little cost.

5. Sound rendering artifacts: Our approach tends to reduce audio artifacts, but cannot eliminate them. Since our rendering algorithm uses the image sources computed by the propagation algorithm, any inaccuracy in image source computation affects its performance. In particular, if a high number of image sources appear or disappear between successive frames, we may observe artifacts.

The governing factor in the accuracy and performance of our approach is the number of ray samples that are cast in a single simulation frame. This directly impacts visibility accuracy and indirectly affects validation accuracy. As we use a higher number of visibility samples, errors are reduced (see Figure 5.12). This aligns well with the current technology trends as the performance of

future GPUs will improve in terms of ray tracing throughput [Aila and Laine, 2009]. Another factor that governs the accuracy is the size of the triangles. Most GA methods are applicable to models where the size of the features or triangles is comparable (or larger) than the audible wavelengths. Moreover, as the size of the primitives increase, it improves the coherence of the multi-viewpoint ray casting algorithm and makes it possible to trace a higher number of ray samples per second.

7.4 Massive scenes

We present a new approach to generating environmental voice reverberation in large virtual worlds. Our algorithm scales with the size of the model and computes early and late acoustic responses complete with diffraction and reverberation effects. We use a local geometrical similarity metric to efficiently sample key positions and create zones with similar acoustical properties. Due to our sample reduction algorithm and efficient storage structures, we observe more than an order of magnitude improvement in precomputation time and storage overhead. We demonstrate results on kilometer-sized virtual worlds with a large number of sources and receivers. In practice, our approach can generate plausible environmental audio effects and is targeted towards gaming and virtual environments.

7.4.1 Limitations

Our approach introduces several approximations with respect to reverberation computation. Our method makes heavy use of quantization and all sound data is relatively low frequency (16 KHz). While this can create viable voice output, it is not suited for higher-frequency data, such as most music. While it would be possible to accommodate higher frequency responses, storage size and compression would be affected. Within regions, the direct path can be rendered with occlusion and distance attenuation. However, the decay data has been precomputed and cannot be adjusted within region. We attempt to mitigate this issue by restricting the region sizes and apply a heuristic scaling to the early response. Nonetheless, the interpolation within regions is not physically based and may lead to incorrect decay responses.

We render integrated decay profiles reconstructed with a random phase, rather than the original impulse response. This reduces the accuracy with which the early reflections can be rendered. In

particular, flutter echoes might not be captured by our approach unless the number of sub-bands is increased. Since we use GA simulation, it cannot accurately simulate all wave effects.

Since our similarity measure computation is a heuristic based on scene geometric representation, which only takes into account first-order reflections or responses, this formulation may fail in some cases to accurately estimate the late responses. In scenes where the depth variance is large, the early response time cannot be reliably estimated from the local geometric representation. Moreover, in scenes with high depth complexity or occlusion, the diffraction paths contribute significantly towards the early responses, and our approach may not work well in such scenes. Finally, our reduction metrics can be overly conservative in some scenes, resulting in less time and storage benefits.

7.5 Future work

There are many avenues for future work. Stronger validation of the found diffraction paths could be conducted to reduce or eliminate the slight path error. Computing more conservative diffraction frusta would reduce the possibility of missing important paths. Conducting a conservative region visibility test from each encountered diffracting edge would make it possible to find paths suitable for auralization with the BTM (Biot-Tolstoy-Medwin) method. While this may have a large computational cost, it would allow more accurate simulation of lower frequencies and shorter diffracting edges. Another way to improve the performance is to reduce the number of diffraction frusta generated. This would allow more time to perform higher levels of subdivision or reflections. There has been some work in the area of diffraction culling [Calamia and Svensson, 2005]. This would reduce the number of insignificant frusta created. We would also like to create conservative diffraction frusta that always contain the full diffraction cone.

We would like to further analyze the accuracy of our RESound approach. It is possible to further improve the accuracy of edge diffraction by using the BTM formulation, as opposed to UTD. Similarly, the accuracy of diffuse reflections can be improved based on better sampling methods. We would also like to investigate ways to improve late reverberation estimation by using decay curves appropriate for the local environment. Many interactive applications such as games or VR need $30 - 60$ Hz update rates and we may need faster methods to achieve such a performance on current commodity hardware. We are also investigating using frustum tracing for very accurate GA

simulations [Chandak et al., 2009]. Finally, we would like to use RESound for other applications such as tele-conferencing and design of sound-based user interfaces.

We would like to extend our guided multiview algorithm to work on scenes with a high number of sound sources based on clustering methods or perceptual rendering algorithms. Online analysis of the propagation cost coefficients may allow for improved guidance algorithms. A preprocess sampling pass could be used select more appropriate initial sampling and radius values for better guidance. Furthermore, we would like to perform perceptual evaluation of our system and perform user studies. Since the ray tracing algorithm can also be used to perform diffuse reflections, it may be possible to adapt our algorithms for rendering diffuse scattering effects. During tracing, some form of dynamic resampling may improve the accuracy of our algorithm. We also want to investigate the use of multi-view tracing with other software, such as NVIDIA's OptiX [1] engine or acoustic precomputation methods [Antani et al., 2012a]. Finally, we would like to integrate our auralization system with other interactive applications and evaluate its performance.

Our current precompute method is limited to static scenes; we plan to investigate techniques to update the acoustic response based on dynamic objects using precomputed filters. We also would like to investigate possible perceptual reduction techniques to further reduce the number of samples that need to be stored. We need to perform more validation and error analysis, especially for large outdoor scenes. Our current error analysis is based on standard ISO-3382 parameters, which were primarily designed for room acoustics. We believe that a user study to evaluate the accuracy of reduced sampling strategies and plausibility would be useful.

[1] http://www.nvidia.com/object/optix.html

BIBLIOGRAPHY

3D Working Group of Interactive 3D Audio SIG (1998). Interactive 3d audio rendering guidelines, level 1.0. http://www.iasig.org/pubs/3dl1v1.pdf. 25

3D Working Group of Interactive 3D Audio SIG (1999). Interactive 3d audio rendering guidelines, level 2.0. http://www.iasig.org/pubs/3dl2v1a.pdf. 24

Aila, T. and Laine, S. (2009). Understanding the efficiency of ray traversal on gpus. In *Proceedings of High-Performance Graphics*, pages 145–149. 63, 111

Algazi, V., Duda, R., and Thompson, D. (2001). The CIPIC HRTF Database. In *IEEE ASSP Workshop on Applications of Signal Processing to Audio and Acoustics*. 25, 51

Allen, J. B. and Berkley, D. A. (1979). Image method for efficiently simulating small-room acoustics. *The Journal of the Acoustical Society of America*, 65(4):943–950. 18

Andō, Y. (1998). *Architectural Acoustics: Blending Sound Sources, Sound Fields, and Listeners*. Modern Acoustics and Signal Processing Series. Springer Verlag. 96

Antani, L., Chandak, A., Savioja, L., and Manocha, D. (2012a). Interactive sound propagation using compact acoustic transfer operators. *ACM Transactions on Graphics*, 31(1):7:1–7:12. 22, 27, 106, 113

Antani, L., Chandak, A., Taylor, M., and Manocha, D. (2012b). Direct-to-indirect acoustic radiance transfer. *IEEE Transactions on Visualization and Computer Graphics*, 18(2):261 – 269. 22, 110

Antani, L., Chandak, A., Taylor, M., and Manocha, D. (2012c). Efficient finite-edge diffraction using conservative from-region visibility. *Applied Acoustics*, 73:218–233. 23, 24

Antonacci, F., Foco, M., Sarti, A., and Tubaro, S. (2004). Fast modeling of acoustic reflections and diffraction in complex environments using visibility diagrams. In *Proceedings of 12th European Signal Processing Conference (EUSIPCO '04)*, pages 1773–1776. 24

Arenberg, J. (1988). Re: Ray/triangle intersection with barycentric coordinates. *Ray Tracing News*, 1(11). http://tog.acm.org/resources/RTNews/html/rtnews5b.html#art3. 67

Bailey, R. and Brumitt, B. (2010). Method and system for automatically generating world environment reverberation from a game geometry. Technical report, U.S. Patent Application US 2010/0008513 A1. 25

Benesty, J., Gaensler, T., and Eneroth, P. (2000). Multi-channel sound, acoustic echo cancellation, and multi-channel time-domain adaptive filtering. In *Acoustic Signal Processing for Telecommunication*, pages 101–120. Kluwer Academic Publishers. 26

Berkhout, A. J., de Vries, D., and Vogel, P. (1993). Acoustic control by wave field synthesis. *The Journal of the Acoustical Society of America*, 93(5). 25

Bertram, M., Deines, E., Mohring, J., Jegorovs, J., and Hagen, H. (2005). Phonon tracing for auralization and visualization of sound. In *Proceedings of IEEE Visualization*, pages 151–158. 22

Biot, M. A. and Tolstoy, I. (1957). Formulation of wave propagation in infinite media by normal coordinates with an application to diffraction. *Journal of the Acoustical Society of America*, 29(3):381–391. 23

Blauert, J. (1997). *Spatial Hearing : The Psychophysics of Human Sound Localization*. M.I.T. Press, Cambridge, MA. 1

Borish, J. (1984). Extension of the image model to arbitrary polyhedra. *The Journal of the Acoustical Society of America*, 75:1827–1836. 18

Botteldooren, D. (1994). Acoustical finite-difference time-domain simulation in a quasi-cartesian grid. *The Journal of the Acoustical Society of America*, 95(5):2313–2319. 17

Boustead, P. and Safaei, F. (2004). Comparison of delivery architectures for immersive audio in crowded networked games. In *Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '04, pages 22–27, New York, NY, USA. ACM. 27

Calamia, P., Markham, B., and Svensson, U. P. (2008). Diffraction culling for virtual-acoustic simulations. *Acta Acustica united with Acustica, Special Issue on Virtual Acoustics*, 94:907–920. 68

Calamia, P. T. and Svensson, U. P. (2005). Edge subdivision for fast diffraction calculations. In *Proc. 2005 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA 2005)*, pages 187–190. 23, 112

Calamia, P. T. and Svensson, U. P. (2007). Fast time-domain edge-diffraction calculations for interactive acoustic simulations. *EURASIP J. Appl. Signal Process.*, 2007(1):186–186. 23, 82

CATT (2002). *CATT-Acoustic User Manual*. CATT, Sweden, v8.0 edition. http://www.catt.se/. 77

Chandak, A., Antani, L., Taylor, M., and Manocha, D. (2009). Fastv: From-point visibility culling on complex models. *Eurographics Symposium on Rendering*. 22, 27, 58, 77, 108, 113

Chandak, A., Lauterbach, C., Taylor, M., Ren, Z., and Manocha, D. (2008). AD-Frustum: Adaptive Frustum Tracing for Interactive Sound Propagation. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1707–1722. xiii, 22, 28, 42, 44, 54, 56, 108

Christensen, C. L. (2009). *ODEON Room Acoustics Program User Manual*. ODEON A/S, Denmark, 10.1 edition. http://www.odeon.dk/. 77

Cook, P. R. (2002). *Real Sound Synthesis for Interactive Applications*. A. K. Peters. 16

Cowan, B. and Kapralos, B. (2011). Gpu-based acoustical diffraction modeling for complex virtual reality and gaming environments. In *Audio Engineering Society Conference: 41st International Conference: Audio for Games*. 24

Dalenbäck, B. (1996). Room acoustic prediction based on a unified treatment of diffuse and specular reflection. *The Journal of the Acoustical Society of America*, 100(2):899–909. 23, 77

Dalenbäck, B.-I., Kleiner, M., and Svensson, P. (1994). A Macroscopic View of Diffuse Reflection. *Journal of the Audio Engineering Society (JAES)*, 42(10):793–807. 23, 42

Dance, S. M. and Shield, B. M. (1994). The effect on prediction accuracy of reducing the number of rays in a ray tracing model. *Inter-Noise94*, 3(1):2127–2130. 74

Dolby (2012). Axon surround sound chat for gamers. http://www.dolby.com/us/en/consumer/technology/gaming/dolby-axon.html. 26, 105

Duyne, S. V. and Smith, J. O. (1993). The 2-d digital waveguide mesh. In *Applications of Signal Processing to Audio and Acoustics, 1993. Final Program and Paper Summaries., 1993 IEEE Workshop on*, pages 177–180. 17

Embrechts, J. J. (2000). Broad spectrum diffusion model for room acoustics ray-tracing algorithms. *The Journal of the Acoustical Society of America*, 107(4):2068–2081. 24, 46, 55, 109

Eyring, C. F. (1930). Reverberation time in "dead" rooms. *The Journal of the Acoustical Society of America*, 1(2A):217–241. 24, 42, 47

Farina, A. (1995). RAMSETE - a new Pyramid Tracer for medium and large scale acoustic problems. In *Proceedings of EURO-NOISE*. 77

Foale, C. and Vamplew, P. (2007). Portal-based sound propagation for first-person computer games. In *Proceedings of the 4th Australasian conference on Interactive entertainment*, IE '07, pages 9:1–9:8. xii, 106

Franck, A. (2008). Efficient Algorithms and Structures for Fractional Delay Filtering Based on Lagrange Interpolation. *J. Audio Eng. Soc.*, 56(12):1036–1056. 72

Funkhouser, T., Carlbom, I., Elko, G., Pingali, G., Sondhi, M., and West, J. (1998). A beam tracing approach to acoustic modeling for interactive virtual environments. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 21–32, New York, NY, USA. ACM. 20, 27, 58, 77

Funkhouser, T., Tsingos, N., and Jot, J.-M. (2003). Survey of Methods for Modeling Sound Propagation in Interactive Virtual Environment Systems. *Presence and Teleoperation*. 41

Funkhouser, T. A., Min, P., and Carlbom, I. (1999). Real-time acoustic modeling for distributed virtual environments. In *Proc. of ACM SIGGRAPH*, pages 365–374. 27

Gerzon, M. A. (1973). Periphony: With-height sound reproduction. *J. Audio Eng. Soc*, 21(1):2–10. 25

Gibbs, M., Wadley, G., and Benda, P. (2006). Proximity-based chat in a first person shooter: using a novel voice communication system for online play. In *Proceedings of the 3rd Australasian conference on Interactive entertainment*, IE '06, pages 96–102, Murdoch University, Australia, Australia. Murdoch University. 26

Goode, B. (2002). Voice over internet protocol (voip). In *Proceedings of the IEEE*, pages 1495 – 1517. 26

Greene, N. (1986). Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, 6(11). 80

Halloran, J. (2009). It's talk, but not as we know it: Using voip to communicate in war games. In *Proceedings of the 2009 Conference in Games and Virtual Worlds for Serious Applications*, VS-GAMES '09, pages 133–140. 26

Heckbert, P. S. and Hanrahan, P. (1984). Beam tracing polygonal objects. In *Proc. of ACM SIGGRAPH*, pages 119–127. 20

Hodgson, M. (1990). Evidence of diffuse surface reflection in rooms. *The Journal of the Acoustical Society of America*, 88(S1):S185–S185. 24, 55

Hollier, M. P., Rimell, A. N., and Burraston, D. (1997). Spatial audio technology for telepresence. *BT Technology Journal*, 15(4):33 – 41. 26

ISO 3382 (2009). Measurement of room acoustic parameters. 47, 95, 96, 97

Jot, J.-M. (1999). Real-time spatial processing of sounds for music, multimedia and interactive human-computer interfaces. *Multimedia Systems*, 7(1):55–69. 24

Kapralos, B., Jenkin, M., and Milios, E. (2004). Acoustic Modeling Utilizing an Acoustic Version of Phonon Mapping. In *Proc. of IEEE Workshop on HAVE*. 23, 55

Kinsler, L. E., Frey, A. R., Coppens, A. B., and Sanders, J. V. (1999). *Fundamentals of Acoustics*. John Wiley and Sons, Inc. 1

Kleiner, M., Dalenbck, B.-I., and Svensson, P. (1991). Auralization-an overview. In *Audio Engineering Society Convention 91*. 17

Kouyoumjian, R. G. and Pathak, P. H. (1974). A uniform geometrical theory of diffraction for an edge in a perfectly conducting surface. *Proc. of IEEE*, 62:1448–1461. 23, 46, 69, 108

Krokstad, A., Strom, S., and Sorsdal, S. (1968). Calculating the acoustical room response by the use of a ray tracing technique. *Journal of Sound and Vibration*, 8(1):118–125. 21, 75

Kurze, U. J. (1974). Noise reduction by barriers. *The Journal of the Acoustical Society of America*, 55(3):504–518. 77

Kuttruff, H. (2007). *Acoustics*. Routledge. 24, 95, 96, 97, 98

Laakso, T. I., Valimaki, V., Karjalainen, M., and Laine, U. K. (1996). Splitting the unit delay [fir/all pass filters design]. *IEEE Signal Processing Magazine*, 13(1):30–60. 72

Laine, S., Siltanen, S., Lokki, T., and Savioja, L. (2009). Accelerated beam tracing algorithm. *Applied Acoustic*, 70(1):172–181. 20, 77

Larcher, V., Warusfel, O., Jot, J.-M., and Guyard, J. (2000). Study and comparison of efficient methods for 3-d audio spatialization based on linear decomposition of hrtf data. In *Audio Engineering Society 108th Convention preprints*, page preprint no. 5097. 25

Lauterbach, C., Chandak, A., and Manocha, D. (2007a). Adaptive sampling for frustum-based sound propagation in complex and dynamic environments. In *Proceedings of the 19th International Congress on Acoustics*. 22

Lauterbach, C., Chandak, A., and Manocha, D. (2007b). Interactive sound propagation in dynamic scenes using frustum tracing. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1672–1679. 28, 34, 42, 44, 108

Lauterbach, C., Garland, M., Sengupta, S., Luebke, D., and Manocha, D. (2009). Fast bvh construction on gpus. In *Proc. Eurographics '09*. 13, 63

Lauterbach, C., Yoon, S., Tuft, D., and Manocha, D. (2006). RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs. *IEEE Symposium on Interactive Ray Tracing*. 42, 44, 52

Lenhert, H. (1993). Systematic errors of the ray-tracing algoirthm. *Applied Acoustics*, 38:207–221. 21, 74, 75

Lentz, T., Schröder, D., Vorländer, M., and Assenmacher, I. (2007). Virtual reality system with integrated sound field simulation and reproduction. *EURASIP J. Appl. Signal Process.*, 2007:187–187. 77

Markopoulou, A., Tobagi, F., and Karam, M. (2002). Assessment of voip quality over internet backbones. In *IEEE INFOCOM 2002*, pages 150 – 159. 26

Medwin, H. (1981). Shadowing by finite noise barriers. *Journal of the Acoustical Society of America*, 69(4):1060–1064. 23

Mehra, R., Raghuvanshi, N., Antani, L., Chandak, A., Curtis, S., and Manocha, D. (2013). Wave-based sound propagation in large open scenes using an equivalent source formulation. *ACM Trans. on Graphics*, 32(2):19:1–19:13. 18, 27, 105

Mehra, R., Raghuvanshi, N., Savioja, L., Lin, M. C., and Manocha, D. (2012). An efficient gpu-based time domain solver for the acoustic wave equation. *Applied Acoustics*, 73:83–94. 18

Merimaa, J. and Pulkki, V. (2004). Spatial impulse response rendering. *Proc. of the 7th Intl. Conf. on Digital Audio Effects (DAFX'04)*. 89

Merimaa, J. and Pulkki, V. (2005). Spatial impulse response rendering i: Analysis and synthesis. *J. Audio Eng. Soc*, 53(12):1115–1127. 25

Moeck, T., Bonneel, N., Tsingos, N., Drettakis, G., Viaud-Delmon, I., and Alloza, D. (2007). Progressive perceptual audio rendering of complex scenes. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 189–196, New York, NY, USA. ACM. 25, 75

O'Brien, J. F., Shen, C., and Gatchalian, C. M. (2002). Synthesizing sounds from rigid-body simulations. In *The ACM SIGGRAPH 2002 Symposium on Computer Animation*, pages 175–181. ACM Press. 16

Ondet, A. M. and Barbry, J. L. (1989). Modeling of sound propagation in fitted workshops using ray tracing. *The Journal of the Acoustical Society of America*, 85(2):787–796. 21

Parker, S. G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., McAllister, D., McGuire, M., Morley, K., Robison, A., and Stich, M. (2010). Optix: A general purpose ray tracing engine. *ACM Transactions on Graphics*. 63

Pierce, A. D. (1974). Diffraction of sound around corners and over wide barriers. *The Journal of the Acoustical Society of America*, 55(5):941–955. 77

Pope, J., Creasey, D., and Chalmers, A. (1999). Realtime room acoustics using ambisonics. *Proc. of the AES 16th Intl. Conf. on Spatial Sound Reproduction*, pages 427–435. 27

Popov, S., Gnther, J., Seidel, H.-P., and Slusallek, P. (2007). Stackless KD-Tree Traversal for High Performance GPU Ray Tracing. *Computer Graphics Forum (Proc. EUROGRAPHICS)*, 26(3):415–424. 63

Radenkovic, M., Greenhalgh, C., and Benford, S. (2002). Deployment issues for multi-user audio support in cves. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '02, pages 179–185, New York, NY, USA. ACM. 27

Raghuvanshi, N., Galoppo, N., and Lin, M. C. (2008). Accelerated wave-based acoustics simulation. In *Symposium on Solid and Physical Modeling*, pages 91–102. 18

Raghuvanshi, N. and Lin, M. C. (2006). Interactive sound synthesis for large scale environments. In *Symposium on Interactive 3D graphics and games*, pages 101–108. 16

Raghuvanshi, N., Snyder, J., Mehra, R., Lin, M., and Govindaraju, N. (2010). Precomputed wave simulation for real-time sound propagation of dynamic sources in complex scenes. In *ACM SIGGRAPH 2010 Papers*, SIGGRAPH '10, pages 68:1–68:11, New York, NY, USA. ACM. xii, 18, 27, 82, 105, 106

RAMSETE (1995). *RAMSETE User Manual*. GENESIS Software and Acoustic Consulting, Italy, version 1.0 edition. http://www.ramsete.com/. 77

Safaei, F. (2005). Dice: Internet delivery of immersive voice communication for crowded virtual spaces. In *Proceedings of the 2005 IEEE Conference 2005 on Virtual Reality*, VR '05, pages 35–41, Washington, DC, USA. IEEE Computer Society. 27

Sallnäs, E.-L. (2005). Effects of communication mode on social presence, virtual presence, and performance in collaborative virtual environments. *Presence: Teleoper. Virtual Environ.*, 14(4):434–449. 26

Savioja, L. (2010). Real-time 3D finite-difference time-domain simulation of low- and mid-frequency room acoustics. In *Proc. Int. Conf. Digital Audio Effects*, Graz, Austria. 18

Savioja, L., Huopaniemi, J., Lokki, T., and Väänänen, R. (1999). Creating interactive virtual acoustic environments. *Journal of the Audio Engineering Society (JAES)*, 47(9):675–705. 49, 71

Savioja, L., Lokki, T., and Huopaniemi, J. (2002). Auralization applying the parametric room acoustic modeling technique - the diva auralization system. In *International Conference on Auditory Display*. 25, 56, 71, 73

Schissler, C. and Manocha, D. (2011). Gsound: Interactive sound propagation for games. *AES International Conference on Audio for Games*. 24

Schröder, D. and Lentz, T. (2006). Real-Time Processing of Image Sources Using Binary Space Partitioning. *Journal of the Audio Engineering Society (JAES)*, 54(7/8):604–619. 20, 77

Schröder, D. and Pohl, A. (2009). Real-time Hybrid Simulation Method Including Edge Diffraction. In *EAA Auralization Symposium*, Espoo, Finland. 24

Schroeder, M. R. (1965). New method of measuring reverberation time. *The Journal of the Acoustical Society of America*, 37:409. 97

Shlager, K. and Schneider, J. (1995). A selective survey of the finite-difference time-domain literature. *Antennas and Propagation Magazine, IEEE*, 37(4):39–57. 17

Shoemake, K. (1998). Pluecker coordinate tutorial. *Ray Tracing News*, 11(1). 43

Siltanen, S., Lokki, T., Kiminki, S., and Savioja, L. (2007). The room acoustic rendering equation. *The Journal of the Acoustical Society of America*, 122(3):1624–1635. 22, 79

Siltanen, S., Lokki, T., and Savioja, L. (2009). Frequency domain acoustic radiance transfer for real-time auralization. *Acta Acustica united with Acustica*, 95:106–117(12). xii, 22, 25, 27, 71, 106

Stavrakis, E., Tsingos, N., and Calamia, P. (2008). Topological sound propagation with reverberation graphs. *Acta Acustica/Acustica - the Journal of the European Acoustics Association*, 94:921–932. xii, 106

Summers, J. E., Torres, R. R., and Shimizu, Y. (2004). Statistical-acoustics models of energy decay in systems of coupled rooms and their relation to geometrical acoustics. *The Journal of the Acoustical Society of America*, 116(2):958–969. 48

Svensson, P. (2008). The early history of ray tracing in room acoustics. In Svensson, P., editor, *Reflections on sound: In honour of Professor Emeritus Asbjørn Krokstad*. Norwegian University of Science and Technology. 22

Svensson, U. P., Fred, R. I., and Vanderkooy, J. (1999). An analytic secondary source model of edge diffraction impulse responses . *Acoustical Society of America Journal*, 106:2331–2344. 23

Taylor, M., Chandak, A., Antani, L., and Manocha, D. (2009a). Resound: interactive sound rendering for dynamic virtual environments. In *MM '09: Proceedings of the seventeen ACM international conference on Multimedia*, pages 271–280, New York, NY, USA. ACM. 25, 66, 77

Taylor, M., Chandak, A., Mo, Q., Lauterbach, C., Schissler, C., and Manocha, D. (2012). Guided multiview ray tracing for fast auralization. *IEEE Transactions on Visualization and Computer Graphics*, 18:1797–1810. 27, 82, 88

Taylor, M., Chandak, A., Ren, Z., Lauterbach, C., and Manocha, D. (2009b). Fast Edge-Diffraction for Sound Propagation in Complex Virtual Environments. In *EAA Auralization Symposium*, Espoo, Finland. 24, 45, 46, 55

Tsingos, N. (2001). A versatile software architecture for virtual audio simulations. In *International Conference on Auditory Display (ICAD)*, Espoo, Finland. 25, 51, 71, 75

Tsingos, N. (2009). Pre-computing geometry-based reverberation effects for games. *35th AES Conference on Audio for Games*. xii, 27, 89, 106

Tsingos, N., Carlbom, I., Elbo, G., Kubli, R., and Funkhouser, T. (2002). Validation of acoustical simulations in the "bell labs box". *IEEE Computer Graphics and Applications*, 22(4):28–37. xiv, 24, 34, 35

Tsingos, N., Dachsbacher, C., Lefebvre, S., and Dellepiane, M. (2007). Instant sound scattering. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)*. 25

Tsingos, N., Funkhouser, T., Ngan, A., and Carlbom, I. (2001). Modeling acoustics in virtual environments using the uniform theory of diffraction. In *Proc. of ACM SIGGRAPH*, pages 545–552. 24, 34, 46, 56

Tsingos, N., Gallo, E., and Drettakis, G. (2003). Perceptual audio rendering of complex virtual environments. Technical Report RR-4734, INRIA, REVES/INRIA Sophia-Antipolis. 51

Tsingos, N., Gallo, E., and Drettakis, G. (2004). Perceptual audio rendering of complex virtual environments. *ACM Trans. Graph.*, 23(3):249–258. 25, 71, 75, 77

Tsingos, N., Jiang, W., and Williams, I. (2011). Using programmable graphics hardware for acoustics and audio rendering. *Journal of Audio Engineering Society*, 59(9):628–646. 93, 105

Välimäki, V. (1995). *Discrete-Time Modeling of Acoustic Tubes Using Fractional Delay Filters*. PhD thesis, Helsinki University of Technology. 72

van den Doel, K. (1998). *Sound Synthesis for Virtual Reality and Computer Games*. PhD thesis, University of British Columbia. 16

van den Doel, K., Kry, P. G., and Pai, D. K. (2001). Foleyautomatic: physically-based sound effects for interactive simulation and animation. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 537–544, New York, NY, USA. ACM Press. 16

Vorländer, M. (1989). Simulation of the transient and steady-state sound propagation in rooms using a new combined ray-tracing/image-source algorithm. *The Journal of the Acoustical Society of America*, 86(1):172–178. 22, 75, 88

Wadley, G., Gibbs, M., and Benda, P. (2007). Speaking in character: using voice-over-ip to communicate within mmorpgs. In *Proceedings of the 4th Australasian conference on Interactive entertainment*, IE '07, pages 24:1–24:8. 26

Wald, I. (2004). *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Computer Graphics Group, Saarland University. 42

Wand, M. and Straßer, W. (2004). Multi-resolution sound rendering. In *SPBG'04 Symposium on Point - Based Graphics 2004*, pages 3–11. 25, 51

Wenzel, E., Miller, J., and Abel, J. (2000). A software-based system for interactive spatial sound synthesis. In *International Conference on Auditory Display (ICAD)*, Atlanta, GA. 25, 51, 71

West, J., Blauert, J., and MacLean, D. (1992). Teleconferencing system using head-related signals. *Applied Acoustics*, 36(3-4):327 – 333. 26

Williams, D., Caplan, S., and Xiong, L. (2007). Can You Hear Me Now? The Impact of Voice in an Online Gaming Community. *Human Communication Research*, 33(4):427–449. 26

Wise, D. K. and Bristow-Johnson, R. (1999). Performance of Low-Order Polynomial Interpolators in the Presence of Oversampled Input. In *Audio Engineering Society Convention 107*. 72

Xiangyang, Z., Ke'an, C., and Jincai, S. (2003). On the accuracy of the ray-tracing algorithms based on various sound receiver models. *Applied Acoustics*, 64(4):433 – 441. 21, 74

Yang, L. N. and Shield, B. M. (2000). Development of a ray tracing computer model for the prediction of the sound field in long enclosures. *Journal of Sound and Vibration*, 229(1):133 – 146. 74

Zhou, K., Hou, Q., Wang, R., and Guo, B. (2008). Real-time kd-tree construction on graphics hardware. In *Proc. SIGGRAPH Asia*. 63

Zimmermann, R. and Liang, K. (2008). Spatialized audio streaming for networked virtual environments. In *Proceedings of the 16th ACM international conference on Multimedia*, MM '08, pages 299–308, New York, NY, USA. ACM. 27