# Efficient Geometric Sound Propagation Using Visibility Culling

Anish Chandak

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
2011

Approved by:

Dinesh Manocha

Gary Bishop

Ming C. Lin

Sorin Mitran

Marc Niethammer

# Abstract

**Anish Chandak: Efficient Geometric Sound Propagation Using Visibility Culling.**
**(Under the direction of Dinesh Manocha.)**

Simulating propagation of sound can improve the sense of realism in interactive applications such as video games and can lead to better designs in engineering applications such as architectural acoustics. In this thesis, we present geometric sound propagation techniques which are faster than prior methods and map well to upcoming parallel multi-core CPUs. We model specular reflections by using the image-source method and model finite-edge diffraction by using the well-known Biot-Tolstoy-Medwin (BTM) model. We accelerate the computation of specular reflections by applying novel visibility algorithms, FastV and AD-Frustum, which compute visibility from a point. We accelerate finite-edge diffraction modeling by applying a novel visibility algorithm which computes visibility from a region.

Our visibility algorithms are based on frustum tracing and exploit recent advances in fast ray-hierarchy intersections, data-parallel computations, and scalable, multi-core algorithms. The AD-Frustum algorithm adapts its computation to the scene complexity and allows small errors in computing specular reflection paths for higher computational efficiency. FastV and our visibility algorithm from a region are general, object-space, conservative visibility algorithms that together significantly reduce the number of image sources compared to other techniques while preserving the same accuracy. Our geometric propagation algorithms are an order of magnitude faster than prior approaches for modeling specular reflections and two to ten times faster for modeling finite-edge diffraction. Our algorithms are interactive, scale almost linearly on multi-core CPUs, and can handle large, complex, and dynamic scenes. We also compare the accuracy of our sound propagation algorithms with other methods.

Once sound propagation is performed, it is desirable to listen to the propagated sound in interactive and engineering applications. We can generate smooth, artifact-free output audio signals by applying efficient audio-processing algorithms. We also present the first efficient

audio-processing algorithm for scenarios with simultaneously moving source and moving receiver (MS-MR) which incurs less than 25% overhead compared to static source and moving receiver (SS-MR) or moving source and static receiver (MS-SR) scenario.

To my parents, Shyam Chandak and Vijaylaxmi Chandak.

# Acknowledgments

The past five years as a graduate student in the Department of Computer Science at UNC-Chapel Hill have been very rewarding, personally and professionally. I would like to thank many people who have been an important part of my journey through these times. First and foremost, I would like to thank my advisor Prof. Dinesh Manocha for his encouragement, guidance, and support over the past five years. I am grateful for the freedom he gave me to work on challenging research projects. I would also like to thank my committee members: Prof. Ming C. Lin, for her guidance and generously sponsoring lunches and dinners during paper deadlines; Prof. Gary Bishop for his insightful comments on my research and passionate discussions on many other applications related to sound; Prof. Sorin Mitran for educating me about numerical methods and other technical discussions; and Prof. Marc Niethammer for his feedback and insightful comments on my thesis.

I have had the privilege to work with some really smart collaborators on various research projects. I would like to thank them for teaching me so much and being patient with me. In particular, I would like to thank Christian Lauterbach, Nikunj Raghuvanshi, Micah Taylor, Zhimin Ren, Lakulish Antani, and Ravish Mehra. I would also like to thank members of GAMMA group for their support and feedback. I would like to thank support staff at the Department of Computer Science at UNC-Chapel Hill for their excellent assistance. In particular, I would like to thanks Missy Wood, Janet Jones, Kelli Gaskill, and Mike Stone.

I would like to thank many good friends for making the past five years in Chapel Hill a lot of fun. In particular, Sasa Junuzovic, Liangjun Zhang, Jason Sewall, Rahul Narain, Sachin Patil, Vishal Verma, Lei Wei, and Qi Mo. I would like to especially thank my dear friends, the Taylors (Micah, Christine, and Charlotte). Lastly, I would like to thank my parents, Shyam and Vijaylaxmi Chandak, my brother and sister-in-law, Manish and Shital Chandak, my sister, Surbhi Maheshwari, and my nephew and niece, Divit and Ananya Chandak, for their unconditional love and support.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Over the last few decades, the fidelity of visual rendering in interactive applications such as video games, virtual reality (VR) training, scientific visualization, and computer-aided design (CAD) has improved considerably [Tatarchuk, 2009]. The availability of high performance, low-cost commodity graphics processors (GPUs) makes it possible to render complex scenes at interactive rates on current laptops and desktops. However, in order to improve the immersive experience and utility of interactive applications, it is also important to develop interactive technologies that exploit other senses, especially the sense of hearing [Anderson and Casey, 1997].

The field of *sound rendering* deals with developing efficient algorithms and tools for computer-generated sounds. *Sound rendering* refers to input modeling, like synthesis of sound from a source, propagation, i.e. modeling propagation of sound waves from a source to a receiver, and audio processing, i.e. generating the output audio signal taking propagation and audio from the sound source into account. It is similar to the concept of graphics rendering in computer graphics for computer generated images. For example, propagation of light is similar to propagation of sound, though their speeds and wavelengths are quite different. Therefore, many advances made in the field of global illumination and ray tracing to model propagation of light for interactive and offline computations can be adapted to propagation of sound. There is no generally accepted

definition of sound rendering as only recently has there been an active interest in these algorithms for interactive applications [Manocha et al., 2009]. There is a term often used in architectural acoustics similar to sound rendering; Auralization. *Auralization* is the process of rendering audible, by physical or mathematical modeling, the sound field of a source in a space, in such a way as to simulate the binaural listening experience at a given position in the modeled space [Kleiner et al., 1993]. Auralization insists simulating binaural listening experience and does not incorporate sound synthesis techniques. Further, given the recent trend towards highly parallel multi-core CPUs and many core GPUs it is possible to implement efficient algorithms for sound rendering.

In this chapter, we give examples of different applications of sound rendering, an overview of our problem statement, and a brief overview of prior approaches. Next, we discuss various issues and challenges in designing efficient techniques for sound propagation and audio processing as well as the main contributions of the thesis.

## 1.1 Applications

There are many applications that could benefit from sound rendering, especially sound propagation and audio processing. In this section, we give a brief overview of some of these applications. The challenges imposed by these applications on sound rendering algorithms are discussed in Section 1.4.

### Video Games

Video games today use advanced graphics rendering techniques like: real-time radiosity, indirect illumination, volumetric shadow maps, and other advanced lighting and shading techniques [Tatarchuk, 2009]. There is an increasing demand for similar advanced techniques for sound rendering to enhance the immersive experience in video games [Haraldsen, 2010]. For example, consider a stealth game like *Thief: Deadly Shad-*


(a) (b) (c)

Figure 1.1: Sound rendering in video games. *(a)* Stealth Games*: In stealth games, the player must avoid detection and use stealth to evade or ambush antagonists. Thief: Deadly Shadows [EIDOS, 2011] is a popular stealth game where protagonist and master thief Garrett aims to steal his way through the City using stealth. Modeling sound propagation can help Garrett, the master thief evade the antagonists by using reflected and diffracted sound to detect their approach. (b)* Multiplayer Games*: Voice chat is an integral part of multiplayer games like Counter Strike [CSS, 2011]. It allows players on the same team to talk to each other, plan their strategies, and gain strategic advantage. Modeling the environment and position of the players during the voice chat can improve the overall realism and experience. Dolby recently released Axon [AXON, 2011], a voice chat middleware which modifies the voice chat between two players by taking into account reflection and diffraction of sound between the players. (c)* First Person Shooter Games*: First person shooter games, like Call of Duty [COD, 2011], are a very popular genre of games where the player experiences the action through the eyes of the protagonist. In first person shooter games sound rendering can help the player to detect the enemies, improve realism, and enhance the overall gaming experience of the players.*

*ows* [EIDOS, 2011] (see Figure 1.1(a)). In this game, the protagonist and master thief Garrett must use stealth to evade or ambush antagonists to steal his way through the game. Modeling sound propagation can increase the realism in the game as Garret can evade the antagonists by listening to the sounds of their footsteps and other noises as the sound is reflected and diffracted in the scene. Spatialized voice chat in multiplayer games like *Counter Strike: Source* [CSS, 2011] (see Figure 1.1(b)) and dynamic environment effects in first person shooter games like *Call of Duty: Black Ops* [COD, 2011] (see Figure 1.1(c)) can significantly enhance the gaming experience as the sound cues in the game match the real-world experience of the players.

Currently, video games use simple sound propagation models. In most games only the direct sound from a source to a receiver is modeled. Occlusion, diffraction, and reflection of sound are approximated by applying artist created filters [Kastbauer, 2010]. These filters are created manually by artists, instead by using a sound propagation simulation. For example, to model echoes in a cathedral, artists may use a filter which sounds like a cathedral, but only a single filter is created for the whole space and it does not vary with relative positions of the sound source and the receiver. There are many reasons for these limitations. Firstly, interactive performance is required in video games and typically only a fraction ($< 10\%$) of the total CPU budget is devoted to sound rendering. Thus, interactive sound rendering algorithms which stay within the allotted CPU budget need to be developed. Secondly, video games typically use geometric models ranging from small rooms to big cities consisting of tens to a few hundreds of thousands of triangles. Such complex models are challenging to handle efficiently for current sound rendering algorithms [Funkhouser et al., 2003]. Thirdly, the sound sources, receiver, and geometric objects in games can move, i.e. they are dynamic. Thus, sound rendering algorithms need to efficiently handle complex, dynamic, and general scenes. Further, it might be acceptable to trade-off accuracy for higher computational performance in video games.

## Virtual Reality Training

Virtual reality (VR) simulators can provide a cost effective way of training. Sound rendering is important in such VR simulators as it provides important sound cues, like the direction of incoming sound and the size of the environment. VR simulators have been used for training [White et al., 2008], therapy [Gerardi et al., 2008], tourism [McGookin et al., 2009, Pielot et al., 2007], and learning [Melzer et al., 2010]. In tele-collaboration systems, spatial reproduction of sound is necessary to achieve realistic immersion [Avendano, 2004]. Sound rendering can also provide environmental context by modeling acoustics in absence of visual cues, provide a feeling of human emotions,

or set the mood [Mann, 2008]. For example, reverberation provides a sense of warmth or immersion in the environment. An example of a VR simulator, where sound rendering can significantly improve its effectiveness, is one used for treating soldiers suffering from post-traumatic stress disorder (PTSD) [Wilson, 2010] (see Figure 1.2(a)). An accurate reproduction of the sound field is important to recreate a believable war experience in the training environment so that the soldiers suffering from PTSD can experience the intense war-like environment in a controlled setting [Hughes et al., 2006]. Other VR simulators where sound rendering could significantly improve their utility are combat training simulators like, the Future Immersive Training Environment (FITE) [Pellerin, 2011], and training simulators for the visually impaired, like the HOMERE system [Lecuyer et al., 2003, Torres-Gil et al., 2010]. These VR simulators can be significantly enhanced by incorporating advanced sound rendering techniques.

Like games, VR training simulators require interactive sound propagation. Also, the output audio should have minimal artifacts due to dynamic sources, receiver, and scene geometry. However, the typical resource limitations of games can be ignored and more computational resources can be devoted to sound rendering in these applications. Accurate sound rendering may be required as it is important to faithfully reproduce the sounds in the VR environment.

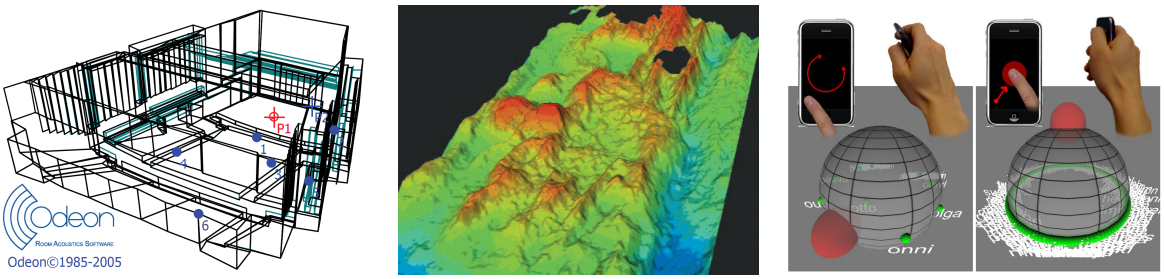
## Architectural Acoustics

Acoustics modeling significantly improves the building design process by detecting flaws in the acoustics of the building during the design phase and ensures that acoustics standards and regulations are met before the construction phase. This can result in significant cost savings as reconstruction and alterations in the post-construction phase can be avoided. Sound rendering for architectural acoustics can also be used to design acoustic walkthroughs of the architecture for demonstration purposes or develop telepresence systems for distributed musical performances [Cooperstock, 2011].


(a) (b) (c)

Figure 1.2: Sound rendering in virtual reality (VR) applications. *(a)* Therapy Systems*: U.S. Air Force Senior Airman Joseph Vargas, uses the Virtual Iraq program at Malcolm Grow Medical Center's Virtually Better training site at Andrews Air Force Base, Md., June 25, 2009. The 79th Medical Wing is one of eight wings that uses this new technology to treat patients suffering from post-traumatic stress disorder (PTSD). U.S. Air Force photo by SRA Renae Kleckner [Wilson, 2010]. Recreating a realistic war experience for a patient is critical for such applications to be useful for PTSD treatment. An accurate and efficient modeling of the sound field during a virtual war simulation increases the presence of the patient in the virtual war simulation [Hughes et al., 2006] and hence, significantly improves the technology to treat PTSD patients. (b)* Combat Training Systems*: US Marines walk through a Future Immersive Training Environment (FITE) scenario. This Defense Department program provides 3D immersive technologies to help Marines and soldiers make better, faster decisions on the ground. Sound rendering can improve the system by providing sound cues which are an integral part of the decision making process on the ground. (c)* Training Systems for the Visually Impaired*: HOMERE system: a multimodal system for visually impaired people to explore and navigate inside virtual environments. It has an auditory feedback for the ambient atmosphere and for other specific events. Sound rendering can significantly improve such a system by providing auditory feedback as the sound bounces around in the virtual environment.*

Accurate sound propagation methods which can efficiently model specular reflections, diffuse reflections, and diffraction are required for acoustic modeling. A key component in these applications is accurate computation of sound propagation paths, which takes into account the knowledge of sound sources, listener locations, the 3D model of the environment, and material absorption and scattering properties. However, many existing commercial room acoustic modeling systems [Christensen, 2009] (see Figure 1.3(a)) do not handle diffraction or do not use state-of-the-art ray tracing algorithms to model specular and diffuse reflections. Only a few research systems can handle diffraction [Funkhouser et al., 2004], but are limited to special types of models (and static scenes),



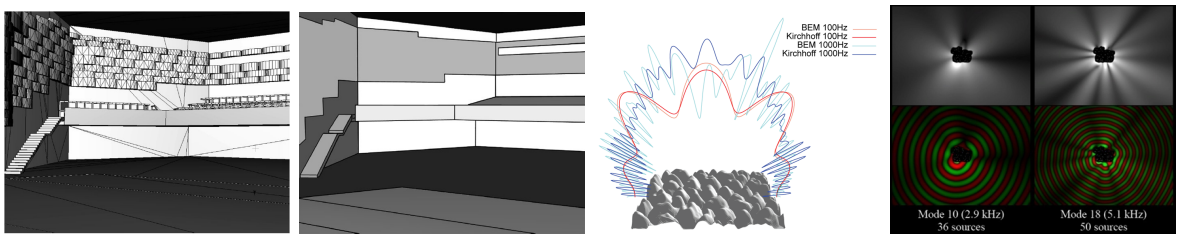

(a) (b) (c)

Figure 1.3: Sound rendering in other applications. *(a)* Architectural Acoustics*: Acoustic modeling is the prediction of the sound field in a particular 3D design of a building, e.g. concert halls, offices, and class rooms. The clarity of speech is very important in class rooms for students to be able to listen and understand the lectures. Likewise, the reverberation of music in concert halls is important for a great listening experience. Standard guidelines for acoustics in such spaces has been defined [ANSI/ASA S12.60-2002, 2002]. ODEON is a popular acoustic modeling software [Christensen, 2009]. Sound rendering can model and correct the acoustics during the design phase and could help save cost by preventing expensive post-construction fixes. (b)* Multimodal Scientific Visualization*: A display of bathymetric data using three dimensions and color; additional data is provided by generating forces and sounds as a user explores the surface with a haptic display device. (Figure courtesy NASA/UH Virtual Environments Research Institute.) (c)* Auditory Interfaces*: A method for browsing eyes-free auditory menus [Kajastila and Lokki, 2010]. Auditory menus are spoken one by one; the user has the ability to jump to the next item and to stop the current playback.*

e.g. architectural models represented as cells and portals, 2.5D urban models, or scenes with large convex primitives. Further, prior acoustics modeling tools do not exploit commodity processors in terms of multiple cores.

## Multimodal Scientific Visualization and Auditory Interfaces

There is an increasing interest in applying sound to analyze data and develop better user interfaces. In multimodal visualization, multiple modalities like vision, sound, haptic, etc. are used to visualize data. Using multiple modalities allows to counter information overload [Brewster, 1997]. Figure 1.3(b) shows an example of a multimodal visualization of geoscientific data [Smith, 1993, Harding et al., 2000]. The authors map a single data variable to a pitch, instrument, and a duration. For low data values, a combination of




(a) (b) (c)

Figure 1.4: Input modeling for sound rendering. *(a) Highly detailed CAD model for graphics modeling (left) and simplified CAD model for geometric acoustic modeling (right) [Vorländer, 2008]. (b) A transfer function computed for a detailed object to encode scattering due to the object. The object can be replaced with a simpler object and a transfer function [Tsingos et al., 2007]. (c) Modeling of sound synthesis and radiation pattern due to a sound source [James et al., 2006].*

low-pitch, bass type instrument, and long duration is used. And for high data values, a combination of high pitch, soprano type instrument, and a short duration is used. Some training may be required for a user to be able to associate a data value to the audio output from the multimodal visualization system [Loftin, 2003]. Auditory interfaces use audio as a key component to design better user interfaces. Figure 1.3(c) show an example of an eye-free method for accessing *auditory menus* [Kajastila and Lokki, 2010]. Auditory displays can augment graphical displays and provide the user with an enhanced sense of presence. Efficient sound rendering tools can be used to develop interesting multimodal visualization techniques and auditory interfaces.

## 1.2 Sound Rendering

In this section we review different components of sound rendering and prior state-of-the art in sound rendering.

### 1.2.1 Input Modeling

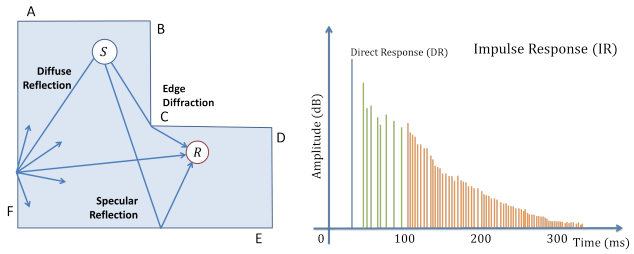
Sound rendering applications specify various inputs like position, orientation, and input audio or vibrational modes of a source; position, orientation, and transfer function of

a receiver; and input geometric models with their acoustic material properties. Significant work has been done in the past few years on interactive modeling of sound source vibrations [Raghuvanshi and Lin, 2006], and modeling their transfer functions [James et al., 2006] (see Figure 1.4(c)). They are collectively studied under *sound synthesis* techniques. Many sound rendering systems focussing on sound propagation assume that the sound source is a point source with a uniform radiation pattern. The input audio to the source is anechoically recorded audio samples. In terms of a receiver, a point receiver with a transfer function, specifically a generalized Head Related Transfer Function (HRTF) [Algazi et al., 2001], which models the scattering of sound due to receivers' head, torso, and shoulders has been used. This allows modeling of binaural audio, i.e. audio for left and right ear, at the receiver.

The CAD model and acoustic material properties are also provided as input by the sound rendering application. Many applications like video games use complex geometric models for visual rendering and the same model is provided as an input for sound rendering. 3D models with only relevant geometric details [Vorländer, 2008] are needed by the sound rendering systems (see Figure 1.4(a)). However, very limited work has been done to automatically simplify a complex model for visual rendering to a simplified model for sound rendering [Siltanen et al., 2008]. Further, the acoustic material properties also need to be specified along with acoustic geometry. Recently, complex transfer functions [Tsingos et al., 2007] (see Figure 1.4(b)) similar to those used in computer graphics have been proposed for sound rendering. However, limited data is available on acoustic transfer functions of acoustic materials and simplified models.

### 1.2.2 Sound Propagation

The propagation of sound in a medium is governed by the *acoustic wave equation*, a second-order partial differential equation [Kuttruff, 1991]. Several methods exist that directly solve the wave equation using numerical methods [Kleiner et al., 1993] and ac-




(a) (b)

Figure 1.5: Example of sound propagation. *(a) Input scene with a point source (S) and a point receiver (R) and input scene model. Example of specular reflection, diffuse reflections, and diffraction from source to the receiver. (b) Propagation of sound in a constant linear medium can be characterized by computing response at the receiver due to an impulse sound emitted at the source, called impulse response (IR).*

curately model sound propagation in a scene. However, despite recent advances in numerical methods to solve the acoustic propagation equation [Raghuvanshi et al., 2009], these methods can take hours to days to solve the wave equation and can be too slow for interactive applications. Further, they are restricted to modeling only low frequencies of sound waves as the computational overhead of numerical methods increase proportional to $f^4$, where $f$ is the maximum frequency modeled by the numerical methods [Botteldooren, 1995].

Most sound propagation techniques used in practical applications model the acoustic effects of an environment using linearly propagating rays. These *geometric acoustics* (GA) techniques are not as accurate as numerical methods in terms of solving the wave equation, and cannot easily model all kinds of propagation effects, but they can simulate early specular reflections at real-time rates [Funkhouser et al., 2003]. They provide approximate solutions to the wave equation for high-frequency sound sources. Figure 1.5(a) shows example of specular reflection, diffuse reflections, and edge diffraction using rays from a source to a receiver. Broadly, geometric acoustics algorithms can be divided into *pressure-based* and *energy-based methods*.

Energy-based methods are typically used to model diffuse reflections and propagation effects when interference of sound waves is not important. The *room acous-*

*tic rendering equation* [Siltanen et al., 2007] is an integral equation which generalizes many existing energy-based geometric techniques. Existing energy-based methods include: ray tracing [Krokstad et al., 1968], phonon tracing [Kapralos, 2006], and radiosity [Nosal et al., 2004].

Pressure-based methods model specular reflections and edge diffraction, and are essentially variations of the image-source method [Allen and Berkley, 1979]. Ray tracing [Vorländer, 1989], beam tracing [Funkhouser et al., 2004], and several other techniques [Funkhouser et al., 2003] have been proposed to accelerate the image-source method for specular reflections. *Edge diffraction* is modeled by the Uniform Theory of Diffraction (UTD) [Kouyoumjian and Pathak, 1974] or the Biot-Tolstoy-Medwin (BTM) model of diffraction [Medwin et al., 1982]. Propagation of sound in a constant linear medium can be characterized by computing the response at the receiver due to an impulse sound emitted at the source, called impulse response (IR). In image source method, and IR is computed from a collection of image sources by taking into account their position and direction relative to the receiver.

## Image Source Method and Visibility Tree

The image source method [Allen and Berkley, 1979] is widely used for modeling specular reflections and has been extended to model finite-edge diffraction [Pulkki et al., 2002]. Given a point source $S$ and a listener $L$, it is easy to check if a direct path exists from $S$ to $L$. This is a ray shooting problem. The basic idea behind the image source method is as follows. For a specular reflector (in our case, a triangle) $T$, a specular path $S \rightarrow T \rightarrow L$ exists if the triangle $T$ is visible from the source $S$ and the listener $L$ is visible from the *image* of $S$, created by reflecting $S$ across the plane of triangle $T$, through triangle $T$. In the absence of any visibility information, image sources need to be computed for *every* triangle in the scene. This process can be applied recursively to check for higher order specular paths from $S$ to $L$, but the complexity increases exponentially as a function of

the number of reflections. The image sources for a sound source computed this way can be encoded in a data structure called *visibility tree*.

For a given source position, this process can be accelerated as follows. Note that first-order image sources only need to be computed for triangles visible to $S$. For a first-order image source $S_1$, second-order image sources only need to be computed for the triangles that are visible to $S_1$ through $T$, and so on for higher order image sources. The same principle applies to finite-edge diffraction. The image source of a diffracting edge is the edge itself and higher order diffraction need to be computed for the edges that are visible to the diffracting edge. Thus, to accelerate the image source method, the goal should be to never compute image sources that do not contribute towards the IRs computed at the listener positions.

### 1.2.3 Audio Processing

In the audio processing step, we take the input audio played at the source and convolve it with the impulse responses (IRs), computed by the sound propagation step, to generate the output audio. To generate audio output for a left and right ear, i.e. binaural audio output, a transfer function is applied to each geometric path reaching the receiver. The transfer function could be a simple parametric left-right panning or more accurate head related transfer function (HRTF) computed by either measurements [Algazi et al., 2001] or physical simulation [Dellepiane et al., 2008] which takes scattering of sound due to the human head, torso, and shoulder into account.

One of the key challenges in audio processing for interactive sound rendering application is handling dynamic environments, corresponding to moving sources or moving objects. As the sources and receiver are moving in the scene, interpolation of output audio [Savioja et al., 1999] or interpolation of IRs by interpolating image sources need to be performed [Tsingos, 2001a]. Otherwise, discontinuities and other artifacts may appear in the final audio output. Further, to keep the computational overhead of au-

dio processing low, efficient techniques based on perceptual optimization are required [Tsingos et al., 2004, Tsingos, 2005]. Additional challenges arise when both the sound source and the receiver are moving simultaneously as for a given receiver position, sound reaches the receiver form different source positions and very limited work has been done to address this issue. We will present a technique to address this issue in Chapter 5.

## 1.3 Visibility Techniques

Visibility computation is one of the classical problems that has been studied extensively due to its importance in many fields such as computer graphics, computational geometry, and robotics. Given a scene, the goal is to compute a potentially visible set (PVS) of primitives that are visible from a single point (i.e. from-point visibility), or from any point within a given region (i.e. from-region visibility). Visibility algorithms can be classified in different ways. One way is to classify them into from-point and from-region visibility. From-point visibility is used in computer graphics for generating the final image from the eye-point based on rasterization [Theoharis et al., 2001] or ray tracing [Arvo and Kirk, 1989]. Other examples of applications of from-point visibility include hard shadow computation for point light sources. From-region visibility has been used in computer graphics for global illumination (i.e., computing the multiple bounce response of light from light sources to the camera via reflections from primitives in the 3D model), interactive walkthroughs of complex 3D models by pre-fetching a smaller set of potentially visible primitives from a region around the active camera position, soft shadow computation from area light sources, etc.

### 1.3.1 Object-Space Exact Visibility

Object-space exact visibility techniques computes exactly the PVS visible from a viewpoint or a view-region. The PVS computed by an object-space exact visibility algo-

rithm is the smallest PVS which contains all the primitives. These technique perform intersection computations at the accuracy of the original model, e.g. IEEE 64-bit double precision arithmetic. Many applications require exact visibility with object-space precision. For example, accurate computation of soft shadows due to area light source in computer graphics [Hasenfratz et al., 2003] requires the computation of exact visible area from all the points of the area light source to compute the contribution of the area light source at the point. Similarly, computing hard shadows due to a point light source requires accurate computation of visible portions of primitives from the point light source [Lloyd et al., 2006] or aliasing artifacts may appear. Many approaches have been proposed for exact from-point [Heckbert and Hanrahan, 1984, Overbeck et al., 2007, Nirenstein, 2003] and from-region visibility [Durand et al., 1996, Nirenstein et al., 2002]. These techniques are discussed in Section 2.1.1.

### 1.3.2 Object-Space Conservative Visibility

Object-space conservative visibility techniques compute a PVS which contains at least all the primitives visible from the view-point or the view-region, but may contain extra primitives which may not be visible. Conservative visibility algorithms are preferred for their computational efficiency and simplicity over exact algorithms. Two widely used but highly conservative visibility techniques are *view-frustum culling* and *back-face culling* [Foley et al., 1993]. They are used to trivially remove some of the hidden primitives. In view-frustum culling, the primitives completely outside the view-frustum are marked hidden; and in back-face culling, the primitives which are facing away from the view-point or view-region are marked as hidden. The choice between a conservative or an exact object-space algorithm is decided by the application on the basis of the trade-off between the overhead of extra visible primitives due to the conservative algorithm versus the time overhead of the exact algorithm. Approaches for conservative from-point and from-region visibility are discussed in Section 2.1.2.

### 1.3.3 Image-space or Sample-based Visibility

Image-space visibility techniques sample a set of rays and compute a PVS that is hit by only the finite set of sampled rays. The choice of sampled rays is governed by the application. Sampling-based methods are widely used in graphics applications due to their computational efficiency and are well supported by current GPUs. Typically, during image generation, an image of a given resolution, say 1K $\times$ 1K pixels and only a constant number of rays per pixel are sampled to generate an image. Sampling based methods are extensively used in computer graphics for image generation. However, these methods can suffer from spatial and temporal aliasing issues and may require supersampling or other techniques (e.g. filters) to reduce aliasing artifacts. Ray tracing [Glassner, 1989] and z-buffer algorithm [Catmull, 1974] are popular sampling based approaches. Further details on these approaches is presented in Section 2.1.3.

## 1.4 Challenges and Goals

There are many challenges in developing a sound rendering system for the interactive applications described earlier. For instance, accurate modeling of acoustics requires numerically solving the acoustic wave equation. The numerical methods to solve this equation tend to be compute and storage intensive. As a result, fast algorithms for complex scenes mostly use geometric methods that propagate sound based on rectilinear propagation of waves and can accurately model transmission, early reflection and edge diffraction [Funkhouser et al., 2003]. We supplement them with efficient finite edge diffraction modeling. Below, we summarize a few key challenges imposed by interactive sound rendering applications like video games and accurate sound rendering applications like architectural acoustics modeling.

- **Interactive performance**: Video games and VR simulators demand interactive performance. Therefore, sound rendering system should be able to do sound prop-

agation as well audio processing at interactive rates, i.e. 10-30 frames per second (FPS). For example, the sound propagation step should perform 2-3 orders of early reflections at runtime at interactive rates.

- **Performance and accuracy trade-off**: Not all applications require high accuracy. Even for the applications that do, like architectural acoustics, it is important that a less accurate but fast simulation can be performed during the design phase to avoid long waits. For games and VR simulators, it might be possible to reduce the accuracy of the simulation and still achieve the same perceptual effects. For example, perceptually, it is important to model the early orders of reflections more accurately than late orders of reflections [Funkhouser et al., 2003].

- **Accurate acoustics modeling**: Accurate acoustic simulation is critical in many applications like architectural acoustics and outdoor acoustics modeling. However, existing techniques either take too much time to model diffraction and higher order reflections or do not model such effects all together. For example, the high-end acoustics software ODEON only models limited first order diffraction [Christensen, 2009] and it is used by many architects and acoustic consultants.

- **Complex, dynamic, and general scenes**: The 3D models used in interactive applications like games could range from small rooms to big cities consisting of tens to a few hundreds of thousands of triangles. Further, the scenes could be dynamic with moving geometry, sound sources, and receiver. For example, in an FPS game, the player and the enemies shooting at the player could both be moving. Modeling specular reflections and diffraction on such complex 3D models is challenging for existing geometric methods. Some methods can handle complex 3D models but they are only limited to static scenes [Funkhouser et al., 1998]. Thus, efficiently handling complex and dynamic scenes is a big challenge for existing geometric sound propagation methods.

- **Smooth audio output**: Due to the dynamic nature of many applications, the response from sound propagation system changes. This could lead to artifacts and discontinuities in the output audio when the input audio is filtered through the dynamic responses. For example, a moving receiver in an architectural model can at times hear clicks, pops, and other artifacts due to dynamic sources or environments. It is challenging to develop algorithms which generate minimal distortions in the final audio when reducing discontinuity artifacts. Also, techniques to reduce discontinuity artifacts should be computationally efficient.

- **Performance scalability on parallel architecture**: The recent computing systems consist of multi-core CPUs and many-core GPUs. As a result, it is important that sound rendering algorithms should scale almost linearly with the number of cores on these commodity processors. For example, the performance should double when the sound rendering system is moved from a 2-core processor to a 4-core processor, with the same cache sizes and processor clocks.

## 1.5 Thesis Statement

*Geometric sound propagation, accelerated with object-space visibility algorithms, can lead to faster and scalable sound rendering.* We primarily focus on modeling specular reflections and finite edge diffraction using geometric propagation techniques. We apply object-space, from-point visibility to accelerate specular reflections and object-space, from-region visibility to accelerate finite-edge diffraction. We develop a complete sound rendering system which uses audio signal from a point source, computes propagation from a source to a receiver, and performs audio processing to generate artifact-free output audio. Our techniques are faster than prior methods and scale with complexity and dynamism in a scene. Further, our algorithms scale almost linearly with the number of cores on parallel multi-core CPUs.

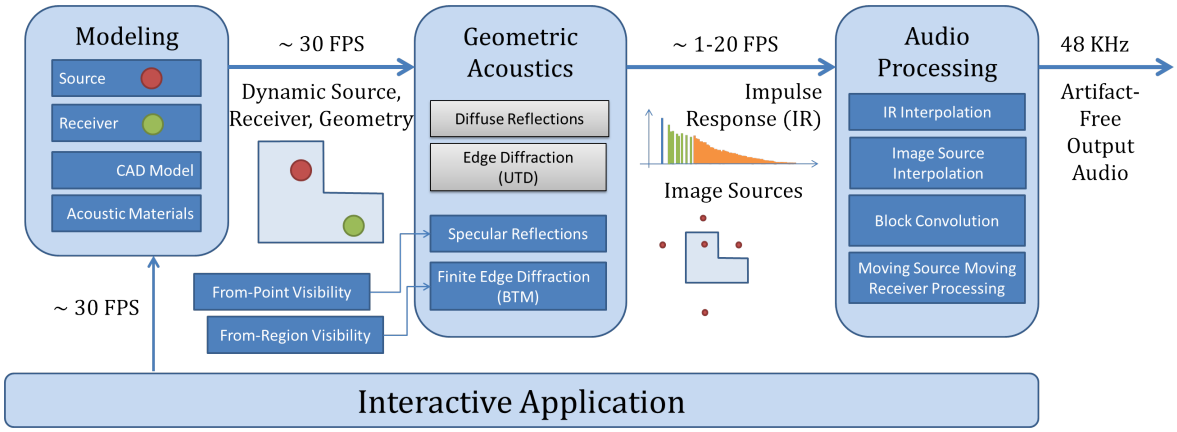



Figure 1.6: Overall sound rendering system. *The models for sound source, receiver, input CAD model, and corresponding acoustic material properties are provided by the sound rendering application. For this thesis, we assume a point source which is playing recorded audio samples, point receiver whose transfer function is modeled with generalized head related transfer function (HRTF), any general triangulated CAD model, and material properties like absorption coefficient. These inputs are passed to the geometric acoustics computation module, which computes specular reflections and finite-edge diffraction using the Biot-Tolstoy-Medwin (BTM) model. Specular reflection computation is accelerated by using from-point visibility techniques. We apply AD-Frustum technique to model specular reflections for interactive application allowing errors in computed specular paths and apply FastV technique to accurately model specular reflections for engineering applications. Finite edge diffraction based on the BTM model is accelerated using from-region visibility techniques. The output from geometric acoustics step is an impulse response (IR) for given source receiver positions. These are input to the audio processing step which produces artifact-free output audio by performing block convolution with the IR and by interpolating output audio and image sources in dynamic scenes.*

## 1.6 Main Results

In this thesis, we present efficient geometric sound propagation and audio processing algorithms. An overview of our prototype sound rendering system is presented in Figure 1.6. We highlight the connection between geometric acoustics (GA) and visibility algorithms in computer graphics and computational geometry. We develop two from-point visibility algorithms, AD-Frustum and FastV, to efficiently compute specular reflections and an object-space from-region visibility algorithm to accelerate from-edge diffraction computation (based on the Biot-Tolstoy-Medwin model of diffraction). We also develop audio processing algorithms which generate artifact-free output audio in

dynamic scenes. Finally, we compare our GA algorithms with numerical acoustics (NA) methods.

### 1.6.1 AD-Frustum: Adaptive Frustum Tracing

We present a novel volumetric tracing approach that can generate propagation paths for early specular reflections by adapting to the scene primitives. AD-Frustum approach is geared towards interactive applications and allows errors in specular paths for higher computational efficiency. Our approach is general and can handle all triangulated models. Further, our approach can also handle dynamic scenes with dynamic source, dynamic receiver, and dynamic geometry. The underlying formulation uses a simple adaptive representation that augments a 4-sided frustum [Lauterbach et al., 2007b] with a quadtree and adaptively generates sub-frusta. We exploit the adaptive frustum representation to perform fast intersection and visibility computations with scene primitives. As compared to prior algorithms for GA, our approach provides an automatic balance between accuracy and interactivity to generate plausible sound in complex scenes. Some novel aspects of our work include:

1. **AD-Frustum**: We present a simple representation to adaptively generate 4-sided frusta to accurately compute propagation paths. Each sub-frustum represents a volume corresponding to a bundle of rays. We use ray-coherence techniques from computer graphics to accelerate intersection computations. The algorithm uses an area subdivision method to compute an approximation of the visible surface for each frustum.

2. **Handling general, complex scenes**: We apply bounding volume hierarchies (BVHs) to accelerate the intersection computations with AD-Frusta in complex, dynamic scenes. We present techniques to bound the maximum subdivision within each AD-Frustum based on scene complexity and thereby control the overall ac-

curacy of propagation by computing all the important contributions.

3. **Interactive performance**: We apply our algorithm for interactive sound propagation in complex and dynamic scenes corresponding to architectural models, outdoor scenes, and game environments. In practice, our algorithm can compute early specular reflection paths for up to 4-5 reflections at 4-20 frames per second on scenes with hundreds of thousands of polygons on a multi-core PC. Our preliminary comparisons indicate that propagation based on AD-Frusta can offer considerable speedups over prior geometric propagation algorithms. We also evaluate the accuracy of our algorithm by comparing the impulse responses with a widely used implementation of an image source method, which models specular reflections accurately.

### 1.6.2 FastV: From-point Visibility Culling on Complex Models

AD-Frustum approach is geared towards interactive application and may miss specular reflection paths. This is not acceptable for engineering applications where it is is important to find all specular paths efficiently. Thus, we present a novel algorithm (FastV) for conservative, from-point visibility computation. Our approach is general and computes a potentially visible set (PVS) of scene triangles from a given view point. The main idea is to trace a high number of 4-sided volumetric frusta and efficiently compute simple connected blockers for each frustum. We use the blockers to compute a far plane and cull away the non-visible primitives. Our guiding principle is to opt for simplicity in the choice of different components, including frustum tracing, frustum-intersection tests, blocker and depth computations. The main contribution is in developing new algorithms for some of these components and combining them in an effective manner. Overall, FastV is the first practical method for visibility culling in complex 3D models due to the following reasons:

1. **Handling general, complex scenes**: Our approach is applicable to all triangulated models and does not assume any large objects or occluders. The algorithm proceeds automatically and is not susceptible to degeneracies or robustness issues.

2. **Conservative**: Our algorithm computes a conservative superset of the visible triangles at object-precision. As the frustum size decreases, the algorithm computes a tighter PVS. We have applied the algorithm to complex CAD and scanned models with millions of triangles and simple dynamic scenes. In practice, we can compute a conservative PVS, which is within a factor of $5 - 25\%$ of the exact visible set, in a fraction of a second on a 16-core PC.

3. **Efficient Visibility**: We present fast and conservative algorithms based on Plücker coordinates to perform intersection tests and blocker computations. We use hierarchies along with SIMD and multi-core capabilities to accelerate the computations. In practice, our algorithm can trace $101 - 200$K frusta per second on a single 2.93 GHz Xeon Core on complex models with millions of triangles.

4. **Efficient Sound Propagation:** We use our PVS computation algorithm to accurately compute specular reflection paths from a point sound source to a receiver. We use a two phase algorithm that first computes image sources for scene primitives in the PVS computed for primary (or secondary) sources. This is followed by finding valid reflection paths to compute actual contributions at the receiver. We have applied our algorithm to complex models with tens of thousands of triangles. In practice, we observe performance improvement of up to 20X using a single-core implementation over prior accurate propagation methods that are based on beam tracing [Laine et al., 2009].

### 1.6.3 Conservative From-Region Visibility Algorithm

The two prior approaches can only model specular reflections. However, it is very important to model diffraction in sound propagation. Thus, we also present an algorithm for fast finite-edge diffraction modeling (based on the BTM model) for GA in static scenes with moving sources and listeners. Efficient BTM-based diffraction requires the capability to determine which other diffracting edges are visible from a given diffracting edge. This reduces to a *from-region visibility* problem, and we use a conservative from-region visibility algorithm which can compute the set of visible triangles and edges at object-space precision in a conservative manner. We also present a novel occluder selection algorithm that can improve the performance of from-region visibility computation on large, complex models and perform accurate computation. The main contributions are as follows:

- **Accelerated higher-order BTM diffraction.** We present a fast algorithm to accurately compute the first few orders of diffraction using the BTM model. We use object-space conservative from-region visibility to significantly reduce the number of edge pairs that need to be considered for second order diffraction. We demonstrate that for typical models or scenes used in room acoustic applications, our approach can improve the performance of BTM edge diffraction algorithms by a factor of 2 to 4.

- **Effective occluder selection for region-based visibility.** We present a fast algorithm for occluder selection that can compute occluders in all directions around a given convex region. Our algorithm can combine connected sets of small primitives into large occluders and is more effective in terms of culling efficiency. The final set of visible primitives can be computed using a state-of-the-art occlusion culling technique. We demonstrate that our occluder selection technique is able to quickly generate occluders, consisting of 2-6 triangles each, in a few seconds per

visibility query on a single core.

### 1.6.4 Efficient Audio Processing

Many interactive applications have dynamic scenes with moving source and static receiver (MS-SR) or static source and moving receiver (SS-MR). We present artifact-free audio processing for these dynamic scenes. We use delay interpolation combined with fractional delay filters and windowing functions applied for IR interpolation to reduces artifacts in output audio. There are also application with moving sound sources as well as moving receiver (MS-MR). In such scenarios, as a receiver moves, it receives sound emitted from prior positions of a given source. We present an efficient algorithm that can correctly model sound propagation and audio processing for MS-MR scenarios by performing sound propagation and signal processing from multiple source positions. Our formulation only needs to compute a portion of the response from various source positions using sound propagation algorithms and can be efficiently combined with signal processing techniques to generate smooth, spatialized audio. Moreover, we present an efficient signal processing pipeline, based on block convolution, which makes it easy to combine different portions of the response from different source positions. Finally, we show that our algorithm can be easily combined with well-known GA methods for efficient sound rendering in MS-MR scenarios, with a low computational overhead (less than 25%) over sound rendering for static sources and a static receiver (SS-SR) scenarios. Some of the new components of our work include:

- **Sound Rendering for MS-MR Scenes**: We present an algorithm to accurately model sound rendering for dynamic scenes with simultaneously moving sources and a moving receiver (MS-MR). We show that our technique can also be used to perform efficient sound rendering for moving sources and a static receiver (MS-SR) as well as static sources and a moving receiver (SS-MR).

- **Efficient Signal Processing Pipeline**: We present a signal processing pipeline, based on block convolution, that efficiently computes the final audio signal for MS-MR scenarios by convolving appropriate blocks of different IRs with blocks of the input source audio.

- **Modified Sound Propagation Algorithms**: We extend existing sound propagation algorithms, based on the image-source method and pre-computed acoustic transfer, to efficiently model propagation for MS-MR scenarios. Our modified algorithms are quite general and applicable to all MS-MR scenarios.

- **Low Computational Overhead**: We show that our sound propagation techniques and signal processing pipeline have a low computational overhead (less than 25%) over sound rendering for SS-SR scenarios.

## 1.7 Thesis Organization

The rest of the thesis is organized as follows. The first few chapters of the thesis (Chapter 3 and Chapter 4) describe efficient geometric sound propagation techniques accelerated by using visibility algorithms. Chapter 5 deals with efficient artifact-free audio processing and Chapter 6 deals with accuracy related issues in our geometric sound propagation algorithms. More specifically,

- In **Chapter 2**, we review the previous work in visibility techniques, sound propagation, audio processing, and validation of geometric acoustics.

- In **Chapter 3**, we highlight the connection between the from-point visibility problem in computer graphics and specular reflection modeling in geometric sound propagation. We provide details and results on AD-Frustum, a frustum data structure for approximate from-point visibility, which is geared towards modeling specular reflections for interactive applications. We also present details and results

on FastV, a conservative from-point visibility algorithm, geared towards efficiently computing accurate specular reflections for engineering applications.

- In **Chapter 4**, we present details and results on a conservative from-region visibility algorithm which is applied to accelerate finite-edge diffraction. We summarize our image source method which integrates from-point and from-region visibility to compute specular reflections and finite-edge diffraction efficiently.

- In **Chapter 5**, we present techniques for artifact-free audio processing for dynamic scenes with moving source and static receiver (MS-SR) or static source and moving receiver (SS-MR). We also present our efficient audio processing framework for scenarios with a moving source and a moving receiver (MS-MR).

- In **Chapter 6**, we compare our geometric sound propagation algorithms with numerical sound propagation methods.

- **Chapter 7** concludes the thesis and proposes directions for future work.

# Chapter 2

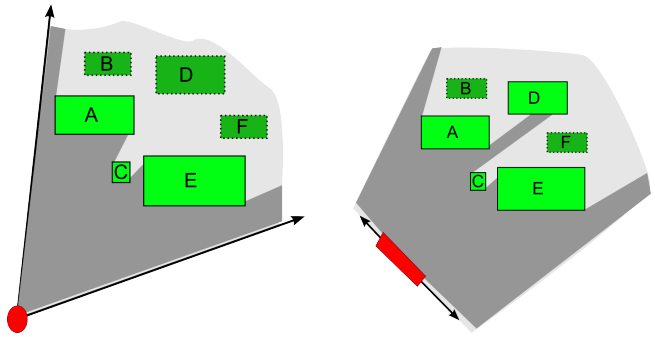
# Previous Work

In this chapter, we review the previous work related to visibility algorithms, sound propagation, audio processing, and validation of geometric acoustics techniques.

## 2.1 Visibility Algorithms

Visibility is a widely-studied problem in computer graphics and related areas. Visibility algorithms can be classified in different ways. One way to classify these algorithms is into object space and image space algorithms. The object space algorithms operate at object-precision, i.e. visibility computations are performed using the raw primitives (e.g. triangles). The image space algorithms resolve visibility based on a discretized representation of the objects and the accuracy typically corresponds to the resolution of the final image in computer graphics. These image space algorithms are able to exploit the capabilities of rasterization hardware and can render large, complex scenes composed of tens of millions of triangles at interactive rates using current graphics processors (GPUs). Alternatively, visibility algorithms can be classified into from-point and from-region visibility.

We now formally define from-point and from-region visibility. Given a view-point ($\mathbf{v} \in \Re^3$, from-point) or a view-region ($\mathbf{v} \subset \Re^3$, from-region), a set of geometry primitives ($\Pi$), and a viewing frustum ($\Phi$), which is a set of infinitely many rays originating in $\mathbf{v}$,




(a) (b)

Figure 2.1: Object-space exact visibility algorithms. *(a) Exact from-point visibility, based on object-space computations. (b) Exact from-region visibility. The red circle and red rectangle denote a view-point and a view-region, respectively. The light gray region bounded by the two arrows is the viewing frustum that is used to compute the visible primitives. The geometric primitives are labeled A, B, C, D, E, and F. The visible primitives are marked as solid bright green boxes. The dark gray region is the region consisting of visible primitives as determined by the visibility algorithm.*

the goal of visibility techniques is to compute a set of primitives $\pi \subseteq \Pi$ hit by rays in $\Phi$. For example, in Figure 2.1(a) the red circle corresponds to the view-point and in Figure 2.1(b) the red rectangle corresponds to the view-region. The set of primitives is $\Pi = \{A, B, C, D, E, F\}$ and the region shaded in light gray bounded by two arrows is spanned by rays in $\Phi$, the viewing frustum. In Figure 2.1(a) the visible set of primitives $\pi = \{A, C, E\}$ and in Figure 2.1(b) the visible set of primitives $\pi = \{A, C, D, E\}$. Note that the set $\pi$ is called the potentially visible set (PVS). Depending on the properties of the computed PVS, visibility techniques can be further classified. Visibility techniques have been extensively studied in computer graphics, computational geometry, robotics and related areas for more than four decades. We refer the readers to excellent surveys [Durand et al., 2000, Cohen-Or et al., 2003] for a comprehensive overview of visibility techniques. In this section, we give a brief overview.

### 2.1.1 Object-Space Exact Visibility

This class of visibility techniques computes a PVS, $\pi_{exact}$. Primitive hit by any ray in $\Phi$ is in $\pi_{exact}$ and every primitive in $\pi_{exact}$ is hit by some ray in $\Phi$. Since every ray in $\Phi$ is considered to compute visibility, these techniques are called object-space techniques. Moreover, these intersection computations are performed at the accuracy of the original model, e.g. IEEE 64-bit double precision arithmetic. The PVS computed by an object-space exact visibility algorithm is the smallest PVS which contains all the primitives visible from $\mathbf{v}$.

**From-Point Visibility**: Figure 2.1(a) shows an example of exact from-point visibility. Primitives A, C, and E block all the rays in the viewing frustum starting at the view-point from reaching the primitives B, D, and F. Thus, the primitives B, D, and F are marked as hidden. The two main approaches for computing exact from-point visibility are based on beam tracing [Heckbert and Hanrahan, 1984, Overbeck et al., 2007] and Plücker coordinates [Nirenstein, 2003].

*Beam tracing* approaches shoot a beam from the view point and perform exact intersections of the beam with the primitives in the scene. As the beam hits the primitives, exact intersection and clipping computations are performed between the beam and the primitive. The portion of the beam which is not hit by any primitive so far is checked for intersections with the remaining primitives. Thus, the complexity of the shape of the beam may increase as more intersection computations are performed. In general, performing exact and robust intersection computations with a beam on complex 3D models is considered a hard problem.

*Plücker coordinates* based approaches perform constructive solid geometry (CSG) operations in Plücker space to compute exact visibility. Plücker space is a six-dimensional space with certain special properties [Nirenstein et al., 2002]. In this approach, the view-frustum and the primitives are represented in Plücker space as CSG primitives and intersection computations are performed between the view-frustum and the primitives

such that when the CSG intersection is transformed back into Euclidean space, it corresponds exactly to the visible primitives. The intersection between the view-frustum and primitives in Plücker space requires complex operations. Thus, these techniques can be used to perform exact from-point visibility computations, but can be computationally expensive and susceptible to robustness problems.

**From-Region Visibility**: Figure 2.1(b) shows an example of from-region exact visibility. Primitives A, C, D, and E are visible from the view-region. Note that no ray starting in the view region reaches B or F, and therefore they are marked as hidden from the view-region. Many complex data structures and algorithms have been proposed to compute exact from-region visibility, including aspect graphs [Gigus et al., 1991], visibility complex and visibility skeleton [Durand et al., 1996, Durand et al., 1997], and performing CSG operations in Plücker space [Nirenstein et al., 2002, Haumont et al., 2005]. These methods have high complexity – $O(n^9)$ for aspect graphs and $O(n^4)$ for the visibility complex, where $n$ is the number of geometry primitives – and are too slow to be of practical use on complex models.

*Aspect graph* is a data structure that incorporates information about the views of an object. It is used in computer vision to determine different aspects of objects and match aspects to determine a camera view location. Therefore, aspect graphs are view-centric and changes in aspects do not necessarily imply changes in PVS. The complexity of aspect graphs for a perspective camera is $O(n^9)$. *Visibility complex* and its low order approximation, *visibility skeleton*, are compact data structures which encode the visibility information of a scene. The complexity of visibility complex is $O(n^4)$. The basic idea of from-region exact visibility computation based on *Plücker coordinates* is that all lines between polygons are represented in a 5D space derived from Plücker space and then geometric subtractions are performed on the lines occluded by other polygons. This approach has been accelerated by using a mechanism which enhances early termination by searching for aperture between query polygons [Haumont et al., 2005] and has been

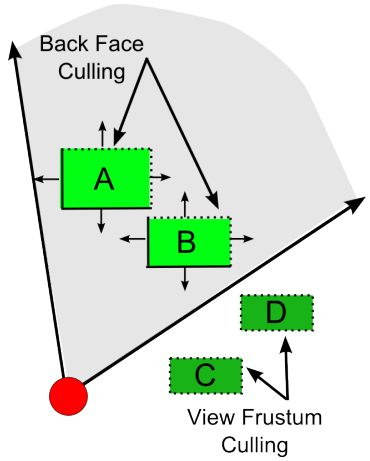


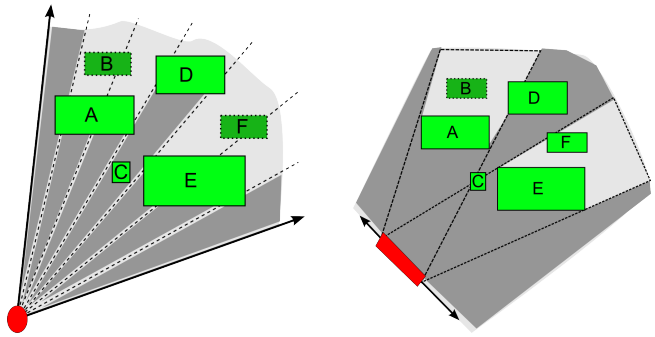

Figure 2.2: View-frustum culling *and* back-face culling *to trivially remove hidden primitives. In practice, these algorithms are easier to implement as compared to advanced culling methods but are highly conservative, i.e., a large number of potentially visible primitives computed by these methods are completely hidden.*

applied to practical scenes with timing performance which is an order of magnitude better than other exact from-region visibility approaches.

### 2.1.2 Object-Space Conservative Visibility

These visibility techniques compute a PVS, $\pi_{conservative}$. Primitive hit by any ray in $\Phi$ is in $\pi_{conservative}$, but $\pi_{conservative}$ may contain primitives which are not hit by any ray in $\Phi$. Thus, $\pi_{conservative}$ is conservative, i.e., $\pi_{conservative} \supseteq \pi_{exact}$. Conservative from-point visibility algorithms are preferred for their computational efficiency and simplicity over exact algorithms. The two simple and widely used but highly conservative visibility techniques are *view-frustum culling* and *back-face culling*. They are used to trivially remove some of the hidden primitives. Figure 2.2 illustrates these methods. In view-frustum culling, the primitives completely outside the view-frustum are marked hidden; and in back-face culling, the primitives which are facing away from the view-point or view-region are marked as hidden. Conservative visibility is preferred in many applications mainly due to its ease of implementation and good performance improvement.

**From-Point Visibility**: In Figure 2.3(a) we show an example of a conservative

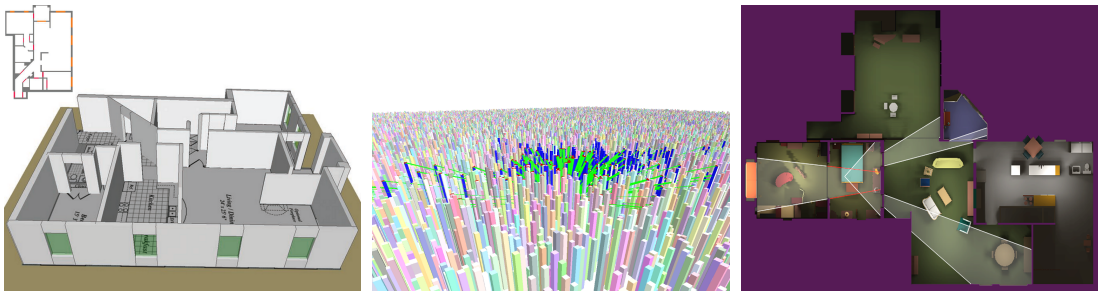



(a) (b)

Figure 2.3: Object-space conservative visibility algorithms. *(a) Conservative from-point visibility, which tends to compute a superset of primitives that are visible from a given view-point. (b) Conservative from-region visibility. In (a), the visible primitives are computed by shooting frusta and in (b), the visible primitives are computed by constructing shadow frusta for primitives A and E.*

from-point visibility approach. Note that primitive D, which is not visible from the view-point, is still reported as potentially visible by the conservative approach. Primitives B and F remain hidden from the view-point. Many techniques have been developed for conservative from-point visibility computations: cell and portal visibility [Luebke and Georges, 1995], visibility computations using supporting and separating planes [Coorg and Teller, 1997], shadow frusta [Hudson et al., 1997], occlusion trees [Bittner et al., 1998], and underestimated rasterization [Akenine-Möller and Aila, 2005]. Many of these algorithms have been designed for special types of models, e.g. architectural models represented as cells and portals, 2.5D urban models, or scenes with large convex primitives. These methods are well suited when the target application of the visibility algorithms is limited to urban scenes or architectural models corresponding to buildings or indoor structures with no interior primitives or furniture. Figure 2.4 gives examples of these special kinds of models that can be handled by these methods.

*Cell and portal visibility* method [Luebke and Georges, 1995] perform a depth-first rendering of the objects in a cell, and recursively perform the rendering for the cells connected to the portal for only the portion of the portal visible from the view-point. This approach is restricted to cell and portal scenes only, where a cell is a polyhedral



(a) (b) (c)

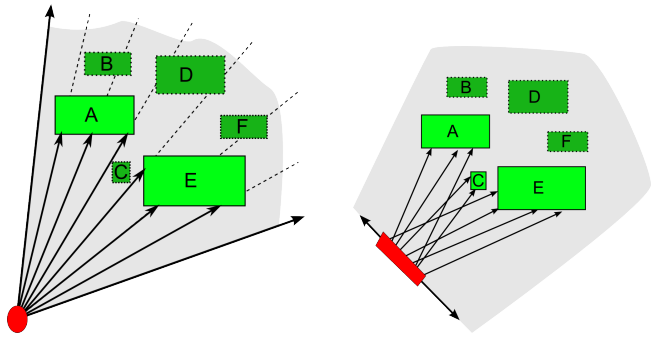
Figure 2.4: Special scenes used for visibility computation. *(a) Buildings with clearly marked cells and portals and no geometry or furniture inside the cells (source [Yin et al., 2009]). (b) Urban scenes, which can be represented using 2.5D objects or height fields (source [Bittner and Wonka, 2005]). (c) Simple scene with large occluders (source [Luebke and Georges, 1995]). The walls of the rooms are used as occluders.*

volume of a space and a portal is a transparent 2D region upon a cell boundary that connects adjacent cells. The work by Coorg and Teller [Coorg and Teller, 1997] computes occlusion caused by exploiting temporal coherence and identifying visibility events with respect to large convex occluders. This approach is limited to identifying large occluders in the scene. The *Shadow frusta* approach proposes to compute shadow frusta from dynamically selected occluders and then hierarchically removes primitives inside the shadow frusta [Hudson et al., 1997]. The basic idea of a shadow frusta is that a viewer cannot see a primitive if it is inside the shadow of an occluder. This approach requires large occcluders otherwise it will generate highly conservative PVS. The *Occlusion tree* approach combines the shadow frusta in an occlusion tree [Bittner et al., 1998]. If a primitive is visible, then it is treated as an occluder and its shadow frusta is inserted into the occlusion tree. Once the tree is built, the scene hierarchy is compared with it to remove hidden primitives. This approach is more efficient than shadow frusta, but also requires large occluders to compute a tight PVS. The *Underestimated raterization* based approach uses GPUs, and does not draw the pixels for the silhouette of a primitive from a view-point. The basic idea behind underestimated rasterization is to only draw a pixel which lies completely inside the primitives, i.e., to not draw pixels which map to the

edges of a primitive. Such an approach requires connectivity information and silhouette primitives to be computed in a streaming fashion to be efficient on GPU, otherwise it will result in an inefficient implementation and highly conservative PVS. We are not aware of any efficient implementation of underestimated rasterization which computes a tight conservative PVS.

**From-Region Visibility**: Figure 2.3(b) shows an example of a conservative from-region visibility. The basic idea is to construct *shadow frusta* (polyhedral beams contained within the *umbrae* between the view-region and primitives) for selected primitives. Typically, these primitives are selected by an *occluder selection* algorithm based on their effectiveness in removing hidden primitives. Primitives which are completely inside the shadow frusta are marked as hidden. In Figure 2.3(b), only the primitive B is completely inside the shadow frusta of primitives A and E. Also, note that primitive F is marked as potentially visible even though there is no ray originating in the view-region which reaches F. A few algorithms have been proposed for conservative from-region visibility: cell and portal from-region visibility [Teller and Séquin, 1991, Teller, 1992], extended projection [Durand et al., 2000], and vLOD [Chhugani et al., 2005].

*Cell and portal from-region visibility* algorithms decompose the scene into cells and portals and compute which cells are visible from a given cell [Teller and Séquin, 1991, Teller, 1992]. The visibility between two cells is computed by determining if any of the portals connected to the two cells are visible to each other. *Extended projection* computes PVS by determining if the projection of a primitive on a plane lies completely behind the projection of an occluder on the plane. To perform projection from a view-region, the projection of the occluder from different points in the region is computed and is underestimated, whereas the projection of the primitives is overestimated to ensure the conservative nature of the computed PVS. *vLOD* compute an optimal shadow frustum from the view region by combining shadow frusta from shrunk occluders [Chhugani et al., 2005]. Further, an optimal view-point for the optimal shadow




(a) (b)

Figure 2.5: Image-space visibility algorithms. *(a) Sample-based from-point visibility. The visibility computation is accurate up to the resolution of the rays used. (b) Sample-based from-region visibility.*

frustum is computed, and a from-point sample-based visibility (see Section 2.1.3) is performed. Since occluders are shrunk, it guarantees conservative PVS and from-point sample-based visibility performs occluder fusion even if the occluders are not connected to each other. However, vLOD still requires that large connected primitives are provided as occluders to create shadow frusta.

### 2.1.3 Image-space or Sample-based Visibility

These approaches sample the set of rays in $\Phi$ and compute a PVS, $\pi_{sampling}$. Primitives hit by the finite set of sampled rays are in $\pi_{sampling}$. Note that since $\pi_{sampling}$ is computed for only a finite subset of rays in $\Phi$, $\pi_{sampling} \subseteq \pi_{exact}$. The choice of sampled rays is governed by the application.

**From-Point Visibility**: We show an example of from-point sample-based visibility in Figure 2.5(a). Only a few rays are sampled and intersected with the geometric primitives to find the visible primitives. This could lead to spatial aliasing, as shown in Figure 2.5(a). The primitive C is marked as hidden because it lies between two sampled rays even though it is visible from the view-point. Despite their short comings, sample-based methods are widely used in computer graphics [Foley et al., 1990] and related areas. Efficient implementation of sample-based visibility algorithms can be achieved

on current graphics processing units (GPUs). The z-buffer algorithm [Catmull, 1974, Theoharis et al., 2001] is a standard sample-based visibility algorithm that is supported by the rasterization hardware in GPUs. Sample-based ray shooting techniques have also been used extensively for visibility computation [Whitted, 1980, Glassner, 1989].

**From-Region Visibility**: We show an example of from-region sample-based visibility in Figure 2.5(b). Similar to from-point visibility, the sampling in from-region algorithms introduces spatial aliasing. In this case, the primitive D is marked as hidden even though there exists at least one ray from the view-region that reaches the primitive D. These methods are fast compared to exact and conservative from-region visibility algorithms and can easily be applied to complex models. However, they have one important limitation: they sample a finite set of rays originating inside the view-region and thus compute only a *subset* of the exact solution (i.e., approximate visibility). Therefore, these methods are limited to sampling-based applications such as interactive graphical rendering, and may not provide sufficient accuracy for applications where an accurate from-region solution is needed.

*Guided visibility sampling* [Wonka et al., 2006] presents smart sampling strategies which combine random sampling with a deterministic exploration. It is applicable for general 3D scenes and does not assume any connectivity information. An extension of guided visibility sampling, *adaptive global visibility sampling* [Bittner et al., 2009] was recently proposed. It exploits coherence in visibility computation between different view regions and progressively refine the PVS.

### 2.1.4 Acceleration Structures

Scene hierarchies or acceleration structures are used extensively in visibility algorithms, especially ray tracing and geometric tracing techniques, to accelerate intersection computation with the primitives in the scene. Many acceleration structures like, uniform space partition, octree [Arvo and Kirk, 1989], K-d tree [Bentley, 1975], and BSP tree

| Method | Note |
|---|---|
| Object-precision exact | |
| [Heckbert and Hanrahan, 1984] | From-point, beam tracing. |
| [Nirenstein et al., 2002] | From-point, Plücker space computation. |
| [Overbeck et al., 2007] | From-point, faster beam tracing. |
| [Gigus et al., 1991] | From-region, aspect graph. |
| [Durand et al., 1996]<br>[Durand et al., 1997] | From-region, visibility complex and its simplified version, visibility skeleton. |
| [Nirenstein et al., 2002]<br>[Haumont et al., 2005] | From-region, Plücker space CSG operations, extension with efficient early termination. |
| Object-precision conservative | |
| [Luebke and Georges, 1995] | From-point, cell and portal scenes. |
| [Coorg and Teller, 1997] | From-point, temporal coherence, large occluders. |
| [Hudson et al., 1997] | From-point, shadow frusta, large occluders. |
| [Bittner et al., 1998] | From-point, occlusion tree, large occluders. |
| [Akenine-Möller and Aila, 2005] | From-point, underestimated rasterization. |
| [Chandak et al., 2009] | From-point, frustum tracing, blocker computation, general 3D scenes. |
| [Teller and Séquin, 1991]<br>[Teller, 1992] | From-region, cell and portal scenes, cell-cell visibility by using visibility of connected portals. |
| [Durand et al., 2000] | From-region, extended projection. |
| [Chhugani et al., 2005] | From-region, vLOD, shadow frusta. |
| [Antani et al., 2011c] | From-region, compute connected occluders using frustum tracing, use shadow frusta for visibility. |
| Image-precision | |
| [Catmull, 1974] | From-point, z-buffer. |
| [Whitted, 1980] | From-point, ray tracing. |
| [Wonka et al., 2006]<br>[Bittner et al., 2009] | From-region, smart visibility sampling, progressive visibility, coherence, general 3D scenes. |

Table 2.1: Visibility techniques [Durand et al., 2000, Cohen-Or et al., 2003].

[Fuchs et al., 1980], have been proposed. As new geometric tracing structures are used for visibility computation, appropriate algorithms need to be developed for the intersection between the scene hierarchy and the geometric tracing structure. Further, as many applications are dynamic in nature, a lot of effort has been invested in updating scene hierarchies efficiently for dynamic scenes [Wald et al., 2007a]. Much work has also been done to compute these hierarchies in parallel fashion on multi-core CPUs [Wald, 2007] and many-core GPUs [Lauterbach et al., 2009].

## 2.2 Sound Propagation

The propagation of sound in a medium is modeled using the *acoustic wave equation* (AWE):

$$\frac{\partial^2}{\partial t^2} P(\mathbf{x}, t) - c^2(\mathbf{x}, t) \nabla^2 P(\mathbf{x}, t) = 0 \tag{2.1}$$

where $P$ is the sound pressure as a function of position $\mathbf{x}$ and time $t$, and $c$ is the speed of sound. This is a second-order hyperbolic partial differential equation (PDE), and can be solved using standard time domain numerical methods.

Alternatively, the Fourier transform can be used to derive a frequency-domain elliptical PDE, called the *Helmholtz equation* (HE):

$$\nabla^2 \psi + k^2 \psi = 0 \tag{2.2}$$

where $P(\mathbf{x}, t) = \psi(\mathbf{x}) e^{\iota \omega t}$ and $k = \omega / c$. The Helmholtz equation can be solved using standard frequency domain numerical methods and other so-called *spectral methods*.

Typically, in sound propagation techniques, given a source, a receiver, and a 3D model with acoustics material properties, the goal is to compute the sounds waves reaching the receiver as they emit from the source. Under the assumptions of linear constant medium, the system is a linear time invariant system. Therefore, the response from a source to a receiver can be characterized by an *impulse response* (IR). The IR is computed at the receiver's position, and represents the pressure signal arriving at the receiver for a unit impulse signal emitted by the isotropic point source. This implies that given an input sound signal emitted by the source, the signal received by the receiver (taking into account propagation effects) can be obtained by convolving the input signal with the impulse response. The two main approaches to sound propagation are numerical methods and geometric methods [Kleiner et al., 1993, Svensson and Kristiansen, 2002, Funkhouser et al., 2003]. Recently, pre-computation based methods are gaining popularity for interactive applications. We review these approaches in the following sections.

### 2.2.1 Numerical Methods

Solving the wave equation or the Helmholtz equation for typical spaces used for architectural acoustics requires significant amounts of processing time (proportional to the fourth power of the sound frequency) and memory (proportional to the volume of the scene). The complexity of the numerical methods is a function of physical parameters like volume or surface area of the scene, the maximum simulated frequency, and time-duration of the simulation. The numerical methods provide highly accurate results by easily modeling complex scattering, reflections, and diffraction of sound waves. However, numerical methods are highly compute intensive. For example, using Finite Difference Time Domain (FDTD) for a domain of size 100m $\times$ 100m $\times$ 100m for frequencies up to 2000Hz would require $\sim$100GB of memory and $\sim$1000 hours of computation time [Mehra et al., 2012]. In fact, a recent FDTD implementation on a cluster of computers when applied to medium-sized scenes, took tens of GBs of memory and tens of hours of computation time [Sakamoto et al., 2004].

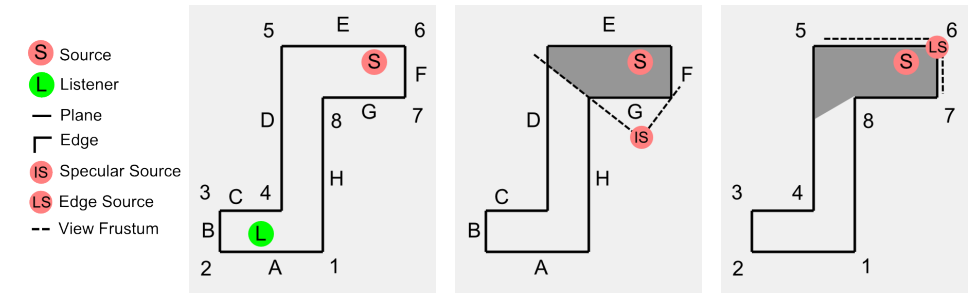
Various classes of numerical methods have been applied to solve the wave equation [Kleiner et al., 1993], such as the Finite Element Method (FEM), the Boundary Element Method (BEM), the Finite-Difference Time-Domain (FDTD) method, and Digital Waveguide Meshes (DWM). FDTD methods are popularly used in room acoustics [Botteldooren, 1995] due to their simplicity. Recently, an Adaptive Rectangular Decomposition (ARD) technique [Raghuvanshi et al., 2009] was proposed, which achieves two orders of magnitude speed-up over FDTD methods and other state-of-the-art numerical techniques. A GPU implementation of ARD achieves an order of magnitude speed-up over its CPU implementation [Mehra et al., 2012]. In practice, numerical methods accurately model the acoustic wave equation but they are quite expensive in terms of handling large acoustic spaces and are not practical for interactive or dynamic scenes.

### 2.2.2 Geometric Methods

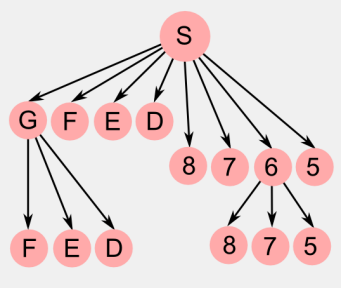
Geometric acoustics (GA) is a high frequency approximation of the acoustic wave equation and can be used to derive efficient algorithms for sound propagation, based on image sources or ray tracing. Most sound propagation techniques used in practical applications model the acoustic effects of an environment using linearly propagating rays. GA techniques are not as accurate as numerical methods in terms of solving the wave equation, and cannot easily model all kinds of propagation effects, but they allow simulation of early reflections at real-time rates. Broadly, geometric acoustics algorithms can be divided into *pressure-based* and *energy-based methods*.

Pressure-based methods model specular reflections and edge diffraction, and are essentially variations of the image-source method [Allen and Berkley, 1979] (see Section 2.2.3). Specular reflections are easy to model using GA methods. Image source methods can guarantee to not miss any specular propagation paths between the source and the receiver. Diffraction is relatively difficult to model using GA techniques (as compared to specular reflections), because it involves sound waves bending around objects. The two commonly used geometric models of diffraction are the Uniform Theory of Diffraction (UTD) [Kouyoumjian and Pathak, 1974] and the Biot-Tolstoy-Medwin (BTM) model [Svensson et al., 1999]. The UTD model assumes infinite diffracting edges, an assumption which may not be applicable in real-world scenes (e.g., indoor scenes). However, UTD has been used successfully in interactive applications [Tsingos et al., 2001, Taylor et al., 2009a, Antonacci et al., 2004]. BTM, on the other hand, deals with finite diffracting edges, and therefore is more accurate than UTD [Svensson et al., 1999]; however it is much more computationally expensive and has only recently been used – with several approximations – in interactive applications [Schröder and Pohl, 2009].

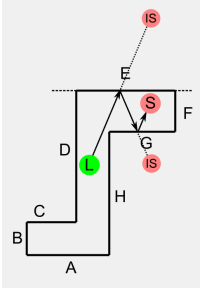
Energy-based methods are typically used to model diffuse reflections. Specular reflections and diffraction can also be modeled with energy-based methods when in-

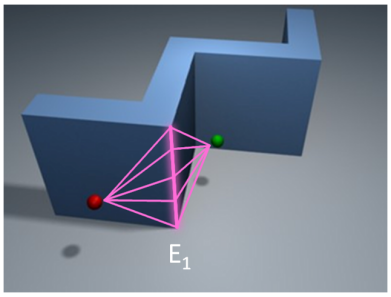



(a) Computing image sources




(b) Visibility tree




(c) Path validation

Figure 2.6: Overview of image source method. *(a) Top down view of the input geometry with source and receiver position. An image source is created by reflecting the source across the plane G. An edge image source is created by edge 6 by treating the diffraction edge as the image source. (b) A visibility tree is computed to store the specular image sources and edge image sources. Note that not all planes and edges are in the visibility tree. For example, no first order specular reflection path is possible from source to plane B, and similarly no first order edge diffraction path is possible from source to edge 4. Thus, reducing the size of visibility tree is the biggest challenge in image source methods. (c) In path validation step, once all image source are computed and stored in the visibility tree, they are validated to make sure that the final paths are not obstructed.*

terference of sound waves is not important. The *room acoustic rendering equation* [Siltanen et al., 2007] is an integral equation generalizing energy-based methods. Existing methods, like ray tracing [Krokstad et al., 1968], phonon tracing [Kapralos, 2006], and radiosity [Nosal et al., 2004], could be considered to solve the room acoustic rendering equation.

### 2.2.3 Image Source Method

The image source method [Allen and Berkley, 1979] is a popular technique for computing specular reflections. It is possible to extend image source method to handle edge

diffraction effects as well [Pulkki et al., 2002]. In this section, we present an overview of image source method and different geometric techniques to accelerate image source method.

Image source method has two main steps: (a) computing image sources step (see Figure 2.6(a)) and (b) validation step to find valid specular and diffraction paths (see Figure 2.6(c)). In the first step, image sources for specular reflection and edge diffraction are computed. Image sources for specular reflection are computed by reflecting the source or image source across the plane for which the image source is being computed. For edge diffraction, the edge itself is the image source. In its naïve form, image sources can be created for all planes and edges in a scene, and it will lead to an inefficient algorithm as specular reflections and edge diffraction are not possible from all the planes and diffraction edges (see Figure 2.6). These image sources are encoded in a visibility tree, which encodes planes and edges that are potentially visible from a source or an image source as shown in Figure 2.6(b). The goal of most pressure-based geometric methods is to reduce the size of the visibility tree. Once a visibility tree is computed, it may contain paths from image sources such that no specular path or edge diffraction is possible as it only contains "potentially" visible image sources. Further, even if an image source is visible, a path may still be obstructed, as only a plane or an edge is partially visible. Thus, a path validation step is performed which creates final specular paths and diffraction paths and ensures that they are not obstructed (see Figure 2.6(c)).

## Acceleration Techniques

Various geometric techniques have been proposed which accelerate image source method: beam tracing [Funkhouser et al., 1998], adaptive beam tracing [Laine et al., 2009], cone tracing [Genetti et al., 1998], and pyramidal tracing [Farina, 1995]. These methods cannot handle general 3D scenes, efficiently scale on the parallel hardware, and handle large complex dynamic scenes efficiently at the same time. Methods based on ray tracing

[Vorländer, 1989] to accelerate image source method suffer from spatial and temporal aliasing issues. Further, none of the previous methods model finite-edge diffraction. The basic idea behind all these techniques is to compute from-point visibility using different geometric structures like beams in the case of beam tracing and cone of rays in the case of cone tracing to compute visible primitives for which to compute image sources.

A few techniques have been proposed to accelerate the image-source method which incorporates finite-edge diffraction. Svensson et al. [Svensson et al., 1999] apply highly conservative from-region frustum culling from an edge and back face culling from an edge to reduce the size of the visibility tree. For finite-edge diffraction, computing all the diffraction path is a compute intensive step and some approaches have been proposed to cull away the paths which are not significant [Calamia and Svensson, 2007].

### 2.2.4 Acoustic Rendering Equation

The energy-based geometric room acoustics can be generalized by an integral equation [Siltanen et al., 2007] called the *acoustic rendering equation* (see Eq. 2.3). The acoustic rendering equation can be seen as an extension of the rendering equation in computer graphics [Kay and Kajiya, 1986].

$$L(x', \omega) = L_0(x', \omega) + \int_S R(x, x', \omega) L\left(x, \frac{x' - x}{|x' - x|}\right) dx \tag{2.3}$$

where $L$ is final outgoing radiance, $L_0$ is emitted radiance and $R$ is the *reflection kernel*, which describes how radiance at point $x$ influences radiance at point $x'$:

$$R(x, x', \omega) = \rho(x, x', \omega) G(x, x') V(x, x') P(x, x') \tag{2.4}$$

Here, $\rho$ is the BRDF (Bidirectional Reflectance Distribution Function) of the surface at $x$, $G$ is the form factor between $x$ and $x'$, $V$ is the point-to-point visibility function, and $P$ is a *propagation term* [Siltanen et al., 2007] that takes into account the effect

of propagation delays. The latter is unique to sound rendering as visual rendering algorithms neglect propagation delays due to the high speed of light. Also, depending on the reflectance function of a surface, different scattering properties of the surface, e.g. diffuse reflectance, can be modeled [Tsingos et al., 2007].

### Radiosity based methods

Acoustic rendering equation was only proposed recently, however, several methods developed before its proposal are applicable in solving the acoustic rendering equation as similar techniques have been used in computer graphics to solve the graphics rendering equation. *Radiosity* is one such approach to solve the acoustic rendering equation and many different variations have been proposed. These methods divide the surface primitives in to patches and compute transfer operators which essentially encode the impulse response between patches. Tsingos and Gascuel [Tsingos and Gascuel, 1997] proposed an approach which propagates energy from one patch to another similar to classical hierarchical radiosity like methods [Sillion and Puech, 1994]. Nosal et al. [Nosal et al., 2004] presented an extensive derivation of radiosity methods and efficient techniques to compute integrals between the patches for diffuse reflection modeling. Aalrcão et al. [Alarcão et al., 2010] proposed a technique which uses image source method for specular reflection and hierarchical radiosity method for modeling diffuse reflection. Siltanen et al. presented a general framework which can model arbitrary reflectance function based on the acoustic rendering equation [Siltanen et al., 2009]. Their method allows interactive walkthroughs with a moving receiver. Recently, these methods for solving the acoustic rendering equation have been extended to efficiently handle moving sound sources as well as moving receiver [Antani et al., 2011b] by applying ideas similar to pre-computed radiance transfer [Sloan et al., 2002], and efficiently modeling purely specular reflections [Antani et al., 2011a] using ideas similar to the two-pass method to model specular reflection and diffuse reflections in computer graphics

[Sillion and Puech, 1989]. These methods can model the acoustic rendering equation accurately but may require large computation time to compute the integral.

#### Ray-based or Particle-based Techniques

Some of the earliest methods for geometric sound propagation are based on tracing sampled-rays [Krokstad et al., 1968] or sound-particles (*phonons*). *Ray tracing* is a popular geometric algorithm for acoustic modeling and can model specular and diffuse reflections easily. Recently, phonon tracing has been extended to take into account the developments in computer graphics on efficient photon tracing [Bertram et al., 2005]. Also, particle-based methods have been extended to model diffraction [Kapralos, 2006, Stephenson, 2010]. Recent improvements in computer graphics to develop fast algorithms for tracing rays and particles, by taking advantage of multi-core and many-core architectures, efficient scene hierarchies, exploiting ray-coherence, and other acceleration techniques have made it possible for ray-based and particle-based methods to handle complex, dynamic scenes on commodity hardware or handle massive models. However, due to discrete sampling of the space, these methods have to trace large number of paths or particles to avoid aliasing artifacts.

### 2.2.5 Precomputation-based Methods

Sound propagation is computationally challenging, and many interactive applications like video games allow a very small compute and memory budget ($< 10\%$ of the total compute and memory budget) for sound computations. Hence, precomputation-based sound propagation approaches are becoming increasingly popular. Following approaches have been proposed: directional propagation cache [Foale and Vamplew, 2007], reverberation graphs [Stavrakis et al., 2008], geometry-based reverberation [Tsingos, 2009], frequency domain acoustic radiance transfer [Siltanen et al., 2009], numerical precomputation method [Raghuvanshi et al., 2010], and direct to indirect acoustic radiance

transfer [Antani et al., 2011b]. These approaches precompute sound propagation and store it in data structures such that it is efficient to model sound propagation at run time. These approaches are highly promising from the point of view of integrating them in interactive sound rendering systems.

*Directional propagation cache* takes advantage of cell and portal division in games and caches sound propagation paths between portals [Foale and Vamplew, 2007]. These paths are then used at run-time to compute a response from a source to a receiver. *Reverberation graphs* compute transfer operators, which encode pressure decay characteristics and coupling between portals in a cell and portal environment [Stavrakis et al., 2008]. During run-time, the reverberation graph is traversed and transfer operators are combined along different paths to compute decay envelopes for a source and a receiver pair. *Geometry-based reverberation* technique precomputes image source gradients to quickly estimate specular reflections at run-time [Tsingos, 2009]. *Precomputed acoustic radiance* computes high order reflections from a source and stores them on surface patches in Fourier space [Siltanen et al., 2009] as precomputed surface response. At run-time, the surface response is gathered at the receiver to compute the response from a source to a receiver. Numerical precomputation methods compute the acoustic response of a scene for several sampled source-receiver position pairs during pre-computation step; at run-time these responses are interpolated given the actual positions of the source and the receiver [Raghuvanshi et al., 2010]. *Direct to indirect acoustic radiance transfer* computes a transfer operator between surface patches [Antani et al., 2011b]. At run-time, a response from source to the patches and a response from receiver to the patches is combined with a transfer operator to compute the final response.

### 2.2.6 Sound Rendering Systems

A few sound rendering systems based on geometric acoustics have been developed, which include sound propagation as well as binaural audio processing. This includes com-

mercial systems like ODEON [Christensen, 2009], CATT [CATT, 2002], RAMSETE [RAMSETE, 1995], and research prototypes like RAVEN [Schröder and Lentz, 2006], beam tracing system [Funkhouser et al., 2004], and RESound [Taylor et al., 2009b]. All these systems, except for RESound are mainly limited to static scenes with a moving receiver.

ODEON is a popular acoustics software which can compute specular reflections, diffuse reflections, and diffraction [Christensen, 2009] and is a 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. CATT-Acoustic [CATT, 2002] is another room acoustic software which performs specular and diffuse reflections using a combination of image source method, ray tracing method, and randomized tail-corrected cone tracing [Dalenbäck, 1996]. It does not have support for diffraction computation. 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 over free edges [Kurze, 1974]. It does not support diffraction for non-free edges. RAVEN at RWTH Aachen University is a framework for real-time sound rendering of virtual environments [Schröder and Lentz, 2006]. It applies image source method for specular reflections. Further, spatial hierarchies are used to accelerate image source computation. To the best of our knowledge, RAVEN does not handle diffraction or dynamic scenes with moving source and scene geometry. Another prototype system for real-time sound rendering is based on beam tracing [Funkhouser et al., 2004, Tsingos et al., 2004]. It can perform specular reflections and diffraction using beam tracing. The diffraction calculations are based on Uniform Theory of Diffraction (UTD) and these systems can handle multiple orders of diffraction. A beam tree is constructed in an offline step which is expensive to update for dynamic

scenes including moving sources. RESound [Taylor et al., 2009b], various sub-parts of which are developed in this thesis, is also a real-time sound rendering system. It is based on a combination of frustum tracing and ray tracing to handle specular reflections, diffuse reflections, and diffraction. Multiple orders of diffraction based on UTD can be handled along with dynamic scenes with moving source and scene geometry.

## Modeling Approaches

Various modeling approaches, such as design of reflectance functions [Tsingos et al., 2007] and model simplification algorithms [Joslin and Magnetat-Thalmann, 2003], have been proposed to improve the performance of sound rendering or to handle complex scenes in interactive sound rendering. These techniques are complementary to sound propagation and can be combined into a sound rendering system to achieve better results.

| Method | Note |
|---|---|
| Numerical methods | |
| Standard methods: Finite Element Method (FEM), Boundary Element Method (BEM), Finite-Difference Time-Domain (FDTD), Digital Waveguide Meshes (DWM). | |
| [Raghuvanshi et al., 2009] | Adaptive rectangular decomposition (ARD). |
| Geometric methods (pressure-based) | |
| [Allen and Berkley, 1979] | image source method. |
| [Vorländer, 1989] | ray tracing acceleration of image source. |
| [Farina, 1995] | pyramidal tracing acceleration for image source. |
| [Funkhouser et al., 1998] | beam tracing acceleration of image source. |
| [Genetti et al., 1998] | cone tracing acceleration for image source. |
| [Laine et al., 2009] | adaptive beam tracing acceleration for image source. |
| [Svensson et al., 1999] [Calamia and Svensson, 2007] | BTM diffraction, frustum culling and back face culling acceleration for edge image sources. |
| Geometric methods (energy-based) | |
| [Krokstad et al., 1968] | ray tracing particles. |
| [Tsingos and Gascuel, 1997] | hierarchical radiosity like method. |
| [Nosal et al., 2004] | radiosity method, diffuse reflections. |
| [Bertram et al., 2005] | phonon tracing. |
| [Kapralos, 2006] [Stephenson, 2010] | sonel mapping (phonon tracing), models diffraction. |
| [Siltanen et al., 2009] | frequency domain radiance transfer, arbitrary reflectance, moving receiver only. |
| [Alarcão et al., 2010] | image source and hierarchical radiosity. |
| [Antani et al., 2011b] | frequency domain, pre-computed radiance transfer, moving source and receiver, diffuse reflections only. |
| [Antani et al., 2011a] | time domain, pre-computed, two-pass radiance transfer, specular and diffuse reflections. |
| Pre-computation based | |
| [Foale and Vamplew, 2007] | directional propagation cache, cell and portal. |
| [Stavrakis et al., 2008] | reverberation graphs, cell and portal. |
| [Tsingos, 2009] | image-source gradients. |
| [Siltanen et al., 2009] | frequency domain acoustic radiance transfer. |
| [Raghuvanshi et al., 2010] | numerical precomputation method. |
| [Antani et al., 2011b] | direct to indirect acoustic radiance transfer. |

Table 2.2: Sound propagation techniques [Funkhouser et al., 2003, Kleiner et al., 1993].

## 2.3 Audio Processing

In most sound rendering systems, there are two more key components other than sound propagation: *sound synthesis* and *audio processing*. Sound synthesis techniques model vibration of sound source using physically-based or parametric modeling. We discuss them only briefly in this section since we assume point sources playing recorded audio clips. Audio processing techniques, take into account the audio played by the sound sources and the propagation of sound from source to the receiver, as modeled by image sources or IRs, and play back the final audio output. There are many challenges in terms of audio processing for a sound rendering system: (a) the direction of the incoming sound is important in many interactive applications. Therefore, it is important that audio processing techniques produce binaural audio output, i.e. one channel for left ear and one channel for right ear, which include cues that help identify the direction of incoming sound waves; (b) geometric techniques are often used for interactive modeling of low order reflections, and therefore high order reflections (late reverberation) need to be incorporated by pseudo-physical or statical techniques; (c) dynamic scenes with moving sources, moving receiver, or both could lead to discontinuity and artifacts in the final audio output from audio processing step, and there are additional challenges in correctly modeling a scenario with both moving source and moving receiver; (d) large number of sound sources may be present in an interactive application, like video games, and it is important to perform audio processing with image sources or IRs efficiently for interactive performance. Below, we give a brief overview of prior work related to the challenges outlined above.

The output from the geometric sound propagation step is provided as input to the audio processing step. The sound propagation step could provide: (a) image sources or (b) impulse response (IR), to the audio processing step. If image sources are provided, then appropriate techniques need to be used to generate binaural audio from the image sources, artifact free audio output if source and hence image sources are moving, or

efficiently compute audio output from a large number of image sources. Similarly, in the case of IRs, IRs need to be provided for left and right channel which are convolved with the input audio to generate binaural audio output, need to be processed appropriately for generating late reverberation and artifact-free audio for dynamic scenes, and need to be efficiently processed in case of large number of IRs due to a large number of sound sources.

### 2.3.1 Sound Synthesis

Sound synthesis generates audio signals based on interactions between the objects in a virtual environment. Synthesis techniques model vibrations of objects which 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. We use recorded audio clips in our sound propagation algorithms, which can be replaced by audio signal generated by sound synthesis modeling. Thus, these approaches are complementary to the presented work and could be incorporated in overall sound rendering system to improve overall experience.

### 2.3.2 Binaural Audio

The key concept behind binaural audio processing is that humans have two ears and the sound waves reaching a human receiver interacts with the receivers' head, shoulders, and torso, and slightly different sound signals reach the two ears. It is the differences between the signals reaching the two ears that provide cues to identify the direction of the incoming sound [Begault, 1994]. The effect of the human body on incoming sound waves can be incorporated in many different ways. The most common approach is to measure a head related transfer function (HRTF), which encodes IRs for the left and the right ear

for sound waves coming from different directions. Many such measurements are available publicly at various angular resolutions [Algazi et al., 2001, Gardner and Martin, 1995]. Recently, such responses have also been computed by numerical modeling of the acoustic wave equation by placing the sources at different angular direction around the 3D model of the human head and torso and computing the response at the human ears [Dellepiane et al., 2008].

In the case of multiple image sources as input to the audio processing step, each image source is treated as an individual source, and sound from each source/image source takes into account the HRTF to generate binaural audio. For a large number of image sources, this could be a very expensive step. In the case of IRs as input to the audio processing step, they should already take into account the HRTF into the binaural IRs provided to the audio processing system.

### 2.3.3 Late Reverberation

The geometric sound propagation techniques are often used to compute early orders of specular reflection and diffraction since modeling late reverberation in real-time is a challenging task. Therefore, reverberation estimated artificially using parameters like the size of the room [Gardner, 1998] or estimated physically by using the IRs or image sources [Lehmann and Johansson, 2010] are needed. These techniques are complementary to the approaches present in this thesis and efficient techniques to model late reverberation should be integrated into a sound rendering system.

### 2.3.4 Dynamic Scenes

Moving sound sources, receiver, and scene objects cause variations in IRs from a source to a receiver or variations in the position of image sources and final specular and diffraction paths from a source to a receiver. These variations could lead to artifacts in the audio output as discontinuities may appear due to changing IRs or image sources. The key idea

to generate artifact free audio output is to appropriately interpolate the IRs or image sources. Below, we present the prior work to reduce artifacts in dynamic scenarios during the audio process step.

**Parametric Interpolation**: In parametric interpolation, a few key parameters in a sound propagation response are identified. These parameters are interpolated in a dynamic scene to generate smooth audio output. For instance, delay and attenuation parameters can be interpolated in the case of image source method [Savioja et al., 1999, Wenzel et al., 2000] or position parameter of the image sources can be estimated by interpolation [Tsingos, 2001a] or prediction [Tsingos, 2001b].

**Windowing-based Interpolation**: Parametric interpolation approaches are suitable where clearly identified parameters could be interpolated. It may not be possible to compute such parameters for an IR provided as input to the audio processing step. In such cases, windowing-based schemes to interpolate the convolved audio output are used [Siltanen et al., 2009]. Essentially, any given output audio sample is a weighted combination of output at that sample due to different IRs.

**Moving Source and Moving Receiver**: In many interactive applications, typically both the source and the receiver are moving. This presents additional challenges to model such scenario as the sound reaching the current receiver position may come from many prior source positions and require IR computation between the current receiver position and many prior source positions. Not much work has been done to address this issue in propagation and audio processing as interactive systems, which can model scenarios with moving source and moving receiver have not been developed.

### 2.3.5 Efficient Audio Processing

Many applications, like video games or VR applications, may have large number of sound sources for which audio processing need to be performed. In the case of image sources, the sound propagation step may output hundreds to thousands of image sources

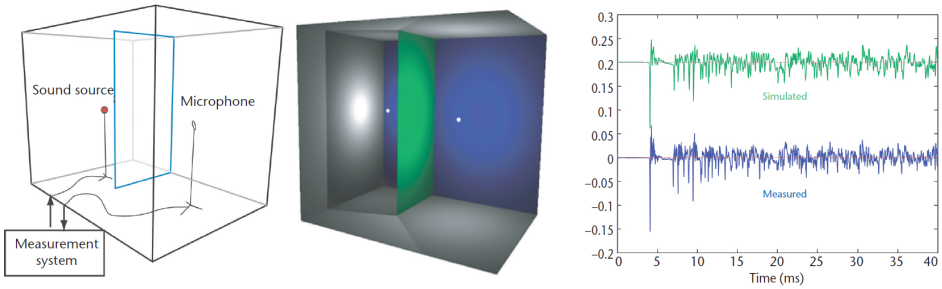
and efficiently computing binaural audio for these many sound sources is compute intensive and interactive performance is not possible on commodity hardware. The same constraint applies when a large number of convolutions for tens or hundreds of IRs need to be performed. Thus, it is important to design efficient techniques to compute the final audio signal for large number of sound sources. Below, we review a few approaches for efficient audio processing.

**Image-source clustering**: In the case of large number of image sources, doing binaural processing for each image source is impractical. Therefore, clustering approaches based on purely statistical measures [Tsilfidis et al., 2009, Wand and Straßer, 2004] or perceptual measures [Tsingos et al., 2004] have been proposed to perform mixing within each cluster and performing binaural audio processing per cluster.

**Signal processing optimizations**: Various techniques have been proposed to preprocess the input audio in blocks and store auxiliary perceptual information with each block. Blocks of audio with most perceptual significance [Moeck et al., 2007] are processed first. Further, alternative representation of blocks of audio, such as Fourier domain representation, can be stored to optimize the convolution processing during the runtime [Tsingos, 2005].

## 2.4 Geometric Acoustics Validation

A lot of work has been done in the past decades on developing geometric propagation algorithms [Funkhouser et al., 2003, Svensson and Kristiansen, 2002]. Some studies for the validation of these techniques have also been presented. One such study is the validation of acoustics simulation using the Bell Lab Box [Tsingos et al., 2002]. Tsingos et al. constructed a controlled environment (see Figure 2.7(a)) and compared measured results with simulated results using the image source method and diffraction modeling using Uniform Theory of Diffraction (see Figure 2.7(b)). Other such studies include the




(a) (b)

Figure 2.7: Bell lab box validation [Tsingos et al., 2002]. *(a) A controlled set up is constructed which contains a baffle dividing a rectangular room into two regions. Measurement setup is also shown. (b) Comparison of simulated early response including first ten orders of reflection and a measured response.*

international round robin competitions for acoustics software [Bork, 2002]. They compare the result of simulation and research software with measured data on real world models with varying levels of details. So far, three international round robin competitions have been conducted and a limited set of data is made available for comparison. Currently, efforts are being made to organize the fourth international round robin competition. Experiments have been also proposed to compare the accuracy of BTM-based finite edge diffraction [Svensson et al., 1999, Calamia, 2009].

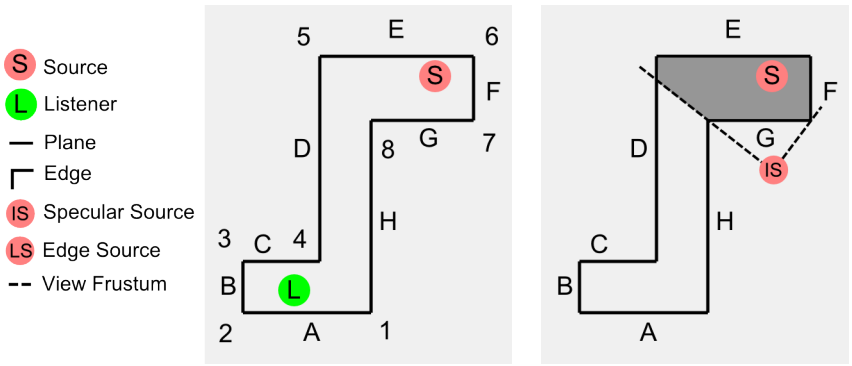
# Chapter 3

# Specular Reflection Modeling

In this chapter, we present efficient geometric techniques to model specular reflections. We present the connection between from-point visibility and specular reflection modeling. We also provide details on two geometric sound propagation algorithms, AD-Frustum and FastV, developed in the thesis to efficiently model specular reflections. *AD-Frustum* models specular reflections for interactive applications and may miss a few specular reflection paths while *FastV* accurately models specular reflections for engineering applications but may not be fast enough for interactive applications.

## 3.1 Geometric Acoustics and Visibility

The image source method [Allen and Berkley, 1979] is a general method to model specular reflections. We briefly give an intuition on the connection between accelerating the image source method and from-point visibility algorithms (see Figure 3.1). Given a point sound source, a CAD model with acoustic material properties, and a listener position, our goal is to compute an impulse response (IR) of the acoustic space for the source and listener position. In Figure 3.1(a), a CAD model (shown in top down view) consists of specular planes A to H. The source and listener positions are also shown. To compute the IR, we need to compute all the specular paths that reach the listener from the source. Consider a source $S$ and a user-specified $k$ orders of reflection. Note that




(a) (b)

Figure 3.1: Accelerating the image-source method by using from-point visibility. *(a) A top down view of a simple configuration with plane primitives A-H and edge primitives 1-8. The source and the listener are also shown. (b) Consider the image source corresponding to the source due to specular reflection from Plane G. Only the planes visible from the image source IS need to be considered to compute second order reflection through plane G.*

we only need to compute image sources for a source (or image source) $S$ with respect to the triangles that are visible to $S$. If $S$ is a point source, this involves from-point visibility computation. For example, consider the image source, *IS* of the source $S$ reflected about plane G in Figure 3.1(b); we need to compute only the image sources of *IS* about planes D, E, and F for the higher order specular reflection from *IS*. Any specular path from the source that reflects off plane G can then only hit planes D, E, and F for their second specular bounce. Thus, considering other planes for computing image sources for IS is unnecessary. The visibility algorithms are applied recursively for point sources. A detailed presentation of accelerated image source method which can model specular reflections as well as edge diffraction is provided in Section 4.2.

## 3.2 AD-Frustum: Adaptive Frustum Tracing

We present an approach to perform interactive geometric sound propagation in complex and dynamic scenes. In this thesis, we focus on computing paths for specular reflections up to a user-specified order of reflections from each source to the receiver. Our approach

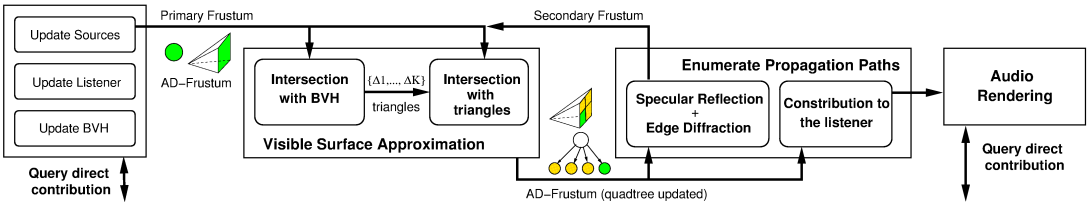



Figure 3.2: Overview of AD-Frustum technique. *AD-Frusta are generated from sound sources (primary frusta) and by reflection and diffraction (secondary frusta) from the scene primitives. Approximate visible surfaces are then computed for each frustum (quadtree update). Next, each updated frustum checks if the listener lies inside it and is visible. If visible, its contributions are registered with the audio rendering system. The audio rendering system queries the direct contribution of a sound source every time it renders its audio block.*

has been extended to model edge diffraction [Chandak et al., 2008] based on Uniform Theory of Diffraction (UTD). Accurate geometric propagation techniques which find all specular paths are compute intensive and may not achieve interactive performance. Therefore, we present an approximate volumetric tracing algorithm which allows small errors in computing specular reflection paths for higher computational efficiency.

### 3.2.1 An Overview

Figure 3.2 gives a top level view of our algorithm. We start by shooting frusta from the sound sources. A frustum traverses the scene hierarchy and finds a list of potentially intersecting triangles. These triangles are intersected with the frustum and the frustum is adaptively sub-divided into sub-frusta. The sub-frusta approximate which triangles are visible to the frustum. The sub-frusta are subsequently reflected and diffracted to generate more frusta, which in turn are traversed. Also, if the listener is inside a frustum and visible, the contribution is registered for use during the audio processing stage.

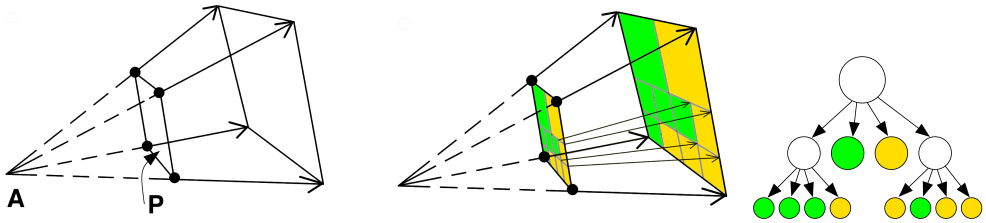
Our approach builds on the use of a ray-frustum [Lauterbach et al., 2007b] for volumetric tracing. The ray-frustum approach traces the paths from the source to the receivers by tracing a sequence of ray-frusta. Each ray-frustum is a simple 4-sided frus-

tum, represented as a convex combination of four corner rays. Instead of computing an exact intersection of the frusta with a primitive, ray-frustum tracing performs discrete clipping by intersecting a fixed number of rays for each frustum. This approach can handle complex, dynamic scenes, but has the following limitations: (a) the formulation uses uniformly spaced samples inside each frustum for fast intersection computations. Using a high sampling resolution can significantly increase the number of traced frusta and (b) the approach cannot adapt to the scene complexity efficiently. In order to overcome these limitations, we propose an adaptive frustum representation, AD-Frustum. Our goal is to retain the performance benefits of the original frustum formulation, but increase the accuracy of the simulation by adaptively varying the resolution.

### 3.2.2 AD-Frustum representation

AD-Frustum is a hierarchical representation of the subdivision of a frustum. We augment a 4-sided frustum with a quadtree structure to keep track of its subdivision and maintain the correct depth information, as shown in Figure 3.3. Each leaf node of the quadtree represents the finest level sub-frustum that is used for volumetric tracing. An intermediate node in the quadtree corresponds to the parent frustum or an internal frustum. We adaptively refine the quadtree in order to perform accurate intersection computations with the primitives in the scene and generate new frusta based on reflections and diffraction.

**Representation:** Each AD-Frustum is represented using an apex and a quadtree. Each 2D node of the quadtree includes: *(a) corner rays* of the sub-frustum that define the extent of each sub-frustum; *(b) intersection status* corresponding to completely-inside, partially-intersecting with some primitive, or completely-outside of all scene primitives; *(c) intersection depth*, and *primitive id* to track the depth value of the closest primitive; *(d) list of diffracting edges* to support diffraction calculations. We also associate a *maximum-subdivision depth parameter* with each AD-Frustum that corresponds to the




(a) (b)

Figure 3.3: AD-Frustum representation. *(a) A frustum, represented by convex combination of four corner rays and apex A. (b) A hierarchically divided adaptive frustum. It is augmented with a quadtree structure, where P is the 2D plane of quadtree. We use two sets of colors to show different nodes. Each node stores auxiliary information about corresponding sub-frusta.*

maximum depth of the quadtree.

### 3.2.3 Intersection Tests

The AD-Frusta are used for volumetric tracing and enumerating the propagation paths. The main operation is computing their intersection with the scene primitives and computing reflection and diffraction frusta. The scene is represented using a bounding volume hierarchy (BVH) of axis-aligned bounding boxes (AABBs). The leaf nodes of the BVH are triangle primitives and the intermediate nodes represent an AABB. For dynamic scenes, the BVH is updated at each frame [Lauterbach et al., 2006]. Given an AD-Frustum, we traverse the BVH from the root node to perform these tests.

**Intersection with AABBs**: The intersection of an AD-Frustum with an AABB is performed by intersecting the four corner rays of the root node of the quadtree with the AABB, as described in [Reshetov et al., 2005]. If an AABB partially or fully overlaps with the frustum, we apply the algorithm recursively to the children of the AABB. The quadtree is not modified during this traversal.

**Intersection with primitives**: As a frustum traverses the BVH, we compute intersections with each triangle corresponding to the leaf nodes of the BVH. We only perform these tests to classify the node as *completely-inside*, *completely-outside*, or *partially-*

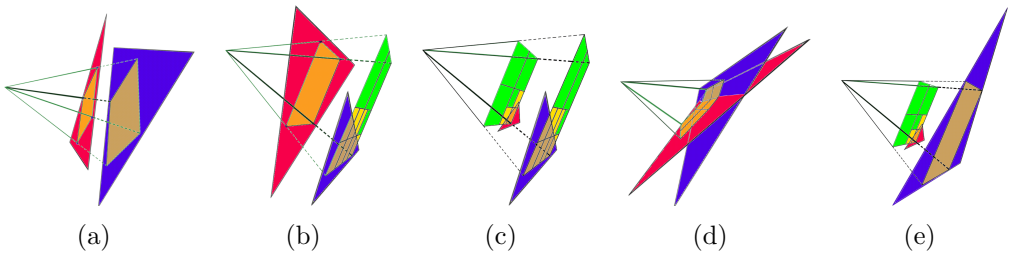



Figure 3.4: Visibility computation based on area subdivision. *We highlight different cases that arise as we resolve visibility between two triangle primitives (shown in red and blue) from the apex of the frustum. In* ***cases (a)-(b)*** *the quadtree is refined based on the intersection depth associated with the outermost frustum and compared with the triangles. However, in* ***cases (c)-(e)*** *the outermost frustum has to be sub-divided into sub-frusta and each of them is compared separately.*

*intersecting*. This is illustrated in Figure 3.4, where we show the completely-outside regions in green and completely-inside regions in orange. This intersection test can be performed efficiently by using the Plücker coordinate representation [Shoemake, 1998] for the triangle edges and four corner rays. The Plücker coordinate representation efficiently determines, based on an assumed orientation of edges, whether the sub-frustum is completely inside, outside, or partially intersecting the primitive. This test is conservative and may conclude that a node is partially intersecting, even if it is completely-outside. If the frustum is classified as partially-intersecting with the primitives, we sub-divide that quadtree node, generate four new sub-frusta, and perform the intersection test recursively. The maximum-subdivision depth parameter imposes a bound on the depth of the quadtree. At maximum-subdivision depth, the partially intersecting nodes are classified as completely-inside and thus, each leaf node of the quadtree is classified either completely-outside or completely-inside.

### 3.2.4 Visible Surface Approximation

A key component of the traversal algorithm is the computation of the visible primitives associated with each leaf node sub-frustum. We use an area-subdivision technique, similar to the classic Warnock's algorithm [Warnock, 1969], to compute the visible prim-

itives. Our algorithm associates intersection depth values of four corner rays with each leaf node of the quadtree as well as the id of the corresponding triangle. Moreover, we compute the minimum and maximum depth for each intermediate node of the triangle, that represents the extent of triangles that partially overlap with its frustum. The depth information of all the nodes is updated whenever we perform intersection computations with a new triangle primitive.

In order to highlight the basic subdivision algorithm, we consider the case of two triangles in the scene shown as red and blue in Figure 3.4. In this example, the projections of the triangles on the quadtree plane overlap. We illustrate different cases that can arise based on the relative ordering and orientation of two triangles. The basic operation compares the depths of corner rays associated with the frustum and updates the node with the depth of the closer ray (as shown in Figure 3.4(a)). If we cannot resolve the closest depth (see Figure 3.4(d)), we apply the algorithm recursively to its children (shown in orange). Figure 3.4(b) shows the comparison between a partially-intersecting node with a completely-inside node. If the completely-inside node is closer, i.e., all the corner rays of the completely-inside node are closer than the minimum depth of the corner rays of the partially-intersecting node, then the quadtree is updated with the completely-inside node. Otherwise, we apply the algorithm recursively to their children as in Figure 3.4(e). Lastly, in Figure 3.4(c), both the nodes are partially-intersecting and we apply the algorithm recursively on their children.

This approach can be easily generalized to handle all the triangles that overlap with an AD-Frustum. At any stage, the algorithm maintains the depth values based on intersection with all the primitives traversed so far. As we perform intersection computations with a new primitive, we update the intersection depth values by comparing the previous values stored in the quadtree. The accuracy of our algorithm is governed by the resolution of the leaf nodes of the quadtree, which is based on the maximum-subdivision depth parameter associated with each AD-Frustum.

**Nodes Reduction**: We perform an optimization step to reduce the number of frusta. This step is performed in a bottom up manner, after an AD-Frustum finishes the scene traversal. We look at the children of a node. Since, each child shares at least one corner-ray with its siblings, we compare the depths of these corner-rays. Based on the difference in depth values, the normals of the surfaces the sub-frustum intersects, and the acoustic properties of those surfaces we collapse the children nodes into the parent node. Thus, we can easily combine the children nodes in the quadtree that hit the same plane with similar acoustic properties. Such an approach can also be used to combine children nodes that intersect slightly different surfaces.

## 3.3 Sound Propagation using AD-Frustum

In the previous section, we described the AD-Frustum representation and presented an efficient algorithm to compute its intersection and visibility with the scene primitives. In this section, we present algorithms to trace the frusta through the scene, and compute reflection frusta. Note that AD-Frustum is a standalone from-point visibility algorithm. We present an instance of using a from-point visibility algorithm tightly integrated into the geometric sound propagation algorithm. We start the simulation by computing initial frusta around a sound source in quasi-uniform fashion [Ronchi et al., 1996] and perform adaptive subdivision based on the intersection tests. Ultimately, the sub-frusta corresponding to the leaf nodes of the quadtree are used to compute reflection and diffraction frusta.

**Specular Reflections**: Once a frustum has completely traced a scene, we consider all the leaf nodes within its quadtree and compute a reflection frustum for all completely-intersecting leaf nodes. The corner rays of sub-frustum associated with the leaf nodes are reflected at the primitive hit by that sub-frustum. The convex combination of the reflected corner rays creates the parent frustum for the reflection AD-Frustum.

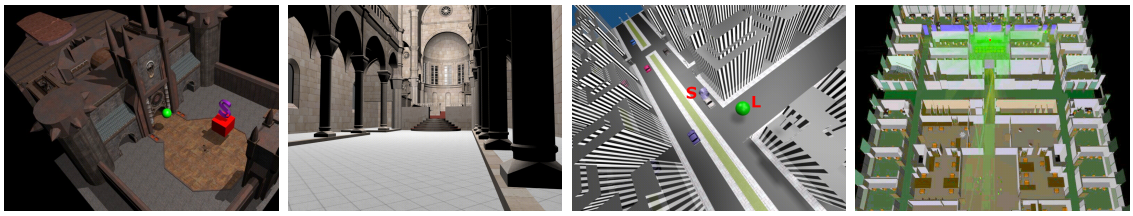
### 3.3.1 Contributions to the Listener

For the specular reflections the actual contribution of a frustum is determined by computing if the receiver is inside a frustum. If the receiver is inside a frustum, the receiver is projected onto the primitive from which the frustum was reflected. This step is performed recursively to compute a specular path to the source from the receiver. Once a path is computed, its delay and attenuation due to its path length, absorption at the reflection surfaces, and attenuation due to air absorption is taken into account [Kuttruff, 1991].

### 3.3.2 Implementation and Results

In this section, we highlight the performance of AD-Frustum on different benchmarks. Our simulations were run on a 2.66 GHz Intel Core 2 Duo machine with 2GB of memory. We perform geometric sound propagation using adaptive frustum tracing on many benchmarks. The complexity of our benchmarks ranges from a few hundred triangles to a million triangles (see Figure 3.22). In Figure 3.6, we show how the computation time for our approach and the number of frusta traced scale as we increase the maximum sub-division depth of AD-Frusta. Also, our algorithm scales well with the number of cores as shown in Figure 3.7. The comparison of our approach with uniform frustum tracing [Lauterbach et al., 2007b] is given in Table 3.1. We chose a sampling resolution of $2^3 \times 2^3$ for uniform frustum tracing and maximum sub-division depth of 3 for adaptive frustum tracing for the purposes of comparisons.

The results of modeling specular reflections using AD-Frustum are presented in Table 3.2. These results are generated for a maximum sub-division depth of 3. Further, the extent of dynamism in these benchmarks is shown in the associated videos. Table 3.3 presents results for different orders of reflection and maximum sub-division depths.



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

Figure 3.5: Benchmarks for AD-Frustum technique. *(a) Game Model with low geometric complexity. It has dynamic objects and a moving listener. (b) Sibenik Cathedral, a complex architectural model with a lot of details and curved geometry. It consists of a moving source and a listener, and a door that opens and closes. (c) City Scene with many moving cars (dynamic objects) and tall buildings. It has a moving sound source, a static sound source, and a listener. (d) Soda Hall, a complex architectural model with multiple floors. The dimensions of a room are dynamically modified and the sound propagation paths recomputed for this new room using AD-Frusta. This scenario demonstrates the potential of our approach for conceptual acoustic design.*

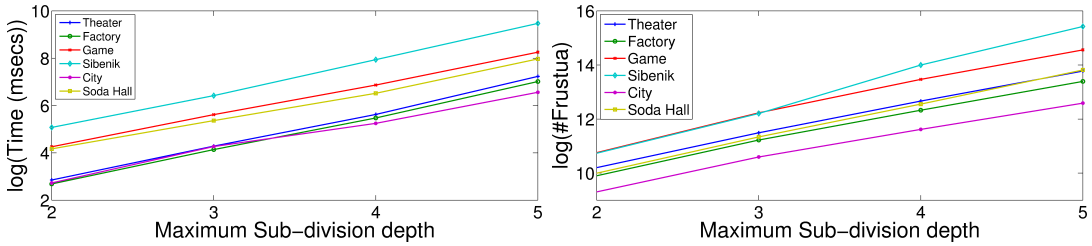



Figure 3.6: Effect of sub-division depth on performance. *This figure shows the effect of increasing the maximum sub-division depth of adaptive frustum tracing on the overall computation time of the simulation and the number of total frusta traced. Different colors are used for different benchmarks.*

### 3.3.3 Accuracy Speed Trade-off

In this section, we give a brief overview of how to govern the accuracy of our algorithm for complex scenes. The complexity of the AD-Frustum tracing algorithm described in Sections 3.2 and Section 3.3 varies as a function of the maximum-subdivision depth parameter associated with each AD-Frustum. A higher value of this parameter results in a finer subdivision of the quadtree and improves the accuracy of the intersection and volumetric tracing algorithms (see Fig. 3.8). At the same time, the number of leaf nodes can potentially increase as an exponential function of this parameter and significantly increase the number of traced frusta through the scene. Here, we present techniques for

| Model | Frusta traced | | Frusta Gain | Time Gain |
|---|---|---|---|---|
| | UFT | AFT | | |
| Theater | 404K | 56K | 7.2 X | 6.1 X |
| Factory | 288K | 40K | 7.2 X | 5.7 X |
| Game | 2330K | 206K | 11.3 X | 9.0 X |
| Sibenik | 6566K | 198K | 33.2 X | 21.9 X |
| City | 377K | 78K | 4.9 X | 5.2 X |
| Soda Hall | 773K | 108K | 7.2 X | 9.8 X |

Table 3.1: Adaptive Frustum Tracing vs. Uniform Frustum Tracing. *This table presents a preliminary comparison of the number of frusta traced and time taken by Adaptive Frustum Tracing and Uniform Frustum Tracing [Lauterbach et al., 2007b] for results with similar number of specular paths. Our adaptive frustum formulation significantly improves the performance as compared to prior approaches.*

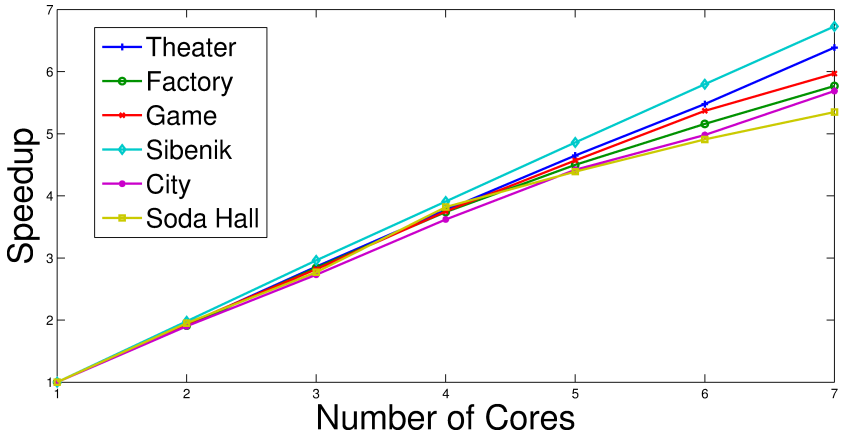



Figure 3.7: Multi-core scaling of AD-Frustum. *This figure shows that our algorithm scales well with the number of cores. It makes our approach favorable for parallel and many-core platforms. Reflections for up to 4th order were computed and larger of maximum sub-division depths similar to those in Table 3.3 were used.*

automatic computation of the depth parameter for each AD-Frustum. We consider two different scenarios: capturing the contributions from all important objects in the scene and constant frame-rate rendering.

Our basic algorithm is applicable to all triangulated models. We do not make any assumptions about the connectivity of the triangles and can handle all polygonal soup datasets. Many times, a scene consists of objects that are represented using connected triangles. In the context of this paper, an object corresponds to a collection of triangles that are very close (or connected) to each other.

| Model | #Δs | Results (7 threads) | |
|---|---|---|---|
| | | #frusta | time |
| Theater | 54 | 56K | 33.3 ms |
| Factory | 174 | 40K | 27.3 ms |
| Game | 14K | 206K | 273.3 ms |
| Sibenik | 71K | 198K | 598.6 ms |
| City | 78K | 80K | 206.2 ms |
| Soda Hall | 1.5M | 108K | 373.3 ms |

Table 3.2: This table summarizes the performance of our system on six benchmarks. The complexity of a model is given by the number of triangles. We use a value of 3 for the maximum sub-division depth for these timings. The results are computed for up to 4 orders of specular reflections.

### Maximum-subdivision depth computation

One of the main challenges is to ensure that our algorithm doesn't miss the important contributions in terms of specular reflections. We take into account some of the characteristics of geometric sound propagation in characterizing the importance of different objects. For example, many experimental observations have suggested that low-resolution geometric models are more useful for specular reflections [Tsingos et al., 2007, Joslin and Magnenat-Thalmann, 2003]. Similarly, small objects in the scene only have significant impact at high frequencies and can be excluded from the models in the presence of other significant sources of reflection and diffraction [Funkhouser et al., 1998]. Based on these characterizations, we pre-compute geometric simplifications of highly tessellated small objects and assign an importance function to each object based on its size and material properties.

Given an object ($O$) with its importance function, our goal is to ensure that the approximate frustum-intersection algorithm doesn't miss contributions from that object. Given a frustum with its apex ($A$) and the plane of its quadtree ($P$), we compute a perspective projection of the bounding box of $O$ on the $P$. Let's denote the projection on $P$ as $\mathbf{o}$. Based on the size of $\mathbf{o}$, we compute a bound on the size of leaf nodes of the quadtree such that the squares corresponding to the leaf nodes are inside $o$. If $O$ has

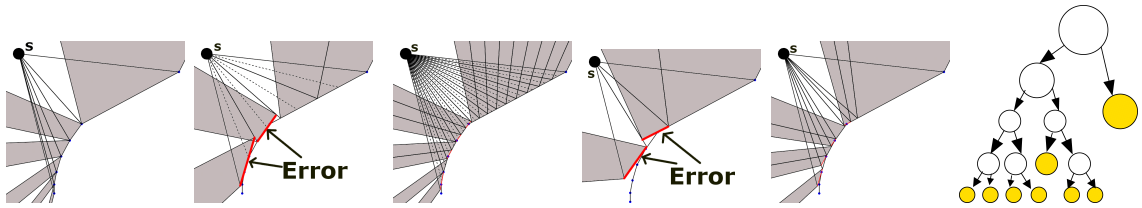
| Model | Tris | Order of Reflection (Running time in msec) | | | | Maximum sub-division depth |
|---|---|---|---|---|---|---|
| | | 1st | 2nd | 3rd | 4th | |
| Theater | 54 | 4.7 | 8.0 | 16.0 | 30.0 | 2 |
| | | 24 | 77 | 201 | 462 | 4 |
| Factory | 174 | 3.3 | 6.7 | 12.7 | 24.7 | 2 |
| | | 18 | 64 | 178 | 406 | 4 |
| Game Scene | 14K | 20 | 46 | 120 | 270 | 2 |
| | | 51 | 167 | 469 | 1134 | 3 |
| Sibenik | 71K | 34 | 83 | 250 | 694 | 2 |
| | | 67 | 236 | 864 | 2736 | 3 |
| City | 72K | 6.7 | 12 | 23 | 44 | 2 |
| | | 18 | 38 | 85 | 174 | 3 |
| Soda Hall | 1.5M | 21 | 38 | 98 | 229 | 2 |
| | | 35 | 102 | 297 | 723 | 3 |

Table 3.3: AD-Frustum performance for different sub-division depths. *This table shows the performance of adaptive frustum tracing on a single core for different maximum sub-division depths. The timings are further broken down according to order of reflection. Our propagation algorithm achieves interactive performance for most benchmarks.*

high importance values attached to it, we ensure multiple leaf nodes of the quadtree are contained in **o**. We repeat this computation for all objects with high importance value in the scene. The maximum-subdivision depth parameter for that frustum is computed by considering the minimum size of the leaf nodes of the quadtree over all objects. Since we use a bounding box of $O$ to compute the projection, our algorithm tends to be conservative.

## Constant frame rate rendering

In order to achieve target frame rate, we can bound the number of frusta that are traced through the scene. We first estimate the number of frusta that our algorithm can trace per second based on scene complexity, number of dynamic objects and sources. Given a bound on maximum number of frusta traced per second, we adjust the number of primary frusta that are traced from each source. Moreover, we use a larger value of the maximum depth parameter for the first few reflections and decrease this value




(a) (b) (c) (d) (e) (f)

Figure 3.8: Beam tracing vs. uniform frustum tracing vs. adaptive frustum tracing. *We show the relative accuracy of three different algorithm in this simple 2D model with only first-order reflections. (a) Reflected beams generated using exact beam tracing. (b)-(c) Reflected frusta generated with uniform frustum tracing for increased sampling:* $2^2$ *samples per frustum and* $2^4$ *samples per frustum. Uniform frustum tracing generates uniform number of samples in each frustum, independent of scene complexity. (d)-(e)Reflected frusta generated using AD-Frustum. We highlight the accuracy of the algorithm by using the values of* 2 *and* 4 *for maximum-subdivision depth parameter. (f) An augmented binary tree for the 2D case (equivalent to a quadtree in 3D) showing the adaptive frusta. Note that the adaptive algorithm only generates more frusta in the regions where the input primitives have a high curvature and reduces the propagation error.*

for higher order reflections. We compute the first few reflections with higher accuracy by performing more adaptive subdivisions. We also perform adaptive subdivisions of a frustum based on its propagated distance. In this manner, our algorithm would compute more contributions from large objects in the scene, and tend to ignore contributions from relatively small objects that are not close to the source or the receiver. We can further improve the performance by using dynamic sorting and culling algorithms along with perceptual metrics [Tsingos et al., 2004].

### 3.3.4 Analysis and Comparison

In this section, we analyze the performance of our algorithm and compare it with other geometric propagation techniques. The running time of our algorithm is governed by three main factors: model complexity, level of dynamism in the scene, and the relative placement of objects with respect to the sources and receiver. As part of a pre-process, we compute a bounding volume hierarchy of AABBs. This hierarchy is updated as some objects in the scene move or some objects are added or deleted from the scene. Our

current implementation uses a linear time refitting algorithm that updates the BVH in a bottom-up manner. If there are topological changes in the scene (e.g. explosions), than the resulting hierarchy can have poor culling efficiency and can result in more intersection tests [Wald et al., 2007b]. The complexity of each frustum intersection test is almost logarithmic in the number of scene primitives and linear in the number of sub-frusta generated based on adaptive subdivision. The actual number of frusta traced also vary as a function of number of reflections as well as the relative orientation of the objects.

### Comparison

The notion of using an *adaptive technique* for geometric sound propagation is not novel. There is extensive literature on performing adaptive supersampling for path or ray tracing in both sound and visual rendering. However, the recent work in interactive ray tracing for visual rendering has shown that adaptive sampling, despite its natural advantages, does not perform as fast as simpler approaches that are based on ray-coherence [Wald et al., 2007b]. On the other hand, we are able to perform fast intersection tests on the AD-Frusta using ray-coherence and the Plücker coordinate representation. By limiting our formulation to 4-sided frusta, we are also able to exploit the SIMD capabilities of current commodity processors.

Many adaptive volumetric techniques have also been proposed for geometric sound propagation. Shinya et al. [Shinya et al., 1987] traces a pencil of rays and require that the scene has smooth surfaces and no edges, which is infeasible as most models of interest would have sharp edges. Rajkumar et al. [1996] [Rajkumar et al., 1996] used a static BSP tree structure and the beam starts out with just one sample ray and is subdivided only at primitive intersection events. This can result into sampling problems due to missed scene hierarchy nodes. Drumm and Lam [Drumm and Lam, 2000] describe an adaptive beam tracing algorithm, but it is not clear whether it can handle

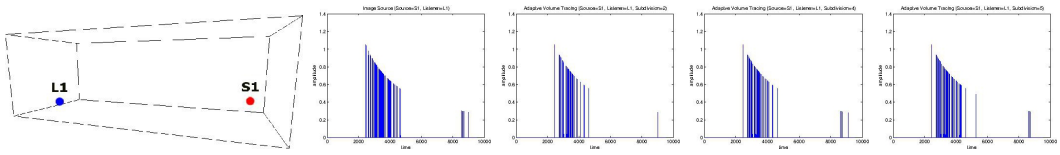
complex, dynamic scenes. The volume tracing formulation [Genetti et al., 1998] shoots pyramidal volumes and subdivides them in case they intersect with some object partially. This approach has been limited to visual rendering and there may be issues in combining this approach with a scene hierarchy. The benefits over the ray-frustum approach [Lauterbach et al., 2007b, Lauterbach et al., 2007a] are shown in Figure 3.8. The propagation based on AD-Frustum traces fewer frusta.

In many ways, our adaptive volumetric tracing offers contrasting features as compared to ray tracing and beam tracing algorithms. Ray tracing algorithms can handle complex, dynamic scenes and can model diffuse reflections and refraction on top of specular reflection and diffraction. However, these algorithms need to perform very high supersampling to overcome noise and aliasing problems, both spatially and temporally. For the same accuracy, propagation based on AD-Frustum can be much faster than ray tracing for specular reflections and edge diffraction.

The beam tracing based propagation algorithms are more accurate as compared to our approach. Recent improvements in the performance of beam tracing algorithms [Overbeck et al., 2007, Laine et al., 2009] are promising and can make them applicable to more complex static scenes with few tens of thousands of triangles. However, the underlying complexity of performing exact clipping operations makes beam tracing more expensive and complicated. In contract, AD-Frustum compromises on the accuracy by performing discrete clipping with the 4-sided frustum. Similarly, image-source methods are rather slow for interactive applications.

### 3.3.5 Accuracy Analysis

Our approach is an approximate volume tracing approach and we can control its accuracy by varying the maximum-subdivision depth of each AD-Frustum. In order to evaluate the accuracy of propagation paths, we compare the impulse responses computed by our algorithm with other geometric propagation methods. We are not aware of any




(a) Shoebox benchmark (b) 63 contributions (reference) (c) 40 contributions (d) 49 contributions (e) 58 contributions

Figure 3.9: Shoebox validation: Configuration 1. *(a) Shoebox benchmark with 12 triangles. (b) Impulse response generated by image source method for 3 reflections. (c)–(e) Impulse responses generated by our approach with maximum sub-division depth of 2, 4, and 5 respectively for 3 reflections.*

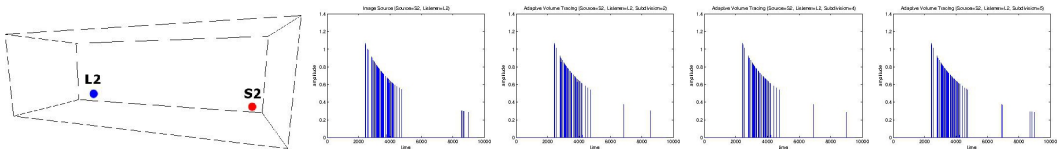
public or commercial implementation of beam tracing which can handle complex scenes with dynamic objects highlighted in Table 3.2. Rather, we use an industry strength implementation of the image source method available as part of the CATT-Acoustic$^{TM}$ software. We compare our results for specular reflection on various benchmarks available with CATT software. The results show that our method gives more accurate results with higher maximum sub-division depths.

### SHOEBOX

We randomly pick two source–listener positions from the ones available in CATT–Acoustic$^{TM}$ for the shoebox model. The two configurations are shown in Figure 3.9(a) and Figure 3.10(a). We then compare IRs generated by the image source method with AD-Frusta with increasing maximum sub-division depth. The results are summarized in Figure 3.9 and Figure 3.10. With increasing levels of maximum sub-division depths, the number of specular paths found by our techniques approaches closer to maximum number of specular paths found by the reference solution.

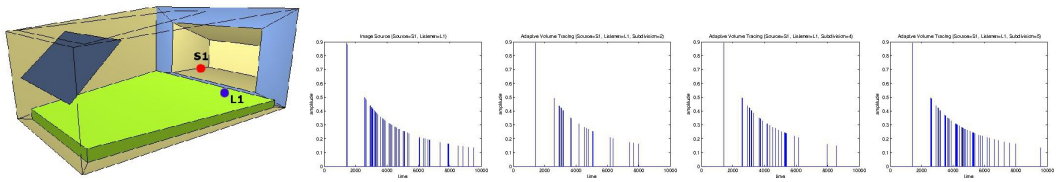
### THEATER

We repeat the same steps as earlier with the theater benchmark shown in its two configurations in Figure 3.11(a) and Figure 3.12(a). The results for configurations 1 and 2




(a) Shoebox benchmark (b) 63 contributions (reference) (c) 41 contributions (d) 43 contributions (e) 56 contributions

Figure 3.10: Shoebox validation: Configuration 2. *(a) Shoebox benchmark with 12 triangles. (b) Impulse response generated by image source method for 3 reflections. (c)–(e) Impulse responses generated by our approach with maximum sub-division depth of 2, 4, and 5 respectively for 3 reflections.*



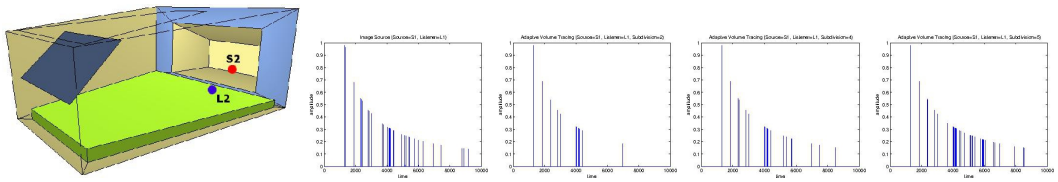

(a) Theater benchmark (b) 65 contributions (reference) (c) 21 contributions (d) 33 contributions (e) 60 contributions

Figure 3.11: Theater validation: Configuration 1. *(a) Theater benchmark with 54 triangles. (b) Impulse response generated by image source method for 3 reflections. (c)–(e) Impulse responses generated by our approach with maximum sub-division depth of 2, 4, and 5 respectively for 3 reflections.*

are summarized in Figure 3.11 and Figure 3.12 respectively. With increasing levels of maximum sub-division depths, the number of specular paths found by our techniques approaches closer to maximum number of specular paths found by the reference solution.

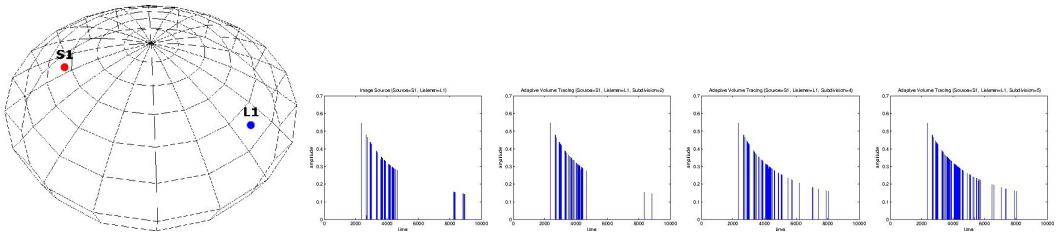
### DOME

The two configurations for the dome benchmark are shown in Figure 3.13(a) and Figure 3.14(a). This benchmark has a curved shape and the CAD model of a polygonal approximation is represented as 316 triangles. Notice that the vertex at the top of the dome has high order of connectivity. A similar vertex also exist at one corner of the floor. Such high order vertices will require a very high subdivision to get accurate results and in absence over-estimation is possible. The results for the configurations 1 and 2




(a) Theater benchmark (b) 44 contributions (reference) (c) 13 contributions (d) 20 contributions (e) 40 contributions

Figure 3.12: Theater validation: Configuration 2. *(a) Theater benchmark with 54 triangles. (b) Impulse response generated by image source method for 3 reflections. (c)–(e) Impulse responses generated by our approach with maximum sub-division depth of 2, 4, and 5 respectively for 3 reflections.*



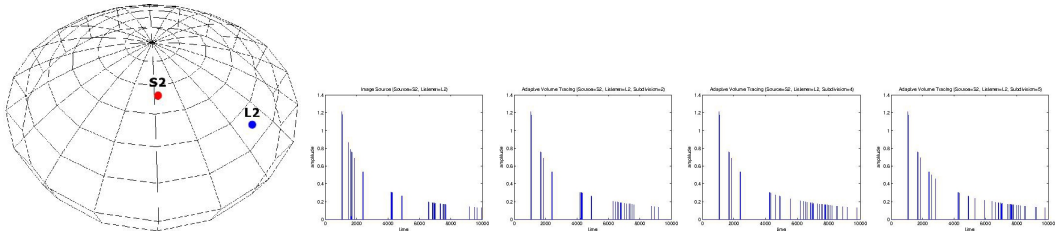

(a) Dome benchmark (b) 138 contributions (reference) (c) 75 contributions (d) 103 contributions (e) 141 contributions

Figure 3.13: Dome validation: Configuration 1. *(a) Dome benchmark with 316 triangles. (b) Impulse response generated by image source method for 3 reflections. (c)–(e) Impulse responses generated by our approach with maximum sub-division depth of 2, 4, and 5 respectively for 3 reflections.*

for the dome benchmark are shown in Figure 3.13 and 3.14 respectively.

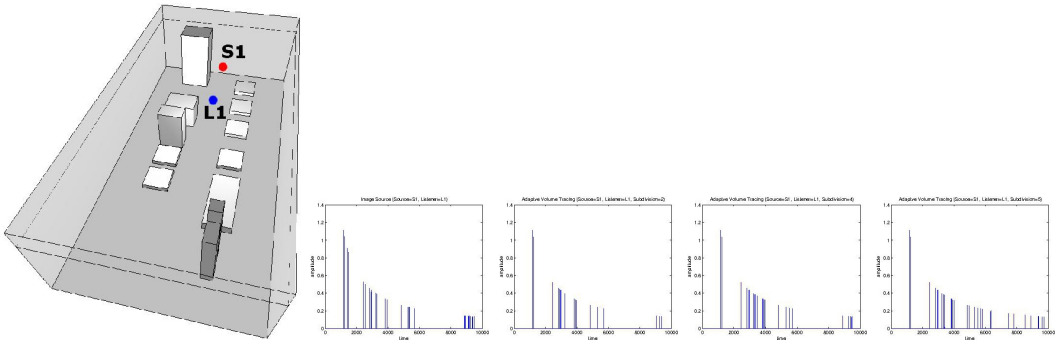
### FACTORY

We pick one configuration for the factory benchmark (174 triangles) as shown in Figure 3.15(a). The results are summarized in Figure 3.15. With increasing levels of maximum sub-division depths, the number of specular paths found by our techniques approaches closer to maximum number of specular paths found by the reference solution.




(a) Dome benchmark (b) 51 contributions (reference) (c) 33 contributions (d) 47 contributions (e) 53 contributions

Figure 3.14: Dome validation: Configuration 2. *(a) Dome benchmark with 316 triangles. (b) Impulse response generated by image source method for 3 reflections. (c)–(e) Impulse responses generated by our approach with maximum sub-division depth of 2, 4, and 5 respectively for 3 reflections.*




(a) Factory benchmark (b) 40 contributions (reference) (c) 23 contributions (d) 32 contributions (e) 41 contributions

Figure 3.15: Factory validation: Configuration 1. *(a) Factory benchmark with 174 triangles. (b) Impulse response generated by image source method for 3 reflections. (c)–(e) Impulse responses generated by our approach with maximum sub-division depth of 2, 4, and 5 respectively for 3 reflections.*

### 3.3.6 Conclusion, Limitations, and Future Work

We presented an adaptive volumetric tracing for interactive sound propagation based on transmissions and specular reflections in complex scenes. Our approach uses a simple, adaptive frustum representation that provides a balance between accuracy and interactivity. Our approach can handle complex, dynamic benchmarks with tens or hundreds of thousands of triangles. The initial results seem to indicate that our approach may be able to generate plausible sound rendering for interactive applications such as conceptual

acoustic design, video games and urban simulations. Our approach maps well to the commodity hardware and empirical results indicate that it may scale linearly with the number of cores. We verify the accuracy of our approach by comparing the performance on simple benchmarks with commercial state-of-the-art software.

**Limitations**: Our approach has many limitations. We have already addressed the accuracy issue above. Our formulation cannot directly handle diffuse, lambertian or "glossy" reflections. Moreover, it is limited to point sources though we can potentially simulate area and volumetric sources if they can be approximated with planar surfaces. Many of the underlying computations such as maximum-subdivision depth parameter and intersection test based on Plücker coordinate representation are conservative. As a result, we may generate unnecessary sub-frusta for tracing.

**Future Work**: There are many avenues for future work. It may be possible to use visibility culling techniques to reduce the number of traced frusta. We can refine the criterion for computation of the maximum-subdivision depth parameter based on perceptual metrics. Finally, we would like to extend the approach to handle more complex environments and evaluate its use on applications that can combine interactive sound rendering with visual rendering.

## 3.4 FastV: From-Point Object-Space Conservative Visibility

In the previous section, we described AD-Frustum technique which can model specular reflections on complex scenes for interactive applications. However, there may be errors in specular paths computed by AD-Frustum as a frustum can only be sub-divided upto a maximum sub-division depth. Such errors may not be acceptable for engineering applications and the existing methods to accurately compute all specular paths are highly inefficient. In this section, we present our conservative from-point visibility algorithm

which can be used to efficiently and accurately compute all specular paths in complex scenes.

### 3.4.1 Overview

The inputs to our from-point visibility algorithm are: a view point ($\mathbf{v} \in \Re^3$), a set of scene primitives ($\Pi$), and a viewing frustum ($\Phi$), with an apex at $\mathbf{v}$. Our goal is to compute a subset of primitives $\pi \subseteq \Pi$ such that every primitive $p \in \Pi$, which is hit by some ray $r \in \Phi$ is included in the computed subset $\pi$. The subset $\pi$ is called the potentially visible set (PVS). The smallest such PVS is the set of exactly visible primitives ($\pi_{exact}$). The subset $\pi$ computed by our algorithm is conservative, i.e., $\pi \supseteq \pi_{exact}$. For the rest of the paper, we assume that the primitives are triangles, though our algorithm can be modified to handle other primitives. We also assume that the connectivity information between the scene triangles is precomputed. We exploit this connectivity for efficient computation; however our approach is also applicable to polygon soup models with no connectivity. In order to perform fast intersection tests, we store the scene primitives in a bounding volume hierarchy (BVH) of axis-aligned bounding boxes (AABBs). This hierarchy is updated for dynamic scenes.

We trace pyramidal or volumetric beams from the viewpoint. Prior beam tracing algorithms perform expensive exact intersection and clipping computations of the beam against the triangles and tend to compute $\pi_{exact}$. Our goal is to avoid these expensive clipping computations, and rather perform simple intersection tests to compute the PVS. Moreover, it is hard to combine two or more non-overlapping occluders (i.e. occluder fusion) using object space techniques. This is shown in Figure 3.16, where object $H_1$ is occluded by the combination of $V_1$ and $V_2$. As a result, prior conservative object space techniques are primarily limited to scenes with large occluders.

We overcome these limitations by tracing a high number of relatively small frusta [Lauterbach et al., 2007b] and computing the PVS for each frustum independently. In

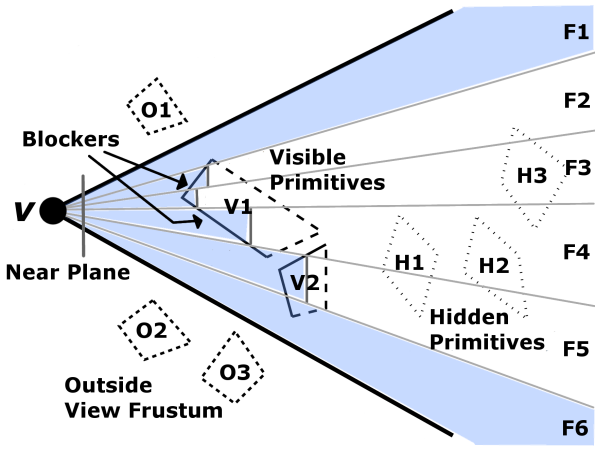



Figure 3.16: Overview of FastV algorithm. *We divide the view-frustum with an apex at* $\boldsymbol{v}$*, into many small frusta. Each frustum is traced through the scene and its far plane is updated when it is blocked by a connected blocker. For example, frustum* $F_5$ *is blocked by primitives of object* $V_2$ *but frustum* $F_1$ *has no blockers.*

order to compute the PVS for each frustum, we compute a *blocker* that is composed of connected triangles (see Figure 3.17). The blockers are computed on the fly and need not correspond to a convex set or a solid object; rather they are objects that are homeomorphic to a disk. We use simple and fast algorithms to perform intersection tests and blocker computation.

### 3.4.2 Frustum Tracing

We use a simple 4-sided frustum, which is represented as a convex combination of four corner rays intersecting at the apex. Each frustum has a near plane, four side planes, and a far plane. The near plane and the four side planes of a frustum are fixed and the far plane is parallel to the near plane. The depth of the far plane from the view point is updated as the algorithm computes a new blocker for a frustum. Our algorithm sub-divides $\Phi$ into smaller frusta using uniform or adaptive subdivision and computes a PVS for each frustum. Eventually, we take the union of these different PVSs to compute a PVS for $\Phi$.

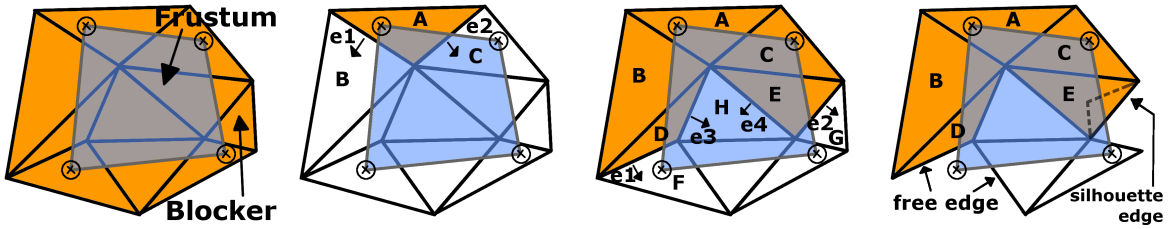
**Algorithm**: The algorithm computes the PVS for each frustum independently. We

initialize the far plane associated with a frustum to be at infinity and update its value if any connected blocker is found. The algorithm traverses the BVH to efficiently compute the triangles that potentially intersect with the frustum. We perform fast Plücker intersection tests between the frustum and a triangle to determine if the frustum is completely inside, completely outside, or partially intersecting the triangle. If the frustum is partially intersecting, we reuse the Plücker test from the frustum-triangle intersection step to quickly find the edges that intersect the frustum (see Section 3.4.4). We perform frustum-triangle intersection with the neighboring triangles that are incident to these edges. We repeat this step of finding edges that intersect with the frustum and performing intersection tests with the triangles incident to the edge till the frustum is completely blocked by some set of triangles. If a blocker is found (see Section 3.4.3), we update the far plane depth of the frustum. Triangles beyond the far plane of the frustum are discarded from the PVS. If there is no blocker associated with the frustum, then all the triangles overlapping with the frustum belong to the PVS. Additionally, we compute the PVS for each frustum in parallel as described in Section 3.4.6.

### 3.4.3 Frustum Blocker Computation

We define a blocker for a frustum as a set of connected triangles such that every ray inside the frustum hits some triangle in the frustum blocker (see Figure 3.17(a)). When we intersect a frustum with a triangle, the frustum could partially intersect the triangle. In such a case, we walk to the neighboring triangles based on that intersection and try to find a blocker for the frustum (see Figure 3.17). We compute all the edges of the triangle that intersect with the frustum. For every intersecting edge, we walk to the neighboring triangle incident to the edge and perform frustum-triangle intersection test with the neighbor triangle.

The intersection and walking steps are repeated until one of the following conditions is satisfied:




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

Figure 3.17: Frustum blocker computation. *(a) Example of a blocker with connected triangles. (b)-(c) Intersection and Walking: Identify intersecting edges (e1, e2, e3, and e4) and walk to the adjacent triangles (denoted by arrows from edge to the triangle). (d) Abort frustum blocker computation if a free-edge or a silhouette-edge is found.*

(a) All triangles incident to every intersecting edge found during the frustum blocker step have been processed. This implies that we have found a blocker.

(b) A *free-edge*, i.e. an edge with only one incident triangle, or a *silhouette edge*, i.e. an edge with incident triangles facing in opposite directions as seen from the viewpoint, intersects with the frustum. In that case, we conclude that the current set of intersecting triangles does not constitute a blocker.

Note that our termination condition (b) for blocker computation is conservative. It is possible that there may exist a frustum blocker with a silhouette edge, but we need to perform additional computations to identify such blockers [Navazo et al., 2003, Laine, 2006]. In this case, we opt for simplicity, and rather search for some other blocker defined by a possibly different set of triangles. Or we subdivide the frustum until the current object becomes a blocker for a smaller sub-frustum.

If we terminate the traversal test due to condition (a), we have successfully found a frustum blocker. All triangles in the frustum blocker are marked visible and the far plane depth associated with the frustum is updated. Note that the depth of the far plane of the frustum is chosen such that all triangles in the frustum blocker lie in front of the far plane. If we terminate due to condition (b), then the algorithm cannot guarantee the existence of a frustum blocker. All triangles processed during this step are still marked

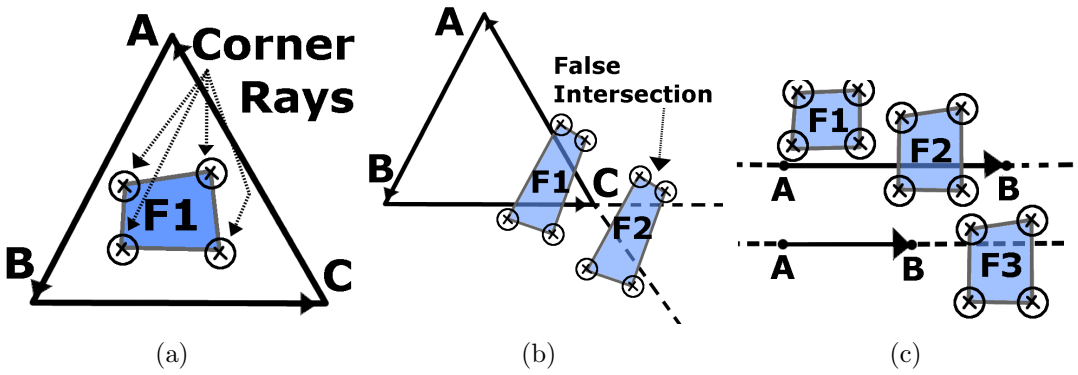
visible but the far plane depth is not updated.

### 3.4.4 Frustum Intersection Tests

A key component of the algorithm is performing the intersection tests of the scene primitives with a frustum. The algorithm traverses the BVH and performs intersection tests between a frustum and the AABBs associated with the BVH. We use the technique proposed by Reshetov et al. [Reshetov et al., 2005] to perform fast intersection tests between a frustum and an AABB. For every leaf node of the hierarchy we perform the intersection test with the frustum and triangle(s) associated with that leaf node. To perform the intersection test efficiently, we represent the corner rays of a frustum and the oriented edges of the triangle using Plücker coordinates [Shoemake, 1998]. The orientation of a ray as seen along the edges of a triangle governs the intersection status of the ray with the triangle (see Figure 3.18(a)). Similarly, the orientation of four corner rays of the frustum as seen along the edges of a triangle governs the intersection status of the frustum with the triangle. We can determine with object-precision accuracy if the frustum is completely inside the triangle, completely outside the triangle, or partially intersects the triangle [Chandak et al., 2008].

In practice, the Plücker test is conservative and it can wrongly classify a frustum to be partially intersecting a triangle even if the frustum is completely outside the triangle (see Figure 3.18(b)). This can affect the correctness of our algorithm as we may wrongly classify an object as a blocker due to conservative intersection tests. We ensure that at least one of the corner rays is inside the blocker to avoid such cases.

**Frustum-Edge Intersection**: When a frustum partially intersects with a triangle, we can quickly determine which edges of the triangle intersect the frustum. We reuse the Plücker test between the frustum and the triangle to find the edges of the triangle that intersect the frustum. As shown in Figure 3.18(c), a frustum intersects with an edge if all four corner rays of the frustum do not have the same orientation as seen




(a) (b) (c)

Figure 3.18: Conservative Plücker tests. *(a) All four corner rays of the frustum $F_1$ have the same orientation as seen along every directed edge of the triangle $ABC$. Thus, $F_1$ is completely-inside $ABC$. (b) Intersection between a frustum and a triangle can be conservative. $F_2$ will be classified as partially intersecting. (c) Different cases of frustum-edge intersections: $F_1$ does not intersect the edge $AB$, $F_2$ intersects $AB$. $F_3$ is falsely classified as intersecting with $AB$.*

along the edge. This test may falsely classify an edge as intersecting even if the frustum does not intersect the edge, as shown in Figure 3.18(c) and thereby make our algorithm conservative. This test is also used in Section 3.4.3 to compute a set of triangles that may block the frustum completely.

**Far Plane Depth Update**: The far plane associated with a frustum is updated whenever a blocker is found. The blocker may correspond to a single triangle or multiple triangles. If a frustum lies completely inside a triangle, the triangle blocks the frustum. We, therefore, mark the triangle as visible and update the depth of the far plane of the frustum as shown in Figure 3.19(a). The frustum intersects the triangle at points $\mathbf{H_1}$ and $\mathbf{H_2}$, and $\mathbf{d_1}$ and $\mathbf{d_2}$ are the projected distances of $|V\mathbf{H_1}|$ and $|V\mathbf{H_2}|$ along the near plane normal. We set the far plane depth of the frustum as the maximum of the projected distances. In other cases, the blocker may be composed of multiple triangles. We update the far plane depth of the frustum as shown in Figure 3.19(b). We compute the far plane depth for every triangle in the frustum blocker, assuming the frustum is completely inside the triangle. In Figure 3.19(b), $d$ and $d'$ are the far plane depths for triangles $T_1$ and $T_1'$, respectively, of the frustum blocker. The far plane depth of the

frustum is set to the maximum of far plane depths computed for the triangles in the frustum blocker, which is $d'$ in this case.

### 3.4.5 Frustum Subdivision

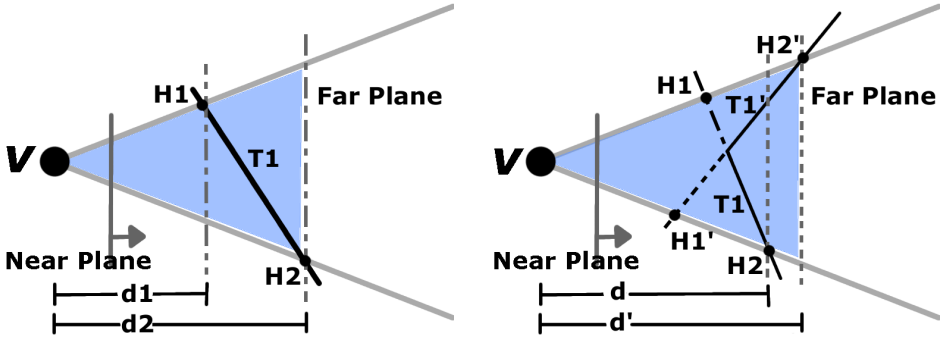



Figure 3.19: Updating far plane depth. *(a) Frustum lies completely inside triangle $T_1$. The depth of the far plane is set to the maximum of $\mathbf{d_1}$ and $\mathbf{d_2}$. (b) Triangles $T_1$ and $T_1'$ constitute the blocker. We compute the far plane depths of each triangle and use the maximum of the depth values.*

Our algorithm implicitly assumes that the size of connected blockers is larger that the cross-section of the frusta. The simplest algorithm subdivides a frustum in a uniform manner. This approach is simpler to implement and also simpler to parallelize on current multi-core and many-core architectures, in terms of load balancing. However, many complex models (e.g. CAD models) have a non-uniform distribution of primitives in 3D. In that case, it may be more useful to perform adaptive subdivision of the frusta. In that case, we use the AD-Frustum representation [Chandak et al., 2008], which uses a quadtree data structure. We use the following criteria to perform subdivision. If the PVS associated with a frustum is large, we recursively compute the PVS associated with each sub-frustum. Whenever the algorithm only computes a partial blocker of connected triangles using the intersection tests, we estimate its cross-section area and use that area to compute the sub-frusta. There are other known techniques to estimate the distribution of primitives [Wonka et al., 2006] and they can be used to guide the

subdivision. As compared to uniform subdivision, adaptive techniques reduce the total number of frusta traced for PVS computation [Chandak et al., 2008]. Moreover, we use spatial coherence to reduce the number of intersection tests between the parent and child frusta.

### 3.4.6 Many-core Implementation

Our algorithm is highly parallelizable as the PVS for each frustum can be computed independently. However, the union of the PVSs has to be performed in a thread safe manner. This is done by maintaining an array of bits, one bit per triangle, and marking a bit visible when the corresponding triangle is found visible. The time to query if a triangle is visible is $O(1)$ but enumerating the visible triangles is $O(N)$, where $N$ is the number of triangles in the scene. To improve upon this we maintain a per thread hash map to compute PVS per thread. The PVS computed per thread is combined in the end to compute the final PVS. The average time to query if a triangle is visible is $O(1)$ and the time to enumerate the visible triangles is $O(K)$, where $K$ is the number of visible triangles.

### 3.4.7 Analysis and Comparison

We analyze our algorithm and compare it with prior techniques. The accuracy of our algorithm is governed by the accuracy of the intersection tests, which exploit the IEEE floating-point hardware. Our approach is robust and general, and not prone to any degeneracies.

**Conservative approach:** We compute a conservative PVS for every frustum. This follows from our basic approach to compute the blockers and far planes for each frustum. In practice, our approach can be overly conservative in some cases. The underlying blocker computation algorithm is conservative. Moreover, we don't consider the case when the union of two or more objects can serve as a blocker. This is shown in Figure 3.27

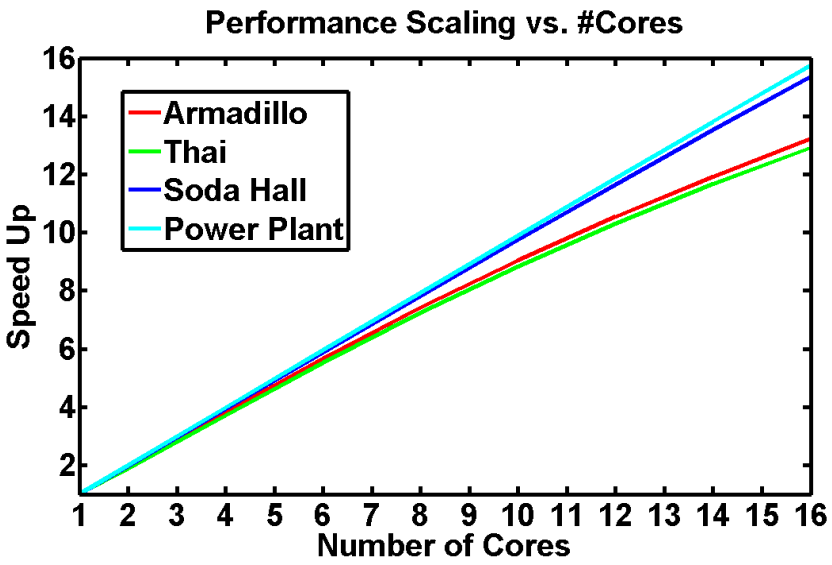



Figure 3.20: Performance scaling vs. #Cores. *The performance of our system scales linearly with the #cores. We have tested our system on up to 16 cores.*

with two disjoint occluders, $V_1$ and $V_2$. Instead of using more sophisticated algorithms for blocker computation, we found it cheaper to subdivide the frustum into sub-frusta and compute blockers for them. As a result, we can make our approach less conservative by using more frusta and the PVS ($\pi$) converges to $\pi_{exact}$ (see Figure 3.23).

**Model connectivity and triangle soup models:** Our algorithm exploits the connectivity information in the model to compute the blockers, which are formed using connected triangles. If the connectivity information is not available, then the algorithm would subdivide the frustum such that each blocker would consist of only one triangle.

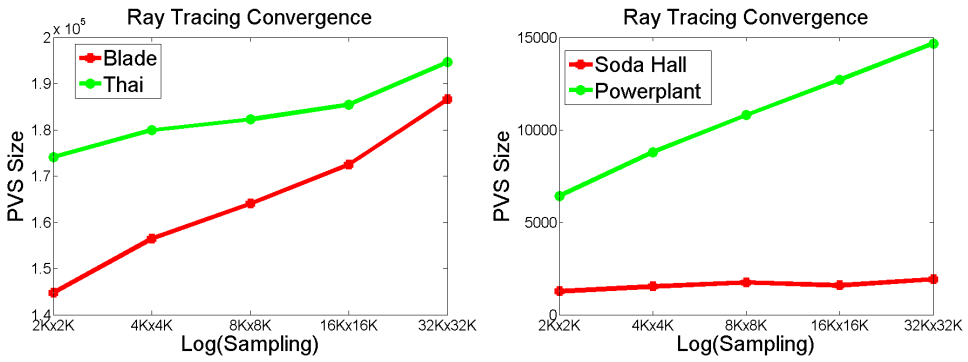
## Comparisons

Our approach performs volumetric tracing, which is similar to *beam tracing*. However, we don't perform exact clipping operations to compute an exact representation of the visible surface. Rather we only estimate the triangles belonging to the PVS by identifying the blockers for each frustum. Beam tracing algorithms can also be accelerated by using spatial data structures [Funkhouser et al., 1998, Laine et al., 2009], but they have mostly been applied to scenes with large occluders (e.g. architectural models). Recently,

Overbeck et al. [Overbeck et al., 2007] presented a fast beam tracing algorithm that is based on spatial hierarchies. We performed a preliminary comparison with an implementation of this beam tracing algorithm with FastV. We chose multiple key frames from the armadillo model sequence (see video) and compared the PVS computed by FastV algorithm (with $4K \times 4K$ uniform frusta) with the PVS computed by the beam tracer. We observed that FastV's PVS converges to within $1 - 10\%$ of the exact from-point beam tracing PVS (see Table 3.5). Furthermore, FastV appears to be more robust than the beam tracing solution as the beams may leak between the triangles due to numerical issues (see video). It is not clear whether current beam tracing algorithms can robustly handle complex models like the powerplant. In terms of performance, FastV is about $5 - 8$ times faster on a single core on the armadillo model, as compared to [Overbeck et al., 2007]. Moreover, FastV is relatively easier to parallelize on multi-core architectures.

Most prior object space conservative visibility algorithms are designed for scenes with large occluders [Bittner et al., 1998, Coorg and Teller, 1997, Hudson et al., 1997, Luebke and Georges, 1995]. These algorithms can work well on special types of models, e.g. architectural models represented using cells and portals or urban scenes. In contrast, our approach is mainly designed for general 3D models and doesn't make any assumption about large occluders. It is possible to perform conservative rasterization using current GPUs [Akenine-Möller and Aila, 2005]. However, it has the overhead of rendering additional triangles and CPU-GPU communication latency.

It is hard to make a direct comparison with image space approaches because of their accuracy. In practice, image space approaches can exploit the rasterization hardware or fast ray-tracing techniques and would be faster than FastV. Moreover, image space approaches also perform occluder fusion and in some cases may compute a smaller set of visible primitives than FastV. However, the main issue with the image space approaches is deriving any tight bounds on the accuracy of the result. This is highlighted




(a) (b)

Figure 3.21: PVS Size vs. Ray Sampling. *The PVS computed by ray tracing with increasing sampling on various benchmarks: (a) Blade and Thai (b) Soda Hall and Powerplant. The PVS computed by ray tracing does not seem to converge even after shooting a large number of rays.*

in Figure 3.21, where we used ray tracing to approximate the visible primitives. In complex models like the powerplant, we need a sampling resolution of at least $32K \times 32K$ to compute a good approximation of the visible primitives. At lower resolutions, the visible set computed by the ray tracing algorithm doesn't seem to converge well. Smarter sampling techniques [Wonka et al., 2006] can be used for better convergence.

### 3.4.8 Implementation and Results

In this section, we present our results on from-point conservative visibility. We also compare our approach with prior visibility computation algorithms. Our results were generated on a workstation with 16-core, 64-bit Intel X7350@2.93 GHz processors. We generate timings by using different number of cores $(1-16)$ for visibility computations. We also use SSE instructions to accelerate frustum intersection tests and OpenMP to parallelize the algorithm on multiple cores.

#### Performance Results

We demonstrate our results on computing from-point object space conservative PVS on a variety of models ranging from simple models (like soda hall, armadillo, blade) to

| Model | | PVS | PVS | Time |
|---|---|---|---|---|
| Name | Tris | Ratio | Size | (ms) |
| Armadillo | 345K | 1.16 | 98K | 30 |
| Blade | 1.8M | 1.05 | 190K | 90 |
| Thai | 10M | 1.06 | 210K | 66 |
| Soda Hall | 1.5M | 1.22 | 2.1K | 15 |
| Powerplant | 12M | 1.25 | 15K | 130 |
| Flamenco | 40K | 1.11 | 7K | 16 |

Table 3.4: FastV performance results. *From-point conservative visibility computation of models with varying complexity (CAD models, scanned models and dynamic scenes). All the timings were computed on a 16-core 64-bit Intel X7350@2.93 GHz. The PVS ratio provides a measures of how conservative is the computed PVS with respect to exact visibility.*

complex models (like powerplant and thai statue) and also dynamic scenes (flamenco animation). These models are shown in Figure 3.22. Our results are summarized in Table 3.4. We are not aware of any prior method that can compute the exact visible set on these complex models. Therefore, we compute an approximation to $\pi_{exact}$ by shooting frusta at $4K \times 4K$ resolution. The *PVS-ratio* refers to: (size of PVS) / (size of $\pi_{exact}$), and is a measure of how conservative is the computed PVS. In all benchmarks, we are able to compute a conservative approximation to the PVS at interactive rates on multi-core PC. The frame sequences used for generating average results are shown in accompanying video. Further, we show that our approach converges well to $\pi_{exact}$ as we shoot higher number of frusta (see Figure 3.23). Detailed results on convergence for each model are provided in Figure 3.26. Also, our approach maps well on multi-core architectures. We observe linear scaling in performance as the number of cores increase (see Figure 3.20).

### Comparison with Beam Tracing

We compare PVS computed by FastV using $4K \times 4K$ uniform frusta ($PVS_{4K \times 4K}$) with a PVS computed by an efficient beam tracer ($PVS_{BT}$) [Overbeck et al., 2007]. We choose 10 key frames from the armadillo sequence (see video) for comparison. Figure 3.24 and

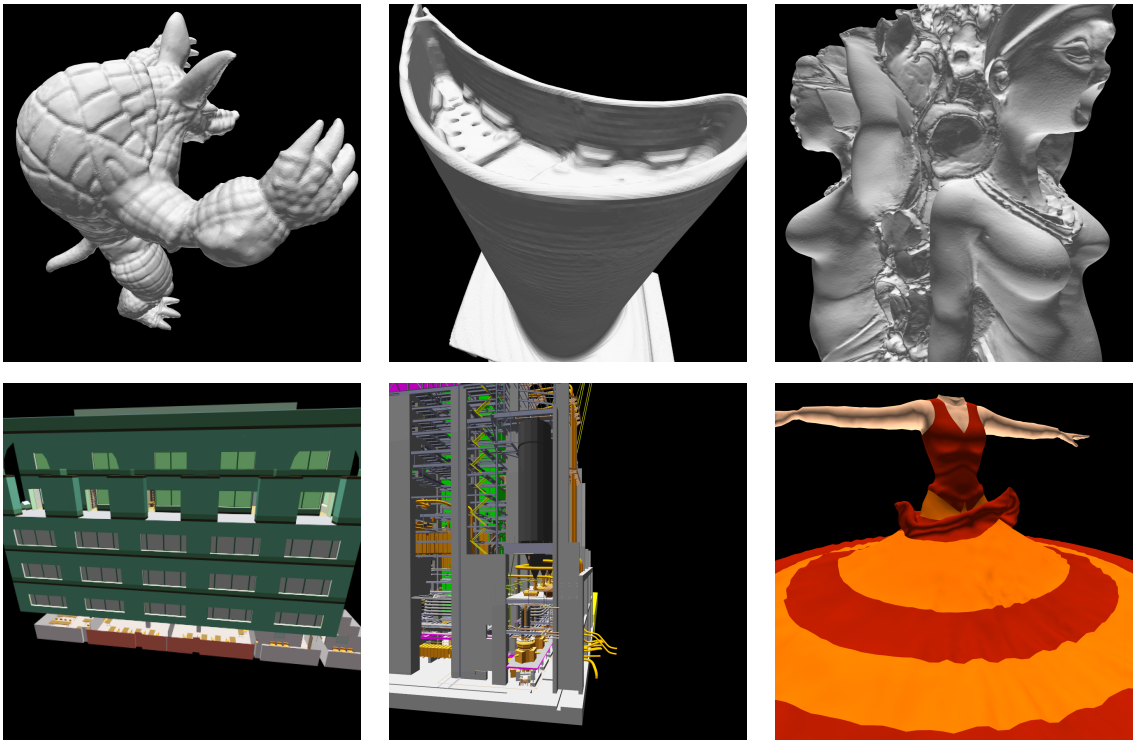


Figure 3.22: Benchmarks for FastV. *(a) Armadillo (345K triangles). (b) Blade (1.8M triangles). (c) Thai Statue (10M triangles). (d) Soda Hall (1.5M triangles). (e) Powerplant (12M triangles). (f) Flamenco (40K dynamic scene)*

Table 3.5 summarize our results. The PVS computed by FastV ($PVS_{4K\times4K}$) converges to within 10% of the exact PVS computed by the beam tracer ($PVS_{BT}$).

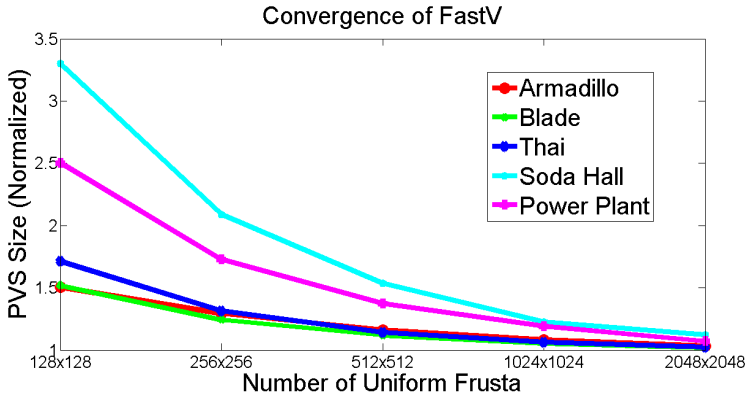



Figure 3.23: PVS ratio vs. #Frusta. *As the number of frusta increase, the PVS computed by our approach converges to $\pi_{exact}$. This graph shows the rate of convergence for different benchmarks. The CAD models have a higher ratio as compared to scanned models.*

| Frame # | PVS Size | | $\frac{PVS_{4K\times4K}-PVS_{BT}}{PVS_{BT}} \times 100$ |
|---|---|---|---|
| | $PVS_{4K\times4K}$ | $PVS_{BT}$ | |
| 1 | 97079 | 89192 | 8.84 |
| 2 | 72758 | 70948 | 2.55 |
| 3 | 34877 | 34434 | 1.28 |
| 4 | 78958 | 73660 | 7.19 |
| 5 | 73558 | 70612 | 4.17 |
| 6 | 89067 | 86596 | 2.85 |
| 7 | 71253 | 65361 | 9.01 |
| 8 | 110062 | 100248 | 9.78 |
| 9 | 121428 | 110210 | 10.17 |
| 10 | 97950 | 89846 | 9.01 |

Table 3.5: *Difference in the PVS computed by FastV ($PVS_{4K\times4K}$) and a beam tracer ($PVS_{BT}$) [Overbeck et al., 2007] as a percentage of $PVS_{BT}$.*

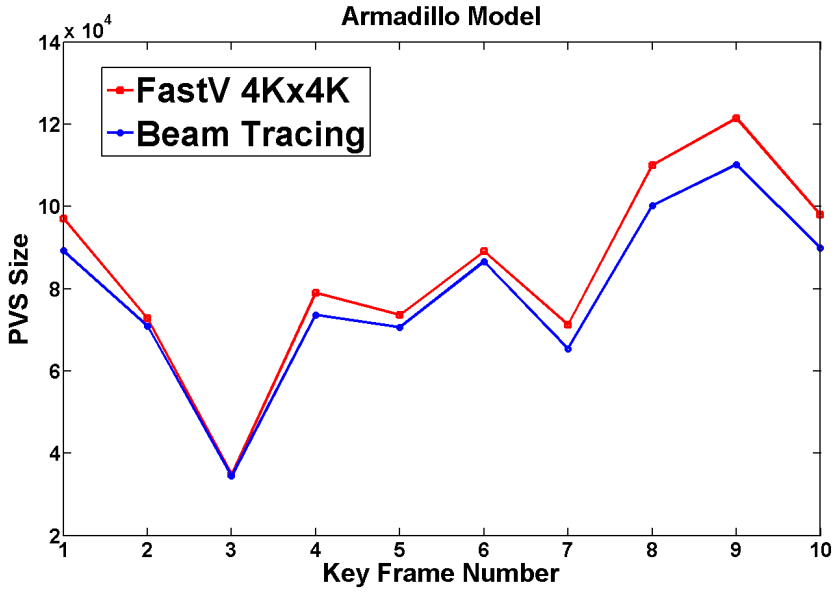



Figure 3.24: FastV vs. Beam Tracing. *Comparison of the PVS computed by FastV ($PVS_{4K\times4K}$) and a beam tracer ($PVS_{BT}$) [Overbeck et al., 2007] for the armadillo model sequence (see video).*

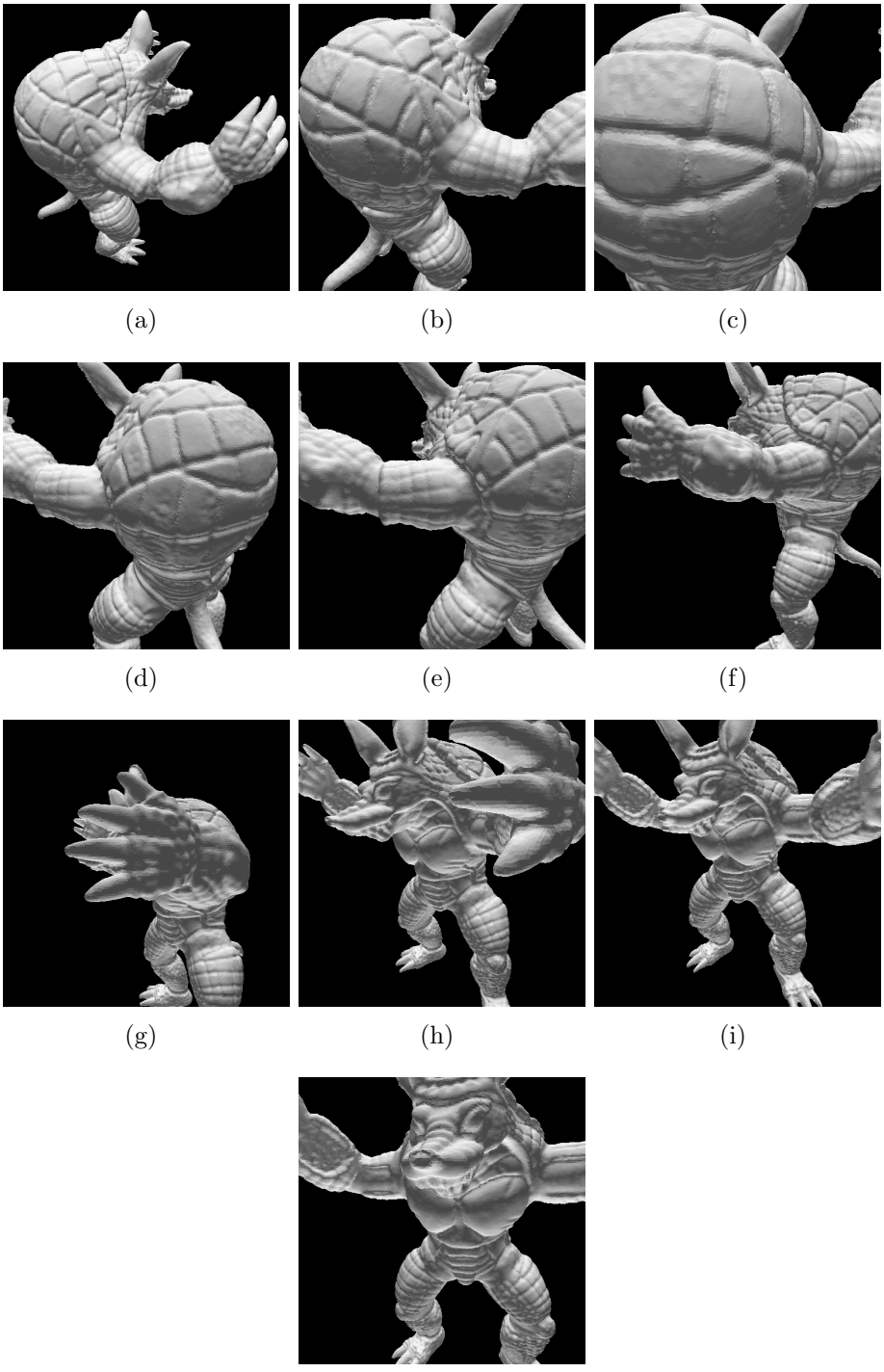


Figure 3.25: *Ten key frames used for comparing the from-point PVS computed by FastV and an efficient beam tracing implementation [Overbeck et al., 2007]. The PVS rendered in the images above was computed using the beam tracer [Overbeck et al., 2007].*

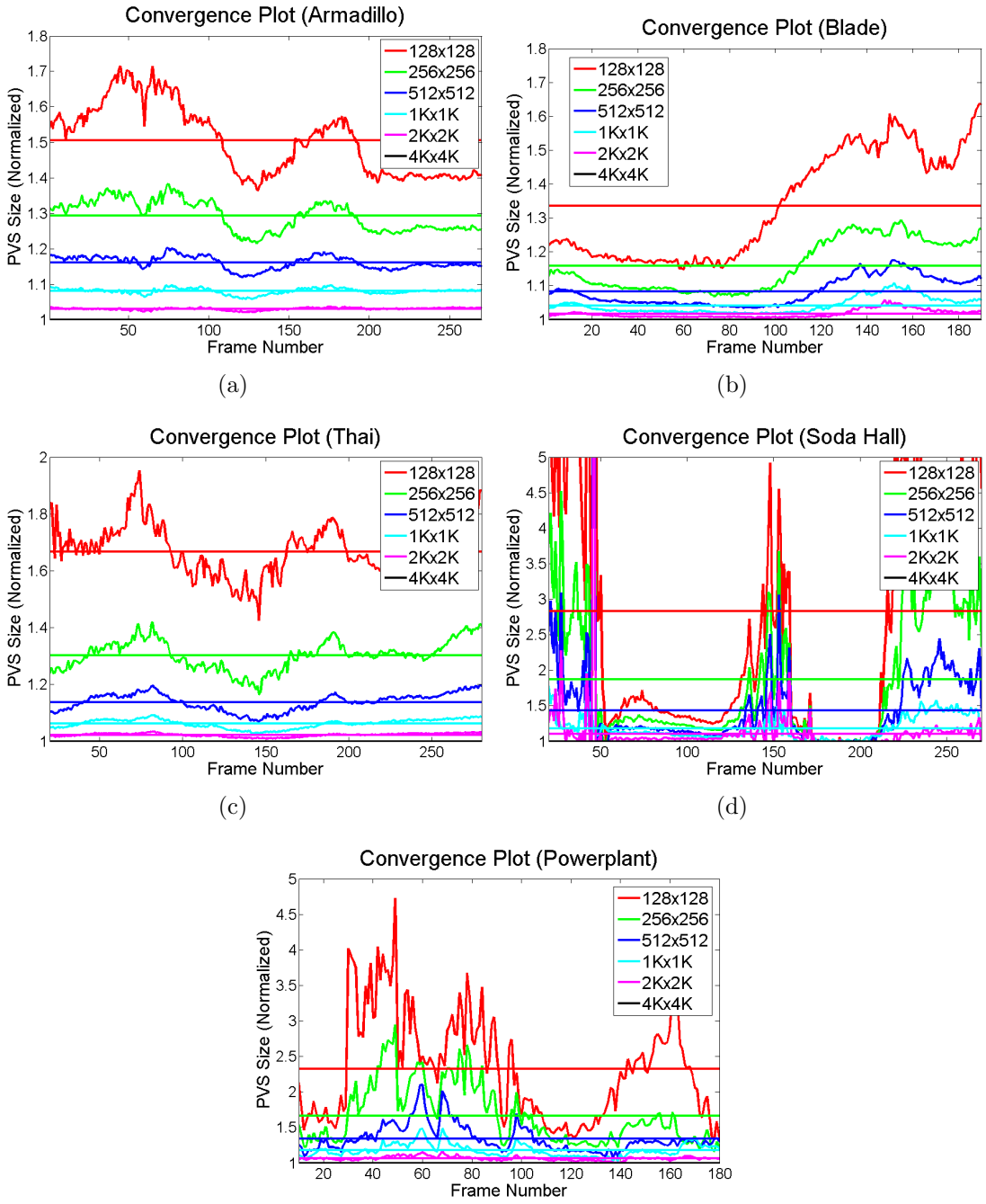



Figure 3.26: Convergence of FastV (see video). *(a) Armadillo, (b) Blade, (c) Thai, (d) Soda Hall, and (e) Powerplant. Solid horizontal lines denote average PVS over the entire video sequence for a given frusta resolution. For each benchmark, PVS computed by our approach converges to* $\pi_{exact}$ *as the frusta resolution increase.*

### 3.4.9 Conclusion, Limitation, and Future Work

We present a fast and simple visibility culling algorithm and demonstrate its performance on complex models. The algorithm is general and works well on complex 3D models. To the best of our knowledge, this is the first from-point object space visibility algorithm that can handle complex 3D models with millions of triangles at almost interactive rates.

**Limitations:** Our approach has some limitations. Since we don't perform occluder fusion, the PVS computed by our algorithm can be overly conservative sometimes. If the scene has no big occluders, we may need to trace a large number of frusta. Our intersection tests are fast, but the conservative nature of the blocker computation can result in a large PVS. The model and its hierarchy are stored in main memory, and therefore our approach is limited to in-core models. Our algorithm is easy to parallelize and works quite well, but is still slower than image space approaches that perform coherent ray tracing or use GPU rasterization capabilities.

Our approach can not perform occlusion fusion and this can make the PVS computation more conservative at times. However, such cases only arise if a frustum's boundary lies along the silhouette edges of one of the objects. Otherwise, the individual objects would eventually become the blockers for the sub-frusta generated by subdivision. This is shown in Figure 3.27(a), where the silhouette edges of both the objects $V1$ and $V2$ lie inside one of the frustum. Based on our blocker computation algorithm, $V3$ will be in the PVS of $F3$. Notice, even in this case the object $H2$ is not in the PVS of $F3$. Further, depending on the distribution of uniform frusta inside the viewing frustum the object $H1$ will not be visible as $V1$ becomes a blocker for $F4$ and $V2$ is a blocker for $F5$.

**Future Work:** There are many avenues for future work. We would like to implement our algorithm on many-core GPUs to further exploit the high parallel performance of commodity processors. This could provide capability to design more accurate rendering algorithms based on object-precision visibility computations on complex models (e.g.

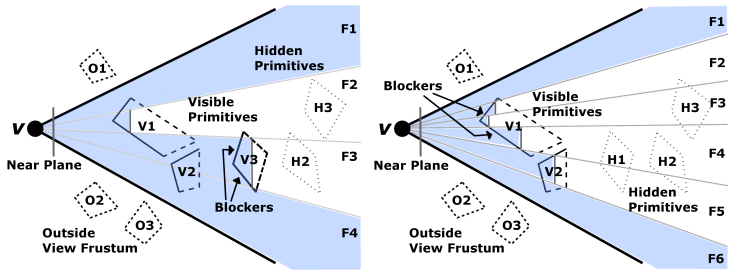



Figure 3.27: Occluder fusion. *(a) No occlusion fusion as the frustum lies along the silhouette edges of object $V1$ and $V2$. (b) Occlusion fusion happens implicitly as no frustum aligns along the silhouette edges of multiple objects.*

shadow generation). Our approach can be adapted to implement an efficient conservative rasterization engine [Akenine-Möller and Aila, 2005] on GPUs. With the release of DirectX ®11, blocker computation similar to our approach can be implemented efficiently on GPUs by randomly accessing scene primitives. Further, the conservative rasterization approach can be used to perform object space occlusion queries [Nguyen, 2007] and thereby design more accurate algorithms for shadow volumes [Lloyd et al., 2004] and collision detection[Govindaraju et al., 2003] computations in large environment. We would also like to evaluate the trade-offs of using more sophisticated blocker computation algorithms [Navazo et al., 2003, Laine, 2006].

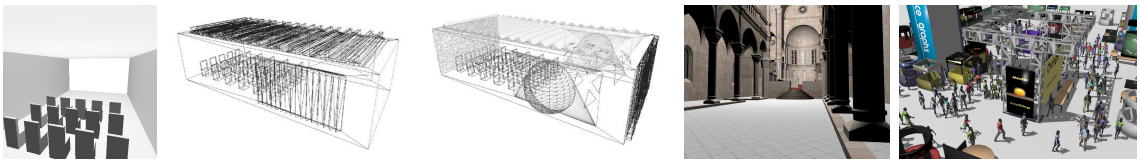
## 3.5 Sound Propagation using FastV

In this section, we describe our geometric sound propagation algorithm based on FastV. Given a point sound source, the CAD model with material properties (i.e. the acoustic space), and the receiver position, the goal is to compute the impulse response (IR) of the acoustic space. Later the IRs are convolved with an audio signal to reproduce the sound.

In order to compute accurate propagation paths we use our PVS computation algorithm described above for fast image-source computation that only takes into account

specular reflections [Allen and Berkley, 1979, Funkhouser et al., 1998, Laine et al., 2009]. In practice, this approach is only accurate for high frequency sources. Each image source radiates in the free space and considers secondary sources generated by mirroring the location of the input source over each boundary element in the environment. For each secondary source, the specular reflection path can be computed by performing repeated intersections of a line segment from the source position to the position of the receiver. In order to accurately compute all propagation paths, the algorithm creates image-sources (secondary sources) for every polygon in the scene. This step is repeated for all the secondary sources up to some user specified (say $k$) orders of reflection. Clearly, the number of image sources are $O(N^{k+1})$, where $N$ is the number of triangles in the scene. This can become expensive for complex models.

We use our PVS computation algorithm to accelerate the computation for complex scenes. We use a two stage algorithm. In the first stage, we use our conservative visibility culling algorithm and compute all the secondary image sources up to the specified orders of reflection. Since we overestimate the set of visibility triangles, we use the second stage to perform a validation step. For the first stage, we use a variant of Laine et al.'s [Laine et al., 2009] algorithm and only compute the secondary image-sources for the triangles that are visible from the source. Specifically, we shoot primary frusta from the sound source. For every primary frustum we compute its PVS. We then reflect the primary frustum against all visible triangles to create secondary frusta, which is similar to creating image-sources for visible triangles. This step is repeated for secondary frusta up to $k$ orders of reflection. In the second stage, we construct paths from the listener to the sound source for all the frusta which reach the listener. As our approach is conservative, we have to ensure that this path is a valid path. To validate the path, we intersect each segment of the path with the scene geometry and if an intersection is found the path is discarded.



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

Figure 3.28: Benchmarks for specular reflection modeling using FastV. *(a) Simple Room (876 triangles) (b) Regular room (1190 triangles) (c) Complex room (5635 triangles) (c) Sibenik (78.2K triangles) (e) Trade show (212K triangles).*

| Model | Tris | Time (sec) | Speed Up (ABT) |
|---|---|---|---|
| Simple Room | 438 | .16 | 10.1X |
| Regular Room | 1190 | .93 | 22.2X |
| Complex Room | 5635 | 6.5 | 11.8X |
| Sibenik | 78.2K | 72.0 | – |
| Trade Show | 212K | 217.6 | – |

Table 3.6: Acceleration for specular reflection due to using from-point visibility. *The performance of sound propagation algorithms for three orders of reflection on a single core. We observe* $10-20X$ *speedup on the simple models over accelerated beam tracing (ABT) [Laine et al., 2009]*

### 3.5.1 Results

We present our results on using FastV to accelerate specular reflection modeling in this section. Table 3.6 summarizes our results. We perform geometric sound propagation on models of varying complexity from 438 triangles to 212K triangles (see Figure 3.28). We use three benchmarks presented in accelerated beam tracing (ABT) algorithm [Laine et al., 2009]. We also used two additional complex benchmarks with 80K and 212K triangles. We are not aware of any implementation of accurate geometric propagation that can handle models of such complexity.

### 3.5.2 Analysis and Comparison

Most accurate geometric acoustic methods can be described as variants of the image-source method. Figure 3.29 compares different accurate geometric sound propagation

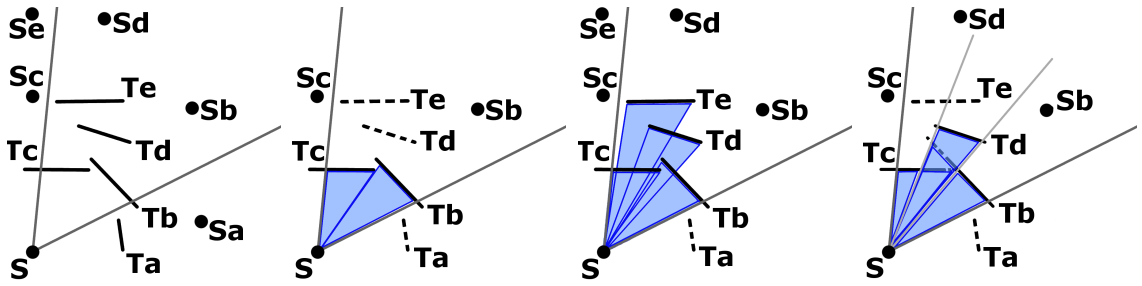



Figure 3.29: Geometric sound propagation approaches. *Given a sound source, S, and primitives (a, b, c, d, and e) the image source method (see 3.29a) creates image-sources of S against all primitives in the scene. The beam tracing method (see 3.29b) computes image-sources for only exactly visible triangles, b and c in this case. The accelerated beam tracing approach computes image-sources for all triangles inside the beam volume (see 3.29c), i.e., b, c, d, and e in this case. Our approach (see 3.29d) computes image-sources for triangles b, c, and d.*

methods. The main difference between these methods is in terms of which image-sources they choose to compute [Funkhouser et al., 1998, Laine et al., 2009]. A naïve image source method [Allen and Berkley, 1979] computes image sources against all triangles in the scene. Beam tracing methods compute the image-sources for exactly visible triangles from a source. Methods based on beam tracing, like accelerated beam tracing [Laine et al., 2009], computes image-sources for every triangle inside the beam volume. Our approach, shown in Figure 3.29(d), finds the conservative PVS from a source and computes image-sources for the triangles in the conservative PVS. Thus, for a given model our approach considers more image-sources compared to beam tracing. It is an efficient compromise between the expensive step to compute exactly visible triangles in beam tracing vs. computing extra image-sources in accelerated beam tracing. We observe $10-20$X speedups over prior accurate geometric sound propagation algorithms.

# Chapter 4

# Edge Diffraction Modeling

In the previous chapter we presented efficient techniques to model specular reflections. In this chapter, we present an efficient from-region visibility technique to model finite-edge diffraction. First, we describe our efficient from-region visibility algorithm in Section 4.1. Next, we summarize the image source method which can handle both specular reflections and finite-edge diffraction. In Section 4.2, we present our results on accelerating the image source method by applying from-region visibility.

## 4.1 From-Region Object-Space Conservative Visibility

In this section, we present our conservative from-region visibility algorithm. This work was performed in collaboration with a fellow graduate student, Lakulish Antani.

### 4.1.1 Overview

We present a novel algorithm to compute the occluders from a given region and combine it with prior methods to compute the potentially visible set (PVS) of primitives from a given region at object-space precision. Figure 4.1 shows an overview of our visibility algorithm. Formally, the from-region visibility problem can be described as follows.

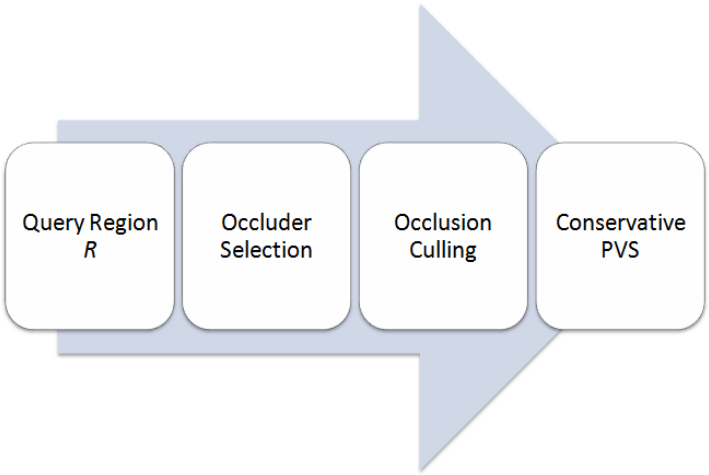



Figure 4.1: Overview of our conservative from-region visibility approach. *In the first step, we choose occluders for the query region R. Next, we use the occluders to compute which primitives are hidden from R by the occluders. The set of primitives not hidden by the occluders is the potentially visible set for R.*

Given a convex region $R \subset \mathbb{R}^3$ and a set of scene primitives $\Pi$, we wish to compute a subset of primitives $\pi \subseteq \Pi$ such that every primitive $p \in \Pi$ which is hit by a ray originating in $R$ is included in $\pi$. $\pi$ is called the *potentially visible set* (PVS) of $R$. The smallest such set is the *exact* PVS $\pi_{exact}$ of $R$. Our algorithm returns a *conservative* PVS, i.e. a superset of the exact PVS ($\pi \supseteq \pi_{exact}$).

Our visibility method can be divided into two steps: *occluder selection* for choosing primitives to be used as occluders for a given region $R$, and *occlusion culling* for computing the PVS of $R$ given the set of occluders. Note that our algorithm is general and can be used to compute the PVS of any convex region, including line segments (edges), triangles and volumetric cells such as bounding boxes. For our sound rendering applications, we only use the algorithm to compute the PVS of the diffracting edges.

### 4.1.2 Occluder Selection

The first step in computing the PVS of convex region $R$ is to compute the potential occluders for $R$. One option would be to simply use every primitive in the scene as an occluder, and use an occlusion culling algorithm that handles occluder fusion. In an ideal

scenario, such an approach would result in a PVS that is as close as possible to $\pi_{exact}$. However, the main issue with such an approach, which limits its practical application, is that the cost of occlusion culling is typically a function of the number of occluders [Chhugani et al., 2005]. Most prior work on occluder selection uses heuristics based on distance, solid angles, or area of primitives [Coorg and Teller, 1997, Hudson et al., 1997, Durand et al., 2000, Koltun and Cohen-Or, 2000]. Although these methods compute a subset of $\Pi$ to be used as occluders, they are unable to exploit the connectivity information of primitives to find any arbitrary set of connected triangles as occluders. Combining small occluders into large occluders can improve the culling efficiency of the visibility algorithm.

Thus, we propose a novel from-region occluder selection algorithm which exploits the connectivity between scene primitives whenever feasible. Our approach is general and applicable to all kinds of models including "polygon soup" models. We make no assumptions about the model or the connectivity of the polygons. In our implementation, the models are assumed to be triangulated; however, this is not a restriction imposed by our algorithm. If the model connectivity information is given or can be extracted, our algorithm can exploit that information to compute large occluders formed by connected sets of primitives.

Our technique can be viewed as a generalization of the conservative from-point visibility technique used in the FastV algorithm [Chandak et al., 2009]. FastV computes from-point visibility by constructing a cubical box around the query point $R$, then subdividing each of its faces into multiple quad patches $Q$ (where the number of quad patches can be user-specified), and then constructing frusta $F(R, q)$ from each quad patch $q \in Q$ and $R$ (see Figure 4.2). Each of these frusta is used to determine which portions of the scene are visible from the query point that use the relevant patch as the viewport. Formally, for each $q \in Q$ we wish to determine the set of primitives $p \in \Pi$ such that there exists a ray from $R$ to some point on $p$ which passes through $q$.

Given a frustum $f = F(R, q)$ (defined by its corner rays), the FastV algorithm tries to compute a *blocker* for $f$. In the context of FastV, a blocker is defined as a connected set of triangles such that any convex combination of the corner rays of $f$ intersects some triangle in the blocker. FastV traverses the scene hierarchy, and whenever a triangle $T$ is found that intersects $f$, it uses the connectivity information associated with $T$ to determine if some set of triangles connected to $T$ can also be used as a blocker for $f$. It is possible that there may be no such triangles. Therefore, once the traversal is completed, FastV returns at most one blocker for $f$ and zero or more connected sets of triangles in front of the blocker which do not completely block $f$.

Consider generalizing the frustum construction approach of FastV to the from-region case (i.e., now $R$ can be any convex region). We compute a fattened oriented bounding box (where the amount of "fattening" can be user-specified) that encloses $R$ and subdivide its faces into a user-specified number of quad patches $Q$. The next step is to determine the set of primitives $p$ such that there exists at least one ray from some point $r \in R$ to $p$ which passes through $q$. Put another way, we wish to determine all points from which $R$ is partially visible through $q$. This corresponds to the region in front of $q$ and bounded by the set $S$ of *separating planes* constructed between $R$ and $q$ [Coorg and Teller, 1997] (see Figure 4.3).

Note that we orient the separating planes such that $Q$ lies in the positive half-space (interior) defined by each separating plane $s \in S$. We then construct a *separating frustum* $f = F(R, q)$ bounded by $S$. The separating frustum need not be pyramidal; it is defined as the intersection of half-spaces bounded by the separating planes. We could use view frustum culling techniques to cull $\Pi$ to $f$ to estimate the PVS of $R$. However, this approach may compute a PVS $\pi$ such that there exist primitives $p_1, p_2 \in \pi$ where $p_1$ occludes $p_2$ from $R$, and the resulting PVS would be too conservative. Instead, we use FastV to trace $f$ (see Figure 4.3). (Note that if $R$ is in fact a single point, our occluder selection algorithm reduces to FastV.)

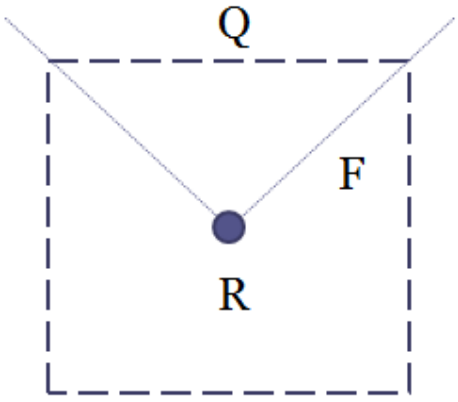



Figure 4.2: Frustum construction performed by FastV. *Given a query point R, we construct an axis-aligned box around it and divide the faces of the box into quad patches, one of which, Q, is shown in the figure. Given R and Q, we then trace a frustum F to compute the PVS for R.*

Ideally, we would like to trace all the rays that start on $R$ and pass through $q$, and the set of primitives reached would approach $\pi_{exact}$. However, tracing $f$ using FastV computes a *subset* of triangles visible from $R$ through $Q$ (i.e., computes $\pi \subseteq \pi_{exact}$). This subtle difference between the from-point and from-region case occurs because using FastV with a separating frustum for a region $R$ is not guaranteed to find all geometry reachable by rays starting on $R$ (for an example, see Figure 4.5) for a given frustum subdivision level. Therefore, after occluder selection, we use a conservative occlusion culling algorithm to compute a superset of the exact PVS.

Tracing $f = F(R, q)$ using FastV can return a blocker for $f$. This blocker is a connected set of triangles such that any ray originating on $R$ and passing through $q$ intersects the blocker. Therefore, we use all blockers returned by FastV as occluders. However, it is possible that FastV may be unable to find a blocker for $f$. In such a case, we use the connected sets of triangles computed by FastV during scene traversal as occluders (see Figure 4.4 for an example).

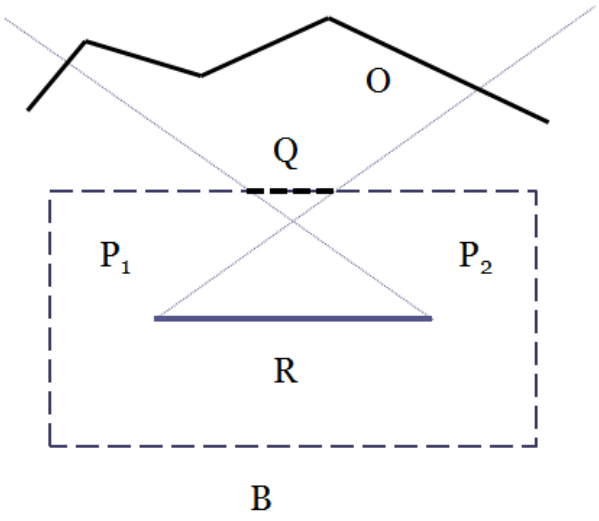



Figure 4.3: Separating frustum construction, in 2D. *Given a line segment $R$, we construct a fattened bounding box $B$, and divide its boundary into line segment patches, one of which is $Q$. We construct separating planes $P_1$ and $P_2$ between $R$ and $Q$, and trace the frustum bounded by these planes and oriented such that $Q$ is in the interior of the frustum. Here $O$ is a blocker for the separating frustum, and is used as an occluder for $R$.*

### 4.1.3 PVS Computation

Given a set of occluders for $R$, the next step is to perform occlusion culling to compute the PVS of $R$. Ideally, we would like to determine the umbra of an occluder $o$ with respect to $R$. Unfortunately, the boundary of an exact umbra is bounded by curved surfaces [Teller and Séquin, 1991]. A common workaround is to compute a *shadow frustum* bounded by these curved surfaces, and use it to determine a subset of triangles occluded by $o$ (thus computing a superset of the exact PVS for $R$). The shadow frustum is bounded by the *supporting planes* between $R$ and $o$ [Chhugani et al., 2005], and can be easily computed.

We can use any existing object-precision technique for occlusion culling, as long as it guarantees that the resulting PVS is conservative. Several methods that fit these requirements exist in the literature [Durand et al., 2000, Chhugani et al., 2005]. In our implementation, we have used a simple CPU-based frustum culling method. For each occluder $o$, we compute the shadow frustum $S(o, R)$ of $o$ from $R$ and mark all primitives

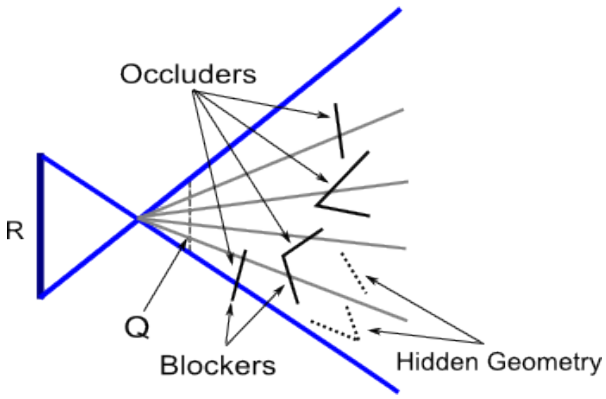



Figure 4.4: Occluder selection in 2D. *The bright blue lines are the corner rays of the separating frustum between query region R and quad patch Q. The grey lines indicate corner rays of sub-frusta formed by uniform frustum subdivision. Primitives chosen as occluders are shown as solid line segments, primitives hidden by the occluders are shown as dotted lien segments. Some of the occluders are frustum blockers, and these are also marked in the figure.*

behind $o$ and completely contained in $S(o, R)$ as occluded from $R$. Once all shadow frusta have been processed in this manner, the primitives not marked hidden are added to the PVS of $R$.

### 4.1.4 Cost Analysis

In this section we present a simple cost model for from-region visibility algorithms. Most from-region visibility algorithms can be decomposed into two steps: (a) Occluder Selection and (b) PVS Computation. These two steps are described in Section 4.1.2 and Section 4.1.3 for our approach. Thus, for a given region, $R$, the cost of from-region visibility from $R$ depends on the cost of the previous two steps.

$$T_{FR}(R) = T_{OS}(R) + T_{PVS}(R, O) \tag{4.1}$$

Where $T_{FR}(R)$, $T_{OS}(R)$, and $T_{PVS}(R, O)$ are the computation cost of from-region visibility, occluder selection, and PVS computation from region $R$. The PVS computation step depends on the occluder set $O$ computed by the occluder selection step. We choose a

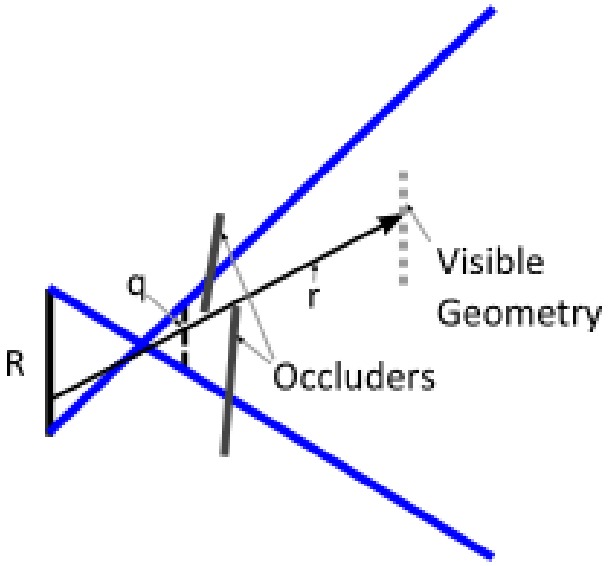



Figure 4.5: Accuracy issues with using from-point algorithms with separating frusta. *The figure shows an example scenario where FastV would be unable to compute a conservative PVS. If FastV is used to process the separating frustum formed by region R and quad patch q, the indicated occluders will be marked as visible. However, there could be geometry behind the occluders which can be reached by a ray originating on R, but which may be marked invisible by FastV depending on frustum subdivision level.*

very simplistic cost model dependent only upon the computation times of occluder selection and PVS computation steps as this is sufficient for comparison with other visibility approaches (see Section 4.1.5). We exclude discussion on simple heuristic-based algorithms for occluder selection. These algorithms choose a few occluders which are likely to occlude a large portion of the scene but such occluders are typically a small fraction of occluders that contribute to occlusion from a region for a scene [Wonka et al., 2000] and can lead to a highly conservative PVS.

### 4.1.5 Analysis and Comparison

The main application of our visibility algorithm is finite edge diffraction computation for sound rendering. We need to find all potentially visible edges from a given edge to compute higher order diffraction from the given edge. Missing and edge which is visible from the given edge leads to a discontinuous sound field around the missed edge. The

discountinous sound field could lead to artifacts in sound rendering, and this is unacceptable for our application. This fits very well into the kind of applications described by Wonka et al. [Wonka et al., 2000] where non-conservative (approximate) visibility algorithms, though beneficial, are not tolerable. Exact from-region visibility algorithms are prohibitively expensive for our application. Thus, we present a comparison of our approach with conservative from-region algorithms [Cohen-Or et al., 2003] in this section with emphasis on the occluder selection choices of these algorithms.

Efficient algorithms have been developed for computing conservative from region visibility for 2D and 2.5D scenes [Koltun and Cohen-Or, 2000, Wonka et al., 2000]. Schaufler et al. [Schaufler et al., 2000] can compute conservative from region visibility for 3D scenes but require the occluders in the scene to be volumetric. Our approach is a generic 3D visibility algorithm and does not impose restrictions on the scene geometry. Thus, the above approaches are not relevant for comparison with our approach.

Extended Projections [Durand et al., 2000] computes underestimated projections of occluders and overestimated projections of occludees from any viewpoint inside the region to perform occlusion culling. Thus, the PVS computation cost $T_{PVS}(R, O)$ depends linearly on the size of the occluder set, and the projection of the occluders is the bottleneck of Extended Projections. The occluder selection algorithm employed by Extended Projections computes occluders based on solid angle hueristics which could lead to a highly conservative PVS in addition to a non-optimal set of occluders for a given size of occluder set [Koltun and Cohen-Or, 2000].

vLOD [Chhugani et al., 2005] computes a shadow frustum for each occluder and finds an optimal projection point from which to compute a shrunk shadow frustum to determine conservative visibility. Computing an optimal projection point is formulated as a convex quadratic optimization problem which depends at least linearly on the size of the occluder set. Furthermore, the occlusion culling step in vLOD can benefit from the connected occluders computed by our occluder selection algorithm.

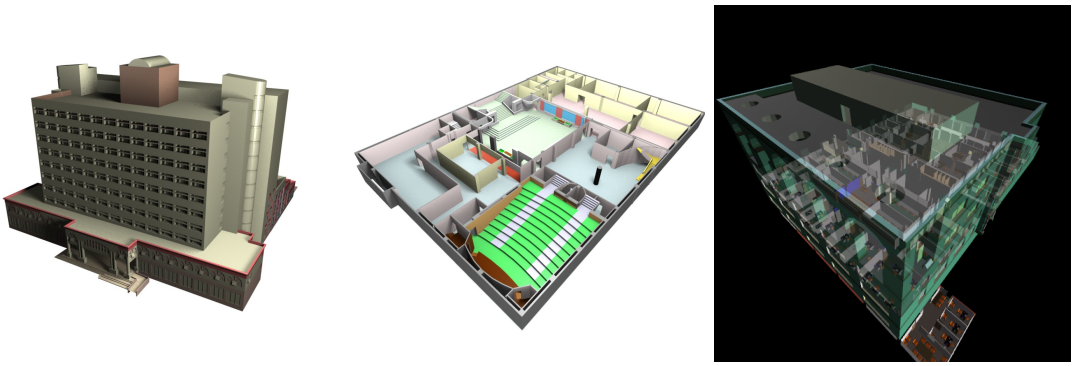


Figure 4.6: Benchmarks for from-region visibility. *Clockwise from top left: (a) Building (69K triangles) (b) Soda Hall (1.5M triangles) (c) Floor (7.3K triangles).*

### 4.1.6 Implementation and Results

In this section, we present experimental results on from-region visibility. Figure 4.6 shows the scenes we use to benchmark our code. The Building, Floor and Soda Hall examples are complex scenes used to benchmark our occluder selection algorithm. All of our tests were performed on an Intel Xeon X5560 workstation with 4GB RAM, running Windows Vista. Our implementation is written in C++ and uses SSE instructions to achieve high performance. All timings are reported for a single CPU core.

**Occluder Selection**: We now present our results for from-region visibility. We report the running times per triangle of our occluder selection step in Table 4.1. The table also reports the average number of triangles in each occluder. This demonstrates how our occluder selection algorithm is able to effectively combine connected triangles into larger occluders. These larger occluders can potentially allow more triangles to be culled. Moreover, the computational cost of state-of-the-art from-region occlusion culling algorithms tends to increase with an increase in the number of occluders. The time required for such computations can be reduced by using fewer, larger occluders formed by connected sets of triangles, such as those selected by our algorithm.

Table 4.2 compares the total running time for from-region visibility (occluder selection and occlusion culling) and the resulting PVS sizes when our occlusion culling implementation is provided with occluders computed as follows: no occluder selection (i.e.,

| Scene | Triangles | Occluder Selection | |
|---|---|---|---|
| | | Time (s) | Average triangles per occluder |
| Floor | 7.3K | 0.1 | 6.0 |
| Building | 69K | 1.3 | 3.0 |
| Soda Hall | 1.5M | 14.8 | 6.7 |

Table 4.1: Performance of our occluder selection algorithm. *All timings are reported for occluder selection for a single triangle (averaged over multiple triangles). The last column indicates the average size of occluders (in no. of triangles) returned by the occluder selection algorithm.*

| Scene | Triangles | No Occluder Selection | | Area Ratio Heuristic | | Separating Frusta | |
|---|---|---|---|---|---|---|---|
| | | Time | PVS Size | Time | PVS Size | Time | PVS Size |
| Factory | 170 | 15.2 | 64 | 14.9 | 64 | 11.5 | 69 |
| Room | 876 | 240 | 356 | 241.4 | 356 | 102 | 379 |
| House | 1150 | 192 | 209 | 112.2 | 261 | 90 | 350 |

Table 4.2: Benefit of our occluder selection algorithm. *Average computation time and PVS size when using different occluder selection algorithms. All times are in ms. "No Occluder Selection" refers to using all triangles as occluders. "Area Ratio Heuristic" refers to the use of Koltun et al.'s [Koltun and Cohen-Or, 2000] heuristic approach. "Tracing Separating Frusta" refers to our algorithm described in Section 4.1.2. Columns 3, 5 and 7 compare total running times for computing from-triangle visibility averaged over all triangles in the respective scenes. Columns 4, 6 and 8 compare the average PVS size per triangle.*

using all primitives as occluders), area-ratio heuristics [Koltun and Cohen-Or, 2000], and our occluder selection algorithm based on tracing separating frusta. The table clearly shows that using our occluder selection algorithm significantly reduces total time spent in visibility computation as compared to the other approaches, at the cost of a relatively small increase in PVS size. Note that when selecting occluders using the area-ratio heuristic, we evaluate the area-ratio for each primitive and choose all primitives whose scores are greater than or equal to the median score as occluders.

### 4.1.7 Conclusion, Limitations, and Future Work

Our from-region visibility algorithm uses a novel, systematic occluder selection method that is fast and can assemble connected triangles into a single larger occluder. This allows for efficient occlusion culling using state-of-the-art techniques. Our approach is easy to parallelize and scales well on multi-core architectures. The modularity of our technique allows us to use our occluder selection algorithm with any from-region occlusion culling algorithm and gain the benefits of combining adjacent triangles into single occluders.

**Limitations**: Our approach has several limitations. It is possible that in the absence of large primitives which can be used as occluders, our algorithm would have to trace a large number of small frusta in order to select occluders, which could adversely affect the performance of our algorithm. Further, current implementation of our from-region visibility algorithm uses a simple object-space frustum culling technique for occlusion culling. This can cause it to miss cases of occluder fusion due to disconnected occluders.

**Future Work**: There are many possible avenues for future work. We can use conservative rasterization methods [Chhugani et al., 2005, Akenine-Möller and Aila, 2005] for occlusion culling, which may be able to fuse occluders and thereby result in a smaller PVS from a given region. Moreover, the occluder selection step itself can be implemented on the GPU for additional performance gains.

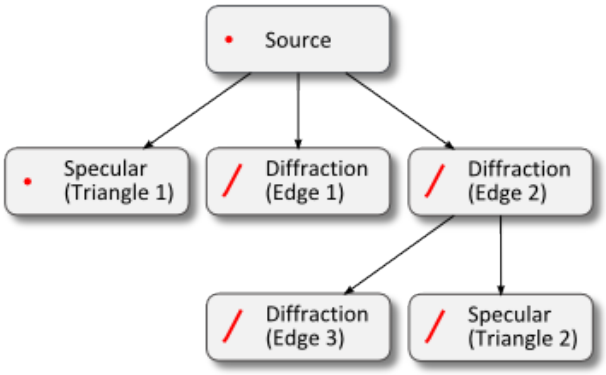



Figure 4.7: An example of a visibility tree. *Each node is labelled with a triangle (for specular reflections) or an edge (for edge diffractions). Each path in this tree corresponds to a sequence of triangles and/or edges that can be encountered by a ray propagating from source to listener. The icon next to each node's label indicates whether the image source stored in the node is a point or line source.*

## 4.2 Accelerated Image Source Method

In this section, we describe the image-source method, whose performance is accelerated by using conservative visibility algorithms [Chandak et al., 2010]. We use a two-step approach similar to all image source methods. First, for a given source position $S$, we construct a *visibility tree* $VT(S, k)$ up to a user-specified depth $k$ (see Figure 4.7). Each path in $VT(S, k)$ is a sequence of (up to $k$) triangles and/or edges that a ray starting from $S$ reflects and/or diffracts about as it reaches the listener at any position $L$. In other words, the paths in $VT(S, k)$ *partition* the set of propagation paths from $S$ to $L$, with each path $P_t$ in $VT(S, k)$ corresponding to an equivalence class $R(P_t)$ of propagation paths between $S$ and $L$. Next, given a listener position $L$, we traverse the visibility tree, and for each path $P_t$ in $VT(S, k)$, we determine which of the propagation paths in $R(P_t)$ are valid (i.e., unoccluded by other primitives) for the given source/listener pair. We refer to the second step of the process as *path validation*.

### 4.2.1 Visibility Tree

Here, we describe the visibility tree construction step. Each node in the tree corresponds to an image source $S_i$. Denote the node corresponding to $S_i$ by $N(S_i)$. We begin by creating a single node $N(S)$ corresponding to the source position $S$. The tree is then built recursively. To compute the children of $N(S_i)$, we compute the set of triangles $T(S_i)$ and edges $E(S_i)$ visible from $S_i$. For each $t \in T(S_i)$ we reflect $S_i$ about $t$, obtaining the reflection image source $S_i^t$, and construct the child node $N(S_i^t)$. For each $e \in E(S_i)$, we construct the child node $N(e)$. Note that computing $T(S_i)$ and $E(S_i)$ requires a from-point visibility query from $S_i$ if it is a point source, or a from-region visibility query if it is a line or edge source. We stop construction of the tree beyond a given maximum depth. This maximum depth can be user-specified in our implementation.

The visibility tree essentially describes the search space of propagation paths that need to be considered when computing the impulse response at $L$. To ensure that we consider all possible diffraction paths between $S$ and $L$, we need to ensure that we do not miss any of the visible edges when constructing the visibility tree. One way to ensure this is to assume each edge is visible from every other edge, or use highly conservative view-frustum culling and back-face culling approaches. However, this means that each node $N(S_i)$ corresponding to an edge source $S_i$ will have a very large number of children, many of which may not be reachable by a ray starting on $S_i$. This can dramatically increase the branching factor of the nodes in the tree, making higher-order paths almost impractical to compute. Therefore, we require visibility algorithms for both from-point and from-region queries that are not overly conservative. Figure 4.8 gives an overview of our technique and algorithms 1–3 show the pseudocode of this approach. The pseudocode clearly shows the crucial role of visibility in the image source method.

An important point to note is that the tree must be reconstructed if the source moves. However, if the scene is static, all necessary from-region visibility information can be precomputed, allowing the tree to be rebuilt quickly. In the next section, we

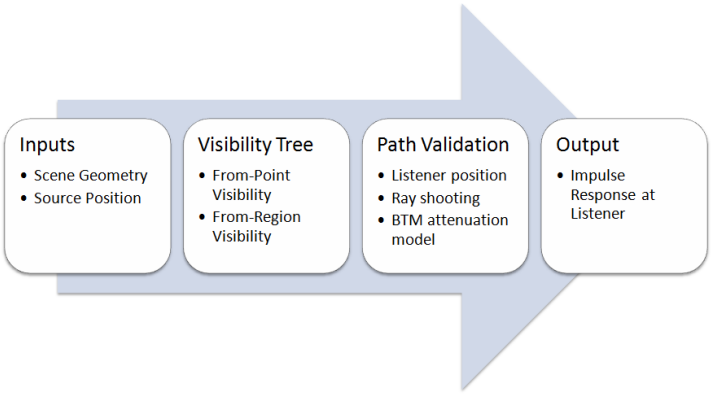



Figure 4.8: Overview of our image source method accelerated by applying from-point and from-region visibility. *Using the scene geometry and source position as the input, we first construct a visibility tree that represents the potential propagation paths. Next, we use the listener position to find valid propagation paths using the visibility tree. Finally, we use the valid paths and the BTM model to compute the impulse response at the listener.*

briefly describe the path validation process required to apply the BTM model for edge diffraction.

**Algorithm 1** $ComputeImpulseResponse(S, L)$

```
// S is the source position, L is the listener position
depth ← 0
VT ← φ
add node for S as root node in VT
VT ← AddNodes(VT, S, depth)
IR ← empty impulse response
node ← root node of VT
IR ← IR + ValidatePath(node, L, IR)
return IR
```

### 4.2.2 Specular Reflections

Given a point source $S$ and a listener $L$, it is easy to check if a direct path exists from $S$ to $L$. This is a ray shooting problem. The basic idea behind the image source method is as follows. For a specular reflector (in our case, a triangle) $T$, a specular path $S \rightarrow T \rightarrow L$ exists if the triangle $T$ is visible from the source $S$ and the listener $L$ is visible from the *image* of $S$ (created by reflecting $S$ across the plane of triangle $T$) through triangle

**Algorithm 2** *AddNodes*$(node, IS, depth)$

```
// node is the current tree node, IS is an image source, depth is the length of the current path
if depth ≥ maxDepth then
  return node
else
  T ← triangles visible from IS
  E ← edges visible from IS
  for all t ∈ T do
    IIS ← image of IS about t
    child ← node for IIS
    add child as a child node of node
    child ← AddNodes(child, IIS, depth + 1)
  end for
  for all e ∈ E do
    IIS ← image of IS along e
    child ← node for IIS
    add child as a child node of node
    child ← AddNodes(child, IIS, depth + 1)
  end for
  return node
end if
```

$T$. In the absence of any visibility information, image sources need to be computed for *every* triangle in the scene. This process can be applied recursively to check for higher order specular paths from $S$ to $L$, but the complexity increases exponentially as a function of the number of reflections. For a given source or image source position, we accelerate this process by applying the efficient from-point visibility technique described in Chapter 3 [Chandak et al., 2009]. Note that first-order image sources only need to be computed about triangles visible to $S$. For a first-order image source $S_1$, second-order image sources only need to be computed for the triangles that are visible to $S_1$ through $T$, and so on for higher order image sources.

### 4.2.3 Edge Diffraction

Analogous to how specular reflection about a triangle is modeled by computing the image of the source with respect to the triangle, diffraction about an edge is modeled

**Algorithm 3** $ValidatePath(node, L, IR)$

```
// node is a node in the visibility tree, L is the listener, IR is the impulse response
if L is visible to image source in node then
  IR ← IR+ contributions from path connecting all image sources in nodes from
  node to root
  for all child ∈ children(node) do
    if image source in child is visible to image source in node then
      IR ← IR + ValidatePath(child, L, IR)
    end if
  end for
end if
return IR
```

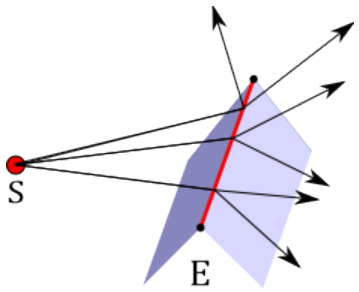



Figure 4.9: Image source of a diffracting edge. *Sound from source $S$ scatters in all directions upon encountering diffracting edge $E$. $E$ itself is therefore the image source of $S$ about $E$. The fact that rays scatter in all directions from $E$ implies that from-region visibility is required to compute all geometry reachable by these rays.*

by computing the image of the source *with respect to the edge*. The key idea is that the image source of a point source $S$ with respect to diffracting edge $E$ is that edge $E$ itself (see Figure 4.9). This is based on the Huygens interpretation of diffraction [Medwin et al., 1982]. (Intuitively, one can think of modeling the diffraction about the edge in terms of infinitesimally small emitters located along the edge.) This means that image sources can now be points or line segments. It follows from the Huygens interpretation that the image of a line source $E_1$ about a diffracting edge $E_2$ is $E_2$ (see Figure 4.10). Further note that the image of a point or line source $S_i$ about a planar specular reflector $T$ is obtained by reflecting $S_i$ across the plane of $T$ (see Figure 4.11).

In practice, most existing Biot-Tolstoy-Medwin (BTM) implementations either approximate visibility information, or use overly conservative culling techniques. For exam-

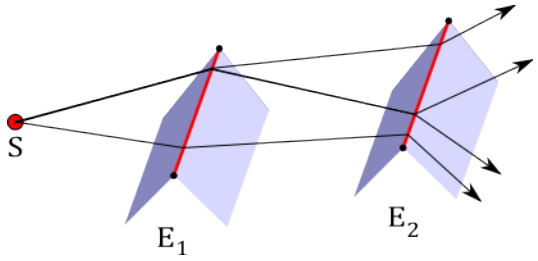



Figure 4.10: Image sources for two successive diffractions. *S is the source and $E_1$ and $E_2$ are diffracting edges. $E_1$ induces a first-order diffraction image source along its length. $E_2$ induces a second-order image source along its length.*

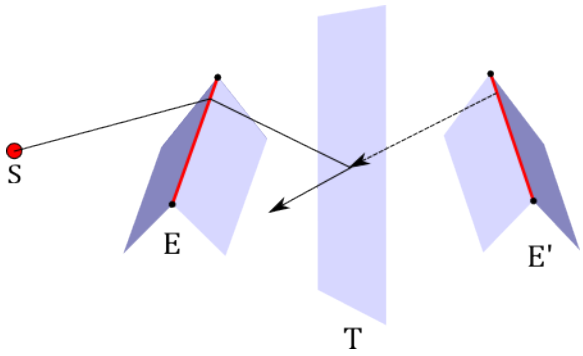



Figure 4.11: Image sources for one diffraction followed by one specular reflection. *S is the source and E is a diffracting edge. T is a specular reflector. E induces a diffraction image source along its length. This is reflected in the plane of T to give E′, which lies along the reflection of E in T.*

ple, the MATLAB Edge Diffraction toolbox is the state-of-the-art BTM implementation [Svensson, 1999]. For any edge $E$ formed by planes $P_1$ and $P_2$, the toolbox implementation culls away edges whose both endpoints are behind both $P_1$ and $P_2$. This is analogous to view frustum culling in graphics. In contrast, our approach uses a conservative from-region visibility algorithm to perform occlusion culling, so as to cull away additional geometry that is known to be invisible from $E$. Our from-region visibility technique is described in Section 4.1 [Antani et al., 2011c].

### 4.2.4 Path Validation

After constructing the visibility tree for a given source position, the next step is to use the tree to find propagation paths between the source and the listener, and to compute

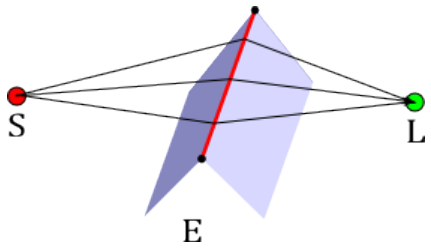



Figure 4.12: Diffraction paths between source $S$ and listener $L$ across edge $E$. *Note that here $n = 3$ ray shooting tests are needed to validate the diffraction paths.*

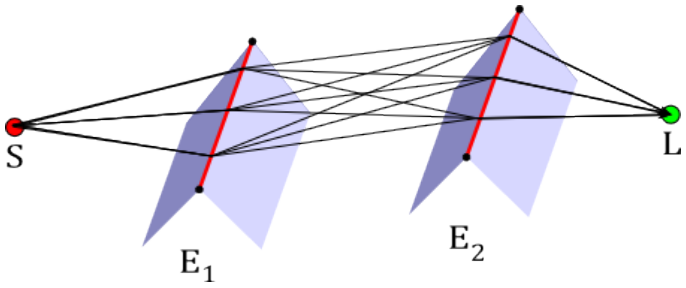
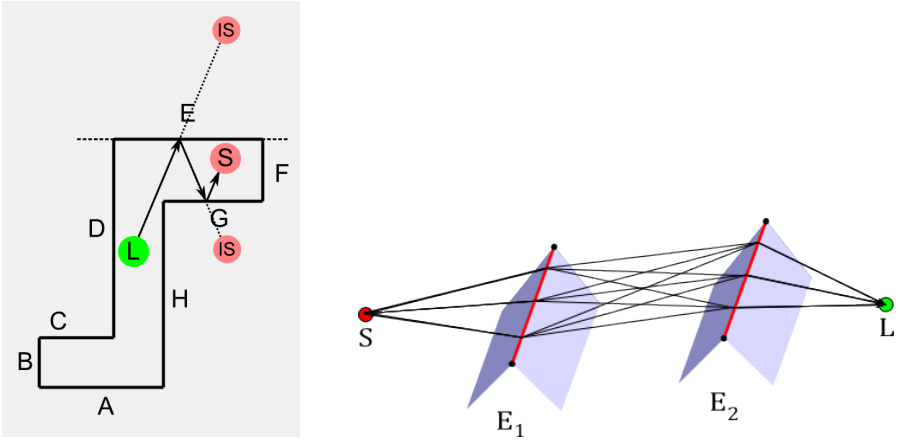



Figure 4.13: Second order diffraction paths between source $S$ and listener $L$ across edges $E_1$ and $E_2$. *Note that here $n = 3$ ray shooting tests are needed between $S$ and $E_1$ and between $E_2$ and $L$, whereas $n^2 = 9$ tests are required between $E_1$ and $E_2$.*

contributions from these paths to the final impulse response at the listener position. The validation step of specular reflection is simple and was described in Section 4.2.2 and Figure 4.14(a). In this section, we discuss the validation step for edge diffraction. We use the model described by Svensson et al. [Svensson et al., 1999], where the impulse response given a source at $S$ and listener $L$ and a single diffracting wedge is given by:

$$h(t) = -\frac{v}{4\pi} \int_{z_1}^{z_2} \delta\left(t - \frac{m(z) + l(z)}{c}\right) \frac{\beta(S, z, L)}{m(z)l(z)} dz \tag{4.2}$$

where $v$ is the *wedge index* for the diffracting edge [Svensson et al., 1999], $z_1$ and $z_2$ are the endpoints of the edge, $z$ is a point on the edge, $m(z)$ is the distance between $S$ and $z$, $l(z)$ is the distance between $z$ and $L$ and $\beta(S, z, L)$ is essentially the diffraction attenuation along a path from $S$ to $z$ to $L$. We evaluate this integral by discretizing the edge into some $n$ pieces and assuming that the integrand has a constant value over each piece (equal to its value at the midpoint of the piece). For each edge piece this gives an




(a) (b)

Figure 4.14: Path validation for specular reflection and diffraction. *(a) Path validation for specular reflection. Ray shooting tests are used to verify whether all segments of the path from source S to listener L are visible. (b) Path validation for second order finite-edge diffraction. Each edge is divided into small segments and ray shooting tests are performed from source to the edge segments, between edge segments, and edge segments to the listener to find visible paths from source S to listener L.*

attenuation of:

$$h_i = -\frac{v}{4\pi}\frac{V(S,z_i)V(z_i,L)\beta(S,z_i,L)}{m(z_i)l(z_i)}\Delta z_i \tag{4.3}$$

where $h_i$ is the IR contribution caused by edge sample $i$ with midpoint $z_i$. We augment the formulation of Svensson et al. [Svensson et al., 1999] by introducing $V(x,y)$, a Boolean valued visibility function which is true iff the ray from point $y$ to point $x$ is unoccluded by scene geometry. For second order diffraction, the corresponding attenuation is:

$$h_{ij} = \frac{v_1 v_2}{16\pi^2}V(S,z_i)V(z_i,z_j)V(z_j,L) \times \frac{\beta(S,z_i,z_j)\beta(z_i,z_j,L)}{m_1(z_i)m_2(z_i,z_j)l(z_j)}\Delta z_i \Delta z_j \tag{4.4}$$

where $h_{ij}$ is the IR contribution from sample $i$ on the first edge and sample $j$ on the second edge, with midpoints $z_i$ and $z_j$ respectively. Here, $v_1$ is the wedge index for the first edge and $v_2$ is the wedge index for the second edge.

Given a path in the visibility tree which may contain any number of specular and/or

diffraction nodes, we wish to use the Equations 4.3 and 4.4 to compute contributions to the final IR. For a given listener position $L$, we perform this step as follows:

1. We traverse each path in the tree in a bottom-up manner.

2. For each leaf node $N(S_l)$, we compute all valid propagation paths between $S_l$ and $L$. For each internal node $N(S_i)$ and its parent $N(S_j)$, we compute all valid propagation path segments between $S_i$ and $S_j$.

3. For each valid path segment with endpoints $p_i$ and $p_j$, we compute the corresponding delay $\|p_j - p_i\| / c$, distance attenuation $1/\|p_j - p_i\|$, specular attenuation $\sigma = \sqrt{1-\alpha}$ where $\alpha$ is the frequency-dependent absorption coefficient of the reflecting triangle (if $S_j$ is a specular node) and diffraction attenuation $\beta(p_i, p_j, p_k)$ where $p_k$ is the path endpoint corresponding to the parent of $S_j$ (if $S_j$ is a diffraction node).

In practice, these delays and attenuations are computed only if the corresponding visibility terms are nonzero. This check is performed using ray shooting between $S_i$ and $S_j$. Ideally, we would like to compute the set of all unoccluded rays between $S_i$ and $S_j$. (If $S_j$ is formed by a specular reflector $T$, then we only consider the rays between $S_i$ and $S_j$ which intersect $T$ and are unoccluded between $S_i$ and their hit point on $T$.) If $S_i$ and $S_j$ are both point sources, this reduces to a simple ray shooting test. However, if either one is a line source, path validation reduces to from-region visibility computation.

In order to accurately compute contributions from each propagation path, we would ideally like to compute the exact visibility information. However, note that the BTM model computes the effect of diffraction about an edge in terms of a line integral over the edge. This integral must be discretized in order to compute impulse responses. We approximate the line integral using the midpoint method – by dividing the edge into $n$ segments, and computing contributions due to paths passing through the midpoints of each segment. This method of integration allows us to use $n$ ray shooting tests (one for

the midpoint of each of the $n$ edge segments) to compute (approximate) visibility. Even though we use ray shooting and compute approximate visibility while computing IRs using the BTM model, it is necessary for us to compute conservative from-edge visibility when constructing the visibility tree. This is to ensure that no potential diffraction paths are missed. If an approximate visibility algorithm is used to construct the visibility tree, it may mark some edges that are visible from a given edge as invisible; this would lead to some diffraction paths being missed when computing IRs.

Observe that a propagation path is essentially a polyline which starts at the source, ends at the listener and whose intermediate vertices lie on triangles and/or edges in the scene. In the case of specular reflections only, path validation is performed using ray shooting to validate each segment of a polyline through the scene. If we also include one edge diffraction in the propagation path, we now need to validate $n$ polylines through the scene, using $n$ ray shooting tests for each image source along a path in the visibility tree. If we include a second edge, we need to validate $n^2$ polylines, and so on. However, in this case, we do not need to perform $n^2$ ray shooting tests for every image source along the path: only for image sources between the two diffracting edges (see Figures 4.12 and 4.13 for details). This is because there are $n$ polylines from the source to the first edge, and $n$ polylines from the second edge to the listener. Therefore the $n^2$ polylines from the source to the listener share several common segments, which allows us to reduce the number of ray shooting tests required. By a similar argument, it can be shown that for third- and higher-order diffraction paths, the number of ray shooting tests required between any two image sources is at most $O(n^2)$, even though the total number of polylines is $O(n^d)$ (where $d$ is the number of diffracting edges in the path).

Once the validation step is complete and all contributions to the IR at the listener position have been computed, the next step is to render the final audio. We simply convolve the input sound signal with the computed impulse response to generate the output audio for a given listener position. To generate smooth audio for a moving

listener, we interpolate between impulse responses at successive listener positions.

### 4.2.5 Cost Analysis

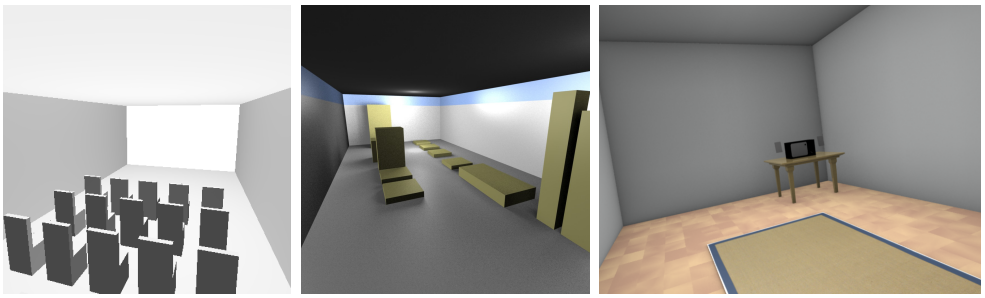
As previously discussed, our diffraction computation proceeds in two steps: first we compute mutually visible edge pairs, and next we compute IR contributions from each mutually visible edge pair. The cost of the first step, which we shall denote by $C_{vis}$, depends on the number of diffracting edges and the scene complexity. Suppose the number of mutually visible edge pairs returned by the first step is $N_{vis}$. Then the cost per edge pair for the second step, which we shall denote by $C_{IR}$, depends on the edge sampling density (see Section 4.2.4 for details). Therefore, the total cost is:

$$C_{total} = C_{vis} + N_{vis}C_{IR} \tag{4.5}$$

$C_{IR}$ can be quite high, and therefore using an efficient, accurate from-region visibility algorithm allows us to reduce $N_{vis}$ with a small overhead ($C_{vis}$) and thus reduce the total cost. In this work, we concentrate on the use of visibility algorithms to reduce $N_{vis}$, and use a simple IR computation approach. Several techniques have been developed in the literature for efficient computation of IRs (which lower only $C_{IR}$) [Calamia et al., 2009]; our approach can be modified to use these techniques to achieve improved overall performance.

### 4.2.6 Implementation and Results

In this section, we present experimental results on sound propagation using from-region visibility and from-point visibility. We compare our sound propagation system with the state-of-the-art to highlight the benefits of using conservative visibility when constructing image sources. Figure 4.15 shows the scenes we use to benchmark our code. Figure 4.16 shows some examples of diffraction paths computed by our algorithm. All of



(a) (b) (c)

Figure 4.15: Benchmarks for accelerated images source method. *(a) Room (876 triangles) (b) Factory (170 triangles) (c) House (1K triangles).*

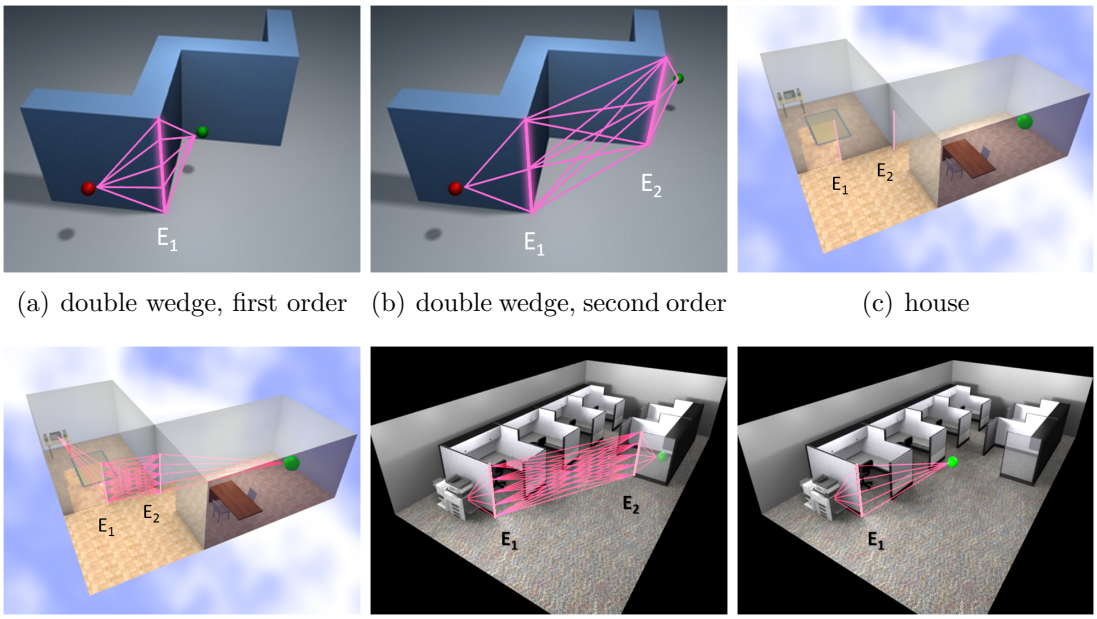
| Scene | Visibility Computation (ms) | Visibility Tree (ms) | IR Computation (s) |
|---|---|---|---|
| Factory | 11.5 | 141.0 | 23.9 |
| Room | 102.2 | 747.6 | 10.4 |
| House | 89.6 | 1045.6 | 24.3 |

Table 4.3: Performance of individual steps of our algorithm. *Column 2 shows the average time taken for visibility computation (PVS computation) per diffracting edge. Column 3 shows the time spent in constructing the visibility tree, averaged over multiple source positions. Column 4 shows the time taken to compute the final IR, averaged over multiple source and listener positions.*

our tests were performed on an Intel Xeon X5560 workstation with 4GB RAM, running Windows Vista. Our implementation is written in C++ and uses SSE instructions to achieve high performance. All timings are reported for a single CPU core.

Table 4.3 shows the breakdown of time spent in each step of our image source method. It is evident from the table that the costliest step of our algorithm is the final IR computation. Computing from-region visibility and constructing the visibility tree is much faster by comparison. Note that all timings in this section are reported for up to two orders of specular reflection and upto 2 orders of edge diffraction.

We first demonstrate the advantage of using from-region visibility in our BTM-based sound propagation system. We compare the performance of our visibility tree construction step (using from-region visibility) against visibility tree construction using only the simple culling approach used in the MATLAB Edge Diffraction toolbox [Svensson, 1999]




(a) double wedge, first order (b) double wedge, second order (c) house

(d) house, second order (e) office, second order (f) office, first order

Figure 4.16: Some examples of diffraction paths computed by our algorithm. *Parts (a) and (b) show first and second order diffraction paths, respectively, around a wall shaped like a double wedge. Parts (c) and (d) show the House scene and a second order diffraction path in it, respectively. Parts (e) and (f) show second and first order diffraction paths, respectively, in an Office scene. In each case, diffracting edges are highlighted and labeled; for second order paths $E_1$ is the first edge encountered along the path from source to listener, and $E_2$ is the second edge encountered. In each case, the listener is indicated by a green sphere, and the source is indicated by a red sphere (Parts (a) and (b)), the speakers (Parts (c) and (d)) or the printer (Parts (e) and (f)). Note that in the interests of clarity, these figures show far fewer diffraction paths than are actually simulated by our algorithm.*

(as implemented in C++). We compare the time required to build the visibility tree as well as the size of the tree constructed for each approach. Table 4.4 also shows the speedup obtained in the validation and IR computation step as a result of using conservative from-region visibility when constructing the visibility tree. As the table demonstrates, even for a very unoptimized implementation running on a single core, using conservative visibility algorithms can offer a significant performance advantage (2X-14X) over state-of-the-art BTM-based edge diffraction modeling methods.

When numerically evaluating the BTM integral, we divide each edge into 44K samples [Calamia et al., 2009]. Evaluating the visibility functions (Equations 4.3 and 4.4)

| Scene | Triangles | Edges | $2^{nd}$-order diffraction paths | | | Validation | |
|---|---|---|---|---|---|---|---|
| | | | Visible | Toolbox | Reduction | Edge intervals | Speedup |
| Factory | 170 | 146 | 4424 | 12570 | 2.84 | $10 \times 10$ | 1.93 |
| Room | 876 | 652 | 43488 | 181314 | 4.17 | $5 \times 5$ | 3.23 |
| House | 1105 | 751 | 133751 | 393907 | 2.95 | $5 \times 5$ | 13.74 |

Table 4.4: Advantage of using conservative from-region visibility for second order edge diffraction. *Columns 4–6 demonstrate the benefit of using from-region visibility to cull away second order diffraction paths between mutually invisible edges. The last column shows the speedup caused by this reduction in the size of the visibility tree. Column 7 refers to the number of rays shot per edge and the number of integration intervals corresponding to each ray. For example,* $5 \times 5$ *refers to 5 rays shot per edge and a total of 25 integration intervals, with each ray shooting test used to compute the visibility term for 5 consecutive integration intervals.*

for each of these 44K samples using ray shooting would be very time-consuming. Therefore, we subsample the edges and perform ray shooting visibility tests only between these subsampled points. The edge sampling densities reported in Table 4.4 refer to these subsampled points.

The table clearly highlights the importance of from-region occlusion culling in the BTM model. In the absence of occlusion culling, the size of the visibility tree grows very rapidly with depth. Our approach uses occlusion culling to reduce the branching factor of the nodes of the visibility tree. Reducing the size of the tree in turn implies faster validation of diffraction paths using the BTM model. Note that the speedup numbers were measured for the particular edge sampling densities specified in the table. However, the speedup numbers compare the values of $N_{vis}C_{IR}$ (see Section 4.2.5), where $N_{vis}$ varies with the visibility algorithm chosen. Since $C_{IR}$ (which depends on the edge sampling density) does not vary with choice of visibility algorithm, the speedup numbers remain roughly constant for different values of edge sampling density.

Figure 4.17 shows the average percentage of total triangles (and diffracting edges) visible from the diffracting edges in various benchmark scenes. These plots clearly show that even in simple scenes used for interactive sound propagation, from-region visibility helps reduce the complexity of the visibility tree computed by our algorithm by a factor

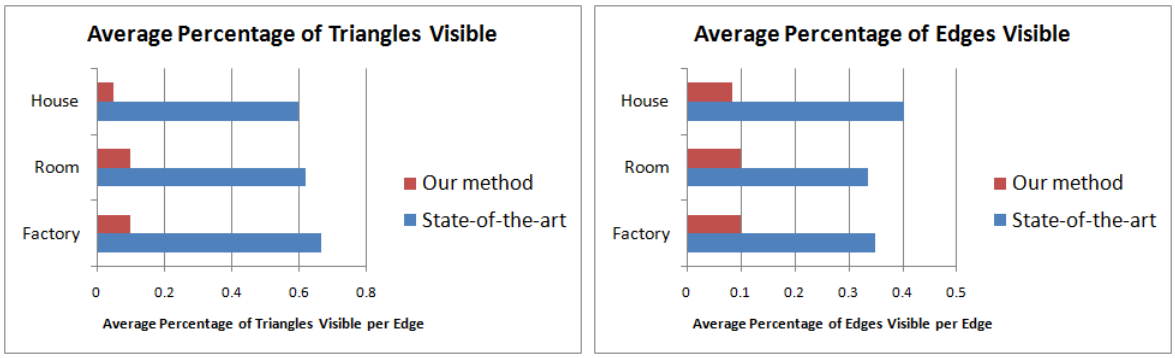


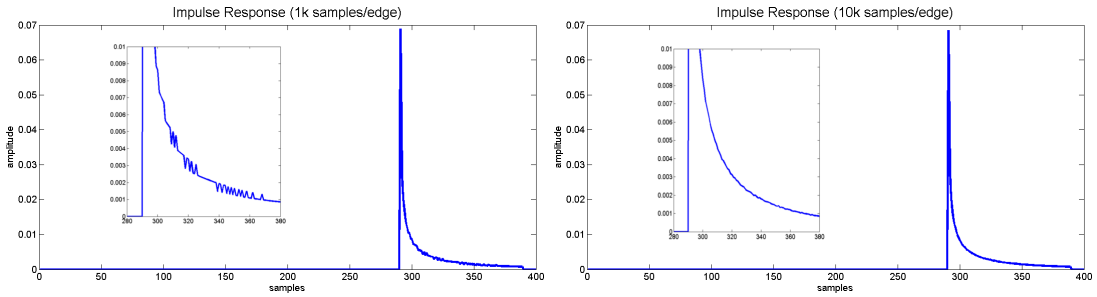

Figure 4.17: Visible geometry comparison. *Average amount of visible geometry returned by our approach as compared to the state-of-the-art for various benchmarks. The horizontal axis measures the fraction of visible geometry (triangles or edges, respectively) averaged over all edges in the scene. Smaller is better.*

of 2 to 4.

### 4.2.7 Accuracy Analysis and Comparison

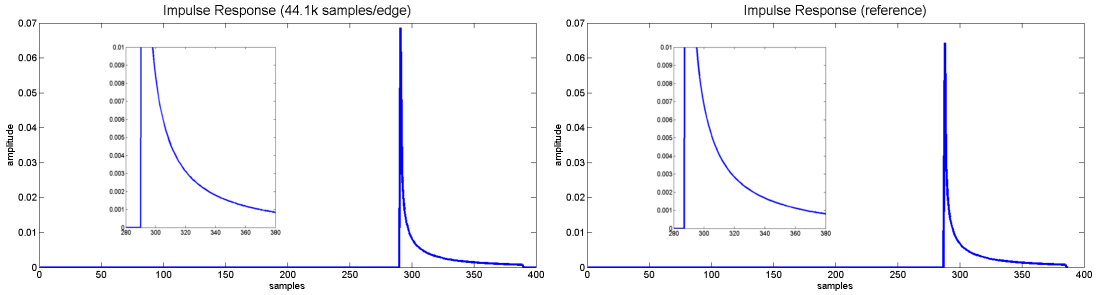
We have implemented the line integral formulation of the BTM model for performing path validation and computing impulse responses. The crucial parameter in the validation step is the number of samples each edge is divided into. A higher number of samples per edge results in more accurate evaluation of the BTM integral at a higher computational cost. Figure 4.18 shows impulse responses computed for diffraction about a simple double wedge for increasing numbers of samples per edge. As can be seen from the figure, increasing the number of samples causes the IRs to converge to the reference IR computed by the MATLAB toolbox [Svensson, 1999] (also shown in Figure 4.18). Although we have used a simplistic approach for calculating IRs in our implementation, the variation of IR accuracy with edge sampling strategy and sampling density has been studied in the literature [Calamia and Svensson, 2005, Calamia and Svensson, 2007].

Further note that although the computational cost of the BTM model remains higher than that of the UTD model, it has been shown [Svensson et al., 1999] that the BTM model is more accurate than UTD model at low frequencies, where diffraction plays an important role. At low frequencies, numerical methods can be used to capture diffraction




(a) IR with 1K samples/edge

(b) IR with 10K samples/edge




(c) IR with 44K samples/edge

(d) Reference IR computed by MATLAB toolbox

Figure 4.18: Accuracy of impulse responses computed by our system for first order diffraction about a single finite wedge. Parts (a)–(c) show the variation in the IR with increasing number of samples per edge. As the sampling increases, the IR approaches the reference IR computed by the MATLAB toolbox, shown in part (d).

effects, but the complexity scales with the volume of the scene, as opposed to BTM-based methods whose complexity scales with the number of diffracting edges. Moreover, combining a numerical acoustics algorithm with geometric acoustics techniques for high frequency simulations remains a challenging problem, whereas the BTM approach can easily be combined with the image source method to compute accurate diffraction effects.

### 4.2.8 Conclusion, Limitations, and Future Work

In this section we present a discussion on different visibility algorithms for computing the *visibility tree* for early specular reflections and finite-edge diffraction. The choice of visibility algorithm depends on the target application. For instance, room acoustics software requires accurate modeling of early specular reflections and edge diffraction, therefore,

exact or conservative object-space visibility algorithms are most suitable. Similarly, for entertainment applications like games it might be possible to use sample-based visibility algorithms as temporal and spatial aliasing issues can be hidden by applying heuristics which reduce the accuracy of the simulation.

Another example is that the cost of computing the diffraction paths and IRs for double or triple diffraction for finite-edge diffraction using the BTM model could be so high that it might be worth looking into exact visibility approaches to compute the smallest PVS from an edge and thus minimize the path validation steps. Exact visibility algorithms are relatively expensive and it is hard to implement them robustly in 3D. However, the savings in the size of the visible set may result in improved overall performance.

**Sample-based Approaches**: Due to their simplicity and efficiency, sampling based approaches are very popular in geometric acoustics [Krokstad et al., 1968]. However, the acoustic space has to be sampled densely to produce a robust solution. Since sampling-based approaches discretely sample the acoustic space, they introduce statistical errors [Lehnert, 1993] and may miss critical early reflection paths [Begault, 1994]. Many techniques like ray tracing [Vorländer, 1989, Krokstad et al., 1968], ray-frustum tracing [Lauterbach et al., 2007b, Chandak et al., 2008], and other sample-based techniques [Funkhouser et al., 2003, Taylor et al., 2010] have been applied to compute early specular reflections.

We are not aware of any work on using sample-based from-region visibility algorithms to accelerate finite-edge diffraction. Some recent techniques for sample-based from-region algorithms [Wonka et al., 2006, Bittner et al., 2009] can be applied on simple scenes but the impact of sampling error needs to be carefully analyzed.

**Object-Space Exact Approaches**: The size of the visibility tree computed by exact object-space algorithms is guaranteed to be optimal. This improves the time taken by the *path validation* step since the number of potential paths to validate is

the smallest. However, using exact visibility to compute the visibility tree is compute-intensive and may require a long time. Such methods have been applied for early specular reflection [Funkhouser et al., 2004] for limited scenes with a cell-and-portal structure. Applying these algorithms for early specular reflections for general scenes is computationally expensive and requires a robust implementation. One possibility is to apply recently developed beam tracing algorithms [Overbeck et al., 2007] for early specular reflection. Like sample-based approaches, no known exact object-space from-region algorithm has been applied to improve the finite-edge diffraction computation. It is possible to apply aspect graphs [Gigus et al., 1991] or the visibility complex [Durand et al., 1996, Durand et al., 1997] to compute from-region visibility from a diffracting edge. However, the computational complexity of such methods – $O(n^9)$ for aspect graphs and $O(n^4)$ for the visibility complex, where $n$ is the number of scene primitives – makes them impractical for even simple scenes. Moreover, these are global visibility algorithms and compute visibility from all points in the scene; they cannot be used to compute visibility from a given list of diffracting edges.

**Object-Space Conservative Approaches**: Given the computational complexity of exact approaches and aliasing issues with sampling-based approaches, conservative approaches offer an interesting alternative. Conservative approaches have lower run-time complexity as compared to exact approaches and do not suffer from the aliasing errors that are common in sample-based approaches. However, the PVS computed by conservative approaches is larger than that computed by exact or sample-based visibility approaches, therefore the size of the visibility tree will be larger. Thus, the path validation step will take longer since there are more paths to validate. Figure 3.29 compares different image-source methods. The main difference between these methods is in terms of which image sources they choose to compute [Funkhouser et al., 1998, Laine et al., 2009]. A naïve image-source method computes image sources for all primitives in the scene [Allen and Berkley, 1979]. Beam tracing methods compute the image

sources for exactly visible primitives from a source (or image source). Methods based on beam tracing, like accelerated beam tracing [Laine et al., 2009], compute image sources for every primitive inside the beam volume. Our approach, shown in Figure 3.29(d), finds a conservative PVS from a source and computes the image sources for the primitives in the conservative PVS. We have presented an approach based on a conservative from-point [Chandak et al., 2009] and a conservative from-region [Antani et al., 2011c] algorithm to compute early specular reflection and finite-edge diffraction. Accelerated Beam Tracing [Laine et al., 2009], a variant of beam tracing, has also been applied for early specular reflections. Regarding conservative visibility algorithms for finite-edge diffraction, only view-frustum culling has been applied [Svensson, 1999] and our approach for reducing edge pairs for edge diffraction [Antani et al., 2011c] is the only known implementation which uses visibility algorithms for finite edge diffraction.

# Chapter 5

# Audio Processing

The audio processing step generates the output audio which can be heard by a listener using headphones or speakers. In this section, we provide details on the real-time audio processing step [Chandak et al., 2011]. The audio processing techniques presented in this chapter have been used in conjunction with the propagation techniques described earlier in the thesis as well as other sound propagation methods developed in the GAMMA group at UNC-Chapel Hill [Taylor et al., 2009b, Antani et al., 2011b].

## 5.1 Integration with Sound Propagation

The output from the sound propagation step is provided as input to the audio processing step. Figure 5.1 provides an overview of the connection between these two steps. Sound propagation algorithms can either generate a list of specular, diffuse, and diffracted paths (or image sources) or compute IRs from these paths from each source to the receiver. We take these image-sources or IRs and generate artifact-free audio output. However, there are many key challenges that arise in efficiently generating artifact-free audio output for interactive sound rendering applications.

Firstly, due to the interactive nature of such applications, audio must be streamed to the user in real-time. For example, voice chat among players in MMOs must be played back to the player in real-time for effective in-game communication. Therefore,

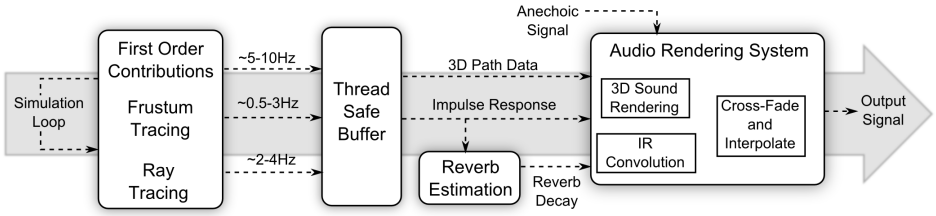



Figure 5.1: Integration of audio processing with sound propagation. *An overview of the integration of audio processing with the sound propagation engine. Sound propagation engine updates the computed paths in a thread safe buffer. The direct path and first order reflection paths are updated at higher frequency. The audio processing step 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.*

the audio played at the source is sent to the signal processing pipeline in small chunks, called *audio frames*. Sound propagation and binaural audio effects are applied to each audio frame. The size of the audio frames is chosen by the application, typically based on allowable latency in the system and the time required to apply propagation effects on a given audio frame. The size of an audio frame could be anywhere from 10-50 ms depending on the application, and therefore 20-100 audio frames need to be processed per second. Thus, the audio processing pipeline in interactive applications needs to handle streaming audio frames in real-time.

Secondly, these interactive applications could involve complex scenes, and current algorithms may not be able to perform sound propagation at 20-100 frames per second. Therefore, sound propagation is performed asynchronously with respect to audio processing, as shown in Figure 5.1. For interactive applications, 50-100 ms is an acceptable latency to update the IRs in a scene [Savioja et al., 1999] and therefore, the sound propagation should be able to asynchronously update the image sources or IRs at 10-20 FPS.

Thirdly, as these interactive applications involve dynamic scenes with moving source and moving receiver (MS-MR), it is important to reduce any artifacts due to the dynamic scenarios, and the final audio signal must be smooth. Due to the movement of sources

and receiver, the IRs used for two subsequent audio frames may be different, as these IRs are updated asynchronously. This could lead to a discontinuity in the final audio signal at the boundary between two audio frames. Hence, interpolation of image sources or IRs, or smoothing of the audio at the frame boundaries is performed to ensure artifact-free output audio signal.

Finally, as the application may have a large number of sound sources, the audio processing must be efficient and should be able to handle the convolution of IRs for a reasonable number of sound sources at 20-100 FPS.

### 5.1.1 Impulse Response

Most indoor sound propagation algorithms operate on the assumption that the propagation of sound from a source to a receiver can be modeled by a *linear, time-invariant* (LTI) system. As a result, the propagation effects are described by computing an *impulse response* (IR) between the source and the receiver. The IR encodes spatial information about the size of the scene and the positions of objects in it by storing the arrival of sound waves from the source to the receiver as a function of time. Given an arbitrary sound signal emitted by the source, the signal heard at the receiver can be obtained by convolving the source signal with the IR. However, this assumption is only valid for static source and static receiver (SS-SR) scenarios. For moving source and moving receiver (MS-MR) scenarios, there is no well-defined IR between the source and the receiver. Fortunately, propagation for MS-MR scenarios can be modeled using *time-varying IRs*. We shall use this observation in Section 5.5 to develop audio processing for MS-MR scenarios.

Figure 5.2 shows an example of an IR; it is usually divided into direct response (DR), early response (ER), and late response (LR). For example, in a church or a cathedral, an IR could be 2-3 seconds long, as sound emitted by a source will reflect and scatter and reach the receiver with a delay of upto 2-3 seconds, decaying until it is not audible. For

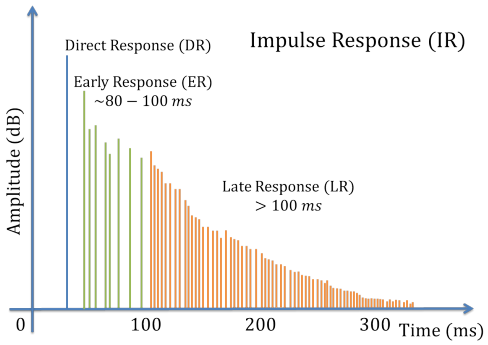



Figure 5.2: Example of a typical IR. *It is composed of direct response (DR), early response (ER), and late response (LR). DR is the direct sound from the source to the receiver, ER is typically the sound arriving at the receiver in the first 80-100 ms, and LR is the sound arriving at the receiver after 100 ms.*

such an IR, the DR is the direct sound from the source to the receiver, ER is typically the sound reaching the receiver within the first 80-100 ms, and LR is the sound reaching the receiver after 100 ms of being emitted from the source.

**Specular and Diffraction IR:** Specular reflections and diffraction are formulated as a function of 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 added 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:** 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 a pressure IR for audio processing. We take the square root of the energy response to compute a pressure IR for each frequency band. This IR is combined with specular and diffraction IRs to produce the final IR used in audio processing.

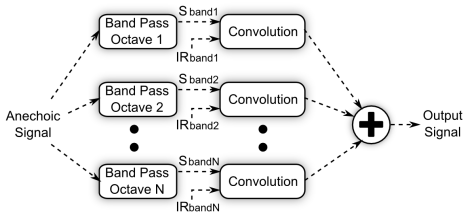



Figure 5.3: IR convolution. *The input audio signal $S$ is band passed into $N$ octave bands which are convolved with the IR of the corresponding band.*

## 5.2 Binaural Audio

In a typical sound propagation simulation, many sound waves reach the listener from different directions. These waves scatter around the listener's head, shoulder, and torso and provide cues regarding the direction of the incoming wave. This scattering effect can be encoded in a Head-Related Impulse Response (HRIR) [Algazi et al., 2001]. Thus, to produce realistic 3D sound rendering, 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 processing 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 processing of large numbers of sound sources [Tsingos et al., 2004, Wand and Straßer, 2004]. These approaches can be integrated in our sound rendering system.

## 5.3 Late Reverberation

The propagation paths computed by the sound propagation step 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. Most audio APIs like XAudio2 and FMOD support the use of user-defined filters and other audio

processing components through interfaces for creating filters. 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 audio API 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, which is defined as the time required for the energy to decay by 60 dB. We estimate the value of $RT_{60}$ from an IR and the reverberation filter is then updated with the estimate. This approach provides a simple, efficient way of complementing the early IRs with late reverberation effects.

### 5.3.1 Reverberation Estimation using Eyring Model

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

$$E(t) = E_0 e^{\frac{cS}{4V} t \log(1-\alpha)} \tag{5.1}$$

where $c$ is the speed of sound, $S$ is the total absorbing surface area of the room, $V$ is the volume of the room and $\alpha$ is the average absorption coefficient of the surfaces in the room.

Given the IR, we perform a simple linear least-squares fit to the IR in log-space. This gives us an exponential curve which fits the IR and can easily be extrapolated to generate the late reverberation. 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 processing step to generate late reverberation effects.

Note that Equation (5.1) 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

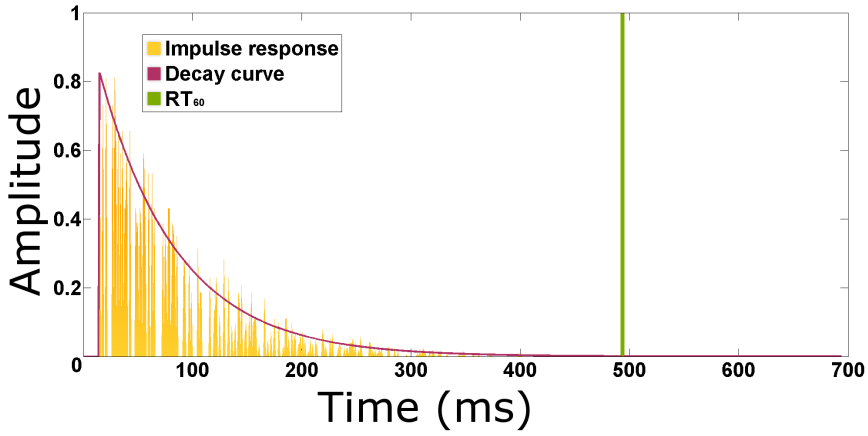



Figure 5.4: Extrapolating the IR to estimate late reverberation. *The red curve is obtained from a least-squares fit (in log-space) of the energy IR computed by GA, and is used to add the reverberant tail to the IR.*

good approximation for large interior spaces. Other models exist for coupled rooms [Summers et al., 2004], but they would require fitting multiple curves to the IR, and the number of curves to fit would depend on the number of rooms in the scene. Other pseudo-physical techniques have been proposed to estimate the reverberation tail by predicting energy decay curves in image-source simulation [Lehmann and Johansson, 2010]. All these approaches are complementary to the techniques present in the thesis and in the interests of speed and simplicity, we have chosen to use a single-room model.

### 5.3.2 Results

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 over 10 frames. During testing, we vary the number of image sources and consequently the number of peaks in the impulse response. Our approach can also be used if an IR is provided directly for reverberation estimation. The reverberation calculation is not threaded due to its minimal time cost. The results are summarized in Table 5.1.

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

Table 5.1: *Timing performance for late reverberation estimation.*

## 5.4 Dynamic Scenes

In many interactive applications, the source and listener movements could be large. This can lead to sudden changes in the propagation paths (i.e. delay and attenuation) from a source to a listener. New paths may suddenly appear when a listener 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 [Wenzel et al., 2000, Savioja et al., 2002, Savioja et al., 1999] and windowing schemes [Siltanen et al., 2009] to reduce the audio artifacts. There may be hundreds of such image sources depending on the orders of reflection modeled during propagation. Handling such a large number of image sources is a challenge in itself. Techniques proposed by Tsingos et al. [Tsingos et al., 2004] can be used to perceptually optimize the processing of image sources.

### Parameter Interpolation

We perform parameter interpolation of propagation paths for audio processing to achieve artifact-free audio output. 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. Such an approach is physically intuitive and different interpolation schemes can be applied [Tsingos, 2001b, Tsingos et al., 2004]. We treat each path from a source to a listener

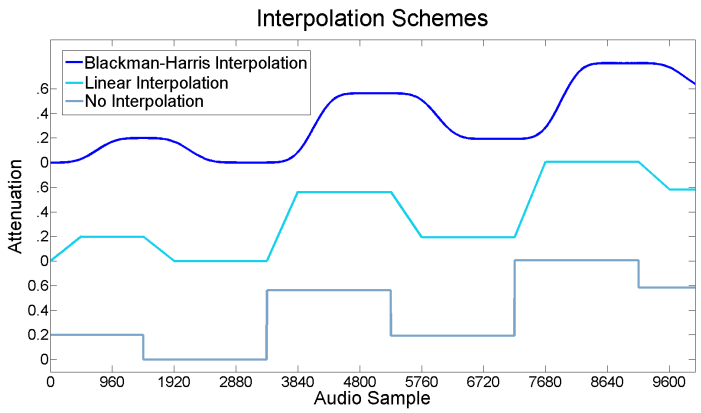



Figure 5.5: Interpolation schemes applied for attenuation interpolation. *Discontinuity in attenuation between two audio frames interpolated with linear interpolation and Blackman-Harris interpolation. Delay interpolation is performed using a linear interpolation. Variable fractional delays due to linear delay interpolation are handled by apply low order Lagrange fractional delay filter on a supersampled input audio signal during the audio processing step.*

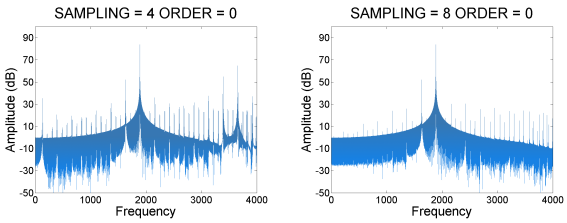
as a parameter and represent it as an equivalent image source, i.e. delay, attenuation, and direction in 3D space relative to the listener. Each image source is treated as an independent audio source during audio processing. Thus, changes in the paths are equivalent to changes in the corresponding image sources.

As an image source changes, its attenuation, delay, or direction relative to listener 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 per sample and leads to a 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. Figure 5.5 shows interpolated attenuation per sample for an image source with attenuation discontinuities.

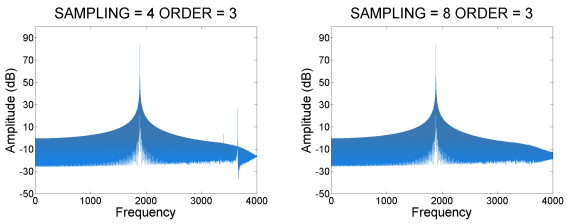
## Variable Fractional Delay

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 listeners move in a virtual environment. Rounding off the interpolated delays to the nearest integer as sources and listeners move can lead to noticeable artifacts in the final audio (see Figure 5.6). Thus, we require efficient *variable* fractional delay filter that can provide the fidelity and speed required in virtual environments. A good survey of FIR and IIR filter design for fractional delay filter is available [Laakso et al., 1996].

We use Lagrange interpolation since it has explicit formulas to construct a fractional delay filter and flat-frequency response for low-frequencies. Combined with a super-sampled 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, 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, an optimal supersampled input signal and low-order polynomial interpolator can be chosen to minimize computational and space complexity. Ideally, a highly supersampled input signal is required (see Figure 5.6) to achieve an SNR of 60 dB or more 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].




(a) Lagrange Filter Order = 0



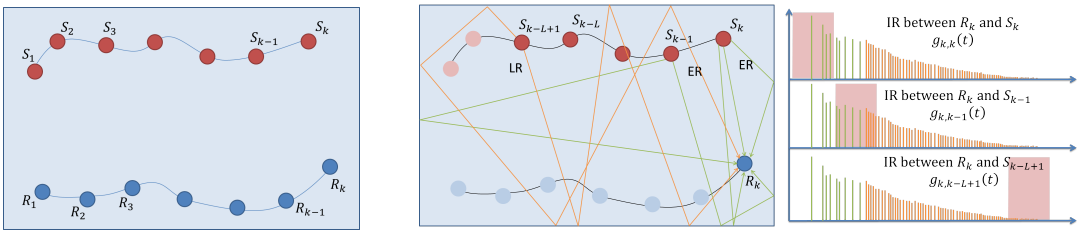

(b) Lagrange Filter Order = 3

Figure 5.6: Fractional delay filter to obtain Doppler effect. *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 listener 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.*

## 5.5 Moving Source and Moving Receiver

In this section, we present our audio processing technique for moving source and moving receiver (MS-MR) scenarios [Chandak et al., 2011]. Moreover, our technique can also be specialized for moving source and static receiver (MS-SR) and static source and moving receiver (SS-MR) scenarios. Our technique takes an IR as input from the sound propagation step, and generates smooth audio output. Table 5.2 presents the notation used for the discussion. Note that the audio emitted in different frames could overlap in time, depending on the window function chosen. A windowing function limits the support of a signal. For clarity of exposition, we use a square window function such that $w(t) = 1$ for $0 \leq t < \Delta T$, and $w(t) = 0$ elsewhere. However, to compute a smooth audio signal, an overlapping windowing function like the Hamming window is used. Next, we present a mathematical formulation to compute the audio signal heard at the receiver in a given frame, $r_k(t)$, for MS-MR scenarios (see Figure 5.7 and Figure 5.8).

In a given MS-MR scenario, sound reaching $R_k$ arrives from many previous source




(a) (b)

Figure 5.7: Sound propagation for MS-MR scenarios. *(a) Path for moving source and moving receiver. Source and receiver positions are updated during each frame.* $S_k$ *and* $R_k$ *denote the position of source and receiver during the* $k^{th}$ *frame. (b) The sound received at the receiver* $R_k$ *comes from the source positions* $\{S_k(t), S_{k-1}, \ldots, S_{k-L+1}\}$*, where corresponding audio frames* $\{s_k(t), s_{k-1}(t), \ldots, s_{k-L+1}(t)\}$ *are modified by direct and early response for the latest source positions* $\{S_k, S_{k-1}\}$ *and by late response from the earlier source positions* $\{S_{k-2}, \ldots, S_{k-L+1}\}$*.*

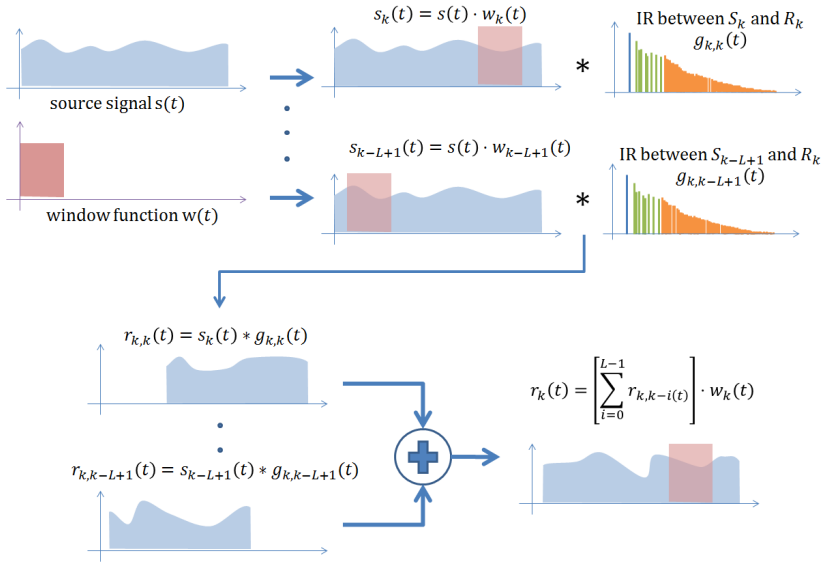



Figure 5.8: Audio processing pipeline for MS-MR scenarios. *The input audio signal* $s(t)$ *is multiplied with window function* $w(t)$ *to generate the audio frames* $\{s_k(t), s_{k-1}(t), \ldots, s_{k-L+1}(t)\}$ *for source positions* $\{S_k, S_{k-1}, \ldots, S_{k-L+1}\}$*. These audio frames are convolved with the corresponding IRs* $\{g_{k,k}(t), g_{k,k-1}(t), \ldots, g_{k,k-L+1}(t)\}$ *and multiplied with the window function* $w_k(t)$ *to generate the audio frame for the receiver position* $R_k$*.*

| Notation | |
|---|---|
| $\cdot$ | multiplication operator |
| $*$ | convolution operator |
| $t$ | time in seconds |
| $\Delta T$ | length of an audio frame in seconds |
| $N$ | frame rate ($= 1/\Delta T$) |
| $k$ | frame number (range from 0 to $\infty$) |
| $S_k$ | position of source in frame $k$ |
| $R_k$ | position of receiver in frame $k$ |
| $w(t)$ | window function<br>(non-zero for $\Delta T$ time period) |
| $w_k(t)$ | window function used to select frame $k$<br>$= w(t - k\Delta T)$ |
| $s(t)$ | signal emitted by source |
| $s_k(t)$ | signal emitted by source in frame $k$<br>$= s(t) \cdot w_k(t)$ |
| $r_k(t)$ | signal received by receiver in frame $k$ |
| $r(t)$ | signal received by receiver<br>$= \sum_{k=0}^{\infty} r_k(t)$ |
| $g_{k_1,k_2}(t)$ | impulse response between $R_{k_1}$ and $S_{k_2}$ |
| $l$ | length of impulse response in seconds<br>$g_{k_1,k_2}(t) = 0$ for $t > L$ |
| $L$ | length of impulse response in frames<br>$= l/\Delta T$ |

Table 5.2: Notation table. *Definitions of symbols and other notation used in this section.*

positions $\{S_k, S_{k-1}, ..., S_{k-L+1}\}$, as the sound emitted by previous source positions (upto $L$ audio frames ago) propagates through the scene before arriving at $R_k$. For any previous source position $S_{k-i}$, the sound reaching the listener can be obtained by convolving $s_{k-i}(t)$, the sound emitted from the source in frame $k-i$, with $g_{k,k-i}(t)$, the IR between $S_{k-i}$ and $R_k$. Since the listener is at $R_k$ only during audio frame $k$, we isolate the relevant portion of the convolved signal by applying a square window $w_k(t)$. This results in the following equation:

$$r_k(t) = \sum_{i=0}^{L-1} [g_{k,k-i}(t) * s_{k-i}(t)] \cdot w_k(t) \tag{5.2}$$

Equation 5.2 correctly models the audio signal that will reach the receiver $R_k$ from prior

source positions. Note that none of the IRs $\{g_{k,k}(t), g_{k,k-1}(t), \ldots, g_{k,k-L+1}(t)\}$ were computed in previous frames, and therefore $L$ new IRs need to be computed during every frame between each of the prior $L$ source positions, $\{S_k, S_{k-1}, \ldots, S_{k-L+1}\}$, and the current receiver position $R_k$. A naïve sound rendering technique may compute $L$ new IRs during sound propagation per frame and may perform $L$ full convolutions of the IRs with the current audio frame. For example, for an IR of length 1 second and audio frames of size 100 ms, sound propagations for 10 different IRs would be performed per frame and 10 full convolutions between the IRs and the current audio frame would be computed per frame. This may not fit within the convolution budget of 10-50 ms per frame or the 50-100 ms budget per frame for sound propagation, and may lead to glitches in the final audio signal or high latency in updating the IRs.

**Key Observations**: Our formulation is based on the following property:

**Lemma 1**: *For a given receiver position $R_k$ and source position $S_{k-i}$, only the interval of the IR $g_{k,k-i}(t)$ in $[(i-1)\Delta T, (i+1)\Delta T]$ will contribute to $r_k(t)$.*

Intuitively, the sound emitted at source position $S_{k-i}$ can arrive at $R_k$ only within a delay of $[(i-1)\Delta T, (i+1)\Delta T]$, assuming that a square window $w_k(t)$ is applied to compute the final audio signal at $R_k$. Thus, only an interval of the IR $g_{k,k-i}(t)$ needs to be computed to generate the final audio signal at $R_k$.

Furthermore, a block convolution framework is well-suited for audio processing in MS-MR scenarios as different blocks of IRs can be convolved with corresponding audio blocks. To compute the final audio signal at $R_k$, the block of the IR $g_{k,k-i}(t)$ in the interval $[(i-1)\Delta T, (i+1)\Delta T]$ is convolved with the input audio frame $s_{k-i}(t)$ to generate the audio signal at $R_k$. This minimizes the computation in the audio processing pipeline, and we will show that this leads to efficient sound rendering for MS-MR scenarios.

### 5.5.1 Audio processing for SS-MR and MS-SR

Equation 5.2 can be specialized to derive similar equations for SS-MR and MS-SR scenarios. In SS-MR scenarios, the source is static, i.e., $S_1 = S_2 = S_3 \ldots = S_k$. Therefore, $g_{k,k-i}(t) = g_{k,k}(t)$, and $g_{k,k-i}(t)$ can be moved out of the summation in Equation 5.2, yielding the following simplified equation:

$$r_k(t) = [g_{k,k}(t) * \sum_{i=0}^{L-1} s_{k-i}(t)] \cdot w_k(t) \tag{5.3}$$

Hence, SS-MR scenarios are easier to model, as they require computation of only one IR, $g_{k,k}(t)$, per frame. However, the past audio frames $\{s_k(t), s_{k-1}(t), \ldots, s_{k-L+1}(t)\}$ need to be buffered in memory.

In MS-SR scenarios, the receiver is static, i.e., $R_1 = R_2 = R_3 \ldots = R_k$. Therefore, $g_{k-i,k}(t) = g_{k,k}(t)$, i.e., of the $L$ IRs needed in every frame, $L-1$ would have been computed in the previous frames, and can be re-used. This observation results in the following simplified equation:

$$r_k(t) = [\sum_{k=0}^{L-1} r_{k,k-i}(t)] \cdot w_k(t) \tag{5.4}$$

$$r_{k,k-i}(t) = g_{k,k-i}(t) * s_{k-i}(t) \tag{5.5}$$

Thus, MS-SR scenarios are easier to model, as they require the computation of only one IR, $g_{k,k}(t)$, per frame. Moreover, the convolved output from the previous $L-1$ frames, $\{r_{k,k-1}(t), r_{k,k-1}(t), \ldots, r_{k,k-L+1}(t)\}$, can be cached in a buffer and used to compute $r_k(t)$.

### 5.5.2 Audio processing for MS-MR

In this section, we present our audio processing pipeline. The pipeline uses block convolution to compute the final audio signal for MS-MR scenarios. During block convolution for receiver $R_k$, the block $[(i-1)\Delta T, (i+1)\Delta T]$ of IR $g_{k,k-i}(t)$ is convolved with the source signal block $s_{k-i}(t)$. The results from these block convolutions for the past $L$ source positions are added together and played back at the receiver after applying the window function $w_k(t)$.

In our formulation, the block convolution is implemented by computing a short-time Fourier transform (STFT) of each frame of input audio and the IRs. Let the STFT of $s_k(t)$ be $s_k(\omega)$ and the STFT of $g_{k,k-i}(t)$ for the interval $[(i-1)\Delta T, (i+1)\Delta T]$ be $g_{k,k-i}(\omega)$. Then the final audio $r_k(\omega)$ can be computed as follows:

$$r_k(\omega) = \sum_{i=0}^{L-1} g_{k,k-i}(\omega) \cdot s_{k-i}(\omega) \tag{5.6}$$

Finally, we compute an inverse-STFT to compute $r_k(t)$ from $r_k(\omega)$. Note that as the IRs change from one frame to another, there may be discontinuities in the final audio signal between the boundaries of consecutive frames. Therefore windowing functions, such as the Hamming window, are used instead of a square window to minimize such artifacts.

### 5.5.3 Implementation and Results

In this section, we present the results of performance evaluation and benchmarking experiments carried out on our proposed framework. All tests were carried out on an Intel Core 2 Quad desktop with 4GB RAM running Windows 7. Figure 5.9 shows the scenes we used for benchmarking. We first summarize the performance of our audio processing pipeline. All timings are reported for a single CPU core. As Table 5.5.3 shows, our audio processing pipeline based on block convolution can efficiently handle

(a) (b) (c)

Figure 5.9: Benchmark scenes for MS-MR modeling. *(a) Room (876 triangles), (b) Hall (180 triangles), (c) Sigyn (566 triangles).*

| Frame Size<br>IR length | 10 ms | 20 ms | 50 ms | 100 ms |
|---|---|---|---|---|
| 1 sec | 50.6 ms | 18.3 ms | 10.4 ms | 7.9 ms |
| 2 sec | 97.8 ms | 36.9 ms | 19.1 ms | 14.8 ms |
| 3 sec | 147.0 ms | 54.4 ms | 27.0 ms | 21.2 ms |

Table 5.3: Timing results for audio processing based on STFT. *The table shows the time needed to compute 1 second of final audio output for different frame sizes and the length of the IR. Thus, for IR of length 2 sec and audio frames of size 50 ms, our audio processing takes about 19 ms to compute 1 second of output audio, and therefore it can efficiently handle up to 50 sound sources.*

a large number of sound sources per audio frame.

### 5.5.4 Conclusion, Limitations, and Future Work

Real-time audio processing for dynamic scenes is a challenging problem. We have presented a framework for performing real-time audio processing in scenes where both the sound sources and the receiver may move simultaneously. A key component of this framework is an equation describing how time-dependent impulse responses can be convolved with streaming audio emitted from a sound source to determine the sound signal arriving at the receiver. This equation can be used to derive simpler equations which describe sound rendering in situations where either the sources or the receiver or both may be static. Correctly performing audio processing for scenes with dynamic geometry remains an interesting avenue for future work. Such scenes commonly occur in interactive virtual environments and video games. Another important question that remains

to be addressed is that of the perceptual impact of correctly modeling audio processing for dynamic scenes.

## 5.6 Efficient Propagation for Dynamic Scenes

In this section, we present two efficient modifications of sound propagation algorithms for modeling MS-MR scenarios. First, based on Lemma 1, we can efficiently compute $L$ IRs from the prior source positions, $\{S_k, S_{k-1}, \ldots, S_{k-L+1}\}$ using the image-source method. Second, we extend a precomputation based sound propagation algorithm [Antani et al., 2011b] that stores responses from the sources at the surface samples, by observing that the response at the surface samples from past source locations $\{S_k, S_{k-1}, \ldots, S_{k-L+1}\}$ would have been computed in previous frames and can be stored to efficiently generate $L$ IRs corresponding to the new receiver position.

### 5.6.1 MS-MR Image Source Method

Figure 5.10 gives a short overview of the image-source method. This method is used to compute specular reflections and can be extended to handle edge diffraction. The image-source method can be divided into two main steps: (a) *image tree* construction based on the geometric representation of the environment, and (b) *path validation* to compute valid specular paths in the image tree.

In the first step, we construct an image tree from receiver $R_k$ up to a user-specified $m$ orders of reflection. It is constructed by recursively reflecting the receiver or an image of the receiver over the scene primitives and storing the image-sources as nodes in the tree. Each node in the tree thus corresponds to a single specular propagation path. We denote this image tree by $T(R_k, m)$. We construct an image tree using the receiver position as the root of the tree as it leads to a more efficient implementation over constructing an image tree from the source position. Figure 5.10 shows an example of an image tree.

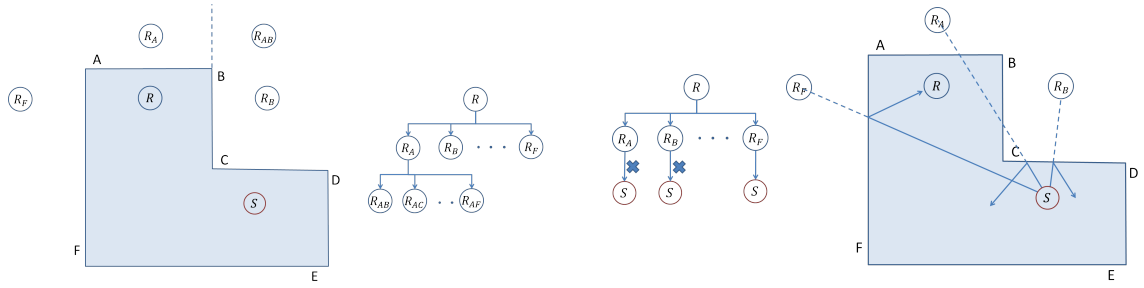
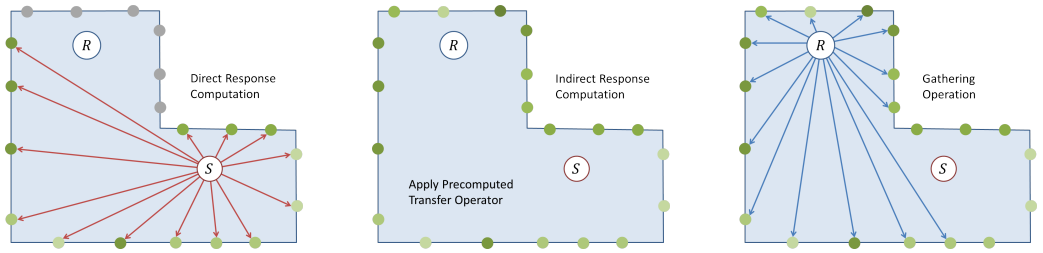



Figure 5.10: Our modified image-source method for MS-MR. *(a) An image tree is constructed by reflecting the receiver recursively across the surfaces in the scenes. Here, receiver $R$ is reflected across the surfaces $\{A, B, C, D, E, F\}$ to construct first order images. Image $R_A$ is reflected across surface $B$ to construct higher order image $R_{AB}$. An image tree, $T(R, m)$, is constructed from these image sources. (b) A path validation is performed by attaching the source $S$ to nodes in the image tree to compute valid specular paths from receiver $R$ to the source $S$. A path is invalid if it is obstructed by objects in the scenes, e.g. image $R_A$, or does not lie within the scene primitive, e.g. image $R_B$. For MS-MR scenarios, we test that the delay of the path lie with the appropriate delay range ($[(i-1)\Delta T, (i+1)\Delta T]$) for receiver $R_k$ and source $S_{k-i}$.*

To compute the final specular paths from the image tree, the source position is required. Hence, in the second step, given a source position $S_k$, we traverse the tree, and for each node in $T(R_k, m)$, we determine which of the corresponding propagation paths are valid. Some of the paths may not be valid because of the following reasons: (a) a path from the source $S_k$ to an image-source or between two image-sources might be obstructed by other objects in the scene, or (b) a path may not lie within the scene primitive which induced the image source. In such cases, a specular path does not exist for the corresponding node in the image-source tree.

In MS-MR scenarios, path validation needs to be performed from all the source positions $\{S_k, S_{k-1}, ..., S_{k-L+1}\}$. However, we require only a portion of the IR in the interval $[(i-1)\Delta T, (i+1)\Delta T]$ for a given receiver position $R_k$ and source position $S_{k-i}$. Therefore, we use the time interval and compute the distance interval $[(i-1)c\Delta T, (i+1)c\Delta T]$, where $c$ is the speed of sound, to eliminate paths during the path validation step. A path is not considered valid if the length of the path lies outside this distance




(a) (b) (c)

Figure 5.11: Our modified Direct-to-Indirect Acoustic Radiance Transfer method. *(a) Direct responses are computed at the sample points from the source. (b) Indirect responses are computed at these sample points by performing a matrix-vector multiplication with the transfer matrix. The transfer matrix is computed in a pre-processing step. (c) The final IR is computed at the receiver by tracing rays (or "gathering") from the receiver. For MS-MR scenarios, the response from the past sources is stored at the surface samples and gathered at run time.*

interval. We will show in Section 5.6.3 that our formulation substantially reduces the overhead of computing image-source IRs for MS-MR scenarios.

### 5.6.2 MS-MR Direct-to-Indirect Acoustic Radiance Transfer

Direct-to-indirect acoustic radiance transfer is a recently proposed precomputation-based approach to compute high orders (up to 100 or more) of diffuse reflections interactively [Antani et al., 2011b]. The key idea is to sample the scene surfaces and precompute a complex-valued *transfer matrix*, which encodes higher-order diffuse reflections between the surface samples. At run-time, the IR is computed as follows: (a) Given the source position, compute a *direct response* at each sample point. (b) Given the direct response at the sample points, *indirect responses* are computed at these sample points by performing a matrix-vector multiplication with the transfer matrix. (c) Given the direct and indirect responses, the final IR is computed at the receiver by tracing rays (or "gathering") from the receiver.

In MS-MR scenarios, we need to compute the IRs from the previous $L$ sound source locations $\{S_k, S_{k-1}, ..., S_{k-L+1}\}$. A naïve implementation would perform all three steps above $L$ times. This is highly inefficient, and would lead to an $L$-fold slow-down over SS-

SR scenarios. We observe that in this case, all the indirect responses from the previous $L-1$ source locations $\{S_{k-1}, ..., S_{k-L+1}\}$ can be stored at the surface samples (as they are independent of the receiver position), and therefore only one indirect response from source position $S_k$ needs to be computed for each surface sample per frame. In the final step, the final IR can be gathered from $L$ different indirect responses. This is much more efficient than the naïve implementation, as can be seen by the overall performance of direct-to-indirect acoustic radiance transfer for MS-MR in Section 5.6.3.

### 5.6.3 Implementation and Results

In this section, we present the results of performance evaluation and benchmarking experiments carried out on our proposed framework. All tests were carried out on an Intel Core 2 Quad desktop with 4GB RAM running Windows 7. Figure 5.9 shows the scenes we used for benchmarking.

Table 5.4 quantifies the computational cost of modeling MS-MR scenarios in the image-source method for sound propagation. In our implementation, the image tree is computed using a single CPU core, whereas path validation is performed using all 4 CPU cores. Column 3 shows the time taken for computing the image tree for 2 orders of reflection. Columns 4 and 6 show the time taken for performing path validation, for SS-SR scenarios and MS-MR scenarios respectively. The computational overhead is shown in Column 8.

Table 5.5 quantifies the computational cost of modeling MS-MR scenarios using direct-to-indirect acoustic radiance transfer. Our implementation of this algorithm uses all 4 CPU cores. Column 3 shows the number of surface samples used for each scene. Column 4 shows the time taken to compute the direct response at each surface sample, and apply the transfer matrix to compute the indirect response at each surface sample. Columns 5 and 7 show the time taken for gathering the final IR at the receiver, for SS-SR scenarios and MS-MR scenarios respectively. The computational overhead is shown

| Scene | $\Delta$s | Tree Construction | SS-SR Validation | SS-SR Total | MS-MR Validation | MS-MR Total | Overhead |
|---|---|---|---|---|---|---|---|
| Hall | 180 | 0.185 s | 1.49 ms | 0.186 s | 3.91 ms | 0.189 s | 1.3 % |
| Room | 876 | 1.001 s | 12.09 ms | 1.013 s | 15.22 ms | 1.016 s | 0.3 % |
| Sigyn | 566 | 1.030 s | 11.09 ms | 1.041 s | 19.35 ms | 1.049 s | 0.8 % |

Table 5.4: MS-MR overhead for image-source method. *Computational overhead due to modeling MS-MR scenarios using the image-source method.*

| Scene | $\Delta$s | Surface Samples | Final Response | SS-SR Gather | SS-SR Total | MS-MR Gather | MS-MR Total | Overhead |
|---|---|---|---|---|---|---|---|---|
| Hall | 180 | 177 | 116.3 | 31.2 | 147.5 | 52.8 | 169.1 | 14.6 % |
| Room | 876 | 252 | 126.7 | 36.4 | 163.1 | 76.9 | 203.6 | 24.8 % |
| Sigyn | 566 | 1024 | 369.2 | 122.9 | 492.1 | 234.5 | 603.7 | 22.7 % |

Table 5.5: MS-MR overhead for D2I method. *Computational overhead due to modeling MS-MR scenarios using direct-to-indirect acoustic radiance transfer. All the timings are in milliseconds.*

in Column 9.

### 5.6.4 Conclusion, Limitations, and Future Work

We can efficiently model propagation for MS-MR scenarios, as for a receiver position in a given audio frame, we do not need full impulse responses from all previous source positions. This insight allows us to suitably modify sound propagation algorithms (such as the image-source method or direct-to-indirect acoustic radiance transfer) to handle moving sources and a moving receiver without incurring a significant computational cost as compared to the naïve approach of computing full impulse responses from all previous source positions. Correctly performing sound propagation for scenes with dynamic geometry remains an interesting avenue for future work. Such scenes commonly occur in interactive virtual environments and video games. Another important question that remains to be addressed is that of the perceptual impact of correctly modeling sound propagation for dynamic scenes.

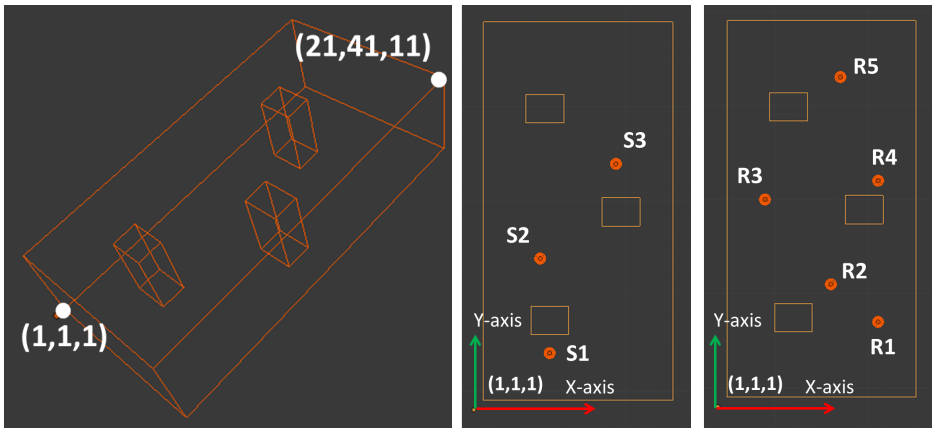
# Chapter 6

# Validation

Geometric acoustics methods are widely used for acoustics modeling due to their efficiency [CATT, 2002, Christensen, 2009]. They are limited in terms of accuracy but such limitations are not clearly understood. Experiments on simple setups have been performed to compare geometric acoustics results with measured data [Tsingos et al., 2002]. Most other studies [Bork, 2002] compare geometric acoustics methods by using standard objective acoustics parameters [ISO 3382-1:2009, 2009, ISO 3382-2:2008, 2008]. We compare the geometric acoustics techniques presented in the thesis with numerical simulation [Raghuvanshi et al., 2009]. In this chapter, we present our preliminary results.

## 6.1 Experimental Setup

Figure 6.1 shows the model we used for our validation experiments. The dimensions of the model are $20 \times 40 \times 10$ meters. The bounding box of the model is $[(1, 1, 1), (21, 41, 11)]$. All surfaces in the model have the same acoustic material (absorption coefficient of 0.1). Figure 6.1(b) and Figure 6.1(c) show the locations of the sound sources ($S1$, $S2$, and $S3$) and the receivers ($R1$, $R2$, $R3$, $R4$, and $R5$), respectively. Table 6.1 provides the locations of the sound sources and receivers.




(a) Perspective projection (b) (c)

Figure 6.1: Experimental setup. *(a) Factory scene used for validation. Scene dimensions are* $20 \times 40 \times 10$ *meters. (b) Source locations (*$S1$*,* $S2$*, and* $S3$*). (c) Receiver locations (*$R1$*,* $R2$*,* $R3$*,* $R4$*, and* $R5$*).*

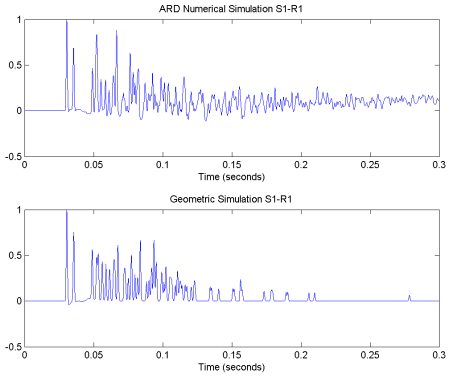
| Sources | Location | Receivers | Location |
|---|---|---|---|
| S1 | (8,6,4) | R1 | (17,9,4) |
| S2 | (7,17,4) | R2 | (12,13,4) |
| S3 | (15,26,4) | R3 | (5,22,4) |
| | | R4 | (17,24,4) |
| | | R5 | (13,35,4) |

Table 6.1: Locations of the sound sources ($S1$, $S2$, and $S3$) and the receivers ($R1$, $R2$, $R3$, $R4$, and $R5$).

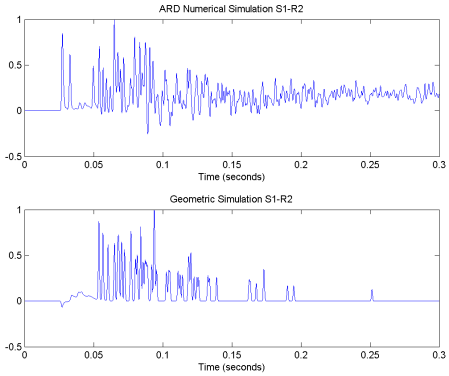
## 6.2 Implementation and Results

We use a numerical simulator based on the Adaptive Rectangular Decomposition (ARD) approach [Raghuvanshi et al., 2009] for our validation study. The numerical simulation was performed on the factory scene for frequencies up to 6000 Hz and for 300 milliseconds. We perform up to 4 orders of specular reflections and edge diffraction for geometric acoustics simulation. Figure 6.2, Figure 6.4, and Figure 6.6 compare the IRs computed by the ARD simulator with our geometric acoustics simulation for sources $S1$, $S2$, and $S3$. Spectrograms of these IRs are presented in Figure 6.3, Figure 6.5, and Figure 6.7. We show the relative magnitude error for the spectrograms computed in Figure 6.3, 6.5, and 6.7 in Figure 6.8. Table 6.2, 6.3, and 6.4 show the relative magni-

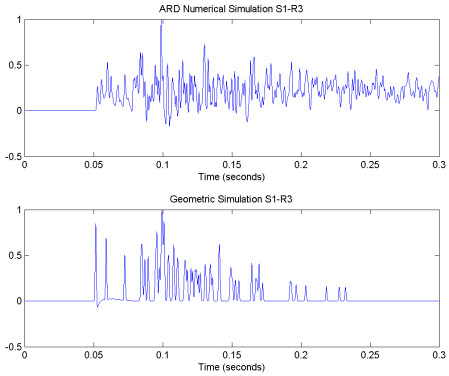
tude error (in decibel) for sources $S1$, $S2$, and $S3$. The results show good agreement for many source-receiver positions. However, they are large errors in the later part of the spectrogram and we believe that they are due to the fact that we limit geometric acoustics simulation to only 4 orders of reflections and diffraction. Other disagreements are due to mismatch between the diffraction contributions computed by ARD and our geometric acoustics simulation.



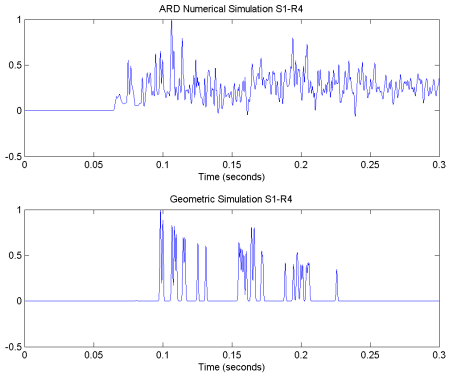

(a) S1-R1



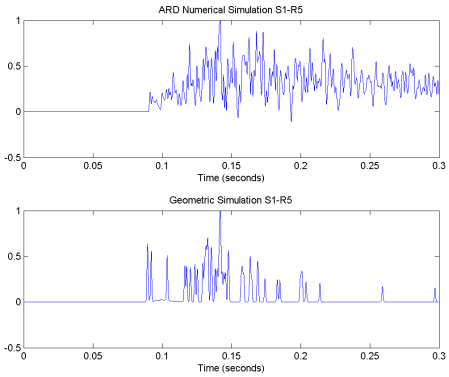

(b) S1-R2




(c) S1-R3

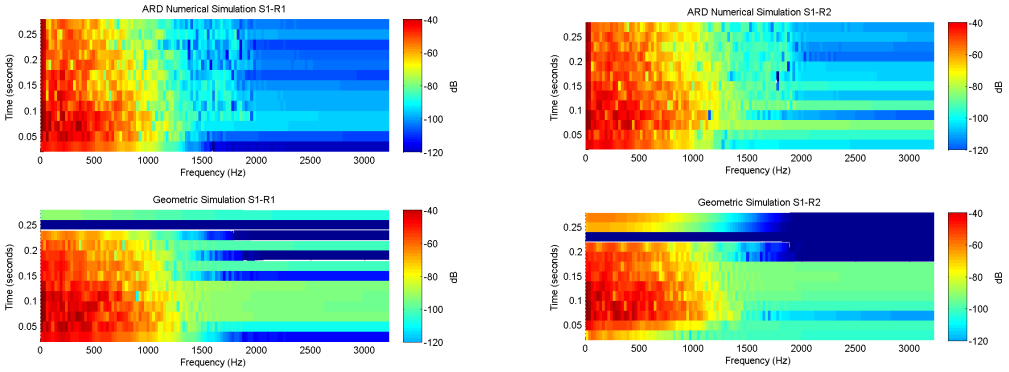



(d) S1-R4




(e) S1-R5

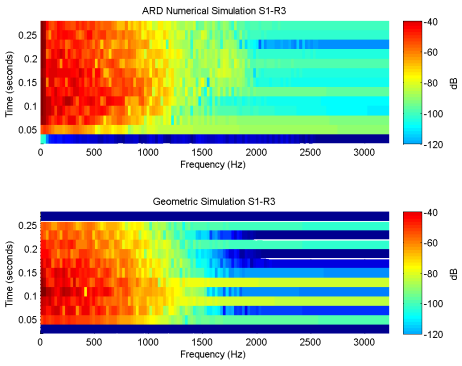
Figure 6.2: Results for source $S1$. *The top row in each figure corresponds to output from ARD numerical simulation and bottom row corresponds to results from our geometric acoustics simulations.*



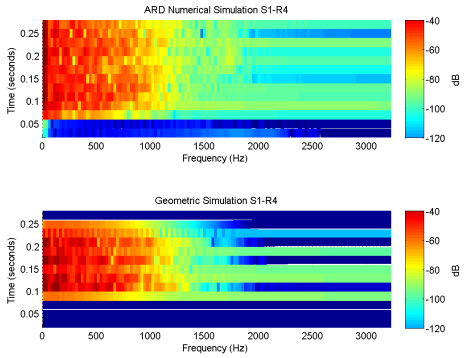

(a) S1-R1



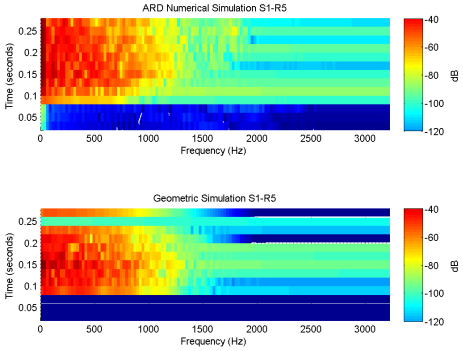

(b) S1-R2




(c) S1-R3

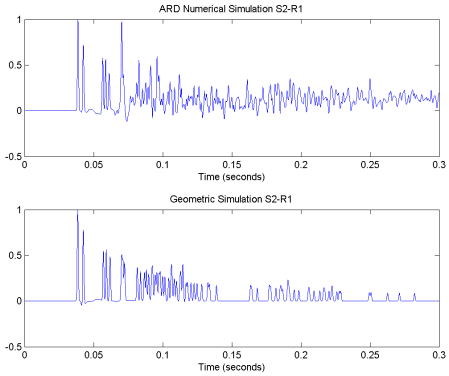



(d) S1-R4

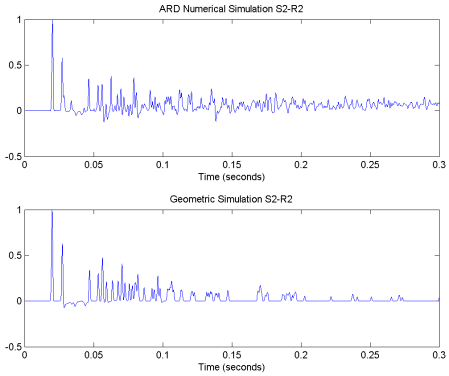



(e) S1-R5

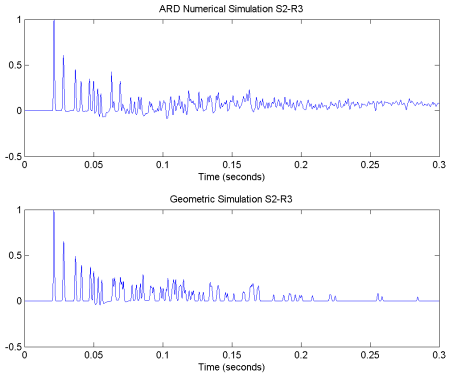
Figure 6.3: Spectrogram results for source $S1$. *The top row in each figure corresponds to the spectrogram of the output from ARD numerical simulation and bottom row corresponds to the spectrogram of the results from our geometric acoustics simulations.*



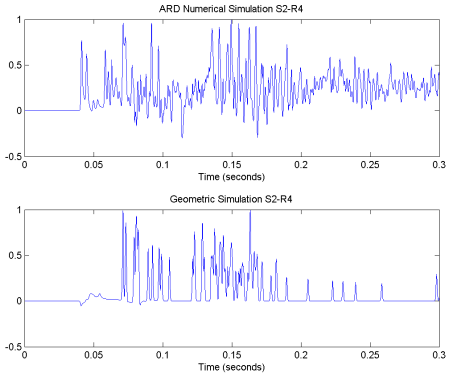

(a) S2-R1



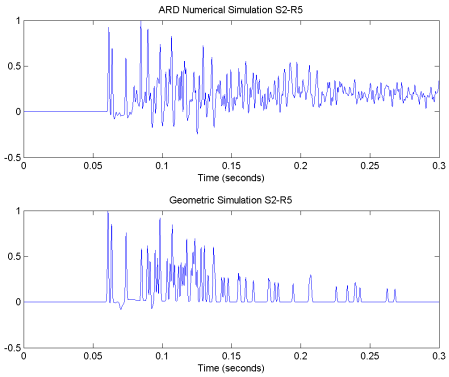

(b) S2-R2




(c) S2-R3

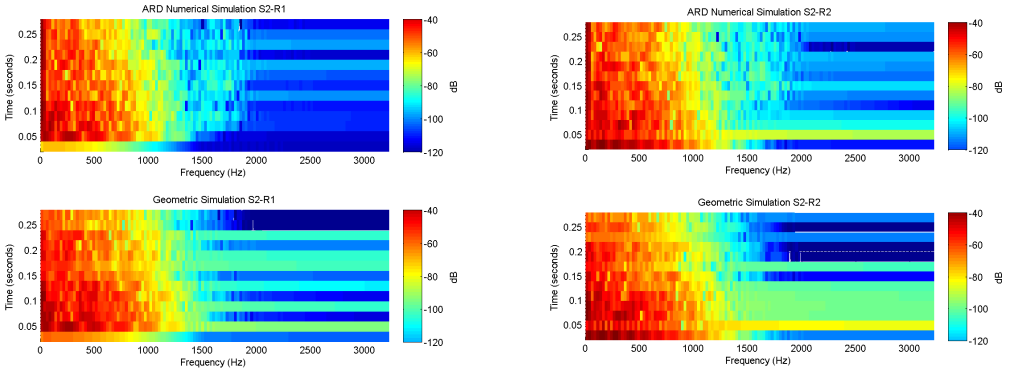



(d) S2-R4




(e) S2-R5

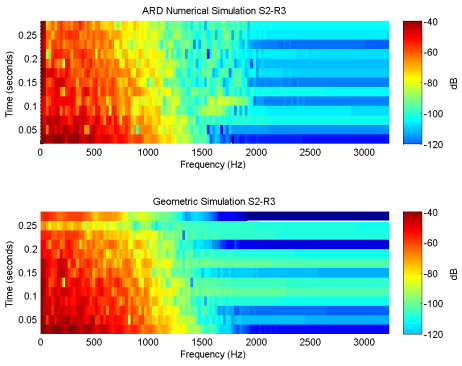
Figure 6.4: Results for source $S2$. *The top row in each figure corresponds to output from ARD numerical simulation and bottom row corresponds to results from our geometric acoustics simulations.*



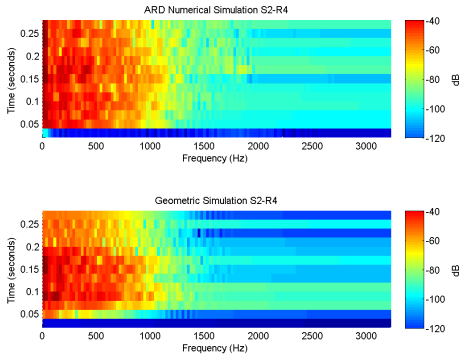

(a) S2-R1



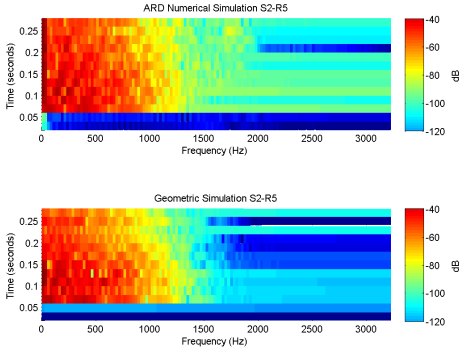

(b) S2-R2




(c) S2-R3

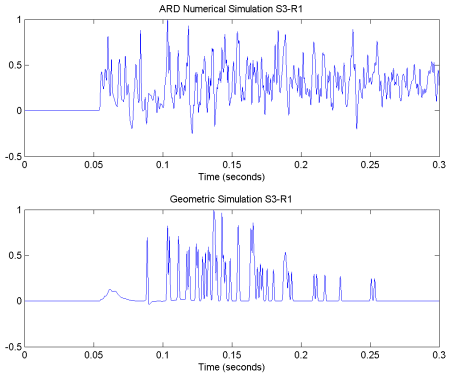



(d) S2-R4

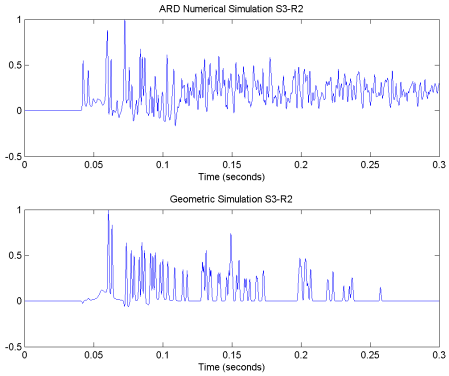



(e) S2-R5

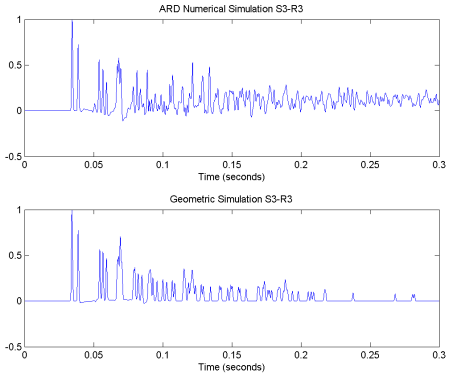
Figure 6.5: Spectrogram results for source $S2$. *The top row in each figure corresponds to the spectrogram of the output from ARD numerical simulation and bottom row corresponds to the spectrogram of the results from our geometric acoustics simulations.*



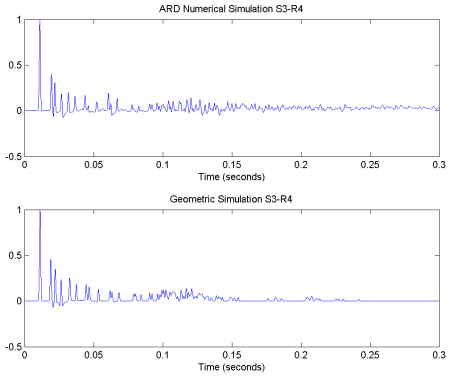

(a) S3-R1



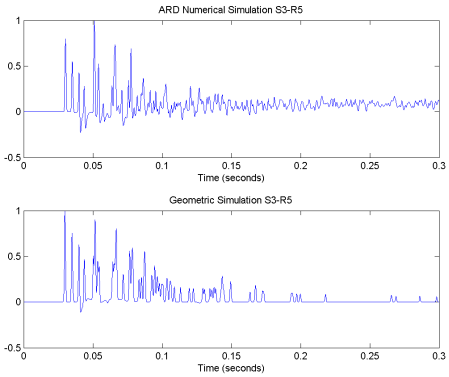

(b) S3-R2




(c) S3-R3

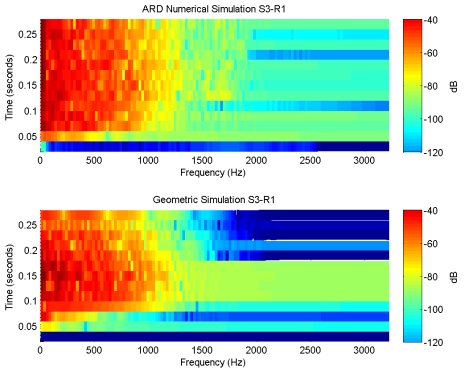



(d) S3-R4

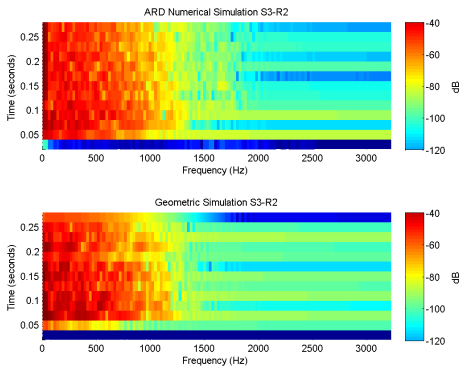



(e) S3-R5

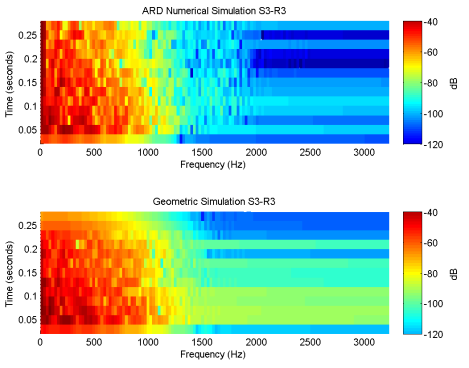
Figure 6.6: Results for source $S3$. *The top row in each figure corresponds to output from ARD numerical simulation and bottom row corresponds to results from our geometric acoustics simulations.*



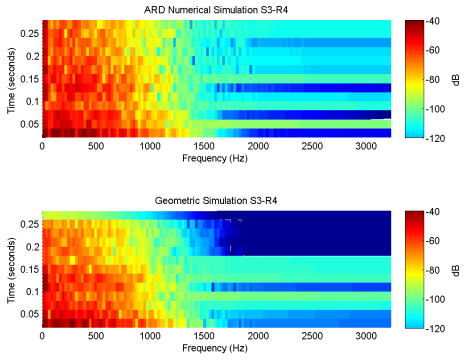

(a) S3-R1



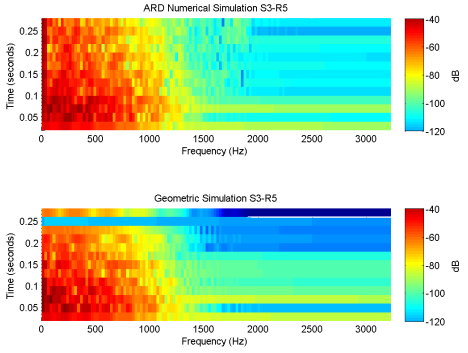

(b) S3-R2




(c) S3-R3




(d) S3-R4




(e) S3-R5

Figure 6.7: Spectrogram results for source $S3$. *The top row in each figure corresponds to the spectrogram of the output from ARD numerical simulation and bottom row corresponds to the spectrogram of the results from our geometric acoustics simulations.*

| Time (sec) → | .06 | .08 | .10 | .12 | .14 | .16 | .18 | .20 | .22 | .24 |
|---|---|---|---|---|---|---|---|---|---|---|
| R1 | -22.6 | -29.6 | -17.7 | -21.0 | -11.2 | -12.0 | -8.8 | -5.5 | -2.8 | 0 |
| R2 | -31.9 | -41.5 | -27.5 | -84.0 | -11.4 | -13.3 | -11.2 | -5.2 | 0 | -1.4 |
| R3 | -22.4 | -23.3 | -20.1 | -27.6 | -23.4 | -10.7 | -6.6 | -6.7 | -4.5 | -3.0 |
| R4 | 0 | -2.1 | -23.1 | -31.4 | -11.2 | -11.3 | -15.5 | -19.1 | -6.3 | -2.2 |
| R5 | 0 | -5.1 | -16.4 | -19.5 | -23.6 | -9.9 | -6.0 | -7.0 | -2.7 | 0 |

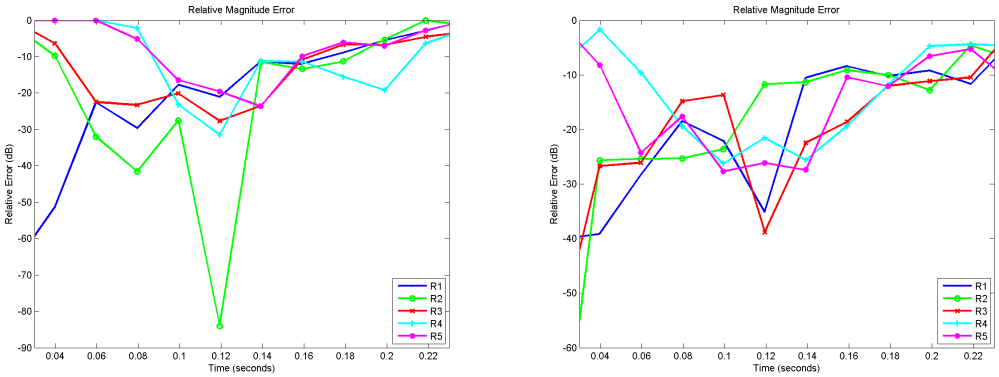
Table 6.2: Relative magnitude error (in decibel) for $S1$. *We compute relative error magnitude for the spectrograms computed in Figure 6.3. The rows highlight error for different receiver positions.*

| Time (sec) → | .06 | .08 | .10 | .12 | .14 | .16 | .18 | .20 | .22 | .24 |
|---|---|---|---|---|---|---|---|---|---|---|
| R1 | -28.3 | -18.4 | -22.0 | -35.0 | -10.5 | -8.4 | -10.2 | -9.1 | -11.6 | -3.8 |
| R2 | -25.3 | -25.3 | -23.5 | -11.7 | -11.3 | -9.1 | -10.0 | -12.7 | -4.6 | -7.0 |
| R3 | -26.1 | -14.8 | -13.6 | -38.8 | -22.4 | -18.5 | -11.9 | -11.1 | -10.4 | -1.7 |
| R4 | -9.5 | -19.4 | -26.2 | -21.5 | -25.5 | -19.3 | -11.8 | -4.7 | -4.3 | -4.5 |
| R5 | -24.2 | -17.6 | -27.6 | -26.1 | -27.4 | -10.4 | -12.0 | -6.5 | -5.2 | -11.5 |

Table 6.3: Relative magnitude error (in decibel) for $S2$. *We compute relative error magnitude for the spectrograms computed in Figure 6.5. The rows highlight error for different receiver positions.*

| Time (sec) → | .06 | .08 | .10 | .12 | .14 | .16 | .18 | .20 | .22 | .24 |
|---|---|---|---|---|---|---|---|---|---|---|
| R1 | -2.5 | -6.4 | -24.3 | -21.1 | -23.1 | -31.2 | -11.5 | -7.9 | -6.1 | -2.1 |
| R2 | -39.1 | -27.1 | -26.1 | -15.6 | -20.6 | -19.1 | -7.0 | -19.3 | -11.9 | -7.6 |
| R3 | -24.9 | -61.0 | -32.4 | -26.8 | -9.6 | -10.1 | -16.9 | -10.8 | -7.9 | -3.8 |
| R4 | -18.6 | -17.9 | -17.3 | -42.2 | -14.9 | -5.7 | -9.2 | -10.2 | -9.3 | -5.1 |
| R5 | -43.3 | -27.3 | -16.7 | -13.2 | -44.3 | -20.1 | -10.2 | -14.2 | -5.3 | 0 |

Table 6.4: Relative magnitude error (in decibel) for $S3$. *We compute relative error magnitude for the spectrograms computed in Figure 6.7. The rows highlight error for different receiver positions.*




(a) S1

(b) S2

(c) S3

Figure 6.8: Relative magnitude error (in decibel). *We compute relative error magnitude for the spectrograms in Figure 6.3, 6.5, and 6.7 for sources* $S1$, $S2$, *and* $S3$, *respectively (for various receiver positions).*

## 6.3 Conclusion, Limitations, and Future Work

We have shown that geometric acoustics methods can be used to achieve accuracy comparable to numerical methods in simple scenes for many source-receiver positions. However, there are also many source-receiver positions with large disagreement. We are investigating into the reasons for these disagreements. We have performed a preliminary validation study on a limited number of source and receiver positions. Also, the model used in our experiments is relatively simple. In future, we would like to do a more systematic validation experiment. We would like to compare numerical and geometric methods against simple basic geometries like, open space, enclosing rectangular box, and open space with a rectangular box. We would also like to perform similar experiments on more complex models and without limiting geometric methods to low orders of reflections and diffraction. Further, a visualization to easily compare the numerical and geometric acoustics results will help identify the regions of disagreement between the results of numerical and geometric simulation.

# Chapter 7

# Conclusions and Future Work

In this thesis, we presented algorithms to develop fast and scalable sound rendering algorithms. The two key components of sound rendering addressed in the thesis are: (a) geometric sound propagation and (b) artifact-free audio processing.

We presented three different object-space visibility techniques to accelerate geometric sound propagation. AD-Frustum can provide interactive performance for sound rendering applications like video games, and can scale to large, complex, and dynamic scenes, as well as multi-core processors. Our FastV and from-region object-space conservative visibility algorithms accelerate the image source method to accurately model specular reflections and finite-edge diffraction for applications like architectural acoustics. Our conservative visibility algorithms accelerate the image source method by applying from-point visibility for accelerating specular reflection computation and from-region visibility for accelerating finite-edge diffraction computation based on the Biot-Tolstoy-Medwin (BTM) model. These algorithms provide much faster performance than prior image source methods and can handle large complex scenes and scale with parallel hardware.

We also present efficient audio processing algorithms that generate artifact-free audio output and can efficiently handle dynamic scenes. We present approaches to estimate late reverberation and handle scenarios with simultaneously moving sources and receivers. To generate artifact-free audio output, we perform delay interpolation of image

sources, and windowing for IR interpolation. For efficient audio processing, we apply block convolution to compute the output audio.

In the following sections, we summarize our algorithms, discuss some of their limitations and suggest a few directions for future work.

## 7.1 AD-Frustum: Adaptive Frustum Tracing

We present a novel volumetric tracing approach that can generate propagation paths for early specular reflections by adapting to the scene primitives. We adaptively generate 4-sided frusta to compute propagation paths. Each sub-frustum represents a volume corresponding to a bundle of rays. We use ray-coherence techniques to accelerate intersection computations with the corner rays. The algorithm uses an area subdivision method to compute an approximation of the visible surface for each frustum. Further, our approach can also handle general, complex, dynamic scenes with dynamic sources, dynamic receiver, and dynamic geometry. We use bounding volume hierarchies (BVHs) to accelerate the intersection computations with AD-Frusta in complex, dynamic scenes. We present techniques to bound the maximum subdivision within each AD-Frustum based on scene complexity and thereby control the overall accuracy of propagation by computing all the important contributions. We have applied our algorithm for interactive sound propagation in complex and dynamic scenes corresponding to architectural models, outdoor scenes, and game environments. In practice, our algorithm can accurately compute early sound propagation paths with up to 4-5 reflections at 4-20 frames per second on scenes with hundreds of thousands of polygons on a multi-core PC.

### 7.1.1 Limitations

Our AD-Frustum based sound propagation has many limitations. Our formulation cannot directly handle diffuse (Lambertian) or "glossy" reflections. Moreover, AD-Frustum

is limited to point sources (though we can potentially simulate area and volumetric sources if they can be approximated with point sources). Regarding AD-Frustum visibility computations, many of the underlying computations such as maximum-subdivision depth parameter and intersection test based on Plücker coordinate representation are conservative. As a result, we may generate too many sub-frusta to satisfy a given error bound.

## 7.2 FastV: From-point Visibility Culling on Complex Models

We present a novel algorithm (FastV) for conservative, from-point visibility computation. Our approach is applicable to all triangulated models and does not assume any large objects or occluders. The main idea is to trace a high number of 4-sided volumetric frusta and efficiently compute simple connected blockers for each frustum. We use the blockers to compute a far plane and cull away the non-visible primitives. As the frustum size decreases, the algorithm computes a tighter PVS. We have applied the algorithm to complex CAD and scanned models with millions of triangles and simple dynamic scenes. We use spatial hierarchies along with SIMD and multi-core capabilities to accelerate the computations. In practice, we can compute a conservative PVS, which is within a factor of $5-25\%$ of the exact visible set, in a fraction of a second on a 16-core PC. Our algorithm can trace $100-200$K frusta per second on a single 2.93 GHz Xeon Core on complex models with millions of triangles.

We use FastV to accurately compute specular reflection paths from a point sound source to a receiver. We use a two-phase algorithm that first computes image sources for scene primitives in the PVS computed for primary (or secondary) sources. This is followed by finding valid reflection paths to compute actual contributions at the receiver. We have applied our algorithm to complex models with tens of thousands of triangles.

In practice, we observe a performance improvement of up to 20X using a single-core implementation over prior accurate propagation methods that are based on beam tracing [Laine et al., 2009].

### 7.2.1 Limitations

Our FastV visibility algorithm has some limitations. Since we don't perform occluder fusion, the PVS computed by our algorithm can sometimes be overly conservative. If the scene has no big occluders, we may need to trace a large number of frusta. Our intersection tests are fast, but the conservative nature of the blocker computation can result in a large PVS. The model and its hierarchy are stored in main memory, and therefore our approach is limited to in-core models. Our algorithm is easy to parallelize and works quite well, but is still slower than image space visibility approaches that perform coherent ray tracing or use GPU rasterization capabilities. However, image space visibility approaches are not guaranteed to find all specular reflection paths.

## 7.3 Conservative From-Region Visibility Algorithm

Efficient BTM-based diffraction requires the capability to determine which other diffracting edges are visible from a given diffracting edge. This reduces to a *from-region visibility* problem, and we use a conservative from-region visibility algorithm which can compute the set of visible triangles and edges at object-space precision in a conservative manner. We present a fast algorithm to accurately compute the first few orders of diffraction using the BTM model. We use object-space conservative from-region visibility to significantly reduce the number of edge pairs that need to be considered as compared to the state-of-the-art for second order diffraction. We show that our approach is able to reduce the amount of visible primitives considered by the image source method by a factor of 2 to 4 for second-order edge diffraction. This allows us to obtain a speedup

factor of 2 to 4 when simulating second-order edge diffraction using the BTM model on various benchmark scenes.

We also present a fast algorithm for occluder selection that can compute occluders in all directions around a given convex region. Our algorithm can combine connected sets of small primitives into large occluders and can be more effective in terms of culling efficiency. The final set of visible primitives can then be computed using a state-of-the-art occlusion culling technique. We demonstrate that our occluder selection technique is able to quickly generate occluders consisting of 2-6 triangles each on the average in complex scenes in a few seconds per visibility query on a single core.

### 7.3.1 Limitations

Our from-region visibility algorithm has many limitations. It is possible that in the absence of large primitives that can be used as occluders, our from-region visibility algorithm would have to trace a large number of small frusta in order to select occluders, which could adversely affect its performance. For sound propagation, the BTM model is computationally intensive, and to the best of our knowledge, there exist no implementations of third- or higher-order edge diffraction based on it. However, there do exist special cases where it is necessary to model very high orders of edge diffraction; one example is the case of sound diffracting around the highly tessellated surface of a large pillar. In our implementation, we combine diffraction effects modeled using BTM with specular reflections which model partially absorbing surfaces. The accuracy of such a combination remains to be studied.

## 7.4 Efficient Audio Processing

Many interactive applications have dynamic scenes with moving source and static receiver (MS-SR) or static source and moving receiver (SS-MR). We present artifact-free

audio processing for these dynamic scenes. We use delay interpolation combined with fractional delay filters and windowing functions applied for IR interpolation to reduce artifacts in the final audio. There are also applications with moving sound sources as well as a moving receiver (MS-MR). In such scenarios, as a receiver moves, it receives sound emitted from prior positions of a given source. We present an efficient algorithm that can correctly model sound propagation and audio processing for MS-MR scenarios by performing sound propagation and audio processing from multiple source positions. Our formulation only needs to compute a portion of the response from various source positions using sound propagation algorithms and can be efficiently combined with audio processing techniques to generate smooth, spatialized audio. Moreover, we present an efficient audio processing pipeline, based on block convolution, which makes it easy to combine different portions of the response from different source positions. Finally, we show that our algorithm can be easily combined with well-known GA methods for efficient sound rendering in MS-MR scenarios, with a low computational overhead (less than 25%) over sound rendering for static sources and a static receiver (SS-SR) scenarios.

### 7.4.1 Limitations

Our approach has many limitations. We only perform delay and attenuation interpolation for generating artifact-free output audio and directional interpolation or prediction of image-source could provide better results. Further, our windowing-based interpolation of IRs performs only amplitude interpolation and does not take into account the interpolation of delays.

## 7.5 Future Work

We have presented efficient algorithms for geometric sound propagation, object-space visibility, and audio processing. There are many ways the presented work can be ex-

tended. Further, there are other domains where the efficient techniques proposed in the thesis are applicable. Below, we present a few directions for future work based on the presented work.

**Geometric Sound Propagation:** We would like to extend our accelerated image source method to construct paths of the form *source* $\rightarrow \cdots \rightarrow$ *diffraction* $\rightarrow$ *specular* $\rightarrow \cdots \rightarrow$ *diffraction*. Currently, we discard such paths and we discard paths with three or more edge diffractions. It would be a simple task to perform the visibility checks required to compute which such paths are valid. However, we are not aware of any BTM-based method for computing attenuations which can handle specular reflections between two edge diffractions, and therefore it requires further investigation. We would like to accelerate IR computation by culling away diffraction contributions based on their amplitude or perceptual significance. It has been shown that prioritizing contributions with large amplitudes and culling contributions with small amplitudes can lead to a significant performance gain for first-order diffraction [Calamia et al., 2009]. It would be worthwhile to extend this idea to second-order and higher-order diffraction.

**Visibility Algorithms:** There are many avenues for future work for the visibility algorithms developed in the thesis. We would like to implement our algorithm on many-core GPUs to further exploit the high parallel performance of commodity processors. This could provide capability to design more accurate rendering algorithms based on object-precision visibility computations on complex models (e.g. shadow generation). We would also like to evaluate the trade-offs of using more sophisticated blocker computation algorithms. The current implementation of our from-region visibility algorithm uses a simple object-space frustum culling technique for occlusion culling. This can cause it to miss cases of occluder fusion due to disconnected occluders. One possibility is to use conservative rasterization methods, which may be able to fuse such occluders and thereby result in a smaller PVS from a given region. Moreover, the occluder

selection step itself can be implemented on the GPU for additional performance gains.

**Audio Processing:** There are many avenues for future work for audio processing. We would like to apply perceptual optimization techniques for efficient binaural audio processing due to the large number of image sources or convolutions. Further, correctly performing sound propagation for scenes with dynamic geometry remains an interesting avenue for future work. Such scenes commonly occur in interactive virtual environments and video games. Another important question that remains to be addressed is that of the perceptual impact of correctly modeling sound propagation for dynamic scenes.

Finally, there are many applications where the techniques developed in the thesis could be useful. We would like to explore the application of the presented techniques to outdoor sound propagation and electromagnetic wave propagation. There are many avenues for future work in terms of developing efficient and accurate sound propagation techniques. We would like to explore geometry simplification techniques to automatically simplify complex graphics models (designed for visual rendering) into simplified models which are suitable for geometric sound propagation. As the complexity of geometric sound propagation is typically a function of the number of primitives in a scene, this could significantly improve the overall performance of geometric sound propagation algorithms. Also, as geometric propagation techniques are limited in their accuracy, we would like to explore hybrid propagation methods which combine geometric sound propagation and numerical sound propagation techniques to perform accurate sound propagation efficiently.

# Bibliography


[Akenine-Möller and Aila, 2005] Akenine-Möller, T. and Aila, T. (2005). Conservative and tiled rasterization using a modified triangle set-up. *journal of graphics tools*, 10(3):1–8. 31, 36, 84, 92, 107

[Alarcão et al., 2010] Alarcão, D., Santos, D., and Bento Coelho, J. L. (2010). Virtusound a real time auralization system. In *Proceedings of the 20th International Congress on Acoustics*. 43, 47

[Algazi et al., 2001] 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*. 9, 12, 49, 131

[Allen and Berkley, 1979] 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. 10, 11, 39, 47, 54, 92, 94, 125

[Anderson and Casey, 1997] Anderson, D. and Casey, M. (1997). The sound dimension. *Spectrum, IEEE*, 34(3):46–50. 1

[ANSI/ASA S12.60-2002, 2002] ANSI/ASA S12.60-2002 (2002). *American National Standard Acoustical Performance Criteria, Design Requirements, and Guidelines for Schools, Part 1: Permanent Schools*. American National Standards Institute. 7

[Antani et al., 2011a] Antani, L., Chandak, A., Savioja, L., and Manocha, D. (2011a). Interactive Sound Propagation using Compact Acoustic Transfer Operators. *ACM Transactions on Graphics (to appear)*. 43, 47

[Antani et al., 2011b] Antani, L., Chandak, A., Taylor, M., and Manocha, D. (2011b). Direct-to-indirect acoustic radiance transfer. *IEEE Transactions on Visualization and Computer Graphics*, 99(PrePrints). 43, 44, 45, 47, 127, 144, 146

[Antani et al., 2011c] Antani, L., Chandak, A., Taylor, M., and Manocha, D. (2011c). Efficient Finite-Edge Diffraction using Conservative From-Region Visibility. *Applied Acoustics (to appear)*. 36, 113, 125, 126

[Antonacci et al., 2004] 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. 39

[Arvo and Kirk, 1989] Arvo, J. and Kirk, D. (1989). *A survey of ray tracing acceleration techniques*, pages 201–262. Academic Press Ltd., London, UK, UK. 13, 35

[Avendano, 2004] Avendano, C. (2004). *Audio Signal Processing for Next-Generation Multimedia Communication Systems*, chapter 13. Kluwer Academic Publishers, Norwell, MA, USA. 4

[AXON, 2011] AXON (2011). Dolby axon. `https://axon.dolby.com`. (accessed June, 2011). 3

[Begault, 1994] Begault, D. R. (1994). *3-D sound for virtual reality and multimedia*. Academic Press Professional, Inc., San Diego, CA, USA. 49, 124

[Bentley, 1975] Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517. 35

[Bertram et al., 2005] Bertram, M., Deines, E., Mohring, J., Jegorovs, J., and Hagen, H. (2005). Phonon Tracing for Auralization and Visualization of Sound. In *Visualization, 2005. VIS 05. IEEE*, pages 151–158. 44, 47

[Bittner et al., 1998] Bittner, J., Havran, V., and Slavik, P. (1998). Hierarchical visibility culling with occlusion trees. In *CGI '98: Proceedings of the Computer Graphics International 1998*, page 207, Washington, DC, USA. IEEE Computer Society. 31, 32, 36, 84

[Bittner et al., 2009] Bittner, J., Mattausch, O., Wonka, P., Havran, V., and Wimmer, M. (2009). Adaptive global visibility sampling. *ACM Trans. Graph.*, 28(3):1–10. 35, 36, 124

[Bittner and Wonka, 2005] Bittner, J. and Wonka, P. (2005). Fast exact from-region visibility in urban scenes. *Eurographics Symposium on Rendering*, pages 223–230. 32

[Bork, 2002] Bork, I. (2002). Simulation and Measurement of Auditorium Acoustics - The Round Robins on Room Acoustical Simulation. In *Institute of Acoustics*, volume 24. 52, 149

[Botteldooren, 1995] Botteldooren, D. (1995). Finite-difference time-domain simulation of low-frequency room acoustic problems. *Acoustical Society of America Journal*, 98:3302–3308. 10, 38

[Brewster, 1997] Brewster, S. A. (1997). Using non-speech sound to overcome information overload. *Displays*, 17(3-4):179–189. 7

[Calamia, 2009] Calamia, P. (2009). *Advances in Edge-Diffraction Modeling for Virtual-Acoustic Simulations*. PhD thesis, Princeton University, Princeton, New Jersey. 53

[Calamia et al., 2009] Calamia, P., Markham, B., and Svensson, U. P. (2009). Diffraction culling for virtual-acoustic simulations. *The Journal of the Acoustical Society of America*, 125(4):2586–2586. 118, 120, 167

[Calamia and Svensson, 2005] Calamia, P. 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*, pages 187–190. 122

[Calamia and Svensson, 2007] 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. 42, 47, 122

[Catmull, 1974] Catmull, E. E. (1974). *A subdivision algorithm for computer display of curved surfaces*. PhD thesis, The University of Utah. AAI7504786. 15, 34, 36

[CATT, 2002] CATT (2002). *CATT-Acoustic User Manual*. CATT, Sweden, v8.0 edition. http://www.catt.se/. 45, 149

[Chandak et al., 2011] Chandak, A., Antani, L., and Manocha, D. (2011). Efficient Auralization for Moving Sources and Receiver. Technical Report TR11-008, University of North Carolina at Chapel Hill. 127, 137

[Chandak et al., 2009] Chandak, A., Antani, L., Taylor, M., and Manocha, D. (2009). FastV: From-point Visibility Culling on Complex Models. *Computer Graphics Forum (Proc. of EGSR)*, 28(3):1237–1247. 36, 98, 111, 125

[Chandak et al., 2010] Chandak, A., Antani, L., Taylor, M., and Manocha, D. (2010). Fast and Accurate Geometric Sound Propagation Using Visibility Computations. In *International Symposium on Room Acoustics, ISRA*, Melbourne, Australia. 108

[Chandak et al., 2008] 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. 56, 79, 81, 124

[Chhugani et al., 2005] Chhugani, J., Purnomo, B., Krishnan, S., Cohen, J., Venkatasubramanian, S., Johnson, D. S., and Kumar, S. (2005). vLOD: High-fidelity walkthrough of large virtual environments. *IEEE Transactions on Visualization and Computer Graphics*, 11(1):35–47. 33, 36, 98, 101, 104, 107

[Christensen, 2009] Christensen, C. L. (2009). *ODEON Room Acoustics Program User Manual*. ODEON A/S, Denmark, 10.1 edition. http://www.odeon.dk/. 6, 7, 16, 45, 149

[COD, 2011] COD (2011). Call Of Duty: Black Ops. `http://www.callofduty.com/`. (accessed June, 2011). 3

[Cohen-Or et al., 2003] Cohen-Or, D., Chrysanthou, Y., Silva, C., and Durand, F. (2003). A survey of visibility for walkthrough applications. *Visualization and Computer Graphics, IEEE Transactions on*, 9(3):412–431. 27, 36, 104

[Cook, 2002] Cook, P. R. (2002). *Real Sound Synthesis for Interactive Applications*. A K Peters. 49

[Cooperstock, 2011] Cooperstock, J. R. (2011). Multimodal Telepresence Systems. *Signal Processing Magazine, IEEE*, 28(1):77–86. 5

[Coorg and Teller, 1997] Coorg, S. and Teller, S. (1997). Real-time occlusion culling for models with large occluders. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 83–ff., New York, NY, USA. ACM. 31, 36, 84, 98, 99

[CSS, 2011] CSS (2011). Counter Strike: Source. `http://www.counterstrikesource.com/`. (accessed June, 2011). 3

[Dalenbäck, 1996] Dalenbäck, B.-I. L. (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. 46

[Dellepiane et al., 2008] Dellepiane, M., Pietroni, N., Tsingos, N., Asselot, M., and Scopigno, R. (2008). Reconstructing head models from photographs for individualized 3D-audio processing. *Computer Graphics Forum*, 27(7):1719–1727. 12, 50

[Drumm and Lam, 2000] Drumm, I. A. and Lam, Y. W. (2000). The adaptive beam-tracing algorithm. *Journal of the Acoustical Society of America*, 107(3):1405–1412. 68

[Durand et al., 1996] Durand, F., Drettakis, G., and Puech, C. (1996). The 3d visibility complex: a new approach to the problems of accurate visibility. In *Proceedings of the eurographics workshop on Rendering techniques '96*, pages 245–256, London, UK. Springer-Verlag. 14, 29, 36, 125

[Durand et al., 1997] Durand, F., Drettakis, G., and Puech, C. (1997). The visibility skeleton: a powerful and efficient multi-purpose global visibility tool. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 89–100, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co. 29, 36, 125

[Durand et al., 2000] Durand, F., Drettakis, G., Thollot, J., and Puech, C. (2000). Conservative visibility preprocessing using extended projections. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 239–248, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co. 27, 33, 36, 98, 101, 104

[EIDOS, 2011] EIDOS (2011). Thief 4. `http://www.eidos.com/games/thief-4`. (accessed June, 2011). 2, 3

[Eyring, 1930] Eyring, C. F. (1930). Reverberation time in "dead" rooms. *The Journal of the Acoustical Society of America*, 1(2A):217–241. 132

[Farina, 1995] Farina, A. (1995). RAMSETE - a new Pyramid Tracer for medium and large scale acoustic problems. In *Proceedings of EURO-NOISE*. 41, 46, 47

[Foale and Vamplew, 2007] 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, Melbourne, Australia, Australia. RMIT University. 44, 47

[Foley et al., 1990] Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F. (1990). *Computer graphics: principles and practice (2nd ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 34

[Foley et al., 1993] Foley, J. D., van Dam, A., Feiner, S. K., Hughes, J. F., and Phillips, R. L. (1993). *Introduction to Computer Graphics*. Addison-Wesley Professional. 14

[Franck, 2008] Franck, A. (2008). Efficient Algorithms and Structures for Fractional Delay Filtering Based on Lagrange Interpolation. *J. Audio Eng. Soc.*, 56(12):1036–1056. 136

[Fuchs et al., 1980] Fuchs, H., Kedem, Z. M., and Naylor, B. F. (1980). On visible surface generation by a priori tree structures. In *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '80, pages 124–133, New York, NY, USA. ACM. 35

[Funkhouser et al., 1998] 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 *Proc. of ACM SIGGRAPH*, pages 21–32. 16, 41, 47, 65, 83, 92, 94, 125

[Funkhouser et al., 2004] Funkhouser, T., Tsingos, N., Carlbom, I., Elko, G., Sondhi, M., West, J. E., Pingali, G., Min, P., and Ngan, A. (2004). A beam tracing method for interactive architectural acoustics. *The Journal of the Acoustical Society of America*, 115(2):739–756. 6, 10, 45, 46, 124

[Funkhouser et al., 2003] Funkhouser, T., Tsingos, N., and Jot, J.-M. (2003). Survey of Methods for Modeling Sound Propagation in Interactive Virtual Environment Systems. *Presence and Teleoperation*. 4, 10, 15, 16, 37, 47, 52, 124

[Gardner, 1998] Gardner, W. G. (1998). *Audio Signal Processing for Next-Generation Multimedia Communication Systems*, chapter 3. Kluwer Academic Publishers, Norwell, MA, USA. 50

[Gardner and Martin, 1995] Gardner, W. G. and Martin, K. D. (1995). HRTF measurements of a KEMAR. *Journal of the Acoustical Society of America*, 97(6):3907–3908. 49

[Genetti et al., 1998] Genetti, J., Gordon, D., and Williams, G. (1998). Adaptive supersampling in object space using pyramidal rays. *Computer Graphics Forum*, 17:29–54. 41, 47, 68

[Gerardi et al., 2008] Gerardi, M., Rothbaum, B. O., Ressler, K., Heekin, M., and Rizzo, A. (2008). Virtual reality exposure therapy using a virtual Iraq: Case report. *Journal of Traumatic Stress*, 21(2):209–213. 4

[Gigus et al., 1991] Gigus, Z., Canny, J., and Seidel, R. (1991). Efficiently computing and representing aspect graphs of polyhedral objects. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(6):542–551. 29, 36, 125

[Glassner, 1989] Glassner, A. S. (1989). *An Introduction to Ray tracing*. Morgan Kaufmann, 1st edition edition. 15, 34

[Govindaraju et al., 2003] Govindaraju, N. K., Redon, S., Lin, M. C., and Manocha, D. (2003). Cullide: interactive collision detection between complex models in large environments using graphics hardware. In *Graphics Hardware 2003*, pages 25–32. 92

[Haraldsen, 2010] Haraldsen, S. (2010). Sound Propagation: Bringing Audio to a Higher Level. In *Game Developer Magazine*, pages 32–35. `http://www.gdmag.com/`. 2

[Harding et al., 2000] Harding, C., Kakadiaris, I. A., and Loftin, R. B. (2000). A Multimodal User Interface for Geoscientific Data Investigation. In *ICMI '00: Proceedings of the Third International Conference on Advances in Multimodal Interfaces*, pages 615–623, London, UK. Springer-Verlag. 7

[Hasenfratz et al., 2003] Hasenfratz, J.-M., Lapierre, M., Holzschuch, N., and Sillion, F. (2003). A survey of real-time soft shadows algorithms. *Computer Graphics Forum*, 22(4):753–774. 13

[Haumont et al., 2005] Haumont, D., Makinen, O., and Nirenstein, S. (2005). A Low Dimensional Framework for Exact Polygon-to-Polygon Occlusion Queries. In *Eurographics Symposium on Rendering Techniques*, pages 211–222. 29, 36

[Heckbert and Hanrahan, 1984] Heckbert, P. S. and Hanrahan, P. (1984). Beam tracing polygonal objects. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 119–127, New York, NY, USA. ACM. 14, 28, 36

[Hudson et al., 1997] Hudson, T., Manocha, D., Cohen, J., Lin, M., Hoff, K., and Zhang, H. (1997). Accelerated occlusion culling using shadow frusta. In *SCG '97: Proceedings of the thirteenth annual symposium on Computational geometry*, pages 1–10, New York, NY, USA. ACM. 31, 32, 36, 84, 98

[Hughes et al., 2006] Hughes, D. E., Thropp, J., Holmquist, J., and Moshell, J. M. (2006). Spatial Perception and Expectation: Factors in Acoustical Awareness for Mout Training. In *Proceedings of the 24th US Army Science Conference: Transformational Science and Technology for the Current and Future Force*, pages 339–343, Orlando, Florida, USA. 5, 6

[ISO 3382-1:2009, 2009] ISO 3382-1:2009 (2009). *Acoustics – Measurement of room acoustic parameters – Part 1: Performance spaces*. ISO, Geneva, Switzerland. 149

[ISO 3382-2:2008, 2008] ISO 3382-2:2008 (2008). *Acoustics – Measurement of room acoustic parameters – Part 2: Reverberation time in ordinary rooms*. ISO, Geneva, Switzerland. 149

[James et al., 2006] James, D. L., Barbič, J., and Pai, D. K. (2006). Precomputed acoustic transfer: output-sensitive, accurate sound generation for geometrically complex vibration sources. *ACM Trans. Graph.*, 25(3):987–995. 8

[Joslin and Magnenat-Thalmann, 2003] Joslin, C. and Magnenat-Thalmann, N. (2003). Significant facet retrieval for real-time 3d sound rendering in complex virtual environments. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '03, pages 15–21, New York, NY, USA. ACM. 65

[Joslin and Magnetat-Thalmann, 2003] Joslin, C. and Magnetat-Thalmann, N. (2003). Significant facet retrieval for real-time 3D sound rendering. In *Proceedings of the ACM VRST*. 46

[Kajastila and Lokki, 2010] Kajastila, R. and Lokki, T. (2010). Eyes-free Methods for Accessing Large Auditory Menus. In *16th International Conference on Auditory Display (ICAD 2010)*, pages 223–230. 7, 8

[Kapralos, 2006] Kapralos, B. (2006). *The Sonel Mapping Acoustical Modeling Method*. PhD thesis, York University, Toronto, Ontario. 10, 39, 44, 47

[Kastbauer, 2010] Kastbauer, D. (January, 2010). The Next Big Steps In Game Sound Design. In *GAMASUTRA*. `http://www.gamasutra.com/`. 4

[Kay and Kajiya, 1986] Kay, T. L. and Kajiya, J. T. (1986). Ray tracing complex scenes. *SIGGRAPH Comput. Graph.*, 20:269–278. 42

[Kleiner et al., 1993] Kleiner, M., Dalenbäck, B.-I., and Svensson, P. (1993). Auralization - an overview. *JAES*, 41:861–875. 2, 10, 37, 38, 47

[Koltun and Cohen-Or, 2000] Koltun, V. and Cohen-Or, D. (2000). Selecting effective occluders for visibility culling. In *Eurographics Short Presentations*. 98, 104, 106, 107

[Kouyoumjian and Pathak, 1974] Kouyoumjian, R. G. and Pathak, P. H. (1974). A uniform geometrical theory of diffraction for an edge in a perfectly conducting surface. *Proceedings of the IEEE*, 62(11):1448–1461. 11, 39

[Krokstad et al., 1968] 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. 10, 39, 43, 47, 124

[Kurze, 1974] Kurze, U. J. (1974). Noise reduction by barriers. *The Journal of the Acoustical Society of America*, 55(3):504–518. 46

[Kuttruff, 1991] Kuttruff, H. (1991). *Room Acoustics*. Elsevier Science Publishing Ltd. 10, 62

[Laakso et al., 1996] 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. 136

[Laine, 2006] Laine, S. (2006). An incremental shaft subdivision algorithm for computing shadows and visibility. Master's thesis, Helsinki University of Technology. 78, 92

[Laine et al., 2009] Laine, S., Siltanen, S., Lokki, T., and Savioja, L. (2009). Accelerated beam tracing algorithm. *Applied Acoustics*, 70(1):172 – 181. 21, 41, 47, 69, 83, 92, 93, 94, 95, 125, 164

[Lauterbach et al., 2007a] 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*. 69

[Lauterbach et al., 2007b] Lauterbach, C., Chandak, A., and Manocha, D. (2007b). Interactive sound rendering in complex and dynamic scenes using frustum tracing. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1672–1679. 19, 56, 62, 64, 69, 75, 124

[Lauterbach et al., 2009] Lauterbach, C., Garland, M., Sengupta, S., Luebke, D., and Manocha, D. (2009). Fast bvh construction on gpus. *Comput. Graph. Forum*, 28(2):375–384. 35

[Lauterbach et al., 2006] Lauterbach, C., Yoon, S.-E., Tuft, D., and Manocha, D. (2006). RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs. *IEEE Symposium on Interactive Ray Tracing*. 58

[Lecuyer et al., 2003] Lecuyer, A., Mobuchon, P., Megard, C., Perret, J., Andriot, C., and Colinot, J.-P. (2003). HOMERE: a multimodal system for visually impaired people to explore virtual environments. In *Virtual Reality, 2003. Proceedings. IEEE*, pages 251–258. 5

[Lehmann and Johansson, 2010] Lehmann, E. and Johansson, A. (2010). Diffuse Reverberation Model for Efficient Image-Source Simulation of Room Impulse Responses. *Audio, Speech, and Language Processing, IEEE Transactions on*, 18(6):1429 –1439. 50, 133

[Lehnert, 1993] Lehnert, H. (1993). Systematic errors of the ray-tracing algorithm. *Applied Acoustics*, 38:207–221. 124

[Lloyd et al., 2006] Lloyd, B., Tuft, D., Yoon, S., and Manocha, D. (2006). Warping and partitioning for low error shadow maps. In *Proceedings of the Eurographics Symposium on Rendering 2006*, pages 215–226. Eurographics Association. 14

[Lloyd et al., 2004] Lloyd, B., Wend, J., Govindaraju, N. K., and Manocha, D. (2004). Cc shadow volumes. In *Eurographics Symposium on Rendering/Eurographics Workshop on Rendering Techniques*, pages 197–206. 92

[Loftin, 2003] Loftin, R. B. (2003). Multisensory Perception: Beyond the Visual in Visualization. *Computing in Science and Engineering*, 5(4):56–58. 7

[Luebke and Georges, 1995] Luebke, D. and Georges, C. (1995). Portals and mirrors: simple, fast evaluation of potentially visible sets. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 105–ff., New York, NY, USA. ACM. 31, 32, 36, 84

[Mann, 2008] Mann, B. L. (2008). The evolution of multimedia sound. *Computers & Education*, 50(4):1157–1173. 4

[Manocha et al., 2009] Manocha, D., Calamia, P., Lin, M. C., Manocha, D., Savioja, L., and Tsingos, N. (2009). Interactive sound rendering. In *SIGGRAPH '09: ACM SIGGRAPH 2009 Courses*, pages 1–338, New York, NY, USA. ACM. 2

[McGookin et al., 2009] McGookin, D., Brewster, S., and Priego, P. (2009). Audio Bubbles: Employing Non-speech Audio to Support Tourist Wayfinding. In *Haptic and Audio Interaction Design*, volume 5763 of *Lecture Notes in Computer Science*, pages 41–50. Springer Berlin / Heidelberg. 4

[Medwin et al., 1982] Medwin, H., Childs, E., and Jebsen, G. M. (1982). Impulse studies of double diffraction: A discrete huygens interpretation. *The Journal of the Acoustical Society of America*, 72(3):1005–1013. 11, 112

[Mehra et al., 2012] 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(2):83 – 94. 38

[Melzer et al., 2010] Melzer, A., Kindsmuller, M., and Herczeg, M. (2010). Audioworld: A spatial audio tool for acoustic and cognitive learning. In *Haptic and Audio Interaction Design*, volume 6306 of *Lecture Notes in Computer Science*, pages 46–54. Springer Berlin / Heidelberg. 4

[Moeck et al., 2007] 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. 52

[Navazo et al., 2003] Navazo, I., Rossignac, J., Jou, J., and Shariff, R. (2003). Shieldtester: Cell-to-cell visibility test for surface occluders. *Computer Graphics Forum*, 22:291–302. 78, 92

[Nguyen, 2007] Nguyen, H. (2007). *Gpu gems 3*. Addison-Wesley Professional. 92

[Nirenstein, 2003] Nirenstein, S. (2003). *Fast and Accurate Visibility Preprocessing*. PhD thesis, University of Cape Town, South Africa. 14, 28

[Nirenstein et al., 2002] Nirenstein, S., Blake, E., and Gain, J. (2002). Exact from-region visibility culling. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 191–202. Eurographics Association. 14, 28, 29, 36

[Nosal et al., 2004] Nosal, E.-M., Hodgson, M., and Ashdown, I. (2004). Improved algorithms and methods for room sound-field prediction by acoustical radiosity in arbitrary polyhedral rooms. *The Journal of the Acoustical Society of America*, 116(2):970–980. 10, 39, 43, 47

[O'Brien et al., 2002] 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. 49

[Overbeck et al., 2007] Overbeck, R., Ramamoorthi, R., and Mark, W. R. (2007). A Real-time Beam Tracer with Application to Exact Soft Shadows. In *Eurographics Symposium on Rendering*, pages 85–98. 14, 28, 36, 69, 83, 84, 86, 88, 89, 125

[Pellerin, 2011] Pellerin, C. (2011). Immersive Technology Fuels Infantry Simulators. In *American Forces Press Service*. 5

[Pielot et al., 2007] Pielot, M., Henze, N., Heuten, W., and Boll, S. (2007). Tangible User Interface for the Exploration of Auditory City Maps. In *Haptic and Audio Interaction Design*, volume 4813 of *Lecture Notes in Computer Science*, pages 86–97. Springer Berlin / Heidelberg. 4

[Pulkki et al., 2002] Pulkki, V., Lokki, T., and Savioja, L. (2002). Implementation and Visualization of Edge Diffraction with Image-source Method. In *Audio Engineering Society Convention 112*. 11, 39

[Raghuvanshi and Lin, 2006] Raghuvanshi, N. and Lin, M. C. (2006). Interactive sound synthesis for large scale environments. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, I3D '06, pages 101–108, New York, NY, USA. ACM. 8, 49

[Raghuvanshi et al., 2009] Raghuvanshi, N., Narain, R., and Lin, M. C. (2009). Efficient and accurate sound propagation using adaptive rectangular decomposition. *IEEE Trans. Vis. Comput. Graph.*, 15(5):789–801. 10, 38, 47, 149, 150

[Raghuvanshi et al., 2010] Raghuvanshi, N., Snyder, J., Mehra, R., Lin, M. C., and Govindaraju, N. K. (2010). Precomputed wave simulation for real-time sound propagation of dynamic sources in complex scenes. *ACM Trans. Graph.*, 29(4). 44, 45, 47

[Rajkumar et al., 1996] Rajkumar, A., Naylor, B. F., Feisullin, F., and Rogers, L. (1996). Predicting RF coverage in large environments using ray-beam tracing and partitioning tree represented geometry. *Wirel. Netw.*, 2(2):143–154. 68

[RAMSETE, 1995] RAMSETE (1995). *RAMSETE User Manual*. GENESIS Software and Acoustic Consulting, Italy, version 1.0 edition. http://www.ramsete.com/. 45, 46

[Reshetov et al., 2005] Reshetov, A., Soupikov, A., and Hurley, J. (2005). Multi-level ray tracing algorithm. *ACM Trans. Graph.*, 24(3):1176–1185. 58, 79

[Ronchi et al., 1996] Ronchi, C., Iacono, R., and Paolucci, P. (1996). The "Cubed Sphere": A New Method for the Solution of Partial Differential Equations in Spherical Geometry. *Journal of Computational Physics*, 124:93–114(22). 61

[Sakamoto et al., 2004] Sakamoto, S., Yokota, T., and Tachibana, H. (2004). Numerical sound field analysis in halls using the finite difference time domain method. In *RADS 2004*, Awaji, Japan. 38

[Savioja et al., 1999] Savioja, L., Huopaniemi, J., Lokki, T., and Väänänen, R. (1999). Creating Interactive Virtual Acoustic Environments. *J. Audio Eng. Soc.*, 47(9):675–705. 12, 51, 128, 134

[Savioja et al., 2002] 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 (ICAD)*, Kyoto, Japan. 134, 136

[Schaufler et al., 2000] Schaufler, G., Dorsey, J., Decoret, X., and Sillion, F. X. (2000). Conservative volumetric visibility with occluder fusion. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 229–238, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co. 104

[Schröder and Lentz, 2006] 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. 45, 46

[Schröder and Pohl, 2009] Schröder, D. and Pohl, A. (2009). Real-time hybrid simulation method including edge diffraction. In *EAA Auralization Symposium*. 39

[Shinya et al., 1987] Shinya, M., Takahashi, T., and Naito, S. (1987). Principles and applications of pencil tracing. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '87, pages 45–54, New York, NY, USA. ACM. 68

[Shoemake, 1998] Shoemake, K. (1998). Plücker coordinate tutorial. *Ray Tracing News*, 11(1). 59, 79

[Sillion and Puech, 1989] Sillion, F. and Puech, C. (1989). A general two-pass method integrating specular and diffuse reflection. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '89, pages 335–344, New York, NY, USA. ACM. 43

[Sillion and Puech, 1994] Sillion, F. and Puech, C. (1994). *Radiosity and global illumination*. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling Series. Morgan Kaufmann Publishers. 43

[Siltanen et al., 2007] 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. 10, 39, 42

[Siltanen et al., 2009] 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). 43, 44, 45, 47, 51, 134

[Siltanen et al., 2008] Siltanen, S., Lokki, T., Savioja, L., and Lynge Christensen, C. (May/June 2008). Geometry Reduction in Room Acoustics Modeling. *Acta Acustica united with Acustica*, 94:410–418(9). 9

[Sloan et al., 2002] Sloan, P.-P., Kautz, J., and Snyder, J. (2002). Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 527–536, New York, NY, USA. ACM. 43

[Smith, 1993] Smith, S. (1993). Auditory Representation of Scientific Data. In *Focus on Scientific Visualization*, pages 337–346, London, UK. Springer-Verlag. 7

[Stavrakis et al., 2008] Stavrakis, E., Tsingos, N., and Calamia, P. (November-December 2008). Topological sound propagation with reverberation graphs. *Acta Acustica united with Acustica*, 94:921–932(12). 44, 47

[Stephenson, 2010] Stephenson, U. M. (November/December 2010). An analytically derived sound particle diffraction model. *Acta Acustica united with Acustica*, 96:1051–1068(18). 44, 47

[Summers et al., 2004] 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. 132

[Svensson, 1999] Svensson, P. (1999). Edge diffraction toolbox for matlab. http://www.iet.ntnu.no/ svensson/Matlab.html. 113, 119, 122, 126

[Svensson and Kristiansen, 2002] Svensson, P. and Kristiansen, U. R. (2002). Computational Modelling and Simulation of Acoutic Spaces. In *Audio Engineering Society Conference: 22nd International Conference: Virtual, Synthetic, and Entertainment Audio*. 37, 52

[Svensson et al., 1999] Svensson, U. P., Fred, R. I., and Vanderkooy, J. (1999). An analytic secondary source model of edge diffraction impulse responses. *The Journal of the Acoustical Society of America*, 106(5):2331–2344. 39, 42, 47, 53, 114, 115, 122

[Tatarchuk, 2009] Tatarchuk, N. (2009). Advances in real-time rendering in 3D graphics and games I. In *SIGGRAPH '09: ACM SIGGRAPH 2009 Courses*, pages 1–1, New York, NY, USA. ACM. 1, 2

[Taylor et al., 2010] Taylor, M., Chandak, A., Mo, Q., Lauterbach, C., Schissler, C., and Manocha, D. (2010). i-sound: Interactive gpu-based sound auralization in dynamic scenes. Technical Report TR10-006, University of North Carolina at Chapel Hill. 124

[Taylor et al., 2009a] Taylor, M., Chandak, A., Ren, Z., Lauterbach, C., and Manocha, D. (2009a). Fast Edge-Diffraction for Sound Propagation in Complex Virtual Environments. In *EAA Auralization Symposium*, Espoo, Finland. 39

[Taylor et al., 2009b] Taylor, M. T., Chandak, A., Antani, L., and Manocha, D. (2009b). 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. 45, 46, 127

[Teller, 1992] Teller, S. J. (1992). Computing the antipenumbra of an area light source. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 139–148, New York, NY, USA. ACM. 33, 36

[Teller and Séquin, 1991] Teller, S. J. and Séquin, C. H. (1991). Visibility preprocessing for interactive walkthroughs. *SIGGRAPH Comput. Graph.*, 25(4):61–70. 33, 36, 101

[Theoharis et al., 2001] Theoharis, T., Papaiannou, G., and Karabassi, E. (2001). The magic of the z-buffer: A survey. *Proc. of 9th International Conference on Computer Graphics, Visualization and Computer Vision, WSCG*. 13, 34

[Torres-Gil et al., 2010] Torres-Gil, M. A., Casanova-Gonzalez, O., and Gonzalez-Mora, J. L. (2010). Applications of virtual reality for visually impaired people. *W. Trans. on Comp.*, 9:184–193. 5

[Tsilfidis et al., 2009] Tsilfidis, A., Papadakos, C., and Mourjopoulos, J. (2009). Hierarchical perceptual mixing. In *Audio Engineering Society Convention 126*. 52

[Tsingos, 2001a] Tsingos, N. (2001a). Artifact-free asynchronous geometry-based audio rendering. In *Proceedings of the Acoustics, Speech, and Signal Processing, 2001. on IEEE International Conference - Volume 05*, pages 3353–3356, Washington, DC, USA. IEEE Computer Society. 12, 51

[Tsingos, 2001b] Tsingos, N. (2001b). A versatile software architecture for virtual audio simulations. In *International Conference on Auditory Display (ICAD)*, Espoo, Finland. 51, 134

[Tsingos, 2005] Tsingos, N. (2005). Scalable perceptual mixing and filtering of audio signals using an augmented spectral representation. In *Proceedings of the International Conference on Digital Audio Effects*. Madrid, Spain. 12, 52

[Tsingos, 2009] Tsingos, N. (2009). Precomputing geometry-based reverberation effects for games. In *Audio Engineering Society Conference: 35th International Conference: Audio for Games*. 44, 45, 47

[Tsingos et al., 2002] Tsingos, N., Carlbom, I., Elko, G., Kubli, R., and Funkhouser, T. (2002). Validating acoustical simulations in the Bell Labs Box. *Computer Graphics and Applications, IEEE*, 22(4):28–37. 6, 52, 53, 149

[Tsingos et al., 2007] Tsingos, N., Dachsbacher, C., Lefebvre, S., and Dellepiane, M. (2007). Instant sound scattering. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)*. `http://www-sop.inria.fr/reves/Basilic/2007/TDLD07`. 8, 9, 42, 46, 65

[Tsingos et al., 2001] Tsingos, N., Funkhouser, T., Ngan, A., and Carlbom, I. (2001). Modeling acoustics in virtual environments using the uniform theory of diffraction. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 545–552, New York, NY, USA. ACM. 39

[Tsingos et al., 2004] Tsingos, N., Gallo, E., and Drettakis, G. (2004). Perceptual audio rendering of complex virtual environments. *ACM Trans. Graph.*, 23(3):249–258. 12, 46, 52, 67, 131, 134

[Tsingos and Gascuel, 1997] Tsingos, N. and Gascuel, J.-D. (1997). A general model for the simulation of room acoustics based on hierachical radiosity. In *ACM SIGGRAPH 97 Visual Proceedings: The art and interdisciplinary programs of SIGGRAPH '97*, SIGGRAPH '97, pages 149–, New York, NY, USA. ACM. 43, 47

[Välimäki, 1995] Välimäki, V. (1995). *Discrete-Time Modeling of Acoustic Tubes Using Fractional Delay Filters*. PhD thesis, Helsinki University of Technology. 136

[van den Doel, 1998] van den Doel, K. (1998). *Sound Synthesis for Virtual Reality and Computer Games*. PhD thesis, University of British Columbia. 49

[van den Doel et al., 2001] van den Doel, K., Kry, P. G., and Pai, D. K. (2001). Foleyautomatic: physically-based sound effects for interactive simulation and animation. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 537–544, New York, NY, USA. ACM. 49

[Vorländer, 1989] 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. 10, 41, 47, 124

[Vorländer, 2008] Vorländer, M. (2008). *Auralization: Fundamentals of Acoustics, Modelling, Simulation, Algorithms and Acoustic Virtual Reality*. RWTHedition (Berlin. Print). Springer. 8, 9

[Wald, 2007] Wald, I. (2007). On fast construction of sah-based bounding volume hierarchies. In *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, pages 33–40, Washington, DC, USA. IEEE Computer Society. 35

[Wald et al., 2007a] Wald, I., Boulos, S., and Shirley, P. (2007a). Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Trans. Graph.*, 26. 35

[Wald et al., 2007b] Wald, I., Mark, W., Gunther, J., Boulos, S., Ize, T., Hunt, W., Parker, S., and Shirley, P. (2007b). State of the Art in Ray Tracing Dynamic Scenes. *Eurographics State of the Art Reports*. 68

[Wand and Straßer, 2004] Wand, M. and Straßer, W. (2004). Multi-Resolution Sound Rendering. In *SPBG'04 Symposium on Point-Based Graphics 2004*, pages 3–11, Zürich, Switzerland. 52, 131

[Warnock, 1969] Warnock, J. E. (1969). *A hidden surface algorithm for computer generated halftone pictures*. PhD thesis. AAI6919002. 59

[Wenzel et al., 2000] 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. 51, 134

[White et al., 2008] White, G. R., Fitzpatrick, G., and McAllister, G. (2008). Toward accessible 3d virtual environments for the blind and visually impaired. In *Proceedings of the 3rd international conference on Digital Interactive Media in Entertainment and Arts*, DIMEA '08, pages 134–141, New York, NY, USA. ACM. 4

[Whitted, 1980] Whitted, T. (1980). An improved illumination model for shaded display. *Commun. ACM*, 23:343–349. 34, 36

[Wilson, 2010] Wilson, J. (2010). Veterans Behavioral Health: Improving Post-combat Care. In *Defense Wide under Defense Issues*. `http://www.defensemedianetwork.com/`. 5, 6

[Wise and Bristow-Johnson, 1999] 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*. 136

[Wonka et al., 2000] Wonka, P., Wimmer, M., and Schmalstieg, D. (2000). Visibility preprocessing with occluder fusion for urban walkthroughs. In *Eurographics Workshop on Rendering 2000*, pages 71–82. 103, 104

[Wonka et al., 2006] Wonka, P., Wimmer, M., Zhou, K., Maierhofer, S., Hesina, G., and Reshetov, A. (2006). Guided visibility sampling. *ACM Trans. Graph.*, 25(3):494–502. 35, 36, 81, 84, 124

[Yin et al., 2009] Yin, X., Wonka, P., and Razdan, A. (2009). Generating 3d building models from architectural drawings: A survey. *IEEE Comput. Graph. Appl.*, 29(1):20–30. 32