iSound: Interactive GPU-based Sound Auralization in Dynamic Scenes

M. Taylor A. Chandak Q. Mo C. Lauterbach C. Schissler D. Manocha
University of North Carolina at Chapel Hill

Abstract

We present an auralization algorithm for interactive virtual environments with dynamic objects, sources, and listener. Our approach uses a modified image source method that computes propagation paths combining direct transmission, specular reflections, and edge diffractions up to a specified order. We use a novel multi-view raycasting algorithm for parallel computation of image sources on GPUs. Rays that intersect near diffracting edges are detected using barycentric coordinates and further propagated. In order to reduce the artifacts in audio rendering of dynamic scenes, we use a high order interpolation scheme that takes into account attenuation, cross-fading, and delay. The resulting system can perform perform auralization at interactive rates on a high-end PC with NVIDIA GTX 280 GPU with 2-3 orders of reflections and 1 order of diffraction. Overall, our approach can generate plausible sound rendering for game-like scenes with tens of thousands of triangles. We observe more than an order of magnitude improvement in computing propagation paths over prior techniques.

1 Introduction

Auditory displays and sound rendering are frequently used to enhance the sense of immersion in virtual environments and multimedia applications. The aural cues combine with the visual cues to improve realism and the user's experience. One of the challenges in interactive virtual environments is to perform auralization and visualization at interactive rates, i.e. 30fps or better. Current graphics hardware and algorithms make it possible to render complex scenes with millions of primitives at interactive rates. On the other hand, current auralization methods cannot generate realistic sound effects at interactive rates even in simple dynamic scenes composed of thousands of primitives.

Given a description of the virtual environment along with the knowledge of sound sources and listener location, the basic auralization pipeline consists of two parts: sound propagation and audio processing. The propagation algorithm computes a spatial acoustic model using impulse responses (IRs) that encode the delays and attenuation of sound traveling from the source to the listener along different propagation paths representing transmissions, reflections, and diffraction. Whenever the source or the objects in the scene move, these propagation paths must be recomputed at interactive rates. The audio processing algorithm generates audio signals by convolving the input audio signals with the IRs. In dynamic scenes, the propagation paths can change significantly, making it challenging to produce artifact-free audio rendering at interactive rates.

There is extensive literature on modeling the propagation of sound, including reflections and diffraction. Most prior work for interactive applications is based on Geometric-Acoustic (GA) techniques such as image-source methods, ray-tracing, path-tracing, beam-tracing, ray-frustum tracing, etc. However, it is widely regarded that current GA methods do not provide enough flexibility and efficiency needed for use in interactive applications [Tsingos 2009]. Therefore, current games precompute and store reverberation filters for a number of locations [Pope et al. 1999]. These filters are typically computed based on occlusion relationships between the sound source and the listener or shooting rays. Other applications tend to use dynamic artificial reverberation filters [Jot 1999] or audio shaders to identify the surrounding geometric primitives and dynamically adjust the time delays. These techniques cannot compute the early acoustic response in dynamic scenes with moving objects and sound sources.

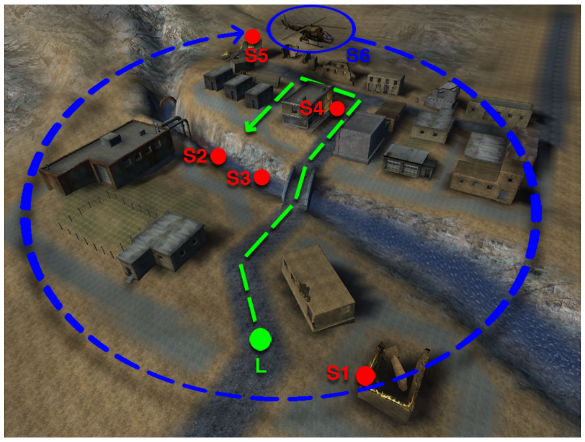


Figure 1: Gamebyro Desert Benchmark: *This is an outdoor scene with 35.5K triangles. It has five fixed sound sources (S1-S5) and a moving source (S6), whose trajectory is shown in blue. The listener's path is shown in green. i-Sound performs 3 orders of reflection and 1 edge diffraction at 33fps for interactive 3D auralization on a PC with an NVIDIA GTX 280.*

Main Results: In this paper, we address the problem of auralization for a single listener at interactive rates, where the goal is to produce audio signals for a given virtual environment. Our work is based on recent developments in GA and interactive ray tracing and exploits the computational capabilities of many-core GPUs for fast sound propagation. We approximate the image-source method using ray visibility to compute propagation paths for specular reflection and edge diffraction. Using a GPU-based parallel visibility algorithm and new method of ray diffraction, our system runs at interactive rates on a single PC. The main components of our work include:

1. **Detection sphere adjustment:** We present a novel algorithm to reduce the cost of visibility and validation operations. By varying the size of the detection sphere, visibility costs can be reduced while maintaining accurate path validation.

2. **Diffraction detection by bary-centric coordinates:** We have developed a low cost method of detecting rays near diffracting edges. Using the bary-centric coordinate of ray intersections, we can create an origin for diffraction propagation.

3. **Multi-viewpoint ray casting:** We describe a new GPU-based ray casting algorithm that performs approximate visible surface computations from multiple viewpoints in parallel.

4. **Smooth audio rendering:** We use a scheme that performs attenuation and delay interpolation per audio sample to reduce the artifacts in final audio output in dynamic scenes.

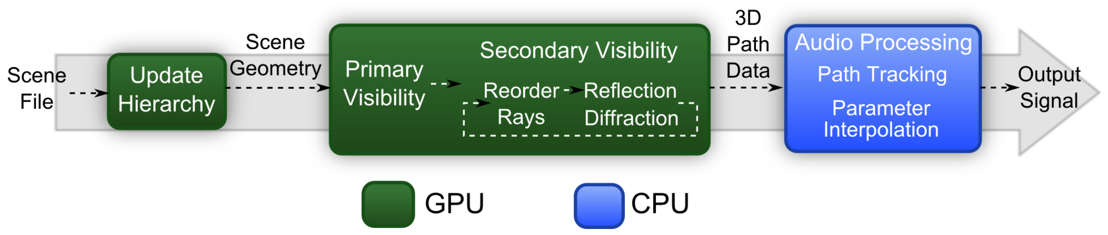


Figure 2: i-Sound Overview: *All scene processing and propagation takes place on the GPU: hierarchy construction, visibility computations, specular and edge diffraction. The sound paths computed using GPU processing are returned to the host for audio processing.*

The overall approach (i-Sound) performs interactive propagation and 3D sound rendering in virtual environments that are used in game-like scenes with tens of thousands of triangles. Moreover, our method is well suited for dynamic scenes with multiple sources. In practice, we observe up to an order of magnitude performance improvement over prior interactive GA methods that use multiple CPU cores. Our algorithm can generate the early acoustic response for dynamic environments, which prior methods based on precomputed reverberation filters cannot render. We highlight its performance on many scenarios and also describe results from our preliminary integration with the Gamebryo game engine.

Organization: The rest of the paper is organized as follows: Section 2 surveys prior work. We present our interactive sound propagation algorithm in Section 3. The details of the multi-viewpoint ray shooting algorithm and acoustic path detection are discussed in Sections 4 and 5. Section 6 describes the audio processing techniques for dynamic scenes. We analyze various sources of error in Section 7 and compare their performance with prior methods.

2 Previous work

The computation of propagation paths and IRs reduces to solving the wave equation. However, current numerical methods used to solve the wave equation have high complexity and are mainly limited to static scenes. In this section, we briefly survey prior work on interactive auralization, that are based on GA and audio rendering.

2.1 Geometric Acoustics

At a broad level, all GA methods compute an acoustic model of the environment with computations based on ray theory and are mainly valid for high-frequency sounds. These include image source methods [Allen and Berkley 1979; Borish 1984]. which compute specular reflection paths by computing virtual or secondary sources. Ray tracing methods [Krokstad et al. 1968; Vorländer 1989] compute propagation paths by generating rays from the source or listener position and follow each of them individually as they propagate through the environment. Accurate geometric propagation algorithms perform object-precision visibility computation based on beam tracing [Funkhouser et al. 1998; Laine et al. 2009], BSP trees [Lentz et al. 2007] or conservative frustum tracing [Chandak et al. 2009]. Recently, fast algorithms based on ray-frustum tracing have been presented to compute propagation paths for specular reflections and edge diffraction [Chandak et al. 2008]. They can be combined with ray tracing for diffuse reflections and use late statistical reverberations [Taylor et al. 2009a] to auralize dynamic scenes at a few frames per second on multi-core processors.

There has been much work combining diffraction with GA methods based on the Uniform Theory of Diffraction (UTD) [Kouyoumjian and Pathak 1974] and Biot-Tolstoy-Medwin (BTM) methods [Biot and Tolstoy 1957]. The BTM method is considered more accurate and can be formulated for use with finite edges [Svensson et al. 1999]. The UTD is used if interactive rates are desired [Tsingos et al. 2001; Taylor et al. 2009a], since even accelerated BTM methods are not fast enough [**?**].

2.2 Acoustics Systems

A few commercial and prototype acoustics systems based on geometric acoustics have been developed, which include sound propagation as well audio processing. This includes ODEON [Christensen 2009], CATT-Acoustic [CAT 2002], RAMSETE [GEN 1995], RAVEN [Schröder and Lentz 2006], beam tracing based systems [Funkhouser et al. 2004; Tsingos et al. 2004], and RESound [Taylor et al. 2009b]. All these systems, except for RESound, are limited to static scenes with moving listeners.

2.3 Audio Processing

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

3 Interactive Sound Propagation

In this section, we present our sound propagation algorithm that computes the propagation paths from the sources to the listener based on direct transmission, specular reflections and edge diffraction (see Fig. 2). We assume that the scene is represented using triangles and we use a bounding volume hierarchy to accelerate the intersection tests for ray casting. For dynamic scenes, the hierarchy is updated using refitting methods. At every frame, we are given the new position of each source and the listener.

Image-Source Methods: Our formulation for computing the propagation paths is based on image source methods, which reflects each source point over all triangles in the scene [Allen and Berkley 1979]. In fact, most of the prior GA algorithms either implicitly or explicitly perform visibility computations to accelerate the performance. We use discrete visibility rays to sample the environment and approximate the image source method. In iSound, the visibility samples are reflected off the scene triangles, creating many secondary viewpoints. For each sample, we check whether the detection sphere is intersected and create a possible contribution path.

The design of i-Sound is governed by two *criteria*: simplicity and efficiency. In order to design faster GA methods, we exploit the computational power of GPUs to perform visibility computations. Specifically, the rasterization pipeline in the GPUs is optimized to perform image-space visibility queries from a given view-

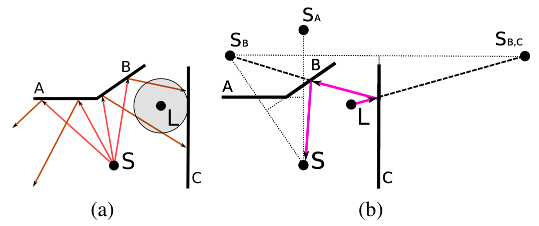


Figure 3: Propagation Path Computation: *(a) From a source point S, ray casting is used to find the zeroth order (or direct) visible set $\{A, B\}$. The rays are reflected to compute the first order visible set that includes $\{C\}$. (b) In order to compute the first order contribution paths between the source and listener L using walls B and C, a ray is first traced to the image-source $S_{B,C}$. From the intersection point on wall C, another ray is traced to S_B. Finally, a ray is traced to the source S. If all these ray segments are unobstructed, a propagation path is computed between S and L.*

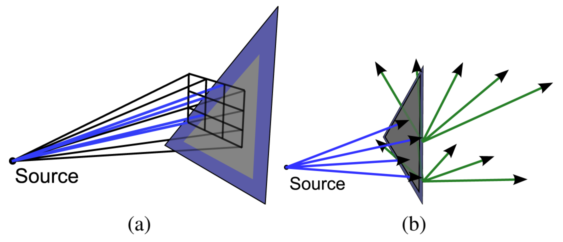


Figure 4: Edge Diffraction*: (a) Rays near the edge are detected for resampling. (b) Diffraction samples are cast through the shadow region, bounded by the adjacent triangle.*

point. However, in terms of image source methods, we need to compute the visible primitives from multiple viewpoints (each representing a virtual sound source) simultaneously. We build on recent developments in GPU-based ray tracing and present a new parallel algorithm to perform visibility computations for image-source methods.

3.1 Specular Reflections

Specular reflections can be computed by finding the visible triangles from a source point and reflecting the source point to create image-points. We use ray casting to compute the visible triangles from the source by tracing sample rays uniformly around the source point. These rays intersect the scene triangles and we compute a zeroth order *visible set* of triangles. For each triangle in this set, reflected rays are emitted from the intersection points and used to compute higher order visible sets. We use a new multi-view ray casting algorithm (see Section 4) to group all the reflection rays together and shoot them in parallel for visibility computations. These reflection and intersection computations are performed repeatedly, until the desired order of reflection is reached (see Figure 3 (a)). The accuracy of this approach is governed by the sampling density used in the primary visibility step (see Figure 5).

3.2 Edge Diffraction

We use an approximate edge diffraction algorithm based on the UTD formulation [Kouyoumjian and Pathak 1974]. The underlying UTD formulation assumes infinite edges and has a lower computation overhead than the BTM method. Similar to other interactive approaches, we only simulate diffraction in *shadow regions* [Tsingos et al. 2001]. The shadow region is the region that is occluded from direct contribution from the source. Only specific edges [Taylor et al. 2009a] are considered to cause diffraction. We select two types of edges as diffracting edges: disconnected edges that are only shared by one triangle and edges shared by triangles with normals that differ by $> \frac{1}{8}\pi$ radians. For each diffracting edge, we store the indices of the triangles that share the edge. This edge data is precomputed before simulation

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

The barycentric coordinates of the hitpoint: $\lambda_1, \lambda_2, \lambda_3$

Triangle with edges e_1, e_2, e_3

A hitpoint can be moved to e_3 as follows:

$s = \lambda_1 + \lambda_2$ $\alpha = \frac{\lambda_2}{s}$ $o = \alpha e_2$

Rays are traced from the diffracting origin according to the UTD: the exitant diffracting rays have the same angle relative to the diffracting edge as the incident ray. However, we only trace diffraction rays in the shadow region, not the full diffraction cone, as described by the UTD. The diffracting edge is shared by the triangle that was hit by the ray and an occluded triangle (possible the backface of the hit triangle). These two triangle faces form a diffracting wedge as describe by the UTD. In order to create diffraction rays for the shadow region, we create a vector based at the origin that was previously found and with the incident ray's direction. This vector is rotated towards the occluded triangle face, sweeping out only the part of the diffraction cone that is in the shadow region. Rays are then sampled on the surface of the cone, discretizing the region. The larger the region, the more rays are traced. These diffraction rays are then traced through the scene and reflected as needed.

Even low order diffraction can branch rapidly [**?**]. Since such branching would require expensive dynamic allocation and kernel managment on the GPU, our system only conducts one order of diffraction.

3.3 Contribution Paths

Most ray casting GA methods use discrete samples both to conduct visibility tests and to find propagation paths from a source to a listener modeled as a sphere [Vorländer 1989]. While these paths can be tested to be valid contribution path, a dense sampling must be used to avoid missing paths. At the other extreme, it is possible to build a visibility tree using the data from the visibility samples. The visibility tree store all possible reflection combinations for a single source position. Using the listener position, the tree can be queried for the valid paths valid paths [Chandak et al. 2009]. However, testing against the visibility tree requires costly ray traversal queries.

We use the sphere detector model [**?**] to create a tree of only the most likely candidate triangles against which we test paths. Visibility rays are tested against a very large detector region surrounding the listener (see Figure 3 (a)). If a ray hits the region, the primitives encountered on the ray's path are added to the visibility tree. The tree is then used to accelerate image-source path tests between the source and the listener (Figure 3 (b)). This method allows sparse visibility sampling, but finds a high number of propagation paths.

Once the visible sets have been computed, contribution paths can be tested. For each triangle in the visible set, a reflection image-point is created. Using these image-points, standard ray based path validation [Allen and Berkley 1979] can be used to compute valid contribution paths.

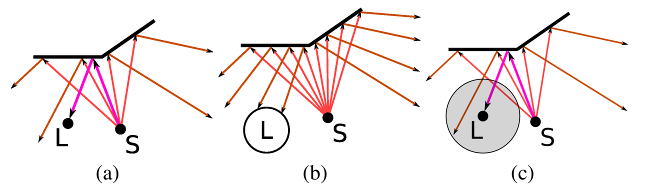


Figure 5: Sample-based visibility: *(a) Sparse visibility samples from source S create a visibility tree which paths are validated against, but this validation step can be costly. (b) Directly testing against a listener sphere L requires dense sampling to avoid missing any paths. (c) Our method combines these ideas by using a large listener, resulting in fewer missed paths, and a smaller visibility tree.*

4 Multi-viewpoint ray casting

The modified image source algorithm presented in Section 3 uses fast ray visibility tests to compute the specular image sources and diffracting edges. In practice, visibility tests are the most expensive component of our propagation algorithm. We use ray casting to perform point visibility queries and evaluate samples from many viewpoints in parallel by using a GPU-based ray tracer.

The underlying formulation of the modified image source method makes it possible to compute the visible primitives from multiple viewpoints simultaneously. For example, if there are multiple sound image sources, we may perform visibility computations from each of them in parallel. After computing the zeroth order visible set of triangles, we obtain multiple secondary image sources and need to compute the visible primitives from each source for higher order reflections or diffraction. In order to achieve interactive performance, we exploit parallelism to process the multiple viewpoints simultaneously.

Ray casting or tracing algorithms can be used to perform point-visibility queries at a fixed resolution. These queries may originate from different viewpoints or sources. Ray intersection tests can also be processed in parallel with no dependencies between separate queries. Previous approaches have investigated efficient methods to implement ray tracing on massively parallel architectures such as a GPU that has a high number of cores as well as wide vector units on each core. Current methods for GPU-based ray tracing mainly differ in the choice of acceleration structure such as kd-trees or BVHs and the parallelization of ray traversal step on each GPU core. For example, rays can be traced as a packet similar to CPU SIMD ray tracing [Popov et al. 2007] and some recent approaches can evaluate them independently [Aila and Laine 2009], as more memory is available for local computation on current GPUs. The main factor that governs the performance of GPU ray tracers is ray coherence, since it affects memory bandwidth and SIMD utilization.

Our multi-view visibility computation algorithm uses ray casting. Each visibility query returns the set of triangles visible from a given point. Specifically, we assume that the ray samples can be described by a viewing frustum, defined by the source and some field of view. We generate ray samples inside the frustum defining the visibility volume to compute a visible set from that viewpoint. This can be achieved using ray casting from the source point or other image sources and record the triangles that intersect the rays. The main difference between our approach and prior GPU ray tracing methods occurs in terms of 1) how the intersection points are processed, 2) generation of secondary visibility samples, and 3) organizing the rays to evaluate those samples in parallel.

4.1 Computing visible sets

After ray casting for each visibility query, we need to compute the set of visible triangles to generate secondary or higher order visibility samples. Given the results of ray casting as a list of triangle IDs,

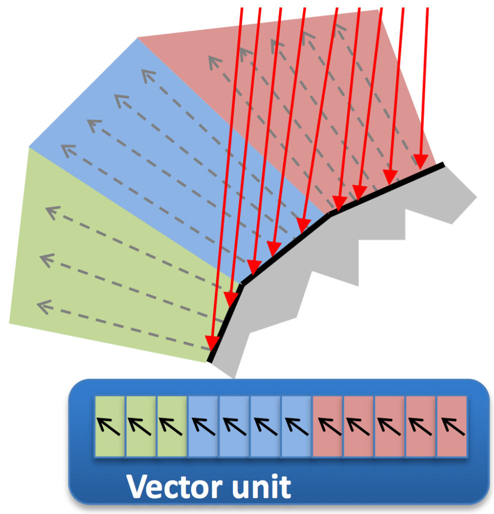


Figure 6: Multiview tracing*: As rays are reflected, they are reordered by viewpoint. This creates the coherent ray groups that can be processed efficiently on GPUs.*

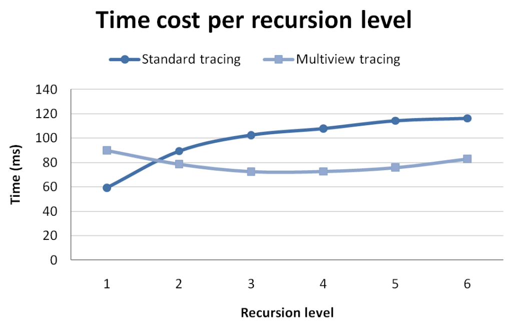


Figure 7: Multiview performance*: Multiview ray tracing out performs standard ray tracing for many specular views. The multiview timings include the time cost of all necessary sorting and reordering.*

we perform a data-parallel bitonic sort using the vector unit. Using the list sorted by ID, it is trivial to find out for each hit or intersection point whether it is a duplicate (i.e. hit by multiple ray samples) by comparing against its successor. In order to compute the visible set, we use a scan operation to compact the list, filtering out all the duplicate triangles. As a result, we have a list of triangles that are hit by rays inside this frustum. Note that this list is only local, i.e. if the same triangle is hit by sample rays from separate queries (running on a different core) the duplicate may not be detected. We handle this case as part of propagation path computation.

4.2 GPU-based algorithm

We build on the bounding volume hierarchy (BVH) ray tracing algorithm [Aila and Laine 2009] and implement our multi-view ray-casting system in CUDA. Rays are scheduled to the vector lanes and traversed through the BVH and intersected against the triangles independently. For primary visibility samples, we use a simple ray tracing kernel that exploits the common ray sources for efficiency. All the secondary rays are evaluated by a general ray tracing kernel with independent traversal stacks for the rays. To decouple the number of visibility samples from the number of CUDA threads used per core, it is possible to distribute the ray samples across multiple blocks. A separate kernel then gathers the results from all these blocks and generates the secondary visibility samples as needed. The algorithm ends when no more active samples exist or the maximum recursion depth is reached in terms of reflections. Our tracing algorithm achieves a 20% speedup over standard ray tracing (see Figure **??**).

4.3 Propagation path computation

As a final step, once the visibility computations have been performed up to a specified order of reflections, we compute the contribution paths to the listener. For each triangle, hit points are retrieved. Then, for each listener, a ray is constructed from the listener towards each point. If the path is unoccluded, there is a contribution. Since we have retained the reflection data for all previous recursions, the entire path is easily checked to verify that the path is valid all the way to the source. It is possible that some paths are computed multiple times (e.g. when the same triangle is found visible from separate primary samples), we also check all contribution passes separately for duplicates. Since all the paths can be specified as a sequence of triangle IDs, this is performed via sorting and testing against the neighboring paths. This avoids counting the contributions multiple times.

5 Path detection

When tracing discrete rays through a scene, a listener detection sphere must be used to record rays that contribute to the acoustic response. The size of this detection sphere is an important consideration when modeling sound propagation using statistical tracing methods. However, if each path is geometrically validated and a sufficiently large number of visibility rays are traced, then the sphere size has no influence on the accuracy of the simulation results. In this section, we show that by varying the size of the detection sphere, we can reduce the number of visibility rays needed, and thus increase the performance of the simulation.

In all geometrical simulations that compute the pressure response in a scene, each possible acoustic path must be validated to be free of occlusion between the source and receiver. In the brute-force image source method, this results in validation rays being cast for every possible sequence of triangle that lie between the source and listener, up to the desired order of reflection. In a ray tracing simulation, each ray that strikes the detection sphere represents a possible acoustic path, and needs to be validated.

The cost of a sound tracing simulation can be represented as:

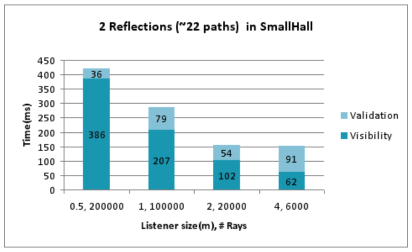
$$S_{time} = N_{rays}C_{visibility} + P_{detect}N_{rays}C_{validation}$$

where S_{time} is the total time cost of the simulation, N_{rays} is the number of visibility rays cast, $C_{visibility}$ is the cost of propagating a visibility ray, P_{detect} is the probability that a visibility ray must be validated, and $C_{validation}$ is the cost of validating an acoustic path.

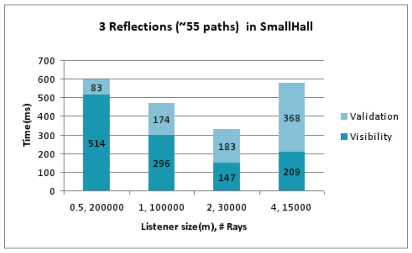
In simulations that trace discrete rays, validation tests must be performed on visibility rays that strike the detection sphere. Since visibility rays that do not intersect the detection sphere do not contribute to the final audio output, it is most productive to only trace rays that eventually intersect the detection sphere. Without a very large detection sphere or a priori knowledge of the scene configuration, this is difficult to achieve in general scenes. In many cases, a significant fraction of visibility rays never intersect the detection sphere.

In order to minimize the cost of the validation tests, it is ideal to only need validate rays that represent valid acoustic paths. Unless the detection sphere is infinitely small, this is difficult to achieve. It is likely that some visibility rays that intersect the detection sphere, but are found to be invalid reflection paths.

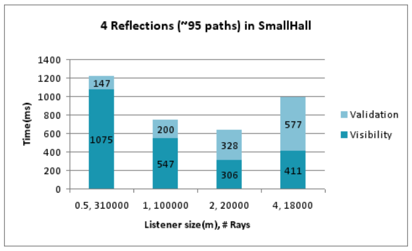
Since the minimal number of visibility rays and the optimal size of the detection sphere are position dependent functions of the scene, we have chosen to retrict ourselves to optimizing only the size of the detection sphere. This allows us to have a fixed ray budget, reducing cost of frame setup time. The size of the detection sphere is chosen to minimize the validation cost in each scene. Figure 8 shows the results from size optimization in a scene.



(a)



(b)



(c)

Figure 8: Detection sphere sizes: *Varying the number of visibility and validation tests reveals an optimal balance for each scene and recursion order. In each graph, the number of visibility samples and detector sphere are adjusted such as always to find 90% of complete set of contribution paths.*

For each scene, iSound finds the optimal detection sphere size using a simple heuristic. Initially, the detection sphere is set to a very large size. For each frame, the size is reduced until the contribution count decreases from the previous frame. The size the grows until no new paths are found. This oscillation continues during the rendering process, allowing the system to adjust to the scene configuration.

6 Audio Processing

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

6.1 Dynamic scenes

In many interactive applications, the source and listener movements could be significantly 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 based schemes [Siltanen et al. 2009] to reduce the audio artifacts.

6.2 Parameter Interpolation

In interactive applications, audio is typically processed in chunks of audio samples, called audio frames. For example, an audio signal sampled at 48 KHz could be processed at 100 audio frames per second, i.e. each audio frame has 480 samples. Between two such adjacent audio frames the propagation paths from a source to a listener may vary leading to a jump in attenuation and delay values per sample at the audio frame boundaries (see Figure 9). We track propagation paths and interpolate the delay and attenuation for a path per audio sample to reduce the artifacts due to changes in the path. To track propagation paths, each path is assigned a unique identifier.

We perform parameter interpolation of propagation paths for audio processing to achieve artifact-free final audio. It is equivalent to computing an IR per sample based on the parameter interpolation per path. Such an approach is physically intuitive and different interpolation schemes can be applied [Tsingos 2001; Tsingos et al. 2004]. We treat each path from a source to a listener 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 on a per sample basis and leads to smooth transition across the audio frame boundary. To interpolate delay, we perform linear interpolation between audio frames. Linear delay interpolation augmented with supersampling and low order fractional delay lines work well to reduce the artifacts due to delay discontinuities between audio frame (see Section 6.3). Figure 9 shows interpolated attenuation per sample for an image source with attenuation discontinuities.

6.3 Variable Fractional Delay

Fractional delay filters have been applied to speech coding, speech synthesis, sampling rate conversion, and other related areas [Välimäki 1995]. In our application, we apply fractional delay filters to handle interpolated delays as sources (or image sources) and listeners move in a virtual environment. Rounding off the interpolated delays to nearest integer as sources and listeners move can lead to noticeable artifacts in the final audio (see Figure 10). Thus, we require efficient *variable* fractional delay filter that can provide fidelity and speed required in virtual environments. Laakso et al. [1996] provide a good survey of FIR and IIR filter design for fractional delay filter.

We use Lagrange interpolation filter due to explicit formulas to construct fractional delay filter and flat-frequency response for low-frequencies. Combined with supersampled input audio signal, we can model fractional delay accurately. Variable delay can be easily modeled by using a different filter computed explicitly per audio sample. To compute an order N Lagrange filter, the traditional methods [Välimäki 1995] require $\Theta(N^2)$ time and $\Theta(1)$ space. However, the same computation can be reduced to $\Theta(N)$ time and $\Theta(N)$ space complexity [Franck 2008]. Many applications requiring variable fractional delay oversample the input with a high-order interpolator and use a low-order variable fractional delay interpolator [Wise and Bristow-Johnson 1999] to avoid computing a high-order variable delay filter during run time. Wise and Bristow-Johnson [Wise and Bristow-Johnson 1999] analyze the signal-to-noise-ratio (SNR) for various low-order polynomial interpolators in the presence of oversampled input. Thus, for a given SNR requirement, optimal supersampled input signal and low-order polynomial interpolator can be chosen to minimize computational and space complexity. Ideally, a highly oversampled input signal is required (see Figure 10) to achieve 60 dB or more SNR for a low-order Lagrange interpolation filter but it might be possible to use low oversampling to minimze artifacts in final audio output [Savioja et al. 2002].

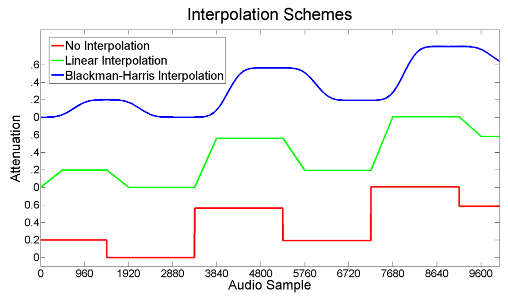


Figure 9: *Different attenuation schemes applied for attenuation interpolation. Discontinuity in attenuation between two audio frames interpolated with linear interpolation and Blackman-Harris interpolation. Delay interpolation is performed using a linear interpolation. Variable fractional delays due to linear delay interpolation are handled by apply low order Lagrange fractional delay filter on a supersampled input audio signal during the audio processing step.*

7 Analysis

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

7.1 Performance

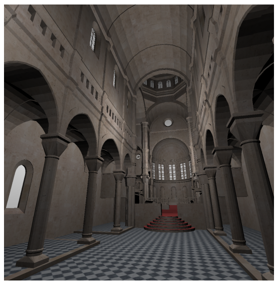
We have used our algorithms on different scenarios with moving sources or objects. The complexity of these scenes is equivalent to those used in current games with tens of thousands of triangles for audio processing. It is possible that the games may use more detailed models for visual rendering. We also add a simple late reverberation filter [Taylor et al. 2009a] to complement the IRs computed by our propagation algorithm. We test the performance of the system on a multi-core PC with NVIDIA GTX 280 GPU, though we use a single core (2.8 Ghz Pentium) for audio processing. We obtain ray tracing performance of about $18-20$ million ray intersections per second on our benchmarks. We used some common benchmarks to evaluate the performance of our system.

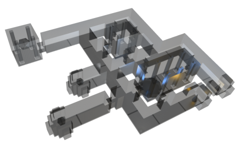
Gamebyro Integration: We have integrated iSound into the Emergent Gamebryo [1] game engine to demonstrate the effectiveness of propagation in dynamic scenes used in current games. This advanced game engine provides animation, physics, visual rendering, and audio effects. The engine consists of a set of tools for game implementation and the focus is primarily on visual rendering. However, like most games, the audio rendering is very simple and only computes the direct contributions from different sources.

In our modification, we have replaced a large section of the engine's audio pipeline (Figure 12) with our custom pipeline. We

[1]http://www.emergent.net/en/Gamebryo-LightSpeed/

(a)


(b)


(c)

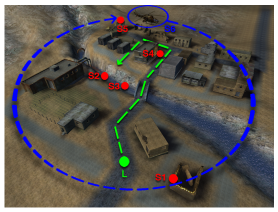
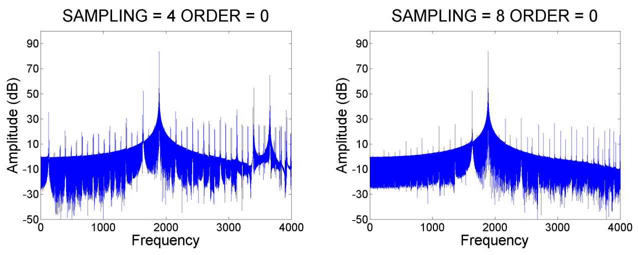
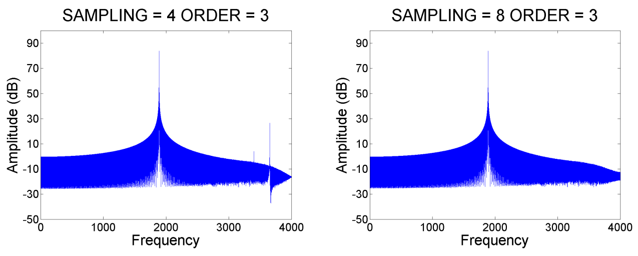

(d)

Figure 11: Benchmarks: *The benchmarks used to test the performance of i-Sound: (a) Small hall model; (b) Sibenik cathedral; (c) Indoor scene generated using Gamebryo; (d) Gamebryo desert scene.*



(a) Lagrange Filter Order = 0



(b) Lagrange Filter Order = 3

Figure 10: *Applying fractional delay filter and supersampling input signal to get accurate doppler effect for a sound source (2KHz sine wave) moving away from the listener at 20 m/s. The sampling rate of the input audio is 8KHz. The supersampling factors are 4x and 8x for left and right figures respectively. Zeroth order and third order Lagrange filters are applied.*

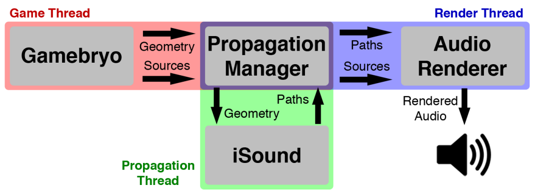
demonstrate the performance on two Gamebyro benchmark scenes.

A propagation manager was added to couple the game engine with our simulation. This manager is responsible for all data communication between the game engine and our sound simulation.

We first begin by extracting the Gamebryo scene graph prior to running our propagation simulation. This scene graph is used to build ray tracing acceleration structures for iSound. If the scene graph changes, updated geometric primitives are sent to iSound to update the tracing hierarchies. The hierarchies are managed on the GPU using recent advances in BVH structures on GPUs.

After the initial scene representation is sent, for each Gamebryo simulation cycle, the propagation manager reads the player position and any source positions from the Gamebryo scene graph. The player position is used as the receiver position in iSound. The propagation simulation then runs, finding acoustic paths between the player to the sound sources, returning an impulse response specific the game engine's scene configuration for that frame.

The resulting impulse response must the be rendered and output to the user's speakers. While it is possible to use the Gamebryo audio rendering pipeline to render the acoustic paths, we use our



(a)

Figure 12: Game engine integration: *Our system has been integrated into the widely used Gamebryo game engine.*

audio processing engine, designed to provide artifact free audio in general dynamic scenes. The propagation manager forwards the data to our rendering engine, and the game loop continues without using the Gamebryo audio rendering.

Model	Tri Count	Bounces	Propagation Time(ms)	Number of Paths	Audio (ms) Processing
Desert	35,534	3R+1D	30	15	3
Indoor scene	1,556	3R+1D	19	27	5
Small Hall	180	3R	16	105	7
Sibenik	79,942	2R	138	16	3

Table 1: Performance of i-Sound on different benchmarks: *The top two benchmarks are from Gamebyro game engine. The Desert is the outdoor scene shown in Figure 1. The third one is a well known simple acoustic benchmark and the fourth one is the model of Sibenik Cathedral. The number of reflections (R) and edge diffraction (D) are given in the second column. The time spent in computing propagation paths (on NVIDIA GTX 280 GPU) and audio processing (on CPU) is also shown. 64k visibility samples were used.*

7.2 Accuracy

Overall, our approach is designed to exploit the computational power of GPUs to perform interactive visibility computations. We use resampling methods and a modified image source method to ensure that most of the contributions to the listener are computed. The overall goal is plausible auralization, but our approach can result in the following errors:

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

2. Diffraction errors: Our formulation is a variation of the UTD method and its application to finite edges can result in errors. Moreover, it is possible that we miss some of the diffraction contributions due to sampling errors.

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

The governing factor in the accuracy and performance of our approach is the number of ray samples that we can shoot in the given time frame. As we can use a higher number of samples on GPUs, we observe no major errors in our benchmarks. This aligns well with the current technology trends as the performance of future GPUs will improve in terms of ray tracing throughput [Aila and Laine 2009]. Another factor that governs the accuracy, is the size of the triangles. Most GA methods are applicable to models where the size of the features or triangles is comparable (or larger) than the audible wavelengths. Moreover, as the size of the primitives increase, it improves the coherence of the multi-viewpoint ray casting algorithm and makes it possible to trace a higher number of ray samples per second.

Model	Direct	1R	2R	3R
Desert	4	12	17	30
Sibenik	32	91	138	185

Table 2: *Average performance (in ms) of our GPU-based path computation algorithm as a function of number of reflections performed. The Desert scene also includes edge diffraction. 64k visibility samples were used.*

7.3 Comparisons

We compare the i-Sound with other general GA methods and specific rendering systems.

Ray tracing algorithms: Previous ray-shooting based propagation algorithms [Krokstad et al. 1968; Vorländer 1989] trace each ray or ray packets independently to compute paths from the sources to the listener. As such, no global visibility information is computed explicitly, and the discrete ray paths are directly used to compute the contributions at the listener. These methods model the listener as an object of some size to determine when a discrete ray is close enough to the listener to be considered a valid contribution path. These can lead to missed contributions, duplicate contributions (see Figure 5), or statistical errors [Lenhert 1993]. Since we can adjust the detection sphere size for maximum performance. much faster than prior ray tracing methods.

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

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

Other systems: ODEON is a popular acoustics software which can compute specular reflections, diffuse reflections, and diffraction [Christensen 2009] and is perhaps the most widely used commercial system for architectural acoustics. ODEON performs early specular reflections and diffraction using a combination of ray tracing and image source method [Christensen 2009]. For diffraction, ODEON computes at most one diffraction path from a source to a listener. The diffraction impulse response is computed [Pierce 1974] only when the direct path between the source and the receiver is obstructed and a diffraction edge is found around the geometry obstructing the source-receiver direct path. CATT-Acoustic [CAT 2002] is another popular 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 [GEN 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. The diffraction contribution for free edges is computed using Kurze-Anderson [Kurze 1974] formula for free edges. It does not support diffraction for non-free edges. RAVEN at RWTH Aachen University is a framework for real-time auralization 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 auralization 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 limits the system to static source positions. RESound [Taylor et al. 2009b] is also a real-time auralization system. It is based on a combination of frustum tracing and ray tracing to handle specular reflectons, diffuse reflections, and diffraction. Multiple orders of diffraction based on UTD can be handled along with dynamic scenes with moving source and scene geometry.

7.4 Audio Processing

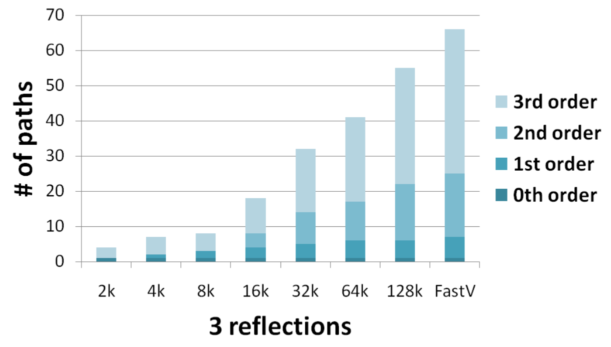
Our interpolation scheme presented in Section 6.2 produces smooth audio. It can be improved by in two different ways: (a) interpolation of image sources could be based on predicting their new position based on their current positions and velocities [Tsingos 2001] and (b) if the number of image sources (or paths) is large, it is possible to apply clustering and perceptual acceleration [Tsingos et al. 2004; Moeck et al. 2007] for efficient audio processing. Currently, our audio processing step does not interpolate direction of an image source relative to the listener but we encode it by computing delays and attenuation for left and right ears for 3D audio.

7.5 Accuracy of Propagation Paths

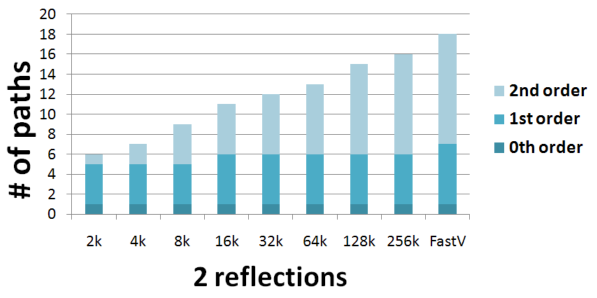
We compared the propagation paths and contributions computed by i-Sound with varying ray density with the accurate object-space geometric propagation method [Chandak et al. 2009]. The results are highlighted in Figure 13 and we compare the number of paths computed after each order of reflections. For the first few orders of reflections, i-Sound is approaches the accuracy of image-source methods. Even after two or three reflections, i-Sound is able to compute $90-95\%$ of the propagation paths at interactive rates on current GPUs.

7.6 Limitations

We have highlighted the main sources of errors in our approach. In addition, our current system only performs 1st order diffraction computation and the UTD method may not be accurate for finite-edges. The overall GA method is only accurate for the first few orders of reflections and not applicable to scenes with very small and detailed features. Moreover, our current approximation of late reverberation is limited to indoor scenes. Furthermore, the system does not model diffuse reflections. Our resampling algorithm is based on a heuristic that may not work well for some benchmarks.



(a)



(b)

Figure 13: Path Accuracy: *We compare the paths computed using i-Sound with the paths generated with an accurate image-source simulation algorithm (FastV) for: (a) Small hall model; (b) Sibenik Cathedral model. We are able to compute* 90% *of all the paths at interactive rates with 128k samples. Moreover, i-Sound is quite accurate for the first few orders of reflections, finding most of the second order paths with only 32k samples.*

8 Conclusion and future work

We have presented a new auralization algorithm that can handle dynamic scenes. Our modified image source method decouples the visibility computation from the actual path contributions. Moreover, we exploit the computational power of GPUs to perform the visibility computations in parallel and thereby achieve significant performance improvement over prior GA methods for the same number of contributions. In practice, we are able to compute most of the contribution paths to the listener in game like scenes with thousands of triangles. We also present a novel audio rendering scheme to reduce the artifacts. Overall, we are able to generate plausible audio rendering in game-line scenes with dynamic objects at $20 - 30$ fps on current PCs. Moreover, our approach aligns well with the current technology trends and its accuracy and performance would improve the increased parallelism available in the GPUs.

There are many avenues for future work. We would like to extend to scenes with a high number of sound sources based on clustering methods or perceptual rendering algorithms. Furthermore, we would like to perform perceptual evaluation of our system and perform user studies. The ray tracing algorithm can also be used to perform diffuse reflections. It may be possible to use other resampling methods to improve the accuracy of our algorithm. Finally, we would like to integrate our auralization system with other interactive applications and evaluate its performance.

9 Acknowledgements

We are grateful to Lauri Savioja for his feedback on an earlier draft of this paper. Emergent Game Technologies provided us an academic license of the development kit of Gamebryo. This work was partially supported by ARO, RDECOM, NSF, and Intel

References

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

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 (April), 943–950.

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

Borish, J. 1984. Extension to the image model to arbitrary polyhedra. *The Journal of the Acoustical Society of America 75*, 6 (June), 1827–1836.

CATT. 2002. *CATT-Acoustic User Manual*, v8.0 ed. Sweden. http://www.catt.se/.

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 (Nov.-Dec.), 1707–1722.

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.

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

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.

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

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

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*, 21–32.

Funkhouser, T., Tsingos, N., Carlbom, I., Elko, G., Sondhi, M., West, J., Pingali, G., Min, P., and Ngan, A. 2004. A beam tracing method for interactive architectural acoustics. *Journal of the Acoustical Society of America 115*, 2 (February), 739–756.

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

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

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

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 (July), 118–125.

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

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 (Jan), 30–60.

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

Lenhert, H. 1993. Systematic errors of the ray-tracing algoirthm. *Applied Acoustics 38*, 207–221.

Lentz, T., Schroder, D., Vorländer, M., and Assenmacher, I. 2007. Virtual reality system with integrated sound field simulation and reproduction. *EURASIP Journal on Advances in Singal Processing*. Article ID 70540, 19 pages.

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*, ACM, New York, NY, USA, 189–196.

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

POPE, J., CREASEY, D., AND CHALMERS, A. 1999. Realtime room acoustics using ambisonics. *Proc. of the AES 16th Intl. Conf. on Spatial Sound Reproduction*, 427–435.

POPOV, S., GNTHER, J., SEIDEL, H.-P., AND SLUSALLEK, P. 2007. Stackless KD-Tree Traversal for High Performance GPU Ray Tracing. *Computer Graphics Forum (Proc. EUROGRAPHICS) 26*, 3, 415–424.

SAVIOJA, L., HUOPANIEMI, J., LOKKI, T., AND VÄÄNÄNEN, R. 1999. Creating interactive virtual acoustic environments. *Journal of the Audio Engineering Society (JAES) 47*, 9 (September), 675–705.

SAVIOJA, L., LOKKI, T., AND HUOPANIEMI, J. 2002. Auralization applying the parametric room acoustic modeling technique - the diva auralization system. *ICAD*.

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 (July), 604–619.

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).

SVENSSON, U. P., FRED, R. I., AND VANDERKOOY, J. 1999. An analytic secondary source model of edge diffraction impulse responses . *Acoustical Society of America Journal 106* (Nov.), 2331–2344.

TAYLOR, M., CHANDAK, A., ANTANI, L., AND MANOCHA, D. 2009. Resound: interactive sound rendering for dynamic virtual environments. In *MM '09: Proceedings of the seventeen ACM international conference on Multimedia*, ACM, New York, NY, USA, 271–280.

TAYLOR, M. T., CHANDAK, A., ANTANI, L., AND MANOCHA, D. 2009. Resound: interactive sound rendering for dynamic virtual environments. In *MM '09: Proceedings of the seventeen ACM international conference on Multimedia*, ACM, New York, NY, USA, 271–280.

TSINGOS, N., FUNKHOUSER, T., NGAN, A., AND CARLBOM, I. 2001. Modeling acoustics in virtual environments using the uniform theory of diffraction. In *SIGGRAPH 2001, Computer Graphics Proceedings*, 545–552.

TSINGOS, N., GALLO, E., AND DRETTAKIS, G. 2004. Perceptual audio rendering of complex virtual environments. *ACM Trans. Graph. 23*, 3, 249–258.

TSINGOS, N. 2001. A versatile software architecture for virtual audio simulations. In *International Conference on Auditory Display (ICAD)*.

TSINGOS, N. 2009. Pre-computing geometry-based reverberation effects for games. *35th AES Conference on Audio for Games*.

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

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.

WAND, M., AND STRASSER, W. 2004. Multi-resolution sound rendering. In *SPBG'04 Symposium on Point - Based Graphics 2004*, 3–11.

WENZEL, E., MILLER, J., AND ABEL, J. 2000. A software-based system for interactive spatial sound synthesis. In *International Conference on Auditory Display (ICAD)*.

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*.