Guided Multiview Ray Tracing for Fast Auralization

Micah Taylor, Anish Chandak, Qi Mo, Christian Lauterbach, Carl Schissler, and Dinesh Manocha,

Abstract—We present a novel method for tuning geometric acoustic simulations based on ray tracing. Our formulation computes sound propagation paths from source to receiver and exploits the independence of visibility tests and validation tests to dynamically guide the simulation to high accuracy and performance. Our method makes no assumptions of scene layout and can account for moving sources, receivers, and geometry. We combine our guidance algorithm with a fast GPU sound propagation system for interactive simulation. Our implementation efficiently computes early specular paths and first order diffraction with a multi-view tracing algorithm. We couple our propagation simulation with an audio output system supporting a high order interpolation scheme that accounts for attenuation, cross-fading, and delay. The resulting system can render acoustic spaces composed of thousands of triangles interactively.

Index Terms—Sound propagation, ray tracing, parallelization.

1 Introduction

Auditory displays and sound rendering are frequently used to enhance the sense of immersion in virtual environments and multimedia applications. Aural cues combine with 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 visually 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 a virtual environment along with knowledge of sound sources and receiver location, the basic auralization pipeline consists of two parts: sound propagation and audio processing. The propagation algorithm computes a spatial acoustic model resulting in impulse responses (IRs) that encode the delays and attenuation of sound traveling from the source to the receiver along different propagation paths representing transmission, reflection, and diffraction (see Figure 1). Whenever the source, receiver, or the objects in the scene move, these propagation paths must be recomputed at interactive rates. An 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.

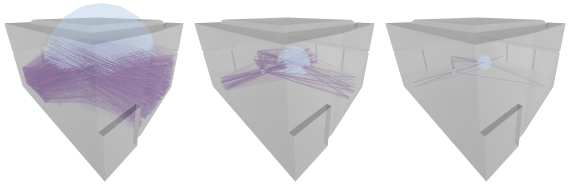
There is extensive literature on modeling the propagation of sound, including reflections and diffraction. Most prior work for interactive applications is based on Geometric-Acoustic (GA) techniques such as image-source methods, ray-tracing, path-tracing, beam-tracing, ray-frustum tracing, etc. However, while interactive systems [44] do exist, it is widely regarded that current GA methods do not provide enough flexibility and efficiency needed for use in general interactive applications [54]. Therefore, current games precompute and store reverberation filters for a number of locations [41]. These filters are typically computed based on occlusion relationships between the sound source and the receiver or tracing a low number of feeler rays into the scene. Other applications use dynamic artificial reverberation filters [28] or other filters 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.

In this paper, we show that by balancing the visibility and validation costs inherent in GA methods, much higher performance can be achieved. We present a simple algorithm to guide visibility and validation cost in GA simulations for practical use. We use our guidance system to direct a fast GPU ray tracer for sound propagation. Our GPU ray tracer uses multi-view ray tracing to compute multiple sound reflections in parallel. Additionally, a fast diffraction detection method is used to further enhance acoustic performance. We show that our system can achieve interactive performance on complex scenes while maintaining accuracy using a cost guidance algorithm.

The main components of our work include:

1) **Guided visibility and validation:** We present a novel algorithm to reduce the cost of the visibility and validation steps. Using simple algorithms, the cost of both operations can often be reduced while retaining an accurate set of sound propagation paths.
2) **Multi-viewpoint ray casting:** We describe a ray casting algorithm that performs approximate visible

- *M. Taylor, A. Chandak, Q. Mo and D. Manocha are with the Department of Computer Science, University of North Carolina, Chapel Hill. E-mail: see http://gamma.cs.unc.edu/Sound/Guided/*
- *C. Lauterbach is with Google Inc.*



(a) (b) (c)

Fig. 1: Cost reduction: *Figures (a) - (c) show how our algorithm progresses from a dense visibility ray sampling to a sparse visibility ray sampling. Additionally, our algorithm reduces the receiver sphere radius to reduce the number of validation tests performed. Time cost is reduced while accuracy is maintained.*

surface computations from multiple viewpoints in parallel. We use this to accelerate specular reflection calculations.

3) **Diffraction computation by barycentric coordinates:** To enhance our implementation, we have developed a low cost method of detecting rays near diffracting edges. Using the barycentric coordinate of ray intersections, we can create an origin for diffraction propagation.
4) **Interactive auralization:** Using the above algorithms, we implemented a GPU based system to demonstrate the method.

The overall approach is easily coupled with GA simulations built on discrete ray tracing and can be adapted for other GA methods. Additionally, our algorithm can easily support moving sources and receivers, as well as moving objects. In practice, our GPU implementation achieves up to an order of magnitude performance improvement over prior interactive GA methods that use multiple CPU cores.

Organization: The rest of the paper is organized as follows: Section 2 surveys prior work. In section 3, we present an overview of GA methods and our algorithm. Section 4 describes the cost function of tracing propagation simulations and our guidance method. Section 5 details our GPU based multi-view tracing, while our audio processing implementation is covered in Section 6. We analyze the accuracy and performance of our method in Section 7.

2 Previous Work

The computation of IRs is the result of solving the wave equation. However, current numerical methods used to solve the wave equation have high complexity and are mainly limited to static scenes. Often, GA methods are used when interactive performance is desired. In this section, we briefly survey prior works 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 [2], [9] which compute specular reflection paths by computing virtual or secondary sources. Ray tracing methods [30], [58], [7], [50] compute propagation paths by generating rays from the source or receiver position and following each ray individually as they propagate through the environment. Some geometric propagation algorithms perform object-precision visibility computation based on beam tracing [24], [33], BSP trees [36] or conservative frustum tracing [12]. Other fast algorithms based on approximate ray-frustum tracing [14] and multipole expansion [21] have been developed. Work has also been done on adapting GA methods for use on GPUs [27], [43], [16], [47]. There have also been advances in propagation of diffuse acoustic reflections [36], [20].

There has been much work combining GA methods with diffraction based on the Uniform Theory of Diffraction (UTD) [29] and Biot-Tolstoy-Medwin (BTM) methods [8]. The BTM method is considered more accurate and can be formulated for use with finite edges [51]. The UTD is often used if interactive rates are desired [55], [52], [17], since even accelerated BTM methods are not fast enough [5]. Some research has focused on interactive systems including diffraction with diffuse reflections [52] or for game-like environments [46].

2.2 Ray Count and Receiver Size

In GA simulations using rays, the rays must be somehow detected at the receiver point. Since intersecting an arbitrary ray with a point is unlikely, a sphere is often used as a detector to collect the rays [38]. The size of the sphere is related to the number of rays collected, as well as the accuracy of the simulation. A large sphere size can lead to incorrect sound paths being detected [35]. Different methods have been developed to select an appropriate sphere size, usually based on the number of rays traced or distance between source and receiver [35], [62]. The methods for selecting an appropriate ray count are based the assumption that all surfaces are visible to any one source position [35]. In scenes where most paths occur after at least one reflection or the scene configuration changes, it is difficult to reliably predict an appropriate sampling density.

2.3 Audio Processing

Moving sound sources, receivers, and scene objects can cause variations in the impulse response from source to receiver and could lead to artifacts in the final audio output. Several methods have been proposed to reduce the artifacts in scenes with moving sources and receivers, including motion prediction [53], simple interpolation and windowing techniques [60], [45], [49], and imposing restrictions on source and receiver motion [52]. 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 [56], [59] and perceptual methods [37].

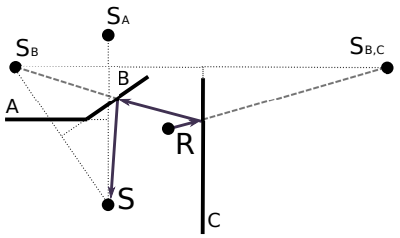


Fig. 2: Image source: *From a source point S, image sources are created by reflecting the source point over the walls. In order to compute the contribution paths between the source and receiver R using walls B and C, a ray is first traced to the image-source $S_{B,C}$. From the intersection point on 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 R.*

3 Geometric acoustics

In most geometric acoustic simulations, the largest time cost is performing sound propagation. The acoustic response from propagation depends on source and receiver position, as well as the layout of the virtual scene. In dynamic scenes, propagation paths must be recalculated if any aspect of the scene configuration changes.

In GA methods, the acoustic response is often divided into three parts: direct, early, and late. Sound that takes a direct unoccluded path from source to receiver forms the direct response. The early response formed by sound waves that travel to the receiver by way of a few (often 4 to 6) orders of reflection or diffraction. The late response is the remaining later portion of the response, often hundreds of reflections.

3.1 Specular reflections

In this paper, we focus on finding the specular reflection paths that form the early response. With respect to specular reflections, all GA methods are variations on the image source method [2]. In the image source method, the first order of reflection is modeled by creating reflection images of the source over all of the triangles in the scene. For each successive order of reflection, these image sources are likewise reflected. This process creates a tree of the possible reflection sequences sound can take. We call this a *visibility tree*.

Not every reflection sequence in the visibility tree may represent a valid sound reflection path to a given receiver. As such, a validation step can test that each sequence in the tree is occlusion free between the source and receiver. A standard method to verify that a path is occlusion free is to trace a ray from the receiver point back to the reflection images, verifying that the path reflects off the triangles that caused the original image source reflection. This results in a set of occlusion free paths between the source and receiver. We call this step *path creation*. Figure 2 shows images and paths for a simple scene.

Most GA simulations are variants of this process: generating a visibility tree and creating paths from it. In the visibility stage, the scene is explored with primitives such as rays, beams, or frusta. Once the visibility of the scene has been queried, the path creation stage creates a set of acoustic contribution paths. Using the material properties in the scene, these contribution paths can be attenuated and combined into an impulse response that is used by the audio processing system to output the final audio.

3.2 Diffraction

Diffraction in sound propagation describes the way sound waves scatter at edges. The scattering can result in the sound wave propagating behind corners and out of line-of-sight. Diffraction is an important effect to simulate since it allows sounds to naturally fade as the source or receiver moves out of line-of-sight.

When a sound wave encounters an edge, diffraction scattering occurs all around an edge. Simulating this scattering in GA methods requires complex visibility queries from the edge and greatly increases the final number of propagation paths in the scene [10]. To reduce computational cost, diffraction calculations can be restricted to the *shadow region* [55]. The shadow region is the region that is occluded from direct contribution from the source.

Similar to specular reflection sequences, diffraction sequences create entries in the visibility tree that must be validated to be occlusion free. Once the visibility of the path sequence has been tested, the path can be attenuated based on the edge and path properties.

3.3 Our GA algorithm

We focus on efficiently finding the first few specular reflections of the acoustic response. In addition, we describe how low order diffraction paths may be found at minimal cost. All our algorithms are designed to easily scale on parallel hardware.

We use ray casting to compute the visible triangles from the source by tracing sample rays distributed randomly around the source point. These rays intersect the scene triangles and the hit data is inserted into the visibility tree. For each ray that strikes a triangle, a reflected ray is emitted from the intersection points and its resulting hit data is also inserted into the visibility tree. If the ray hitpoint is near a diffracting edge on the triangle, a possible diffraction path exists and the path details are recorded. In our simulation, diffraction paths are not validated, instead, only the shortest and most accurate unique diffraction paths are retained.

The use of ray tracing to accelerate image source algorithms is not a new idea [58], but to our knowledge, no work has been done on adjusting ray tracing algorithms to suit the needs of GA methods. We use a new multi-view ray casting algorithm (see Section 5) 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.

Once the visibility tree is constructed, we find valid propagation path sequences in the tree. We model a sphere

at the receiver and only visibility rays that intersect the sphere are validated. The accuracy of this approach is governed by the sampling density used in the primary visibility step and the size of the receiver sphere.

In addition to our new ray tracing algorithm, we propose a method to dynamically select appropriate sample densities and receiver sphere size during the simulation process. Section 4 details our approach to minimizing cost while maximizing accuracy.

4 GUIDED PROPAGATION

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

Some GA methods, such as beam tracing, compute a very accurate, minimal visibility tree. Since the reflection data in the tree is very detailed, the complete set of valid paths can be created very quickly [24]. However, generating such an accurate tree is costly. Other methods, such as conservative frustum culling [13], compute accurate trees that may be overly conservative. The visibility tree can be generated faster, but the path creation time may increase, since some of the paths will be occluded and must be discarded. This concept is similar for other GA based methods, leading to a simple cost function for specular propagation:

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

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

4.1 Ray traced propagation cost

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

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

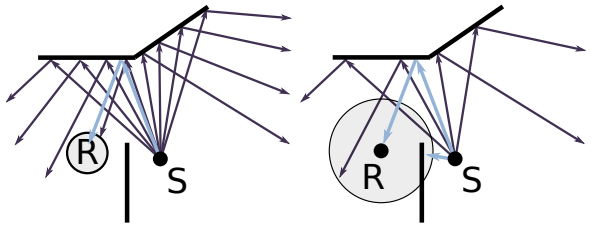


Fig. 3: Sample-based visibility: *Visibility rays are traced from source S into the scene. Paths that strike receiver R are then validated. (a) A small receiver requires dense visibility sampling to find the propagation path. (b) Using a larger receiver allows sparse sampling resulting in fewer visibility tests, however more validation tests are need to remove invalid path sequences.*

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

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

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

The cost of visibility and path creation are scene and position dependent and can be minimized by efficient ray tracing techniques or reducing the number of tests in the propagation step. We discuss efficient tracing techniques in Section 5. To minimize the number of tests, we decouple visibility and validation and use a guidance algorithm to minimize the cost of each independently.

4.2 Guidance algorithm

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

Conducting a large number of visibility tests N with a high probability P of validating each path sequence (by way of a large receiver size) will likely find the wanted paths, but with high time cost. If N is reduced, there will be fewer rays that encode each unique path sequence. Accuracy can be maintained if the probability of these rays being detected is increased by increasing receiver size. Similarly, with a high N, the receiver size can likely be decreased (reducing P) while still detecting the necessary path sequences (see Figure 3).

The base cost values of visibility rays V, and of path validation L, vary vastly based on the underlying algorithms and implementations. Also, when the source or receiver

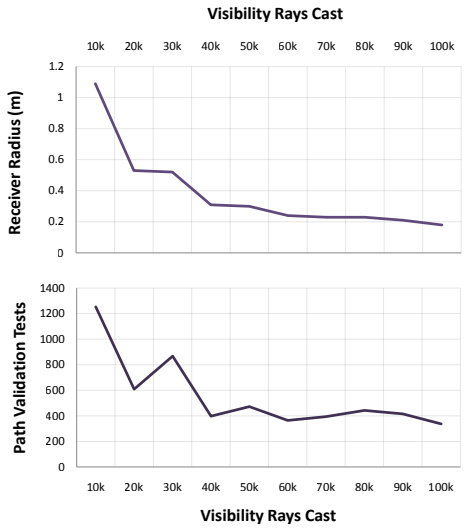


Fig. 4: Propagation test count: *With a goal of finding 90% of the total paths in the scene, an increasing number of visibility rays are traced and the minimum required size of the receiver sphere changes accordingly. With sparse visibility sampling, a large sphere is required, resulting in many validation tests. With dense sampling, the sphere size can be reduced. For specific cost values for visibility and path validation tests, some minimal total cost exists.*

move, or objects in the scene move, V and L can change, altering the total cost function. Indeed, since the optimal values may change throughout the course of the simulation, it is difficult to find the best values to reduce the time cost.

Instead of attempting to find the optimal values, our guidance algorithm seeks to independently minimize the number of visibility and validation tests used in propagation. It works by adjusting the number of visibility rays cast and the size of the detection sphere. These two factors correspond to N and P, respectively. Figure 4 shows an example cost function in terms of the minimal number of tests needed to find a specific percentage of the total paths in a scene.

Our algorithm monitors the count of unique contribution paths found during a single simulation frame. The goal on subsequent frames is to find an equal or greater number of paths to this maximum recorded path count, while using a minimal number of visibility and validation tests. The algorithm achieves this by reducing the number of rays traced and the size of the detection sphere. If at any time the path count decreases (i.e. a path is lost), the algorithm responds by increasing the number of rays and receiver size until the path is recovered. If the path cannot be recovered after aggressive adjustment, the lower path count is selected as the maximum known path count. If at any time the current path count exceeds the recorded maximum count, the maximum count is updated to the new higher count. This allows our method to respond conservatively to scene changes.

On startup, the algorithm begins by tracing a user specified number of rays and with a user specified receiver sphere size. We use 50k rays and a sphere radius of $\frac{1}{4}$ the length of the maximal scene axis in our tests. From this point, the number of rays and sphere size are reduced to find local minima of the total cost function without decreasing accuracy. A small initial number of visibility rays can lead to sampling errors that are further discussed in Section 7.

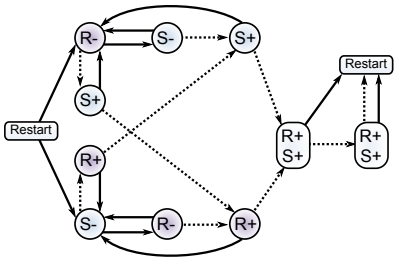


Fig. 5: Guiding state machine: *This state machine tracks the number of unique contribution paths found. Solid lines are followed if the current path count matches the recorded maximum count, dashed lines are followed if the path count is less than the recorded maximum. States marked $R+$ and $S+$ increase the ray count and sphere size, while states marked $R-$ and $S-$ decrease the ray count and sphere size, respectively. At the* Restart *state, the maximum paths count is set to the current count. The $(R+, S+)$ states attempt to recover lost paths before recording a new count. The main top and bottom arms focus on reducing rays and receiver size respectively.*

Our guiding algorithm is easily represented as a state machine. Figure 5 shows the details of the state machine. After each simulation cycle a new state is found and the propagation parameters are adjusted. This process continuously adjusts the number of rays traced and the size of the receiver spheres. Each receiver sphere is adjusted independently; if the state machine enters a state where sphere size is increased, but no paths have been missed to a certain receiver, that specific receiver sphere is not increased. The accuracy and performance of the algorithm is discussed in Section 7.

5 Multi-view GPU Ray Tracing

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

5.1 GPU Propagation

We divide the processing work between the host and GPU device. The host handles all audio processing, while the GPU device computes the propagation results. Figure 6 shows the overall details. Our propagation algorithm traces visibility rays through the scene, intersects them with a receiver sphere, and validates the possible propagation paths to be occlusion free.

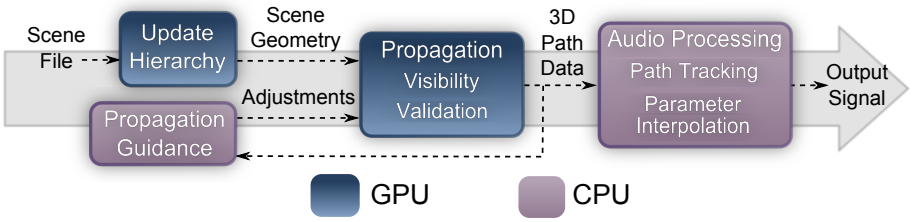


Fig. 6: Implementation Overview: *All scene processing and propagation takes place on the GPU: hierarchy construction, visibility computations, specular and edge diffraction. The sound paths computed using GPU processing are returned to the host for guidance analysis and audio processing. The guidance results are used to direct the next propagation cycle.*

For general ray tracing, previous approaches have investigated efficient methods to implement ray tracing on massively parallel architectures such as GPUs, which have a high number of cores as well as wide vector units on each core. Current methods for GPU-based ray tracing mainly differ in the choice of acceleration structure such as kd-trees [64] 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 [42] and some recent approaches can evaluate them independently [1], as more memory is available for local computation on current GPUs.

We build on the bounding volume hierarchy (BVH) ray tracing ideas in [34] and implement our multi-view ray casting system in CUDA. This allows us to render scenes with dynamic geometry, as the BVH can be refit or rebuilt as needed. While NVIDIA provides a ray tracing system [39] for use on CUDA hardware, we use our own fast ray tracer due to its flexibility.

Rays are bundled into packets that are executed on each core while scheduling each ray on a lane in the vector unit. The rays are then traversed through the BVH and intersected against the triangles. For primary visibility samples, we use a simple ray tracing kernel that exploits the common ray sources for efficiency. Reflections are handled by a secondary kernel which loads the previous hit data and traces a reflection ray. To decouple the number of visibility samples from the number of threads allocated for processing, we iteratively process visibility samples in small thread blocks until all samples have been traced. As rays exit the scene, they are removed from the work queue and no longer processed. The algorithm ends when no more active samples exist or the maximum recursion depth is reached in terms of reflections. At any point during tracing, if the ray coherence is reduced past a user specified threshold, multi-view tracing is employed.

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

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

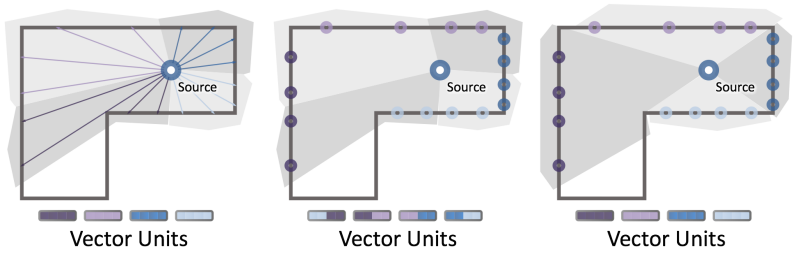
5.2 Multi-View Tracing

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

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

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

On the GPU, each thread block is treated as if it is running on independent hardware from all other blocks. Our ray packets are formed with a ray for each thread in the thread block. When all the rays in a packet share a common origin, the packet represents a single ray traced view that can be traced very efficiently. The goal of our



(a) (b) (c)

Fig. 7: Multiview tracing: *(a) From the source, rays are grouped into packets that can be efficiently processed on the vector units. (b) However, a single packet may hit multiple surfaces, resulting in reflection packets that are inefficient. (c) We reorder packets so that each reflection view can be traced efficiently.*

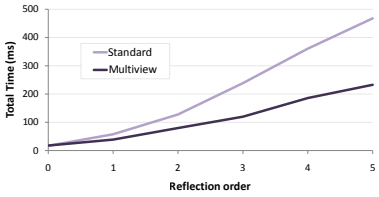


Fig. 8: Multiview performance*: Multi-view ray tracing out performs standard ray tracing for scenes (80k triangle scene shown) with many specular views. The multi-view timings include the time cost of all necessary sorting and reordering.*

multi-view system is to achieve this as often as possible. Our system detects when packets become incoherent and restructures all rays into more efficient packets.

Given the results of visibility ray casting as a list of triangle IDs, we perform a data-parallel bitonic sort using the vector unit. Using the list sorted by ID, it is trivial to find out for each hit or intersection point whether the ID is a duplicate (i.e. hit by multiple ray samples) by comparing against its successor. If all IDs are duplicates, all rays in the packet hit the same triangle and reflection rays are likely to share the same origin (at the image source) and direction. Such a packet is likely to be coherent and efficient. However, if the sort reveals multiple triangle IDs, the reflection rays will likely not share a common origin, and are probably incoherent.

After each trace recursion, the coherency test is applied to each ray packet. The number of packets that are likely to be incoherent is then recorded. The percent of packets that are incoherent is compared to a user specified limit (we use 80% in all our tests). If the threshold is exceeded, the ray packets are reordered into more efficient views (see Figure 7).

This is done by performing a parallel radix sort on triangle ID across all hit data. The hit and ray data is reordered according to the sort results. Since each ray's index is no longer indicative of its parent ray, an index table is also created to find parent hit and ray data. As a result of this view reordering process, our multi-view tracing algorithm performs high specular reflection orders faster than standard ray tracing (see Figure 8).

5.3 Diffraction

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

Similar to other approaches, only certain edge types [52] 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 [6]. These coordinates represent how far an intersection point in the interior of a triangle is from the triangle vertices; in the intersection routine, the barycentric coordinates are used to detect if the ray hit point lies within the triangle boundaries. We reuse the barycentric coordinates when detecting if diffraction rays need to be traced. If a hit point's barycentric coordinates show that the hit point is within 10% of a diffracting edge, as measured along the triangle surface, we consider the ray close enough to the edge to continue diffraction propagation. Using the barycentric coordinates of the hit point, we project the hit point on to the diffracting edge. This point on the edge becomes the origin from which diffraction propagation takes place.

Given the barycentric coordinates of the hit point: λ_1, λ_2, λ_3 and a triangle with edges e_1, e_2, e_3, a diffraction origin o can be created on e_3 from a hit point that hit near e_3 as follows. Figure 9 shows the arrangement visually.

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

Rays are then traced from the diffracting origin according to the UTD: the outgoing diffracting rays have the same angle relative to the diffracting edge as the incident ray. However, we only trace diffraction rays in the shadow region, not the full diffraction cone, as described by the UTD. The diffracting edge is shared by the triangle that was hit by the ray and an occluded triangle (possibly the backface of the hit triangle). These two triangle faces form a diffracting wedge as describe by the UTD. In order to create

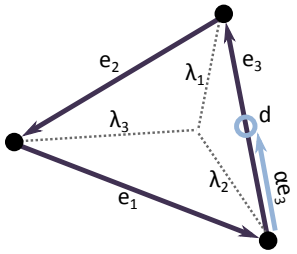


Fig. 9: Barycentric diffraction hit points: *Using the barycentric coordinates of a ray hitpoint, a diffraction origin d can be found on the triangle edge.*

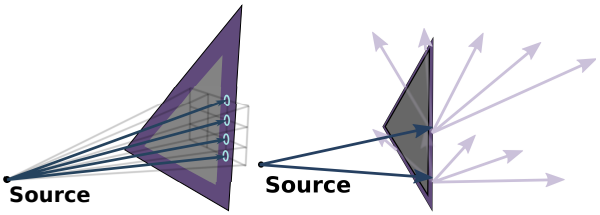


Fig. 10: Edge Diffraction*: (a) Rays near the edge are detected for resampling. (b) Diffraction samples are cast through the shadow region, bounded by the adjacent triangle.*

diffraction rays for the shadow region, we create a vector based at the origin that was previously found and with the incident ray's direction. This vector is rotated towards the occluded triangle face, sweeping out only the part of the diffraction cone that is in the shadow region. Since we only conduct first order diffraction in our simulation, we only trace rays that pass near the receiver spheres. This greatly reduces the required computation. It is also possible to trace higher order diffraction rays by sampling the swept surface of the cone, discretizing the region. See Figure 10 for details.

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

If a diffraction path is found to reach the receiver, the acoustic signal must be attenuated based on the diffraction interaction. Our system uses the UTD attenuation function [29], adjusted for smoothness at the shadow region. This function gives the band attenuation based on the properties of the diffracting edge and the ray geometry. We apply the function for each of the frequency bands that the user has selected, resulting in appropriate attenuation of high frequencies when the path is diffracted.

5.4 Path Creation

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

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

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

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

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

Unlike specular paths, our diffraction validation is approximate. Rather than creating each optimally short UTD path, we select the most optimal path from the unmodified visibility samples. On the host, the diffraction path are sorted by a visibility sequence hash. For each unique diffraction sequence, the shortest path is selected for audio output. This path has a possible error of $< 10^\circ$ in edge angle and $< 10\%$ error spatially on the triangle surface.

6 Audio Processing

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

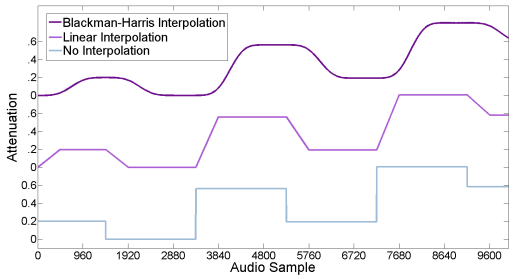
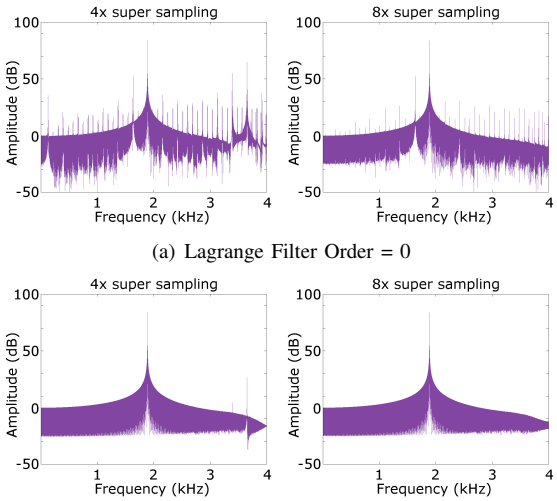


Fig. 11: Interpolation schemes: *Different attenuation schemes applied for attenuation interpolation. Discontinuity in attenuation between two audio frames interpolated with linear interpolation and Blackman-Harris interpolation. Delay interpolation is performed using a linear interpolation. Variable fractional delays due to linear delay interpolation are handled by applying low order Lagrange fractional delay filter on a supersampled input audio signal during the audio processing step.*



(a) Lagrange Filter Order = 0

(b) Lagrange Filter Order = 3

Fig. 12: Fractional delay: *Applying fractional delay filter and supersampling input signal to get accurate Doppler effect for a sound source (2 KHz sine wave) moving away from the receiver at 20 m/s. The sampling rate of the input audio is 8 KHz. The supersampling factors are 4x and 8x for left and right figures respectively. Zeroth order and third order Lagrange filters are applied.*

dynamic scenes and to minimize artifacts in final audio output.

6.1 Dynamic scenes

In many interactive applications, the source and receiver movements could be quite large. This can lead to sudden changes in the propagation paths (i.e. delay and attenuation) from a source to a receiver. New paths may suddenly appear when a receiver comes out of a shadow region or due to the movement of scene geometry. Existing paths may disappear due to occlusion or sampling errors. To handle such scenarios, we track the paths and interpolate the changes in the paths to produce artifact-free audio output. Our approach combines parameter interpolation [60], [45], [44] and windowing based schemes [49] 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 receiver may vary, leading to a jump in attenuation and delay values per sample at the audio frame boundaries (see Figure 11). 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 [53], [56]. We treat each path from a source to a receiver as a parameter and represent it as an equivalent image source, i.e. delay, attenuation, and direction in 3D space relative to the receiver. Each image source is treated as an independent audio source during audio processing. Thus, changes in the paths are equivalent to changes in the corresponding image sources.

As an image source changes, its attenuation, delay, or direction relative to receiver may change. We perform attenuation interpolation between audio frames by applying a windowing function (Blackman-Harris) to smoothly interpolate attenuation at the audio frame boundary. This interpolation is performed on a per sample basis and leads to smooth transition across the audio frame boundary. To interpolate delay, we perform linear interpolation between audio frames. Linear delay interpolation augmented with supersampling and low order fractional delay lines work well to reduce the artifacts due to delay discontinuities between audio frames (see Section 6.3). Figure 11 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 [57]. In our application, we apply fractional delay filters to handle interpolated delays as sources (or image sources) and receivers move in a virtual environment. Rounding off the interpolated delays to nearest integer as sources and receivers move can lead to noticeable artifacts in the final audio (see Figure 12). Thus, we require efficient *variable* fractional delay filter that can provide fidelity and speed required in virtual environments. A good good survey

of FIR and IIR filter design for fractional delay filter is provided in [32].

We use a Lagrange interpolation filter due to explicit formulas to a construct fractional delay filter and flat-frequency response for low-frequencies. Combined with supersampled input audio signal, we can model fractional delay accurately. Variable delay can be easily modeled by using a different filter computed explicitly per audio sample. To compute an order N Lagrange filter, the traditional methods [57] 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 [23]. Many applications requiring variable fractional delay oversample the input with a high-order interpolator and use a low-order variable fractional delay interpolator [61] to avoid computing a high-order variable delay filter during run time. Wise and Bristow-Johnson [61] 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 12) to achieve 60 dB or more SNR for a low-order Lagrange interpolation filter, but it might be possible to use low oversampling to minimize artifacts in final audio output [45].

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 several different scenarios and scenes. The complexity of these scenes is similar to those used in current games with tens of thousands of triangles for audio processing. We test the performance of the system on a multi-core PC with NVIDIA GTX 480 GPU and use a single CPU core (2.8 GHz Pentium) for audio processing. We used some common benchmarks to evaluate the performance of our system (Figure 13). Results for static source and receiver positions are shown in Table 1. We also show the cost of conducting higher order recursion in Table 2.

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

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

- **Lehnert model**: This model increases the receiver radius for rays that travel farther, adjusting the radius as rays spread out [35].
  $r = \ell\sqrt{\frac{2\pi}{N}}$
- **NORMAL model**: Originally a method for predicting the number of rays needed based on scene volume [19], [63], this algorithm has been adapted as a receiver size model [62].
  $r = \sqrt[3]{\frac{15V}{2\pi N}}$
- **Xiangyang model**: This model accounts for the minimal sphere receiver size needed for detection and adjusts the radius based on scene volume [62].
  $r = \log_{10}(V)d\sqrt{\frac{4}{N}}$

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

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

TABLE 1: Performance in static scenes: *The top two represent simple indoor and outdoor scenes. The third one is a well known acoustic benchmark and the fourth one is the model of Sibenik Cathedral. The number of reflections (R) and edge diffraction (D) are given in the second column. The time spent in computing propagation paths (on GPU) is shown in the PT column and audio processing (on CPU) is shown in the AT column. The simulation begins with 50k visibility samples; we measure the performance after 50 frames.*

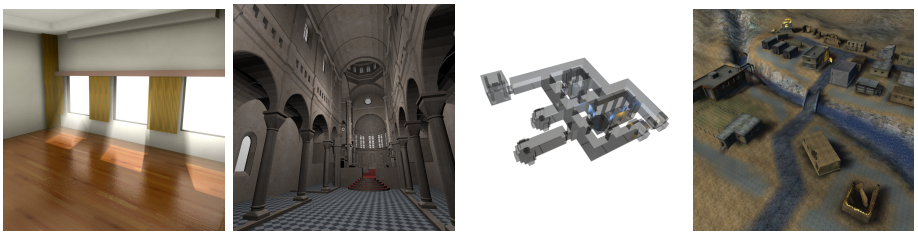
Model	1R	2R	3R	4R
Desert	30	41	53	57
Sibenik	51	90	153	226

TABLE 2: Performance per recursion: *Average performance (in ms) of our GPU-based path computation algorithm as a function of number of reflections performed. The Desert scene also includes edge diffraction. 50k visibility samples were used.*

7.2 Accuracy and Limitations

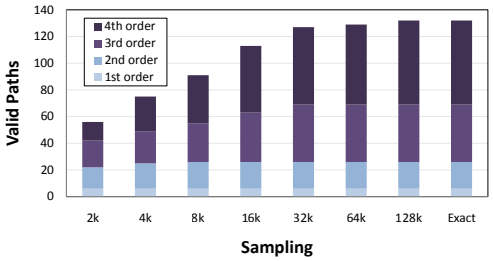
Overall, our approach is designed to exploit the computational power of GPUs to perform interactive visibility queries. The overall goal is accurate auralization, but our approach can result in the following errors:

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

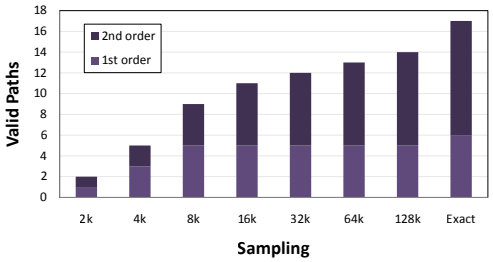


(a) (b) (c) (d)

Fig. 13: Benchmarks: *The benchmarks used to test the performance of our implementation: (a) Music hall model; (b) Sibenik cathedral; (c) Indoor scene; (d) desert scene. While the music hall scene is not often used for low order acoustic simulation, we selected it to show the animation sequence in Appendix A. Sibenik cathedral was selected as a very challenging visibility test case.*



(a) Music Hall



(b) Sibenik Cathedral

Fig. 14: Recursion path count: *These figures show the number of paths found for varying visibility rays. The receiver size is fixed at 1 meter. As visibility ray count increases, low triangle count scenes like the Music hall (a) are quickly saturated. However, in complex scenes like Sibenik cathedral (b), higher visibility ray counts are required to explore the scene.*

more accurate for the first few orders of reflections and diffraction. Figure 14 compares the found paths on two scenes of varying complexity.

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

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

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

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

The governing factor in the accuracy and performance of our approach is the number of ray samples that are cast in a single simulation frame. This directly impacts visibility accuracy and indirectly affects validation accuracy. As we use a higher number of visibility samples, errors are reduced (see Figure 14). This aligns well with the current technology trends as the performance of future GPUs will improve in terms of ray tracing throughput [1]. Another factor that governs the accuracy is the size of the triangles. Most GA methods are applicable to models where the size of the features or triangles is comparable (or larger) than the audible wavelengths. Moreover, as the size of the primitives increase, it improves the coherence of the multi-viewpoint ray casting algorithm and makes it possible to trace a higher number of ray samples per second.

7.3 Comparisons

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

Ray tracing algorithms: Previous ray-shooting based propagation algorithms [30], [58] trace each ray or ray packets independently to compute paths from the sources to the receiver. These methods model the receiver as an object of some size to determine when a discrete ray is close enough to the receiver to be considered a valid contribution path. This can lead to missed contributions, duplicate contributions (see Figure 3), or statistical errors [35]. Since

we can adjust the visibility sampling and detection sphere size, our method can achieve better performance than prior ray tracing methods.

Exact GA algorithms: Exact GA algorithms are based on beam tracing [33] and conservative ray-frustum tracing [12]. These methods can accurately compute all the specular reflection and edge diffraction paths. However, frustum tracing methods [12] can take a $6-8$ seconds on simple models composed of a few thousand triangles with three orders of reflections on a single core and beam tracing algorithms are almost an order of magnitude slower than frustum tracing.

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

Other systems: ODEON is a popular acoustics software which can compute specular reflections, diffuse reflections, and diffraction [15] 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 [15]. For diffraction, ODEON computes at most one diffraction path from a source to a receiver. The diffraction impulse response is computed [40] 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 [11] is another popular room acoustic software which performs specular and diffuse reflections using a combination of image source and ray tracing methods, along with randomized tail-corrected cone tracing [18]. It does not have support for diffraction computation. RAMSETE [26] is a GA based prototype acoustic system. It performs indoor and outdoor sound propagation using pyramid tracing [22]. It can perform specular reflections, diffuse reflections, and multiple orders of diffraction over free edges. The diffraction contribution for free edges is computed using the Kurze-Anderson [31] 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 [48], [36]. It applies image source method for specular reflections. RAVEN uses spatial hierarchies to render dynamic scenes with hundreds of triangles. Additionally, RAVEN supports a simplified form of diffraction tests. Another prototype system for real-time auralization is based on beam tracing [25], [56]. 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 either a static source or receiver position. RESound [52] is also a real-time auralization system. It is based on a combination of frustum tracing and ray tracing to handle specular reflections, diffuse reflections, and UTD diffraction in dynamic scenes.

7.4 Audio Processing

Our interpolation scheme presented in Section 6.2 produces smooth audio. It could be improved by interpolating image sources by predicting their new position based on their current positions and velocities [53]. Additionally, in cases where the number of image sources (or paths) is large, it is possible to apply clustering and perceptual acceleration [56], [37] for efficient audio processing. Currently, our audio processing step does not interpolate direction of an image source relative to the receiver but we encode it by computing delays and attenuation for left and right ears for 3D audio.

8 Conclusion and Future Work

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

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. Online analysis of the propagation cost coefficients may allow for improved guidance algorithms. A preprocess sampling pass could be used select more appropriate initial sampling and radius values for better guidance. Furthermore, we would like to perform perceptual evaluation of our system and perform user studies. Since the ray tracing algorithm can also be used to perform diffuse reflections, it may be possible to adapt our algorithms for rendering diffuse scattering effects. During tracing, some form of dynamic resampling may improve the accuracy of our algorithm. We also want to investigate the use of multi-view tracing with other software, such as NVIDIA's OptiX [2] engine or acoustic precomputation methods [3]. Finally, we would like to integrate our auralization system with other interactive applications and evaluate its performance.

References

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

2. http://www.nvidia.com/object/optix.html

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

[3] L. Antani, A. Chandak, L. Savioja, and D. Manocha. Interactive sound propagation using compact acoustic transfer operators. *ACM Transactions on Graphics (To appear)*. http://www.cs.unc.edu/ lakulish/Papers/2011-tog.pdf.

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

[5] L. Antani, A. Chankak, M. Taylor, and D. Manocha. Fast geometric sound propagation with finite edge diffraction. Technical Report TR10-011, University of North Carolina, Chapel Hill, 2010.

[6] J. Arenberg. Re: Ray/triangle intersection with barycentric coordinates. In E. Haines, editor, *Ray Tracing News*, volume 1. November 1988. http://tog.acm.org/resources/RTNews/html/rtnews5b.html#art3.

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

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

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

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

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

[12] A. Chandak, L. Antani, M. Taylor, and D. Manocha. Fastv: From-point visibility culling on complex models. *Computer Graphics Forum (Proc. of EGSR)*, 28(3):1237–1247, 2009.

[13] A. Chandak, L. Antani, M. Taylor, and D. Manocha. Fastv: From-point visibility culling on complex models. In *Eurographics Symposium on Rendering*, 2009.

[14] A. Chandak, C. Lauterbach, M. Taylor, Z. Ren, and D. Manocha. AD-Frustum: Adaptive Frustum Tracing for Interactive Sound Propagation. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1707–1722, Nov.-Dec. 2008.

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

[16] B. Cowan and B. Kapralos. Gpu-based real-time acoustical occlusion modeling. *Virtual Real.*, 14:183–196, September 2010.

[17] B. Cowan, Brent; Kapralos. Gpu-based acoustical diffraction modeling for complex virtual reality and gaming environments. In *Audio Engineering Society Conference: 41st International Conference: Audio for Games*, 2 2011.

[18] B.-I. L. Dalenbäck. 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, 1996.

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

[20] J. L. B. C. Diogo Alarcao, David Santos. An auralization system for real time room acoustics simulation. In *Proceedings of Tecniacustica 2009*, 2009.

[21] R. Duraiswami, D. N. Zotkin, and N. A. Gumerov. Fast evaluation of the room transfer function using multipole expansion. *IEEE Trans. Audio, Speech, Language Processing*, 15:565–576, 2007.

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

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

[24] T. Funkhouser, I. Carlbom, G. Elko, G. Pingali, M. Sondhi, and J. West. A beam tracing approach to acoustic modeling for interactive virtual environments. In *Proc. of ACM SIGGRAPH*, pages 21–32, 1998.

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

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

[27] M. Jedrzejewski and K. Marasek. Computation of room acoustics using programmable video hardware. In K. Wojciechowski, B. Smolka, H. Palus, R. Kozera, W. Skarbek, and L. Noakes, editors, *Computer Vision and Graphics*, volume 32 of *Computational Imaging and Vision*, pages 587–592. 2006.

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

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

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

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

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

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

[34] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha. Fast bvh construction on gpus. In *Proc. Eurographics '09*, 2009.

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

[36] T. Lentz, D. Schröder, M. Vorländer, and I. Assenmacher. Virtual reality system with integrated sound field simulation and reproduction. *EURASIP Journal on Advances in Singal Processing*, 2007:187–187, January 2007. Article ID 70540, 19 pages.

[37] T. Moeck, N. Bonneel, N. Tsingos, G. Drettakis, I. Viaud-Delmon, and D. Alloza. 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, 2007. ACM.

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

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

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

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

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

[43] N. Rber, U. Kaminski, and M. Masuch. Ray acoustics using computer graphics technology. In *Proceedings of the 10th International Conference on Digital Audio Effects (DAFx'07)*, pages 274–279, 2007.

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

[45] L. Savioja, T. Lokki, and J. Huopaniemi. Auralization applying the parametric room acoustic modeling technique - the diva auralization system. *ICAD*, 2002.

[46] C. Schissler and D. Manocha. Gsound: Interactive sound propagation and rendering for games. Technical report, University of North Carolina, 2011. http://gamma.cs.unc.edu/GSOUND.

[47] A. Schmitz, T. Rick, T. Karolski, T. Kuhlen, and L. Kobbelt. Efficient rasterization for outdoor radio wave propagation. *IEEE Transactions on Visualization and Computer Graphics*, 17:159–170, February 2011.

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

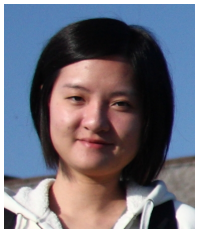
[49] S. Siltanen, T. Lokki, and L. Savioja. Frequency domain acoustic radiance transfer for real-time auralization. *Acta Acustica united with Acustica*, 95:106–117(12), 2009.

[50] P. Svensson. The early history of ray tracing in room acoustics. In P. Svensson, editor, *Reflections on sound: In honour of Professor*

Emeritus Asbjørn Krokstad. Norwegian University of Science and Technology, 2008.
[51] U. P. Svensson, R. I. Fred, and J. Vanderkooy. An analytic secondary source model of edge diffraction impulse responses . *Acoustical Society of America Journal*, 106:2331–2344, Nov. 1999.
[52] M. Taylor, A. Chandak, L. Antani, and D. Manocha. Resound: interactive sound rendering for dynamic virtual environments. In *MM '09: Proceedings of the seventeen ACM international conference on Multimedia*, pages 271–280. ACM, 2009.
[53] N. Tsingos. A versatile software architecture for virtual audio simulations. In *International Conference on Auditory Display (ICAD)*, Espoo, Finland, 2001.
[54] N. Tsingos. Pre-computing geometry-based reverberation effects for games. *35th AES Conference on Audio for Games*, 2009.
[55] N. Tsingos, T. Funkhouser, A. Ngan, and I. Carlbom. Modeling acoustics in virtual environments using the uniform theory of diffraction. In *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 545–552, 2001.
[56] N. Tsingos, E. Gallo, and G. Drettakis. Perceptual audio rendering of complex virtual environments. *ACM Trans. Graph.*, 23(3):249–258, 2004.
[57] V. Välimäki. *Discrete-Time Modeling of Acoustic Tubes Using Fractional Delay Filters*. PhD thesis, Helsinki University of Technology, 1995.
[58] M. Vorländer. 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, 1989.
[59] M. Wand and W. Straßer. Multi-resolution sound rendering. In *SPBG'04 Symposium on Point - Based Graphics 2004*, pages 3–11, 2004.
[60] E. Wenzel, J. Miller, and J. Abel. A software-based system for interactive spatial sound synthesis. In *International Conference on Auditory Display (ICAD)*, Atlanta, GA, April 2000.
[61] D. K. Wise and R. Bristow-Johnson. Performance of Low-Order Polynomial Interpolators in the Presence of Oversampled Input. In *Audio Engineering Society Convention 107*, 9 1999.
[62] Z. Xiangyang, C. Ke'an, and S. Jincai. On the accuracy of the ray-tracing algorithms based on various sound receiver models. *Applied Acoustics*, 64(4):433 – 441, 2003.
[63] L. N. Yang and B. M. Shield. Development of a ray tracing computer model for the prediction of the sound field in long enclosures. *Journal of Sound and Vibration*, 229(1):133 – 146, 2000.
[64] K. Zhou, Q. Hou, R. Wang, and B. Guo. Real-time kd-tree construction on graphics hardware. In *Proc. SIGGRAPH Asia*, 2008.

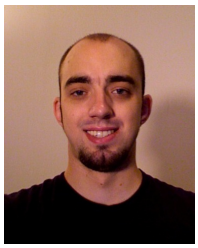
Micah Taylor Micah Taylor received a BS in computer science from Rose-Hulman Institute of Technology in 2004. He is currently a Ph.D. student at the University of North Carolina at Chapel Hill. He has worked as a research intern at Dolby Laboratories. His research interests are interactive sound propagation and real-time ray tracing.

Anish Chandak Anish Chandak received B.Tech. in Computer Science and Engineering (2006) from the Indian Institute of Technology (IIT) Bombay, India. He received M.Sc. and Ph.D. in Computer Science from the University of North Carolina at Chapel Hill in 2010 and 2011, respectively. His research interests include acoustics (sound synthesis, sound propagation, and 3D audio rendering) and computer graphics (ray tracing, visibility, and GPGPU).



Qi Mo Qi Mo received Bachelor's degree in Computer Science (2004) from Nanjing University and Master's degree (2006) from the University of Iowa. She is currently a Ph.D. student in the Computer Science Department at the University of North Carolina at Chapel Hill. She has worked as a research intern at NVIDIA. Her research interests include real-time sound propagation, ray tracing and collision detection.

Christian Lauterbach Christian Lauterbach is currently working at Google. He received a PhD in Computer Science from the University of North Carolina at Chapel Hill under the advice of Dinesh Manocha, and a Diploma in Computer Science from the University of Bremen, Germany.



Carl Schissler Carl Schissler is a Computer Science B.S. graduate of the University of North Carolina at Chapel Hill and professional live sound and recording engineer with interests in real-time physics, sound simulation and digital signal processing. He has developed novel techniques for fast sound propagation and presented these in a paper at the 41st AES conference for game audio. Carl is currently developing a commercial middleware library implementing these algorithms called GSound.

Dinesh Manocha Dinesh Manocha is currently the Phi Delta Theta/Mason Distinguished Professor of Computer Science at the University of North Carolina at Chapel Hill. He received his Ph.D. in Computer Science at the University of California at Berkeley 1992. He has received Junior Faculty Award, Alfred P. Sloan Fellowship, NSF Career Award, Office of Naval Research Young Investigator Award, Honda Research Initiation Award, Hettleman Prize for Scholarly Achievement. Along with his students, Manocha has also received 12 best paper & panel awards at the leading conferences on graphics, geometric modeling, visualization, multimedia and high-performance computing. He is an ACM and AAAS Fellow and received Distinguished Alumni Award from IIT Delhi. Manocha has published more than 300 papers in the leading conferences and journals on computer graphics, geometric computing, robotics, and scientific computing and supervised 19 Ph.D. dissertations.