Interactive Sound Rendering in Complex and Dynamic Scenes using Frustum Tracing

Christian Lauterbach, Anish Chandak, and Dinesh Manocha, *Member, IEEE*

Abstract—We present a new approach for real-time sound rendering in complex, virtual scenes with dynamic sources and objects. Our approach combines the efficiency of interactive ray tracing with the accuracy of tracing a volumetric representation. We use a four-sided convex frustum and perform clipping and intersection tests using ray packet tracing. A simple and efficient formulation is used to compute secondary frusta and perform hierarchical traversal. We demonstrate the performance of our algorithm in an interactive system for complex environments and architectural models with tens or hundreds of thousands of triangles. Our algorithm can perform real-time simulation and rendering on a high-end PC.

Index Terms—Acoustic propagation,Interactive systems

1 Introduction

Traditionally, the focus in interactive visualization has been on high-quality, realistic visual rendering of complex datasets. These developments are supported by high growth rates and programmability of current graphics hardware as well as advances in rendering acceleration algorithms. However, at the same time it is important to develop interactive algorithms for sound or auditory rendering. In a multi-sensory visualization system, spatial sound can be combined with visual rendering to provide a more immersive experience for many applications [28]. These can be used for the development of an auditory display [38, 16, 31] to convey intuitive spatial cues directly and therefore can result in better understanding and evaluation of complex datasets.

In this paper we address the problem of interactive sound rendering and visualization in complex and dynamic environments.. Some of the driving applications include acoustic design of architectural models or outdoor scenes, walk-throughs of a virtual prototype of a large CAD model with sounds of machine parts or moving people, virtual environments with multiple avatars, or even visualization of multi-dimensional datasets [28]etc. The sound rendering algorithms take into account the knowledge of sound sources, listener locations, 3D models of the environments, and material absorption data to generate realistic and spatialized sound effects.

Over the last few decades, the problem of fast visual rendering of complex datasets has received considerable attention in computer graphics and visualization literature. Current algorithms and systems are able to handle complex datasets composed of millions of primitives at interactive rates on commodity hardware. In contrast, prior sound rendering are limited to relatively simple models and cannot handle complex or dynamic datasets at interactive rates. The main challenge in sound rendering is to compute the reflection paths from the sound sources to the listeners at interactive rates. Prior approaches for complex environments have been based on geometric methods that use either ray or beam tracing methods to explicitly follow the paths. However, ray tracing methods are prone to inaccuracies due to sampling or aliasing errors, and beam tracing methods involve considerable preprocessing and are limited to static, densely-occluded environments. As a result, current interactive applications are limited to using sound sources that are associated with a static, precomputed effect.

• *The authors are with the Department of Computer Science, Campus Box 3175, Sitterson Hall, University of North Carolina-Chapel Hill, Chapel Hill, NC 27599. E-mail: {cl, achandak, dm}@cs.unc.edu.*

Manuscript received 31 March 2007; accepted 1 August 2007; posted online 27 October 2007. Published 14 September 2007.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

Main Results: We present an interactive algorithm for sound rendering using frustum tracing. Our approach uses a simple volumetric representation based on a four-sided convex frustum, for which describe efficient algorithms to perform hierarchy traversal, intersection and specular reflection and transmission interactions at the geometric primitives. Unlike beam tracing and pyramid tracing algorithms, we perform approximate clipping by using a subdivision into sub-frusta. As a result our rendering algorithm reduces to tracing ray packets and maps well to the SIMD instructions available on current CPUs. We support dynamic scenes by using bounding volume hierarchies (BVHs) to accelerate the computations on complex models. Overall, our approach combines the efficiency of interactive ray tracing with the accuracy of tracing a volumetric representation.

We have implemented our algorithm and have used it for interactive sound rendering in complex environments composed of tens or hundreds of thousands triangles and dynamically moving objects. The performance of our system varies with the complexity of the environments, especially as a function of the number of reflections. In practice, our approach can trace enough frusta to simulate sound on a current high-end PC at interactive rates with up to 7 reflections.

As compared to prior geometric approaches for sound rendering, our approach offers the following advantages:

- **Generality:** No special or logical scene representation is necessary and our algorithm is able to handle all polygonal models.
- **Efficiency:** Our algorithm scales with the complexity of the scenes as a logarithmic function of the model size (although a linear complexity update step is needed whenever geometry moves). Most of the benefits of ray packet tracing are directly applicable, including SIMD implementation and trivial parallelization on multi-core processors.
- **Dynamic, complex scenes:** We can handle all kind of dynamic scenes and make no assumptions on the motion of sound sources, listener or objects in the scene.
- **Integrated visual and sound rendering:** We use a BVH to perform fast intersection tests between ray packets and the primitives. The same hierarchy can be used for ray tracing for visual rendering and frustum tracing for sound rendering.

Organization: The rest of the paper is organized in the following manner: we give a brief overview of prior work on sound propagation in Section 2. Section 3 presents our frustum tracing algorithm and shows how to use the algorithm to compute the reflection paths from the sound sources to the listeners. We describe our implementation in Section 4 and demonstrate its performance on different models in Section 5. We analyze the performance in Section 6 and highlight a few limitations of our approach.

2 PREVIOUS WORK

There has been considerable work on sound generation and propagation in computational acoustics, computer graphics, computational geometry and related areas for more than four decades [5, 7, 14]. These include physically-based sound synthesis algorithms [19, 32], numerical and geometric methods for sound propagation and acceleration techniques. In this section we give a brief overview of sound propagation algorithms.

Numerical methods: Numerical solutions [24] attempt to accurately model the propagation of sound waves by numerically solving the wave equation. These methods are general and highly accurate [33]. However, they can be very compute and storage intensive [41]. Current approaches are too slow for interactive sound propagation in complex environments and are mainly limited to simple scenes.

Geometric methods: These algorithms model the propagation of sound based on rectilinear propagation of waves and can accurately model the early reflections. Most of these methods are closely related to parallel techniques in global illumination, and many advances in either field can also be applied to the other. The earliest of these approaches were particle and ray based [23, 25] and simulated the propagation paths by stochastically sampling them using rays. Based on recent advances in interactive ray tracing, these methods are also applicable to dynamic scenes [45, 26]. Approaches using discrete particle representations called *phonons* or *sonels* [3, 9, 22] have been developed in the last few years. These methods look very promising but are currently limited to simple scenes. Moreover, particle and ray-based algorithms are susceptible to aliasing errors and may need a very high density of samples to overcome those problems.

The *image source* algorithms create virtual sources for specular reflection from the scene geometry and can be combined with diffuse reflections and diffractions [4, 8]. They accurately compute the propagation paths from the source to the listener, but the number of virtual sources can increase exponentially for complex scenes [4]. This makes these techniques suitable only for static scenes.

The third type of geometric methods is based on *beam tracing*, which recursively traces pyramidal polyhedra from the source to the listener [18, 10, 11]. In their seminal work, Funkhouser et al. [12, 13] showed how beam tracing methods can be used for sound propagation at interactive rates in complex virtual environments. Some algorithms have been proposed to use beam tracing on moving sources [2, 15]. However, current algorithms take large pre-processing time and are not directly applicable to dynamic scenes with moving objects.

Interactive Sound Propagation: Many other methods have been presented for rendering of room acoustics [29, 36, 42] or have been integrated with VR systems [30]. Joslin and Thalmann [21] present a technique to reduce the number of facets in order to accelerate the reflection computations in sound rendering. A point-based algorithm for multi-resolution sound rendering has been presented for scenes with a large number of emitters [46]. Doel et al. [43] present an algorithm for interactive simulation of complex auditory scenes using model-pruning techniques based on human auditory perception. Our approach is complementary to many of these algorithms and can be combined to further improve the performance.

3 FRUSTUM TRACING

In this section we present our algorithm for interactive sound propagation in complex and dynamic scenes. Our approach is built on recent advances in interactive ray tracing, including packet traversal algorithms [44] and dynamic scenes [45, 26].

3.1 Frustum Representation

As discussed above, ray tracing algorithms for sound propagation suffer from noise and aliasing problems [27], both spatially and temporally. In order to avoid these sampling issues, we trace a simple convex polyhedron instead of infinitesimal rays. Specifically, we perform *frustum tracing*[1], which is similar to beam tracing and pyramid tracing. We use a simple convex frustum so that we can perform fast intersection tests with the nodes of the hierarchy and the primitives. Unlike beam tracing algorithms, we perform approximate clipping using ray packets. Overall, our representation combines some of the speed advantages of ray packet tracing with the benefits of volumetric formulations.

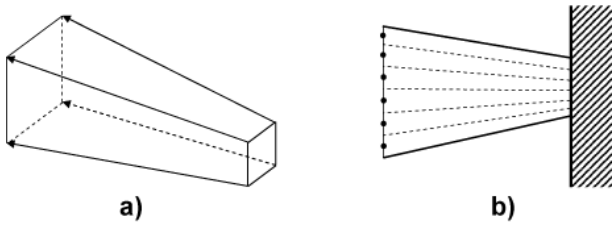

Fig. 1. **Frustum-based packet:** *The frustum primitive used in our algorithm. a) The frustum is defined by the four side faces and the front face, or equivalently by the boundary rays on the sides where the faces intersect. b) the frustum is uniformly subdivided into sub-frusta defined by their center sample rays (dots), depending on a sampling factor.*

We use a convex four-sided frustum, i.e. a pyramid with a quadrilateral base (see Fig. 1(a)) that is defined by its four side faces and one front face. Equivalently, the frustum can be represented as the convex combination of four corner rays defining the frustum. At a broad level, the main difference between frustum and beam tracing is how we keep track of intersections with the primitive and the scene. Beam tracing performs exact clipping with each primitive in the scene and therefore needs to maintain a full list of clipped edges or faces of the beam. We avoid these relatively expensive operations by subdividing the frustum uniformly into smaller sub-frusta to perform discrete clipping, and only keep track of intersections at the level of those sub-frusta (see 1(b)). Moreover, each sub-frustum is represented by a sample ray, and a sub-frusta is considered to intersect a primitive only if its sample ray hits the primitive. Essentially, this can be interpreted as a discrete version of a clipping algorithm and can introduce some errors in our propagation algorithm.

The difference between the frustum and beam tracing process is also highlighted in Fig. 2. We show the intersection of the beam (left) and frusta (right) with three primitives and the resulting secondary beams and frusta computed for reflection and transmission. Note that since the intersection is determined by the location of the sample ray, the frustum tracing algorithm in this example will underestimate the size of secondary beams at the primitive on the left. The amount of error introduced depends on the sampling rate, i.e. the rate of subdivision of the frustum.

Benefits: Our formulation of the frustum and the clipping algorithm allows a faster and more general algorithm for propagation. We use the main frustum as a placeholder for all the enclosed sub-frusta during hierarchy traversal or intersection computations. As a result we are able to achieve very efficient and fast traversal using our representation in both static and dynamic scenes. In addition, we organize our sample rays in ray packets similar to those used in interactive ray tracing, and exploit the uniform subdivision of frusta for faster primitive intersection computations. Finally, we defer constructing the actual sample ray computation until the sub-frusta are actually needed, i.e. if the whole frustum does not fully hit a primitive. This reduces the set-up cost, especially for very small beams.

3.2 Frustum Tracing

The goal of frustum tracing is to identify the primitives (i.e. triangles) that intersect the frustum and then to construct new secondary beams that represent specular reflection and transmission of sound. This involves traversing the scene hierarchy, computing the intersection with primitives and then constructing secondary frusta. We present algorithms for each of these computations.

Construction of secondary frusta: Whenever a frustum hits a primitive, we construct secondary frusta for transmission and specular

[1] We use the term *frustum tracing* in a different sense than earlier work on radio propagation presented in [40], which is very similar to beam tracing.

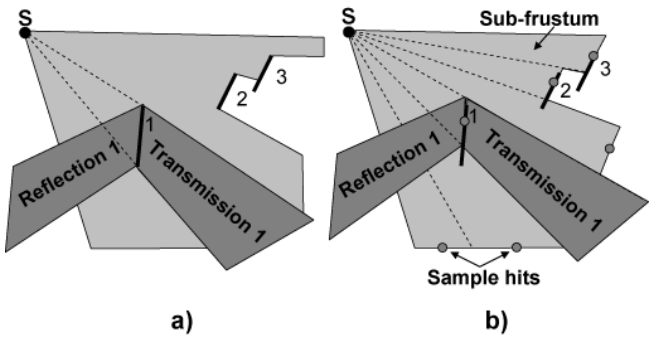

Fig. 2. **Beam vs. frustum tracing:** *Our approach compared to beam tracing for a simple example.(Left): beam tracing. (Right): frustum tracing. The discrete sampling in our frustum based approach underestimates the size of the exact reflection and transmission frustum for primitive 1 and overestimates the size for primitives 2 and 3.*

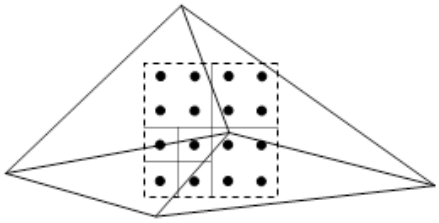

Fig. 3. **Constructing secondary frusta:** *We compute reflected and transmitted frusta efficiently by grouping sub-frusta that hit the same primitive together in a single secondary frustum instead of having to trace each of them individually. Using a hierarchical process, we combine groups of four sub-frusta together as long as they hit the same primitive.*

reflection. If the entire frustum hits one primitive, the construction of the secondary frusta is simple and can be accomplished by just using the four corner rays. For the general case, when different sub-frusta hit different primitives, multiple secondary frusta have to be generated. A naïve solution would be to generate reflection and transmission for each single sub-frustum defined by a sample ray. However, this could result in an extremely high number of additional frusta, and the complexity of the algorithm will grow as an exponential function of the number of reflections. To avoid this, we combine those sub-frusta that hit the same primitive by hierarchically comparing four neighboring samples and treating them as one larger frustum (see Fig. 3). This can be seen as a quad-tree structure, although we do not compute the tree explicitly. If the samples hit neighboring primitives that have the same material and normal, we combine those primitives in the same way to avoid splitting too many sub-frusta. This is especially useful when rectangles are represented by two triangles, which is a common case in architectural models. In practice, we have found that our approach yields a good compromise between the time taken to find optimal groups of sub-frusta and the number of secondary frusta needed. We also exploit the fact that the combined frustum exactly represents the sub-frusta, and there is no loss of accuracy due to this hierarchical grouping. If the primitives in the scene are over-tessellated, we could use simplification algorithms to decrease their size [21]. This can introduce some additional error in our propagation algorithm, but big triangles in the scene would result in fewer secondary sub-frusta.

Hierarchy traversal: We use a bounding volume hierarchy (BVH) as our choice of scene hierarchy, as it has been shown to work well for general dynamic scenes. However, our algorithm can also be adapted to be used with kd-trees or other hierarchies. The main operation for traversal of the BVH is checking for intersection with a BV, most commonly an axis-aligned bounding box (AABB). As described by Reshetov et al. [35], a frustum can be tested for overlap with an AABB quickly. If the frustum does not intersect the AABB node, the entire subtree rooted at that node can be culled. Otherwise the children of the node are tested in a recursive manner. However, this traversal method can result in traversing too many nodes, because traversal cannot stop until the first hit between the scene geometry and the frustum has been computed. Interactive ray tracing algorithms using BVHs also track which rays in the packet are still currently active (i.e. hit the current node) at any point during traversal [45, 26]. Since we want to avoid performing intersection tests with the frustum's sample rays as long as possible, we also keep track of the farthest intersection depth found so far to rule out intersecting nodes that cannot possibly contribute.

Efficient primitive intersection: We assume that the models are triangulated. The main goal for intersection with triangles is to minimize the number of ray-triangle intersections, as they can be more expensive than the traversal steps. Most importantly we want to avoid performing any ray intersections at all if we can determine that the entire frustum hits the primitive, which can happen many times. Consider Fig. 4, which shows the different configurations that can arise when intersecting a frustum with a primitive. Case 1 shows that the frustum fully misses the primitives (i.e. no overlap at all); therefore, we can skip that intersection right away. Case 2 shows that the frustum fully hits the primitives, which means we can construct secondary frusta right away without having to consider subdividing the frustum, unless a closer hit is found later on. In cases 3 and 4, the frustum partially overlaps the primitive or contains the primitive and we have to consider the individual sub-frusta.

We test for these four cases by using a Plücker coordinate representation for the triangle edges and frustum rays [37], which gives us a way to test the orientation of any ray relative to an edge. Given a consistent orientation of edges (clockwise or counter-clockwise), we can test for intersection if all the edge orientations have the same sign. When testing the corner rays of the frustum, which can be performed in parallel using SIMD instructions, we check for Case 1 and Case 2 simply by testing whether all the corner rays are inside the triangle (Case 2) or fully outside one or more edges (Case 1). Note that the latter test is conservative and may conclude that the frusta are intersecting the triangle, even if they are not. These intersections will eventually be culled in our handling of Cases 3 and 4.

If no early culling is possible, we then perform a ray-triangle intersection using the actual sample rays. As the number of rays that actually intersect the triangle may be small compared to the number of sample rays representing all the sub-frusta, we first compute the subset of potential intersections efficiently. Since the sample rays are uniformly distributed in the frustum space, we compute bounds on the projected triangle in that space and only test those samples that fall within those bounds. In order to perform these computations, we clip the triangle to the bounds of the frustum by projecting the triangle to one of the coordinate planes and use a line clipping algorithm against the frustum's intersection with the plane. Finally, when looking at the clipped polygon's vertices, we can compute their bounding box in frustum parameter space (see Fig 5). The actual triangle intersection is only performed for the sample rays that fall within the boundary of the clipped triangle, and can easily be performed by using the indices. Note that this can also be reduced to a rasterization problem: given a triangle that is projected into the far plane of the frustum, we want to find the sub-frusta it covers. Therefore, we can use other ways to evaluate this intersection. By using a higher set-up cost, the triangle could be projected and processed with a scan-line rendering algorithm, intersecting with the respective sample ray for each covered sub-frustum. Another interesting approach would be to use a modified A-buffer [6] for computing the sub-frusta covered by the triangle through lookup masks, at the cost of some precision.

Handling non-specular interactions: As described above, specular reflections and transmissions can be handled directly. Although we have not implemented this, our frustum tracing approach can also use the diffraction formulation described by Funkhouser et al. [13] based on the uniform theory of diffraction. For diffuse scattering the frustum tracing approach could be adapted to also generate secondary frusta on a hemisphere around the hit point. However, this could increase the

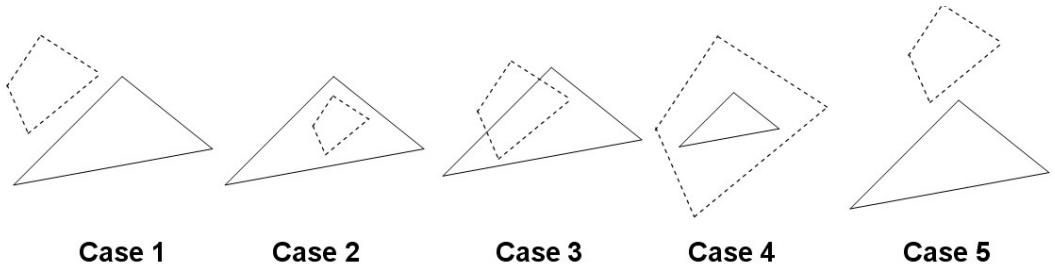

Fig. 4. **Primitive intersection:** *Five different cases can occur when intersecting a frustum with a triangle. From left to right: Frustum misses completely, frustum is contained, frustum intersects partially, frustum contains triangle. The last case shows a situation where the frustum is clearly outside the triangle, but is not detected by the edge based test since it is not fully on one side of any edge. This case is handled as intersecting, but is culled later on during the clipping test.*

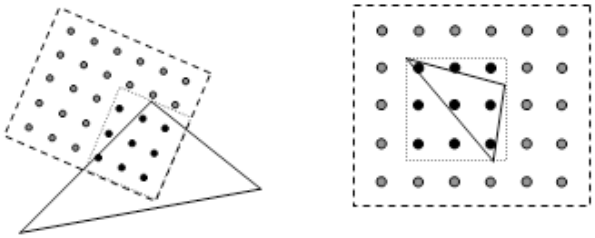

Fig. 5. **Packet-triangle intersection:** *Our novel intersection algorithm quickly computes the potential ray intersections in frustum space by clipping the triangle to the frustum's edges in 2-D, then finding the rectangular bounds of the clipped point in frustum space. The bounds can then be used to effectively limit the number of actual sample rays that have to be tested.*

branching factor per interaction dramatically and therefore have a high impact on performance.

3.3 Sampling and Aliasing

Our algorithm uses a discrete approximation of the exact secondary beams that would be computed by using an exact clipping algorithm. As a result the reflections obtained by our method can suffer from aliasing artifacts, especially along object boundaries. As shown in Fig. 2, reflected frusta often subtend areas that are outside of the primitive or do not cover all of the area. This is due to the fact that our tracing algorithm assumes that a sub-frustum hits the primitive in its full projected area if its sample ray hits the primitive. This can result in other possible effects such as missing paths, e.g. a small hole in the object might be missed due to our sampling density. Fortunately, these artifacts only result in some missed contribution paths from the reflections. Moreover, in a dynamic environment these effects would be far less obvious to the listener as compared to the noise artifacts that can arise due to stochastic sampling in ray tracing methods. Note that our algorithm will also avoid creating holes or overlaps in the reflections field during the computation of reflected or transmitted frusta. These holes or overlaps can have a far larger contribution of error since they tend to be more apparent in an interactive application because of abrupt changes in the contribution. An interesting aspect of our approach is that having small geometric objects or primitives (i.e. a statue) in the scene will not result in a very high number of small secondary frusta. Instead, the number of reflections is bounded by the sampling density in the packet. These very small frusta would be computed by an exact clipping algorithm, though they have very little or no contribution.

One of the main challenges is to compute an appropriate sampling rate (i.e. the number of rays in the frustum). Ideally, the sampling rate could be chosen by taking the highest detail in the scene and setting the frequency so that detail could be reconstructed. Similar to rasterization algorithms, performing this computation in a view-independent manner is almost infeasible due to its high complexity and can lead to very conservative bounds. As a result we use realistic sampling rates and allow some error. There are several approaches for choosing the sampling rate in this context: first, a good way of choosing the subdivision is to select the number of rays depending on the angular spread of the packet. For example, a very narrow frustum will likely need a lower sampling density than a wide frustum. Since the actual rays are not constructed until a sufficiently small primitive is encountered, it is also possible to select the sampling rate relative to the local geometric complexity in order to avoid under-sampling. One way to measure local complexity, for instance, would be to use the current depth of the subtree in the BVH. Finally, the sampling rate can also be made dependent on the energy carried by a frustum or the number of reflections before reaching the current position. This is a useful approximation as the actual contribution will likely decrease, and we can lower the sampling rate after a few reflections.

4 Implementation

We now describe the overall sound rendering system that uses our sound propagation algorithm. Our system is designed to be fully real-time and dynamic. We allow movement of the listener, the sound sources and the geometric primitives in the scene. The sound propagation algorithm is run as an asynchronous thread from the rest of the system.

The sound propagation simulation starts out from each point sound source and constructs frusta from that origin that span the whole sphere of directions around it according to a predefined subdivision factor. Each of the frusta is traced through the scene, and secondary frusta are constructed based on the algorithm described in Section 3. There is a user-specified maximum reflection order that limits the number of total frusta that need to be computed. Attenuation and other wavelength-dependent effects are applied according to the material properties per frequency band. Since we regenerate the sound contributions at each frame, we do not save the full beam tree of the simulation, but just the those that actually contain the listener.

Handling dynamic scenes: The choice of a BVH as an acceleration structure allows us to update the hierarchy efficiently in linear time if the scene geometry is animated, or rebuild it if a heuristic determines that culling efficiency of the hierarchy is low [26]. As the BVH is a general structure, our algorithm can handle any kind of scene including unstructured 'polygon soup' and models with low occlusion. Furthermore, we can use lazy techniques to rebuild the nodes of a hierarchy in a top-down manner.

Auralization: So far we have not described how the actual sound output is generated from the simulation algorithm described in the previous section, i.e. the auralization process (we refer the reader to a more detailed overview such as [14] for an introduction). As mentioned above, the simulation is performed asynchronously to the rendering and auralization, so we have a dedicated rendering thread and one or more simulation threads. During the simulation, we do not store the actual frusta, but test each frustum on whether the listener's position is contained in it. If so, we store the sound information such as source, delay and power for all bands in a temporary buffer. The rendering thread reloads this buffer at regular intervals and computes the contribution of each source as an impulse response function (IRF)

for each band and channel. Conceptually, each contributing frustum represents a virtual source located at the apex of the frusta such as in image source methods. Note that this approach can therefore update the sound more often even if the simulation itself is only updated infrequently, which reduces the impact of listener movement.

Furthermore, to incorporate frequency dependent effects, each source's sound signal is decomposed into 10 frequency bands at $20, 40, 80, 160, 320, 640, 1280, 2560, 5120, 10240$ and 20480 Hz and processed for two channels. For each channel the band-passed signal is convolved with the impulse response for that band and the channel. The convolved signals are then added up and played at the corresponding channel. We also have provision for binaural hearing and we use Head Related Transfer Functions (HRTFs) from a public-domain HRTF database [1]. The sound pipeline is set up using the FMOD Ex sound API. We currently perform all convolutions in software in the rendering thread, but it would be possible to do this in dedicated sound hardware using DSPs as well.

Implementation details: Our ray packet tracing implementation utilizes current CPUs' SIMD instructions that allow small-scale vector operations on 4 operands in parallel. In the context of packet tracing, this allows us to perform intersections of multiple rays against a node of the hierarchy or against a geometric primitive in parallel. In our case this is especially efficient for all intersection tests involving the corner rays as we use exactly four rays to represent a frustum. Therefore most operations involving the frustum are implemented in that manner. The frustum-box culling test used during hierarchy traversal is also implemented very efficiently using SIMD instructions [35]. Finally, since all the frusta can be traced in parallel, performing the simulation using multiple threads on a multi-core processor is rather simple and can be easily scaled to multi-processor machines.

5 Results

We now present results of using frustum tracing in our system on several scenes. All benchmarks were run on an Intel Core 2 Duo system at 3.0 GHz with a total of 4 cores. Our sound simulation runs asynchronously to the rendering thread and can be executed in parallel on the other three threads to exploit parallelism. As future CPUs will offer more cores, the performance of our sound propagation algorithm can therefore improve accordingly. Results are shown both for using just one thread and using all three threads.

We tested our system on several different environments and conditions (see Fig. 6). Our main performance is summarized in table 1 and shows that we can handle all of the benchmark models at interactive rates on our test system. The theater model is an architectural scene that is very open and therefore would be very challenging for beam tracing approaches. Even with 7 number of reflections per frustum, we can perform our simulation in less than one second with dynamic geometric primitives and sound sources. The Quake model was chosen as a typical example of a game-like environment and features densely-occluded portions as well as open parts. Some dynamic geometric objects and moving sound sources are also included in our benchmark. We also tested a more complex, static scene with 190K triangles with just one moving sound source.

The results in table 1 show that even though performance as measured by frusta per second decreases with increasing number of primitives, the decrease is still sub-linear. This is due to the logarithmic scaling of ray packet tracing methods. We recompute the BVH whenever the geometric objects in the scene move. Even though the time complexity of updating a BVH is linear in the number of primitives, the total time needed for updating a BVH is still negligible compared to the simulation time, as shown in table 2. Moreover, the BVH update can easily be parallelized using multiple threads between the simulation runs.

A key measure in our algorithm is the number of sample rays that are used per frustum. It can have a significant impact on the performance. Figure 7 shows the overall simulation performance as well as the total number of frusta used in our benchmark models when changing the sampling rate. The graph shows that the scaling is logarithmic, which is due to the ray-independent frustum traversal as well as

Model	Triangles	Construction	Update
Theater	9094	319 ms	2 ms
Quake	11821	53 ms	1 ms
Cathedral	196344	1615 ms	26 ms

Table 2. **Construction and maintenance cost:** *Our results show that for all the models maintaining or updating the BVH hierarchy adds a negligible cost to the overall simulation. Note that construction only needs to be performed once and then the hierarchy is maintained through updates.*

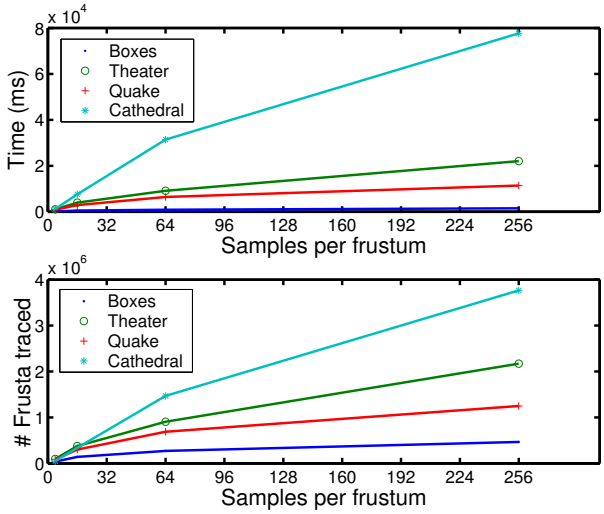

Fig. 7. **Sampling rates:** *The graphs show the impact of increasing the sampling rate per frustum on both the simulation times as well as number of frusta generated (all simulations are performed for 7 reflections.) In addition to the benchmark scenes used in Table 1, the 'Boxes' scene is a simple environment of two boxes connected by a small opening. Due to our frustum traversal algorithm, efficient triangle intersection and secondary frustum construction, increasing the sampling rate only causes logarithmic growth in the simulation time and number of frusta generated. This suggests that changing the frusta sampling rate can be an efficient method to control the accuracy of our simulation.*

our merging algorithm for constructing secondary frusta. This scaling makes the sampling rate a good parameter for trading off quality and runtime performance, depending on the requirements on the simulation.

6 Analysis and Comparison

We now analyze the performance of our algorithm and discuss some of its limitations. As discussed in section 3 our approach introduces errors due to discrete clipping as compared to beam tracing. We have found that the artifacts created through aliasing are usually hardly noticeable except in contrived situations, and they are far less obtrusive than temporal aliasing that arises in ray tracing algorithms based on stochastic approaches. Note that the sample location in the sub-frusta does not need to be the center, so the aliasing due to sub-sampling could be ameliorated by stochastic sampling of the locations, e.g. by jittering. However, this may introduce temporal aliasing in animated scenes as stochastic sampling may change simulation results noticeably over time. It is possible that Quasi-Monte Carlo sampling could eliminate these problems.

Another source of potential errors stems from the construction of secondary frusta: since the reflected or transmitted frustum is constructed from the corner rays of the sub-frustum, the base surface of the new frustum can significantly exceed the area of the primitive if the incoming frustum comes from a grazing angle and the sample rays hits close to he boundary of the object.

Another limitation of the frustum-based approach are that we assume surfaces are locally flat, and our algorithm may not be able to

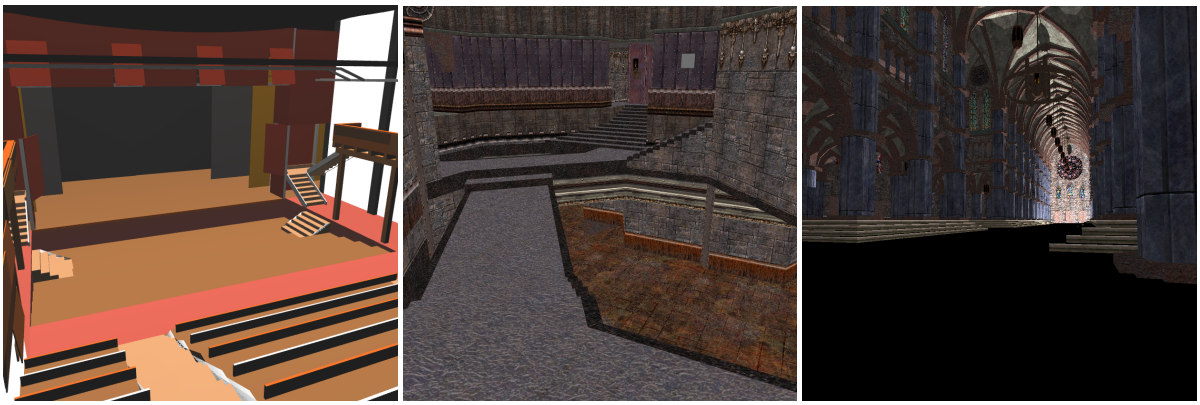

Fig. 6. **Benchmark scenarios**: *We achieve interactive sound propagation performance on several benchmark models ranging from 9k to 235k triangles while simulating up to 7 reflections. From left to right: Theater (9k), Quake (12k), Cathedral (196k).*

Model	Size (triangles)	Dynamic Objects			Simulation results		Simulation update time (avg.)		Frusta/second
		Listener	Source	Geometric objects	Reflections	Frusta	1 thread	3 threads	1 thread
Theater	9094	D	D	D	6	132k	754 ms	276 ms	175k-
Quake	11821	D	D (x3)	D	5	157k	861 ms	290 ms	182k
Cathedral	196344	D	D	-	5	60k	1607 ms	550 ms	37k

Table 1. **Results:** *This table highlights the performance of our system on different benchmarks. The "D" indicates that listener, source or the scene objects are dynamic. Note that the frustum tracing performance does scale logarithmically with scene complexity and linearly with the number of threads. Please see the video for demonstration of the benchmark scenes.*

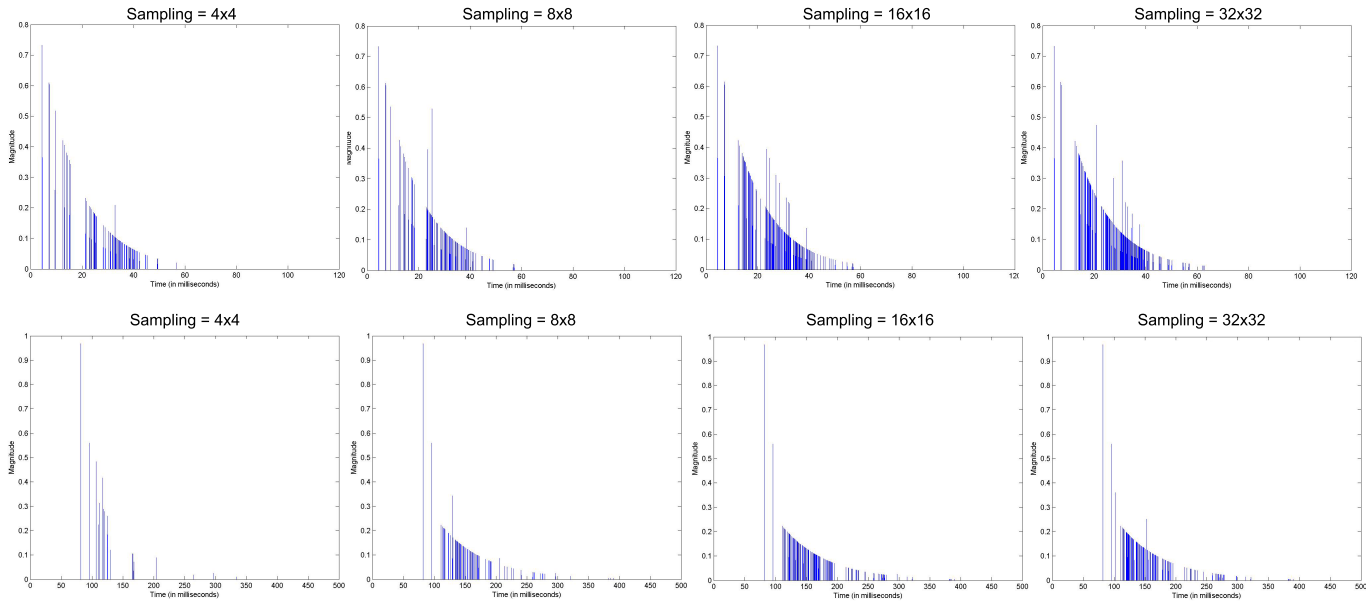

Fig. 8. **Impulse Response (IR) vs Sampling Resolution**: The above picture shows IRs generated from our frustum-tracing approach for a simple scene of two connected boxes (top) and the Theater scene (bottom), with reflection order = 4 and varying frustum sampling resolution $\{4 \times 4, 8 \times 8, 16 \times 16, 32 \times 32\}$. Notice that the sampling resolution of 4x4 misses some contributions compared to higher ones, but captures most of the detail correctly. As the sampling resolution increases, the accuracy of our method approaches that of the beam tracing method. These results indicate that the accuracy of our method for 4×4 or 8×8 sampling resolution can be close to that of beam tracing.

handle non-planar geometry correctly. This is common to most volumetric approaches, but we can still approximate the reflections by increasing the number of sample rays and using the planar approximation defined by the local surface normal. Our implementation is also currently limited to point sound sources. However, we can potentially simulate area and volumetric sources if the source can be approximated by planar surfaces. The lack of non-specular reflections is another limitation of our approach. For example, it could be hard to create a frusta for diffuse reflection from a surface based on a scattering coefficient without significantly affecting the performance of our algorithm.

We also studied the behavior of our algorithm for different sampling rates. As Fig. 7 shows, the simulation time increases sub-linearly to the sampling rate due to our optimized intersection and sample combination algorithm. Fig 8 compares the resulting impulse response functions on two different models for varying sampling rates, which is significant since it is obvious that – as the sampling rate goes to infinity – our algorithm essentially becomes beam tracing. As the results show, even for low sampling resolutions, the response converges very quickly, which suggests that we can achieve almost the same quality with very low sampling rates. Of course, our approach is still a geometric algorithm and like all others its accuracy for high-quality simulation is may therefore be limited compared to full numerical simulation[47].

Note that our frustum tracing technique is could be seen to be related to adaptive super-sampling techniques in computer graphics such as [48, 17, 20]. However, recent work in interactive ray tracing (for visual rendering) has shown, that adaptive sampling – despite its natural advantages – does not perform near as fast as simpler approaches that are based on ray packets and frustum techniques. While high uniform sampling, as used in our algorithm, may seem uneconomical at first, our clipping algorithm reduces the actual work and simplicity makes this approach map much better to current hardware. Combining samples only after the sampling has been performed reduces the detail of the uniform sampling to the same that adaptive sampling would generate, but does not add any overhead to the traversal process. Similarly, there were parallel approaches in other areas such as radio [34] and sound propagation [39, 10] using adaptive beam methods, but for the same reasons they do not perform nearly as well and are limited in the scale and generality of scenes they can handle.

7 Future Work and Conclusions

There is a rich history on the synergies between the research directions in sound and light, and we apply the lessons from one wave phenomenon to the other. Our goal was to utilize the recent developments in interactive ray tracing for sound propagation. As a result, we have presented an interactive frustum tracing algorithm, which combines the speed efficiencies of ray tracing with many of the accuracy benefits of volumetric representation. All the other benefits of ray packet tracing, including SIMD optimizations, multi-threaded implementations and handling dynamic scenes are directly applicable to sound rendering. As a result we are able to render sound in complex and dynamic scenes at interactive rates. We hope that this will be a step towards including physical sound propagation into interactive applications such as games and virtual environments with dynamic environments.

For future work we would be interested in further exploring the sampling issues in our discrete clipping algorithm to minimize the error. A promising direction may be to investigate adaptive subdivision to adjust sampling rates to local geometric complexity. We are also interested in adding diffraction into the simulation, which has been shown to add important contributions to the realism. Finally, we would like to apply our algorithm to more complex scenarios and integrate them into interactive applications such as games.

Acknowledgments

We would like to thank Paul Calamia for his feedback and Charles Ehrlich for the Candlestick theater model. This work was supported in part by ARO Contracts DAAD19-02-1-0390 and W911NF-04-1-0088, NSF awards 0400134, 0429583 and 0404088, DARPA/RDECOM Contract N61339-04-C-0043, Disruptive Technology Office and Intel.

References

[1] V. Algazi, R. Duda, and D. Thompson. The CIPIC HRTF Database. In *IEEE ASSP Workshop on Applications of Signal Processing to Audio and Acoustics*, 2001.

[2] F. Antonacci, M. Foco, A. Sarti, and S. Tubaro. Real time modeling of acoustic propagation in complex environments. In *Proc. of 7th International Conference on Digital Audio Effects*, 2004.

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

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

[5] C. Brebbia, editor. *Computational Acoustics and its Environmental Applications*. Transactions of the Wessex Institute, 1995.

[6] L. Carpenter. The a-buffer, an antialiased hidden surface method. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 103–108, New York, NY, USA, 1984. ACM Press.

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

[8] B.-I. Dalenbäck, P. Svensson, and M. Kleiner. Room acoustic prediction and auralization based on an extended image source model. *The Journal of the Acoustical Society of America*, 92(4):2346, 1992.

[9] E. Deines, M. Bertram, J. Mohring, J. Jegorovs, F. Michel, H. Hagen, and G. Nielson. Comparative visualization for wave-based and geometric acoustics. *IEEE Transactions on Visualization and Computer Graphics*, 12(5), 2006.

[10] I. A. Drumm. *The Development and Application of an Adaptive Beam Tracing Algorithm to Predict the Acoustics of Auditoria*. PhD thesis, 1997.

[11] A. Farina. Ramsete - a new pyramid tracer for medium and large scale acoustic problems. In *Proceedings of EURO-NOISE*, 1995.

[12] 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.

[13] 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.

[14] T. Funkhouser, N. Tsingos, and J.-M. Jot. Survey of methods for modeling sound propagation in interactive virtual environment systems. *Presence and Teleoperation*, 2003.

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

[16] M. A. Garcia-Ruiz and J. R. Gutierrez-Pulido. An overview of auditory display to assist comprehension of molecular information. *Interact. Comput.*, 18(4):853–868, 2006.

[17] J. Genetti and D. Gordon. Ray tracing with adaptive supersampling in object space. In *Graphics Interface '93*, pages 70–77, 1993.

[18] P. S. Heckbert and P. Hanrahan. Beam tracing polygonal objects. In *Proc. of ACM SIGGRAPH*, pages 119–127, 1984.

[19] D. L. James, J. Barbic, and D. K. Pai. Precomputed acoustic transfer: output-sensitive, accurate sound generation for geometrically complex vibration sources. In *Proc. of ACM SIGGRAPH*, pages 987–995, 2006.

[20] D. G. Jon Genetti and G. Williams. Adaptive supersampling in object space using pyramidal rays. *Computer Graphics Forum*, 17(1):29–54, 1998.

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

[22] B. Kapralos, M. Jenkin, and E. Milios. Acoustic modeling utilizing an acoustic version of phonon mapping. In *Proc. of IEEE Workshop on HAVE*, 2004.

[23] 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.

[24] K. Kunz and R. Luebbers. *The Finite Difference Time Domain for Electromagnetics*. CRC Press, 1993.

[25] K. H. Kuttruff. Auralization of impulse responses modeled on the basis of ray-tracing results. *Journal of Audio Engineering Society*, 41(11):876–

880, November 1993.

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

[27] H. Lehnert. Systematic errors of the ray-tracing algorithm. *J. Applied Acoustics*, 38(2-4):207–221, 1993.

[28] R. B. Loftin. Multisensory perception: Beyond the visual in visualization. *Computing in Science and Engineering*, 05(4):56–58, 2003.

[29] T. Lokki, L. Savioja, R. Vaananen, J. Huopaniemi, and T. Takala. Creating interactive virtual auditory environments. *IEEE Computer Graphics and Applications*, 22(4):49–57, 2002.

[30] M. Naef, O. Staadt, and M. Gross. Spatialized audio rendering for immersive virtual environments. In *Proceedings of the ACM VRST*, 2002.

[31] K. V. Nesbitt. Modelling human perception to leverage the reuse of concepts across the multi-sensory design space. In *APCCM '06: Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling*, pages 65–74, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[32] J. F. O'Brien, P. R. Cook, and G. Essl. Synthesizing sounds from physically based motion. In *Proc. of ACM SIGGRAPH*, pages 529–536, 2001.

[33] T. Otsuru, Y. Uchinoura, R. Tomiku, N. Okamoto, and Y. Takahashi. Basic concept, accuracy and application of large-scale finite element sound field analysis of rooms. In *Proc. ICA 2004 (Kyoto)*, pages I–479–I–482, April 2004.

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

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

[36] L. Savioja. *Modeling Techniques for Virtual Acoustics*. PhD thesis, Helsinki University of Technology, 1999.

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

[38] S. Smith. Auditory representation of scientific data. In *Focus on Scientific Visualization*, pages 337–346, London, UK, 1993. Springer-Verlag.

[39] U. Stephenson. Quantized pyramidal beam tracing - a new algorithm for room acoustics and noise immission prognosis. *Acustica - Acta Acustica*, 82(3):517–525, 1996.

[40] H. Suzuki and A. S. Mohan. Frustum ray tracing technique for high spatial resolution channel characteristic map. In *Radio and Wireless Conference (RAWCON) 98*, pages 253–256. IEEE Press, 1998.

[41] R. Tomiku, T. Otsuru, Y. Takahashi, and D. Azuma. A computational investigation on measurements in reverberation rooms by finite element sound field analysis. In *Proc. ICA 2004 (Kyoto)*, pages II–941–II–942, April 2004.

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

[43] K. van den Doel, D. Knott, and D. K. Pai. Interactive simulation of complex audio-visual scenes. *Presence: Teleoperators and Virtual Environments*, 13(1):99–111, 2004.

[44] I. Wald, C. Benthin, M. Wagner, and P. Slusallek. Interactive rendering with coherent ray tracing. In A. Chalmers and T.-M. Rhyne, editors, *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2001)*, volume 20, pages 153–164. Blackwell Publishers, Oxford, 2001.

[45] I. Wald, S. Boulos, and P. Shirley. Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics*, 2006.

[46] M. Wand and W. Straßer. Multi-resolution sound rendering. In *SPBG'04 Symposium on Point - Based Graphics 2004*, pages 3–11, 2004.

[47] L. M. Wang, J. Rathsam, and S. R. Ryherd. Interactions of model detail level and scattering coefficients in room acoustic computer simulation. In *International Symposium on Room Acoustics: Design and Science*, 2004.

[48] T. Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, 1980.