Dynamic Sound Rendering Based on Ray-Caching

Ken Chan[1,2], Rynson W.H. Lau[2,3], and Jianmin Zhao[3]

[1] Department of Computer Science, City University of Hong Kong, Hong Kong
[2] Department of Computer Science, University of Durham, United Kingdom
[3] College of Math., Physics and Info. Engineering, Zhejiang Normal University, China

Abstract. Dynamic sound rendering is attracting a lot of attention in recent years due to its applications in computer games and architecture simulation. Although physical based methods can produce realistic outputs, they typically involve recursive tracing of sound rays, which may be computationally too expensive for interactive dynamic environments. In this paper, we propose a ray caching method that exploits ray coherence to accelerate the ray-tracing process. The proposed method is tailored for interactive sound rendering based on two approximation techniques: spatial and angular approximation. The ray cache supports intra-frame, inter-frame and inter-observer sharing of rays. We show the performance of the new method through a number of experiments.

1 Introduction

Real-time spacious sound synthesis in dynamic virtual environments is important in many applications, such as computer games and architecture simulation. It does not only provide a sense of presence to the users, but also allow the users to experience the sound effect of a virtual environment.

The main process of generating real-time spacious sound is sound rendering, which generates impulse responses to an observer from all the input sound sources of an environment. The inputs to the sound rendering process include the geometry of the environment, position and orientation of the observer, and position and waveform of each sound source. An approach to generate impulse responses is the physical approach [Gard98], which deals with the geometrical modeling of sound wave transmission. First, the paths of sound sources are traced to obtain a set of virtual sound sources. The Head-Related Transfer Functions (HRTFs) at the virtual sound source positions are then convolved with the sound waves to produce spacious sound.

In general, the physical approach is not efficient as it requires recursive tracing of sound paths in the environment. A popular method for tracing sound paths is ray-tracing, by which rays are projected from the observer in all directions. When a ray hits an object surface, a reflected ray is typically generated. For a transmissive surface, a transmitted ray may also be generated. Each new ray is further projected to the environment and recursively traced until either the ray hits a sound source or its effect to the observer becomes too small. As this approach traverses logarithmic number of surfaces for each ray, it is generally too expensive for real-time use.

H.H.S. Ip et al. (Eds.): PCM 2007, LNCS 4810, pp. 336–346, 2007.

In this paper, we propose a method that exploits ray coherence of sound to accelerate the sound ray tracing process. Our method is based on the ray-history concept [Sber04]. It incorporates multi-resolution approximation of rays in both spatial and angular dimensions to maximize the caching performance. Experimental results show that our method offers a significant performance improvement, making it suitable for use in interactive dynamic environments.

The rest of the paper is organized as follows. Section 2 describes related work on sound rendering and acceleration techniques. Section 3 presents an overview of our ray caching method. Section 4 presents the ray caching method. Section 5 shows the performance of the proposed method with a number of experiments. Section 6 briefly summarizes this paper and suggests possible future work.

2 Related Work

There are a few sound rendering methods proposed which are based on the physical approach. For example, [Funk98] proposes a beam tracing method that considers the preprocessing of reflection paths. [Funk99] accelerates the method by applying priority driven and estimation techniques. This method can achieve good results of discovering reflection paths, but has to deal with expensive beam intersections. [Muel99] proposes a scalable PC-based software sound rendering system based on ray tracing. Unfortunately, their experimental results indicate that the system cannot be used for real-time applications. [CATT] is a software system targeted for architectural acoustic simulation but it is not for interactive purpose. [LTL] is a hardware/software system for multi-purpose sound rendering. It makes use of dedicated DSPs to generate real-time spatialized sounds. Unfortunately, it is expensive and not scalable.

Our method is based on tracing sound rays. There are already a lot of acceleration techniques proposed for ray-tracing in computer graphics to exploit coherence. These techniques can be roughly classified into four categories [Rein02, Suth74]:

Object Coherence: It exploits position adjacency between objects. It works by pre-processing data structures including grids, BSPs (binary space partitions) and BVHs (bounding volume hierarchies) [Arvo89] to group related objects together. With these data structures, the number of intersection tests for each ray can be reduced.

Image Coherence: It exploits local constancy between neighboring pixels. In [Mart01, Havr03], caching methods are proposed to cache intersection points of previous primary rays in object space with their corresponding pixels in image space. The cache is then used to direct new rays towards objects that have been previously hit. In [Havr03], an additional cache storing object orders is maintained, providing information on which nearest object that a new ray is intersected with. Each entry in the cache, indexed by pixel position, stores a reference to the nearest intersected object and the number of times that it is referenced by neighboring pixels. If this number exceeds a threshold, the same pixel in the next frame would be reprojected to the same object. Otherwise, the traditional ray tracing technique is applied.

Ray Coherence: It exploits similarity between adjacent rays on traveling directions and intersected objects. [Wald01] employs parallelism provided by SIMD instructions and careful memory layout for the BSP data structure to improve the ray tracing

performance. Packets of rays are traversed and intersected in parallel since they have high chance of intersecting the same objects. Ray coherence can be further extended by tracking the ray paths across consecutive frames. [Sber04] proposes a fast global illumination method to support moving light sources. As a preprocessing step, all direct and reflected paths to light sources are traced. The illumination by the traced paths and the first hit points from the light sources are cached. Then, all the traced paths are weighted among all frames for indirect illumination and the weighed paths are cached. During frame generation, direct illumination is recomputed for each frame while the weighted paths are reused from cache to produce the final image. With this method, direct illumination should be recomputed while indirect illumination requires only computing the weighted impact of all virtual light sources. Path splitting and joining are useful ideas of ray caching. However, the method benefits mostly on indirect illumination due to its aim on caching shadow paths. As it is based on preprocessing, it is only suitable for predefined animations.

Frame Coherence: It exploits similarity of the projected rays between consecutive frames. One approach to frame coherence is spatial subdivision. [Mura90] proposes a ray-object intersection tree structure with voxel traversal history, which is a list of all voxels traversed by a path segment, and intersection history, which is a list of intersection points that hit a moving object. The intersection history helps eliminate unnecessary intersection tests to non-moving objects. Due to the voxels architecture, this method has high memory cost and does not support moving camera. In [Davi99], frame coherence is applied to accelerate animation. It subdivides the scene into regular voxels and involves a preprocessing step to test all object movements in an animation against the voxels. All voxels passed through by objects are marked with frame numbers. During ray tracing, primary and higher order rays are projected from a pixel, intersecting the voxels. The lowest frame number of all passed through voxels is referenced and marked in a cache structure called t-buffer, which is then used to determine the next frame number that the pixel should project rays again. Caching of rays is inherently achieved by the voxel structure and reuse of rays is inherently achieved by the t-buffer. This method suffers from the costly preprocessing step, and is less useful in moving cameras as coherence is only exploited for static background.

3 Method Overview

The core of a sound rendering process based on the physical approach is the search for virtual sound sources. In our method, this is done by ray tracing. Rays are projected from the observer in different directions to search for the nearest intersected objects or sound sources. If an object surface is hit, a reflected ray is generated from the surface, which further intersects other objects. New rays would be generated until one hits a sound source. A *reverberation path* is a sequence of projected rays starting from the observer until reaching a sound source. By mirroring the reverberation path, we could locate the position of the virtual sound source at the end of the path. Each observer maintains a virtual source map in each frame, storing information of all the virtual sound sources reaching him/her. During the auralization process, we generate spacious impulse responses by convoluting the virtual source map with HRTFs.

Spacious waveform can then be generated by multiplying the spacious impulse responses with the input sound sources.

In a dynamic environment, observer and sound sources may move around. For each observer or sound source movement, we need to update all reverberation paths. With a traditional ray tracing method, we need to perform all the intersection tests again, making it difficult to achieve real-time response. Our research objective here is to discover and exploit the coherencies between observer/sound source movements. Typically, an observer moves step by step, rather than jumps abruptly, from his current position. During each frame, there are likely a lot of rays with similar directions to the rays projected in the previous frame, intersecting the same objects at similar angles and reflecting in a similar way. If we could cache the intersection history of previous rays and reuse them, we may save a lot of computations.

Our method employs the ray coherence concept for acceleration. Intersection history of traced rays is cached for future reuse. Rays with similar starting points and projection directions as previously projected rays do not have to be retraced again. To cache the intersection history, we subdivide the objects into discrete patches. Ray transmission is represented by pointers from one patch to another. Thus, the ray cache is a graph with all the patches as nodes and the rays as edges. When a ray intersects a previously intersected patch, we can easily locate the next patch to be intersected from the cache, replacing the costly intersection tests with memory referencing operations. However, there are a number of issues to be considered when designing a ray cache. For example, schemes have to be set up to fully utilize the ray cache in dynamic environments with moving observers, sound sources, and objects.

3.1 Patch Subdivision

We subdivide each object into patches small enough that we may treat all incident rays the same way if they hit the same patch at the same direction, even at different intersection points. For rays hitting the same patch from the same direction, all of their reflected rays will be projected from the center of the patch. In this way, the original reflective rays of the incoming rays are *spatially approximated* by the outgoing ray. In addition, each patch is also subdivided into a number of angular divisions. Incoming rays falling within the same angular division would be reflected by the same ray. In this way, the incoming rays are *angularly approximated* by the outgoing reflected ray.

3.2 Multi-resolution Patches

The nature of ray tracing is that spatial resolution decreases along with increasing distance of each projected ray as the separation between two neighbor rays is proportional to their ray lengths. Missing hits may occur for distant sound sources if the angular width between neighbor rays is too high or the diameter of the sound source is too small. To avoid missing hits without adjusting the initial ray sampling rate, we may increase the diameter of the sound source as the path length increases.

We apply this concept to the size of the patches. Without adjusting the sampling rate of the projected rays, we may increase the patch size as the path length increases. This may increase the probability of reusing projected rays without increasing the

approximation error. When a ray hits a patch, we apply spatial approximation and re-project the ray from the patch center. To support variable patch sizes, we apply multi-resolution modeling techniques here. Each surface of an object is subdivided into a hierarchy of patches. Although many different multi-resolution methods have been proposed [To01], we have adopted a simple method in our implementation. For each surface, we uniformly subdivide it recursively to form a quad-tree. This means that the patch length reduces by half as we move down the quad-tree from one level to the next. Hence, when a ray interests a surface patch, we determine the appropriate resolution of the patch according to its path length.

3.3 Intra-frame, Inter-frame and Inter-observer Coherencies

Intra-frame coherence occurs when a re-projected ray is shared by more than one reverberation path within the same frame. This could be due to, for example, neighboring rays that hit the same patch at similar angle. These rays will then have the same reverberation path, making it possible to reuse the traced paths.

Inter-frame coherence occurs when ray history of previous frames are reused. This could happen easily if an observer is moving step by step. Figure 1 shows that an observer moves forward by one step. Paths traced for patches B, C, and E in the last frame can be reused in the current frame since the rays in both frames intersect the same patches at similar angles. We may observe that it is more likely to reuse rays intersecting distant patches than those intersecting nearby patches. For example, the angle between the dashed arrow and the solid arrow hitting D is larger than that of E. Hence, there is a higher chance that the two rays intersecting E fall within the same angular division used in angular approximation, hence reusing the rays intersecting E.

Inter-observer coherence occurs when ray history of another observer is reused. One scenario of inter-observer coherence is when a group of observers move in a similar direction, e.g., in a virtual touring.

4 Ray Cache

The ray cache is composed of a tree and a graph attached to the tree. Each object is composed of a number of surfaces. Each surface is divided into multiple levels of patches and each patch is further divided into angular divisions. Figure 2 shows an example object tree, starting from object at the root to angular divisions at the leaves. Typically, there is only a small amount of angular division nodes which are ever accessed. To save run-time memory, only the accessed angular division nodes are instantiated during ray-tracing. We reference each leaf node in the tree by a multiple-component division index, (object, surface, {patch, patch, ...}, division), which will simply be referred to as *division* in remaining text.

When a surface is intersected by a ray, it is recursively subdivided into smaller patches until the patch length is just small enough proportional to the path length of the ray. To save memory, only the patches containing intersection points are further subdivided. When we reach the appropriate patch level, we calculate the angular division to approximate the incoming ray direction. Figure 3 shows the patch hierarchy of surface **W** found in Figure 2. For clarity, we only show the reflective

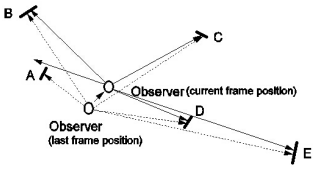


Fig. 1. Inter-frame coherence

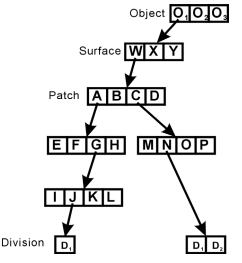


Fig. 2. The ray cache tree

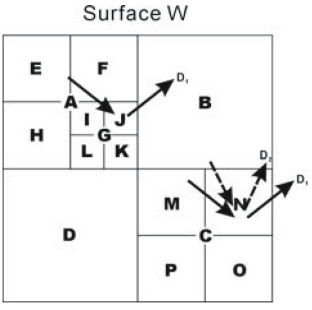


Fig. 3. Patch hierarchy of surface **W** in Fig. 2

property of the patch. There should be another patch hierarchy to represent the transmissive property. For each ray we stored in cache, we also store the backward version of the ray to facilitate path finding in backward direction. This is particularly useful for locating paths from sound source to observer, in a sound source movement.

4.1 Constructing the Ray Cache

We use an example to illustrate how the ray cache is being constructed. To simplify the description, we assume in this example that all rays are equidistant and each ray intersection happens to cause the patch resolution to reduce by one level.

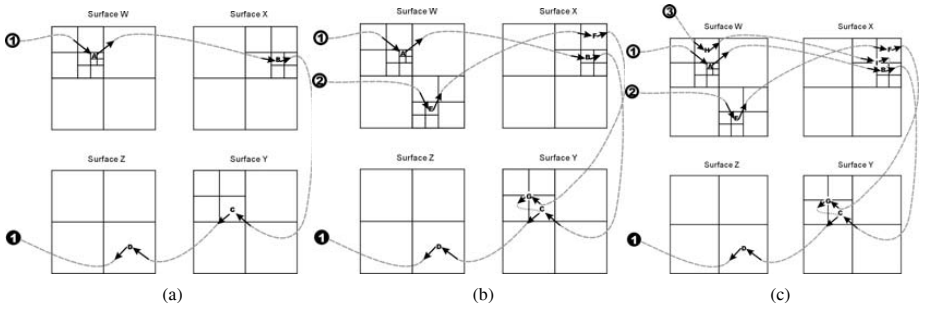


Fig. 4. Constructing the ray cache (white circles are observers; black circles are sound sources)

Initially the ray cache is empty. The first path is constructed by tracing the reverberation path from the observer to the sound source as in Figure 4(a). The intersected *divisions* are instantiated in the tree. The division intersection sequence is recorded by creating an edge entry from one *division* to another. This intersection sequence is retained in the ray cache even if the path does not hit a sound source. In this example, the path traced for observer 1 intersects 4 surfaces via patches A, B, C and D. The path finally reaches sound source 1. When another path is to be traced, in addition to instantiating the new *divisions* and caching the new rays, the process will seek reuse opportunities as in Figure 4(b). (Note that the patch resolution is reduced

whenever a ray intersects a patch.) If we reach a patch of required patch resolution, which contains child patches, we search the child patches for cached records of the same incoming *division*. If the same incoming *division* exists, we may reuse and follow the cached edge from the child patch. In other words, each edge entry can be propagated upwards to parent patches for reuse. In the example, a path is traced for observer 2 through patches E, F and G. In patch G, a child patch C is found which has a cached edge of the same *division*. Thus, patch G is connected to patch C and finally reaches sound source 1 by reusing the cached path.

The reuse of cached edges follows a policy that only the nearest child patches will be chosen to minimize the spatial error as in Figure 4(c). Since the re-projection point of the parent patch is at the center of the patch, the child patch which is nearest to the center will be chosen. In the figure, observer 3 traces a path through patches H and I. In patch I, 2 reusable cached edges are found in child patches B and F. The cached edge in patch B is chosen for reuse since patch B is nearer to the center of patch I.

4.2 Ray Cache Purging

In a dynamic environment, the size of the ray cache will continue to grow in time as the number of rays projected increases in time. When the cache is full, we need to remove some of the cached rays. Our experiments (as will be shown in Section 5.2) show that most of the reused rays are computed in the recent frames. Thus, we may use time-stamping to keep track of the usage of cached rays for purging.

Each ray in the ray cache is time-stamped during its creation or when it is revisited. A pointer list is created in each frame to reference to all rays which are created or reused in that frame. When the ray cache is full, purging of rays need to be carried out starting from those with earlier frame numbers. The pointer lists of the earlier frames are used to find out rays which have not been reused in recent frames. These rays, with their associated divisions, patches, and parent patches, will be all removed.

4.3 Movement of Observers, Sound Sources and Scene Objects

Observer Movement: An observer is a node that initiates sound rays in all directions to the initial intersected *divisions* to form edges. When it moves, its virtual source map and all edges connecting it to the initial intersected *divisions* are cleared. We reconstruct new edges from the new position of this observer by projecting new rays to all direction to find the initial intersected *divisions*. Once a new ray hits a cached *division*, we follow the path starting from the cached edge. If a sound source can be reached finally, a new path is completed and a new virtual source map entry would be created. If the initial intersected *division* has no cached edges, the new path will be further traced until it reaches a cached *division* or its effect becomes insignificant.

The main cost of observer movement is on updating the reverberation paths from the observer. The ray cache helps reduce the number of intersection tests if the newly projected rays hit a cached *division*.

Sound Source Movement: A sound source is the reverse version of an observer. A sound source attaches backward edges to all *divisions* linked to it. When it moves, all forward and backward edges linked to it will need to be removed. Then, new initial rays will be projected from the sound source to search for intersections. When a

surface is intersected by a ray, we need to check if it contains any cached *divisions*. We start from the surface level, search down the tree for the patch node that contains the intersection point, until we reach the lowest level. If we find a node with cached *divisions* matching the outgoing direction, an edge is created to link that node to the sound source. A backward edge is also created.

Scene Object Movement: With the ray cache, when a scene object is moved, only a small number of rays may need to be re-traced in typical situations. To do this, we first remove the edge entries of all patches of the moved object. We then test all the remaining edge entries in the cache for intersection with the moved object. If an edge is found intersecting the moved object, it is removed. At the same time, we also test all the edges initiated from each of the observers and sound sources for intersection with the moved object. All the intersected edges would be removed. Finally, we check all the previously constructed reverberation paths starting from all observers to see whether there are broken links. If a broken link is found, we apply ray-tracing from the broken point to re-trace the remaining path.

5 Results and Discussions

To study the performance of the proposed method, we have conducted a number of experiments, all on a single PC with a P4 2.0GHz CPU and 512MB of memory. There are a total of 20 sound sources used.

5.1 Inter-frame Coherence vs. Different Enclosure Volumes

As stated in Section 3.3, the degree of inter-frame coherence depends on how far away the patch is from the observer. To verify this statement, we study the ray cache usage in 3 different environments with a room of different dimensions but of the same architecture. Within each of the environments the observer walks into the room from outdoors, walks around the room and then leaves it.

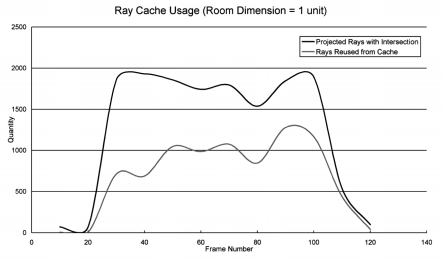
Figure 5 shows the experimental results. In both the initial and the final frames, there are only a few projected rays that intersect objects in the environment in all three cases. This is because the observer is outdoors during these frames. In the middle frames, where the observer is inside the room, the number of intersected rays increases significantly. This is due to both the intersection of rays with the room interior surfaces, and the generation of reflective rays from these surfaces. If we look at the quantity difference between the two curves in each diagram, i.e., number of projected rays vs. number of rays reused, we can see that in a small room, the proportion of rays reused from the ray cache is smaller than that in a large room. This agrees with our discussion earlier that there is a higher degree of inter-frame coherence if the objects (or room surfaces) are further away.

5.2 Cache Reuse Performance

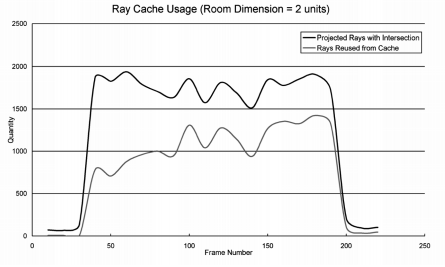
To study how the ray cache grows in time, we have plotted the number of projected rays and the number of cached edges in the ray cache as a function of the frame number as shown in Figure 6. The memory consumption of the ray cache is

proportional to the total number of edges cached. We can see that the size of the ray cache grows sub-linearly with the total number of projected rays, while the memory consumption is continuously growing with the frame number.

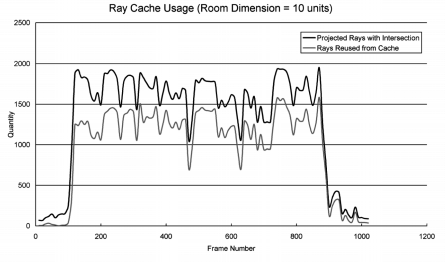
We have analyzed the distribution of reused rays, as shown in Figure 7. The test scene is similar to the one used in Section 5.1, where an observer moves from outdoors to inside a room, walks around and then leaves the room. There are a total of 100 frames and we divide them into three groups: the initial frames when the observer is outdoors, the middle frames when the observer is inside the room and the final frames when the observer is again outdoors. The height of each bar in Figure 7 indicates the average number of rays being reused from the cache within a frame



(a) Room dimension = 1 unit.



(b) Room dimension = 2 units.



(c) Room dimension = 10 units.

Fig. 5. Ray cache usage

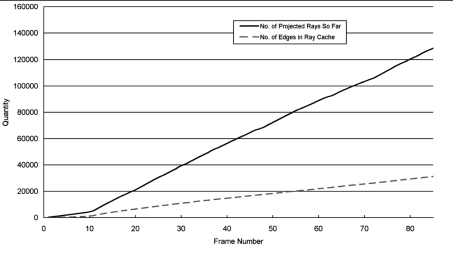


Fig. 6. Growth of the ray cache

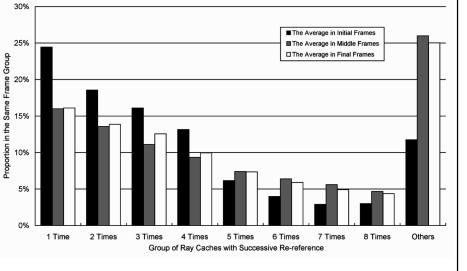


Fig. 7. Amount of reused rays

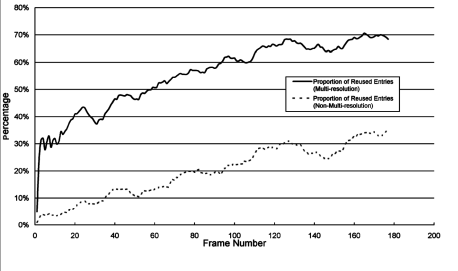


Fig. 8. Percentage of reused rays with/without multi-resolution patch subdivision

group. In general, there are more than 74% of the reused rays that are reused within 8 consecutive frames. This high percentage of reused rays is due to the inter-frame coherence of observer movement. Hence, we may conclude that if we apply the time-stamping approach to purge the cache up to the last 8 frames, we may still be able to keep about 74% of reused rays in the cache.

5.3 Multi-resolution vs. Non-multi-resolution Patches

We presented our spatial approximation technique based on multi-resolution patch subdivision in Section 3.2,. In this experiment, we compare the percentage of reused rays from the ray cache with and without applying the multi-resolution patch subdivision technique. The same test scene and same observer movement (only within the room) are used here. Figure 8 shows the results of the ray cache usage. As the observer is moving inside the room only, the two percentage reuse curves (with and without multi-resolution patch subdivision) are increasing as the frame number. This means that in time, more and more reverberation paths in the ray cache are reused. We can see that with the multi-resolution technique, the percentage of reused rays is much higher, meaning that the technique allows more efficient use of the ray cache.

5.4 Performance in Multi-observer Environments

In this experiment, we would like to study the performance improvement of our method with and without ray caching. We compare the performance by measuring the average time spent on projecting a single ray in a multi-observer environment. A number of observers are set to move inside the same room randomly for a period of 25 frames and the response time of the sound rendering server is recorded. Table 1 shows our experimental results. We can see from the table that in general, the ray caching method could speed up ray-tracing by 38% of the original time when there is only one observer. The speed up increases as the number of observers increases, due to more cached rays being reused among the observers.

Table 1. Ray-tracing time with and without ray cache in a multi-observer environment

No. of observers	Time per ray (Non-cached)	Time per ray (Cached)	Speed-up
1	0.0876 ms.	0.0542 ms.	38%
2	0.0822 ms.	0.0486 ms.	41%
3	0.0806 ms.	0.0455 ms.	44%
4	0.0819 ms.	0.0444 ms.	46%

6 Conclusions and Future Work

Sound rendering for dynamic environments based on the physical approach poses a great challenge due to the high computational cost in computing the reverberation paths. In this work, we have successfully improved the performance of the physically based sound rendering process by exploiting ray coherence. Our method is based on two approximation techniques in caching traced rays for reuse: spatial approximation and angular approximation. Our experimental results show that significant

performance improvements are achieved by using the ray cache. The new method is particularly beneficial to multi-user, interactive environments, in which a higher percentage of rays can be reused.

We have attempted to extend our proposed method to a multi-server environment. However, we observe from our experimental results that networking overheads can significantly affect the performance. As a future work, we would like to investigate an efficient scheme to share the cached information. Such a sharing scheme is essential if the ray caching method is to be used in a multi-server environment.

References

[Arvo89]Arvo, J., Kirk, D.: A Survey of Ray Tracing Acceleration Techniques. In: Glassner (ed.) An Introduction to Ray Tracing, Academic Press, London (1989)

[CATT]CATT-Acoustic, CATT, Sweden, http://www.netg.se/ catt/

[Davi99]Davis, T., Davis, E.: Exploiting Frame Coherence with the Temporal Depth Buffer in a Distributed Computing Environment. In: Proc. IEEE Symp. on Parallel visualization and graphic (1999)

[Funk98]Funkhouser, T., Carlbom, I., et al.: A Beam Tracing Approach to Acoustic Modeling for Interactive Virtual Environments. In: Proc. ACM SIGGRAPH, pp. 21–32 (1998)

[Funk99]Funkhouser, T., Min, P., Carlbom, I.: Real-Time Acoustic Modeling for Distributed Virtual Environments. In: Proc. ACM SIGGRAPH, pp. 365–374 (1999)

[Gard98]Gardner, B.: Reverberation Algorithms. In: Applications of Digital Signal Processing to Audio and Acoustics, Kluwer Academic Publishers, Dordrecht (1998)

[Havr03]Havran, V., Bittner, J.: Exploiting Temporal Coherence in Ray Casted Walkthroughs. In: Proc. Spring Conf. on Computer Graphics (2003)

[LTL]Lake Technology Limited, Huron acoustic virtual reality

[Mart01]Martin, W., Parker, S., Reinhard, E., Shirley, P., Thompson, W.: Temporally Coherent Interactive Ray Tracing. Technical Report UUCS-01-005, University of Utah (2001)

[Muel99]Mueller, W., Ullmann, F.: A Scalable System for 3D Audio Raytracing. In: Proc. IEEE ICME (1999)

[Mura90]Murakami, K., Hirota, K.: Incremental Ray Tracing. In: Proc. EG Workshop on Photosimulation, Realism and Physics in Computer Graphics, pp. 15–29 (June 1990)

[Rein02]Reinhard, E.: Parallel Global Illumination Algorithms. Practical Parallel Rendering, AK Peters (2002)

[Sber04]Sbert, M., László, S., László, S.: Real-time Light Animation. In: Proc. Eurographics (2004)

[Suth74]Sutherland, I., Sproull, R., Schumacker, R.: A Characterization of Ten Hidden-Surface Algorithms. ACM Computing Surveys 5(1), 1–55 (1974)

[To01]To, D., Lau, R., Green, M.: An Adaptive Multi-Resolution Method for Progressive Model Transmission. Presence 10(1), 62–74 (2001)

[Wald01]Wald, I., Slusallek, P., Benthin, C., Wagner, M.: Interactive Rendering with Coherent Ray Tracing. In: Proc. Eurographics, pp. 153–164 (2001)