YET ANOTHER REAL-TIME RAY TRACER

Cuong Bui Phu [1] **Ngoc Minh Le** [2]

[1] *Faculty of Information Technology, Pham Van Dong University*

Email: bpcuong@pdu.edu.vn

[2] *Faculty of Computer Science & Engineering, Ho Chi Minh City University of Technology*

Email: minhle@cse.hcmut.edu.vn

Abstract—Augmented reality is now rapidly growing and playing an important role in diverse applications such as assembly, entertainment, medicine… Computer graphics applications can be divided into two different categories: offline applications and interactive applications. Interactive computer graphics applications can be considered as an enabling technology for augmented reality. Although ray tracing has become an important method for rendering images offline for a long time, it was slow in interactive environments. With recent developments in powerful hardware, along with continuous improvement in algorithms, real-time ray tracing has become reality. In this paper, we describe the design and implementation of a real-time ray tracer that combines known efficient acceleration data structures (or acceleration structures for short) and exploits ray coherence for multi-core processor architecture to improve further the performance. A factor for the good running time of the ray tracer is that it handles static and dynamic objects separately.

Key words: ray tracing, real-time ray tracing, acceleration structure, kd-tree, BVH, lazy BVH

1. INTRODUCTION

In ray tracing technique the visibility of an object's surface is determined by tracing the path of rays from the viewer's eye to the object. Starting from the eye point, rays passing through the pixels in the image plane are sent out and the closest intersection (if any) between each ray and the object's surface is identified. The pixel, through which a ray has passed, will be updated according to the color that the ray sends back.

Although image rendering techniques based on ray tracing are not a new technology, the application of ray tracing to real-time image rendering systems as well as interactive systems has been impossible for a long time because of the limited hardware and algorithms. The survey [11] gives a complete overview on recent research in animated ray tracing.

In this paper we combine known efficient solutions [1][9][8] to the following subproblems to construct a new efficient real-time ray tracer that is implemented as a library. We have performed some experiments to evaluate the performance of our ray tracer.

a. Implement an algorithm to efficiently build a kd-tree based on a surface area heuristic (SAH) [10]: kd-tree will be built in a preprocessing stage in $O(n \log n)$ time, under certain assumptions. A thorough analysis of SAHs is given in [2]. In addition, to increase the cache hit rate, kd-tree nodes will be arranged in a contiguous memory area and the size of each node will be 8 bytes (for both internal and leaf nodes). To take advantage of the computing power of multi-core processors, the kd-tree building process will be parallelized based on the number of active processor cores [1].

b. Exploit the coherence between rays that are close to each other to improve the performance of the ray tracing process by grouping rays into ray packets. A selection criterium for ray grouping is the closeness of the vectors showing the ray directions. To take advantage of SIMD instructions supported by the hardware, ray packets are created with the size of the 'vector' size of the Streaming SIMD Extensions (SSE) instruction, which implies that each ray packet consists of exactly four rays [1].

c. Ray tracing applications for dynamic scenes in real time -- Manage objects in motion or deformable objects using acceleration structures to achieve rendering that is fast enough for real-time applications. This problem has been approached by classifying objects based on their kinematic

dependence on each other, i.e. grouping objects into static and dynamic ones, and applying suitable acceleration structures to each group [8].

d. Take advantages of the ability to update and edit the bounding volume hierarchy (BVH) acceleration structure [1] to reduce the time for refining it, so as to make the BVH compatible with the changes in each frame rendered from the dynamic models. The BVH is built following a strategy of lazy contruction, i.e. while tracing a ray only actually needed parts are built. In addition, for each frame the BVH is not rebuilt entirely from the beginning, but only the parts being determined as 'degenerate'. Degeneracy is triggered by changes from frame to frame, taking the advantages of frame-to-frame coherence [1].

e. The process of tracing rays can be parallelized on multi-core processor architectures.

The rest of this paper is organized as follows. Section 2 describes acceleration structures. Section 3 presents some details. The structure of our ray tracer is described in Section 4. Experimental results are discussed in Section 5. We conclude in Section 6.

2. ACCELERATION STRUCTURES

To accelerate ray tracing, acceleration structures that partition the triangles in the scene, and thus reduce the number of tests on intersection between rays and triangles have been developed. Tracing a ray takes $O(\log n)$ time, a significant improvement compared with the linear running time of previous algorithms. However, these structures must be built up and maintained for the active objects, which makes their implementation difficult, and may increase the total running time.

Adaptive acceleration structures such as kd-tree and BVH are commonly used in constructing real-time ray tracing systems. They can be built in $O(n \log n)$ time.

To ensure the quality of acceleration structures for objects in motion, an update strategy is necessary. By using the information from the last image frame rendering and performing a partial modification to the current acceleration structure, the total cost for maintaining the acceleration structure can be reduced. The method is particularly suitable for rigid or even deformable objects. For fully dynamic objects, most acceleration structures reduce in quality. In this case, the reconstruction method is possibly faster than the selective-restructuring method [11]. This is because the selective-restructuring method raises additional costs for the process of determining the subtrees that are already degraded and need to be updated. It is clear that the incremental update method is not always better than a rebuild-from-scratch method, so the incremental update method should be applied carefully. The incremental update strategy may also have negative effects on the tracing process. This is because auxiliary data need to be stored to save the previous state of the acceleration structure. For adaptive acceleration structures such as BVH, this method may increase the size of the node, thus increase costs for using the memory bandwidth during the tracing process.

Tracing secondary rays

Another challenge in implementing real-time ray tracing is how to quickly tracing the rays. Efficiency of traversing an acceleration structure may be achieved by exploiting the coherence of rays so that they can be traced together as a ray packet. This approach is efficient for primary rays that can be arranged in groups so that the corresponding components of the direction vectors have the same sign and are less 'divergent'. These rays traverse the acceleration structure with most of the same information, which allows a significant cost reduction for tracing all rays in the packet. Secondary rays are generated at ray-surface intersection points to realize effects such as shadow, reflection and refraction. Depending on the distribution of light sources and the curvature of the object surface, secondary rays are generated correspondingly. Tracing of rays packet may quickly degenerate to tracing each individual ray, thus reduce the running time performance. Therefore, efficiently tracing secondary rays is still a problem to be solved.

In addition, tracing secondary rays may give wrong things that show up on the rendered image. This might be due to the limitation of precision when performing calculations using floating point approximating real numbers. A practical method to solve this problem is using an 'epsilon'. The epsilon value is generally determined based on experience with each particular situation; there is no single epsilon value for many scenes. However, the epsilon value should be chosen carefully to avoid ignoring e.g. valid intersections. More sophisticated techniques such as 'symbolic perturbation' or 'exact geometric computation' in computational geometry can be applied here.

3. SOME DETAILS

Grouping geometric primitives (triangles) according to their motion dependence on each other into objects

We classify the triangles in the scene in three kinds of objects: static objects, hierarchical transformed objects (e.g. open kinematic trees) and independent geometric primitives (e.g. triangles). In architectural scenes, most parts of the scene are static, this occurs both in offline as well as real-time applications. For static objects, we can organize their triangle meshes in the acceleration structure efficiently in a preprocessing step prior to the rendering of the scene.

Using kd-tree on static objects to accelerate ray tracing

When applying ray tracing to static objects, the kd-tree as described in [1] is an efficient acceleration structure that supports accelerating ray tracing for static objects well. This is because we just have to build the kd-tree only once in the preprocessing stage. After that, it can be reused for

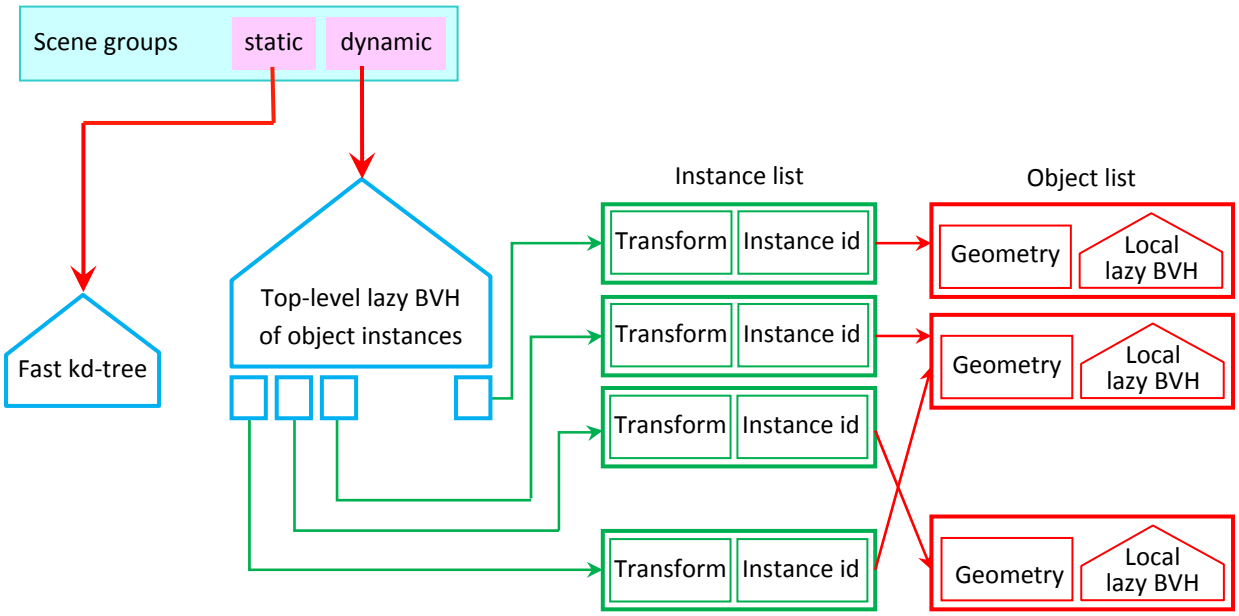

Figure 1. Kd-tree and the two-level acceleration structure. Refer also [8].

rendering the next image frames again and again. Kd-trees can be built based on SAH [10].

Utilizing ray coherence for kd-tree traversing algorithm

By grouping multiple 'similar' rays into a ray packet as described in [1], the performance of traversing kd-tree for such packed rays can be improved further. When rays have the same set of attributes, some operations on them can be done only once. This is usually utilized when testing intersection between a triangle and rays having the same origin, e.g. in case of primary rays.

On the x86 architecture, ray packets allow more performance improvement when using vector instructions that the SSE instruction set provides. These instructions perform arithmetic and comparison operations on multiple data [1]. By grouping together a number of rays, the amount of work done for each instruction will be increased and the amount of memory access cycles will be reduced.

Using lazy BVH to accelerate ray tracing on dynamic objects

A scene can have objects occluded by other ones so that a ray from the view point cannot intersect with occluded objects. Moreover the viewing angle is limited, so there may be objects outside the viewing area. Therefore we don't have to build a full acceleration structure, especially for dynamic objects. This approach may significantly reduce the time required to build the acceleration structure, and reduce the necessary storage space in memory. For dynamic objects, we use the BVH because with this acceleration structure we can restructure selectively [11] from frame to frame without the need to build the acceleration structure from scratch.

Using a two-level acceleration structure

After testing the intersection between a ray and static objects organized in the kd-tree, we must continue tracing the rays through the acceleration structure to find a closer intersection (if any) between the ray and the dynamic objects. Based on Wald proposal [8], we use a BVH acceleration structure for dynamic objects and a two-level acceleration structure to support ray tracing on dynamic objects, including rigid, deformable and fully dynamic ones. Our approach differs from Wald's in that we use BVH instead of BSP.

4. LIBRARY SYSTEM ARCHITECTURE

Our ray tracing software is implemented as a library. A basic concept in the implementation is the use of containers to store object geometry and motion parameters so that we can combine them to create an efficient ray tracing system allowing both static and dynamic objects. We chose the C++ language as the implementation language. With C++ we can manage data structures at a low level, e.g. specify address of memory area and treating a pointer as a positive integer.

Kitchen	BVH only	Kd-tree + BVH	Speedup
a) Diffusion	81.55s	87.89s	-7.2%
b) Diffusion with shadow	596.04s	214.78s	+177.5%
c) Diffusion with reflection and refraction	461.72s	252.11s	+83.1%
d) All of the effects	966.10s	380.06s	+154.2%
Robots	**BVH only**	**Kd-tree + BVH**	**Speedup**
a) Diffusion	88.36s	93.42s	-5.4%
b) Diffusion with shadow	506.26s	143.64s	+252.5%
c) Diffusion with reflection and refraction	2879.15s	456.13s	+531.2%
d) All of the effects	3275.54s	509.00s	+543.5%

Table 1. Experimental data for rendering the BART models with a resolution of 512x400 pixels. The maximum depth of recursion for reflection and refraction is 2 .

According to the analysis in Sections 2 and 3 above, the kd-tree acceleration structure is used for static objects. This structure is created using the scanning method [10] to generate a high quality structure.

A container contains object motion parameters and geometry can be created directly and allows access to the buffer containing triangle vertices and index. Once the application has completed an update to the buffer container upon a motion parameter change request, the application must invoke one of the update methods for the corresponding container to refresh the BVH. Edit and selective reconstruction is a technique often used to keep the update time low. However, if the triangles are added to or removed from the buffer of the container, we must build BVH from scratch to allow resizing the internal data structure.

5. EXPERIMENTAL RESULTS

Measurement data presented in this paper have been collected from a dual-core PC, core speed 1.73 GHz and memory of 1 GB. The PC runs under the operating system Fedora Core 9 (i386 Red Hat Linux). We have configured the rendering to handle tiles, each of size 8x8 pixels. Works on the tiles are dynamically distributed to two threads. We have used the GNU compiler GCC 4.3.0 to compile the application, SSE instructions have been generated.

We have measured the time necessary for rendering each image frame corresponding to different configurations such as the rendering method and the acceleration structure used. We used datasets provided by Lext and his collaborators [6].

Table 1 above provides experimental data for analyzing the rendering performance in two configurations:

- Configuration 1: use only a two-level BVH acceleration structure for all triangles irrespective of whether objects are static or dynamic.
- Configuration 2: use a kd-tree for static objects and a two-level BVH acceleration structure for dynamic objects.

We see that, in the absence of secondary rays, the BVH-only method performs better than the method of combining both acceleration structures. This is the case when rendering uses only diffuse light. However, once secondary rays are generated, using a kd-tree for static objects reduces the rendering time. The reason is that the BVH traversal does not achieve good running time performance when tracing packets that are formed by only few rays. This is the common case for reflection and refraction rays, and even shadow rays in case where only a limited number of light sources are present in the scene.

By separately handling static objects using a kd-tree, the BVH contains less triangles, so it may be traversed faster. Since the kd-tree traversal by a small number of rays has only minimal performance impact, the whole rendering time is reduced. In particular, rendering the robot model with high levels of reflection has many benefits from this approach, with speed-ups up to 543%.

Figs 2 and 3 show a detail view of the time distribution for rendering image frames in case of combination with different effects such as diffuse light, shadow, reflection, and refraction. The data in these figures are calculated using following formulas.

$$t_{shadows} = t_{diffuse+shadows} - t_{diffuse} \quad (1)$$

$$t_{reflections+refractions} = t_{diffuse+reflections+refractions} - t_{diffuse} \quad (2)$$

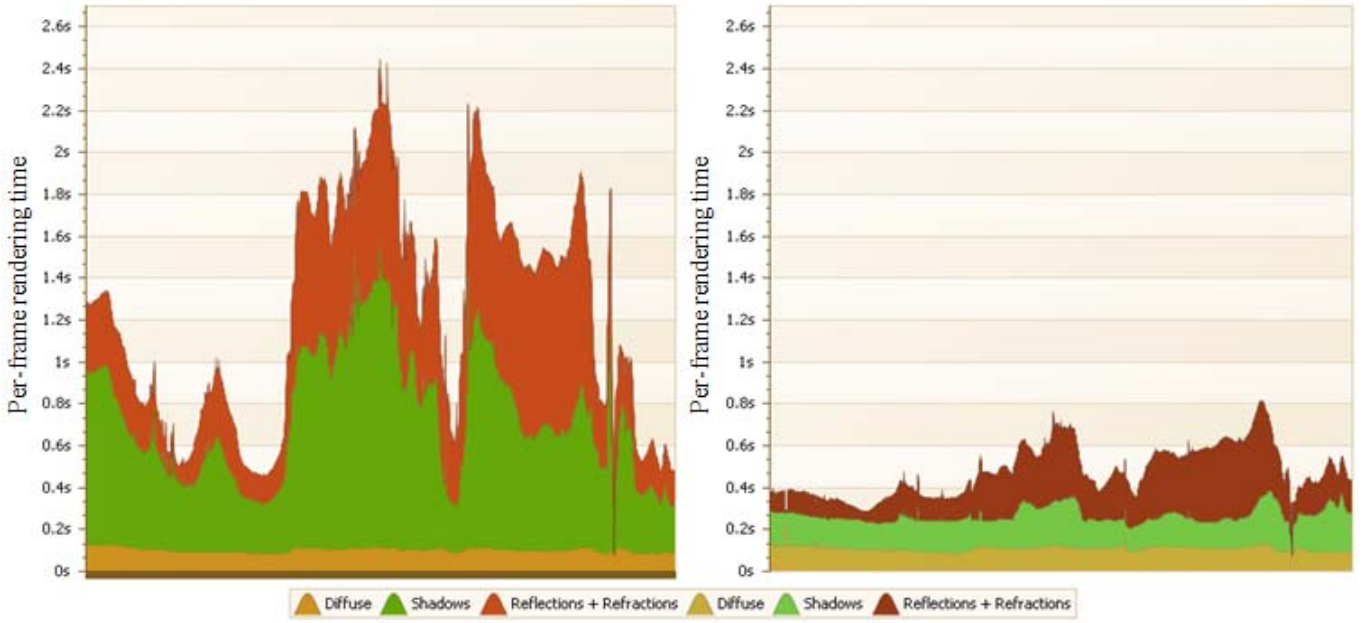

Figure 2. Rendering times for individual frames for the rigid BART kitchen

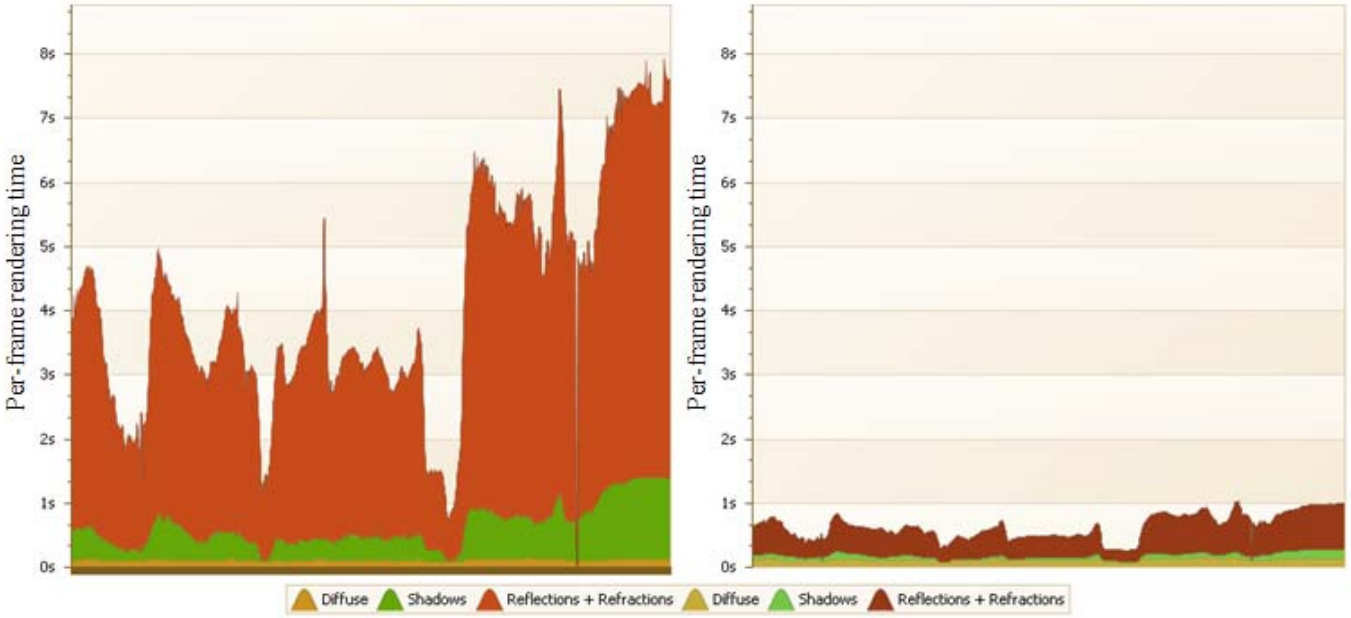

Figure 3. Rendering times for individual frames for the rigid BART robots

$$t_{total} = t_{diffuse} + t_{shadows} + t_{reflections+refractions} \quad (3)$$

In Figs 2 and 3 above, the graphs on the left depict the rendering time for a sequence of frames generated for the BART robots and kitchen models [6] in the case of using a BVH two-level acceleration structure, whereas the graphs on the right show the rendering time for a sequence of frames generated for the BART robots and kitchen models but in the case of a combination of a kd-tree and a BVH two-level acceleration structures. The image resolution is

512x400 pixels; the maximum depth of recursion for reflection and refraction rays is 2.

6. CONCLUSION AND FUTURE WORK

Our future work includes

a. How to trace secondary rays efficiently is still an issue. This issue is closely related to the problem of assembling sufficiently large ray packets. Especially, traversing a BVH with small ray packets might increase the running time. This problem can be approached by processing multiple interleaved image tiles and arranging secondary rays appropriately. However, this approach raises costs for swapping processes between tiles and costs for sorting rays. Therefore, we need further study on this issue to improve the core of ray tracing.

b. We are planning to test the performance of our ray tracing software in augmented reality applications, in which the user controls the navigation in real-time through an RGBD sensor such as a Microsoft Kinect. Because the hard task of delivering depth information is performed by the RGBD sensor, most CPU time can be reserved for the ray tracer to achieve real-time rendering.

REFERENCES

[1] C. Benthin, "Realtime Ray Tracing on current CPU Architectures," PhD thesis, Computer Graphics Group, Saarland University, Germany, 2006.

[2] S. Boulos, I. Wald, and P. Shirley, "Geometric and Arithmetic Culling Methods for Entire Ray Packets," University of Utah, 2006.

[3] A.Y. Chang, "Theoretical and Experimental Aspects of Ray Shooting," PhD thesis, Polytechnic University, 2004.

[4] V. Havran, "Heuristic Ray Shooting Algorithms," PhD thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University, 2000.

[5] C. Lauterbach, S.-E. Yoon, D. Tuft, and D. Manocha, "RT-DEFORM: Interactive ray tracing of dynamic scenes using BVHs," In Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing, 39-46.

[6] J. Lext, U. Assarsson, and T. Möller, "A benchmark for animated ray tracing," IEEE Computer Graphics Applications, 21(2): 22-31, 2001. http://www.ce.chalmers.se/BART/

[7] Gordon Stoll, William R. Mark, Peter Djeu, Rui Wand, and Ikriam Elhassan, "Razor: An Architecture for Dynamic Multiresolution Ray Tracing," Technical Report 06-21, University of Texas at Austin Dep. of Comp. Science, 2006.

[8] I. Wald, "Realtime Ray Tracing and Interactive Global Illumination," PhD thesis, Computer Graphics Group, Saarland University, Germany, 2004.

[9] I. Wald and V. Havran, "On building fast kd-trees for ray tracing, and on doing that in $O(N \log N)$," In Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing, 61-69.

[10] I. Wald, "On fast construction of SAH based bounding volume hierarchies," In Proceedings of the 2007 Eurographics/IEEE Symposium on Interactive Ray Tracing.

[11] I. Wald et al., "State of the Art in Ray Tracing Animated Scenes," Computer Graphics Forum 28(6), pages 1691-1722, 2009.

[12] S.-E. Yoon, S. Curtis, and D. Manocha, "Ray tracing dynamic scenes using selective restructuring," In SIGGRAPH'07: ACM SIGGRAPH 2007 sketches, page 55. ACM, 2007.