REAL TIME RAY TRACING

A SEMINAR REPORT

Submitted by

BEDABRATA BORA

(1706123)

In partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

At

SCHOOL OF COMPUTER ENGINEERING

KIIT Deemed to be University
BHUBANESHWAR

March 2021

School of Computer Engineering

KIIT Deemed to be University, Bhubaneswar, Odisha, 751024, India.

CERTIFICATE

This is to certify that the **SEMINAR REPORT** entitled Real Time Ray Tracing, is a bonafide work done by Bedabrata Bora (1706123) in partial fulfillment for the requirement for the award of the degree of Bachelor of Technology in Information Technology

Dr. Tanmay Swain

Seminar Supervisor

ABSTRACT

Real time ray tracing produces a high-quality image at interactive frame rates. Though the domain of real-time graphic rendering is dominated by raster graphics, ray tracing technology has started becoming more powerful. This report aims to address the importance of real-time ray tracing and various acceleration techniques that yield interactive performance. It also discusses the specialized hardware used to achieve ray tracing efficiently. Compared to raster graphics, ray tracing allows the synthesis of a highly realistic image and it scales them well for massively complex scenes. Taking into account these advantages and the fact that real-time raytracing software performance on PCs has finally surpassed the performance of dedicated rasterization hardware for highly complex models, all indications point that ray tracing may soon replace rasterization-based model rendering for scientific visualization and games in the near future.

TABLE OF CONTENT

1. INTRODUCTION

Ray tracing is an algorithm that supports highly realistic image synthesis. For this, light rays are cast from the eye point of view through the pixels of an image plane and intersected with the scene geometry. The shortest distance from the image to and intersection determines the visibility of the scene through that pixel. The brightness and contrast of that pixel is an estimate of the irradiance emitted from the found intersections through that pixel towards the view point. It is gathered by shooting more rays from that point into the source of light and weighting their energy by the point's bidirectional reflectance distribution function. Two major methods are used for the estimation of irradiance from the scene: i) Whitted-style Ray Tracing (or simply referred to as ray tracing) ii) Global Illumination. For the former method, the estimate only includes direct illumination, perfect reflection and refractions. The later method for simulation is more realistic than the former. It accounts for the diffuse inter-reflection from the complete scene arriving in that point, describes by the rendering equation stated by Kajiya [1986].

Due to its high computational cost, ray tracing has not been considered for real time rendering for a long time. But in recent times, with the increase in computation power of CPUs, interactive ray tracing has become a reality and offered a number of benefits over traditional rasterization-based algorithm.

There are many approaches to achieving interactive frame rates for ray tracing. The most successful ones accelerate those parts which have the highest computational cost. Moreover, the inherently parallel nature of ray tracing can be massively exploited using SIMD (Single Instruction/Multiple Data) instructions on modern CPUs by distributing workload in PC clusters or by using dedicated highly parallel ray tracing hardware. The effectiveness of parallelization can be even increased by taking advantage of coherence between rays. These techniques with cache-awareness can improve the performance of ray tracing by several magnitudes.

This report explores the computational cost of different parts of a ray tracing algorithm. A very important class of optimization are represented by spatial acceleration structures which are discussed in this report. It explores the necessity of parallelization and caching to effectively take advantage of coherence. Last but not the least, the advantages and limitations of real-time ray tracing in comparison to rasterization is discussed.

2. COMPUTATIONAL COMPLEXITY

In order to maximize optimization profits, one needs to carefully analyze the computational complexity of the ray tracing algorithm. The most expensive operation is the search of ray-generating intersection. To reduce the number of primitives to be intersected, acceleration structures like KD-trees are used. They typically reduce the complexity for finding a small set of potentially intersecting geometric primitives to be $O(\log n)$ if the number of primitives is n. Yet the tree traversal operations are the most frequent operations per pixel. A carefully chosen acceleration structure and a scene traversal algorithm that is optimized for that structure are most essential.

Computing ray-geometry intersection points is also very costly because it happens quite often per pixel. The number of intersections to be calculated depends on the spatial resolution of the search structure. The higher the resolution the deeper the search tree. In other words, there is always a trade-off between the number of traversal operations and the number of intersection calculations which can be controlled by adjusting the maximum number of geometric primitives in a volume element of a scene acceleration structure. Typically, the number of intersections calculations is less than the number of accelerations structure traversal operations for a given pixel.

In contrast, the computational complexity of texture mapping and shading is insignificantly compared to the complexity of the other operators since it has to be done only once for a found intersection point. From this analysis, one can tell that optimization of the scene traversal and reduction of ray-geometry intersections yields the most performance payoff.

3. SPATIAL ACCELERATION STRUCTURES

Spatial acceleration structures have the most impact on rendering performance because they restrict the potential visible set for a given ray to a very small number of primitives independently of the overall size of the scene. This makes visibility queries by shooting rays into the scenes very efficient.

A spatial indexing structure is a tree of voxels (volumetric pixels) or bounding volumes. Each volume element is the tree may contain some sub-elements and a list of geometric primitives and their shading information. Intersecting a ray with the scene means traversing the acceleration structure by intersecting the rays with the children of a bounding volume recursively until the ray hits an opaque primitive or an empty leaf of the tree.

Following are some of the acceleration structures that have been successfully used for real time ray tracing.

3.1. The Kd-Tree

The kd-Tree or k-dimensional tree, is a space partitioning data structure for organizing objects in a k-dimensional space. It is a special case of the BSP (Binary Space Partition) tree. A kd tree uses only splitting planes that are perpendicular to one of the coordinate system axes. This differs from the BSP trees, in which arbitrary splitting planes can be used. Building a static kd tree from n points takes O(*n log n*) time and orthogonal range search takes O(*log n)* where *n* is the number of cells in the kd tree.

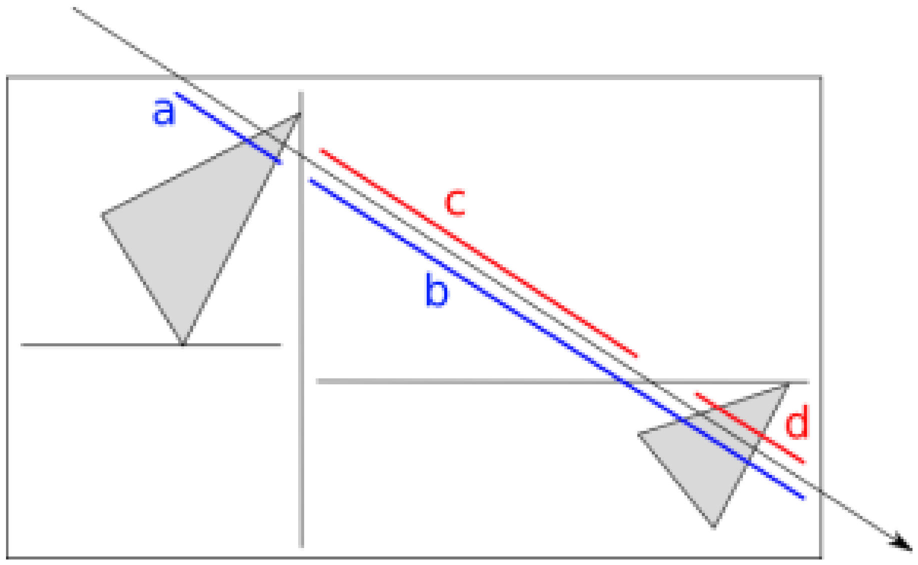

Fig 1: Ray intersection with a kd-tree. The rays intersecting with the top-level splitting plane and is therefore split into the ranges *a* and *b*. First, the left tree is checked recursively against *a*. The right tree is checked against *b* only if *a* does not hit any triangle. In that case *b* is split up to *c* and *d*. Since the upper cell is empty, *c* is immediately discarded and *d* is tested against the triangle in the lower cell

As a consequence of the a-priory unknown number of references in the kd-tree, memory management becomes an issue during the construction of the hierarchy. For efficient creation of the data structure complete pre-allocation would be most efficient. There are heuristics to predict memory size of a kd-tree but they do not work well for arbitrary scenes.

Another general disadvantage of the kd-tree is its inability to be adjusted efficiently for dynamic scenes. Complete or partial rebuilding of a kd-tree for every frame is too costly for complex scenes. However, it is possible to overcome this problem by using a 2 level kd tree. The scene acceleration structure is divided in low level kd tree representing the objects which reside in a high-level kd tree for the scene. The low-level structures for animated objects can be rebuilt if necessary. In some cases where the animation of whole objects can be described by an affine transformation, there is no need to rebuild the low-level tree. Instead, the intersecting rays are transformed with the inverse transformation.

3.2. The Bounding Volume Hierarchy

A Bounding Volume Hierarchy is a tree of bounding volumes where each bounding volume stores references to child nodes and each leaf node has a list of geometric primitives. In addition to that, the bounding volume is guaranteed to entirely enclose the bounding volumes of its descendants.

In contrast to spatial subdivision structures such as the kd-tree, the bounding volume hierarchy divides the object hierarchy, and a given object hierarchy is more robust than a given sub division of space, Bounding volume hierarchies differ from space partitioning data structures like kd-tree, in that overlapping bounding volumes are allowed and empty space is explicitly represented. Due to the fact that overlapping bounding volumes are allowed, a bounding volume hierarchy can be quickly updated for every frame without invalidating the whole structure. This advantage makes them very well suited for dynamic scenes. Moreover, it can be built very efficiently because the memory requirements can be bounded a priori.

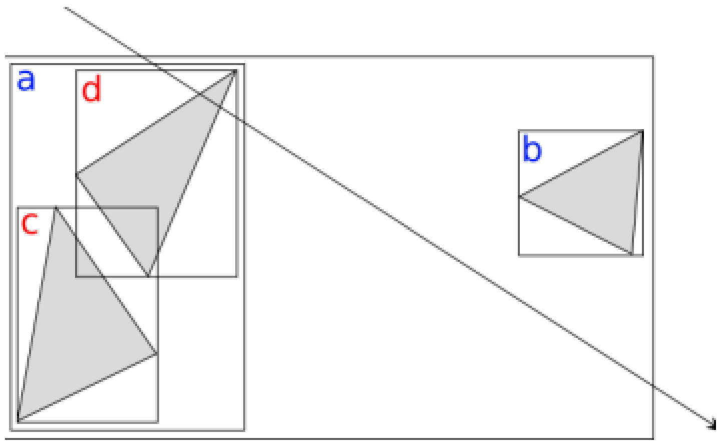

Fig 2: Ray intersection with traditional BVH. The ray has to be intersected with boxes *a* and *b* because they are not ordered. Since the ray overlaps with a, it is tested against *c* and *d* too.

The biggest disadvantage of bounding volume hierarchies is that unlike space partitioning structures, the child volumes are not spatially ordered, therefore it is not possible to abort the intersection procedure early on first hit. Even if the ray hits a child, it is not guaranteed that there are no other sibling volumes which would yield intersections in front of it. This is why bounding volume hierarchies perform worst of all spatial acceleration structures for visibility queries. Despite this, recent research shows that bounding volume hierarchies can perform as well as kd-trees.

3.3. The Bounding Interval Hierarchy

The bounding interval hierarchy is a crossover of space partitioning and bounding volume hierarchies combining the advantages of both. Unlike bounding volume hierarchies, which store full axis-aligned bounding box for each child, the idea of the bounding interval hierarchy is to only store two parallel planes perpendicular to one of the axes of the parent volume. The bounding interval hierarchy as acceleration structure for real-time raytracing has been shown to outperform other fast acceleration structures significantly.

The efficiency of bounding interval hierarchy is due to the advantages inherited from space partitioning structures, namely ordered traversal which allows early exit. Opposite to space partitioning, bounding interval hierarchies have a fixed pre-allocation size depending on the number of primitives.

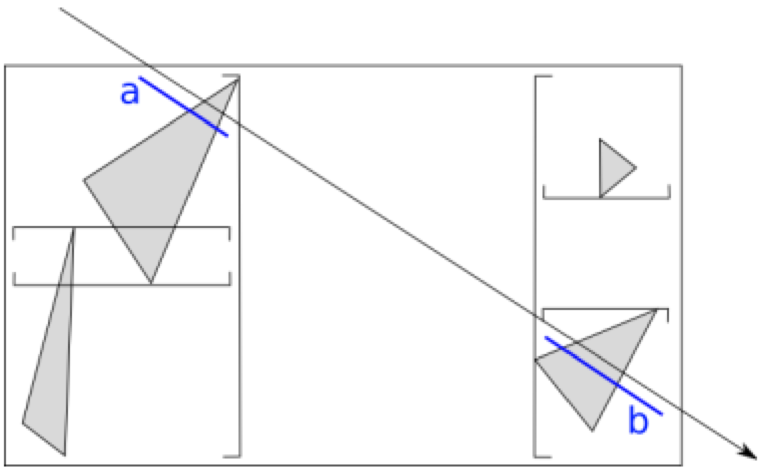

Fig 3: Ray intersects with a bounding interval hierarchy. The ray intersects both vertical split planes and is divided into the ranges *a* and *b*. *a* is tested against the left tree first by starting with the top triangle because *a* is entirely before split planes with respect to the vertical axis. If that test yields no intersection point *a* will not be tested against the lower triangle. Then *b* is tested against the right tree by testing against the lower triangle immediately because *b* is entirely after second split plane.

All the modern interactive ray tracers exploit coherence between rays to improve scene traversal and ray-geometry intersection performance. Together with good acceleration structures the traversal algorithms that take advantage of coherence between rays are responsible for the dramatic speed up that makes real-time performance possible.

4. RAY TRACING ON PHONE

The simple answer is yes. The GPU in today's smartphones has made massive strides since they first launched, not only in terms of feature-set but also in real-world achievable performance. Indeed, premium smartphones are already breaching the 1 TFLOPS compute barrier, previously the exclusive arena of dedicated gaming consoles. The real question in play here then is efficiency. Smartphones depend on battery life, and as ray tracing is more efficient than traditional rendering methods it stands a good chance to be quickly added to the mobile experience.

Using the innovations described above, Imagination is making efficient ray tracing possible. In smartphones, the cost of faking shadows and reflections in games with traditional rasterization is already very high. In a modern game engine, such as Unity or Unreal, reflections are generated using cascaded shadow maps. This requires rendering the geometry of the screen many times and writing out shadow map lookup tables to memory, all of which costs cycles and bandwidth, consuming significant GPU and system power.

5. REALTIME RAY TRACING versus REALTIME RASTER GRAPHICS

The difference between dater and ray traced graphics can be discussed by comparing several aspects of both:

5.1. Realism

Rasterization accounts for direct illumination only. Shadows and reflections can be faked but their realism is very limited. In particular self-reflections of a concave object are not possible. Physically correct refractions are not possible either with pure rasterization. On the contrary real-time ray tracing features correct pixel-accurate shadows from point lights, soft shadows from area lights, correct display of reflecting materials and physically correct refractions3 (i.e. glass, water). The rasterization pipeline can handle translucent materials but requires to render them back to front which involves a sorting overhead. Altogether real-time ray tracing offers a whole lot of more realism and physical correctness for material appearance and lighting.

5.2. Scene Complexity

The most important advantage of ray tracing over rasterization is its logarithmic complexity in the number of geometric primitives, whereas the complexity of raster graphics scales only linear. As a result, massive scenes consisting of billions of triangles can still be ray traced with interactive frame rates. This is due to the logarithmic complexity of search algorithms in scene acceleration trees. Actually, real-time ray tracing has built-in visibility Similar techniques are used with raster graphics to reduce the workload of the GPU but the user has to take care of it. With real-time ray tracing all the advanced and highly complex visibility determination algorithms currently needed for real-time rasterization of complex scenes are no longer necessary.

5.3. Computation power

To some extent, ray tracing requires significantly higher computation power than rasterization for scenes of low complexity because rasterization interpolates lighting information between vertices across the pixels in a triangle in image space while ray tracing evaluates lighting per pixel. Then again for very complex scenes, where most of the triangles are smaller than a pixel, raster graphics are not considered to be significantly faster. To deliver a high frame rate on a single PC it is clear, that real-time ray tracing hardware is necessary.

5.4. Ease of Use

Realtime ray tracing relieves application programmers from a lot of tedious optimizations such as view-frustum culling which are necessary for today’s applications of raster graphics. Also, there are not so many performance limiting bottle-necks that impose restrictions on the content to be rendered as for the rasterization pipeline. On top of that, users are relieved from writing complicated algorithms that fake shadows, reflections, etc.

5.5. Dynamics

Limited support of dynamic scenes has long been the biggest disadvantage of real-time ray tracing systems compared to raster graphics. Despite this, better solutions have been found recently to support real-time ray tracing of dynamic scenes in general and skinned animations in particular.

5.6. Geometric scene description

The only graphics primitives that can be handled by rasterization pipelines are triangles. By contrast the list of geometric primitives that are supported by real-time ray tracing systems (without triangulation or other modifications) is quite large: triangles, freeform surfaces, point based surfaces, Bezier patches, volume data in all kinds of grids etc.

6. HARDWARE ACCELERATION

Software based ray tracing is decades old and movie makers have been using it for a long time now. But until recently, raytracing algorithm has not been used to their full potential. With the emergence of specialized hardware like the RT cores built into NVIDIA’s Turing architecture, makes a huge difference if ray tracing is being done in real time. The use of GPUs to accelerate ray-tracing algorithms gained fresh momentum in 2018 with the introduction of Microsoft's DirectX Raytracing API. Introduced in 2009, NVIDIA OptiX, target design professionals with GPU-accelerated ray tracing. Over the decade, OptiX rode the steady advance in speed delivered by successive generations of NVIDIA GPUs. In 2015, NVIDIA demonstrated how ray tracing could turn a CAD model into photorealistic images, indistinguishable from a photograph in seconds, speeding up the work of architects, product designers and graphic artists.

Following are some of the most popular GPUs that currently support day tracing:

1. Nvidia Geforce Rtx 3080
2. Nvidia Geforce Rtx 2080 Ti
3. Nvidia Quadro Rtx 6000/8000
4. Nvidia Geforce Rtx 3060
5. Nvidia Geforce Rtx 2060

7. CONCLUSION

For a long time, researchers have argued that ray tracing is superior to rasterization in various points, but nobody would have imagined that ray tracing could be applied so effectively for real-time rendering. As recent software real-time ray tracing systems become much more powerful than high-end rasterization hardware, it seems like the advent of a new age for real-time graphics. Hardware solutions are not yet as optimized and feature rich as rasterization hardware, though. Only the future can tell if real-time ray tracing hardware will eventually become good enough, so that ray tracing may supersede rasterization as the major real-time rendering paradigm.

We have seen, that kd-trees are not primarily the fastest acceleration structures anymore and that bounding volume hierarchies as well as the bounding interval hierarchies are less restrictive for animated geometry. The analysis of the typical computational complexity of a ray tracer motivates the hypothesis that the use of a well-built scene acceleration structure combined with a carefully optimized scene traversal algorithm are most effective in order to increase the frame rates of real-time ray tracers.

Comparison of real-time ray tracing versus raster graphics showed that as far as realism and scalability are concerned real-time ray tracing is more than competitive to rasterization hardware but ray tracing hardware is not yet ready for the prime time. In near future, when real-time ray tracing has been established in the real-time rendering domain the development of interactive graphics applications will be significantly simplified because faked shadows and reflections, complicated rasterization-pipeline-aware optimizations and advanced view-frustum-culling algorithms will not be necessary any more.

REFERENCE

1. https://blogs.nvidia.com/blog/2018/03/19/whats-difference-between-ray-tracing-rasterization/
2. Havran, V. 2001. *Heuristic Ray Shooting Algorithms*. Ph.d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague.
3. Jansen, F. W. 1986, *Data structures for raytracing. Springer-Verlag New York Inc., New York, NY, USA, 57-53.*
4. MacDonald, J.D and Booth, K.S. 1989. Heuristics for ray tracing using space subdivision. In *Graphics Interface*, 152-163.
5. https://medium.com/gametextures/2020-the-year-of-real-time-ray-tracing-f8b29e89523b