ORIGINAL RESEARCH PAPER
Volume - 11 | Issue - 06 | June - 2022 | PRINT ISSN No. 2277 - 8179 | DOI : 10.36106/ijsr
INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH

RAY TRACING IN 3D COMPUTER GRAPHICS

Engineering

Veena Sanath Kumar* Assistant Professor, Dept of ECE, Acharya Institute of Technology. *Corresponding Author

Dr. Rajeswari Prof and Head, Dept of ECE, Acharya Institute of Technology.

ABSTRACT

Ray tracing is a rendering approach in 3D computer graphics that generates an image by tracing the course of light as pixels on an image plane and modeling the impact of its meetings with virtual objects. Traditional scanline rendering methods can provide greater visual realism than this technique but at a higher computing cost. As a result, ray tracing is best suited to applications with extended render times, such as still computer-generated graphics and visual effects in film and television (VFX). It is less appropriate for real-time applications, such as video games, where the frame rate is crucial [1]. Real-time ray tracing hardware acceleration has become standard for commercial graphics cards and APIs, allowing game developers to implement real-time ray tracing algorithms. Ray tracing can simulate optical effects such as reflection, refraction, scattering, and dispersion (chromatic aberration). Path tracing, a type of ray tracing, can provide soft shadows, depth of field, motion blur, caustics, ambient occlusion, and indirect lighting [3]. The paper discusses implementing a Ray-tracing model considering the optical effects -reflection, refraction, and shadow. The Ray tracing algorithm for a test scene of 6 spheres is verified and the performance measured with a 182.11 Mrays / sec.

KEYWORDS

Ray Tracing, 3D Computer Graphics, Virtual Image Rendering,

INTRODUCTION

Light rays begin from a light source and refract or reflect off objects in their path in real life. Some of these rays finally end up in a spectator's eyes, which assembles

Figure 1: Ray tracing reflective colored spheres demonstrates the effects of shallow depth of field, light sources, and diffuse interreflection.Ray tracing is an image rendering approach in which the path of individual light rays is traced from the viewer to each pixel in the picture, refracting or reflecting off objects in the scene and finally ending at the light source. It is a rendering algorithm that can create realistic visuals, as shown in figure 1. This is accomplished by tracing light through the image plane and copying its interaction with the scene's objects. A scene describes the things to be rendered and the lighting and camera viewpoint. The ray-tracing approach may replicate optical effects such as reflection, refraction, scattering, and dispersion.

Even though ray tracing produces extremely high-quality photorealistic images, it is not widely used in real-time rendering applications such as modeling software and video games because it is computationally intensive when written in software, and frames do not render quickly enough at high resolutions.

LITERATURE REVIEW

Ray Tracing

Ray tracing has been the subject of study for more than four decades. The ray-tracing algorithm was first proposed by Appel [6]. The ray casting method was the first iteration, as it simply shoots rays from the eye and then analyses the intersection. It merely allowed for the projection of a rough representation of the scene. This technique is often used to overcome the visibility problem. Whitted [7] proposed a ray-tracing technique to implement more complicated illumination effects. When a central ray collides with a primitive scene, it can produce secondary rays such as reflection, refraction, and shadow. Those rays can then be handled recursively.

The Whitted algorithm [7] is the most popular ray-tracing approach for real-time applications. Earlier works have a large amount of GPU ray traversal implementations. Purcell et al. [8] demonstrated how a GPU ray tracing pipeline might use a stream programming paradigm. They expanded their ray tracer into stream programming using a Uniform Grid as an acceleration framework.

Acceleration Structures

CUDA allowed for parallel raytracing, which saved time by eliminating repetitive resurfacing intersections. To get around this problem, acceleration structures are used. Scene objects are divided into bins using the appropriate techniques like the ray-surface intersection. The ray-surface intersection for the objects discovered within the group can be skipped if the ray does not touch the bin. This reduces the number of redundant ray-surface intersections. The development in the building and traversal of acceleration structures is the driving force behind the real-time rendering of computation-intensive ray tracing.

Spatial subdivision and object subdivision structures are the two most common acceleration structures. Furthermore, as demonstrated in Figures 2 and 3, both are hierarchical based on the implementation level.

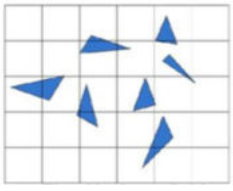

Figure 2: Flat Uniform Grid

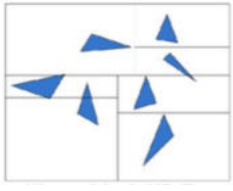

Figure 3: Hierarchical KD-Tree

Instead of ray tracing acceleration hardware, an instruction processor develops acceleration systems for static and offline rendering applications [9].

However, as the number of applications for rendering dynamic scenes grows, there is a greater necessity for online development and maintenance of the acceleration structure. Recent advances in completely data-parallel construction algorithms have also enabled the development of effective construction hardware [10]. Vaidyanathan et al. [12] examined the ray-tracing accuracy requirement and proposed a reduced accuracy approach now implemented in GPUs. Li et al. [11] propose a ray tracer that uses a highly efficient, entirely parallel building approach that applies to Kd-tree and BVH (Bounding Volume Hierarchies) and a leading-edge ray traversal procedure.

RAY TRACING: SOFTWARE ACCELERATOR

Introduction

The essential components of the ray tracing algorithm are depicted in Figure 4. A primary ray is cast from the camera point across a pixel in the image. Intersection tests are run on the scene's objects to find the closest intersecting object. A shadow ray is traced towards the light source to assess whether an object is in shadow. The affected object is

subsequently subjected to a local lighting model. Reflection and refraction rays can be generated and tracked in the same way in recursive ray tracing. Each ray's contribution is totaled, and the final pixel color is returned.

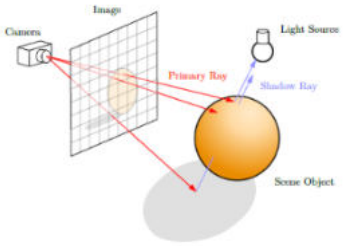

Figure 4: The ray tracing algorithm builds an image by extending rays into a scene

Since 1980, the recursive ray tracing algorithm has been used. Since then, tremendous effort has improved ray tracing efficiency and feature set. The focus of the study is on reducing the algorithm's processing timing in scenarios with a large number of objects. The algorithm's most straightforward implementation seeks to intersect any ray with all objects in the scene. For large scenes, this is inefficient and has been handled by separating the scene objects into data structures of inexpensive objects on which intersection tests can be performed. The number of intersection tests required is significantly reduced with the help of data structures.

Rendering converts a three-dimensional virtual scene into a two-dimensional image in computer graphics. Nowadays, rendering is used in many computer graphics applications, including computer games, visualization, and graphical effects in film productions. Rasterization-based rendering and ray tracing-based rendering are the two basic rendering techniques classified depending on the rendering methodology.

Ray tracing has been used exclusively for high-quality rendering for decades, and it has a reputation for taking a long time to render. As a result, ray tracing has only been used for off-line rendering. On the other hand, Rasterization-based hardware rendering has dominated the field of interactive rendering.

Without a doubt, ray tracing's low performance had been a significant cause of its non-existence in interactive rendering. Ray tracing has long been thought to be quicker than rasterization-based rendering due to its logarithmic behavior in scene complexity; nonetheless, its performance has never been impressive. Even ray-tracing performance might reach interactivity if enough parallel compute capacity was available. Unfortunately, sufficient computing capacity necessitated the use of a large-scale supercomputer.

In recent years, researchers have refocused on ray tracing performance, mainly focusing on off-the-shelf technology. Regarding the advantages and shortcomings of present hardware architectures, they focused on an efficient and optimized implementation of the ray tracing algorithm and associated data structures. Ray tracing's performance has been raised to an interactive level despite running on off-the-shelf hardware with an effective combination of algorithmic and hardware-specific enhancements.

Ray tracing implementation is available for a range of architectures, including GPUs and specialized hardware. Because GPUs are either too slow or not generally available, existing CPU architectures are the most appropriate hardware platform (custom hardware).

The fundamental disadvantage of utilizing processor-specific optimized algorithms for high-performance ray tracing is that they are difficult to define, especially at a high level.

The latest techniques and algorithms for real-time ray tracing on current processor architectures are presented in this study. When producing this paper, the goal was to avoid taking a strictly high-level approach and instead provide precise and extensive information about low-level implementation challenges and optimizations. For each illustrated algorithm, an example code is included where possible.

The Main Contributions

SIMD-optimized algorithms for traversing coherent beams. Comprehensive analysis and SIMD-optimized implementation of intersection techniques for rays with various geometric primitives such as triangles and bicubic Bezier patches. An algorithm for dealing with entirely dynamic scenarios involving the quick generation of spatial index structures. A parallelization solution for ray tracing with linear scalability in the number of linked computing nodes. A ray tracing-based system also allows for interactive global illumination computation. Even though optimizing takes time and requires significant integrity, it can sometimes be necessary to bring algorithms to a new level of speed.

Summary

Raytracing is a growingly popular image creation technology. The way of coherence appears to have an advantage over approaches commonly referred to as "beam-tracing" [13], which may be helpful in certain sorts of scenes. However, it lacks ray-generality tracings and flexibility. Finally, new illumination models were created, allowing for even better image optical quality. Distributed ray tracing [14] is closely connected to ray tracing. Distributed raytracing enables the creation of depth of field and motion blur at nearly the exact computational cost as traditional anti-aliasing. Diffuse interreflection and refraction from primary light sources are possible with the radiosity technique.

Raytracing works well for pipelined processing on vectorial supercomputers and systolic computing. There are other contributions to the parallel raytracing topic.

IMPLEMENTATION METHODOLOGY

A flexible and easy ray tracing algorithm is developed and tested using a MATLAB simulator. The code has been reduced to a single-ray (i.e., pack-less) raycaster with a bounding volume hierarchy (BVH) as the acceleration structure. Figure 5 shows further information about this code.

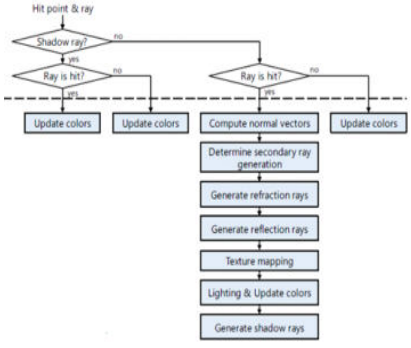

Figure 5: Design Flow of Ray Tracing Simulation

One frame is rendered in one core from scratch for each simulation using empty caches. Because the instructions do not change from frame to frame, they are already in the instruction cache. As a result, the findings appropriately depict altering the scene information on every frame and necessitating cache invalidation. The results are conservative because much of the scene may remain the same from frame to frame and remain in the cache even in a dynamic scene. The time is taken to render a single frame is calculated and multiplied by the number of frames per second. Figure 6 shows the rendered frame.

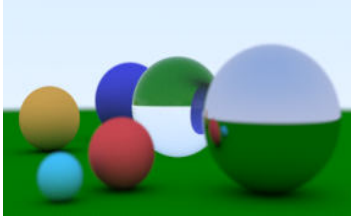

Figure 6: Simulation Result Table 1: Time bounds for shaded image generation.

Table 1: Time bounds for shaded image generation.

	Depth Buffer Strategy	Visible Surface Strategy	Ray Tracing Algorithm

Shading (Incl. Highlights)	i Polypixel(pi) + n + L · mx · my	(n + k) · logn + L · mx · my	(n + k) · logn + (L + logn) · mx · my
+ Correct Generation Of Shadows		(n + k) · logn + L((n + k) · logn + mx · my)	(L + 1) · logn · ((n + k) + mx · my)
+ Correct Modelling Of Mirrors, Glass			P (n) + mx · my · (2D − 1) · RI(n)

mx: pixels in the x-axis
my: pixels in the y-axis
n: polygon edges, polygons are triangles
k, k: scene dependant maximum 2D intersections
L: Sources of light Count
Polypixel(pi): Pixels intersected by polygon pi in screen space
P (n): preprocessing time for general ray tracing
D: depth of ray tracing
RI(n): time-bound for ray query problem

Table 2: Simulation Results

Test Scene	Cache Hit Rate	Bandwidth	Performance (Mrays/sec)
6 Sphere	96.76	1.9	182.11

CONCLUSION

A Ray tracing model incorporating shadowing, reflection, and refraction effects for generating nonmetallic spheres has been implemented. This implementation aims to improve performance while sharing the multiplier resource restriction and renders a 320 X 240 scene featuring a plane and a sphere. Further work includes optimizing the design using parallel computing and implementing using Xilinx FPGA.

REFERENCES

[1] Henrik. File: Ray trace diagram.SVG. Apr. 2008. URL: https://en.wikipedia.org/wiki/File:Ray_trace_diagram
[2] Kevin Suffern. Ray Tracing from the Ground Up. A K Peters/CRC Press, 2007.
[3] C Erickson. Real-time Collision Detection. Morgan Kaufmann, 2005.
[4] A Keller, F Slomka. Fixed Point Hardware Ray Tracing. Universit¨at Ulm, 2007.
[5] M Pharr, G Humphreys. Physically based rendering: From theory to implementation. Morgan Kaufmann, 2004.
[6] Arthur Appel. Some techniques for shading machine renderings of solids. In Proceedings of the April 30{May 2, 1968, spring joint computer conference, pages 37{45, 1968.
[7] Turner Whitted. An improved illumination model for shaded display. In ACM Siggraph 2005 Courses, pages 4{es. 2005.
[8] Timothy J Purcell, Ian Buck, William R Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. In ACM SIGGRAPH 2005 Courses, pages 68{es. 2005.
[9] Jae-Ho Nah, Jeong-Soo Park, Changmin Park, Jin-Woo Kim, Yun-Hye Jung, Woo-Chan Park, and Tack-Don Han. T&i engine: traversal and intersection engine for hardware-accelerated ray tracing. In Proceedings of the 2011 SIGGRAPH Asia Conference, pages 1{10, 2011.
[10] Zonghui Li, Tong Wang, and Yangdong Deng. Fully parallel kd-tree construction for real-time ray tracing. In Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pages 159{159, 2014.
[11] Zonghui Li, Yangdong Deng, and Ming Gu. Path compression kd-trees with multilayer parallel construction a case study on ray tracing. In Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pages 1{8, 2017.
[12] Karthikeyan Vaidyanathan, Tomas Akenine-Moller, and Marco Salvi. Watertight ray traversal with reduced precision. In High-Performance Graphics, pages 33{40, 2016.
[13] Dadoun, N., Kirkpatrick, D.G., Walsh, J.P.: The geometry of beam tracing. 1. Symposium on Computational Geometry, Baltimore, 1985, 55-61
[14] Lee, M.E., Redner, R.A., Uselton, S.P.: Statistically optimized sampling for distributed ray tracing, CG 19 (1986) 61-68