Master Project
Computer Graphics: Rendering Track

Alhajras Algdairy
Advisor: Prof. Dr.-Ing. Matthias Teschner

September 3, 2021

Acknowledgment

I want to express my special thanks of gratitude to Prof. Dr.-Ing. Matthias Teschner, who guided me through my master's program in general and in this master's project in specific. This project gave me the golden opportunity to dig into an exciting topic in computer graphics that I am keen on. The flexibility in research made me learn more about the related topics I am interested in, in rendering. Reading scientific topics and reviewing codes from different repositories expanded my expertise, where I had a full responsibility to control my time and resources, to learn more about the research process.

Abstract

This report investigates four different acceleration data-structure methods for implementing a simple raytracer on CPU. These methods are KD-tree, Uniformgrid, BVH, and LBVH. The report aims to compare the different approaches and their impact on the raytracer rendering time. The raytracer used in this project builds upon a previous lab project where the foundation of the raytracer is already implemented; however, without using a sophisticated data-structure to enhance the rending performance; hence the focus of this report will be how to improve the raytracer performance by using data-structures. ***Keywords: Ray Tracing; kD-Tree; BVH; LBVH; Uniformgird; Data structure;***

1 Introduction

This report summarises my journey to implement a simple raytracer focusing on data structure level, where the performance of rending a scene will be the lion's share of the report. Theory, implementation, and results will be discussed in depth in the report, where an introduction and motivation of what is raytracer and how to implement it will be briefly discussed; because data structures are more interesting for us, I will only explain the topics that are used in this raytracer.

2 What is Raytrcer and how it works

In Computer Graphics, scientists are interested in simulating a real-life phenomenon, such as Gravity Fraction, Rain, Snow, and the exciting part for us, light. Simulating light is arguably the most challenging part because light has always been difficult to characterize as it can behave as particles and waves, which makes it spread into the whole scene based on probabilities. This

makes it difficult to compute as it involves complex computation and infinite simulations to execute to have a perfect result without an error.

Why do we need to simulate light? Adding light into a scene will generate shadows, reflection, and refraction consequently will illuminate the scene, making the scene look like a reality, which can be helpful for some applications. For the internal designers, it is vital to simulate the final result of the lightning inside a room, such as sunlight coming from the windows, the light of lamps, and fireplace, these with react together and create shadows, before constructing the building and payloads of money, simulating the right angle and place of the room is helpful to imagine the final result. Solar engineers use 3D tools to build solar panel farms where the angle between the sunlight and the panel is essential to gather as much light energy as possible. In gaming, different companies compete to design engines that can produce natural scenes; this includes lighting and shadowing.

Figure 1, shows an example of rending a scene that can not be distinguished from reality, the details it catches as glossy materials, the reflection of surfaces, and the shadow. Capturing these details requires rendering techniques. Two popular methods are Rasterization and Raytracing. Raytracing outperforms rasterization in capturing more details; however this leads to performance issues, that is why most application uses Raytracing is offline rendering and not real-time rendering like games, where speed is vital to render each frame with compromising details.

Figure 1: https://www.pcgamer.com/unreal-engine-5-tech-demo-pc-performance/ The raytracing algorithm builds an image by extending rays into a scene and bouncing them off surfaces and towards sources of light to approximate the color value of pixels [Piotr Dubla, "Interactive Global Illumination on the CPU."]

2.1 Raytracing definition

So what is Raytracing? Raytracing is a rendering technique that provides highly lifelike lighting effects. In other words, an algorithm can track the source of light and then mimic how the light interacts with the virtual objects it eventually encounters in the computer-generated

environment. Raytracing produces far more lifelike shadows and reflections, as well as significantly enhanced translucence and dispersion. The algorithm considers where the light falls and calculates the interaction and interplay in the same way as the human eye does with actual light, shadows, and reflections.

2.2 Raytracing mechanism

Raytracing follows three main steps: *Casting Rays*: This is similar to how the eye works, however rather than eyes works as a sensor, we have a camera, we shoot rays from the camera to the scene, and we calculate which is the closest object it hits, its color and brightness. This is done for each pixel. *Path tracing*: Casting rays can solve the visibility issue; this means which object appears on the camera and at what position and color. However, if we want to simulate the following effects: soft shadows, depth of field, motion blur, caustics, ambient occlusion, and indirect lighting, then we need to trace the rays from or to the light source, this will accumulate the brightness of an object or model in a scene, also will give the depth of different objects. Note that the more rays and depth we need, the more quality we get but the slow the simulation becomes due to the complexity. *Shading*: In addition to tracing rays, we need a model to simulate different materials like Transparent, Glossy, and Diffuse. This is where the Shading phase is needed.

Raytracing has two main methods:

- *Forward Raytracing*: The light particles (photons) are tracked from the light source to the object via Forward Raytracing. While forward ray tracing is the most exact method for determining the color of each object, it is also the most inefficient.
- *Backward Raytracing*: An eye ray is formed at the eye in backward ray tracing, and it travels through the viewplane and out into the scene. If it hits an object, it will return it to the viewplane immediately. This method is more efficient than Forward Raytracing but less accurate due to reducing the rays used. In this implementation this method is used.

Computer graphics has three main pillars: *Modelling*, *Rendering*, and *Simulation*. The focus of this report is Rendering. Rendering cares about solving issues like the model's visibility towards sensors; this is done by two methods: *Rasterization* or *Raytracing*, this report is about a Raytracer implementation. Rendering also focuses on which color and intensity does a model has. How light interacts with surfaces and how it propagates throw the media.

3 Object subdivision

4 Spatial Subdivision

This lab was a great way to gain practical experience of the concepts we learned in our computer graphics course. I would like to thank **Prof. Dr.-Ing. Matthias Teschner** again for advising me throughout this lab. As a final remark, I would like to mention that I really enjoyed working on area lights. This includes debugging issues as I was working on them as well as how they were fixed. At the end, the results were worth the effort.

References