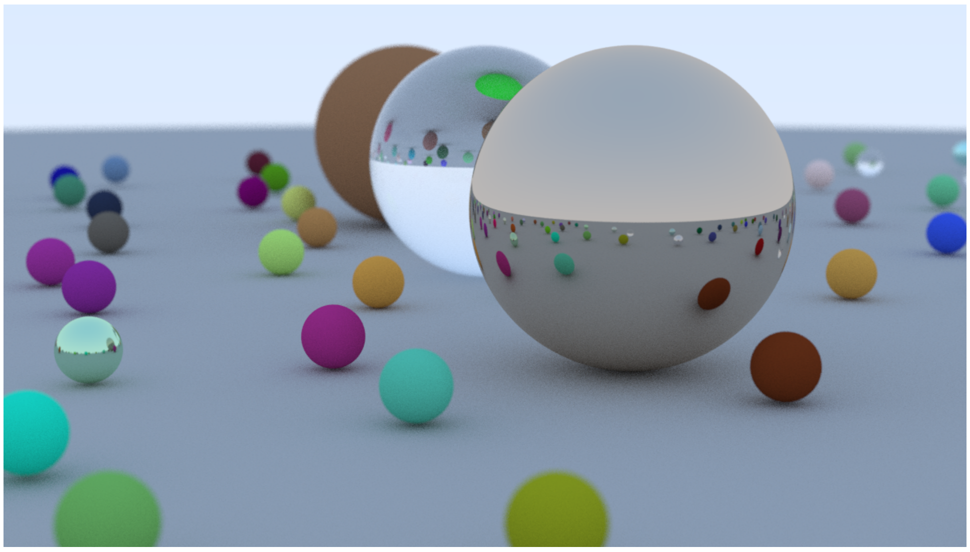
Raytracer from Scratch



Mentees:
Aaditya Joil
Rakshitha Kowlikar
Rudrakshi Kubde

Mentor:
Anish Mahadevan

SRA VJTI
September 1, 2024

Contents

Chapter 1

Introduction

A *Raytracer* is an application that aims to simulate graphics with realistic lighting, reflections and refraction of light using a technique called *Raytracing*.

1.1 Raytracing

It is a process of graphics rendering where the trajectories of multiple rays are calculated and the surfaces they hit determines the colour of the ray at that point. This results in real(ish) looking scenes with proper lighting and shading.

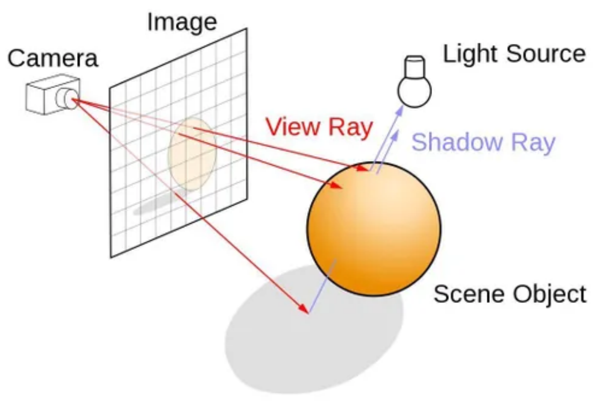


Figure 1.1: Raytracing

Usually we send the rays from the camera out into the world and calculate their trajectories and hits. This reduces a LOT of computation as in the real world, rays are emitted from the light sources and many of these rays never make it to our eyes.

Chapter 2

Methodology

Throughout the course of *Eklavya*, our team created 2 Raytracers. One which creates static PPM images and another which displays the pixels every frame. Both of them work in the same way:

1. Send out a bunch of rays into the scene.
2. Calculate ray trajectories and find out if they hit anything.
3. Calculate ray colours based on the materials they hit.
4. Display these colours at their corresponding pixel locations.

For the first static image generator, we only made use of the `C++` programming language and everything was being rendered using the CPU. This was fine as we were only generating Static PPM images.

If we think about it, the trajectory and colour of one ray should not affect any other ray at all, so we should take the load of the CPU and make use of the parallelism and S.I.M.D.(Single instruction multiple data) operations present in the GPU.

So for the dynamic frame-by-frame generator, we made use of the `OpenGL` library (more specifically the `C` API of the `OpenGL` specification) in order to offload the rendering operations to the GPU. We made use of *compute shaders* to perform the actual ray calculations.

Chapter 3

Weekly Updates

For every week here was the work which was completed:

- **24th June 2024 to 17th July 2024**
  The first three and a half weeks were spent in getting familiar with `Github`, `Markdown` and the `OpenGL` specification and solving a few basic rendering problems given by our mentor. We also followed the tutorial series of "The Cherno" in this time frame and watched through the "Essence of Linear Algebra" playlist by 3B1B.
- **18th July 2024 to 13th August 2024**
  The next month was spent in writing the static image Raytracer by following the book "Raytracing in one weekend" by Peter Shirley and adding the required code for being able to render more primitives apart from Spheres.
- **14th July 2024 to 23rd July 2024**
  The final weeks was spent in actually implementing the "Scratch" part of our Raytracer as all the required code and knowledge we gained was ported over to `OpenGL` during this time span. We also worked on trying to implement a very rudimentary GUI but it could not be integrated due to time constraints.

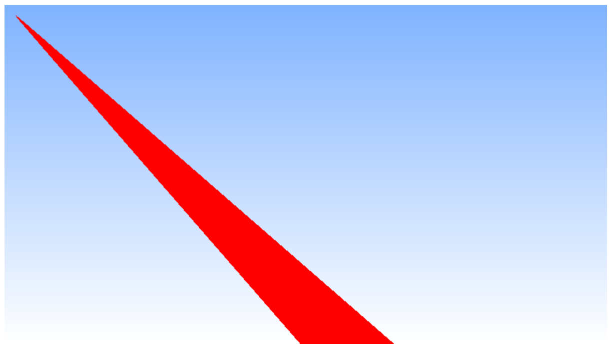
3.1 Problems Faced

Throughout the 2 months we faced various different problems, they are listed over here:

- *Non-rendering of pyramid* (Entire group)
  For the first few exercises involving rendering of primitives, we were

asked to render a 3-dimensional pyramid which was rotating about its axis. All three of us could not figure out how to achieve 3-dimensional rendering as we had only worked with 2 dimensions before this. This problem was solved after we learned about different coordinate spaces and how to project 3-dimensional points onto 2 dimensions.

- *Weird spheres* (Aaditya)
  While trying to write code to detect a sphere, Aaditya encountered an extremely weird issue. The sphere for some reason, was being rendered as a triangle.

  This issue took a long time to resolve was occurring due to an extremely silly bug. Aaditya had divided by the wrong values in the driver code causing the sphere to be stretched and squished in weird ways.

- *Frame jitter* (Aaditya)
  Aaditya encountered another bug just prior to the ending of *Eklavya*, there was a frame jitter being caused due to incorrect frame calculations.

- *Debugging the Scatter function* (Rakshitha)
  While debugging the `ray_scatter()` function in `World.glsl`, Rakshitha was encountering a lot of run-time errors due to the Shaders having logical errors. The logic was straightened out and the errors were rectified.

Chapter 4

Results

Here are a few images of the primitives we have added to the application.

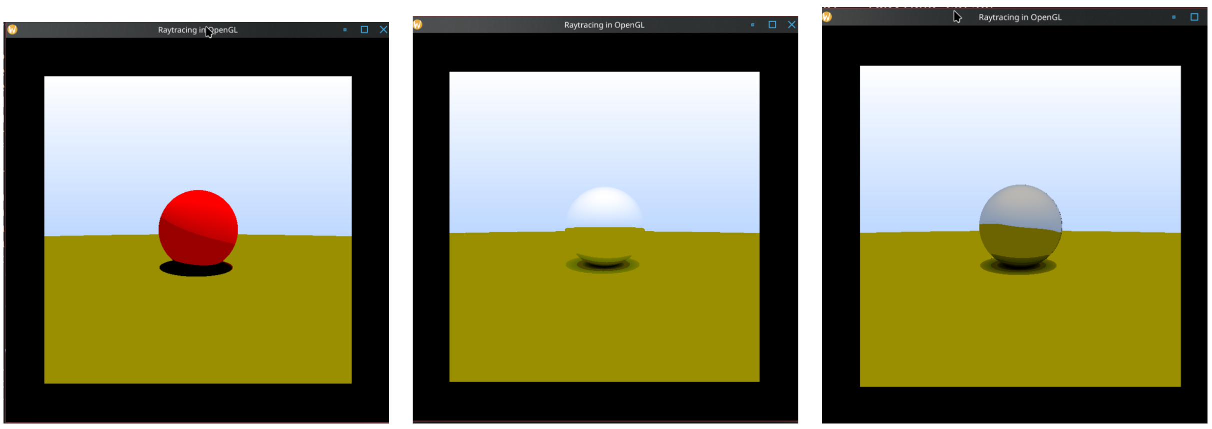


Figure 4.1: Spheres with different Materials

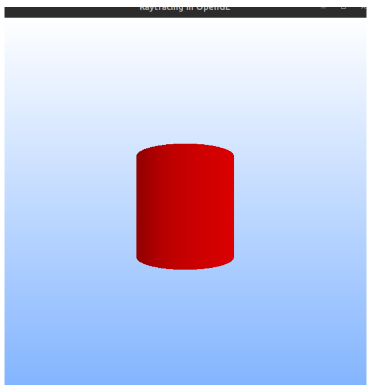
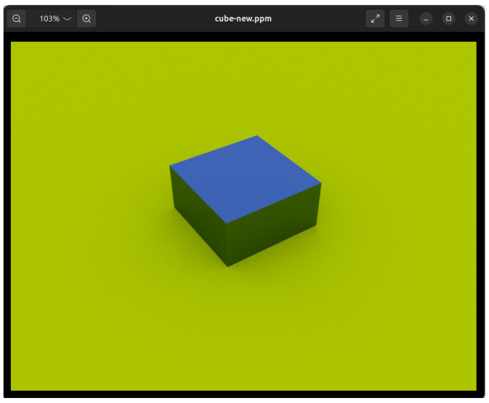


Figure 4.2: Cylinder with a Lambertian Material



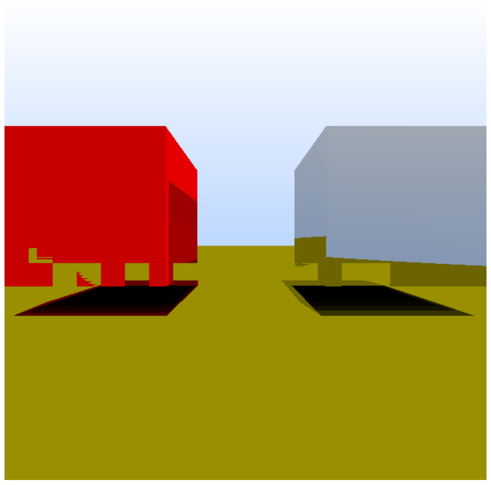


Figure 4.3: Lambertian and Metallic Cubes

We are also working on integrating a simple GUI into the application as well.

Chapter 5

Future Scope

We wish to improve our raytracer by making code optimizations and adding the following features to it:

1. Motion blur
2. Camera Movement and Graphical User Interface
3. Anti-aliasing
4. Physics

Thank you for your time.