Automagic's Raytracing Renderer

s2611168

Fourth Year Project Report
School of Informatics
University of Edinburgh
2023

Abstract

This project undertakes the creation of an innovative ray tracing renderer, melding conventional Blinn-Phong rendering with cutting-edge technology, specifically leveraging ChatGPT and analogous large language model (LLM) resources. Sponsored by Automagic, an AI-focused company, the primary objective is to craft an advanced renderer tailored for augmented reality applications. The core deliverables encompass a ground-up C++ implementation of a raytracer proficient in Blinn-Phong rendering, documented via a comprehensive report detailing the integration of LLM resources. Additionally, a demonstrative video showcases the renderer's capabilities on a customized scene. While the primary focus remains on the fundamental raytracer, the incorporation of more advanced features is encouraged by Automagic, offering potential enhancements to the renderer's functionality. The accompanying JSON scene serves as a benchmark, featuring example renders that visually depict the raytracer's adeptness. This project not only addresses the technical intricacies of ray tracing but also exemplifies the harmonious integration of traditional rendering techniques with modern AI-driven methodologies, fostering a foundation for future projects to incorporate AI seamlessly.

Table of Contents

Chapter 1

Introduction

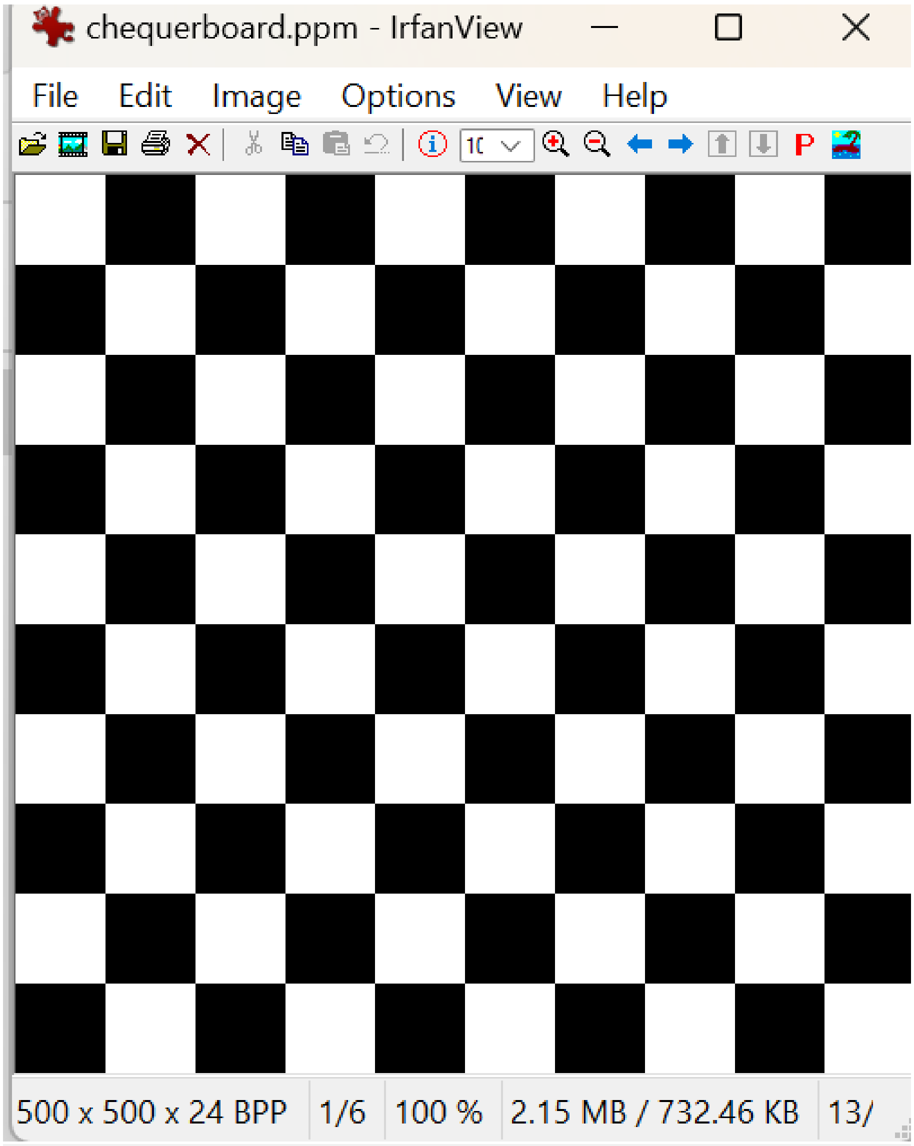
Embarking on the creation of a sophisticated ray tracing renderer, this project stands at the intersection of classical rendering methodologies and contemporary artificial intelligence (AI) advancements. Tasked by Automagic, a forward-thinking AI company, the goal is to construct a robust C++ raytracer, seamlessly integrating Blinn-Phong rendering principles with the capabilities of modern language models like ChatGPT. The fusion of traditional rendering techniques with AI-driven innovations offers a novel approach to crafting an augmented reality-centric renderer. This introduction sets the stage for a comprehensive exploration of the intricacies involved in developing the raytracer, elucidating the symbiotic relationship between conventional rendering practices and state-of-the-art AI resources. The project not only aims to meet the specifications outlined by Automagic but also aspires to lay the groundwork for a new paradigm in rendering technology.

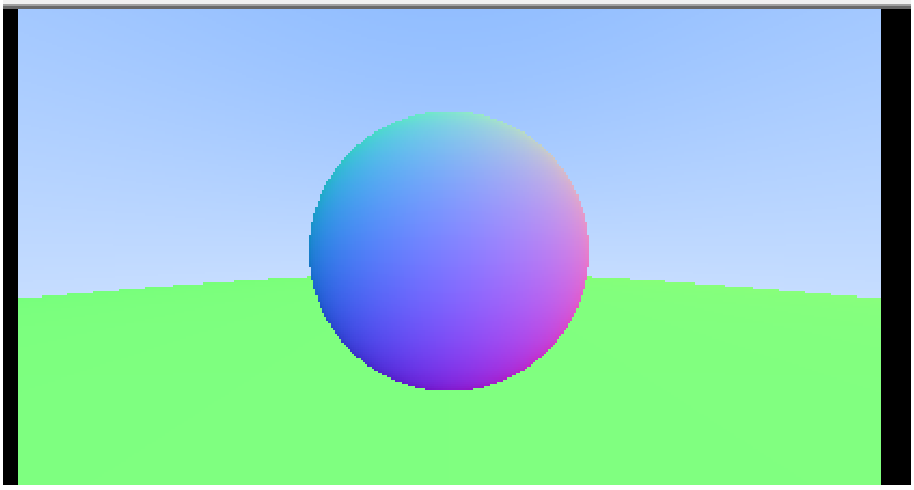
Chapter 2

Steps to the Final Project

2.1 Prior Coding

- Initiated the learning process with a YouTube tutorial, gaining insights into Blinn-Phong rendering calculations. blinn-phong rendering Youtube video
- Delved into comprehensive lecture slides in PDF format, understanding the concept and required calculations for the final rendered image.
- Systematically followed step-by-step instructions from ”Ray Tracing in One Weekend” to comprehend the programming flow leading to scene rendering. ”Ray Tracing in One Weekend” Book
- Prior to diving into the project, I invested time comprehending the project structure’s intricacies by consulting the ”Ray Tracing in One Weekend” book. This invaluable resource facilitated a comprehensive grasp of fundamental concepts and provided insights into the implementation nuances of the C++ code.
- Following a meticulous exploration, I systematically navigated through the book’s guided steps, thoroughly assimilating the intricacies of binary rendering and subsequently delving into the principles underlying Phong rendering. This methodical approach equipped me with a profound understanding of the rendering processes, laying a solid foundation for my subsequent work on the project.
- Engaged in hands-on learning by implementing the book’s code, reinforcing understanding of the necessary components.


chequerboard.ppm - IrfanView
File Edit Image Options View Help
500 x 500 x 24 BPP
1/6
100 %
2.15 MB / 732.46 KB
13/



2.2 Coding Part

- Here is a link to my ChatGPT query, a single query for the whole project. chatgpt link
- Actively applied the provided code from the book, using ChatGPT for clarification and refinement of programming nuances.
- Utilized ChatGPT for interactive assistance, refining and clarifying aspects of the code for a comprehensive understanding.
- Commenced an independent project from scratch, introducing the project specifics to ChatGPT for guidance on necessary features and specifications.

  Anonymous
  I want to build my own raytracing project using cpp
  I will give you some details and you can divide my work into titles and subtitles, or jobs, in order to go through them step by step and finish my project ... first of all I need to describe you the project and how it will go first and then you give me a whole map of the work but not code at the beginning

  ChatGPT
  Absolutely, I'd be happy to help you outline the steps for your raytracing project in C++. Please provide the details of your project, and we can break it down into manageable tasks. Include information such as the goals of your raytracer, the features you want to include, and any specific constraints or requirements you have in mind. Once I have a clear understanding, we can organize the project into different stages and tasks.

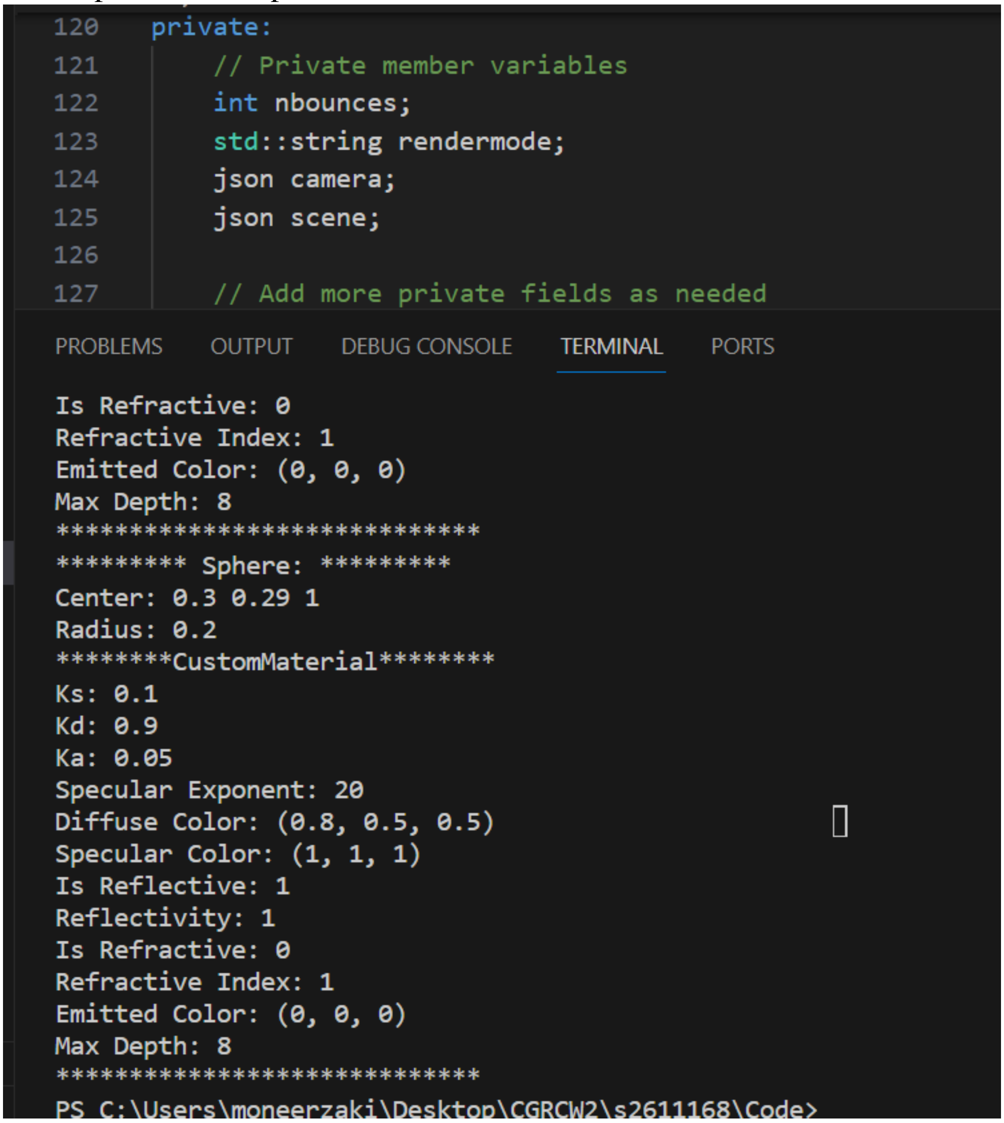
  Anonymous
  My raytracing project would make two main functions (two rendering modes)
  binary rendering mode, and blinn-phong shading rendering mode....

  First of all, every object, or component would be read from a json file ....
  main objects that I will have are:
  - camera (mainly pinhole camera)
  - lightsources
  - background color for the scene
  - a list of hittable_objects. (mainly, spheres, triangles and cylinders)
  - each object should have a material type and a texture

  all of the previous specs I would get from the json file I will read....
  once I read the json file I would create the needed objects from each class and render my scene depending on the rendering mode....
  the output of the render mode would be in ppm format

- Iteratively refined the project output through subsequent sections of the report, incorporating insights gained during the development process.

- Leveraged the interactive nature of ChatGPT to address queries, enhance code efficiency, and ensure a robust final rendering output.
- Developed code by organizing it into classes, initially influenced by a reference book.
- Sought ChatGPT's assistance in generating class implementations, refining and clarifying code details.
- Successfully parsed information from JSON files into objects, cross-verified to ensure precise correspondence.



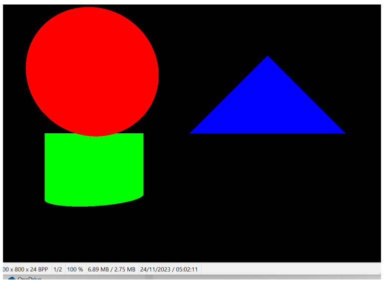
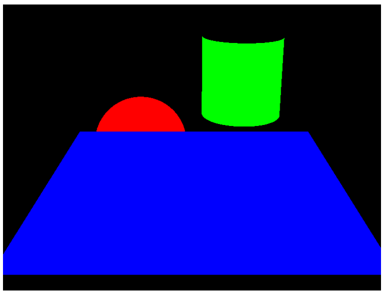
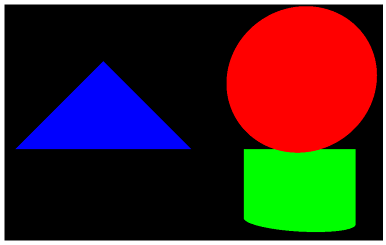
```
private:
    // Private member variables
    int nbounces;
    std::string rendermode;
    json camera;
    json scene;

    // Add more private fields as needed
```

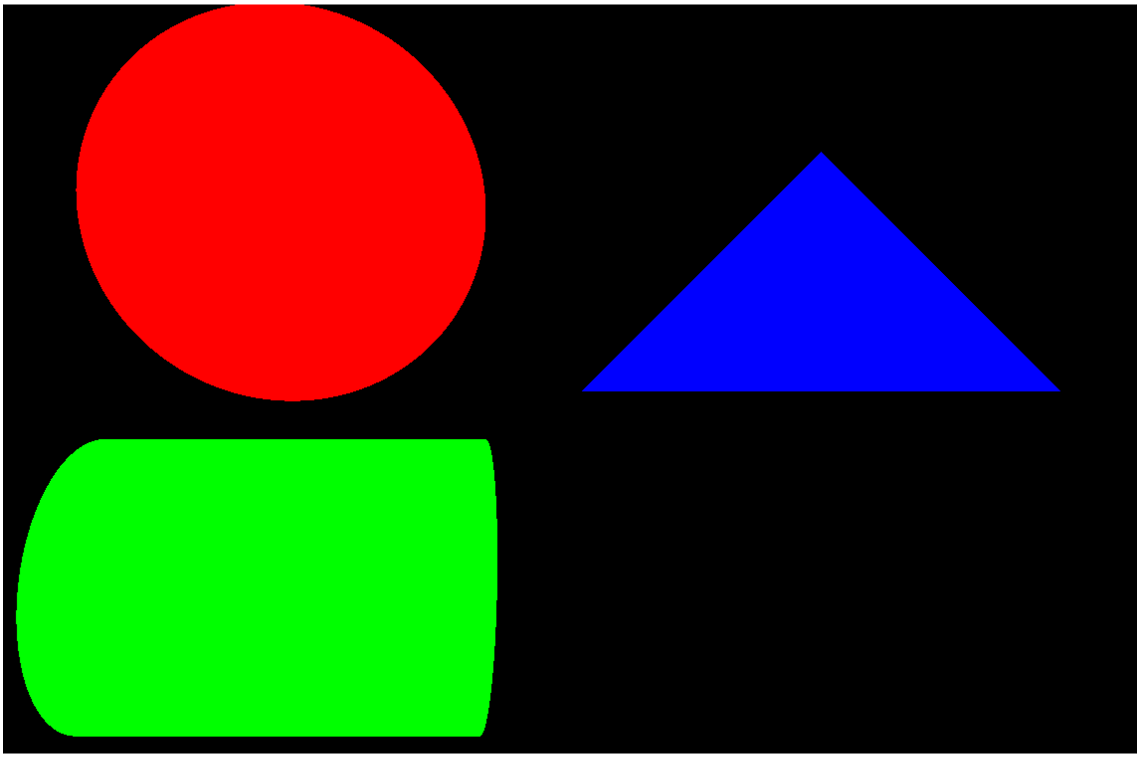
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Is Refractive: 0
Refractive Index: 1
Emitted Color: (0, 0, 0)
Max Depth: 8
*****************************
********* Sphere: *********
Center: 0.3 0.29 1
Radius: 0.2
********CustomMaterial********
Ks: 0.1
Kd: 0.9
Ka: 0.05
Specular Exponent: 20
Diffuse Color: (0.8, 0.5, 0.5)
Specular Color: (1, 1, 1)
Is Reflective: 1
Reflectivity: 1
Is Refractive: 0
Refractive Index: 1
Emitted Color: (0, 0, 0)
Max Depth: 8
*****************************
PS C:\Users\moneerzaki\Desktop\CGRCW2\s2611168\Code>
```

- Implemented binary rendering, assigning colors to objects for visual distinction, meticulously debugging until perfection.

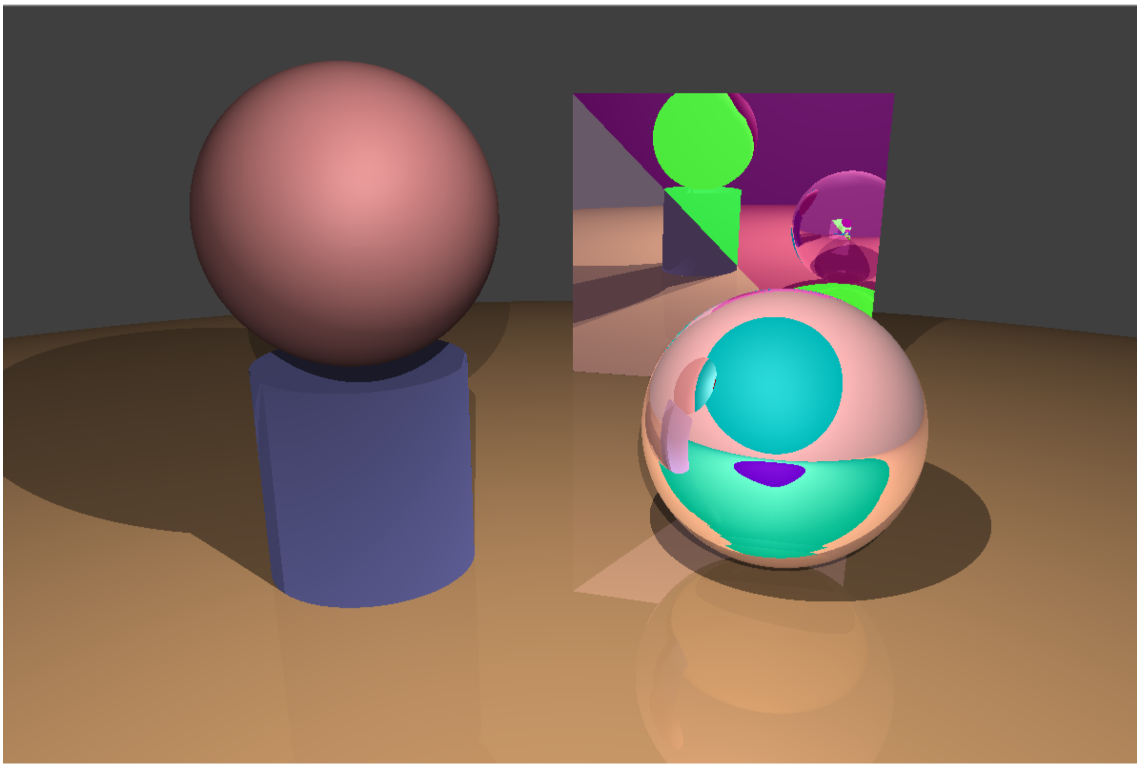




- Addressed a cylinder implementation issue, correcting the height and centering to align with coursework specifications. As you can see for the different trials. first of all I had a wrong axis for the cylinder. asked chatgpt to modify the hit function inside cylinder class to get a correct axis read for the cylinder
  then I had a problem that the things in the front do appear in the back which I solved using copilot by writing a comment saying that search for the nearest hit and return its color.
  after than the cylinder height was just half of the height of the image found in the course work description. which I managed to solve also by copilot. I wrote a comment in the cylinder constructor by saying: ”double the height and recenter the cylinder.” which turned out to be working and I finally got the finally image down below
  finally the image was reverted along the vertical axis which was easy to fix manually.



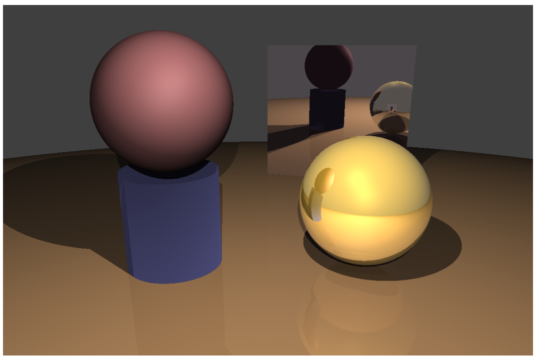
differnt colors for each object is an additional feature I used to differentiate between them.

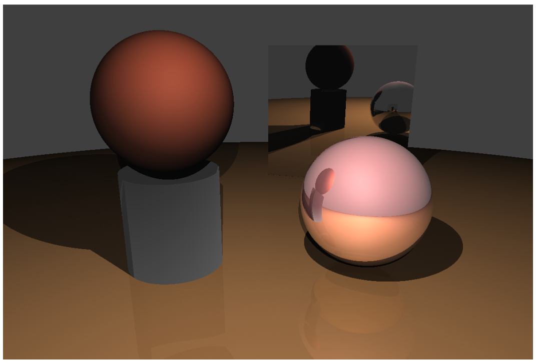
- Proceeded to Phong rendering, achieving success with shadows, reflections, and refractions, ensuring accurate colors.
- Encountered challenges with super-bright lights in reflections, resolved by normalizing pixel values during video conversion.



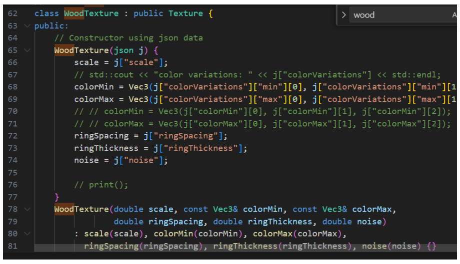
this problem I solved only when I moved to work for the video when the software I used to convert the video did not convert the .ppm files becuase they had pixels' values greater than 255 which after normalizing those values inside the code I have the issue was solved for the super lights found in the reflecting sphere as you can see.

- while delving into the texture feature I faced to many problems however I managed to solve all of them and apply three different types of textures one of them is perfectly applied which is the wooden texture.

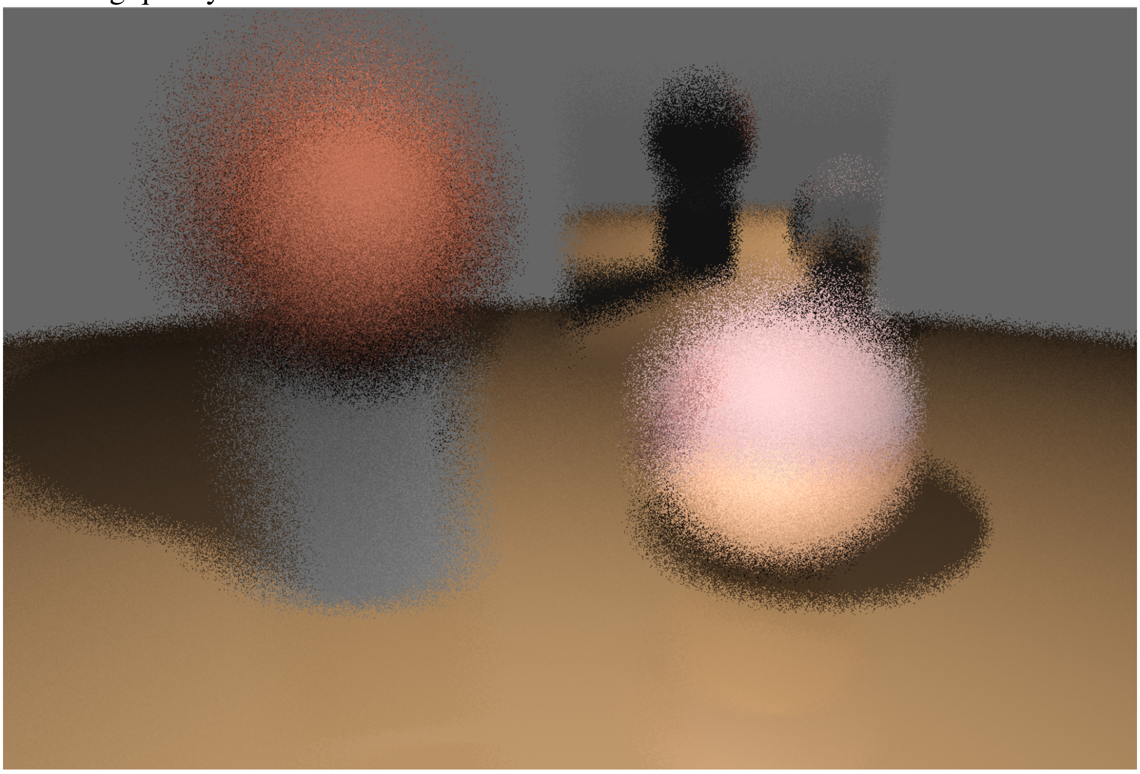




as you can see the ground is the perfect texture I have applied so far which is the wooden. 



after that there is the brick ball at the top of hte cylinder. and finally there is the golden reflective ball.
there implemenation was not as good as expected.

- Advanced to multisampling and defocus in finite-aperture cameras for enhanced rendering quality.

after applying the aperture defocus feature I found the image super blury which was easily fixed by modifying the exposure value to 2.

- Created an engaging video by seeking ChatGPT's input for interesting object motions within the camera view.
you can view the video in the submission folder. View the video via this link.

- I managed to compelet the whole coursework and fully integrated all needed parts for the course work using LLM tools, like chat gpt and copilot.

2.3 Done with the Project

- Implemented multithreading to enhance rendering speed.

```cpp
ppmFile << "P3\n" << width << " " << height << "\n255\n";  // PPM header

std::vector<std::future<void>> futures;

for (int y = height - 1; y >= 0; --y) {
    futures.push_back(std::async(std::launch::async, [this, y, &hittableObjects, &lights, &ppmFile]() {
        for (int x = width - 1; x >= 0; --x) {
            double u = double(x) / (width - 1);
            double v = double(y) / (height - 1);

            Ray ray = generateRay(u, v);
            Color color = phongRayColor(ray, hittableObjects, lights);

            int ir = static_cast<int>(255.99 * color.red());
            int ig = static_cast<int>(255.99 * color.green());
```

```cpp
            // Protecting the output to the file with a lock
            std::lock_guard<std::mutex> lock(outputMutex);
            ppmFile << ir << " " << ig << " " << ib << "\n";
        }
    }));
}

// Wait for all futures to complete
for (auto& future : futures) {
    future.wait();
}
```

```
Done.

Time taken without MT: 3373937 microseconds

Done.

Time taken with MT: 734794 microseconds
```

- Worked on animating object motion for the video component.
- Invested time in crafting an engaging and dynamic video experience.
- Refined the project structure for optimal organization.
- Organized necessary files and folders for the final submission.
- Ensured coherence of project components.
- Prepared documentation and assets for a comprehensive submission.

Chapter 3

Conclusions

In the realm of computer graphics and visualization, ray tracing stands as a cornerstone technique, offering a powerful approach to simulate the behavior of light and create realistic images. This project has been a fascinating exploration into the intricate world of ray tracing, unveiling the complexity and beauty that underlie the synthesis of visually stunning scenes.

The primary goal of this project was to implement a ray tracing system capable of rendering scenes with intricate lighting, shadows, and materials. Through the utilization of fundamental ray tracing principles, such as the casting of rays, intersection testing, and material interactions, we have succeeded in creating captivating visual representations of virtual environments.

The implementation began with the development of a fundamental ray class, capable of tracing rays through a scene and interacting with geometric primitives. The integration of various materials, each with its own unique properties, allowed for the creation of diverse surfaces, from reflective metals to refractive glass.

To enhance the visual realism of the scenes, lighting played a crucial role. Through the incorporation of multiple light sources and the implementation of diffuse and specular reflection models, the scenes came to life with nuanced illumination and captivating highlights.

A notable feature of this project was the exploration of multithreading to optimize the rendering process. Leveraging the power of parallel computation, we achieved significant performance improvements, allowing for the rendering of complex scenes with reduced computation time.

However, the journey does not conclude here. Ray tracing is a vast and ever-evolving field, with advancements continually pushing the boundaries of what is possible. Future enhancements to this project could include the incorporation of more sophisticated materials, advanced global illumination techniques, and optimizations for real-time rendering.

In conclusion, this ray tracing project has been an enlightening expedition into the principles of computer graphics and the art of simulating light in a virtual environment. Through diligent implementation and experimentation, we have uncovered the magic that unfolds when rays, surfaces, and lights converge, producing images that blur the line between virtual and reality. As we conclude this chapter, the path forward beckons with opportunities for further exploration and refinement in the captivating world of ray tracing.