THE UNIVERSITY *of* EDINBURGH

Bachelor in Computer Engineering.

Automagic's Raytracing Renderer

by

s2611134

2023

Abstract

This project involves the development of a cutting-edge raytracing renderer for Automagic's augmented reality software, blending traditional techniques with modern technologies like ChatGPT. The core raytracer, implemented in C++, features essential capabilities such as image rendering, camera transformation, intersection tests for various geometries, and advanced shading with Blinn-Phong model. Leveraging linked language model (LLM) resources, specifically ChatGPT, the development process is documented in a comprehensive report. The project also extends to a pathtracer, enhancing the raytracer with antialiasing, defocus effects, complex material rendering, soft shadows, and multi-bounce path tracing. A video showcasing the renderer's capabilities in motion, including interesting scenes, textures, and dynamic lighting, adds a compelling visual dimension to the submission.

Contents

Chapter 1

Introduction

In response to Automagic's visionary approach to augmented reality, this project embarks on creating a robust raytracing renderer, blending conventional methodologies with cutting-edge technologies. The primary goal is to develop a feature-rich raytracer in C++, incorporating essential elements like image rendering, camera transformations, and complex intersection testing for spheres, triangles, and cylinders. Uniquely, this endeavor integrates the power of linked language model (LLM) resources, specifically ChatGPT, to streamline the development process. The report outlines the meticulous steps taken, including queries and responses from ChatGPT. Furthermore, the project expands into a pathtracer, enhancing realism with antialiasing, defocus effects, intricate material rendering, and dynamic lighting scenarios. The culmination is a captivating video showcasing the renderer's capabilities in motion, underscoring its potential for immersive augmented reality experiences.

1.1 Resources

- ChatGPT:
  - Link for my queries using ChatGPT: link 1, link 2, link 3, link 4. link 5.
  - Leveraged GitHub **Copilot** for code generation, obtaining snippets and suggestions that expedited the development process.
    Integrated Copilot's suggestions for efficient implementation of specific functionalities within the raytracer.
  - Lecture Slides - "Computer Graphics: Rendering" in 2023 by Professor Kartic

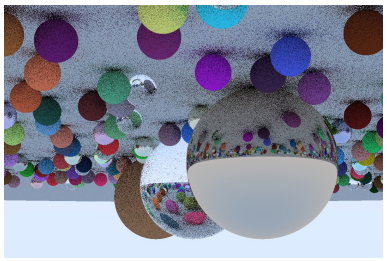
– Consulted the book "Ray Tracing in One Weekend" as a practical guide for understanding the basics of raytracing.

  Used the provided code and examples from the book as a starting point for the raytracing project, gaining insights into key algorithms and techniques.

Chapter 2

Implementation process

2.1 Initial Exploration and Understanding

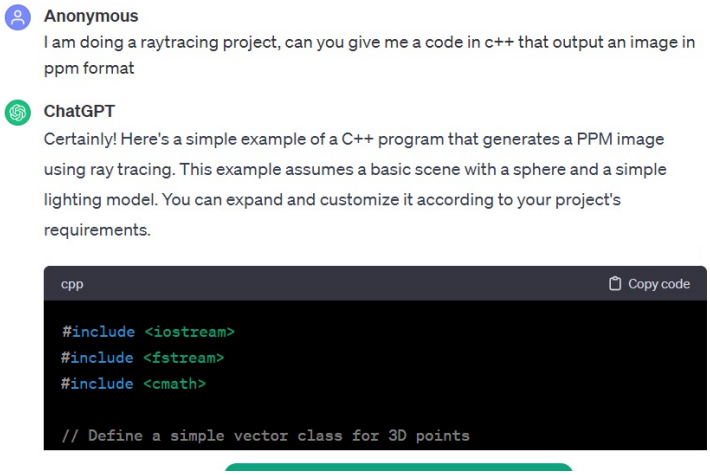


- Began by reviewing and editing code from the book "Ray Tracing in One Weekend" as a starting point for the raytracing project.
- Utilized the provided code to grasp the fundamental concepts of raytracing, gaining insights into implementation strategies and foundational principles.
- Explored the overall idea behind raytracing, understanding its execution process and identifying key starting points for implementation.

2.2 Collaboration with ChatGPT for Project Planning

- Engaged with ChatGPT to discuss the course description and specific raytracing features intended for implementation.
- Leveraged ChatGPT's assistance to outline a comprehensive plan for project completion, ensuring clarity on the necessary steps and requirements.

2.3 Initial Implementation Using ChatGPT



Anonymous
I am doing a raytracing project, can you give me a code in c++ that output an image in ppm format

ChatGPT
Certainly! Here's a simple example of a C++ program that generates a PPM image using ray tracing. This example assumes a basic scene with a sphere and a simple lighting model. You can expand and customize it according to your project's requirements.

cpp — Copy code

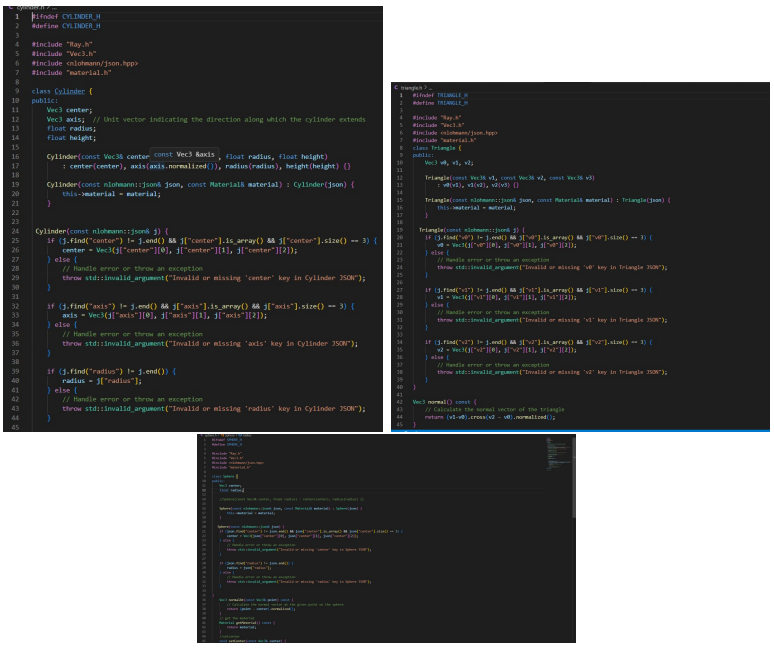
```
#include <iostream>
#include <fstream>
#include <cmath>

// Define a simple vector class for 3D points
```

- Initiated the implementation phase by creating an intersection ray originating from a non-pinhole camera.

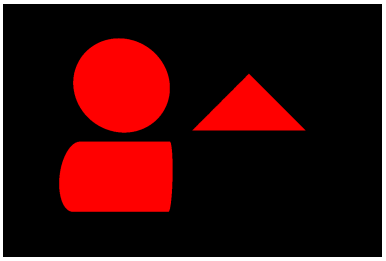
- Developed a sphere class with guidance from ChatGPT to handle ray-sphere intersections, successfully replicating provided book images.

- Extended implementation to include other geometric shapes (triangles and cylinders) with dedicated classes for each, achieving successful intersections within the scene.

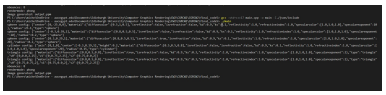


2.4 Camera Type Transformation and Bug Resolution

- Switched the camera type to a pinhole camera and sought ChatGPT's assistance to address resulting bugs.
- Overcame challenges, including rendering issues and object misplacement, through iterative debugging with ChatGPT's guidance.
- Achieved a functional pinhole camera, ensuring accurate rendering and object positioning within the scene.



2.5 Integration of JSON File Reading

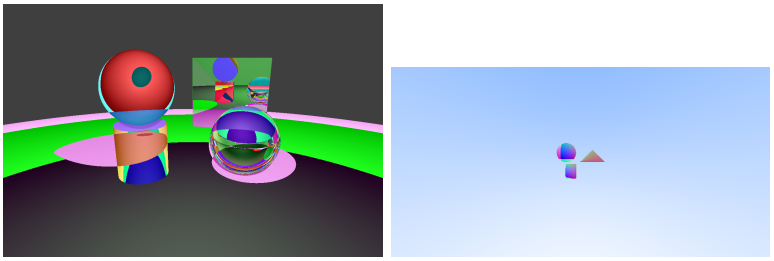


- Transitioned from hard-coded specifications to dynamic reading from JSON files.
- Established a structured code approach, with a function responsible for parsing data from the file and initializing objects (Cylinder, Sphere, Triangle) accordingly.
- Verified the accuracy of object attributes by printing them and systematically resolved errors encountered during this phase.

2.6 Binary Rendering Completion

- Successfully completed the binary rendering stage, ensuring accurate representation of the scene.
- Moved on to the more complex Phong rendering, recognizing it as a challenging step in the project.

2.7 Challenges in Phong Rendering



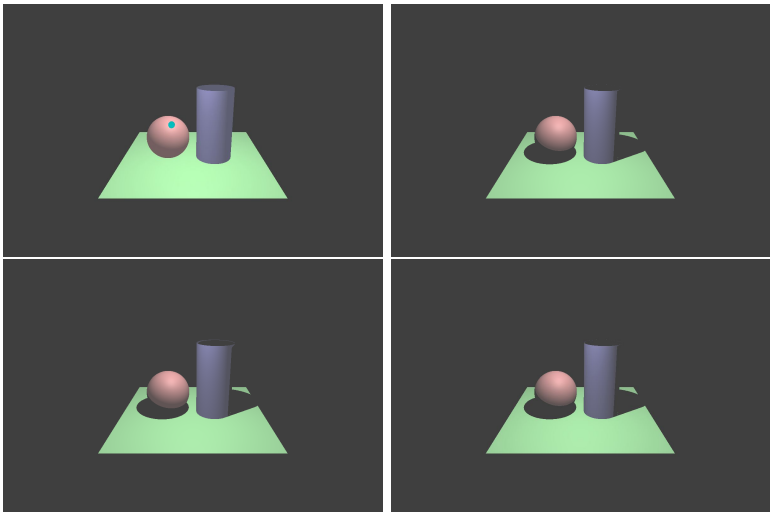
- Encountered difficulties in implementing the Blinn-Phong rendering function, leading to persistent bugs.
- Sought ChatGPT's assistance in obtaining a rendering function description, but initial attempts were unsuccessful due to potential camera specification issues.

2.8 Revisiting Binary Rendering and Successful Phong Implementation

- Returned to a stable state (binary rendering) when faced with challenges in Phong rendering.
- Restarted the description process with ChatGPT, ultimately receiving a functional Phong rendering function that resolved the issues.

- Currently, the Phong rendering function is operational, overcoming earlier challenges in the project.

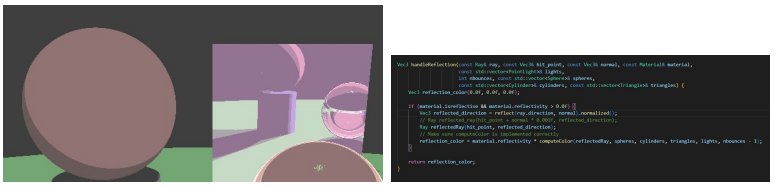
2.9 Blinn-Phong Rendering with Shading:



- Description to ChatGPT focused on implementing the Blinn-Phong shading model for realistic lighting.

- Swiftly achieved a functional rendering process that accurately captured shading effects, setting a solid foundation for subsequent enhancements.

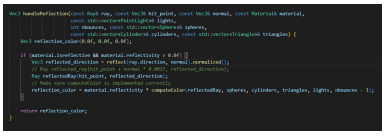
2.10 Reflection and Refraction



- Requested ChatGPT to extend the rendering function to incorporate reflection and refraction processes concurrently.

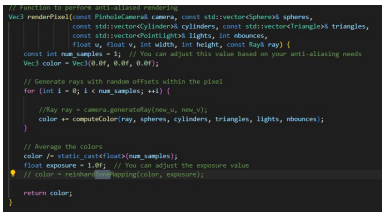
- Successfully integrated reflection and refraction capabilities, enhancing the renderer's realism by accounting for reflective and transparent surfaces.

2.11 Handling Nbounces from JSON Files



- Informed ChatGPT about the need to consider the nbounces value, retrieved from JSON files and initialized within camera attributes.
- Collaboratively implemented a mechanism that dynamically adjusted the number of allowed reflection and refraction bounces based on the provided JSON specifications.

2.12 Multisampling per Pixel

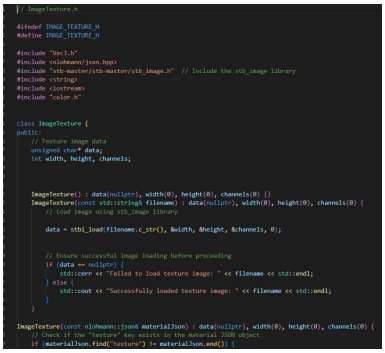


- Described the concept of multisampling per pixel to ChatGPT for improved anti-aliasing.
- Efficiently implemented and seamlessly integrated multisampling, reducing aliasing artifacts and enhancing the overall image quality.

2.13 Tone Mapping

- Tasked ChatGPT with incorporating tone mapping to adjust the final rendered image's contrast and brightness.
- Successfully integrated tone mapping, ensuring visually appealing and well-balanced rendered images.

2.14 Texture Mapping and UV Mapping Challenges



```
// ImageTexture.h

#ifndef IMAGE_TEXTURE_H
#define IMAGE_TEXTURE_H

#include "Vec3.h"
#include <nlohmann/json.hpp>
#include "stb-master/stb-master/stb_image.h"  // Include the stb_image library
#include <string>
#include <iostream>
#include "color.h"


class ImageTexture {
public:
    // Texture image data
    unsigned char* data;
    int width, height, channels;


    ImageTexture() : data(nullptr), width(0), height(0), channels(0) {}
    ImageTexture(const std::string& filename) : data(nullptr), width(0), height(0), channels(0) {
        // Load image using stb_image library

        data = stbi_load(filename.c_str(), &width, &height, &channels, 0);


        // Ensure successful image loading before proceeding
        if (data == nullptr) {
            std::cerr << "Failed to load texture image: " << filename << std::endl;
        } else {
            std::cout << "Successfully loaded texture image: " << filename << std::endl;
        }
    }

ImageTexture(const nlohmann::json& materialJson) : data(nullptr), width(0), height(0), channels(0) {
    // Check if the "texture" key exists in the material JSON object
    if (materialJson.find("texture") != materialJson.end()) {
```

- Requested ChatGPT to implement texture mapping, focusing on UV mapping for realistic surface appearances.
- Encountered challenges in the complex implementation of UV mapping within the given time constraints.
- Despite the difficulty, the majority of features, including UV mapping, were implemented and refined through iterative debugging and testing.