



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

Abstract	i
1 Introduction	1
1.1 Resources	1
2 Implementation process	3
2.1 Initial Exploration and Understanding	3
2.2 Collaboration with ChatGPT for Project Planning	3
2.3 Initial Implementation Using ChatGPT	4
2.4 Camera Type Transformation and Bug Resolution	5
2.5 Integration of JSON File Reading	5
2.6 Binary Rendering Completion	6
2.7 Challenges in Phong Rendering	6
2.8 Revisiting Binary Rendering and Successful Phong Implementation	6
2.9 Blinn-Phong Rendering with Shading:	7
2.10 Reflection and Refraction	7
2.11 Handling Nbounces from JSON Files	8
2.12 Multisampling per Pixel	8
2.13 Tone Mapping	8
2.14 Texture Mapping and UV Mapping Challenges	9

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

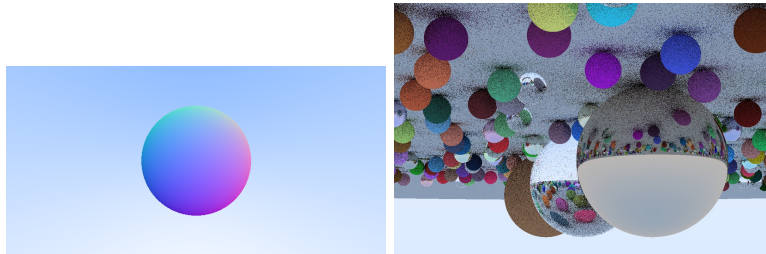
- Consulted the book "Ray Tracing in One Weekend" as a practical guide for understanding the basics of raytracing.

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
```

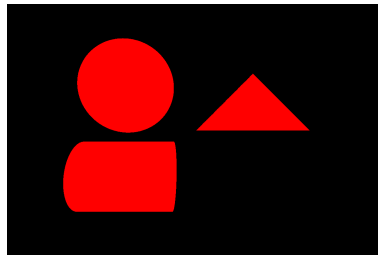
- Initiated the implementation phase by creating an intersection ray originating from a non-pinhole camera.
- Developed a sphere class with guidance from ChatGPT to handle ray-sphere intersections, successfully replicating provided book images.
- Extended implementation to include other geometric shapes (triangles and cylinders) with dedicated classes for each, achieving successful intersections within the scene.

```
1 // Cylinder.h
2 #ifndef CYLINDER_H
3 #define CYLINDER_H
4
5 #include "Ray.h"
6 #include "Vec3.h"
7 #include "Intersection.h"
8 #include "Material.h"
9
10 class Cylinder {
11 public:
12     Vec3 center;
13     Vec3 axis; // Unit vector indicating the direction along which the cylinder extends
14     float radius;
15     float height;
16
17     Cylinder(const Vec3& center, const Vec3& axis, float radius, float height) :
18         center(center), axis(axis.normalized()), radius(radius), height(height) {}
19
20     Cylinder(const json& j, const Material& material) : Cylinder(j["center"], j["axis"], j["radius"], j["height"], material) {}
21
22     // Cylinder(const json& j) {
23     //     if (!j.find("center").is_array() || j["center"].size() != 3) {
24     //         throw std::invalid_argument("Invalid or missing 'center' key in Cylinder JSON");
25     //     }
26     //     center = Vec3(j["center"][0], j["center"][1], j["center"][2]);
27     //     if (!j.find("axis").is_array() || j["axis"].size() != 3) {
28     //         throw std::invalid_argument("Invalid or missing 'axis' key in Cylinder JSON");
29     //     }
30     //     axis = Vec3(j["axis"][0], j["axis"][1], j["axis"][2]);
31     //     if (!j.find("radius").is_array() || j["radius"].size() != 1) {
32     //         throw std::invalid_argument("Invalid or missing 'radius' key in Cylinder JSON");
33     //     }
34     //     radius = j["radius"][0];
35     //     if (!j.find("height").is_array() || j["height"].size() != 1) {
36     //         throw std::invalid_argument("Invalid or missing 'height' key in Cylinder JSON");
37     //     }
38     //     height = j["height"][0];
39     // }
40
41     // Intersection methods
42     bool intersect(const Ray& r, float& t) const {
43         // Calculate the vector from the ray origin to the cylinder center
44         Vec3 v = r.o - center;
45         // Calculate the squared distance from the ray origin to the cylinder center along the axis
46         float d = v.dot(axis);
47         // Calculate the squared distance from the ray origin to the cylinder center in the plane perpendicular to the axis
48         Vec3 w = v - axis * d;
49         float r2 = radius * radius;
50         float h2 = height * height;
51         float w2 = w.dot(w);
52         if (w2 > r2) return false;
53         if (d < -height || d > height) return false;
54         float t1 = 0, t2 = 0;
55         if (w2 < r2) {
56             t1 = d - height;
57             t2 = d + height;
58         } else {
59             t1 = d - height;
60             t2 = d + height;
61         }
62         if (t1 < 0) t1 = 0;
63         if (t2 > 0) t2 = 0;
64         if (t1 > t2) return false;
65         return true;
66     }
67
68     Vec3 normal(const Ray& r, float t) const {
69         Vec3 v = r.o - center;
70         Vec3 w = v - axis * v.dot(axis);
71         float r2 = radius * radius;
72         float w2 = w.dot(w);
73         if (w2 > r2) {
74             Vec3 n = w / w2;
75             if (d < 0) n = -n;
76             return n;
77         } else {
78             Vec3 n = axis;
79             if (d < 0) n = -n;
80             return n;
81         }
82     }
83
84     Material material;
85 };
```

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.

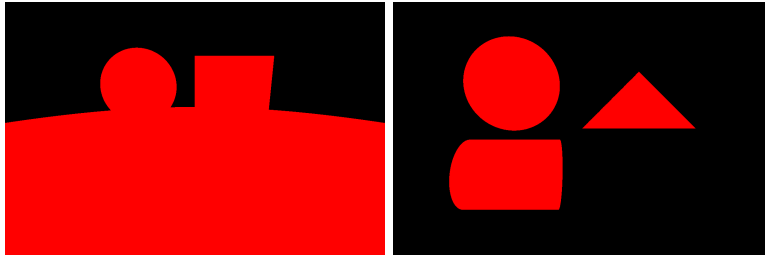


2.5 Integration of JSON File Reading



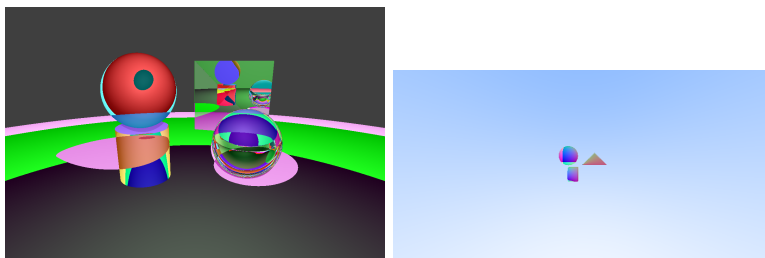
- Transitioned from hard-coded specifications to dynamic reading from JSON files.
- Established a structured code approach, with a function responsible for parsing data from the file and initializing objects (Cylinder, Sphere, Triangle) accordingly.
- Verified the accuracy of object attributes by printing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



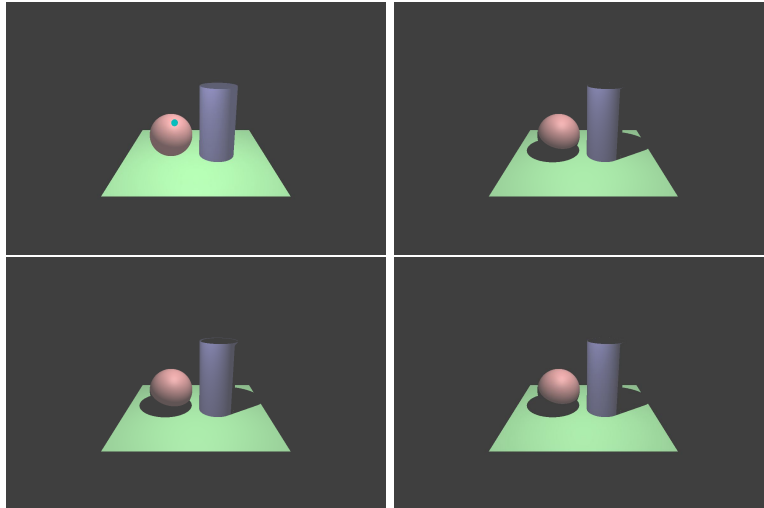
- Encountered difficulties in implementing the Blinn-Phong rendering function, leading to persistent bugs.
- Sought ChatGPT's assistance in obtaining a rendering function description, but initial attempts were unsuccessful due to potential camera specification issues.

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.

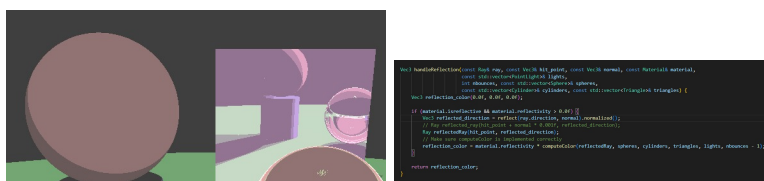
- Currently, the Phong rendering function is operational, overcoming earlier challenges in the project.

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

```
Vec handleRefraction(const Ray& ray, const Vec& hit_point, const Vec& normal, const Material& material,
                  const std::vector<Sphere>& spheres,
                  int& nbounces, const std::vector<Sphere>& spheres,
                  const std::vector<Triangle>& triangles) {
    Vec reflection_color(0.0f, 0.0f, 0.0f);
    // material.refraction = material.refractiveIndex * 0.5f;
    Vec reflected_direction = ray.direction - 2 * normal * normal.dot(ray.direction);
    // If the material is a dielectric, we need to calculate the reflection and refraction probabilities
    Vec refracted_direction;
    float reflect_prob;
    reflect_prob = material.refractiveIndex * material.refractiveIndex;
    reflection_color = material.refractiveIndex * material.refractiveIndex;
    return reflection_color;
}
```

- 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

```
// Function to generate anti-aliased rendering
Vec renderPixel(const PinholeCamera& camera, const std::vector<Sphere>& spheres,
               const std::vector<Triangle>& triangles, const std::vector<Light>& lights, int& nbounces,
               float x, float y, int width, int height, const Ray& ray) {
    const int num_samples = 1; // You can adjust this value based on your anti-aliasing needs
    Vec3 color = Vec3(0.0f, 0.0f, 0.0f);
    // Generate rays with random offsets within the pixel
    for (int i = 0; i < num_samples; ++i) {
        // Ray origin = camera position + offset
        Vec3 offset = Vec3((float)rand() / RAND_MAX * width, (float)rand() / RAND_MAX * height, 0.0f);
        Ray r(ray.origin + offset, ray.direction);
        color += computeColor(r, spheres, cylinders, triangles, lights, nbounces);
    }
    // Average the colors
    color /= static_cast<float>(num_samples);
    float exposure = 1.0f; // You can adjust the exposure value
    // color = reinhardToneMapping(color, exposure);
    return color;
}
```

- 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

```
Vec reinhardToneMapping(const Vec3& color, float exposure) {
    float Lw = 0.2126f * color.x + 0.7152f * color.y + 0.0722f * color.z; // Luminance
    // Tone mapping formula
    Vec3 mappedColor = color * (exposure / (exposure + Lw));
    // Optionally, you can perform gamma correction for display
    float gamma = 2.2f;
    mappedColor.x = std::pow(mappedColor.x, 1.0f / gamma);
    mappedColor.y = std::pow(mappedColor.y, 1.0f / gamma);
    mappedColor.z = std::pow(mappedColor.z, 1.0f / gamma);
    return mappedColor;
}
```

- 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 <glm/glm.hpp>
#include "shaders/shader/sh_image.h" // Include the sh_image library
#include <string>
#include <stringstream>
#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;
        }
        std::cout << "Successfully loaded texture image: " << filename << std::endl;
    }

    ImageTexture(const glm::ivec3& materialID) : data(nullptr), width(0), height(0), channels(0) {
        // Check if the "texture" key exists in the material JSON object
        if (materialID.find("texture") != materialID.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.