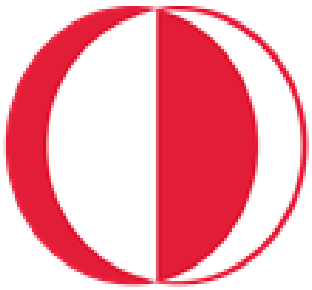


ORTA DOĞU TEKNİK ÜNİVERSİTESİ

Ray Tracer Project Report

(CNG 477)

Prepared by: *Raed Alsheikh Amin - 2528271*

Date: *03/25/2025*

1. Introduction

In this project, I implemented a basic ray tracer in C++ as part of the CNG 477 Computer Graphics course. The goal was to build a renderer capable of handling spheres, triangles, and triangle meshes with Blinn-Phong shading, reflections, and multiple light types. The output is a .ppm image rendered from a scene file input.

2. Project Goals and Scope

 Parse a text-based scene description

 Support the perspective camera model

 Implement intersection methods for:

- Spheres
- Triangles
- Triangle meshes

 Support multiple materials and lights:

- Ambient and point lights

 Apply **Blinn-Phong shading**

 Implement **mirror reflections**

 Render the final image and save it as .ppm

3. System Architecture

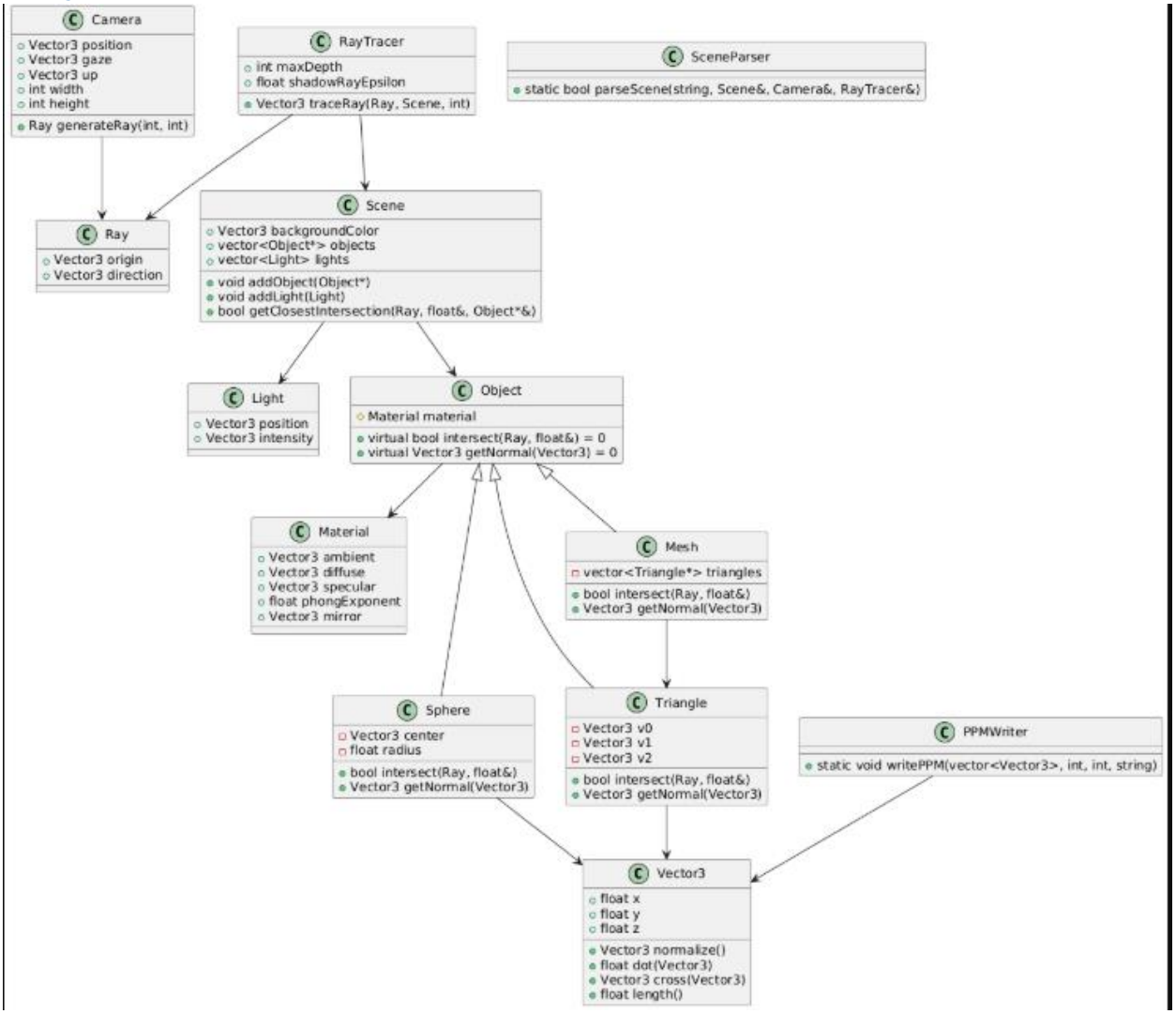


Figure 1: Class Diagram

This class diagram presents a well-structured object-oriented design for a ray tracing system. At the center of the rendering process is the RayTracer, which handles ray casting, shading, and reflections using a recursion depth limit and shadow ray epsilon. Rays are generated by the Camera based on pixel coordinates and are tested for intersections against a Scene containing objects and lights. The Scene manages a collection of polymorphic Object pointers, allowing it to handle various geometry types like Sphere, Triangle, and Mesh. Each object has a Material that defines its lighting response, including ambient, diffuse, specular, and mirror properties. Lights are represented by the Light class, which holds position and intensity, while the Vector3 class supports 3D vector operations for positions, directions, and colors. Intersection logic is implemented in each geometric class, and the Mesh aggregates multiple triangles. Final pixel colors are written to an image using PPMWriter. This design enables modular rendering and cleanly separates scene description, geometry, lighting, and output.

4. Tools and Environment

 Language**:** C++

 IDE**:** CLion (JetBrains)

 Compiler**:** g++ via Makefile (-std=c++11 -O3)

 Platform**:** Windows with MSYS2 terminal

 Image **Format:** PPM (P3)

5. Scene Parsing

The SceneParser class reads input files and extracts:

- Background color
- Camera parameters
- Material definitions
- Light positions and intensities
- Geometric primitives (sphere, triangle, mesh) Each object is stored in a corresponding class (e.g., Sphere, Triangle, etc.) for rendering.

6. Rendering Pipeline

The ray tracing logic follows these steps:

1. Generate primary rays from the camera
2. Find closest intersection using all scene objects
3. Apply shading:
    - Ambient + Diffuse + Specular (Blinn-Phong)
4. If the object has mirror properties, recursively trace reflection rays
5. Combine and clamp final color values

7. Reflection and Shadows

- Reflections are handled recursively using mirror vectors and depth tracking.
- Shadow rays are cast toward each light source to test for occlusion.

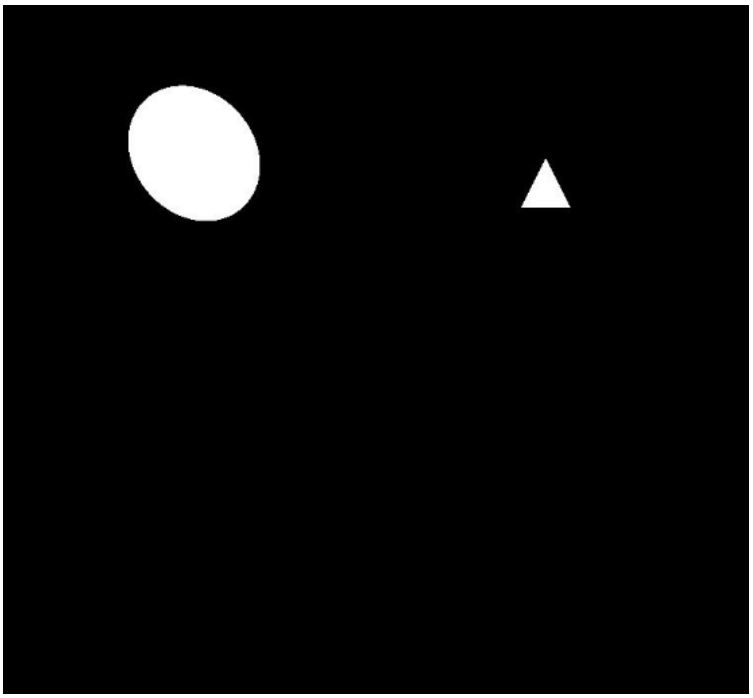
8. Challenges and Debugging

- Handling mesh normals was tricky; I debugged by printing normals to ensure correctness.
- Triangle orientation and winding order affected lighting.
- Getting the Makefile to work outside of CLion took trial and error, but I managed it after 3 hours of learning how to do it.
- One bug caused by accidentally deleting Vector3.h was fixed by restoring the header and updating file references.

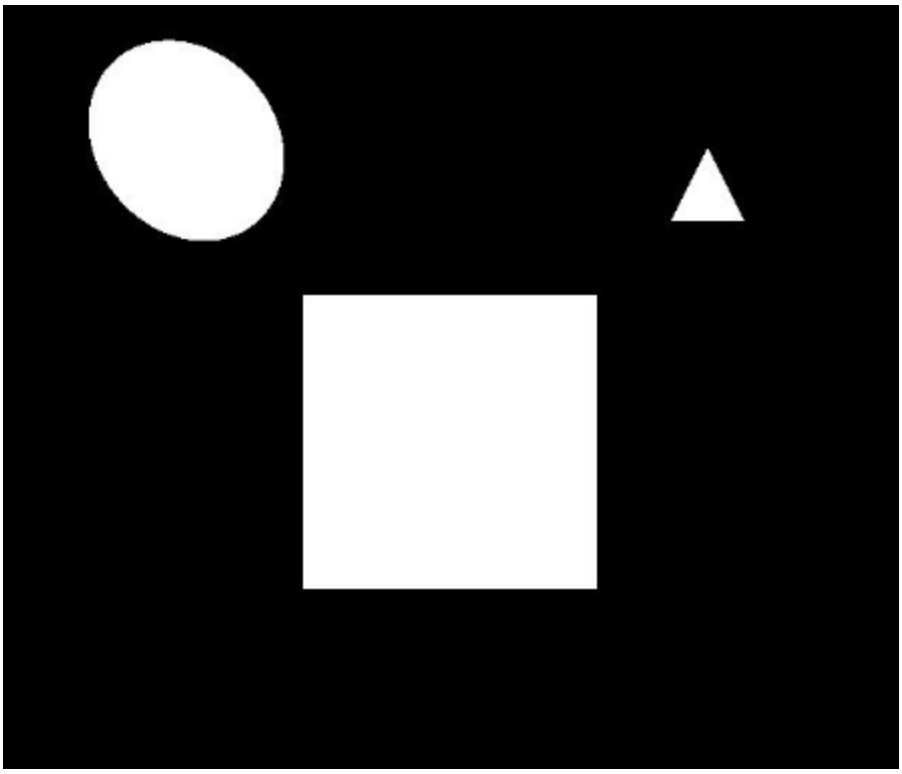
Some output during the process:

- I am skipping the part where I spent few hours with black background only. (Not few hours actually)

Input1.txt:



This picture was taken during the early stages of debugging, where I finally managed to get the objects rendered correctly, the sphere on the left and the triangle on the right. But clearly, something was still missing: there's no shading at all. Everything looks flat and fully white, even though I had point lights defined in the input. At first, I thought maybe the materials weren't parsed properly or the lights weren't added to the scene. After checking the logs and outputs, I realized that only ambient light was affecting the objects. The problem turned out to be with how I handled shadow rays and light attenuation. It took me a good couple of hours to isolate the issue and fix it by making sure the light contribution was being calculated correctly and that the shadow rays weren't falsely reporting occlusion. This moment was a bit frustrating, but once I fixed it, the shading finally appeared and it felt great to see the difference. However, the shadow wasn't obvious in this input file.



This image marks Stage 2 of the project. At this point, I was able to successfully parse all geometry types—spheres, triangles, and meshes—and render them visibly in the scene. The sphere, triangle, and square (made of two triangles) are all positioned correctly and appear with full color fill. However, it's still pretty clear that shading isn't working yet. There's no light interaction—everything is fully white regardless of position or angle to the light source. So while this was a win in terms of parsing and rendering geometry, the next step was to fix lighting calculations and shadow ray handling to bring the objects to life with proper shading and depth.

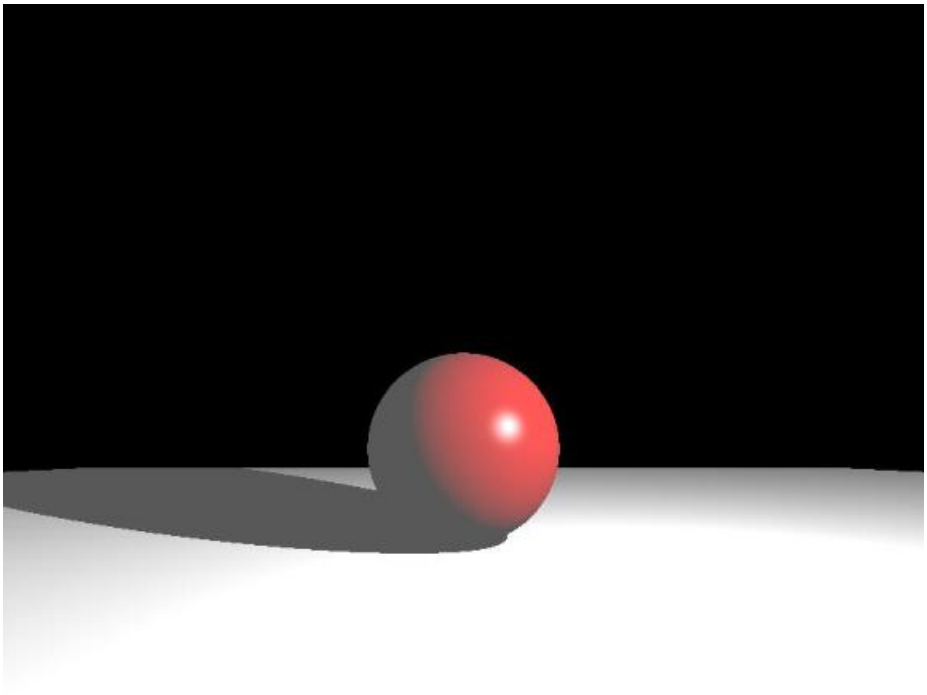
After that, I felt depressed, and I wasn't able to show the real image, something was missing from the sphere. Therefore, I decided to move to input2.txt

---

Input2.txt

and this was my first output after trying Input2.txt, so I decided to take a break for couple of days because I knew that there is a big problem happening.

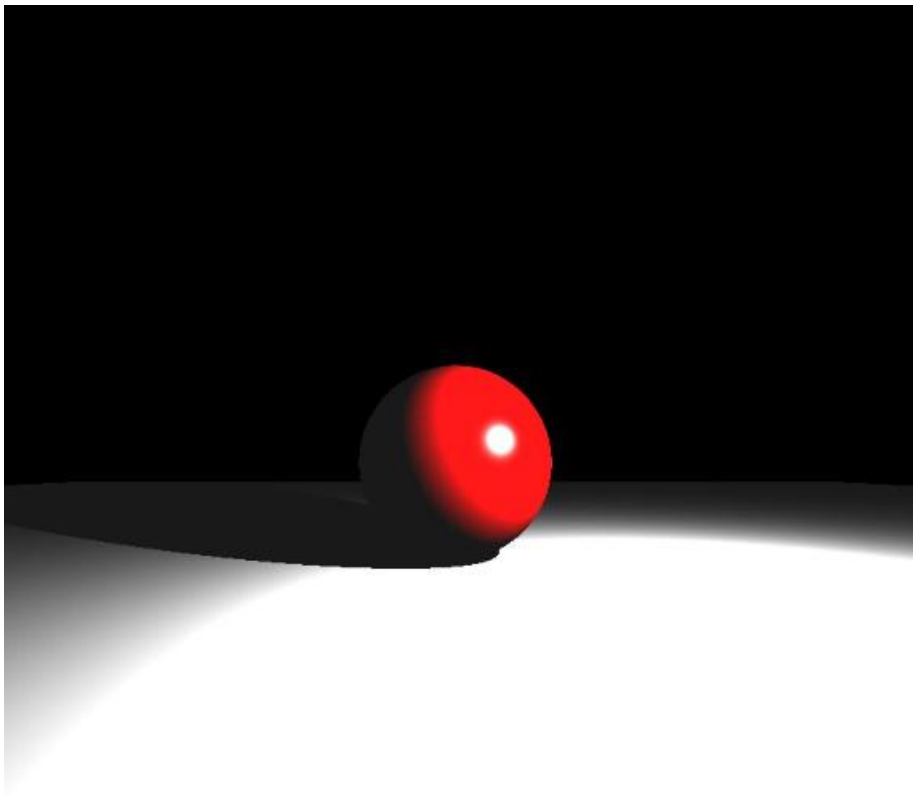
it is not clearly shown here, but finally some differences were happening, and I was able to detect another colour after getting tired of seeing a black background all the time.



This was the result from testing with input2.txt, and honestly, it was a pretty satisfying moment. After spending quite a bit of time fixing lighting calculations, shadows, and especially tweaking the Blinn-Phong shading model, seeing the red sphere rendered with proper shading, a clear shadow, and a visible specular highlight felt like a breakthrough. It finally looked like a real 3D scene. One small issue I noticed was that the shadow edge seemed a bit too sharp and the white surface had a bit of harsh brightness in some areas—possibly due to not applying gamma correction or using high intensity lights without attenuation tuning. Still, the sense of depth and realism improved drastically compared to earlier stages. Definitely a moment where I felt like things were finally coming together.

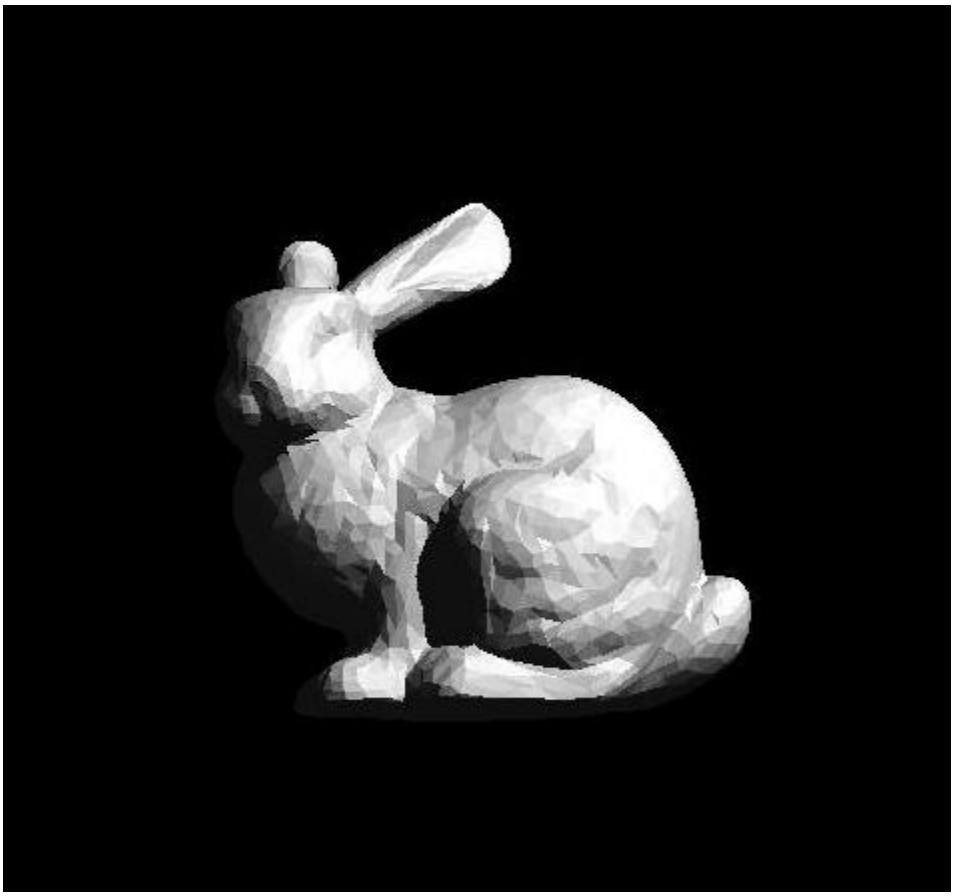
This was the point where I started tweaking values in the code—mostly light intensity, attenuation, and maybe even material properties—to see how it would affect the output. Instead of improving, I ended up flattening the entire scene visually. The red color of the sphere disappeared, and all the shading was gone, leaving everything in stark black and white. The specular highlight vanished too. It was a bit frustrating because I knew I was close just a few moments earlier. I probably broke the diffuse or specular calculation accidentally or zeroed out the intensity somehow. Still, it was useful because it confirmed that even small changes can completely ruin the visual result, which made debugging more focused moving forward.

This was the moment everything finally clicked. After adjusting the light attenuation, fixing the diffuse and specular lighting calculations, and tweaking material values, I got this result—and honestly, it felt great. The red sphere now looks properly shaded, the highlight is sharp and noticeable, and the shadow feels natural and clean. Compared to the earlier flat or broken renders, this felt like a big win. It really showed that the Blinn-Phong model was working correctly, and I was definitely on the right track with the lighting and material parameters. A very satisfying milestone.



Input3.txt

This was the moment my Stanford bunny looked like it was made of wet cement. No shading, no detail—just a mysterious grey blob sitting in the dark like it's judging my code. Clearly, something was off with the normals or lighting, but hey, at least the mesh loaded!



The bunny finally came to life with proper shading and lighting—no more ghost blob. Fixing the normal calculations for the mesh made all the difference. The details in the fur and shadows started popping, and seeing this render felt like a small victory dance moment. However, some minor details were missing for some reason that I couldn’t figure.

9. Command Line execution – Makefile

```
D:\8 sem\CNG 477\Assignments\Assignment 1\477hw1>Raytracer input1.txt

--- Parsing scene file: input1.txt ---
[Parsed] BackgroundColor: 0, 0, 0
[Parsed] MaxRecursionDepth: 3
[Parsed] ShadowRayEpsilon: 0.001
[Parsed] Camera
  Position: (0, 0, 0)
  Gaze:     (0, 0, -1)
  Up:       (0, 1, 0)
  Near Plane: left=-1 right=1 bottom=-1 top=1
  Distance: 1
  Resolution: 800 x 800
[Parsed] Material 1:
  Ambient:  (1, 1, 1)
  Diffuse:  (1, 1, 1)
  Specular: (1, 1, 1)  Phong Exp: 1
  Mirror:   (0, 0, 0)
[Parsed] AmbientLight: 25, 25, 25
[Parsed] PointLight ID: 1 at (0, 0, 0) with Intensity: (1000, 1000, 1000)
[Parsed] Vertex list:
  Vertex: -0.5, 0.5, -2
  Vertex: -0.5, -0.5, -2
  Vertex: 0.5, -0.5, -2
  Vertex: 0.5, 0.5, -2
  Vertex: 0.75, 0.75, -2
  Vertex: 1, 0.75, -2
  Vertex: 0.875, 1, -2
  Vertex: -0.875, 1, -2
  Total vertices: 8
[Parsed] 1 spheres...
[Parsed] Sphere with Material ID: 1, Center Index: 8, Radius: 0.3
[Parsed] Triangle ID: 1, Material ID: 1, Vertices: 5, 6, 7
[Parsed] Mesh with Material ID: 1
  Mesh Triangle: 3, 1, 2
  Mesh Triangle: 1, 3, 4
  Total Mesh Triangles: 2
--- Scene parsing completed successfully ---

Writing PPM image: output.ppm
PPM file successfully written: output.ppm
Rendering complete! Image saved as output.ppm
```

```
D:\8 sem\CNG 477\Assignments\Assignment 1\477hw1>Raytracer input2.txt

--- Parsing scene file: input2.txt ---
[Parsed] BackgroundColor: 0, 0, 0
[Parsed] MaxRecursionDepth: 3
[Parsed] ShadowRayEpsilon: 0.001
[Parsed] Camera
  Position: (0, 5, 25)
  Gaze:     (0, 0, -1)
  Up:       (0, 1, 0)
  Near Plane: left=-1 right=1 bottom=-1 top=1
  Distance: 1
  Resolution: 800 x 800
[Parsed] Material 1:
  Ambient:  (1, 1, 1)
  Diffuse:  (1, 1, 1)
  Specular: (1, 1, 1)  Phong Exp: 1
  Mirror:   (0, 0, 0)
[Parsed] Material 2:
  Ambient:  (1, 1, 1)
  Diffuse:  (1, 0, 0)
  Specular: (1, 1, 1)  Phong Exp: 100
  Mirror:   (0, 0, 0)
[Parsed] AmbientLight: 25, 25, 25
[Parsed] PointLight ID: 1 at (10, 10, 10) with Intensity: (100000, 100000, 100000)
[Parsed] Vertex list:
  Vertex: -100, 0, 100
  Vertex: 100, 0, 100
  Vertex: 100, 0, -100
  Vertex: -100, 0, -100
  Vertex: 0, 5, 0
  Total vertices: 5
[Parsed] 1 spheres...
[Parsed] Sphere with Material ID: 2, Center Index: 5, Radius: 5
[Parsed] Mesh with Material ID: 1
  Mesh Triangle: 3, 1, 2
  Mesh Triangle: 1, 3, 4
  Total Mesh Triangles: 2
--- Scene parsing completed successfully ---

Writing PPM image: output.ppm
PPM file successfully written: output.ppm
Rendering complete! Image saved as output.ppm
```

```
D:\8 sem\CNG 477\Assignments\Assignment 1\477hw1>Raytracer input3.txt

--- Parsing scene file: input3.txt ---
[Parsed] BackgroundColor: 0, 0, 0
[Parsed] MaxRecursionDepth: 3
[Parsed] ShadowRayEpsilon: 0.001
[Parsed] Camera
  Position: (-0.02, -0.05, 1.5)
  Gaze:     (0, 0, -1)
  Up:       (0, 1, 0)
  Near Plane: left=-0.1 right=0.1 bottom=0 top=0.2
  Distance: 1
  Resolution: 512 x 512
[Parsed] Material 1:
  Ambient:  (0.2, 0.2, 0.2)
  Diffuse:  (0.8, 0.8, 0.8)
  Specular: (0.2, 0.2, 0.2)  Phong Exp: 3
  Mirror:   (0, 0, 0)
[Parsed] AmbientLight: 81, 81, 81
[Parsed] PointLight ID: 1 at (2, 2, 2) with Intensity: (3000, 3000, 3000)
[Parsed] Vertex list:
  Mesh Triangle: 2281, 2333, 2287
  Mesh Triangle: 2188, 2187, 2183
  Mesh Triangle: 2188, 2190, 2194
  Mesh Triangle: 2187, 2188, 2194
  Mesh Triangle: 2308, 2315, 2300
  Mesh Triangle: 2407, 2375, 2362
  Mesh Triangle: 2443, 2420, 2503
  Mesh Triangle: 2420, 2411, 2503
  Mesh Triangle: 2411, 1493, 2503
  Mesh Triangle: 1493, 1487, 2503
  Mesh Triangle: 1487, 1318, 2503
  Mesh Triangle: 1318, 1320, 2503
  Mesh Triangle: 1320, 2443, 2503
  Total Mesh Triangles: 4968
--- Scene parsing completed successfully ---

Writing PPM image: output.ppm
PPM file successfully written: output.ppm
Rendering complete! Image saved as output.ppm
```

We can see that the Makefile is working according to the assignment requirements, and the output files are matching the ones provided by the instructor.
Note: some debug statements might be left in the code for future updates.

10. Conclusion

From a programming perspective, this assignment pushed me to structure my code in a modular and object-oriented way. Designing the class hierarchy—especially the abstract Object class and its derived classes (Sphere, Triangle, Mesh)—gave me firsthand experience with polymorphism and inheritance in C++. The Scene class became the central hub, storing all geometry and lights, while RayTracer handled all logic related to light interaction, shading, and recursive reflections. One of the more challenging parts was debugging triangle mesh shading, which required precise normal calculations and careful handling of shadow rays. Another key part was writing a portable Makefile that worked outside of CLion, which taught me how to control builds manually in a real UNIX-like environment. Overall, the code reflects a step-by-step evolution—from basic ray-object intersections to fully shaded, recursive rendering—all built and debugged by hand. I now feel more confident working with graphics pipelines, recursive algorithms, and low-level rendering systems.

References

https://raytracing.github.io/books/RayTracingInOneWeekend.html#outputanimage/theppmimageformat

https://www.youtube.com/watch?v=nQ3TRft18Qw

https://www.scratchapixel.com/