COSC363

Computer Graphics

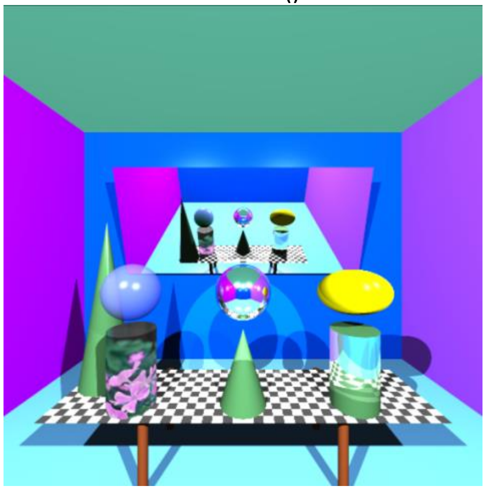
Assignment 2 :: Ray Tracing

Yumeng Shi
74341182

Declaration

I declare that this assignment submission represents my own work (except for allowed material provided in the course), and that ideas or extracts from other sources are properly acknowledged in the report. I have not allowed anyone to copy my work with the intention of passing it off as their own work.
Name: Yumeng Shi
Student ID: 74341182
Date: 27/3/2025

This report documents the development of a ray tracing project for a static scene.
The ray tracer renders a scene containing various geometric objects and lighting effects, including shadows, reflections, and transparency. The eye position is set at (0, 0, 0), located at the centre of a box formed by six axis-aligned planes. The scene includes two light sources, one of which is a spotlight.
The project uses several header files: `Sphere.h`, `SceneObject.h`, `Ray.h`, `Plane.h`, `Cylinder.h`, `Cone.h`, and `TextureOpenIL.h`. The main function is defined in RayTracer.cpp, where the `trace()` function plays a central role. It performs the complete computation of the ray colour for each pixel (or sub-pixel), considering lighting and material properties.
All shape objects in the scene are created in the initialize() function.



Basic Ray Tracing Features

A. **Scene objects**: The scene includes six axis-aligned planes forming a box, each with a distinct colour on its respective face. Additionally, there are 3 spheres, 2 cylinders, and 2 cones, all well-organized and positioned in the scene.

B. **Transparency**: A transparent sphere (sphere1) is placed on the left side with a Transparency Coeff of 0.7. Transparency is implemented using alpha blending. The transparency effect is clearly visible, as cone2 (the green, taller cone) can be seen through sphere1.

C. **Shadows**: Both opaque and transparent objects cast shadows. Shadows from transparent objects are rendered lighter to reflect realistic physical behaviour.

D. **Reflections**: One of the planes in the scene is reflective, acting like a mirror. It displays accurate mirror-like reflections of surrounding objects and is oriented to face a small edge in the -z direction from the eye position.

E. **Patterned Surface**: A black and white chequered pattern is applied to the table surface, adding visual detail and realism to the scene.

Extensions

To go beyond the basic requirements, the following additional features were implemented:

A. **Cone (1mark):** The `Cone.h` and `Cone.cpp` files are used to generate a cone using a custom cone function. The normal vector is constructed at different levels based on the cone's geometry. Using the cone's slope (defined by radius/height), the y value is scaled accordingly. The parametric intersection equation used is:

```
X = p0.x - center.x; Y = height + center.y - p0.y  ; Z = p0.z - center.z;

(dir.x^2+ dir.z^2- dir.y^2* tan(angle)^2)*t^2 + (2*X*dir.x + 2*Z*dir.z + 2
      *tan(angle)^2*Y*dir.y)*t + X*X + Z*Z - tan(angle)^2  * Y * Y
```

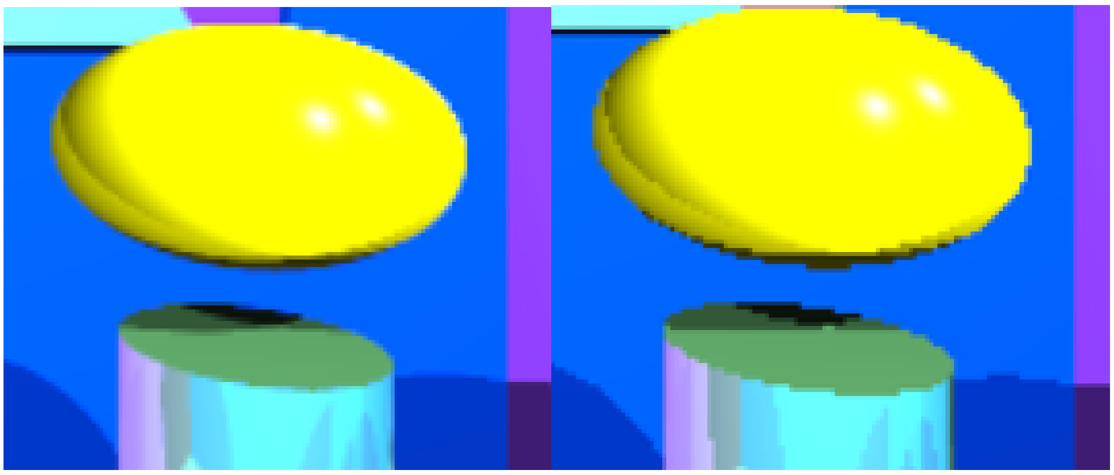
B. **Cylinder with clearly visible cap (1mark) :** A cylinder is generated using `Cylinder.h` and `Cylinder.cpp`. For the side surface aligned along the y-axis, t1 and t2 solutions from the quadratic equation (as taught in class) are used. After calculating the intersection positions, the height is used to verify whether the ray hits the valid side surface. If t1 exceeds the cylinder height, it checks if the ray intersects with the top cap.

```
(dir.x * dir.x + dir.z * dir.z)*t^2 + (2 * ((p0.x - center.x) * dir.x + (p0.z -
center.z) * dir.z))*t + (p0.x - center.x) * (p0.x - center.x) + (p0.z - center.z) *
                        (p0.z - center.z) - radius * radius
```

C. **Refraction (1mark)**: True light refraction is implemented through a transparent sphere using Snell's Law. The ray direction is adjusted accordingly, with visible results when light passes through the green cylinder on the right.

D. **Anti-Aliasing (1mark)**: Super-sampling anti-aliasing is implemented using a 2×2 sample grid per pixel (4 rays). The final pixel colour is computed by averaging the four samples to reduce jagged edges. In the process, `col = antiAliasing(eye, xp, yp, cellX, cellY, threshold, 0, maxDepth)` is called inside display , also need to remove `col = trace(ray, 1)`. In the `antiAliasing` function, each pixel is subdivided into four subpixels, and a ray is traced through the centre of each subpixel. The colour of each ray is stored in superSamples. To achieve **adaptive anti-aliasing**, the algorithm compares the colour of the bottom-left ray `superSamples[0]` with the other three. The maximum colour difference is computed.

- If the largest difference is **smaller than a threshold**, it means the colours are similar and recursion stops — the average of the four samples is returned.
- If the maximum difference is **larger** and the current depth is less than maxDepth, the pixel is recursively subdivided further, and antiAliasing() is called for each subpixel.
- Once depth >= maxDepth, no further subdivision is performed, and the average colour of the current samples is returned.

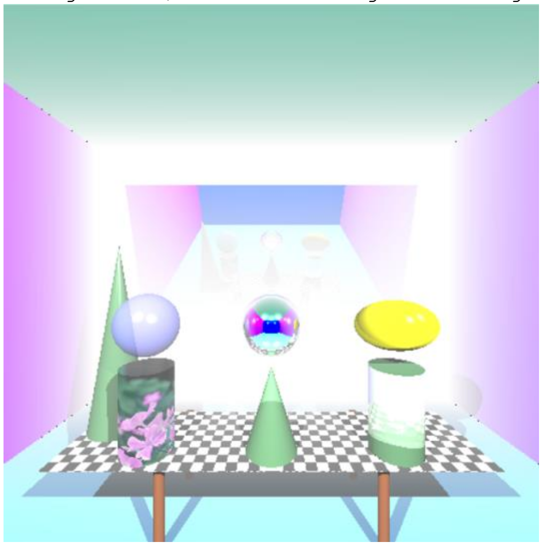


Anti-Aliasing Enabled(maxDepth = 5) **Anti-Aliasing Disabled**

E. **Spotlight (0.5mark A spotlight is implemented with a half cut-off angle of 15°, located at (0, 5, -5) and directed toward (0, -15, -40) and (20, 23, 15). The spotlight casts a visible circular shadow of the middle sphere on the blue back wall.**

F. **Fog (0.5mark)**: An atmospheric fog effect is added based on the distance from the camera. Distant objects gradually blend into the background color. This can be enabled by uncommenting the following line in trace():

```
color = (1 - fogFactor) * color + fogFactor * glm::vec3(1);
```



G. **Multiple Light Sources (1mark)**: The scene includes two point light sources at (20, 23, 15) and (-20, 23, 15). Each object casts two shadows. Areas not reached by either light only receive ambient lighting.

H. **Textured non-planar (curved) object (1mark)**: A texture is mapped onto a curved surface with a cylinder centred at `(-12, -17, -23)` with `halfHeight = 8` and `radius = 2.5`.

Convert the 3D hit point to local cylinder coordinates: `lowHit = hitPoint - cylCenter;`

Compute angle around the y-axis in z, x plane is `atan2(lowHit.z, lowHit.x);`

Convert Alpha to s ➔[-π, π] to [0, 1] `s = (alpha + pi) / (2 * pi);`

Convert y to t ➔[-8, +8] to [0, 1] `t = (loHit.y+8)/16(the whole height)`

I. **Object-space transformations (2mark)**: Applied transformations to sphere3, initially generated at (0, 0, 0) with radius = 3:

Translation to new position

Scaling by (1.3, 0.8, 1.0)

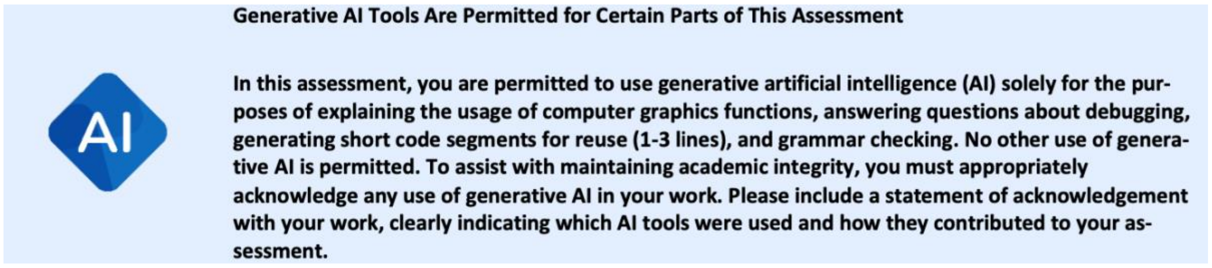
Rotation by 15° (in radians) around the y-axis

This results in an elliptical yellow shape visible on top of the right cylinder.
This transformation can found in `Sphere.cpp` for `tScale` and Translation+ Rotation in `RayTracer.cpp initialize()`.

Run Time

On my personal machine (AMD Ryzen 5 5600X 6-Core @ 3.7GHz, 16 GB RAM, Windows) the ray tracer took ~13 seconds to render a 600×600 image with recursion depth = 5, NUMDIV=500 and anti-aliasing disabled.

Generative AI Usage



Generative AI Tools Are Permitted for Certain Parts of This Assessment

In this assessment, you are permitted to use generative artificial intelligence (AI) solely for the purposes of explaining the usage of computer graphics functions, answering questions about debugging, generating short code segments for reuse (1-3 lines), and grammar checking. No other use of generative AI is permitted. To assist with maintaining academic integrity, you must appropriately acknowledge any use of generative AI in your work. Please include a statement of acknowledgement with your work, clearly indicating which AI tools were used and how they contributed to your assessment.

1. **Code generate and general idea on Anti-Aliasing:** I initially wrote all the anti-aliasing code directly into the `display()` function, but it didn't work as expected. ChatGPT helped me understand how to restructure the code by suggesting that I split it into a separate function and call it within `trace(ray, 1)` from the `display()` function. It also provided guidance on the necessary variables and how to structure the function. The final function from AI is:

```
glm::vec3 antiAliasing(glm::vec3 eye, float xp, float yp, float
cellX, float cellY, float threshold, int depth, int maxDepth);
```

2. **Debugging on incorrect Refraction:** ChatGPT also helped me debug an issue with incorrect refraction behaviour in my ray tracer. With its suggestions, I was able to identify and fix the logic related to the refracted ray direction and the handling of total internal reflection.

3. **Grammar-check this report:** For grammar corrections in this report and throughout the project documentation, I used ChatGPT and Grammarly.

The tools used were **ChatGPT** and **Grammarly** for these limited purposes.