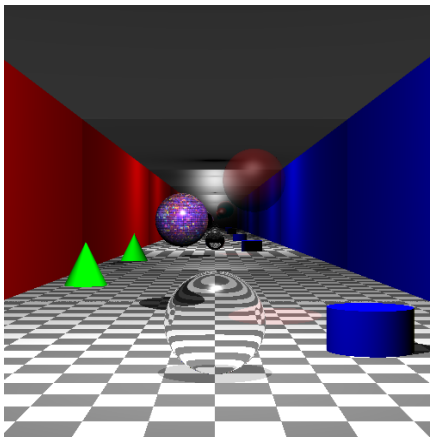
Geordie Gibson
18059623



My ray tracer consists of a box with five scene objects each representing a different ray tracing feature. The first object on the left is a green cone, the sphere in the back is a textured sphere to look like a disco ball, the middle sphere is refractive, bending light as it passes through it causing the floor to look distorted. The red sphere in the top right is transparent meaning you can see through it and the scene behind it. The shadows produced by the transparent/refractive objects are lighter. The final object is a cylinder with a cap. It also has mirrors on the front and back wall causing it to look like an infinite hallway, as well as some fog. Fog has been turned off by default after talking to the tutor before submission but has still been implemented as shown below.

Extra Features (7.5/7 Marks):

Fog (0.5 Marks):
The fog was created using the following equations.

$$\lambda = \frac{(ray.hit.z) - z1}{z2 - z1} \qquad colour = (1 - \lambda)\, colour + \lambda\, white$$

The fog was also clamped outside of the fog range using glm::clamp(). Below is a comparison of my scene with and without fog.

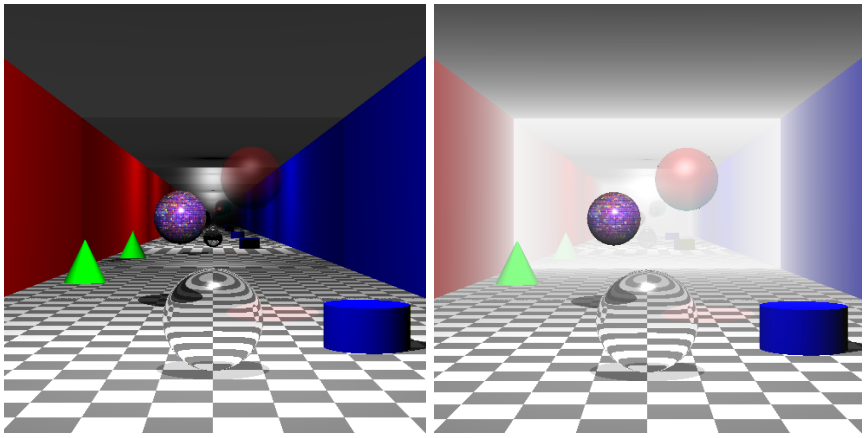


Figure Two and Three: Before and After Fog

Basic Anti Aliasing (1 mark) :

According to SelectHub, "Anti-aliasing is a technique that smooths out the edges seen in images while playing games on a PC. It makes them appear less blurred and blends colours to make visuals look natural".
I wanted to use this in my application because as a whole, it makes it look a lot smoother and less janky. Before implementing anti-aliasing there were many areas of my scene that were

noticeably not being blended properly such as the shadow outlines, the edge of my spheres and cylinder, as well as the chequered floor. Below is a comparison of my scene with and without anti-aliasing enabled. Fog has also been disabled so that the difference is more apparent.

The way it is implemented is by checking the colour of the surrounding pixels and returning the average colour. This allows for colour to be a lot smoother and a lot less clear cut colour changes.

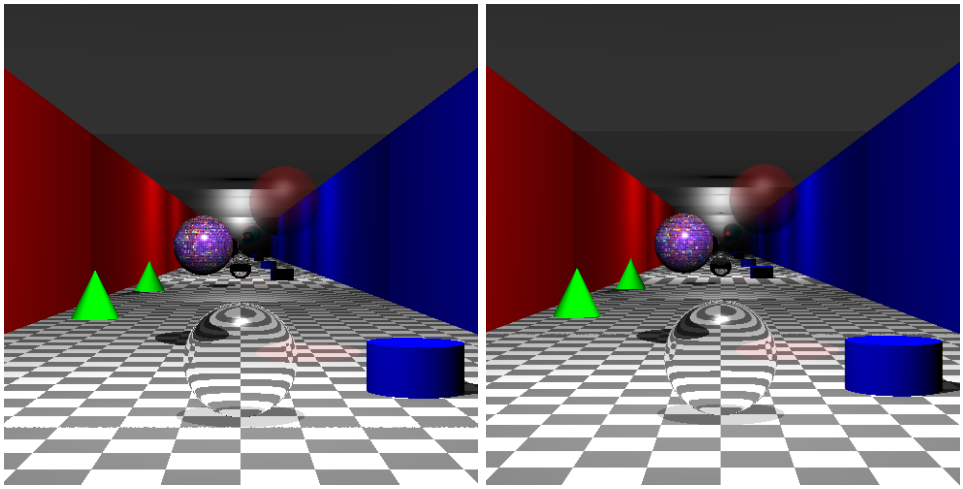


Figure Four and Five: Before and After Anti-Aliasing Respectively

Textured Sphere (1 Mark):
To give a sphere a texture I used the following equations from the UV Mapping Wikipedia page to represent the textcoords and textcoordt respectively.

$$u = 0.5 + \frac{\arctan2(d_z, d_x)}{2\pi}, v = 0.5 + \frac{\arcsin(d_y)}{\pi}.$$

Parallel Mirrors (1 Mark):
The front and back wall of my scene are both mirrors causing multiple reflections and an effect of an “infinite hallway”. There were no mathematical calculations for this feature. This is a lot more obvious when fog is not enabled.

Refractive Sphere (1 Mark):
My refractive sphere was simply created by modifying the provided code in the lecture notes with an eta of 1.01.

It first computes a ray that collides with the object, it then computes another ray from that hit point (with glm::refract) until it hits the other side of the sphere and from there you send another ray (with glm::refract again) until it hits the ground, calculating the colour from the result. This creates a sphere that appears to bend the light as it passes through it.

Geordie Gibson
18059623

Cylinder (1 Mark):
To implement my cylinder I had to create new files, Cylinder.cpp and Cylinder.h, similar to how the default sphere was created. I then had to implement my own functions for intersect() and normal().

To calculate the intersection method I used the following equations given in the lecture notes. I rearranged it and using the quadratic equation, found t1 and t2. I then returned different values based on the different possible scenarios.

$$t^2\left(d_x^2 + d_z^2\right) + 2t\{d_x(x_0 - x_c) + d_z(z_0 - z_c)\} + \{(x_0 - x_c)^2 + (z_0 - z_c)^2 - R^2\} = 0.$$

$$x = x_0 + d_x t; \quad y = y_0 + d_y t; \quad z = z_0 + d_z t;$$

To calculate the normal I returned a normalised vector of the following equation given in the lecture notes.

$$\boldsymbol{n} = (x - x_c,\ 0,\ z - z_c)$$

Cap (0.5 Marks):
To implement the cap on my cylinder, I simply had to extend my cylinder functionality. I had to consider the situation where t1 would miss but t2 would hit. This would indicate that the ray was hitting the cap. So instead of simply returning t2, I could instead return the result of the following equation.

$$t_1 = \frac{y_c + h - y_0}{d_y}$$

This would provide the cap of the cylinder, but I also needed to adapt the normal function to consider the case of when a ray is hitting the top of the cylinder. In this case I just returned a normal vector of (0,1,0).

Cone (1.5 Marks):
Like the cylinder, to implement my cone I had to create new files, Cone.cpp and Cone.h, and calculate the intersect() and normal() functions specific to a cone.
To calculate the intersection method I rearranged the following equation and used the quadratic equation to give values for t1 and t2 similar to the cylinder calculation. I then performed some calculations to return the value of t1 or t2 depending on the possible scenarios.

$$(x - x_c)^2 + (z - z_c)^2 = \left(\frac{R}{h}\right)^2 (h - y + y_c)^2$$

To calculate the normal method I rearranged the following equation substituting the values theta given by rearranging tan = (r/h) and the value for alpha given below.

$$\boldsymbol{n} = (\sin\alpha\cos\theta, \sin\theta, \cos\alpha\cos\theta)$$
$$\text{where} \quad \alpha = \tan^{-1}\left(\frac{x - x_c}{z - z_c}\right)$$

Geordie Gibson
18059623

Time Estimate:
My program takes 5-12 seconds to generate an output on the computers in Lab 4.

Reflection:
I am very happy with the result of my raytracer. It uses many different techniques that I had only ever heard of before through playing various video games. One thing I would implement if I had more time is Adaptive Anti-Aliasing.

Build Instructions:
Using the terminal:
First:

```
g++ -o RayTracer RayTracer.cpp Ray.cpp SceneObject.cpp Sphere.cpp Plane.cpp cylinder.cpp cone.cpp TextureBMP.cpp -lm -lGL -lGLU -lglut
```

Second:
./RayTracer

Using QTCreator:
Open QTCreator, Open File or Project, select the assignment, click ‘Projects’, click ‘Run’, click ‘Browse’, verify that it is in the directory of the assignment then click the big green button. This should run the assignment.

External Resources:
Disco Ball image was sourced from Adobe Stock (https://stock.adobe.com/nz/search?k=disco+ball+texture) and then converted to a bmp (https://image.online-convert.com/convert-to-bmp)
Equations for textured sphere - https://en.wikipedia.org/wiki/UV_mapping
UV Mapping - https://en.wikipedia.org/wiki/UV_mapping
Anti-Aliasing - https://www.selecthub.com/resources/what-is-anti-aliasing/#:~:text=Anti-aliasing%20is%20a%20technique,to%20make%20visuals%20look%20natural.

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 Geordie Gibson **Student ID** 18059623 **Date** 1/6/23