Intro:

This is Bob. Bob seems to get bored really easily. However, he was recently gifted a fancy set of shapes and lights. Bob was mesmerized by the refracted spheres, and the weird looking cones. Bob started placing objects around creating something like a museum. Bob was very impressed with the scene he had created and went to get his friends to check out his creation.

Build:

This program uses a cmake file to create and run the program. Execute the follow commands the build and run the program:

- cmake .
- make
- ./RayTracer.out

Extra Feature – Creating a Cone and Cylinder

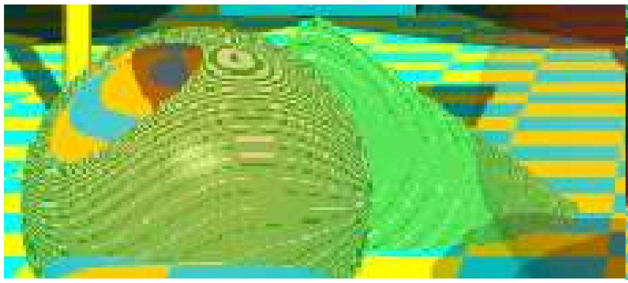


Figure 1: The Generated Cone

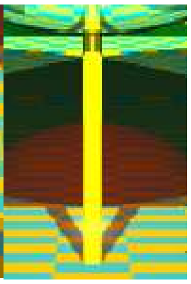


Figure 2: The Generated Cylinder

In regard to the ray, intersection, and surface normal vector equations below:

- d represents the direction the of the ray vector
- u represents the distance from the origin of the ray to the location of the ray
- o represents the origin of the ray
- c represents the center of the cone or cylinder
- r represents the radius of the base of the cone or cylinder
- h represents the height of the cone or cylinder
- n represents the unnormalized surface normal vector
- x, y, z represent coordinates in three-dimensional space

Cone and Cylinder Ray Equation

$$x = d_x u + x_o$$

$$y = d_y u + y_o$$

$$z = d_z u + z_o$$

Cone Intersection Equation

$$2u\left(d_x(x_o - x_c) + d_y\left(tan^2(h - y_o + y_c)\right) + d_z(z_o - z_c)\right) + u^2\left(d_x^2 - tan^2 d_y^2 - d_z^2\right) + ((x_o - x_c)^2 - tan^2(h - y_0 + y_c)^2) = 0$$

Cone Surface Normal Vector Equation

$$n = ((x_o - x_c), \quad \left(\sqrt{(x_o - x_c)^2 + (z_o + z_c)^2}\right)\left(\frac{r}{h}\right), \quad (z_o - z_c))$$

Cylinder Intersection Equation

$$2u\big(d_x(x_o - x_c) + d_z(z_o - z_c)\big) + u^2(d_x^2 + d_z^2) + (x_o - x_c)^2 + (z_o - z_c)^2 - r^2 = 0$$

Cylinder Surface Normal Vector Equation

$$n = (x_o - x_c, \qquad 0, \qquad z_o - z_c)$$

Extra Feature – Refractions

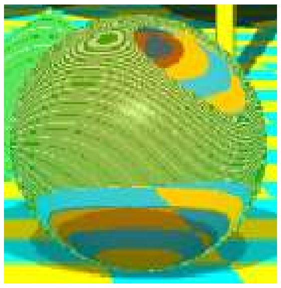


Figure 3: Refractive Sphere

At the front of the scene there are three spheres, two of which are refractive. For an object to be refractive within a globally illuminated scene the rays passing through that object must be recursively traced. During each recursive step, the ray must be traced twice, once when the ray enters the sphere, and again once the ray exits the sphere. Thus, two normal vectors and two refracted rays are calculated. Additionally, an ETA factor of 1.02 meant that the spheres were refractive, but also transparent to highlight the procedurally generated patterns on the cones.

Extra Feature – Multiple Light Sources

Within this scene there are two light sources with a light intensity of 50% place on either side of the camera. The lighting intensity represents a factor that combines the specular and diffuse terms. Because of the two light sources every object within the scene has two shadows, with most of them overlapping to create darker shadows. Additionally, the two light sources create two specular highlights on the reflected sphere, however they are so close together that they appear as one wide specular highlight.

Extra Feature – Anti-Aliasing

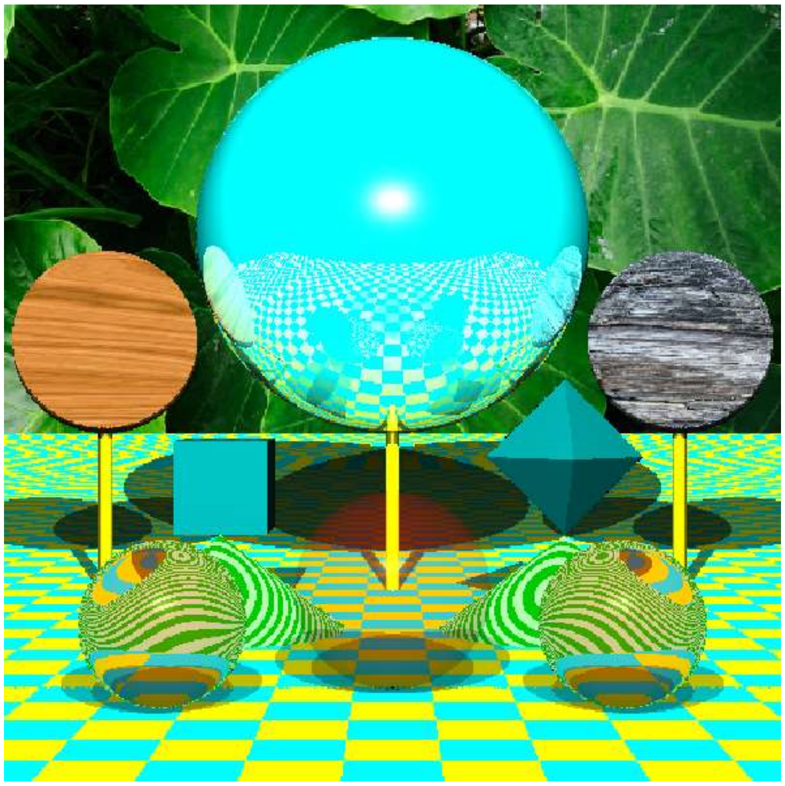


Figure 4:The Scene Without Anti-Aliasing

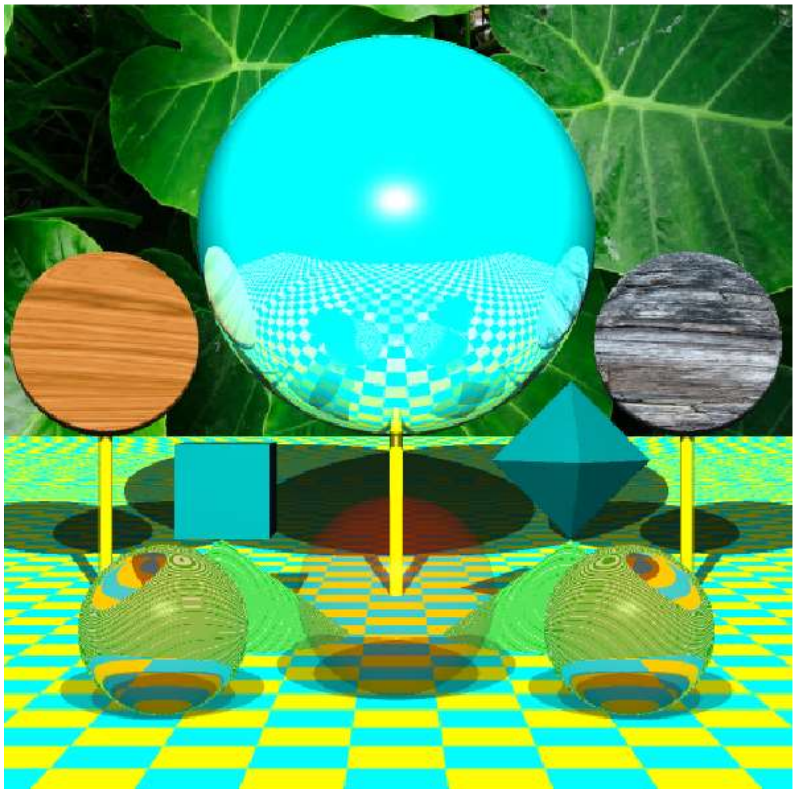


Figure 5: The Scene With Anti-Aliasing

I have implemented SSAA or super sampling anti-aliasing. The main purpose of anti-aliasing is to smooth out jagged edges on objects and shadows. Super sampling is one of the most expensive types of anti-aliasing in terms of calculations as it effectively quadruples the pixel count of the image. This is because every pixel is divided into four equally sized segments. For each segment, a ray is created to travel through the center of said segment to trace and calculate the colour value. Afterwards the four colour values are averaged and a single colour value is returned and displayed on the pixel. As you can see in the screenshots the edges are no longer jagged, however now the program takes exactly four times longer to run.

Extra Feature – Textured Sphere

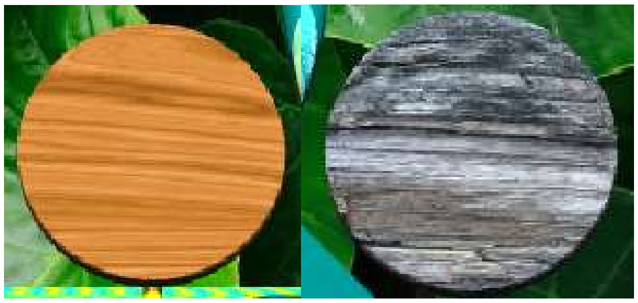


Figure 6: Images Textured on Spheres

By modifying the $textcoords$ and $textcoordt$ vector values that are used for textured a plane, it possible to texture a sphere. This was done by using the following equations:

- k represents the point in three-dimensional space where the ray hit the sphere
- r represents the radius of the sphere

$$texcoords = \frac{0.5 + tan^{-1}\left(\frac{\left(\frac{x_k - x_c}{r}\right)}{\left(\frac{z_k - z_c}{r}\right)}\right)}{2\pi}$$

$$texcoordt = \frac{0.5 - \operatorname{asin}\left(\frac{y_k - y_c}{r}\right)}{\pi}$$

Note: This equation was derived from the UV mapping Wikipedia page, however I broke it down to its fundamental form, including the normalization of the ray.

Extra Feature – Procedurally Generated Pattern on Cones

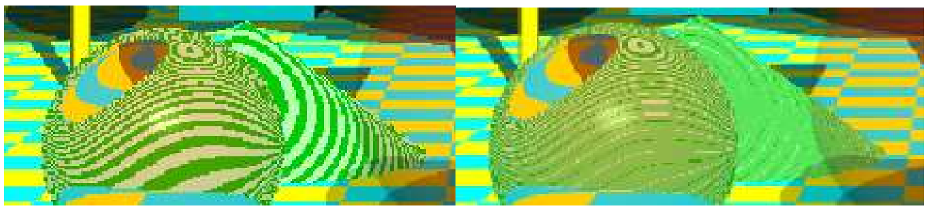
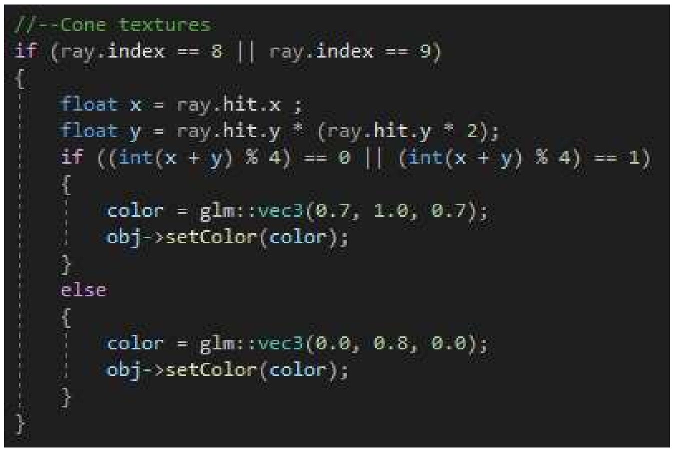


Figure 7: Procedurally Generated Patterns on Two Cones

This procedurally generated pattern is a several circles in striped fashion being rendered on the two cones. This was created using this snippet of code:



```
//--Cone textures
if (ray.index == 8 || ray.index == 9)
{
    float x = ray.hit.x ;
    float y = ray.hit.y * (ray.hit.y * 2);
    if ((int(x + y) % 4) == 0 || (int(x + y) % 4) == 1)
    {
        color = glm::vec3(0.7, 1.0, 0.7);
        obj->setColor(color);
    }
    else
    {
        color = glm::vec3(0.0, 0.8, 0.0);
        obj->setColor(color);
    }
}
```

Figure 8: Code Snippet for Implementing Procedurally Generated Patterns

Runtime

The program took 25 seconds to run without anti-aliasing and 100 seconds to run with super sampling anti-aliasing.

Successes

There is no single part of this project that stands out to me as a success, but I am very happy with how the scene has turned out and I have really enjoyed working on these two assignments.

Failures

Implementing the cone and cylinder intersection and surface normal vector functions took longer than I am willing to admit. I was really eager to try and create a torus, however with how long it took me to create the cone and cylinder I eventually gave up this idea.

This is less of a failure and more an issue with time constraints. I would have liked to implemented fog. I had an idea that I could calculate the length of the ray and a scale factor that would make it whiter the further away it was. This would have likely been easy enough to implement, however I have other class I have to attend to, and since I have enough extra features, I cannot justify the time. Although I would be interested in seeing if this is how other people would have implemented it.

References

1. UV Mapping Sphere Textures from Wikipedia
   https://en.wikipedia.org/wiki/UV_mapping
2. 2021 COSC363 Lectures and Labs