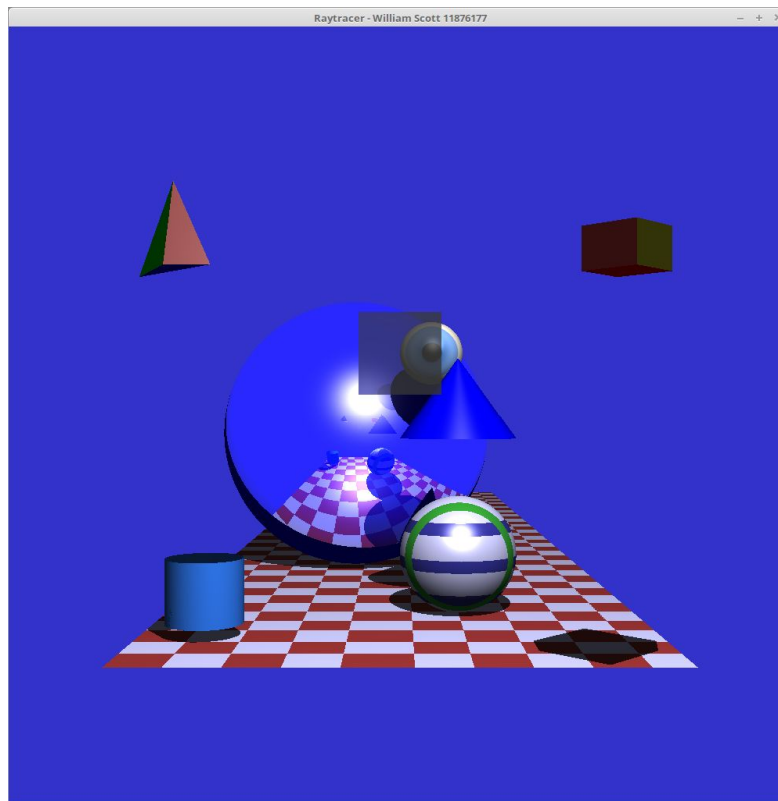


# The Basic Ray Tracer

The Ray Tracer can be compiled using the following g++ command:

```
g++ -o COSC363Assignment2 COSC363_Assignment2.cpp Cone.cpp Cylinder.cpp Plane.cpp  
Ray.cpp SceneObject.cpp Sphere.cpp TextureBMP.cpp -lm -lGL -lGLU -lglut
```

## Image of Scene

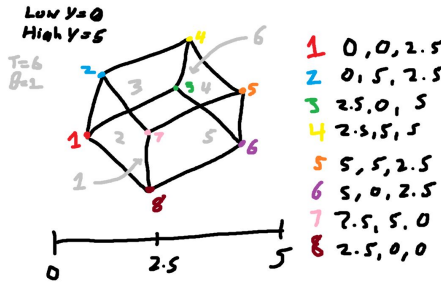
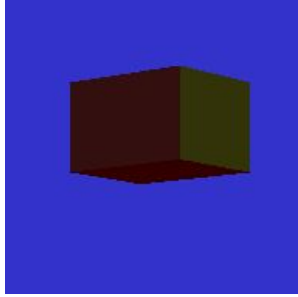


*Figure A. Image of Scene*

## Overview

In creating this basic ray tracer I built upon the existing code and work done in labs 7 and 8. I had no problems implementing lighting, shadows, and reflections, however I made an early mistake by determining specular reflections using the normal vector for “V”, instead of the inverse direction of the ray. Once I corrected this, what oddities I was experiencing with specular reflections cleaned up. In the next two sections I will detail my process in implementing the cube, and the floor of the scene.

## Scene Feature: Cube

Fig. B - Cube: Rough Blocking in MS Paint to determine points for planes	Fig. C - Cube in the Scene
 <p>Low Y=0 High Y=5 T=6 B=1</p> <p>1 0, 0, 2.5 2 0, 5, 2.5 3 2.5, 0, 5 4 2.5, 5, 5 5 5, 5, 2.5 6 5, 0, 2.5 7 2.5, 5, 0 8 2.5, 0, 0</p>	

To create the cube, I implemented a supporter function `void drawCube(void)`, rather than creating a new class. In this function I designated 8 points for use in the creation of 6 planes, making up the cube. An offset is incorporated into each point, allowed for easy movement of the cube around the scene. I do not think this is the best approach, as adding additional cubes to the scene would require substantial copy-pasting, however with some planning as seen in Fig. B, it was overall easy to implement.

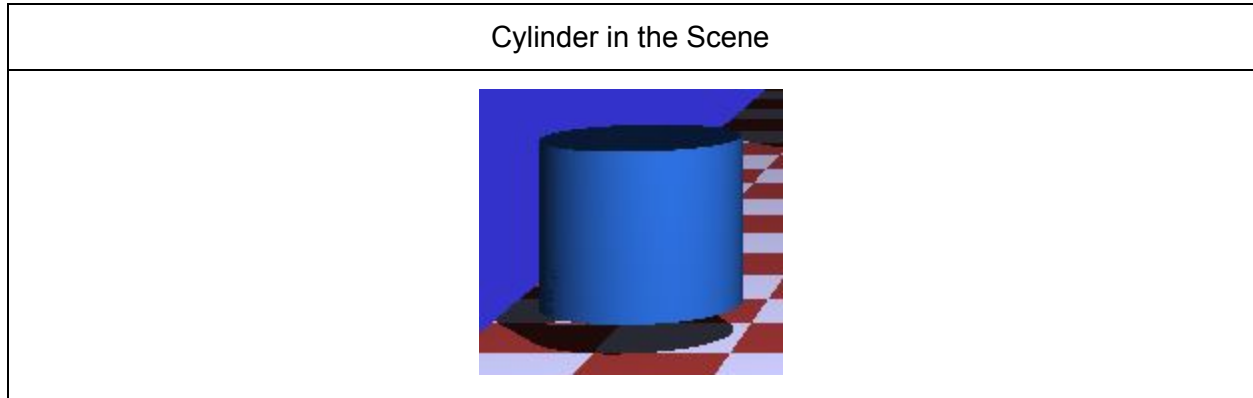
## Scene Feature: Chequered Floor

To chequer the floor plane of the scene, rather than using a texture I identified the plane during the trace function and updated the color of the plane based on parity of the sum of the x point and z point. I found that adding an offset to the points and a divider, helped to both eliminate a wide stripe occurring in the middle of the floor, as well as increase the scale of the squares which I found more aesthetically pleasing.

COSC363_Assignment2.cpp - Code Extract, Chequered Floor
<pre>// Chequered Floor if (ray.xindex == 0) {     if (((int(ray.xpt.x) - 1000) / 4) + ((int(ray.xpt.z) - 1000) / 4)) % 2 == 0)     {         color = glm::vec3(0.6f, 0.2f, 0.2f);     }     else     {         color = glm::vec3(0.8f, 0.8f, 1.f);     } }</pre>

## Extensions

### Primitive: Cylinder



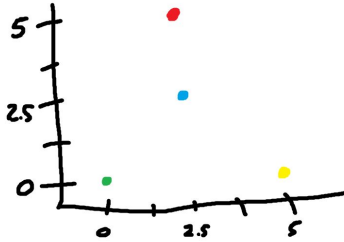
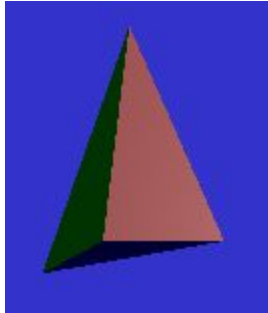
With the help of the formulas provided in Lecture 9 (Pgs. 30 - 31) I was able to implement a cylinder class extending from `SceneObject`, allowing for cylinders to be added to the scene. As most of the formulas were provided, the only challenge was capping the cylinder with the min and max to prevent it from extending infinitely. At one point I thought I had it working, however it was still extending out of the bottom of the scene - this is why I decided to make the cylinder float slightly above the ground.

### Primitive: Cone

Adding the cone, although very similar to the cylinder proved to be one of the most difficult additions to the program. The formula for the cone's normal was provided in Lecture 9 (Slide 32), however the intersection equation had to be found by substituting the ray equation provided on Slide 33. After a few lengthy attempts I managed to get the cone working with the following intersection equation.

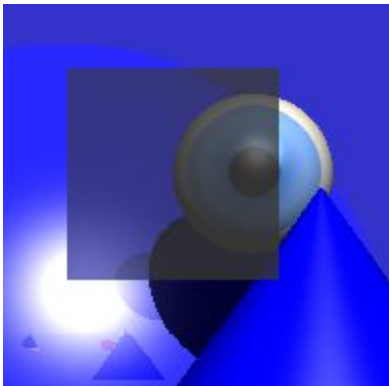
Cone Intersection Equation:
$t^2 \{d_x^2 + d_z^2 - (\frac{R}{h})^2 * d_y^2\} + 2t \{(x_o * d_x - x_c * d_x) + (z_o * d_z - z_c * d_z) - ((\frac{R}{h})^2 * y_o * d_y) \dots$ $\dots - ((\frac{R}{h})^2 * y_c * d_y) - ((\frac{R}{h})^2 * h * d_y)\} + \{(x_o - x_c)^2 + (z_o - z_c)^2 - (\frac{R}{h})^2 * ((2h * y_c) \dots$ $\dots - (2h * y_o) - (2y_o * y_c) + y_o^2 + y_c^2 + h^2)\}$

### Primitive: Tetrahedron

Tetrahedron: Rough Blocking in MS Paint to determine points for planes	Tetrahedron in the Scene
	

To create the Tetrahedron in the scene I used a collection of 4 planes where one of the points was a duplicate to create triangles. This is certainly not the best solution to the problem, and in retrospect I think it would be fun to attempt adding this as its own class.

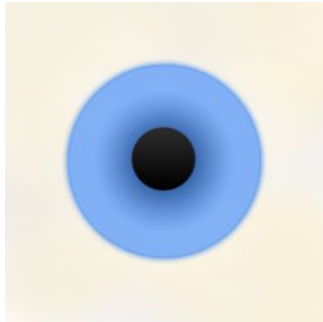
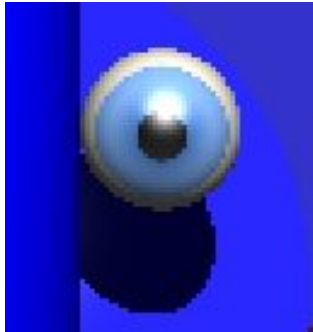
### Transparent Object - The Window

The Window in the Scene


Transparency was my biggest failure in the creation of the ray tracer as I could not get transparency working nicely with non-planar objects. I experimented with a few different techniques, but it was either not accurate looking, or only offered transparency through one side of the object. I implemented transparency similarly to reflections as detailed in Lab 8.

## Non-planar Object Textured Using Image

For texturing a non-planar object, I decided to try texturing a sphere to resemble an eyeball. The texture 'eye.bmp' was created by myself using Adobe Photoshop, and loaded and interacted with in code using the provided TextureBMP module.

'eye.bmp'	Example of Eye in Scene
	

COSC363_Assignment2.cpp - Code Extract, Textured Sphere
<pre>// Textured Sphere if (ray.xindex == 2) {     float pi = 3.14159;     float u = (atan2(normalVector.x, normalVector.z) / (pi)) + 0.5;     float v = 0.5 - asin(normalVector.y) / pi;     color = texture1.getColorAt(u, v); }</pre>
Above Code Based On: Formula for finding UV on a Sphere
$u = 0.5 + \frac{\arctan 2(d_z, d_x)}{2\pi}$ $v = 0.5 - \frac{\arcsin(d_y)}{\pi}$ <p>Source: <a href="https://en.wikipedia.org/wiki/UV_mapping#Finding_UV_on_a_sphere">https://en.wikipedia.org/wiki/UV_mapping#Finding_UV_on_a_sphere</a></p>

## Non-planar Object Textured Using a Procedural Pattern

Procedural Pattern Sphere in Scene



Finally the last extension I implemented with texturing of a non-planar object using a procedural pattern. Using basic modulo math I was able to add two different colors of rings going around a sphere on top of a base color.

COSC363\_Assignment2.cpp - Code Extract, Procedural Pattern Sphere

```
// Procedural Pattern
if (ray.xindex == 3)
{
    if (((int)ray.xpt.z) % 3 == 0)
    {
        color = glm::vec3(0.2f, 0.6f, 0.2f); // Ring 1 color
    }
    else if (((int)ray.xpt.y) % 3 == 0)
    {
        color = glm::vec3(0.2f, 0.2f, 0.6f); // Ring 2 color
    }
    else
    {
        color = glm::vec3(0.8f, 0.8f, 1.f); // Base color
    }
}
```

## Resources & References

- Adobe Photoshop for the creation of 'eye.bmp' used in texturing a non-planar object.
- Formula for finding UV Coordinates on a Sphere on Wikipedia:  
[https://en.wikipedia.org/wiki/UV\\_mapping#Finding\\_UV\\_on\\_a\\_sphere](https://en.wikipedia.org/wiki/UV_mapping#Finding_UV_on_a_sphere)
- Files provided in COSC363's Labs 8 and 9
- Lecture Notes by Mukundan, Department of Computer Science and Software Engineering University of Canterbury, Christchurch, New Zealand.