Name: Jack Hay
Student ID: 82432942

COSC363: Computer Graphics Assignment 2 Report

Build Command: g++ -Wall -o "%e" "%f" Plane.cpp Cylinder.cpp Ray.cpp Sphere.cpp SceneObject.cpp TextureBMP.cpp -lGL -lGLU -lglut -lGLEW

Scene Description:
This scene contains 3 spheres, a cube, a cylinder and a plane as the floor. There are shadows on all objects created from the light sources. Two of the three spheres are reflective The cube is created from 6 planes, and the floor of the scene has stripes as a texture.

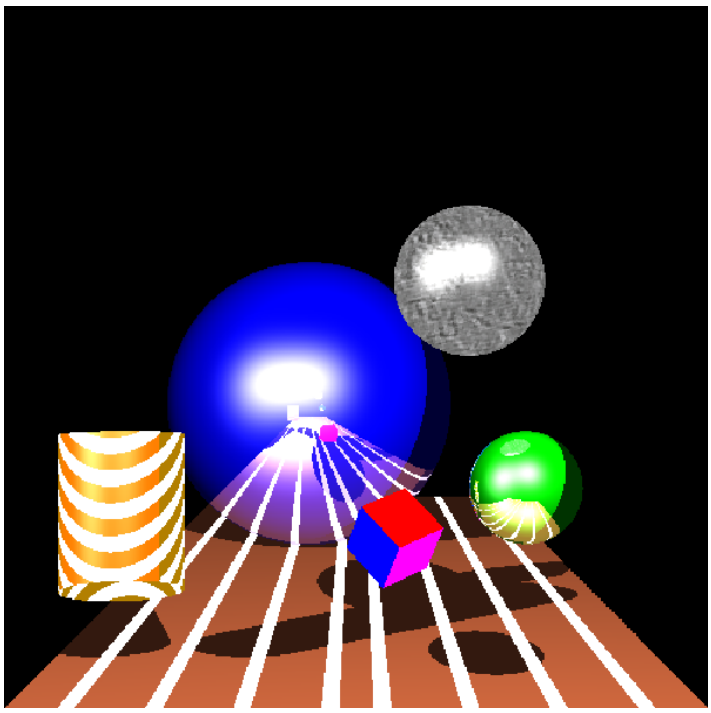

Figure 1: Scene containing objects.

Extra features:

Cylinder:

Ray equation from lecture notes as well as the intersection equation for the cylinder was used.

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

Name: Jack Hay
Student ID: 82432942

The above equation was solved for t:

```
//quadratic equations
float quadratic1 = (-b - sqrt(b*b - 4*(a*c)))/(2 * a);
float quadratic2 = (-b + sqrt(b*b - 4*(a*c)))/(2 * a);

float yPos = position.y + dir.y*quadratic2;

if((yPos >= center.y) && (yPos <= center.y + height)){
    return quadratic1;
} else if (quadratic2< 0.01) quadratic2 = -1;

yPos = position.y + dir.y*quadratic1;
if ((yPos >= center.y) && (yPos <= center.y + height)) {
    return quadratic2;
} else {
    return -1;
}
```

The normal was calculated using glm::normalize on $x - x_c$, 0, $z - z_c$. These equations were obtained from the lecture notes.

```
glm::vec3 Cylinder::normal(glm::vec3 p)
{
    glm::vec3 normal = glm::vec3((p-center).x, 0, (p-center).z);
    return glm::normalize(normal); //normalize
}
```

Multiple Light Sources:
A second light source was added and shifted along the negative x-axis from the original light source. The light can be seen refracting off the spheres. Both light sources have the same intensity, as I found that making them any brighter caused too large of a glare. The three images below show the scene with the original light source, the second light source, and then both light sources.

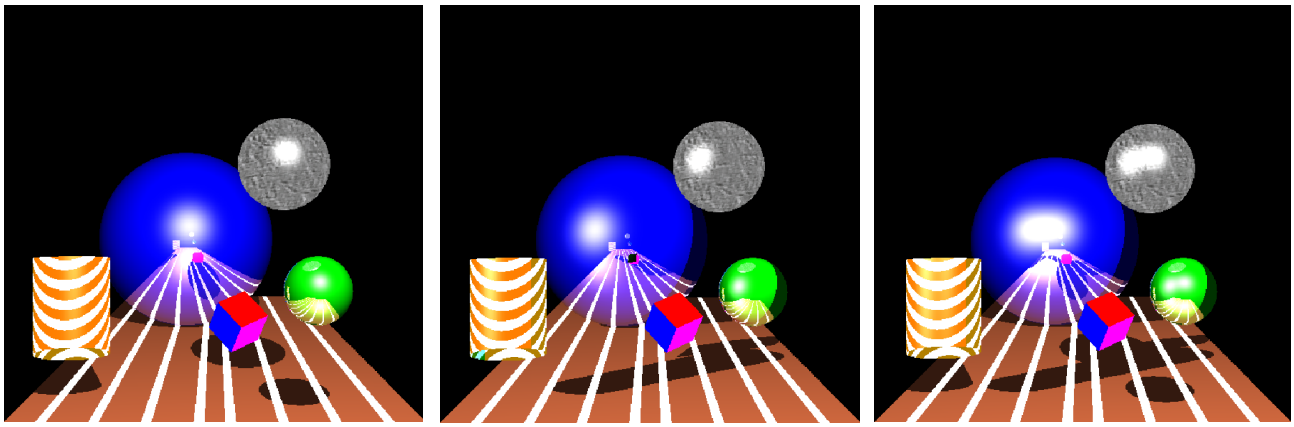

Figure 2: Original light source (Left). Second light source (Middle). Both light sources (Right). The above figures also demonstrate the generated shadows.

An object transformed using rotation:
The below cube has been rotated 60 degrees about the x and y axis. The rotation was performed by creating a rotation matrix, then multiplying each vec3 point in the cube by this rotation matrix. To do this, I had to convert the vec3 to a vec4, so that I could multiply it by the matrix. It was then converted back to a vec3 and each plane was made in the cube with the new rotated vector.

```
glm::mat4 im = glm::mat4(1.0f);
glm::mat4 rotation = glm::rotate(im, THETA, glm::vec3(1, 1, 0));
```

Rotation matrix with a theta of 60 degrees.

Name: Jack Hay
Student ID: 82432942

```
// -- Cube
Plane *cubeTop = new Plane (glm::vec3(rotation * glm::vec4(cubeA, 1)), glm::vec3(rotation * glm::vec4(cubeB, 1)),
glm::vec3(rotation * glm::vec4(cubeC, 1)), glm::vec3(rotation * glm::vec4(cubeD, 1)), glm::vec3(128,0,0));

Plane *cubeBottom = new Plane (glm::vec3(rotation * glm::vec4(cubeE, 1)), glm::vec3(rotation * glm::vec4(cubeF, 1)),
glm::vec3(rotation * glm::vec4(cubeG, 1)), glm::vec3(rotation * glm::vec4(cubeH, 1)), glm::vec3(128,0,0));

Plane *cubeLeft = new Plane (glm::vec3(rotation * glm::vec4(cubeA, 1)), glm::vec3(rotation * glm::vec4(cubeD, 1)),
glm::vec3(rotation * glm::vec4(cubeH, 1)), glm::vec3(rotation * glm::vec4(cubeE, 1)), glm::vec3(0,0,128));

Plane *cubeRight = new Plane (glm::vec3(rotation * glm::vec4(cubeB, 1)), glm::vec3(rotation * glm::vec4(cubeC, 1)),
glm::vec3(rotation * glm::vec4(cubeG, 1)), glm::vec3(rotation * glm::vec4(cubeF, 1)), glm::vec3(128,0,128));

Plane *cubeFront = new Plane (glm::vec3(rotation * glm::vec4(cubeD, 1)), glm::vec3(rotation * glm::vec4(cubeC, 1)),
glm::vec3(rotation * glm::vec4(cubeG, 1)), glm::vec3(rotation * glm::vec4(cubeH, 1)), glm::vec3(128,0,128));

Plane *cubeBack = new Plane (glm::vec3(rotation * glm::vec4(cubeA, 1)), glm::vec3(rotation * glm::vec4(cubeB, 1)),
glm::vec3(rotation * glm::vec4(cubeF, 1)), glm::vec3(rotation * glm::vec4(cubeE, 1)), glm::vec3(128,0,128));
```

The above code demonstrates each vector being multiplied by the rotation matrix, then the planes being created with the new rotated vectors.

A non-planar object textured using an image:
The top right sphere is textured by wall.bmp file.

```
// -- image texture
if (ray.xindex == 2) {
    float s = (ray.xpt.x+50)/100;
    float t = (ray.xpt.y+20)/70;
    colorSum = texture.getColorAt(s,t);
}
```

A non-planar object textured using a procedural patter:
The cylinder has a procedurally generated texture on it. As the cylinder is placed at eye level, I had to solve the problem of only the top half of it being textured. This was due for my code only equating for the positive y values. Once I figured this out I just had to add on OR to the if statement that also returned a white color for a negative value.

```
// -- Cylinder Texture
if (ray.xindex == 1) {
    if ((int(ray.xpt.y+ray.xpt.z-4)) % 2 == 1 || int(ray.xpt.y+ray.xpt.z-4) % 2 == -1) {
        colorSum = glm::vec3(1, 1, 1);
    } else {
        colorSum = glm::vec3(0.5, 0.5, 0);
    }
}
```

Successes and Failures.
A pretty obvious failure is the base of my cylinder. I could not figure out how to correctly place this, or the ray tracing isn't correct at these points. After looking through my code I believe that the intersection equation demonstrated in my report was incorrectly solved for t.

Another failure is the slightly inaccurate mapping of the image onto the sphere. I have stretched the image too much so it is fuzzy. To fix this, I would have to calculate more precise points using tan and sin.

Name: Jack Hay
Student ID: 82432942

One success was my translated cube. It took a while to get it centred on screen, as when I first created it, it was not rotated. After rotating it about the x and y axis it ended up slightly off-screen. However, performing some simple translations got it right back centre stage.

References:

- https://www.khronos.org/opengl/wiki/Texturing_a_Sphere (Texturing the sphere)
- COSC363 Lecture notes