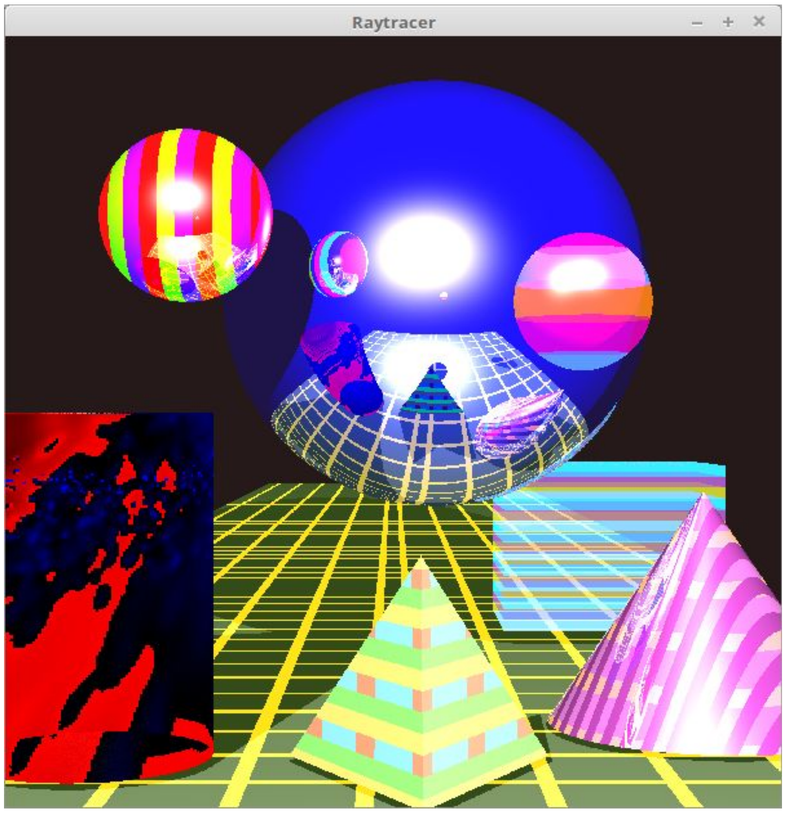
COSC363: Computer Graphics Assignment 2
Name: Matthew Belworthy Lewthwaite
Student ID: 11030423

Ray Tracer



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

Extensions:

Primitives (Cone , Cylinder , Tetrahedron) (3 marks)

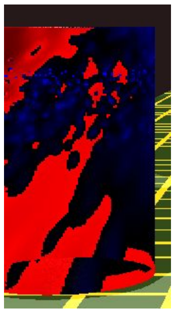


Figure 1: Cylinder

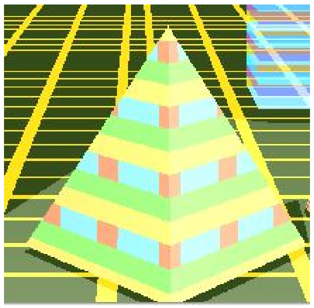
A cylinder class was made to produce what can be seen in Figure 1. Intersections were calculated through the 3 points, with help of the position and direction of a point, generated points of the primitive shape. The intersection of the 3 points was calculated by the following code, where dir is direction, pos is position and rad length is the length of the desired radius:

```
glm::vec3 d = pos - origin;
float a = (dir.x*dir.x)+(dir.z*dir.z);
float b = 2*(dir.x*d.x+dir.z*d.z);
float c = d.x*d.x+d.z*d.z-(radLength*radLength);
```

The surface normal vector was calculated by the following code:

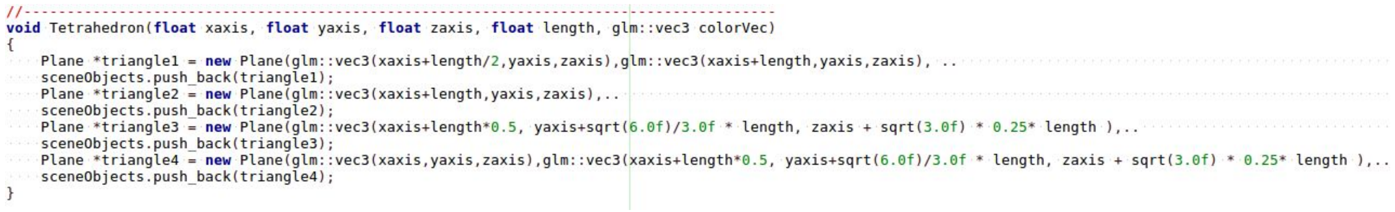
```
glm::vec3 Cylinder::normal(glm::vec3 p)
{
    glm::vec3 d = p - origin;
    glm::vec3 n = glm::vec3 (d.x,0,d.z);
    n = glm::normalize(n);
    return n;

}
```

COSC363: Computer Graphics Assignment 2
Name: Matthew Belworthy Lewthwaite
Student ID: 11030423

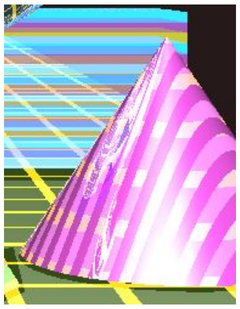


The tetrahedron was produced by a function that took in a set of axis, a length and a colour. The tetrahedron is made of 4 planes. As planes are made of 4 points, the third point is located along the intersection of two of the points, thus generating triangles and not squares. The code used to generate this primitive is shown below:

Figure 2: Tetrahedron

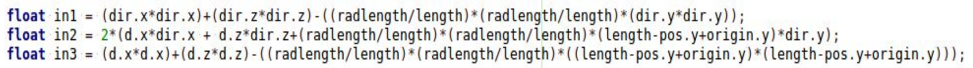


```
//-------------------------------------------------------------------------------
void Tetrahedron(float xaxis, float yaxis, float zaxis, float length, glm::vec3 colorVec)
{
    Plane *triangle1 = new Plane(glm::vec3(xaxis+length/2,yaxis,zaxis),glm::vec3(xaxis+length,yaxis,zaxis), ..
    sceneObjects.push_back(triangle1);
    Plane *triangle2 = new Plane(glm::vec3(xaxis+length,yaxis,zaxis),..
    sceneObjects.push_back(triangle2);
    Plane *triangle3 = new Plane(glm::vec3(xaxis+length*0.5, yaxis+sqrt(6.0f)/3.0f * length, zaxis + sqrt(3.0f) * 0.25* length ),..
    sceneObjects.push_back(triangle3);
    Plane *triangle4 = new Plane(glm::vec3(xaxis,yaxis,zaxis),glm::vec3(xaxis+length*0.5, yaxis+sqrt(6.0f)/3.0f * length, zaxis + sqrt(3.0f) * 0.25* length ),..
    sceneObjects.push_back(triangle4);
}
```



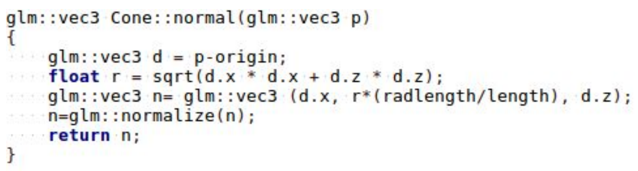
The cone was generated almost the same way as the cylinder (as they have the same base) but with mild alterations. Intersections were calculated through the 3 points, with help of the position and direction of a point, generated points of the primitive shape. The intersection of the 3 points was calculated by the following code, where dir is direction, pos is position and rad length is the length of the desired radius at the base:

Figure 3: Cone



```
float in1 = (dir.x*dir.x)+(dir.z*dir.z)-((radlength/length)*(radlength/length)*(dir.y*dir.y));
float in2 = 2*(d.x*dir.x + d.z*dir.z+(radlength/length)*(radlength/length)*(length-pos.y+origin.y)*dir.y);
float in3 = (d.x*d.x)+(d.z*d.z)-((radlength/length)*(radlength/length)*((length-pos.y+origin.y)*(length-pos.y+origin.y)));
```

The surface normal vector was calculated by the following code:



```
glm::vec3 Cone::normal(glm::vec3 p)
{
    glm::vec3 d = p-origin;
    float r = sqrt(d.x * d.x + d.z * d.z);
    glm::vec3 n= glm::vec3 (d.x, r*(radlength/length), d.z);
    n=glm::normalize(n);
    return n;
}
```

COSC363: Computer Graphics Assignment 2
Name: Matthew Belworthy Lewthwaite
Student ID: 11030423

Multiple Light Sources (1 mark)

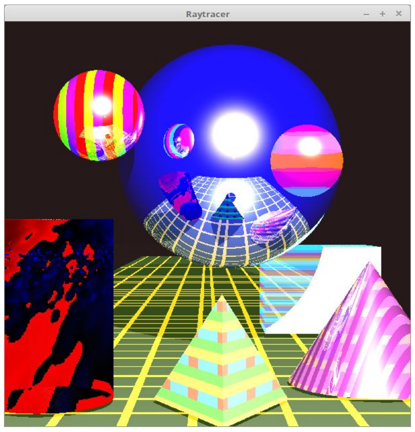


Figure 4: One Light Source

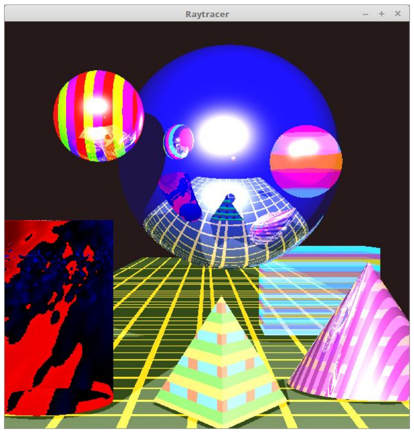


Figure 5: Two Light Sources

A second light source was added at the coordinates (-30, 40, -3), this being -40 units across the x axis over from the initial light source. This was achieved by normalizing two different light vectors, generating two separate reflection vectors and also two separate shadow rays. This was generated the same was as the initial light source was. The two sets of shadows are evident when comparing Figure 4 to Figure 5.

Transparent Object (1 mark)

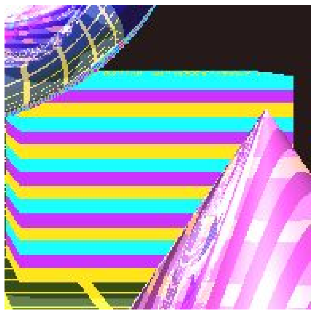


Figure 6:Opaque cube

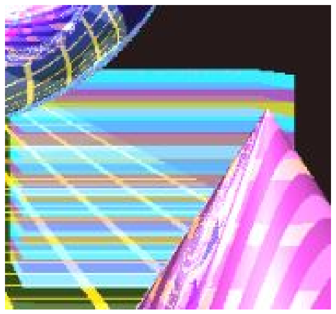


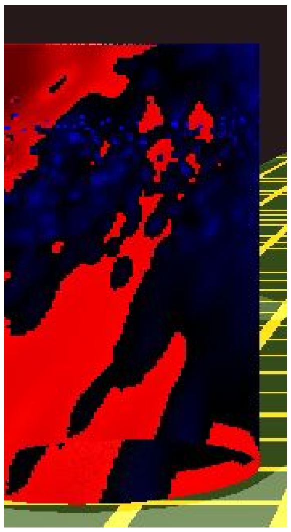
Figure 7: Transparent cube

The cube is transparent with a procedural pattern covering it, this was achieved by amplifying the cubes current procedural pattern (that was also reduced) by the transparency color, as shown in Figure 6 and Figure 7. The cubes transparency was generated with the following code:

```
Ray transRay(ray.xpt, ray.dir);
glm::vec3 transColor = trace(transRay, step+1);
colorSum = 0.4f*colorSum + 0.6f*transColor;
```

COSC363: Computer Graphics Assignment 2
Name: Matthew Belworthy Lewthwaite
Student ID: 11030423

Non-Planar Object Textured Using An Image (1 mark)



The cylinder was textured using void.bmp. This was achieved by the following code:

```
float s = (ray.xpt.x+49)/99;
float t = (ray.xpt.y+21)/71;
colorSum = texture.getColorAt(s,t);
```

Figure 8: bmp textured cylinder

Non-Planar Object Textured Using A Procedural Pattern (2 mark)

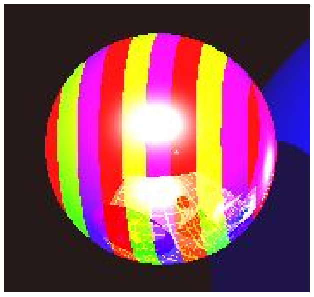


Figure 9

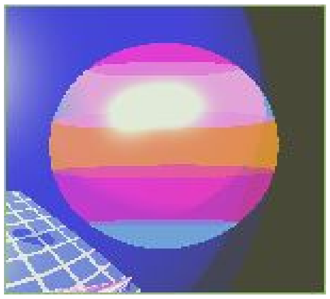


Figure 10

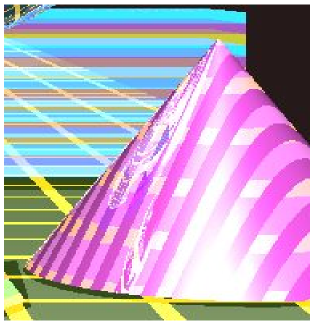


Figure 11

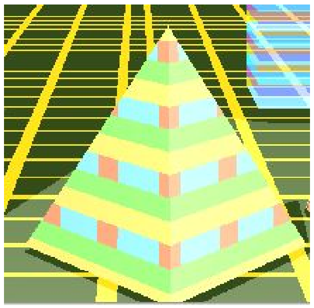
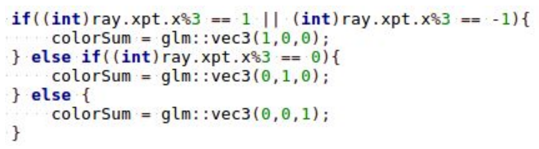


Figure 12

COSC363: Computer Graphics Assignment 2
Name: Matthew Belworthy Lewthwaite
Student ID: 11030423

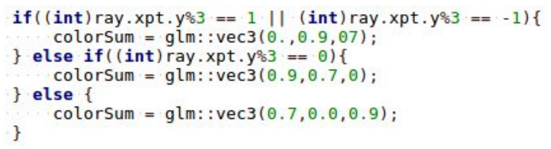
Procedural Texturing was experimented with and shown across 5 Non-Planar Objects to achieve a range of different patterns by coloring the objects along the x and y axis, as well as diagonally in some cases. This was achieved by the following parts of code:

Figure 9:



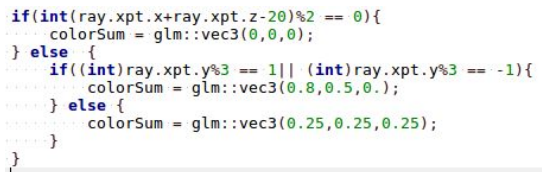
```
if((int)ray.xpt.x%3 == 1 || (int)ray.xpt.x%3 == -1){
    colorSum = glm::vec3(1,0,0);
} else if((int)ray.xpt.x%3 == 0){
    colorSum = glm::vec3(0,1,0);
} else {
    colorSum = glm::vec3(0,0,1);
}
```

Figure 10:



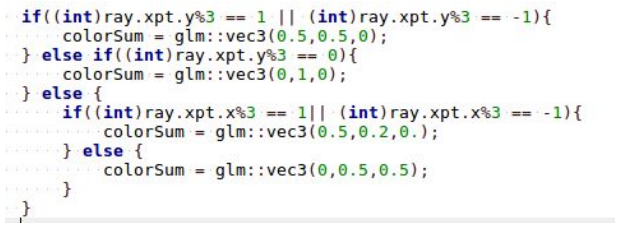
```
if((int)ray.xpt.y%3 == 1 || (int)ray.xpt.y%3 == -1){
    colorSum = glm::vec3(0.,0.9,07);
} else if((int)ray.xpt.y%3 == 0){
    colorSum = glm::vec3(0.9,0.7,0);
} else {
    colorSum = glm::vec3(0.7,0.0,0.9);
}
```

Figure 11:



```
if(int(ray.xpt.x+ray.xpt.z-20)%2 == 0){
    colorSum = glm::vec3(0,0,0);
} else  {
    if((int)ray.xpt.y%3 == 1|| (int)ray.xpt.y%3 == -1){
        colorSum = glm::vec3(0.8,0.5,0.);
    } else {
        colorSum = glm::vec3(0.25,0.25,0.25);
    }
}
```

Figure 12:



```
if((int)ray.xpt.y%3 == 1 || (int)ray.xpt.y%3 == -1){
    colorSum = glm::vec3(0.5,0.5,0);
} else if((int)ray.xpt.y%3 == 0){
    colorSum = glm::vec3(0,1,0);
} else {
    if((int)ray.xpt.x%3 == 1|| (int)ray.xpt.x%3 == -1){
        colorSum = glm::vec3(0.5,0.2,0.);
    } else {
        colorSum = glm::vec3(0,0.5,0.5);
    }
}
```

<u>Successes and Failures:</u>

I would say that a majority of my extra features were successes, these being the primitive objects, the additional light source, the transparent cube and procedural patterning. These were all generated how I initially imagined.

The only thing that was even remotely a 'failure' was my non-planar object textured using an image. The image that was used is meant to be a picture of a void, though this isn't too obvious. I wasn't sure how to approach this task, and I couldn't find all too much anywhere that helped with the problems I faced. Here I compensated by moving the image around to a place where I felt looked the best

<u>References:</u>

- COSC363 - Computer Graphics lecture notes and lab material.