Student ID: 27033004
Name: Jonathan Edwards

COSC363 Assignment 2 Report

<u>Build Command:</u> **Linux** = cd Linux && make, **Windows** = cd Windows && make

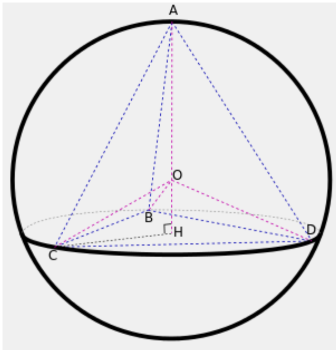


Figure 1- https://en.wikipedia.org/wiki/Tetrahedron

<u>Features:</u>

1. **<u>Tetrahedron</u>**
   - The Tetrahedron is drawn as shown
   - Constructed 4 faces using the plane class.

```
// 4 different faces
Plane *face1 = new Plane(A,B,C);
Plane *face2 = new Plane(C,D,B);
Plane *face3 = new Plane(B,D,A);
Plane *face4 = new Plane(C,A,D);
```

<u>Ray equation</u> $x = x_0 + d_x t; \quad y = y_0 + d_y t; \quad z = z_0 + d_z t;$

- Sub Ray equation(s) in to intersection equation to find roots.
- Related to specific object: d = direction, o = origin, c = centre, t = distance from ray's origin to point of ray

2. **<u>Cone</u>**
   - Cone class was created in *Cone.h* with a *intersect* and a *normal* method.
   - <u>Normal Method</u>:
     Normalize's the given point and returns a unit normal vector
   - <u>Intersect Method</u>:
     Contains the intersection equation of the cone object and finds the roots below

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

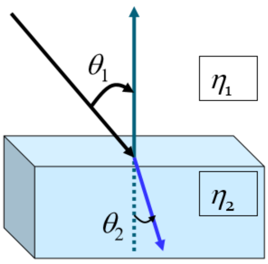
```
float a = SQUARE(dir.x) + SQUARE(dir.z) - (SQUARE(dir.y)*SQUARE((radius/height)));
float b = 2*(MAG(pos).x*dir.x + MAG(pos).z*dir.z + (SQUARE((radius/height))*DIR_POINT(height, pos.y, center.y)*dir.y));
float c = SQUARE(MAG(pos).x) + SQUARE(MAG(pos).z) - (SQUARE((radius/height))*DIR_POINT(height, pos.y, center.y));
```

3. **<u>Cylinder</u>**
   - Cylinder class was created in *Cylinder.h* with a *intersect* and a *normal* method.
   - <u>Normal Method</u>:
     Normalize's the given point and returns a unit normal vector
   - <u>Intersect Method</u>:
     Contains the intersection equation of the cone object and finds the roots below

```
float a =  SQUARE(dir.x) + SQUARE(dir.z);;
float b = 2*(DIR_POINT(dir.x, pos.x, center.x) + DIR_POINT(dir.z, pos.z, center.z));
float c = SQUARE((pos.x-center.x)) + SQUARE((pos.z - center.z)) - (radius*radius);
```

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

4. **<u>Refraction and Transparency</u>**
   - Algorithm:
     ray is traced twice, need 2 normals and refractive rays
   - This is described using Snell's Law



Snell's Law of Refraction:

- $\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2$

Figure 4.1 – Taken from lecture 8, slide 19

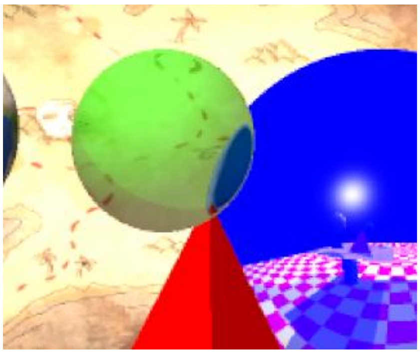


Figure 2: ETA=1.003

Transparent

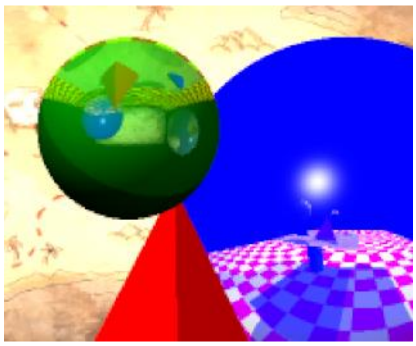
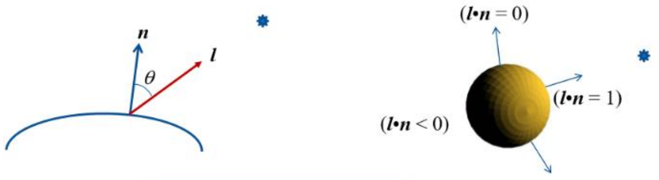


Figure 3: ETA=1.6

Refraction

Student ID: 27033004
Name: Jonathan Edwards



$$I_d = L_d M_d \max(\boldsymbol{l} \bullet \boldsymbol{n}\ , 0)$$

L_d: Light's diffuse color
M_d: Material's diffuse color

Figure 5 – Taken from lecture 7, slide 5

```
// Light sources
#define      LIGHT_SOURCE_1          glm::vec3(20, 40, -3)
#define      LIGHT_SOURCE_2          glm::vec3(-20, 30, -20)
```

```
// -- Transparency and Refraction -------------------------------------------------------
if (step < MAX_STEPS && ray.index == TRANSPARANT_SPHERE) {
    // pg 19-22
    // 1. Find the first refraction vector
    // 2. Find the second refraction vector by the refracting the negative of the first normal
    // 3. Trace the color
    // Note: Refracted ray - use snells law

    // first vector
    glm::vec3 normalVec = obj->normal(ray.hit);
    glm::vec3 dirFrac = glm::refract(ray.dir, normalVec, (1.0f/ETA));
    Ray rayFrac1(ray.hit, dirFrac);
    rayFrac1.closestPt(sceneObjects);

    // second vector
    normalVec = sceneObjects[rayFrac1.index]->normal(rayFrac1.hit);
    dirFrac = glm::refract(dirFrac, -normalVec, ETA);
    Ray rayFrac2(rayFrac1.hit, dirFrac);
    rayFrac2.closestPt(sceneObjects);

    // trace
    glm::vec3 refracColor = trace(rayFrac2, step+1);
    color = TRANSPARANT * (color+refracColor);
}
```

Figure 4.2 – Code for part 4

5. **Light Sources – shadows**
   - There are two light sources in the scene.
   - Light source 1 reflections are made up of *ambient + diffuse* (shown in fig 5 and 6) and a brightness of 30 %
   - Light source 2 is made up of *ambient + diffuse + specular* but reflections are just *ambient.* At 100 %

```
if ((lDotn <= 0) || (rayShadow1.index > -1 && (rayShadow1.dist < magLight)))
    color += AMBIENT*(lDotn*currColor) + glm::vec3((BRIGHT_LEVEL/10.f))*(lDotn*currColor);  // Shadow overlap - make darker
else
    color += AMBIENT*currColor + BRIGHT_LEVEL*(lDotn*currColor);                             // default
```

Figure 6 – First two lines controls shaded area – shown below

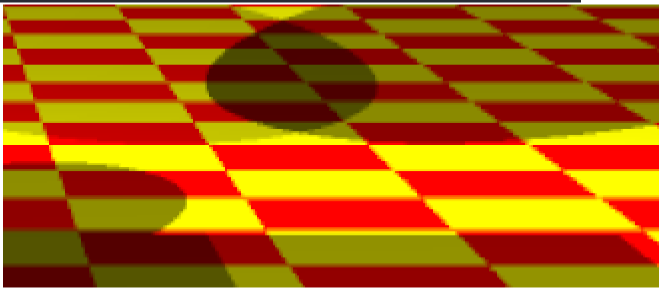
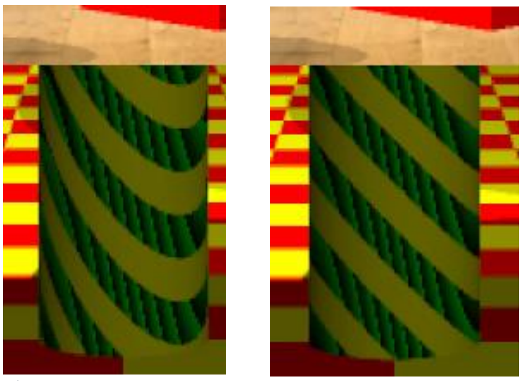


Figure 7 – Shadows overlapping

6. **Non-Planar object textured**
   Earth.bmp is mapped to the sphere – algorithm in function – *texS* and *texT* lines

```
void rayWorldGlobe(SceneObject *obj, Ray ray, TextureBMP texture)
{
    glm::vec3 color(0);

    if (ray.index == WORLD_GLOBE) {
        glm::vec3 center = WORLD_GLOBE_POS;           // Centre of the sphere obj - to be mapped
        glm::vec3 d = glm::normalize(ray.hit-center);

        // Find coord pos - s and t
        float texS = (0.5-atan2(d.z, d.x)+M_PI) / (2*M_PI);  // X coord of the normal
        float texT = acos(-d.y) / M_PI;                      // Y coordinate of the normal

        color = texture.getColorAt(texS, texT);
        obj->setColor(color);
    }
}
```

7. **Non-Planar object textured pattern**



2nd degree slope- quadratic

```
// By adding the hit.z - adds a second degree slope
if ((int)(ray.hit.x+ray.hit.y+ray.hit.z) % 2 == 0)
    obj->setColor(glm::vec3(0, patternCount/100.f, 0));
else
    obj->setColor(glm::vec3(1, 1, 0));

if (++patternCount > 100) patternCount = 0;
```

The Cylinder is textured by the code above
- Note: patternCount is random and changes the color pattern when the window is clicked on or moved – this is because ray is refreshed.

Student ID: 27033004
Name: Jonathan Edwards

8. **Anti-Aliasing – Super Sampling**

Algorithm:

1. Square Pixel is divided in to 4 sub pixels – 1 beam.
2. Two beams are generated by dividing again.
3. 4 beams generated by dividing by a quarter and a half i.e 0.75

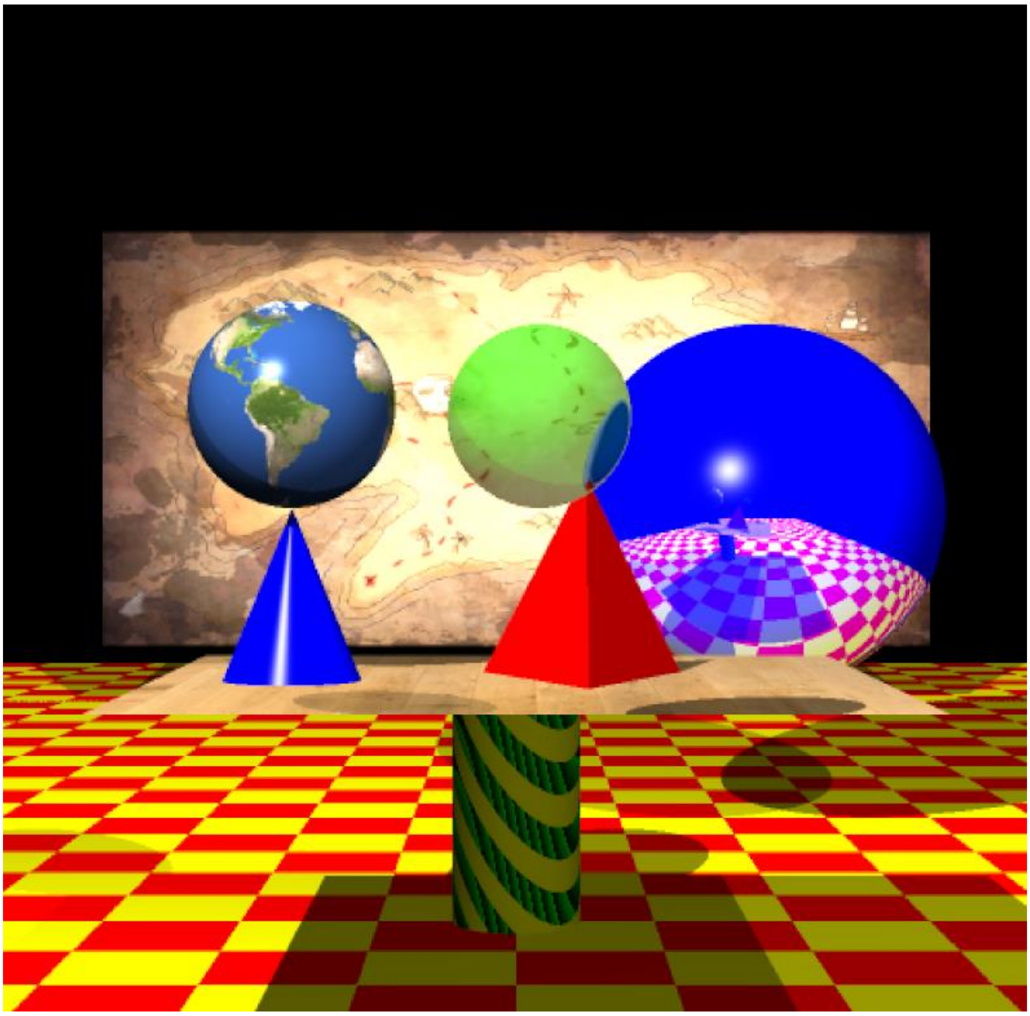
Figure 8 shows Aliasing enabled and 9 is with no aliasing

```cpp
// -----------------------------------------------------------------------------
//                          Anti-Aliasing Algorithm - Supersampling
//
//  - Generate several rays through each square pixel (eg. divide the pixel into
//    four equal segments) and compute the average of the colour values.
//
//   - returns the average color value of 4 pixels from one pixel one - pg 34 lec 8
// -----------------------------------------------------------------------------
glm::vec3 antiAlising(Ray ray, glm::vec3 eye, float size, float xp, float yp)
{
    float quarter = size * (1/SAMPLES);
    float quarterHalf = size * (1 -(1.f/SAMPLES));

    glm::vec3 colorTot(0);

    // Divide ray into quaters - 1 beam in middle
    ray = Ray(eye, glm::vec3(xp+quarter, yp+quarter, -EDIST));
    colorTot += trace(ray, 1);

    // Divide quaters  - 2 beams
    ray = Ray(eye, glm::vec3(xp+quarter, yp+quarterHalf, -EDIST));
    colorTot += trace(ray, 1);
    ray = Ray(eye, glm::vec3(xp+quarterHalf, yp+quarter, -EDIST));
    colorTot += trace(ray, 1);

    // divide half quater - 4 beams
    ray = Ray(eye, glm::vec3(xp+quarterHalf, yp+quarterHalf, -EDIST));
    colorTot += trace(ray, 1);

    return colorTot /= SAMPLES;
}
```

```cpp
// Trace the primary ray and get the colour value - Anti-Alising algorithm used below
// glm::vec3 col = trace (ray, 1);
glm::vec3 col = antiAlising(ray, eye, cellX, xp, yp);
```

Code and Usability

Fig 8 - Aliasing enabled



Student ID: 27033004
Name: Jonathan Edwards

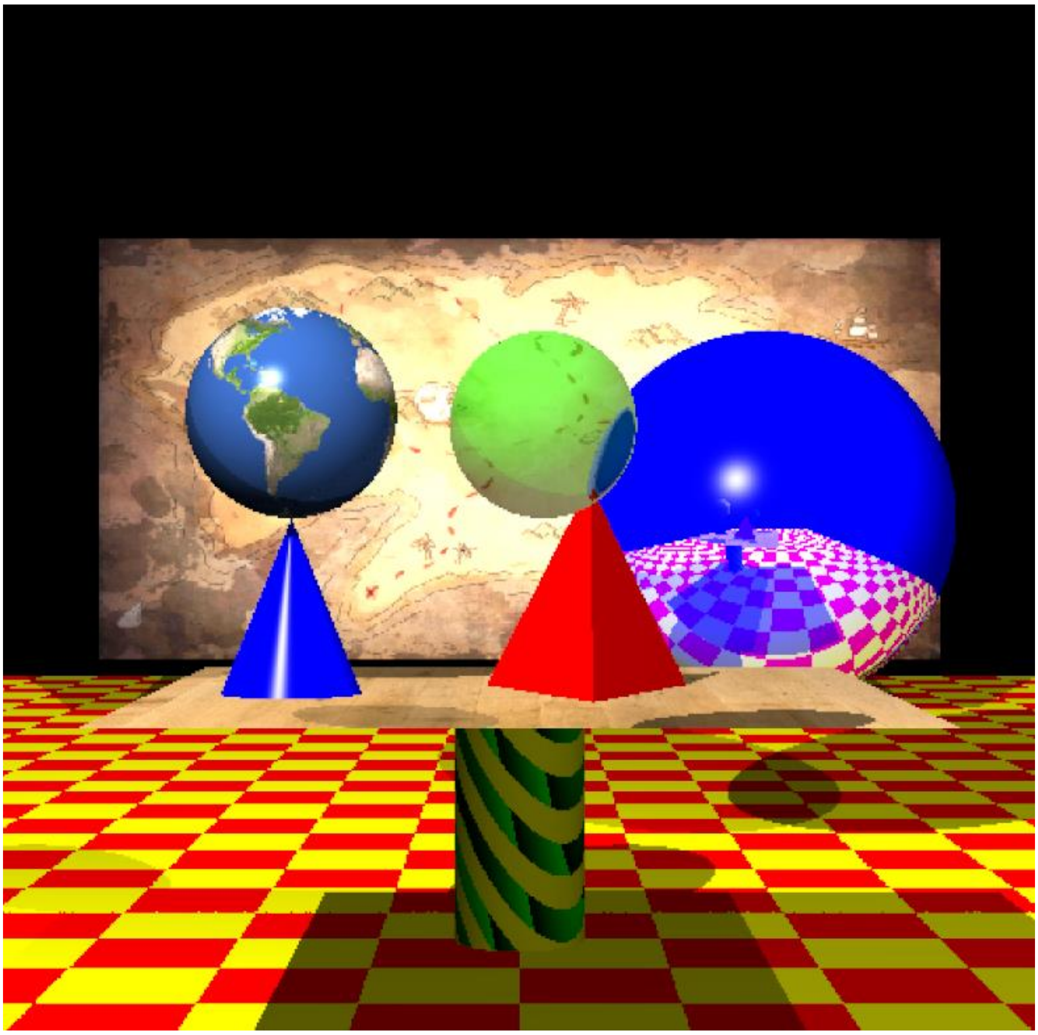


Fig 9 - Aliasing disabled

9. Success And Failures

The Scene was planned to have more objects at the start but decided to cut back and focus on the quality of the main parts of each object in the scene rather than quantity.
I was able to create all my objects in a *SceneShapes.cpp* which allowed me to easily go back and find parts of coded that needed fixing.
I was very happy with my:

- Shadows
- Earth sphere map and other textures
- Cylinder, Cone and planes
- Refracted and transparent object
- Pattern generation
- Simple effective Anti-Aliasing Algorithm
- Good scene design

$$F(r(t)) = c_4t^4 + c_3t^3 + c_2t^2 + c_1t + c_0 = 0$$

where

$$\begin{cases} c_4 = (d_x^2 + d_y^2 + d_z^2)^2 \\ c_3 = 4(d_x^2 + d_y^2 + d_z^2)(o_xd_x + o_yd_y + o_zd_z) \\ c_2 = 2(d_x^2 + d_y^2 + d_z^2)(o_x^2 + o_y^2 + o_z^2 - (r^2 + R^2)) + 4(o_xd_x + o_yd_y + o_zd_z)^2 + 4R^2d_y^2 \\ c_1 = 4(o_x^2 + o_y^2 + o_z^2 - (r^2 + R^2))(o_xd_x + o_yd_y + o_zd_z) + 8R^2o_yd_y \\ c_0 = (o_x^2 + o_y^2 + o_z^2 - (r^2 + R^2))^2 - 4R^2(r^2 - o_y^2) \end{cases}$$

https://marcin-chwedczuk.github.io/ray-tracing-torus

Challenges – tried:

- I tried to create a torus but found it was very complex and had a lot a issues with the intersect method. At the end, I had to move on but it had just to many roots for me to solve and code

Student ID: 27033004
Name: Jonathan Edwards

```
texture[0] = TextureBMP("../Models/Earth.bmp");     // http://www.world-maps.org/

texture[1] = TextureBMP("../Models/Table.bmp");
// https://freestocktextures.com/texture/floor-wood-oak,765.html

texture[2] = TextureBMP("../Models/Treasure_Map.bmp");
// http://www.aljanh.net/map-pirate-wallpapers/1436032319.html
```

Images reference