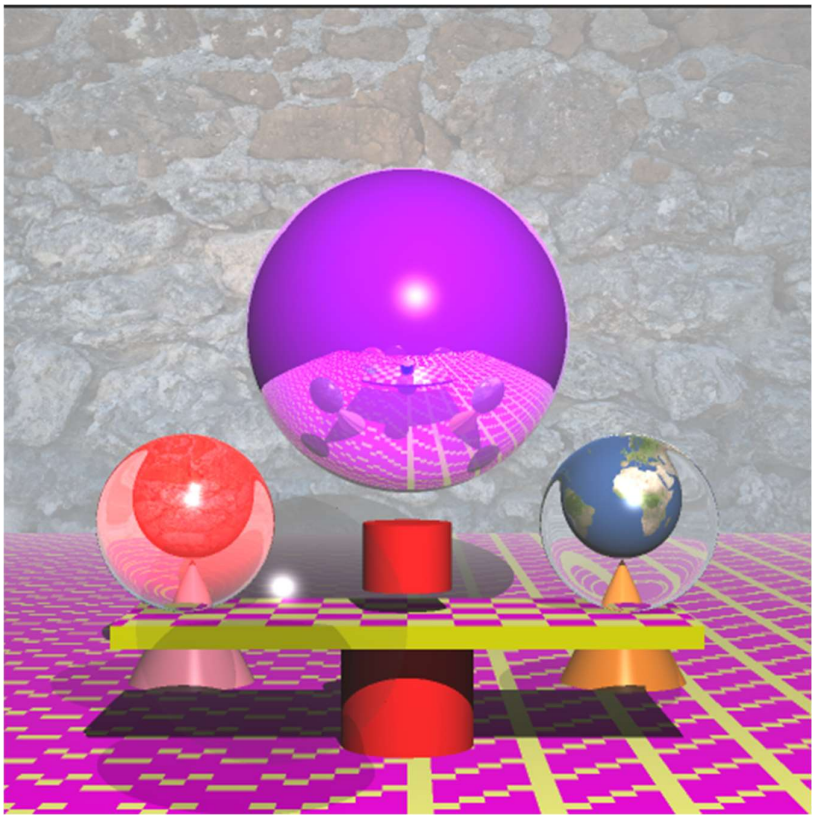
COSC 363 Assignment 2
Ray Tracing

Name: Kei Carden

Student Number : 75535091

Image of render with all features

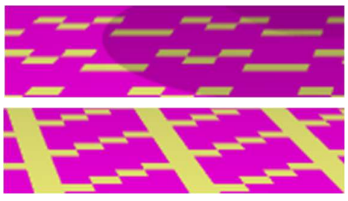


Run Program

To run the program, you can import it into QT creator with the use of the make file. Open QT creator go to file go open projects and files navigate to the make file in my document once it has been imported click the green play button to run.

Procedural Pattern

```
if (ray.index == 4) {
    //2 Patterns
    int checkSize = 1;
    int iz = (ray.hit.z) / checkSize + 50;
    int ix = (ray.hit.x) / checkSize + 50;
    int k = (ix > 45) ? (iz * ix + ix * ix) % 5 : (iz + ix * ix) % 5; //2 colours
    colour = (k != 0) ? glm::vec3(1, 0, 0.92) : glm::vec3(1, 1, 0.5);
    obj->setColour(colour);
}
```



The patterns implemted can be seen above. They have been made by altering the checkered pattern and I have made it so half the floor is one pantern and the other half is another pattern.

Cone

Two cones can be seen holding up two textured spheres these cones were implemented using the following equations. The equations were solved for t as to create an intersection equation. This was done by first substituting the ray equations in the cone equation then solving.

Cone Equation

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

```
float a = dir.x * dir.x + dir.z * dir.z - r * dir.y * dir.y;
float b = 2 * (dir.x * vdif.x + dir.z * vdif.z +  r * dir.y * vdif.y);
float c = vdif.x * vdif.x + vdif.z * vdif.z - vdif.y * vdif.y * r;
float delta = b * b - 4 * a * c;

if(delta < 0.001) return -1;

float t1 = (-b - sqrt(delta)) / (2 * a);
float t2 = (-b + sqrt(delta)) / (2 * a);
```

Ray Equation

$x = x_0 + d_x t; \quad y = y_0 + d_y t; \quad z = z_0 + d_z t;$

Normal Vector

The equation for a cones normal vector was implemented as shown below.

$$\boldsymbol{n} = (\sin\alpha\cos\theta, \sin\theta, \cos\alpha\cos\theta)$$

```
glm::vec3 n = p - center;
float theta = atan(radius / height);
float alpha = atan(n.x / n.z);

return glm::vec3(sin(alpha) * cos(theta), sin(theta), cos(alpha) * cos(theta));
```

Cylinder

The cylinder was crated by creating a class that implemented the following equations. As the intersection equation is a quadratic it must be solved so that we can find the intersection points on the cylinder. When constructing the cylinder a few checks are required to see when we have got to the hight we want so it does not infinitely go up.

Intersection Equation

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

```
float a = dir.x * dir.x + dir.z * dir.z;
float b = 2 * (dir.x * vdif.x + dir.z * vdif.z);
float c = vdif.x * vdif.x + vdif.z * vdif.z - radius * radius;
float delta = b * b - 4 * a * c;

if(delta < 0.001) return -1;

float t1 = (-b - sqrt(delta)) / (2 * a);
float t2 = (-b + sqrt(delta)) / (2 * a);
```

Ray Equation

$x = x_0 + d_x t; \quad y = y_0 + d_y t; \quad z = z_0 + d_z t;$

Normal Vector

The normal vector of a cylinder was based on the equation outlined in the lecture slides however as the cylinder has a cap the normal vector must change to one pointing strait up. A check is done to see if the point is at the hight of the cylinder if so, it changes the normal over.

```
glm::vec3 n = p - center;
return (fabs(n.y - height) < 0.001) ? glm::vec3(0, 1, 0) : glm::vec3(n.x/radius, 0, n.z/radius);
```

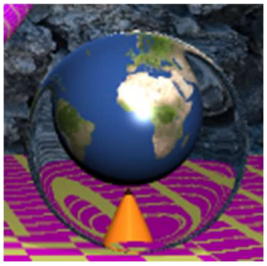
I initially had a bit of an issue with the cap of my cylinder were the top would suddenly have lots of black spots this was due to my height checking being incorrect.

Refraction

There are two spheres with refraction in the scene one with a both with a refractive index of 1.01 which gives a nice effect. The refraction is done by using glm refract with the initial ray direction and

the refractive index of the object a new ray is constructed with this new direction. The next point the ray hits will be inside of the sphere, so we ignore this hit and wait till we hit something else this colour value is then used multiplied by the opacity of the object. The shadow of this object is based on the opacity or refraction coefficient of the object.

```
if (obj->isRefractive() && step < MAX_STEPS) {
    float rc = obj->getRefractionCoeff();
    float eta = 1/obj->getRefractiveIndex();
    glm::vec3 n = obj->normal(ray.hit);
    glm::vec3 g = glm::refract(ray.dir, n, eta);
    Ray refrRay(ray.hit, g);
    refrRay.closestPt(sceneObjects);
    glm::vec3 m = obj->normal(refrRay.hit);
    glm::vec3 h = glm::refract(g, -m, 1.0f/eta);
    Ray refracRay(refrRay.hit, h);
    colour += rc * trace(refracRay, step + 1);
}
```



Anti-Aliasing

A basic method of anti-aliasing was used to help with the scene quality, the results can be seen in the images below. The method used was to create four more rays for each initial ray and then take the average of the four returned colours and setting this to the initial rays colour.

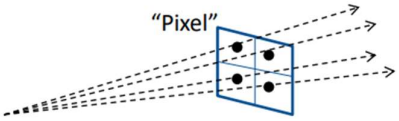


Illustration of how one ray be comes four rays.

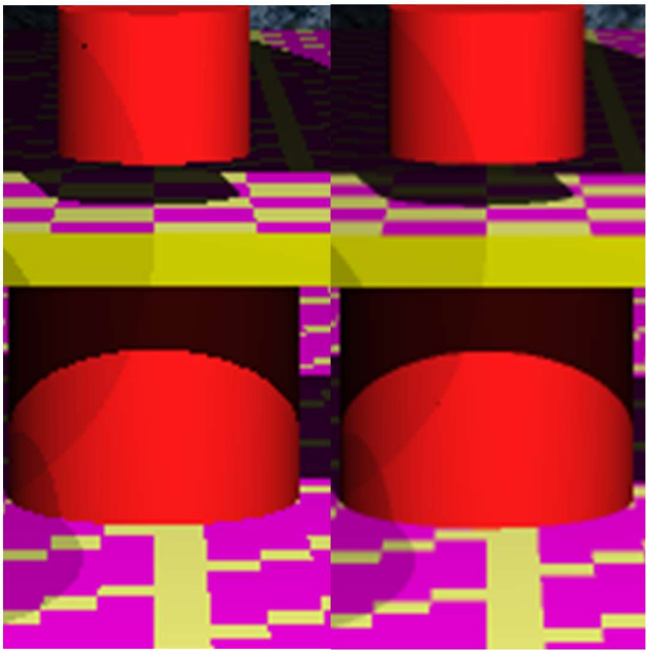
```
float y1 = yp + 0.25 * cellY;
float y2 = yp + 0.75 * cellY;

float x1 = xp + 0.25 * cellX;
float x2 = xp + 0.75 * cellX;

glm::vec3 dir0(x1, y1, -EDIST);
glm::vec3 dir1(x2, y1, -EDIST);
glm::vec3 dir2(x1, y2, -EDIST);
glm::vec3 dir3(x2, y2, -EDIST);

Ray ray = Ray(eye, dir0);
col = trace(ray, 1);
ray = Ray(eye, dir1);
col += trace(ray, 1);
ray = Ray(eye, dir2);
col += trace(ray, 1);
ray = Ray(eye, dir3);
col += trace(ray, 1);

col /= 4;
```



Sphere Texturing

Mapping a texture to a sphere was done with a UV mapping. To find the UV of a sphere the following equations were used.

```
glm::vec3 n = obj->normal(ray.hit);
float u = 0.5 + atan2(n.x, n.z) / (2 * M_PI);
float v = 0.5 + asin(n.y) / M_PI;
colour = earthTexture.getColorAt(u, v);
obj->setColour(colour);
```

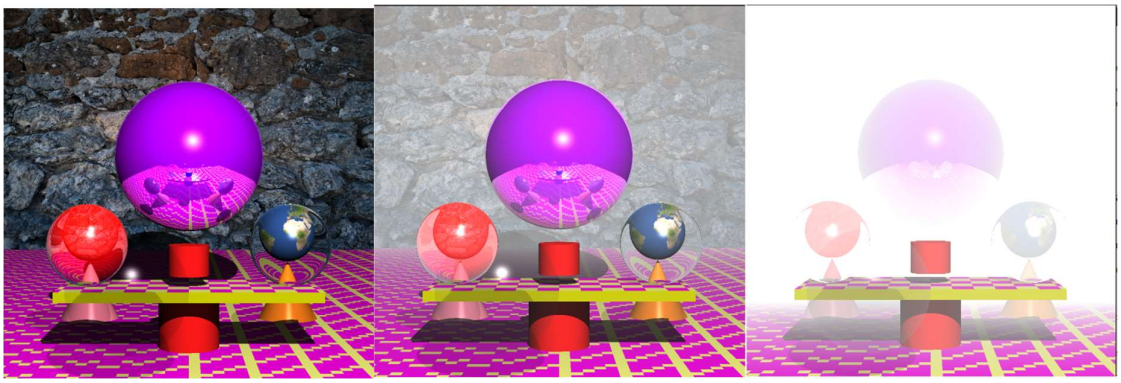
$$u = 0.5 + \frac{arctan2(d_x,\ d_z)}{2\pi}$$

$$v = 0.5 + \frac{arcsin(d_y)}{\pi}$$

These two equations were used as coordinates on an image so the colour value could be used for the point that the ray hit on the sphere. The equations were found on Wikipedia.

Fog

Fog has been implemented quite simply by using the equation (1 - t) * colour + * fogColour. As the t value gets larger (this value gets larger as the ray hits in the z direction get further away) the fog effect gets more intense as can be seen below the starting and ending point can be alters to create a scene with more or less fog. Equation created from notes.



No Fog, Light Fog and Lots of Fog

```
float t = (ray.hit.z + MIN_FOG)/(MIN_FOG - MAX_FOG);
colour = (1 - t) * colour + t * glm::vec3(1, 1, 1);
```

Render Time

With all features the time to render is about 15s.

With no anti-aliasing and fog the render time is about 6s.