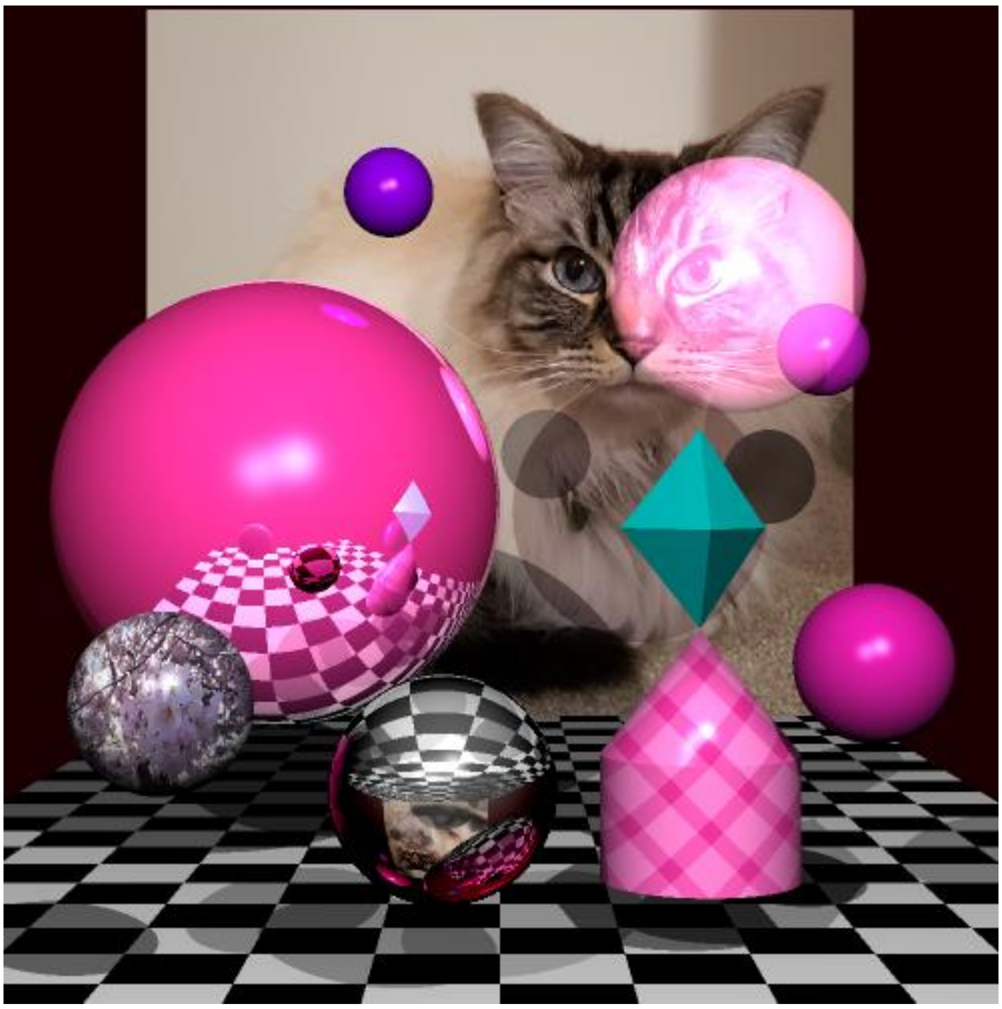
COSC363 Assignment 2 Report

Jkw81 – 91390060

My project builds off of the code in labs 7 and 8. In the scene I created, there is a reflective sphere, a refractive sphere, a sphere textured with flowers, a hollow sphere, 3 extra spheres, a cylinder with a cone on top, and a tetrahedron (created using 6 planes) on top of the cone. The cylinder and cone both have procedurally generated patterns resembling plaid. The objects are placed above a checkered floor, with an image of my cat textured behind them. The entire scene can be seen below.



To build this scene:
1. Download and unzip folder
2. Open CMakeLists.txt in QtCreator and configure project.
3. Make sure that the working directory is the folder with everything in it, i.e., the textures.
4. Press the green triangle at the bottom left corner of your screen.
5. Enjoy!

Extra Features

Cone: The cone's point of intersection and surface normal are calculated to generate the cone. The equations for the intersection and normal vectors is shown below.

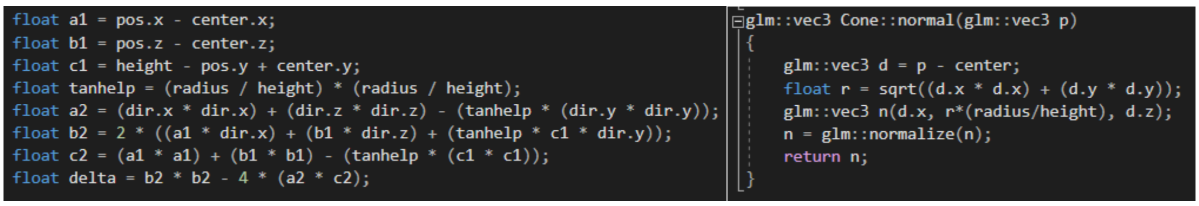
Cone equation: $(x-x_c)^2+(z-z_c)^2=\left(\frac{R}{h}\right)^2(h-y+y_c)^2$

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

Normal vector: $\boldsymbol{n} = (\sin\alpha\cos\theta,\ \sin\theta,\ \cos\alpha\cos\theta)$ where $\alpha = \tan^{-1}\left(\frac{x-x_c}{z-z_c}\right)$

Intersection equation: Substitute ray equation into cone equation and solve for t.
Important: $\tan(\theta) = R/h$ (θ = half cone angle)

The implementation of the equations is shown below. Intersect on left. pos & dir are the position and direction of the ray.



```
float a1 = pos.x - center.x;
float b1 = pos.z - center.z;
float c1 = height - pos.y + center.y;
float tanhelp = (radius / height) * (radius / height);
float a2 = (dir.x * dir.x) + (dir.z * dir.z) - (tanhelp * (dir.y * dir.y));
float b2 = 2 * ((a1 * dir.x) + (b1 * dir.z) + (tanhelp * c1 * dir.y));
float c2 = (a1 * a1) + (b1 * b1) - (tanhelp * (c1 * c1));
float delta = b2 * b2 - 4 * (a2 * c2);
```

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

Cylinder: The cylinder's point of intersection and surface normal are calculated to generate the cone. The equations for the intersection and normal vectors is shown below.

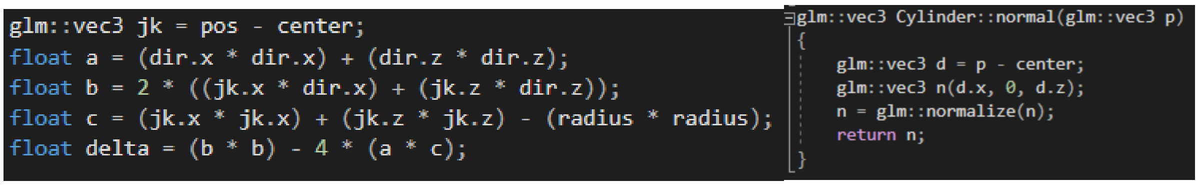
Cylinder equation: $(x - x_c)^2 + (z - z_c)^2 = R^2 \quad 0 \le (y - y_c) \le h$

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

Normal vector: (un-normalized) $\boldsymbol{n} = (x - x_c,\ 0,\ z - z_c)$
(normalized) $\boldsymbol{n} = (\ (x - x_c)/R,\ \ 0,\ \ (z - z_c)/R\ )$

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

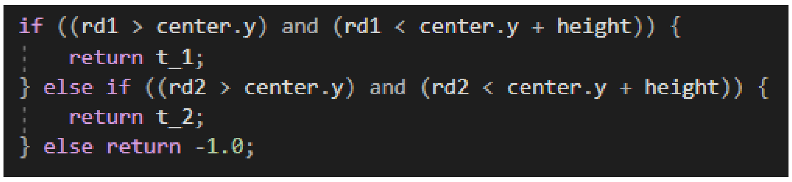
The implementation of the equations is shown below. Intersect on left. pos & dir are the position and direction of the ray.



```
glm::vec3 jk = pos - center;
float a = (dir.x * dir.x) + (dir.z * dir.z);
float b = 2 * ((jk.x * dir.x) + (jk.z * dir.z));
float c = (jk.x * jk.x) + (jk.z * jk.z) - (radius * radius);
float delta = (b * b) - 4 * (a * c);
```

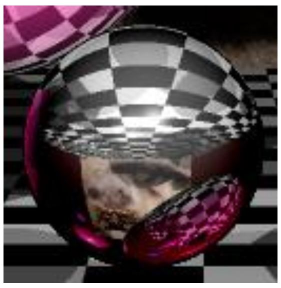
```
glm::vec3 Cylinder::normal(glm::vec3 p)
{
    glm::vec3 d = p - center;
    glm::vec3 n(d.x, 0, d.z);
    n = glm::normalize(n);
    return n;
}
```

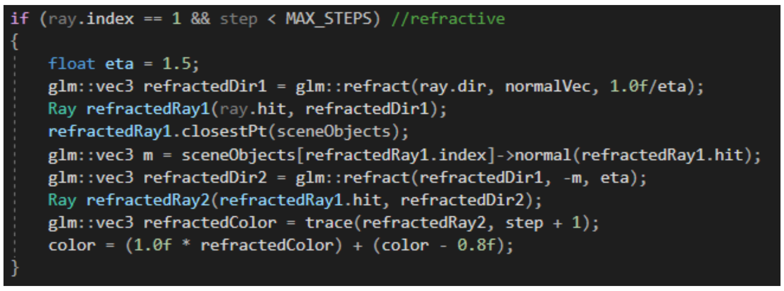
The following code limits both the heights of the cone and cylinder, and also generates the front and back sides.



```
if ((rd1 > center.y) and (rd1 < center.y + height)) {
    return t_1;
} else if ((rd2 > center.y) and (rd2 < center.y + height)) {
    return t_2;
} else return -1.0;
```

Refractive Sphere: This sphere is represented as a glass sphere with an air index of 1 and a glass index of 1.5 causing light to refract when passing through the sphere. Because of this refraction, the image that appears in the sphere is upside down. The implementation is shown below, where the second ray exits the sphere at a different angle.



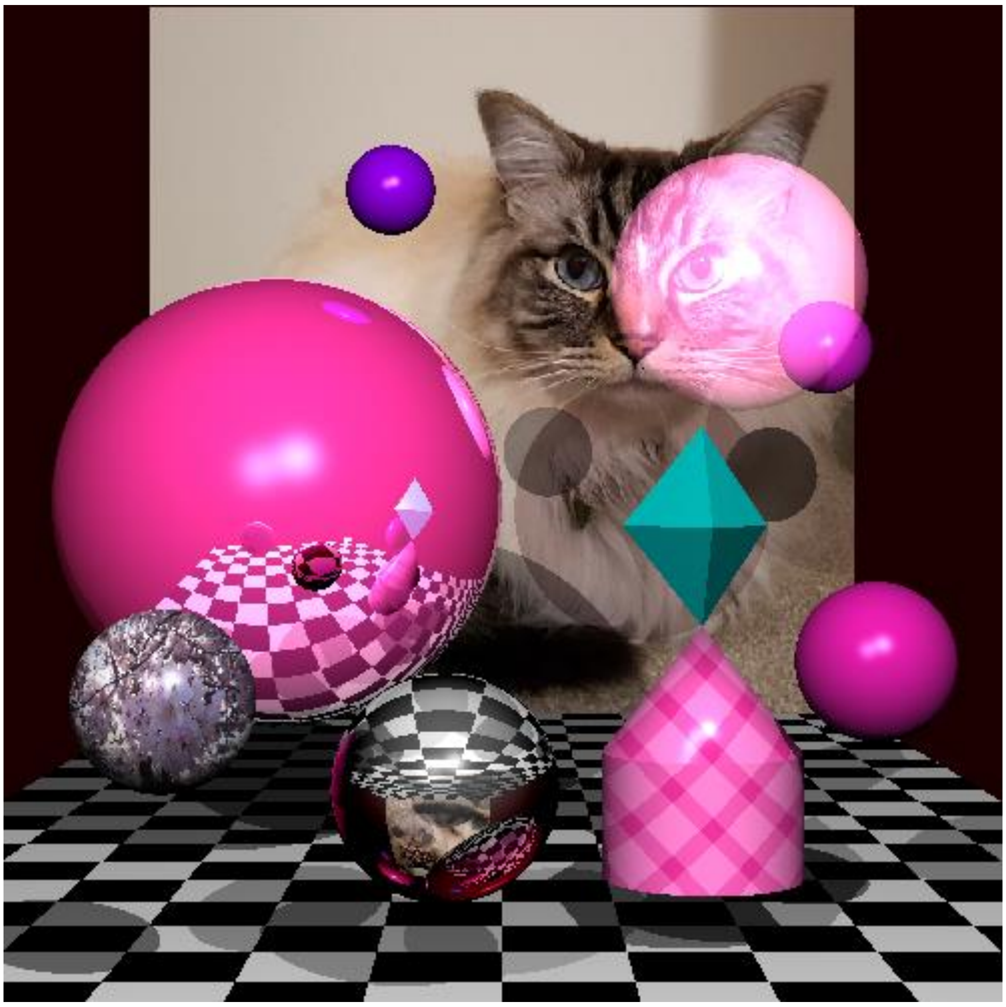


```
if (ray.index == 1 && step < MAX_STEPS) //refractive
{
    float eta = 1.5;
    glm::vec3 refractedDir1 = glm::refract(ray.dir, normalVec, 1.0f/eta);
    Ray refractedRay1(ray.hit, refractedDir1);
    refractedRay1.closestPt(sceneObjects);
    glm::vec3 m = sceneObjects[refractedRay1.index]->normal(refractedRay1.hit);
    glm::vec3 refractedDir2 = glm::refract(refractedDir1, -m, eta);
    Ray refractedRay2(refractedRay1.hit, refractedDir2);
    glm::vec3 refractedColor = trace(refractedRay2, step + 1);
    color = (1.0f * refractedColor) + (color - 0.8f);
}
```

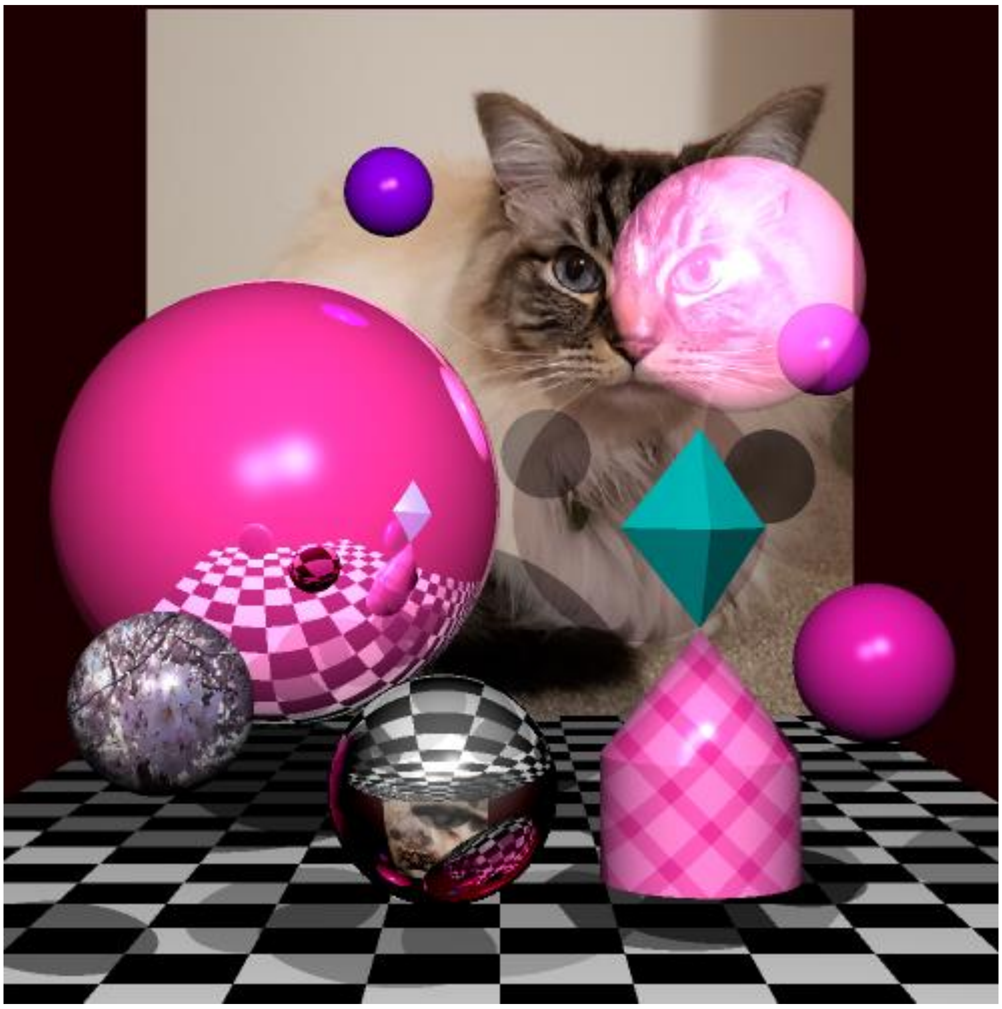
Multiple Light Sources: Two light sources were included in the scene, generating two sets of shadows and specular highlights where applicable. The shadows for the refractive and transparent spheres are lighter than the other objects. The transparent sphere gives a slight pink tinged shadow.

Anti-aliasing: Anti-aliasing is used for generating smoother images, so instead of the objects having jagged edges, the edges will appear smoother. This is done by creating 4 rays for each pixel, instead of one, and averaging the result between them. The effect of anti-aliasing can be seen below.

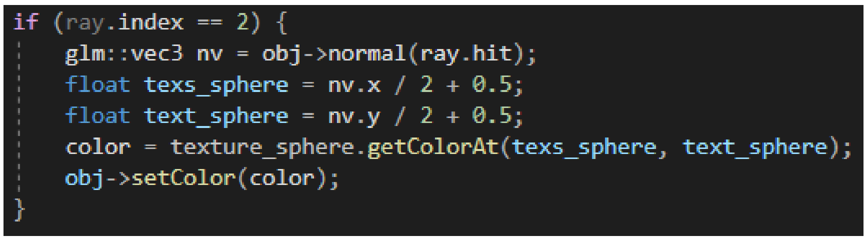
<u>Without Anti-aliasing</u> – approximate time taken to generate: **3 seconds**



With Anti-aliasing – approximate time taken to generate: **9-12 seconds**

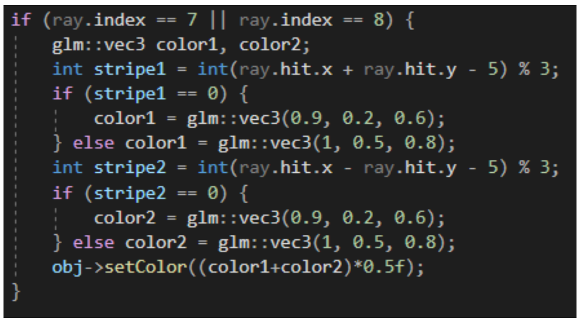


Textured Sphere: This sphere was textured with an image I had taken. It was textured using an equation I found online, with the implementation shown below.



```
if (ray.index == 2) {
    glm::vec3 nv = obj->normal(ray.hit);
    float texs_sphere = nv.x / 2 + 0.5;
    float text_sphere = nv.y / 2 + 0.5;
    color = texture_sphere.getColorAt(texs_sphere, text_sphere);
    obj->setColor(color);
}
```

Procedural Pattern on Cylinder and Cone: The implementation of the procedural pattern for the cylinder and cone is shown below. stripe1 and stripe2 are the main equations.



```
if (ray.index == 7 || ray.index == 8) {
    glm::vec3 color1, color2;
    int stripe1 = int(ray.hit.x + ray.hit.y - 5) % 3;
    if (stripe1 == 0) {
        color1 = glm::vec3(0.9, 0.2, 0.6);
    } else color1 = glm::vec3(1, 0.5, 0.8);
    int stripe2 = int(ray.hit.x - ray.hit.y - 5) % 3;
    if (stripe2 == 0) {
        color2 = glm::vec3(0.9, 0.2, 0.6);
    } else color2 = glm::vec3(1, 0.5, 0.8);
    obj->setColor((color1+color2)*0.5f);
}
```

References

COSC363 Labs and Lectures

https://mvps.org/directx/articles/spheremap.htm