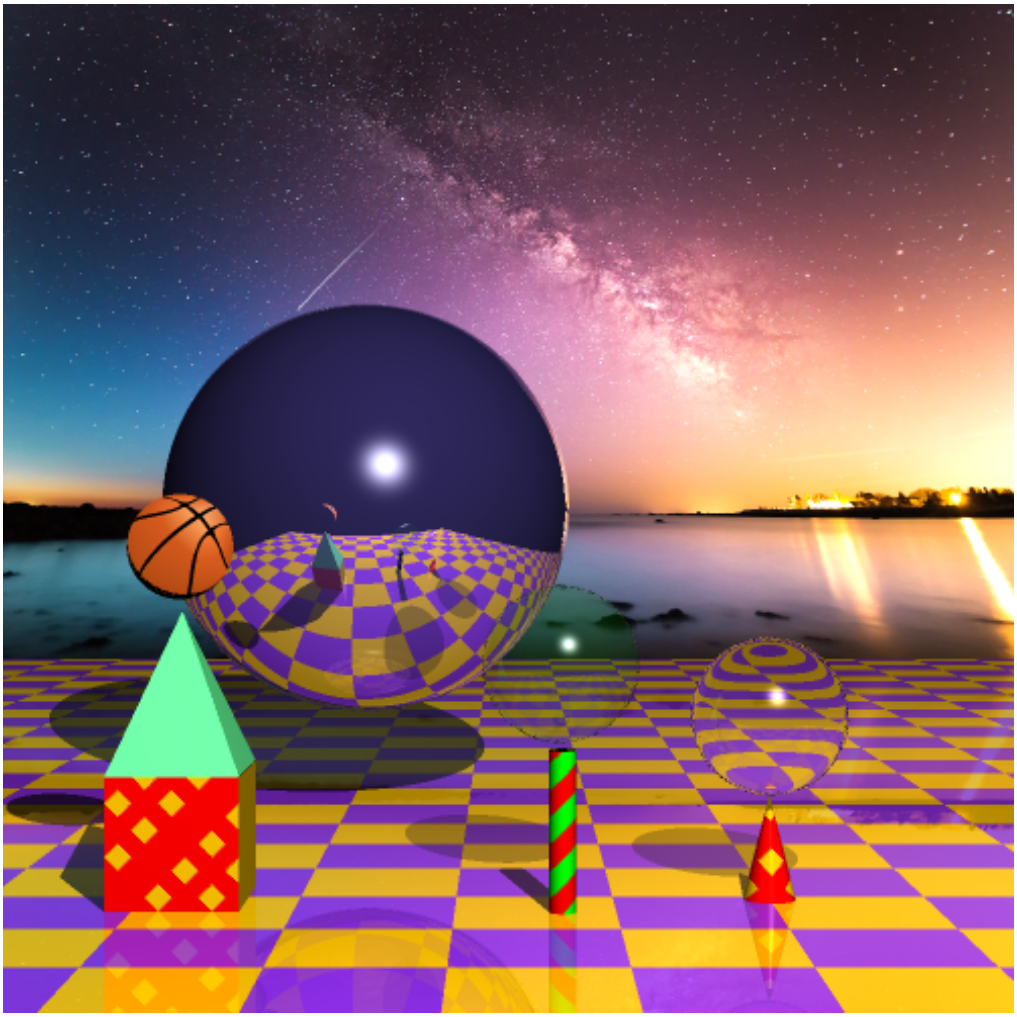
Rchi Lugtu: 89933448

COSC363 Assignment 2
Ray Tracing



The estimate time taken by my program to generate the output on my virtual machine is approximately 33 seconds (with anti-aliasing), on NUMDIV = 500. The images used in texturing my objects are from unsplash.com, and the program is built on from lab8.

Success and Failures: One of the things I struggled with is ensuring my transparent and refracted sphere is rendered correctly. At first when I created those spheres, the spheres had white colors inside, thus making the spheres not transparent. It turns out, I had to make sure the sphere color was set to black. After doing this, I got another problem where, my spheres would have black dots around them, and they are still evident even in my final version of my assignment. Another one was the time it takes for my ray tracer program to load; it took 33 seconds to load with anti-aliasing, and I'm guessing it's because of the high-resolution images that I used in my program. But all in all, I manage to implement all the minimum features as well as the extra features required to get maximum marks.

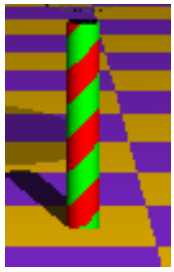
Rchi Lugtu: 89933448

Plane: There is a plane in the background, with a galaxy image texture on it, and the box, and pyramid were created through combinations of planes.

Cylinder: A cylinder has been created by applying the ray and intersection equations provided in the lecture notes, and it has been textured with red and green stripes. The intersection equation is used to find the closest point of intersection:

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

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



Code aspect:

```
float xDiff = p0.x - center.x;
float zDiff = p0.z - center.z;

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

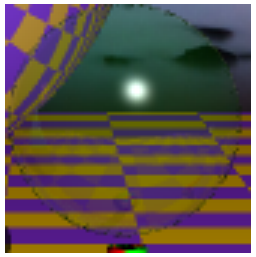
Cone: A cone was created using the same ray equation of the cylinder, but has more equations, and is also textured (explained further down). The equations used is shown below:

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

Code aspect:

```
float lower, higher;
float xDiff = p0.x - center.x;
float zDiff = p0.z - center.z;
float yDiff = height - p0.y + center.y;
float radHeight = (radius / height);

float a = (dir.x * dir.x) + (dir.z * dir.z) - (radHeight * radHeight) * (dir.y * dir.y);
float b = 2 * ((xDiff * dir.x + zDiff * dir.z) + (radHeight *radHeight) * (yDiff * dir.y));
float c = (xDiff * xDiff) + (zDiff * zDiff) - (radHeight * radHeight) * (yDiff * yDiff);
float delta = (b * b) - (4 * a * c);
```

Transparent Sphere: The ray that goes in the sphere, has the same direction as the ray that goes out of the sphere. Reflectivity was also applied for better rendering. The transparency and reflectivity coefficient of the sphere was set to 0.8, and 0.1 respectively. The colour is (0, 0.1, 0) for visual reasons. The implementations for the transparency effect is shown below:

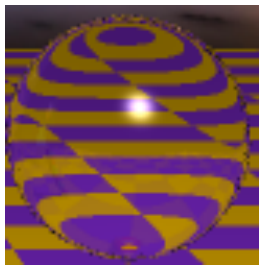


```
// Transparent
if (obj->isTransparent() && step < MAX_STEPS)
{
    Ray rayTHROUGH(ray.hit, ray.dir);
    rayTHROUGH.closestPt(sceneObjects);

    Ray rayOUTsphere(rayTHROUGH.hit, ray.dir);
    rayOUTsphere.closestPt(sceneObjects);

    glm::vec3 translatedColor = trace(rayOUTsphere, step + 1);
    color = color + (obj->getTransparencyCoeff() * translatedColor);
}
```

Rchi Lugtu: 89933448

Refracted Sphere: The ray refracts as it enters in the sphere, due to the ETA value (ETA = 1/1.005, for aesthetic effects, the higher the ETA value is the less refractivity it has), and the ray refracts back, as it exits out of the sphere. The implementation of this is shown below:

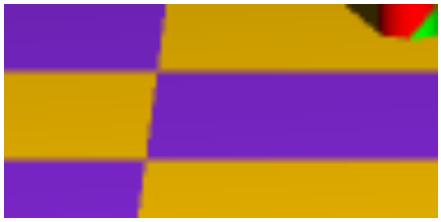
```
if (obj->isRefractive() && step < MAX_STEPS)
{
    float eta = 1/1.005;
    glm::vec3 n = obj->normal(ray.hit);
    glm::vec3 g = glm::refract(ray.dir, n, eta);
    Ray refractedRay(ray.hit, g); // Refracted ray inside the sph
    refractedRay.closestPt(sceneObjects);

    glm::vec3 m = obj->normal(refractedRay.hit);
    glm::vec3 h = glm::refract(g, -m, 1.0f/eta);
    Ray refractedRayOut(refractedRay.hit, h); // Refracted ray an
    refractedRayOut.closestPt(sceneObjects);
    glm::vec3 refractedColor = trace(refractedRayOut, step + 1);
    color = color + (obj->getRefractionCoeff() * refractedColor);
}
```



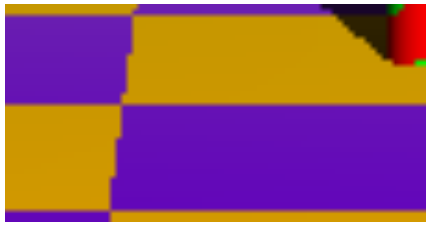
Lighter shadow for transparent and refractive spheres: Lighter shadows are obtained by getting the shadow ray index of the object, and multiplying the colour value by 0.5, to obtain a lighter shade. Implementation of this are shown below:

```
if (shadowRay.index == 1 || shadowRay.index == 3) {
    color = 0.5f * obj->getColor();
}
```

Anti-Aliasing: Super sampling was used, and it works by subdividing the pixel into four segments and is averaged, to obtain colour value of the pixel. After anti-aliasing is implemented, the edges of each object are smoother. It is most obvious in the floor pattern, where on the right hand side, the jaggedness of the lines of the floor are visible, whereas on the left, it's smoother.



With Anti-aliasing



Without Anti-aliasing

```
glm::vec3 anti_aliasing(float red, float green, float blue, float cellX, float cellY, float xp, float yp, glm::vec3 eye) {
    for (int j = 0; j < 2; j++) {
        for (int k = 0; k < 2; k++) {
            glm::vec3 dir((xp + (j * 0.5 + 0.25) * cellX), (yp+(k * 0.5 + 0.25)*cellY), -EDIST);

            Ray primaryRay = Ray(eye, dir);

            glm::vec3 colour = trace(primaryRay, 1); // Trace the primary ray and get the colour value
            red += colour.r;
            green += colour.g;
            blue += colour.b;
        }
    }
    return glm::vec3(red, green, blue);
}
```

Inside the for loop of the display() function

```
// Anti-aliasing
red = 0; green = 0; blue = 0;
col = anti_aliasing(red, green, blue, cellX, cellY, xp, yp, eye);
red = col.r/4; green = col.g/4; blue = col.b/4;
glColor3f(red, green, blue);
```

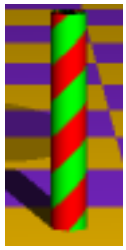
Rchi Lugtu: 89933448

Non-planar textured object: A sphere is textured with an image of a basketball. The implementation of this is shown below, where the index of the sphere is 2.

```
if(ray.index == 2){
    glm::vec3 normalVec = obj->normal(ray.hit);
    float patternX = asin(normalVec.x) / M_PI + 0.5;
    float patternY = asin(normalVec.y) / M_PI + 0.5;
    color += ball.getColorAt(patternX, patternY);
}
```

Procedural Pattern: To obtain 2 different line directions I used “ray.hit.x + ray.hit.y” and “ray.hit.x – ray.hit.y”. I coloured both lines red, and the remaining space that forms a square has been coloured yellow. The pattern that this produces is shown on the image of the box down below. The pattern displayed on the cylinder uses the same code, but the difference is one line is coloured red, while the other line and the square shape are both coloured green, thus producing the pattern on the cylinder.

```
if (ray.index == 17)
{
    if ((int(ray.hit.x - ray.hit.y)) % 2 == 0) {
        obj->setColor(glm::vec3(1, 0, 0));
    } else if ((int(ray.hit.x + ray.hit.y)) % 2 == 0) {
        obj->setColor(glm::vec3(0, 1, 0));
    } else {
        obj->setColor(glm::vec3(0, 1, 0));
    }
}
```



Fog: To implement the fog, the equation provided in the lecture notes was used, and placed inside the tracer() function. I made sure to set the background colour to white, as well as removed the background plane (galaxy textured), to improve rendering quality. The images with and without the fog is shown below:

```
// Fog - Make sure you set background colour to WHITE (start of trace function)!!
float t = (ray.hit.z - -80.0) / (-180.0 - -80.0);
color = ((1 - t) * color) + (t * glm::vec3(1.0, 1.0, 1.0));
```