ID: 45239028
Name: Yu Duan

COSC363 Assignment 2 Ray Tracer Report

ID: 45239028
Name: Yu Duan

Build process

The application was developed and built using Microsoft Visual Studio 2019 on Windows with the provided zip files on learn following the "Setting up Visual Studio with OpenGL on Windows" pdf. The main program is the RayTracer.cpp. The program takes approximately 50 seconds to generate the output on my laptop which has the graphics card GTX 1050 Ti and the CPU is an Intel Core i7-7700HQ.

Features

Cylinder with a top cap

The cylinder was implemented by extending the SceneObject class to create the Cylinder class. The intersect function was implemented using the intersection equation and ray equation given from lecture note 8 page 38.

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

Intersection equation:

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

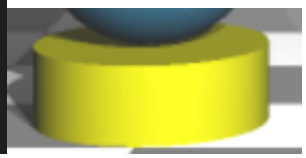
The following shows the quadratic equation being used and a cylinder that was created.

```
float xdif = p0.x - center.x;
float zdif = p0.z - center.z;

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

if (fabs(delta) < 0.001) return -1.0;
if (delta < 0.0) return -1.0;

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



To make sure that the cylinder isn't of infinite height, more checks were required to see if the y value of the point at the closest t value is above or below the cylinder, if above or below we require to check if the y value of the second t value is within the cylinder, if within the cylinder then the second t value if returned. I have implemented cap on the top of the cylinder, so if the cylinder has a cap, then we check if the y value of the first point is above the cylinder, and then check the y value of the second point is within the cylinder. If those conditions are met, we calculate a new t value that would be at the height of the cylinder by equating the height of the cylinder with the ray equation such that $y_c + h = y_0 + d_y t$ and solve for t.

The equation of the surface normal vector of the cylinder was provided in the lecture note, but the surface normal vector of the cap of the cylinder needs to be different as it is a flat surface, this normal would just be (0, 1, 0).

Cone

The cone was implemented by extending the SceneObject class to create the Cone class. The intersect function was implemented by finding the intersection equation by substituting ray equation into the cone equation which were both given in the lecture notes.

ID: 45239028
Name: Yu Duan

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

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

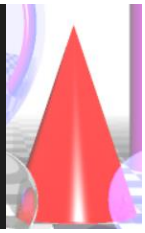
The following shows the solved quadratic equation being used and a cone that was created.

```
float rhRatio = radius / height;
float a = dir.x * dir.x + dir.z * dir.z - rhRatio * rhRatio * dir.y * dir.y;
float b = 2 * vdif.x * dir.x + 2 * vdif.z * dir.z + rhRatio * rhRatio * 2 * height * dir.y - rhRatio * rhRatio *  2 * vdif.y * dir.y;
float c = vdif.x * vdif.x + vdif.z * vdif.z - rhRatio * rhRatio * (height - vdif.y) * (height - vdif.y);

float delta = b * b - 4 * a * c;

if (fabs(delta) < 0.001) return -1.0;
if (delta < 0.0) return -1.0;

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



The equation of the surface normal vector of the cone was provided in the lecture notes.

Refraction of light through a sphere

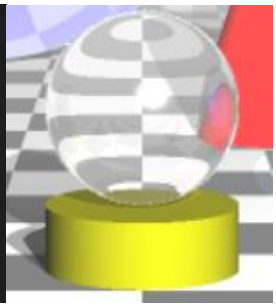
The implementation was mostly given in lecture note 8 page 21. The normal vector and refracted ray are calculated twice, once for when the ray hits the sphere and once when the ray leaves the sphere. The implementation and the refracted sphere with an index of refraction of 1.01 are shown below.

```
float eta = 1 / obj->getRefractiveIndex();
float refractionCoeff = obj->getRefractionCoeff();

glm::vec3 normalVec = obj->normal(ray.hit);
glm::vec3 refractedDir = glm::refract(ray.dir, normalVec, eta);
Ray refractedRayIn(ray.hit, refractedDir);
refractedRayIn.closestPt(sceneObjects);

glm::vec3 m = obj->normal(refractedRayIn.hit);
glm::vec3 h = glm::refract(refractedDir, -m, 1.0f / eta);

Ray refractedRayOut(refractedRayIn.hit, h);
glm::vec3 refractiveColor = trace(refractedRayOut, step + 1);
color = color + (refractionCoeff * refractiveColor);
```



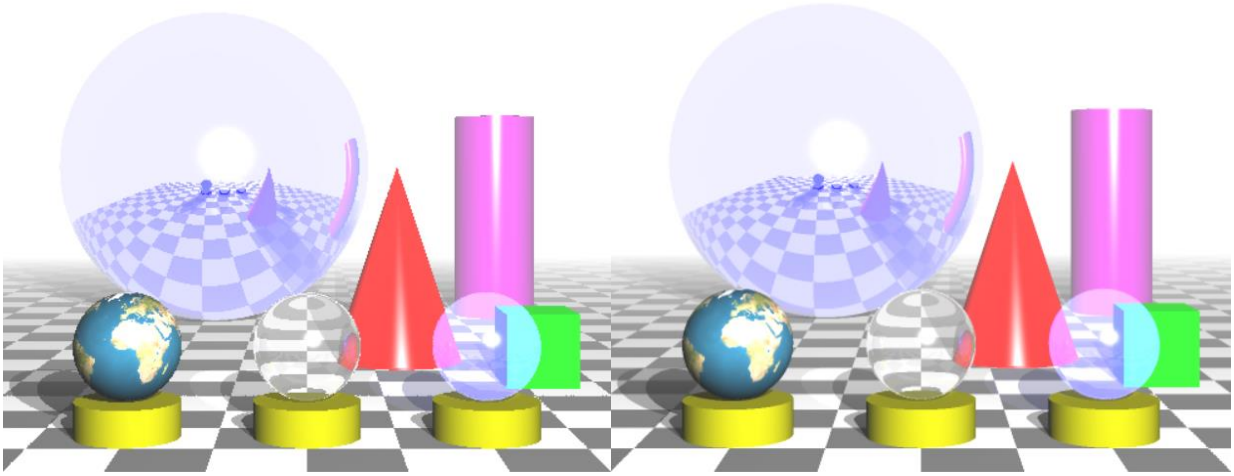
Multiple light sources including multiple shadows generated by them

I have implemented multiple light sources that will produce multiple shadows, the implementation was based on what was given in the ray tracing assignment lecture notes. I have modified the lighting function of SceneObject to only return the specular term and the diffuse term. My implementation will initially set the color as the specular term and diffuse term of both light sources, then if a shadow ray intersects an object, I would remove the specular term and diffuse term from the appropriate light source. Then I would add the ambient term after all the removal of specular term and diffuse term were completed. The image below shows multiple shadows.

ID: 45239028
Name: Yu Duan

Anti-aliasing (Supersampling)

Supersampling is used to reduce the distortion artefacts such as jaggedness along edges of polygons and shadows. Supersampling was implemented by further splitting up the cell into four smaller cells, where we then get the color of each of those cells and then get the average of those colors which will represent the color of the pixel. Below is a comparison between no anti-aliasing (left) and with supersampling anti-aliasing (right).



You can see that the edges on polygons and shadows are less smooth on the left image, while on the right you can see that it has become smoother and more pleasing to the eye.

A non-planar object textured using an image

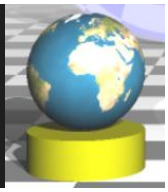
Texturing of a sphere was done using UV mapping where two equations from Wikipedia were used to find the UV coordinates on a sphere in the range of [0, 1].

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

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

However my implementation required the equation for the V coordinate needed to be changed to $v = 0.5 + \frac{\arcsin(d_y)}{\pi}$, as beforehand it was texturing the texture upside down. Below displays the implementation and the textured sphere as the Earth.

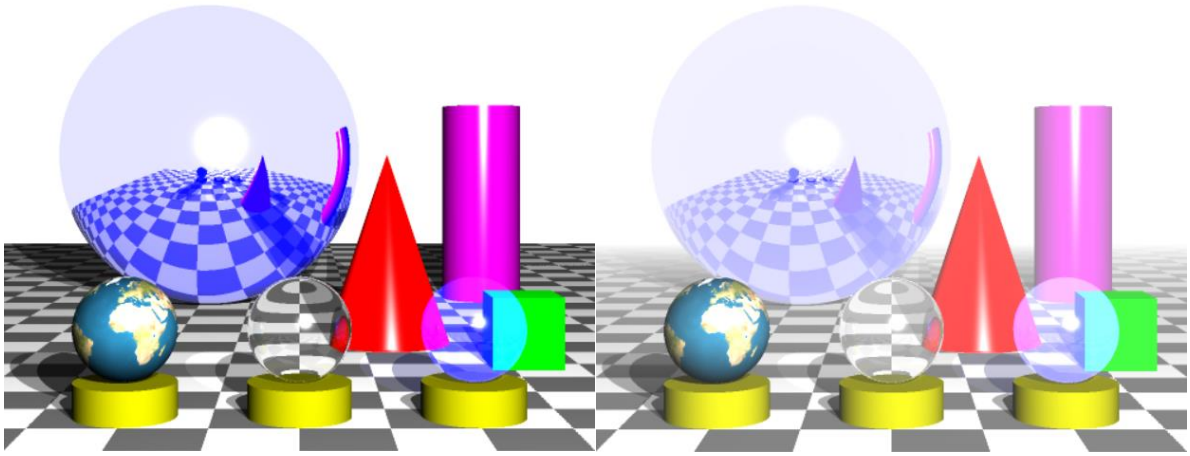
```
glm::vec3 d = glm::normalize(ray.hit - ((Sphere*)obj)->getCenter());
float u = 0.5 + (atan2(d.x, d.z) / (2 * M_PI));
float v = 0.5 + (asin(d.y) / M_PI);
color = texture.getColorAt(u, v);
obj->setColor(color);
```



ID: 45239028
Name: Yu Duan

Fog

The fog was implemented using the method mentioned in the lecture. I have set the background to white and defined a fog range of [-40, -200]. An interpolation parameter t which varies between 0 and 1 is calculate with $t = (ray.hit.z - z1 \ / \ (z2 - z1))$. I have modified the color value just before it is returned as $(1 - t) * color + t.white$ which will give us the foggy effect as the t value increases the further the object intersected by the ray is. Below is a comparison between no fog (right) and with fog (left).



Successes and failures

Successes

- Implementing the refractive sphere was cool, once it was implemented, I was mind blown on how it refracts the objects around it. I tried different index of refractions just to see what it could do.
- Implementing the anti-aliasing was satisfying, it was straight forward and once implemented the ray tracing looked much better.

Failures

- One challenge that I had difficulty with was correctly implement the cone. Initially when I implemented the cone, the backside of the cone would not render correctly and would not have shadows on the back. I checked my normal function multiple times and couldn't find the solution, I eventually found out I had implemented the intersection function wrong where I did not compare the two t values to find which one was larger, after I rewrote the function, the cone was finally rendering correctly on the backside.
- Another challenge was getting the cap of the cone to render correctly, it was not rendering correctly because I did not realize that the surface normal should be (0, 1, 0), but I eventually figured it out.

References

Earth texture file: http://planetpixelemporium.com/earth.html

Texture mapping for sphere: https://en.wikipedia.org/wiki/UV_mapping