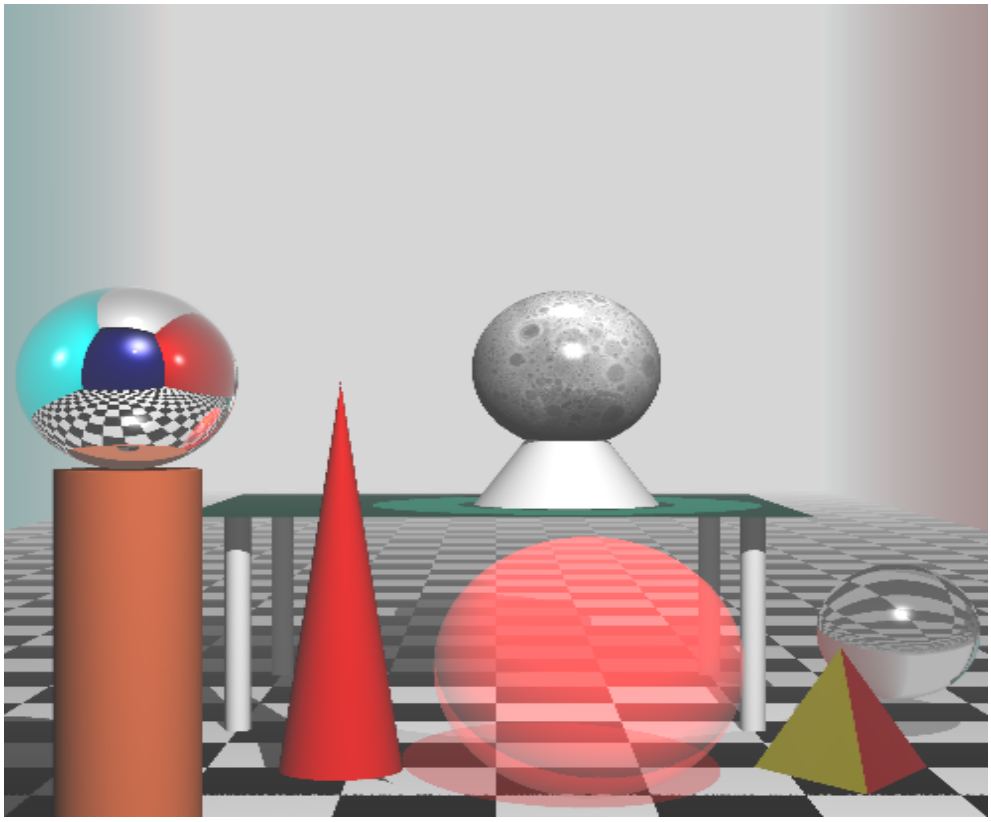
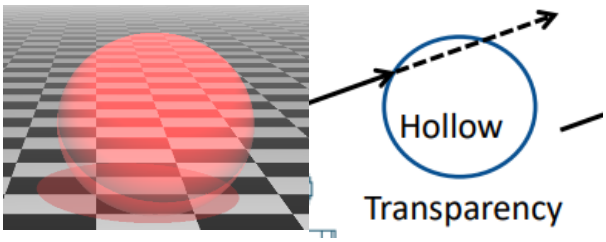
COSC363 Assignment 2

Katsu Lee (48792793)



The Estimated time taken to generate the scene is 54.4 seconds. In my ray tracer I have successfully implemented the minimum requirements and seven of the extensions.
The scene has good spatial arrangement because you can see every object clearly as well as the differing colors and textures.
The extensions are:
textured sphere, a cone, a cylinder, a refractive sphere, a spotlight, anti-aliasing and a fog.
Some other things in my ray tracer are a checkered floor (black and white), a reflective sphere on top of a cylinder, and several colored walls. There is also a table made of a plane and four cylinders. The transparent and refractive objects also have lighter shadows than the other objects which can be seen in the image above. One major failure is my spotlight as it does not look good/isn't very noticeable in the scene.

Transparent Sphere:



I made a transparent sphere with a lighter shadow that has the same color as the object. This was done by making a ray through the object in the same direction that the ray hits the object. Then another ray goes in the same direction, after the inner ray hits the second intersection (Lecture 8 slide 15). For the transparent shadow (and refractive) I made them lighter by adding a condition then scaling the color and the shadow color by a smaller value.

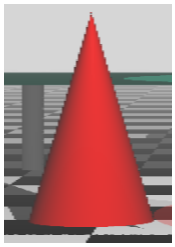
Code for transparent objects

```
// For transparency
if (obj->isTransparent() && step < MAX_STEPS){
    float transRho = obj->getTransparencyCoeff();
    Ray transparentRay1 = Ray(ray.hit, ray.dir);
    transparentRay1.closestPt(sceneObjects);
    Ray transparentRay2 = Ray(transparentRay1.hit, transparentRay1.dir);
    glm::vec3 transparentColor = trace(transparentRay2, step + 1);
    color = color + (transRho * transparentColor);
}
```

Code for transparent and refractive objects shadow

```
glm::vec3 surface color(0);
surface color = obj->lighting(lightPos, -ray.dir, ray.hit);
glm::vec3 lightVec = lightPos - ray.hit;
float lightDist = glm::length(lightVec);

Ray shadowRay(ray.hit, lightVec); //direction of shadow ray
shadowRay.closestPt(sceneObjects);

if (shadowRay.index > -1 && shadowRay.dist < lightDist) {
    //~ // ight from the spotlight is obstructed.
    SceneObject* obj shadow = sceneObjects[shadowRay.index];
    if (obj shadow->isTransparent() && step < MAX STEPS || obj shadow->isRefractive() && step < MAX STEPS){
        surface color = surface color * 0.6f + 0.3f * obj shadow->getColor();
    }

    else {surface color = 0.1f * surface color;} //0.2 = ambient scale factor
}

color += surface color;
```

Cone:



A cone was implemented using the equations given in the lecture notes (lecture 8 slide 46-47):

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

$$tan(\theta) = \frac{R}{h}$$

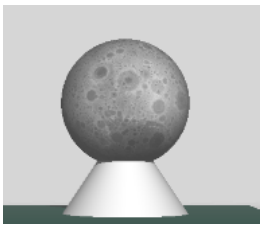
The normal for the cone is:

$$n = (sin(\alpha)cos(\alpha),\ sin(\theta),\ cos(\alpha)cos(\theta))$$

$$\alpha = arctan(\frac{x - x_c}{z - z_c})$$

$$\theta = half\ of\ the\ cone\ angle$$

Textured Sphere.



The sphere is covered with a moon texture. The equations used to texture the sphere are (from wiki):

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

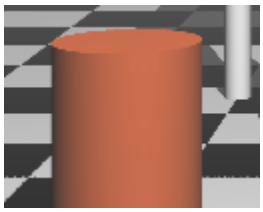
$$v = 0.5 - \frac{arcsin(d_y)}{\pi}$$

These values are then subbed into the setColor function used in lab 8.

```
//sphere texture
if (ray.index == 3)
{
    glm::vec3 textnormvec = obj->normal(ray.hit);
    float u = (atan2(-textnormvec.x, -textnormvec.z) / (2*M PI)) + 0.5;
    float v = 0.5 - (asin(-textnormvec.y) / M PI);
    obj->setColor(texture.getColorAt(u, v));
}
```

Cylinder:

A cylinder with a cap is in the scene. We can see this in the image to the right. The cylinder was made with the intersection equation (lecture 8 slide 39-40):



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

Where:

$$a = d_x^2 + d_z^2$$

$$b = d_x(x_0 - x_c) + d_z(z_0 - z_c)$$

$$c = (x_0 - x_c)^2 + (z_0 - z_c)^2 - R^2$$

Then subbing in a, b, c into the quadratic formula:

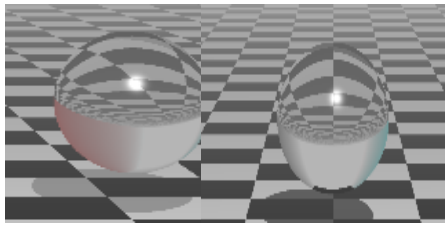
$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

to get t1 and t2.
For the cap there is a condition to check if the point is less than the height and greater than the center.

Refractive sphere:
For the refractive sphere the shadow is lighter, as seen in the figure to the left compared to the right. The method for the refractive object is the same as the lecture notes (lecture 8 slide 21-28)

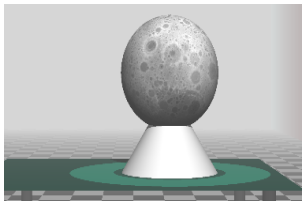
n = obj->normal(ray.hit);
g = glm::**refract**(***d***, ***n***, eta); //eta = η_1/η_2
Ray refrRay(ray.hit, ***g***)
refrRay.closestPt(sceneObjects);
m = obj->normal(refrRay.hit);
h = glm::**refract**(***g***, ***–m***, 1.0f/eta);

```
//For retraction
if (obj->isRefractive() && step < MAX_STEPS)
{
    float refractCoeff = obj->getRefractionCoeff();
    float eta = 1.0f / obj->getRefractiveIndex();
    glm::vec3 refractN = obj->normal(ray.hit);
    glm::vec3 d = ray.dir;
    glm::vec3 g = glm::refract(d, refractN, 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 refrRay2(refrRay.hit, h);
    glm::vec3 refractColor = trace(refrRay2, step + 1);
    color += (refractColor * refractCoeff);
}
```

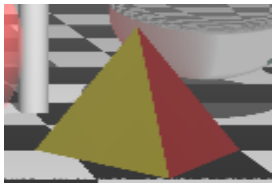


I used an eta value of 1.2 so that it looks like the floor is upside down in the sphere as seen in the image to the right.

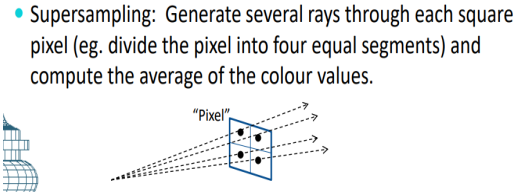
Spotlight:
The moon textured sphere has a spotlight that is positioned above the sphere with a higher y axis value. The spotlight direction is going straight down towards the sphere (0, -1, 0) which gives the spotlight shown in the figure to the right (notes 7).



Pyramind:
This pyramid is made of four planes each with 3 points. The front sides need to be in anti clockwise order so the normal is facing outwards, however the back sides of the pyramid need to be in clockwise direction so the normal will be facing outwards instead of inwards.



Anti-aliasing:
Supersampling anti-aliasing was also implemented. The method is the same as the ones in the lecture notes where instead of one ray there are four rays through each square pixel into four equal segments, then the average color was computed (lecture 8 slide 36). The purpose of anti-aliasing is so that it will have smoother surfaces and edges in the scene but will slow down the generation as it is producing four times the original ray.



From the figure below we can see that the scene with anti-aliasing with 100 num divs has more rays hitting the pixels while the scene with no anti-aliasing has less/only hitting one.

Figure with anti aliasing with 100 num divs

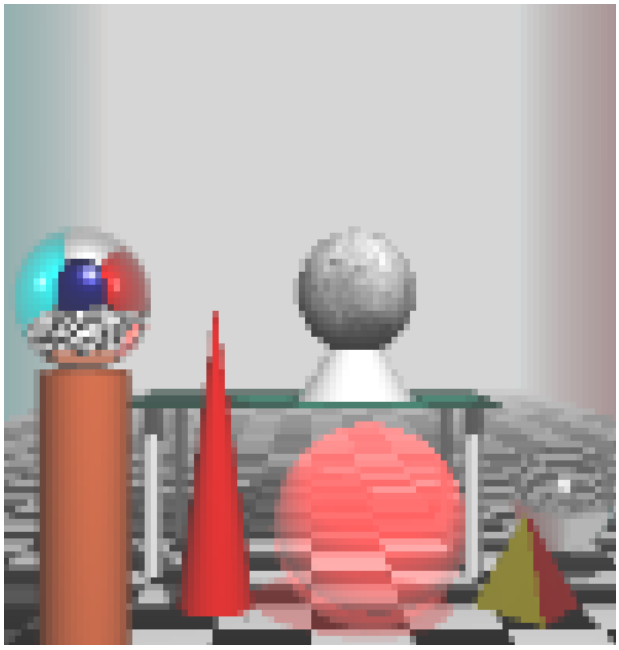
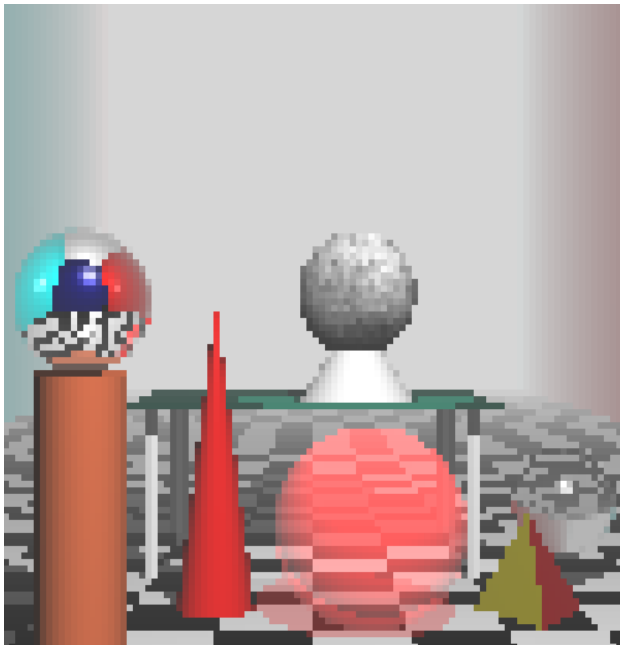


Figure without anti aliasing with 100 num divs



Fog:

For the fog I used the equation given to us in the notes (notes 7):

$$t = \frac{(ray.hit.z) - z_1}{z_2 - z_1}$$

$$color = (1 - t) \times color + t \times white$$

Code for fog:

```
//~ //fog
float z1 = 20;
float z2 = -280;
float t = (ray.hit.z - z1) / (z2 - z1);
color = ((1-t) * color + t * glm::vec3(0.8, 0.8, 0.8));


return color;
```

Build instructions:

Unzip to the desired folder then open terminal in that directory.

In the terminal run:

1) cmake
2) make
3) ./RayTracer.out->Finished.

References:

https://en.wikipedia.org/wiki/UV_mapping
https://learn.canterbury.ac.nz/pluginfile.php/3393370/mod_resource/content/27/Lectures/Lec08_RayTracing.pdf
https://learn.canterbury.ac.nz/pluginfile.php/3393458/mod_resource/content/15/Labs/Lab08_RayTracing2.pdf
https://learn.canterbury.ac.nz/pluginfile.php/3393450/mod_resource/content/20/Labs/Lab07/Lab07_RayTracing1.pdf
https://learn.canterbury.ac.nz/pluginfile.php/3689718/mod_resource/content/22/Lectures/Note07_RayTracing.pdf
Moon texture: https://svs.gsfc.nasa.gov/4720