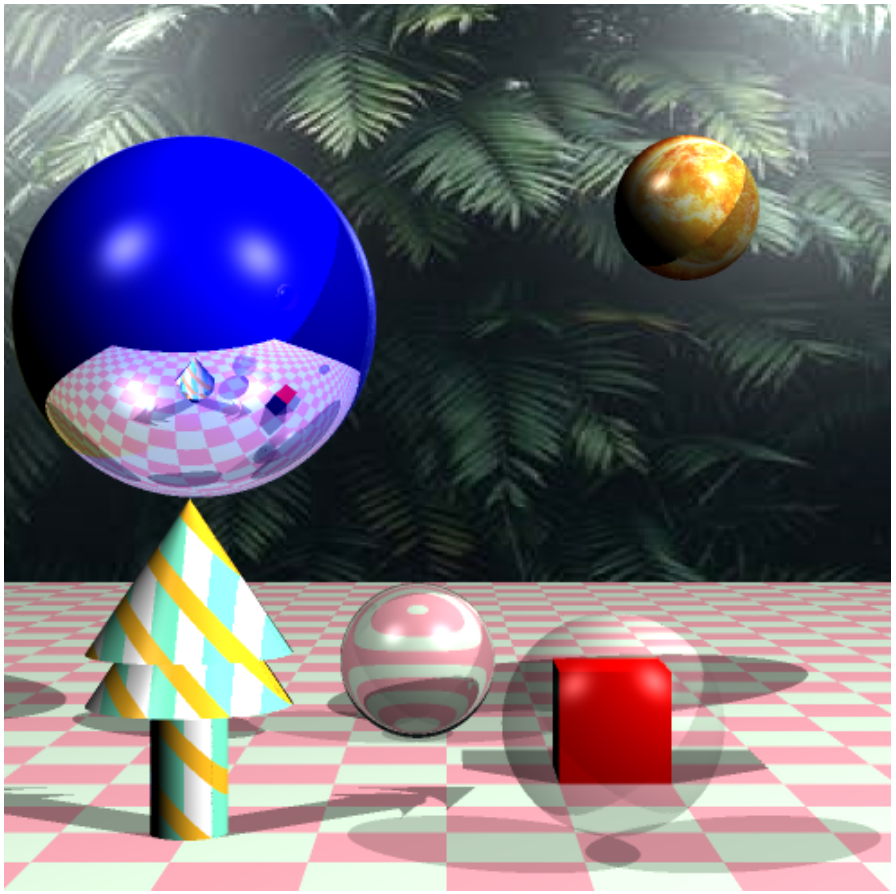
Minfang Yu 75219495

Commands to run:

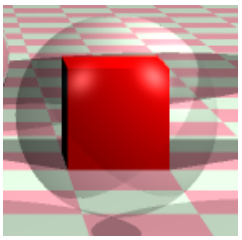
1. **cmake CMakeLists.txt**
2. **make**
3. **./RayTracer.out**

My ray tracer is in a forest with a sun, a tree, two magic bubbles(one is transparent and another is refraction) and one magic reflective blue bubble mirror.



Basic Requirements:

1. **Transparent object:**

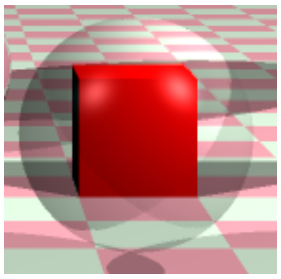


It is a hollow sphere(transparent object). I implemented it by setting the eta to 1/1.003, which makes it different to refractions.

2. **Shadows:**

It is obvious that transparent and refractive spheres have lighter shadows. I implemented it by detecting when the shadow ray hits the transparent and refractive spheres and then letting the color lighter.

3. **Box:**



Behind the hollow sphere is the red box. I make the box by constructing a set of planes.

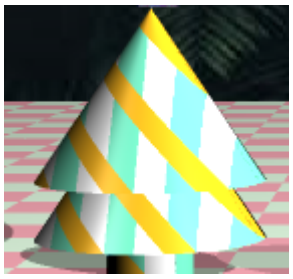
```
Plane *plane1 = new Plane(p1, p2, p3, p4);//6
Plane *plane2 = new Plane(p4, p3, p6, p7);//7
Plane *plane3 = new Plane(p8, p5, p2, p1);//8
Plane *plane4 = new Plane(p4, p7, p8, p1);//9
Plane *plane5 = new Plane(p2, p5, p6 ,p3);//10
Plane *plane6 = new Plane(p5, p8, p7, p6);//11
```

4. **Chequered pattern:**

The floor plane is generated by a chequered pattern in pink and white.

Extensions:

1. **Cone:**



The cone is generated by the math equation (in lecture8):

For any point (x, y, z) on the cone,

$(x - x_c)^2 + (z - z_c)^2 = r^2$

where, $r = \left(\frac{R}{h}\right)(h - y + y_c)$

```
float a = pow(dir.x, 2)+pow(dir.z,2)-pow((radius/height)*dir.y, 2);
float b = 2*(dir.x*(p0.x-center.x)+dir.z*(p0.z-center.z)-(pow(radius/height,2.0)*dir.y*(p0.y - height  - center.y)));
float c = pow(p0.x-center.x,2)+pow(p0.z-center.z,2)-pow(radius/height,2.0)*(pow(p0.y-center.y,2.0)+height*(height-2*p0.y+2*center.y));

float delta = pow(b,2) - 4*a*c;
```

2. **Cylinder:**

The cylinder is generated by the math equation (in lecture8):

Ray equation:

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

- 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 = pow(dir.x, 2)+ pow(dir.z, 2) ;
float b = 2 * ((p0.x - center.x)*dir.x + (p0.z - center.z)*dir.z);
float c = pow((p0.x - center.x), 2) + pow((p0.z - center.z), 2) - pow(radius, 2);
```

3. **Refraction:**

It is a refraction sphere. I implemented it by setting the eta to 1/1.03, which makes it different to transparent.

```
if (ray.index == 2) //refraction
{
   float coeff_refraction = 0.4;
   float eta = 1/1.03;
   glm::vec3 n = sceneObjects[ray.index]->normal(ray.hit);
   glm::vec3 g = glm::refract(ray.dir, n, eta);
   Ray refrRay(ray.hit, g);
   refrRay.closestPt(sceneObjects);
   if(refrRay.index == -1) return backgroundCol;
   glm::vec3 m = sceneObjects[refrRay.index]->normal(refrRay.hit);
   glm::vec3 h = glm::refract(g, -m, 1.0f/eta);
   Ray refrRay2(refrRay.hit, h);
   refrRay2.closestPt(sceneObjects);
   if(refrRay2.index == -1) return backgroundCol;

   glm::vec3 refractionColor = trace(refrRay2, step+1);
   color = color * coeff_refraction + refractionColor*(1 - coeff_refraction);

    obj->setColor(color);
}
```

4. **Multiple light sources and shadows:**

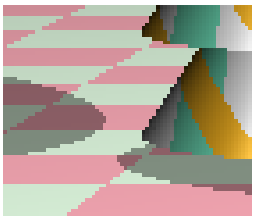
```
glm::vec3 lightPos(90, 90, -3);
glm::vec3 lightPos2(-90, 90, -3);
```

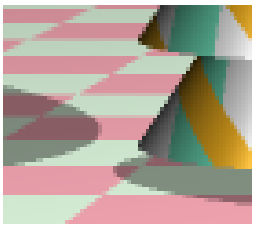
It is obvious that there are two light sources and every object has two shadows. I implemented it by detecting whether an object was hit by light1 and whether an object was hit by light2 respectively.

5. **Anti-aliasing:**

I use anti-aliasing because it can reduce distortion artefacts such as jaggedness along edges of polygons and shadows. After using anti-aliasing, the output is smoother and looks nicer. I implemented it by supersampling that divides one pixel into four equal segments and computes the average of the colour values.

without anti-aliasing: with anti-aliasing:





```
//anti-alisaing
glm::vec3 dir1(xp+0.25*cellX, yp+0.25*cellX, -EDIST);
glm::vec3 dir2(xp+0.75*cellX, yp+0.75*cellX, -EDIST);
glm::vec3 dir3(xp+0.75*cellX, yp+0.25*cellX, -EDIST);
glm::vec3 dir4(xp+0.25*cellX, yp+0.75*cellX, -EDIST);
Ray ray1 = Ray(eye, dir1);
Ray ray2 = Ray(eye, dir2);
Ray ray3 = Ray(eye, dir3);
Ray ray4 = Ray(eye, dir4);
```

6. **A non-planar object textured using an image:**

The sphere is textured by a sun image. I implemented it by converting angle to texture coordinate.(Wikipedia: UV Mapping)

```
if(ray.index == 3)  //sun
{
    glm::vec3 centre(10, 10.0, -75);
    glm::vec3 d = glm::normalize(ray.hit-centre);
    float u = atan2(d.x, d.z) / (2*M_PI) + 0.5;
    float v = 0.5 - asin(d.y) / M_PI;
    color=texture1.getColorAt(u, v);
    obj->setColor(color);
}
```

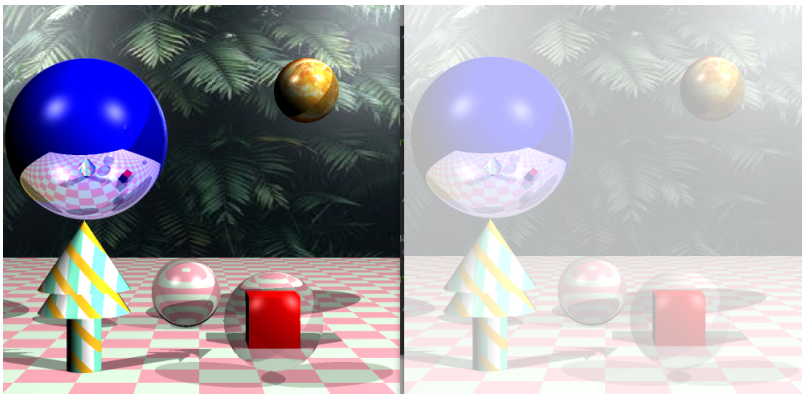
7. **Procedural pattern:**

I generated the pattern on the tree.

```
if(ray.index == 12 || ray.index == 13 || ray.index == 14) //procedural pattern
{
    if ((int(ray.hit.x + ray.hit.y) % 3 == 0)){
        color = glm::vec3(1,0.73,0.13);
    }
    else if((int(ray.hit.x) % 2 == 0)){
        color = glm::vec3(0.46,0.93,0.77);
    }
    else{
        color = glm::vec3(1,1,1);
    }
    obj->setColor(color);
}
```

8. **Fog:**

I generated the fog by using GL_FOG.



```
glEnable(GL_FOG);
glHint(GL_FOG_HINT, GL_NICEST);
float fogColor[4] = {1,1,1,1.0f};
glFogi(GL_FOG_MODE, GL_LINEAR);
glFogfv(GL_FOG_COLOR, fogColor);
glFogf(GL_FOG_DENSITY, 0.5f);
glHint(GL_FOG_HINT, GL_DONT_CARE);
glFogf(GL_FOG_START, -50.0f);
glFogf(GL_FOG_END, 20.0f);
```

Estimate of time for my program run on my VM: 15s

Reference:

1. Lectures, Labs
2. Texturing a Non-Planar Object: https://en.wikipedia.org/wiki/UV_mapping
3. Shadow Mapping: https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping
4. Fog: https://nehe.gamedev.net/tutorial/cool_looking_fog/19001/