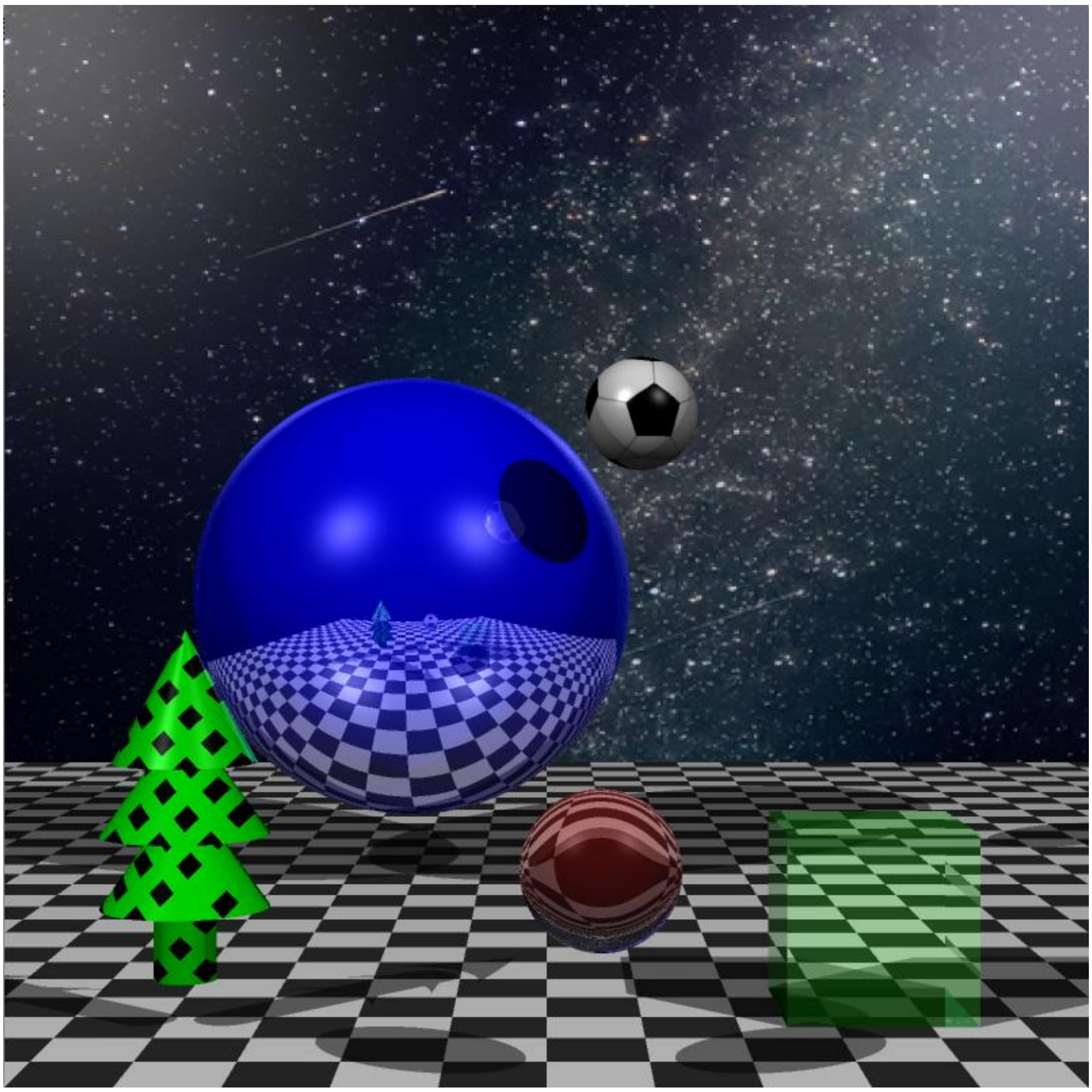
COSC 363 Assignment2

Hao Li

83838861



Plane: There is a background plane behind all of the objects and there is a 'star' texture on it.

Cylinder: The cylinder applied the ray equations and intersection equation from slide 38, lecture 9, which is x =$d_0 + d_x t$; x =$y_0 + d_y t$; z = $z_0 + d_z t$. The intersection equation is to find the closest point if intersection gives cylinders of infinite height. The related equation I implemented is below.

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

The aspect of the code is:

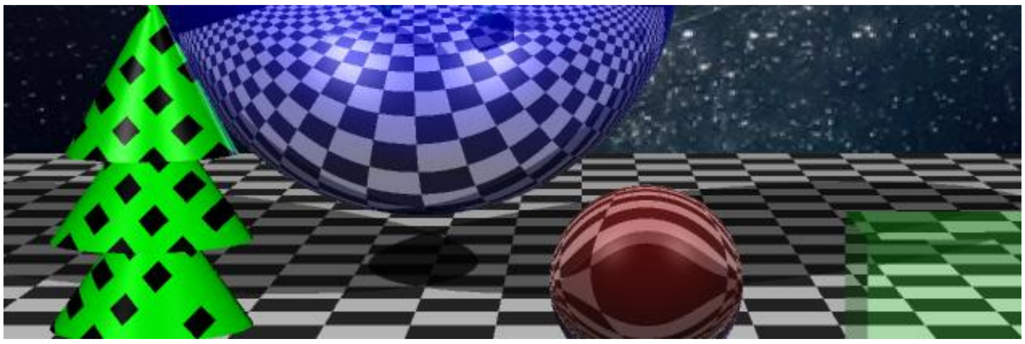
```
float x = posn.x - center.x;
float z = posn.z - center.z;

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

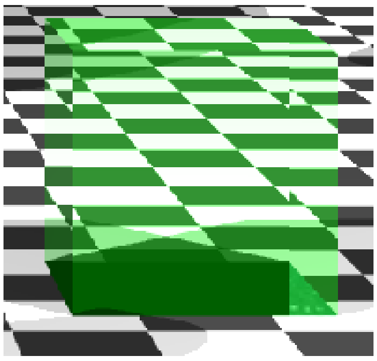
Cone: The cone used the same ray equations as Cylinder, but it takes more equations, $(x - x_c)^2 + (z - z_c)^2 = r^2$ where $r = \left(\frac{R}{h}\right)(h - y + y_c)$
The aspect of the code is:

```
float a = pow(dir.x, 2.0)+pow(dir.z,2.0)-pow((radius/height)*dir.y, 2.0);
float b1 = pow(radius/height,2.0)*dir.y*(posn.y-center.y-height);
float b = 2*(dir.x*(posn.x-center.x)+dir.z*(posn.z-center.z)-b1);
float c1 = (pow(posn.y-center.y,2.0)+height*(height-2*posn.y+2*center.y));
float c = pow(posn.x-center.x,2.0)+pow(posn.z-center.z,2)-pow(radius/height,2.0)*c1;
```

Multiple light sources including multiple shadows:



There are two separate light sources at (100, 100, 5) and (-100, 100, 5), every object had two shadows. Because of multiple light sources, one shadow is easy to be covered by another light source, it causes the color of this shadow is faded as grey instead of black, but the overlapping area of two shadows is staying black even darker because there is no more light source is able to cover that shadow. Aiming at this, I increased the shadow vector and decreased the light vector. The related code is *colorSum = (ambientCol*materialCol) * 0.1f; colorSum = ambientCol*materialCol + (lDotn * materialCol + specularTerm)*0.5f.*



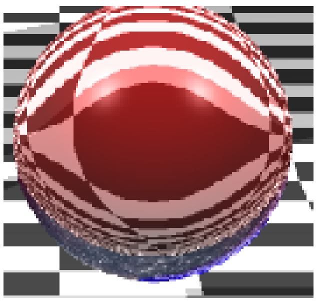
Transparent: The cube is transparent, there is a variable called transp_rate, it is set for adjusting the level of transparency, the initial rate is 85%, and eta set as 1/1.005 to prevent the refraction on the cube. There are two rays tranRay1 and Rayout, the transparency is done by recursively tracing the rays, and hence we have two normal rays. The code is below

```
//transparent  -  cube
if(ray.xindex >= 7 && step < MAX_STEPS){
    float transp_rate = 0.15;
    float eta = 1/1.005;

    glm::vec3 normalVector1 = sceneObjects[ray.xindex]->normal(ray.xpt);
    glm::vec3 refraction1 = glm::refract(ray.dir, normalVector1, eta);
    Ray tranRay1(ray.xpt, refraction1);
    tranRay1.closestPt(sceneObjects);
    if(tranRay1.xindex == -1) return backgroundCol;

    glm::vec3 normalVector2 = sceneObjects[tranRay1.xindex]->normal(tranRay1.xpt);
    glm::vec3 refraction2 = glm::refract(refraction1, -normalVector2, 1.0f/eta);
    Ray Rayout(tranRay1.xpt, refraction2);
    Rayout.closestPt(sceneObjects);
    if(Rayout.xindex == -1) return backgroundCol;

    glm::vec3 tranCol = trace(Rayout, step+1);
    colorSum = colorSum * transp_rate + tranCol*(1 - transp_rate);
    return colorSum;
}
```



Refraction: There is a sphere with the refraction of the floor, it is implemented through the similar the way as a transparent cube, and the only thing changed is an important variable eta, means the ratio of refractive indices. In order to achieve refraction instead of transparency, I set eta is 1/1.05 and transp_rate is 70% in the refractive sphere. It is different from the transparent cube.

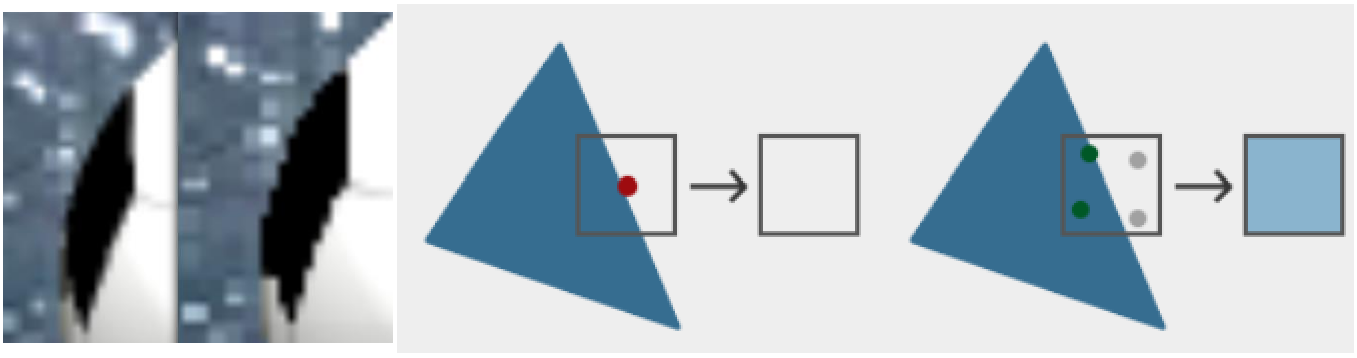
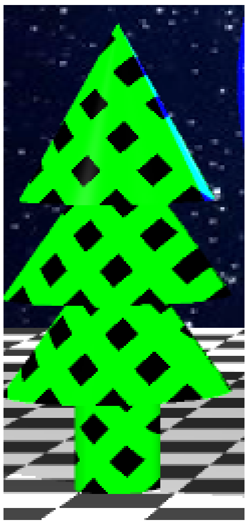


Figure1

Anti-aliasing: It's clear to see the difference after anti-aliasing implemented, objects have smoother borders. I take the screenshot of the cylinders as a detail to show the difference. In figure 1, the jagged border is obvious, it is important and necessary to estimate that if there is a high requirement for legibility. According to figure 1, I applied multisampling to achieve anti-aliasing, it does not use a single sampling point for determining coverage of the triangle but use multiple sample points. It could make more sampling points along the border, the tighter relationship of sampling points causes the border being smoother.



Procedural pattern: This tree is constructed by three cones and one cylinder. I use ray.xpt.x - ray.xpt.y and ray.xpt.x + ray.xpt.y, to indicate the two directions of green lines, finally set the rhombus between two green lines to black.

```
if(ray.xindex == 1 || ray.xindex == 3 || ray.xindex == 7 || ray.xindex == 8)
{
    if ((int(ray.xpt.x - ray.xpt.y) % 2 == 0)){
        materialCol = glm::vec3(0,1,0);
    }
    else if((int(ray.xpt.x + ray.xpt.y) % 2 == 0)){
        materialCol = glm::vec3(0,1,0);
    }
    else{
        materialCol = glm::vec3(0,0,0);
    }
}
```

A non-planar object textured using an image: There is a sphere textured a football image.

```
if (ray.xindex == 2)      //texture football
{
    float a = asin(normalVector.x) / M_PI + 0.5;
    float b = asin(normalVector.y) / M_PI + 0.5;
    materialCol = texture2.getColorAt(a, b);
}
```

Reference:
COSC363-19S1 Lectures and Labs
https://learnopengl.com/Advanced-OpenGL/Anti-Aliasing