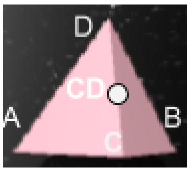
ID: 97245310
Name: Melody Zhu

COSC 363 Assignment 2 Ray Tracer Report

Build Command: g++ -Wall -o "%e" RayTracer.cpp Ray.cpp SceneObject.cpp Sphere.cpp Plane.cpp Cylinder.cpp Cone.cpp TextureBMP.cpp -lm -lGL -lGLU -lglut

Extra Features

1. Tetrahedron: The regular tetrahedron is constructed using the plane class by setting the fourth point to be a point on one of the edges of that particular plane. ie, the middle point of CD is chosen to be the fourth point:




```
glm::vec3 CD = glm::vec3((C.x+D.x)/2,(C.y+D.y)/2,(C.z+D.z)/2);
```

Then we can compute the triangle plane BDC as:

```
Plane *triangle2 = new Plane(B,D,CD,C,color);
```

2. Cylinder: The cylinder class calculates the point of intersection and surface normal of the traced ray. It is constructed using the following formulas:

<u>**Ray equation**</u>: $x = x_o + d_x t$; $y = y_o + d_y t$; $z = z_o + d_z t$

where d denotes the direction of the ray, o denotes the ray's origin, c denotes the center of the cylinder and the value of t denotes the distance from the ray's origin to the point on the ray.

<u>**Intersection equation**</u>: The intersection point is computed by solving the quadratic equation for t where R is the radius of the cylinder:

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

```
float Cylinder::intersect(glm::vec3 pos, glm::vec3 dir)
{
    glm::vec3 d = pos - center;
    float a = (dir.x * dir.x) + (dir.z * dir.z);
    float b = 2 * (dir.x * d.x + dir.z * d.z);
    float c = d.x * d.x + d.z * d.z - (radius * radius);
```

<u>**Surface normal vector**</u>:

(un-normalized) $\boldsymbol{n} = (x - x_c,\ 0,\ z - z_c)$

(normalized) $\boldsymbol{n} = (\ (x - x_c)/R,\ \ 0,\ \ (z - z_c)/R\ )$

```
glm::vec3 d = p - center;
glm::vec3 n = glm::vec3 (d.x,0,d.z);
n = glm::normalize(n); //normalize
```

3. Cone: The cone class calculates the point of intersection and surface normal of the traced ray. Besides using the same ray equation as in the cylinder class, the cone also uses: (1) $(x - x_c)^2 + (z - z_c)^2 = r^2$; (2) $r = (R/h)(h - y + y_c)$;

(3) $tan(\theta) = R/h$ ($\theta = half\ cone\ angle$)

where (x,y,z) is any point on the cone; R is the radius and r is height of the cone.
The intersection equation is obtained by substituting (3) into (2), then (2) and the ray equation into (1). Solving for t we get the intersection point.

<u>**Intersection equation:**</u>

$$t^2(d_x^2 + d_z^2 - tan^2 d_y^2) + 2t\{d_x(x_o - x_c) + d_z(z_o - z_c) + tan^2(h - y_o + y_c)d_y\}$$
$$+ \{(x_o - x_c)^2 + (z_o - z_c)^2 - tan^2(h - y_o + y_c)^2\} = 0$$

ID: 97245310
Name: Melody Zhu

```
float Cone::intersect(glm::vec3 pos, glm::vec3 dir)
{
    glm::vec3 d = pos - center;
    float yd = height - pos.y + center.y;
    float stan = (radius / height) * (radius / height);
    float a = (dir.x * dir.x) + (dir.z * dir.z) - (stan*(dir.y * dir.y))
    float b = 2*(d.x*dir.x + d.z*dir.z + stan*yd*dir.y);
    float c = (d.x*d.x) + (d.z*d.z) - (stan*(yd*yd));
    float delta = b*b - 4*(a*c);
```

Surface normal vector:

```
glm::vec3 d = p-center;
float r = sqrt(d.x * d.x + d.z * d.z);
glm::vec3 n= glm::vec3 (d.x, r*(radius/height), d.z);
n=glm::normalize(n);
```

4. Multiple light sources: The scene includes two light sources with intensity 60% and 40%.(light intensity is a scaling factor of the (diffuse + specular) term, otherwise the scene would be too bright) Each object satisfying the ray-shadow constraints has two shadows. The two shadows of the transparent green sphere were adjusted to be visibly different due to the different intensities of the light sources.

5. Refraction: The green sphere is refractive. This is done by recursively tracing the rays. In each step, a ray is traced twice as we need to compute 2 normals and refractive rays when transmitting in and out of the sphere. The spheres shown in Figure 1 and 2 demonstrates refractions with different ETAs.

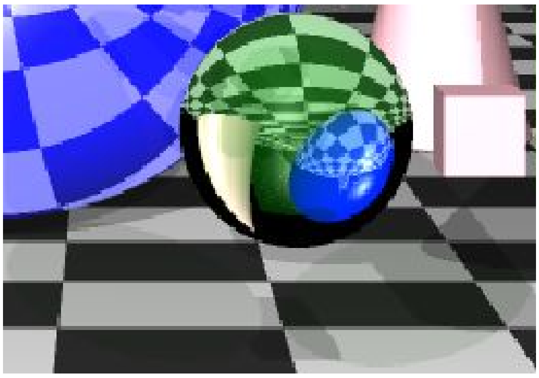


Figure 1: ETA=1.5

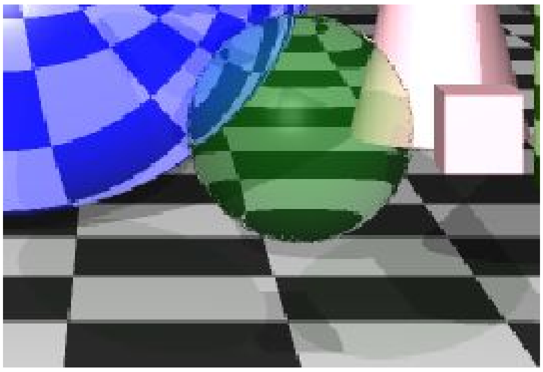


Figure 2: ETA=1.003

6. Transparent object: The green sphere is 80% transparent (transparency=0.2). This is achieved by multiplying the sphere's current color by the transparency (colorSum * transparency) and then adding the color that can be refracted through the sphere (refracCol2 *(1-transparent)). As shown in figure 1 and 2, the shadows of the transparent sphere is also made lighter with a hint of the sphere's color by the following operation:

```
if(shadow.xindex == 2){
    colorSum += (lDotn*col + specular2)*glm::vec3(0.4)+sceneObjects[2]->getColor()*glm::vec3(0.02);
}//transparent objects:lighter shadow with a hint of the obj's color
```

ID: 97245310
Name: Melody Zhu

```
//----------------Refraction with transparency---------------------------
    if((ray.xindex == 2) && (step < MAX_STEPS)){
        glm::vec3 refracDir1 = glm::refract(ray.dir,normalVector,1.0f/ETA);
        Ray refracRay1(ray.xpt,refracDir1);
        refracRay1.closestPt(sceneObjects);
        if(refracRay1.xindex==-1){
            return backgroundCol;
        }
        glm::vec3 normalVector2 = sceneObjects[refracRay1.xindex]->normal(refracRay1.xpt);
        glm::vec3 refracDir2 = glm::refract(refracDir1, -normalVector2, ETA);
        Ray refracRay2(refracRay1.xpt,refracDir2);
        refracRay2.closestPt(sceneObjects);
        if(refracRay2.xindex==-1){
            return backgroundCol;
        }
        glm::vec3 refracCol2 = trace(refracRay2, step+1);
        colorSum = colorSum * transparency + refracCol2*(1-transparency); //transparent object
        return colorSum;
    }
```

7. Non-planar object textured using an image: The Earth is a sphere textured by Earth.bmp:

```
glm::vec3 center(6.5, 5.0, -70.0);
glm::vec3 d=glm::normalize(ray.xpt-center);
float u=(0.5-atan2(d.z,d.x)+M_PI)/(2*M_PI);
float v=0.5+asin(d.y)/M_PI;
col = texture2.getColorAt(u, v);
```

8. Non-planar object textured using a procedural pattern: The cylinder is textured procedurally by:

```
if ((int(ray.xpt.x+ray.xpt.y-13) % 2 == 0)){
    col = glm::vec3(0.2,0.4,0);
}else{
    col = glm::vec3(0.8,1,0.6);
}
```

9. Anti-aliasing: Supersampling is used here in order to minimize distortion artefacts such as jaggedness along edges of polygons and shadows caused by the finite set of rays generated through a discretized image space. A square pixel is divided into four equal segments and four rays are generated through the center of each segment. The average of the color values traced from the 4 rays is computed and returned. As shown in Figure 3 and 4, it's clear that with anti-aliasing implemented, objects have smoother edges and hence improves the overall rendering quality.

ID: 97245310
Name: Melody Zhu

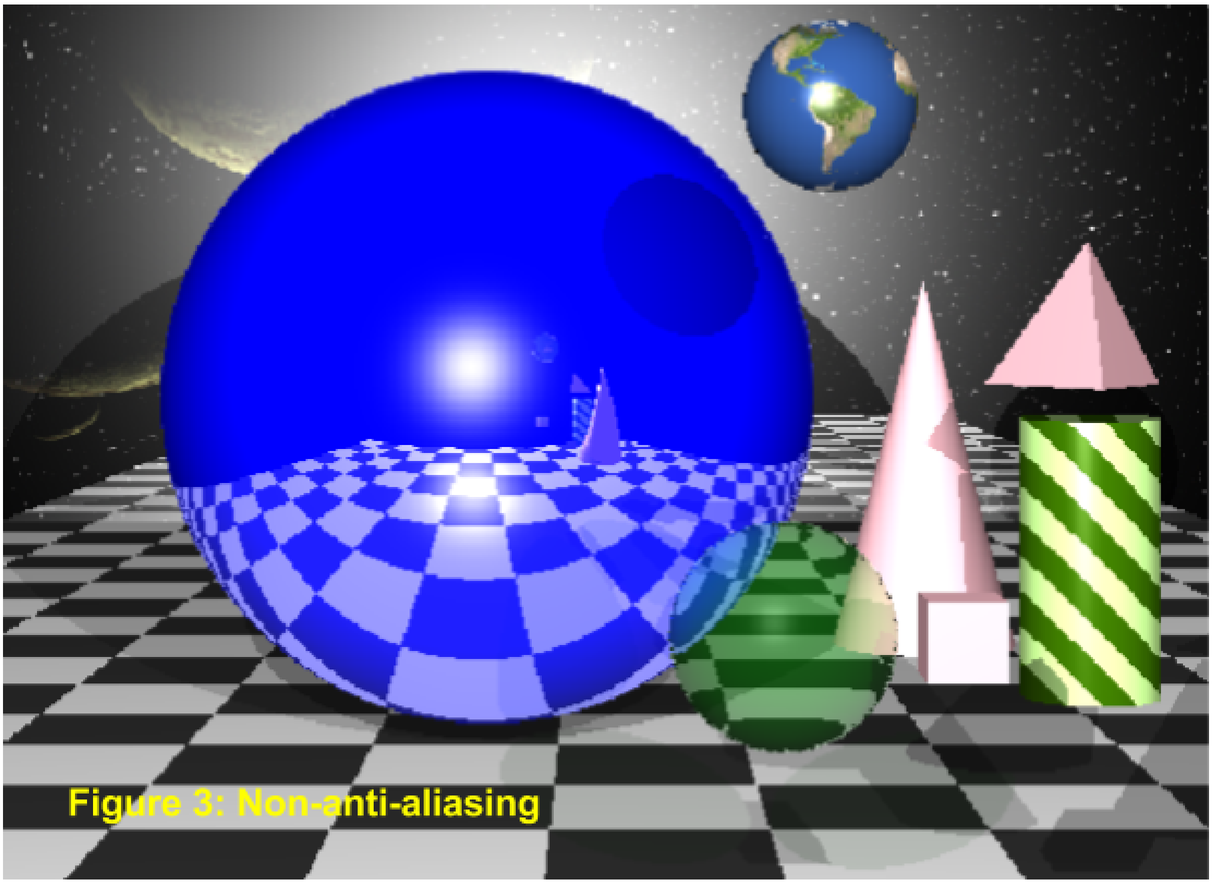


Figure 3: Non-anti-aliasing

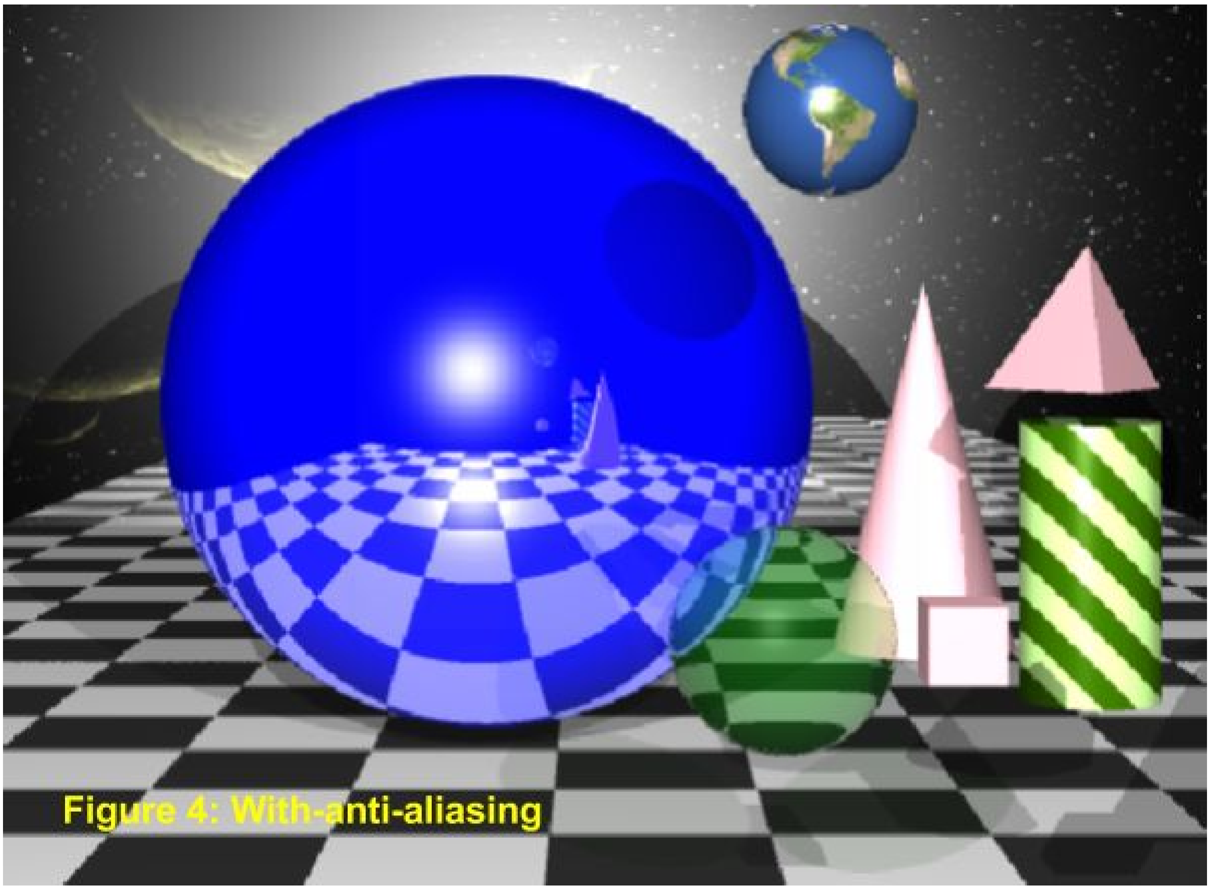


Figure 4: With-anti-aliasing

ID: 97245310
Name: Melody Zhu

Success And Failures

While hard to create a bigger scene as limited by the computation cost, the wall is textured and made reflective to extend the vision. Colors are chosen carefully to create a calm and mysteries tone to match the theme 'universe' of my ray tracer.

One challenge that I came across was creating a spot light. To set light2 as a spotlight with cut off angle $\pi/4$ along the direction of the spotVector, I tried the following:

```
----------spot light-----------------------------------------
 float alpha = M_PI/4.0f;
 glm::vec3 spot(16,-20,-40);
 glm::vec3 spotVector = glm::normalize(spot-light2);
 float theta = glm::acos(glm::dot(spotVector, lightVector2));
 float fade = 1.0f-theta/alpha;
 fade = max(fade,0.0f);
```

And then updated colorSum in the shadow computation of light2 as:

```
colorSum=ambientTerm*col + (lDotn2*col + specular2)*fade
```

As shown in Figure 5, such operation seems to bright up all the shadows caused by light2 rather than create a proper spot light domain. I'm keen to figure out the reason for that and a way to correctly set my spotlight.

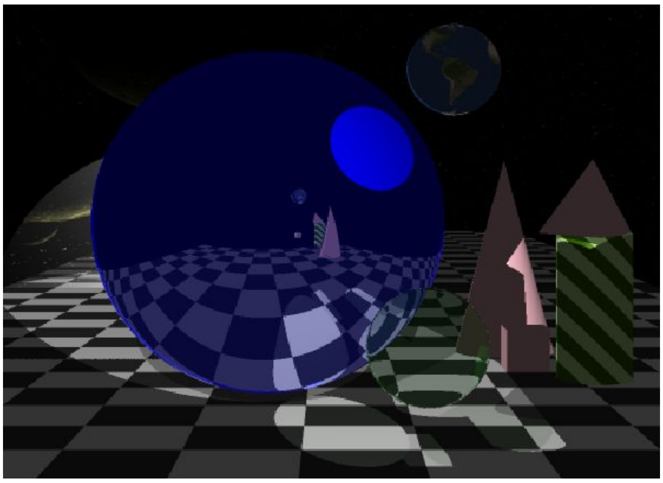


Figure 5: Incorrect spotlight

References

1. COSC 363 lecture and lab notes by Professor R. Mukundan
2. https://en.wikipedia.org/wiki/UV_mapping