COSC363 – Assignment 2
Yuhang Wu - 31992820

CYLINDER

Intersection Equation:
$t^2(d_x^2 + d_z^2) + 2t\{(d_x(x_0 - x_c) + d_z(z_0 - z_c)\} + \{(x_0 - x_c)^2 + (z_0 - z_c)^2 - R^2\} = 0$
Normalized Equation: $\mathrm{n} = (\frac{x-x_c}{R},\ 0,\ \frac{z-z_c}{R})$
There are four inputs, which are the position of the center, on the center of the bottom cycle, the radius, height and color of the cylinder. In the intersection part, instead of selecting the minimum t, keep both t1 and t2, set the small t as the rear side of the cylinder and the bigger t as the front side of the cylinder. Moreover, use the equation $\mathrm{y} = y_0 + d_y t$ limiting the height of the cylinder.

CONE

Intersection Equation:
$\mathrm{k} = \left(\frac{R}{h}\right)(h - y + y_c)$
$t^2\left(d_x^2 + d_z^2 - k\left(d_y^2\right)\right) + 2t\left(d_x(x_0 - x_c) + d_z(z_0 - z_c) + k\left(d_y\right)h\right) + ((x_0 - x_c)^2 + (z_0 - z_c)^2 - kh^2) = 0$
According to the equations: $(\mathrm{x} - x_c)^2 + (z - z_c)^2 = r^2$ where $\mathrm{r} = \left(\frac{R}{h}\right)(\mathrm{h} - \mathrm{y} - y_c)$. The normalized method would be: $\mathrm{n} = (\frac{x-x_c}{R},\ \mathrm{r}(\frac{Height}{Radius}),\ \frac{z-z_c}{R})$.
There are four inputs, which are the position of the center, on the center of the bottom cycle, the radius, height and color of the cone. In the intersection part, instead of selecting the minimum t, keep both t1 and t2, set the small t as the rear side of the cylinder and the bigger t as the front side of the cone. Moreover, use the equation $\mathrm{y} = y_0 + d_y t$ limiting the height of the cone.

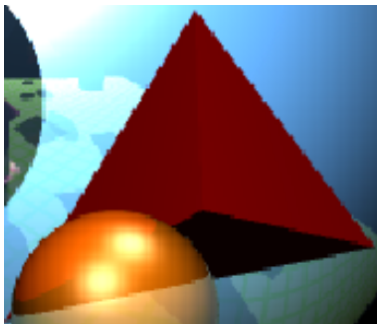
NON-PLANE PROCEDURAL PATTERN TEXTURE

```
if (ray.xindex == 5) //cylinder
{
    if((int)(ray.xpt.y) % 2 == 0) col = glm::vec3(1.0, 0.2, 0.6);
}

if (ray.xindex == 6) //cone
{
    if((int)(ray.xpt.z + ray.xpt.x) % 2 == 0) col = glm::vec3(1, 0.6, 0.8);
}
```

The first if block maps the cylinder with horizontal stripes, every even or odd ray.xpt.y has different color. And the second if block map the cone with diagonal stripes, every oven or odd (ray.xpt.z + ray.xpt.x) has different color.

TETRAHEDRON

The tetrahedron object is created by referencing the idea of Plane.h and Plane.cpp in the lab files. Define a plane by using three vertexes and check the point p is inside the triangle.



REFRACTION OBJECT

Followed the instruction provided on the lecture note and substituted the suitable variables. Add a condition to return the background and a ray which shoot out of the object and recursively trace it to get the corresponding refraction color.

```
glm::vec3 refractDir = glm::refract(ray.dir, normalVector, eta);
Ray refractRay(ray.xpt, refractDir);
refractRay.closestPt(sceneObjects);

if(refractRay.xindex != -1)
{
    glm::vec3 normalRefracted = sceneObjects[refractRay.xindex] -> normal(refractRay.xpt);
    glm::vec3 outRayDir = glm::refract(refractDir, -normalRefracted, 1.0f);
    Ray outRay(refractRay.xpt, outRayDir);
    glm::vec3 refractCol = trace(outRay, step+1);

    if (ray.xindex == 1) colorSum = refractCol; // refraction
    else if (ray.xindex == 3)
        colorSum = colorSum * glm::vec3(0.7) + refractCol* glm::vec3(0.5); // transparent
}
```



TRANSPARENT OBJECT

Similar idea of the refraction, reserve the partial material color of the object and weaken the strength of the refraction color, then combine them together.

```
colorSum = colorSum * glm::vec3(0.7) + refractCol* glm::vec3(0.5); // transparent
```

NON-PLANE TEXTURE

The instruction of texturing the sphere was found in the Github (by Yamohas) and reuse the Earth.bmp which is provided by Lab3.

```
float s = asin(normalVector.x)/M_PI + 0.5;
float t = asin(normalVector.y)/M_PI + 0.5;
col = earth.getColorAt(s,t);
```

MULTI-LIGHT SOURCES & SHADOWS

Follow the instruction provided in the labs, create a second light source:

```
glm::vec3 light2(-10, 20, 23); // light2
```

Use r2Dotv and sepcularCol2 to compute the specular color on the objects and the shadows of the second light source.

```
//light2 reflection
glm::vec3 lightVector2 = light2 - ray.xpt;
glm::vec3 normalLight2 = glm::normalize(lightVector2);
float l2Dotn = glm::dot(normalLight2, normalVector);

glm::vec3 reflVector2 = glm::reflect(-lightVector2, normalVector);
glm::vec3 normalReflVector2 = glm::normalize(reflVector2);
float r2Dotv = glm::dot(normalReflVector2,normalVector);
```

```
// ------------------- light2 shadow --------------------
Ray shadow2(ray.xpt, normalLight2);
shadow2.closestPt(sceneObjects);
float light2Dist = glm::length(lightVector2);

if ((0 >= l2Dotn) || (shadow2.xindex > -1 && shadow2.xdist < light2Dist))
    col =  ambientTerm*col;
else col =  ambientTerm*col + l2Dotn*col + specularCol2;
```

ANTI-ALIASING

The basic idea of Anti-Aliasing is generating 4 rays, rather than creating only one ray, through each cell. Trace all four rays to get the colors of each quarter of the cell and compute the average of the color values. The edges of the objects are smooth.

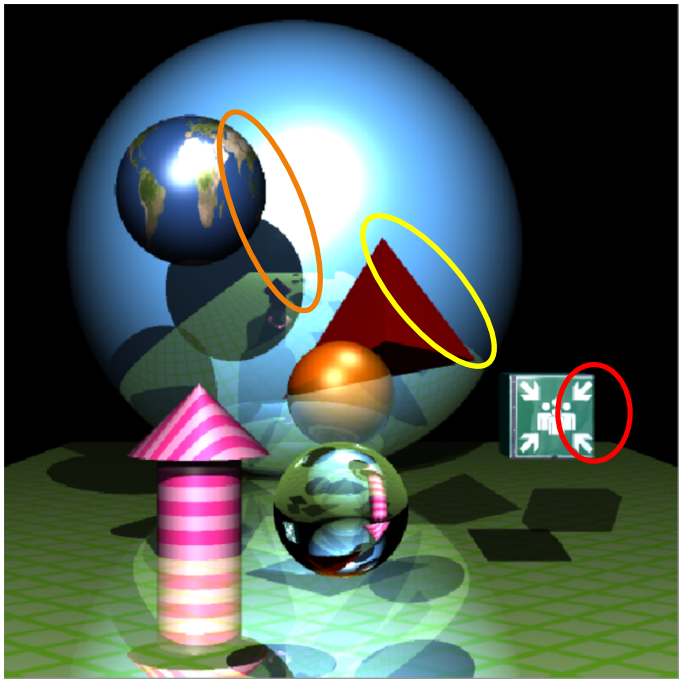
```
float xp, yp;
float cellX = (XMAX-XMIN)/NUMDIV;
float cellY = (YMAX-YMIN)/NUMDIV;

float quaterX = cellX/4;
float quaterY = cellY/4;
float quater3X = 3 * cellX/4;
float quater3Y = 3 * cellY/4;
```

Example of left bottom of a unit cell:

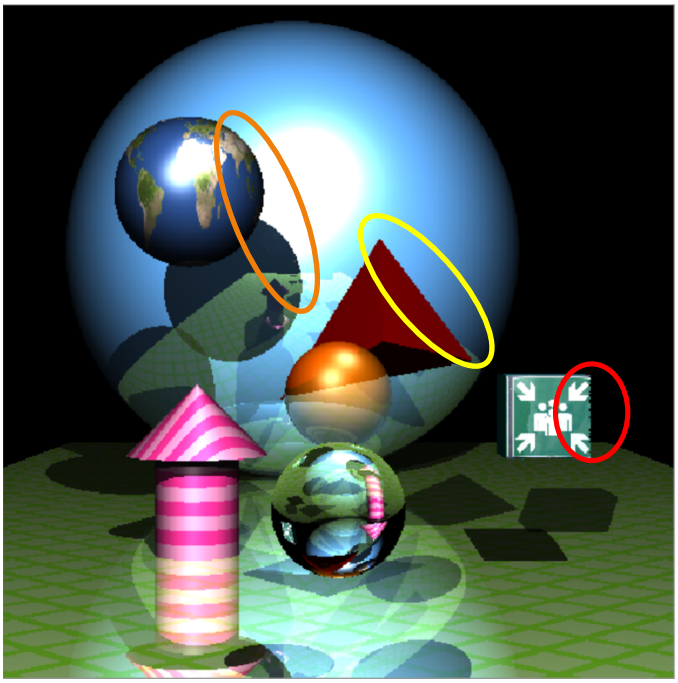
```
glm::vec3 dir1(xp+quaterX, yp+quaterY, -EDIST);
Ray ray1 = Ray(eye, dir1);
ray1.normalize();
glm::vec3 col1 = trace (ray1, 1);
```

Compute the average color value and pass to glColor3f:

```
float R = (col1.r + col2.r + col3.r + col4.r) / 4;
float G = (col1.g + col2.g + col3.g + col4.g) / 4;
float B = (col1.b + col2.b + col3.b + col4.b) / 4;
glColor3f(R, G, B);
```



With Anti-Aliasing



Without Anti-Aliasing

REFERENCES

- Some codes, functions, and methods were obtained from LEARN, lecture notes, and lab files.
- Reused the ‘Earth.bmp’ from lab3 file.
- The textures for box were download from: www.textures.com