COSC 363: Computer Graphics

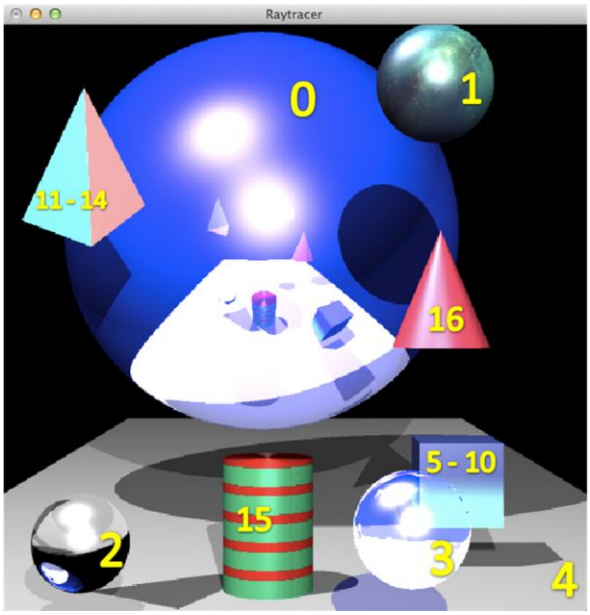
Assighment 2

Student name: Xiao Meng

Student number: 88211337

1. Description of Ray Tracer

As including all the functions in lab7, lab8 and some other functions in lectures, my ray tracer has implements reflactions and refractions for spheres, planes, cylinder, cone and so on. In my sence, there are three shperes, a grey plane as a floor to show the shadows of other objects, a box with six textured planes, a tetrahedron which was made by four triangle planes, a cylinder with front side, back side and two caps, and a cone. The whole scene is shown as the picture below, yellow numbers are the indexes of these objects in scenceObjects list.



2. Features of Basic Requirement

2.1 Lighting:

The scence has two light sources. Objects' color value influenced by both diffuse reflection and specular reflection generated by these two light sources.

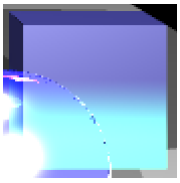
2.2 Shadows:

Object generated two shadow rays towards to each light sources and traced them to check if it is in the shadow.

2.3 Reflections:

Two ojbects have reflective surface: Sphere 0 and 3.

2.4 Box & Texture planar surface:



There is a box made up of six planes textured by simple gradual color graphes which I draw with photoshop.

3. Features of Extensions

3.1 Extra objects:

- Tetrahedron:

The tetrahedron was constructed of four triangles, the Class Triangles is modified the functions : isInside(), intersect(), normal() from the Class Plane by reducing the amount of points to three. The form of Tetrahedron is as below:

```
Triangle(glm::vec3 pa, glm::vec3 pb, glm::vec3 pc, glm::vec3 col)
```

- Cylinder:

The Class Cylinder return distance from position of eye to the point of intersection and normal vector of cylinder on that point. With the equation of cylinder and ray, we can get the intersection equation.

Cylinder Equations: $(x-x_c)^2+(z-z_c)^2=R^2,\quad 0\le(y-y_c)\le h$

Ray Equations: $x=x_0+d_x t;\quad y=y_0+d_y t;\quad z=z_0+d_z t;$

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

Xc, Yc, Zc represent center vertex of the bottom of cylinder. dx, dy, dz represent the direction of primary ray. Xo, Yo, Zo represent position of eye. h, r mean the height and radius of cylinder.

Intersection Equation: $t^2\left(d_x^2+d_z^2\right)+2t\{d_x(x_0-x_c)+d_z(z_0-z_c)\}+\{(x_0-x_c)^2+(z_0-z_c)^2-R^2\}=0.$ Then I set variables to find the roots t1 and t2 of this equations:

```
float a = pow(dir.x, 2) + pow(dir.z, 2);
float b = 2 * (dir.x * vx + dir.z * vz);
float c = pow(vx, 2) + pow(vz, 2) - pow(radius, 2);
float delta = pow(b, 2) - 4 * a * c;
```

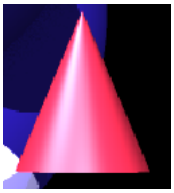
with the equations I get the value of t which means the distance. Using the distance and some limitation conditions in the lecture 9a, the function can return the distance to every side of a cylinder.

```
t = fmin(t1, t2);
float s = fmax(t1, t2);

float m1 = posn.y + dir.y * t;  // the y value of closer point of intersection
float m2 = posn.y + dir.y * s;  // the y value of further point of intersection

float cap_t1 = (center.y + height - posn.y) / dir.y;
float cap_t2 = (center.y - posn.y) / dir.y;

if (center.y < m1 && m1 < (height + center.y)) return t;
else if (center.y < m2 && m2 < (height + center.y) && m1 > (center.y + height)) return cap_t1;
else if (center.y < m2 && m2 < (height + center.y) && m1 < center.y) return cap_t2;
else return -1.0;
```

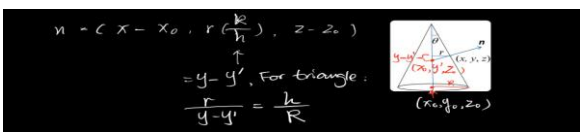


- Cone:

The returned value of Class Cone is the same as other classes of objects.

Cone Equations: $(X-X_c)^2+(Z-Z_c)^2=r^2,\quad r=\left(\frac{R}{h}\right)(h-y+y_c)$

Normal Vector:



with the same ray equations, I get the intersection eqution as below,

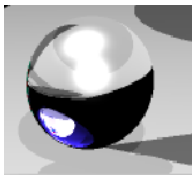
Intersection Equation:

As the same methods in Class Cylinder, the function return the distance and normal vector of cone.

3.2 Multiple light source

I set two light source at , for them I generated two shadow rays which as mentioned in the basic part. it can be seen in the main sence picture where there are two light pots on shpere and each object has two shadows.

3.3 Refraction object

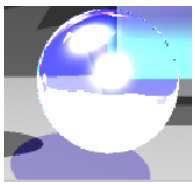


As the lecture shown, I set sphere 2 as a glass ball which rate of refraction is 1 / 1.5 with air index is 1 and glass index of refraction is 1.5. The scene in the glass ball is the opposite impression of reality. I set shadow color value of refraction ball with intensity of 40% of normal shadow color value for both two light sources. The implement code is as below and the same as light 2.

```
if (ray.xindex == 2 && step < MAX_STEPS)
{
    float eta = 1/1.5;  //refracted to a glass sphere
    glm::vec3 g = glm::refract(ray.dir, n, eta);
    Ray refrRay1(ray.xpt, g);
    refrRay1.closestPt(sceneObjects);
    glm::vec3 m = sceneObjects[refrRay1.xindex]->normal(refrRay1.xpt);
    glm::vec3 h = glm::refract(g, -m, 1.0f/eta);
    Ray refrRay2(refrRay1.xpt, h);
    glm::vec3 refractedCol = trace(refrRay2, step + 1);
    sumCol = 1.0f * refractedCol + spec1 * glm::vec3(1.0f) + spec2 * glm::vec3(1.0f);
}
```

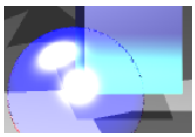
```
if (shadow1.xindex == 2 && shadow1.xdist < lightDist1)
col1 = glm::vec3(0.4f) * materialCol;
```

3.4 Transparent object



The sphere 3 is a transparent light blue object. Almost the same as refraction, transparent just need to be set the index of refration as the same value to make rate of refraction is 1. Hense, when light tracing through the object, the direction of light would not change. So the refraction part would not need function glm::refract any more, just get the point of intersection between the retracted ray inside of sphere and surface, and generate the ejection ray and traced it. Also the shadow of this ball is 40% intensity of normal value with light blue for both light sources.

At the beginning I set transparent ball, I recognise There are some small black dots on the edge of the sphere. I thought the reason might be that, if with the same implement of refraction when incident ray would reach the edge of sphere and generate a refraction ray1, but the ray 1 would not reach any point of intersection of sphere any more, it will return the color value of background. So I implement a limited condition for that, if there refraction ray1 could not find that point of intersection, it trace the ray1 as an ejection ray. Also I set the sphere as an reflective surface, that will solve the black dots better. The implement code and transparent shpere with black dots is shown as below.

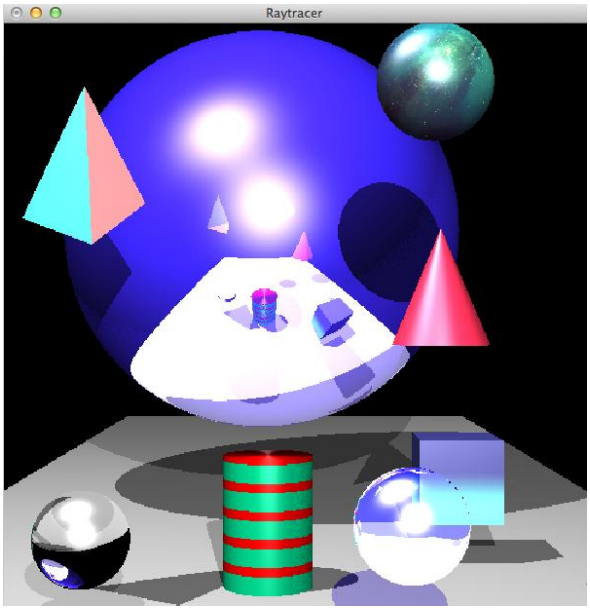


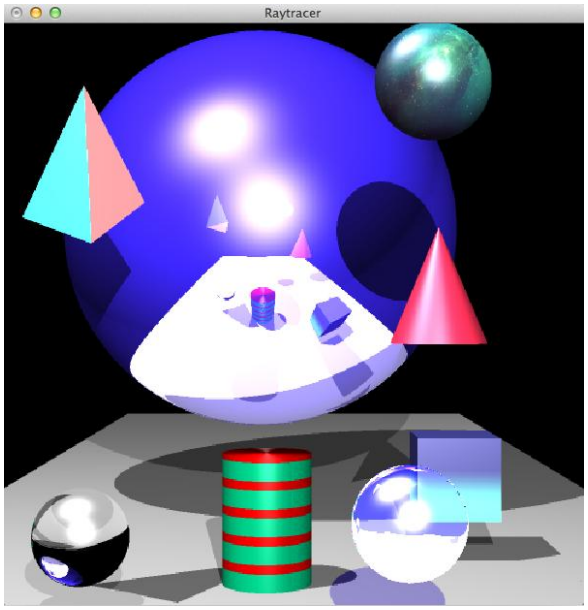
```
//-- Transparency objects
if (ray.xindex == 3 && step < MAX_STEPS)
{
    Ray refrRay1(ray.xpt, ray.dir);
    refrRay1.closestPt(sceneObjects);

    Ray refrRay2(refrRay1.xpt, ray.dir);
    glm::vec3 refractedCol;
    if (refrRay1.xdist < 0.2)
    refractedCol = trace(refrRay1, step + 1);
    else
    refractedCol = trace(refrRay2, step + 1);

    sumCol += refractedCol + spec1 * glm::vec3(1.0f) + spec2 * glm::vec3(1.0f);
}
```

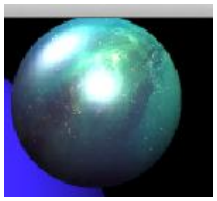
3.5 Anti-aliasing





As the pictures shown, the obvious result of anti-aliasing can be seen at the shadows on the floor plane. The assignment used 500 pixels to display, I thought if the number of pixels increases, the effect of anti aliasing will be more obvious. I implemented anti-aliasing by dividing one pixel cell into four parts and generate primary ray separately. I also thought if it divided into eight parts, the effect of anti-aliasing would be much better.

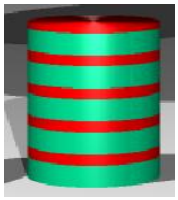
3.6 An non-planar object textured using an image




The sphere 1 is a textured sphere with an image downloaded on the website. with reference on website, I implemented formulation as below:

```
float texcoords_sphere = n.x / (2 * M_PI) + 0.5;
float texcoordt_sphere = n.y / (2 * M_PI) + 0.5;
```

3.7 An non-planar object textured using a procedural pattern

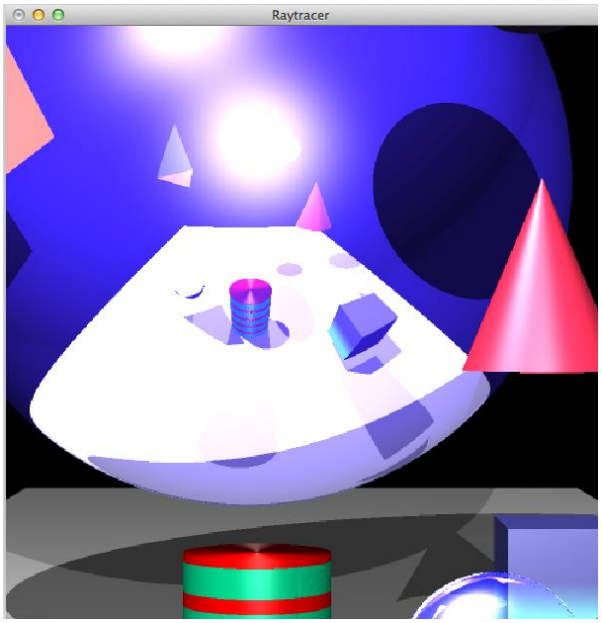


```
glm::vec3 materialCol = sceneObjects[ray.xindex]->getColor(); //else return object's colour
glm::vec3 proc_materialCol(0.7, 0., 0.);

if (ray.xindex == 15)  //procudural pattern texture with cylinder
{
   if (int(ray.xpt.y) % 3 == 0) materialCol = proc_materialCol;
}
```

After implementing normal materialCol in a local illumination model, I set a procedural material color value as red for cylinder. As the picture shown, after every 3 units of the height of cylinder, there would be a red zone with 1 unit of height. The height of this cylinder is 14.5, so the cap is also in a red zone.

3.8 Camera motion



I also implemented a keyboard event for making the camera moving forward or backward 25 units by pressing down and up keys. As the very slow speed of my computer, I did not try moving camera left or right by change the position eye.

```
void special(int key, int x, int y)
{
   if (key == GLUT_KEY_UP) EDIST += 25;
   else if (key == GLUT_KEY_DOWN) EDIST += 25;

   glutPostRedisplay();
}
```

4. Resources and References

Source code:

Lecture 9 and Lab 7 and Lab 8

References:

Texture for Sphere Map

https://blog.csdn.net/huangzhipeng/article/details/7957963