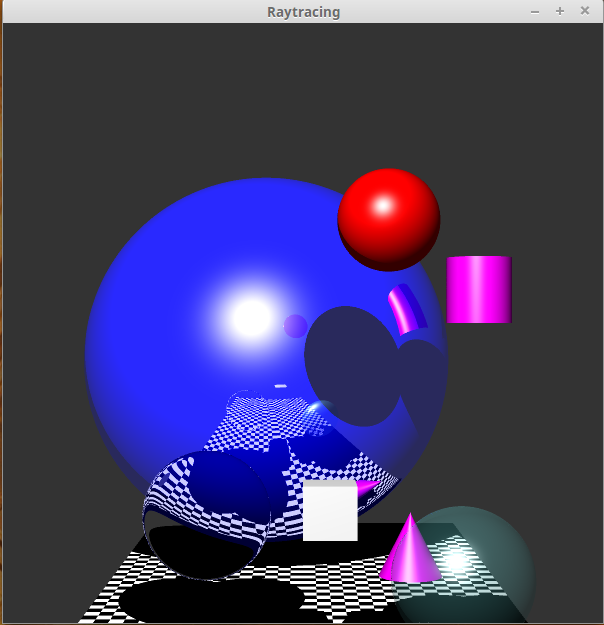
COSC 363

Assignment 2

Jingwei Li

39923031



I did this assignment based on the lab7 and lab8. It has more features than labs and it is more complex. There are many geometric objects such as spheres, cube, cylinder and cone, they are all implemented by ray tracer. Also, the program generates shadows and reflections. The ray tracing algorithm makes the image more close to realism but it is really expensive for GPU cost.

I have done all minimum requirements and some extra features. It may take about 1 minute to run when the anti-aliasing is on.

**Extensions**

**Cylinder:**

the equation of cylinder is :

it defined the shape of a cylinder.

The normal vector at (x, y, z) is :

but it is un-normalized, so I call normalise() function after applied it.

The equation of is used for solving quadratic formula. It is used for intersected parts.

The intersection equation of a cylinder is on lecture note which is :

so I use it to get the a b and c values:

according to that, we got :

the first condition is for base and the last condition is to control the height.

And I add some if statements to consider the situation that the light is towards cylinder's back.

if (t1 > 0){

if (t1 < t2)

t = t1;

else{

if (t2 > 0)

t = t2;

else t = t1;

}

}

else{

if (t2 > 0)

t = t1;

else t = -1;

}

**Cone:**

It is similarly to the cylinder, but the equation is :

and the normalized normal vector is :

n=(sinα cosΘ, sinΘ, cosα cosΘ)

where α=tan^[-1] ((x-xc)/(z-zc)

so the values of a b and c are :

**Multiple lights and shadows :**

The light source vector is obtained as

Vector l2 = light2 -q.point;

Normalise this vector, and compute the dot product l•n

:

l2.normalise();

lDotn2 = l2.dot(n);

The material colour is already made available in the function:

Color col = sceneObjects[q.index]->getColor();

If the dot product is ≤ 0,

compute Phong lighting using only the ambient light

(background colour) and return this colour:

return col.phongLight(backgroundCol, 0.0, 0.0);

Otherwise compute the specular term as follows, by first calculating the reflection vector r, normalize it, and then computing the r•vterm:

Vector r2 = ((n \* 2) \* lDotn2) -l2;

r2.normalise();

Vector v(-dir.x, -dir.y, -dir.z);

float rDotv2 = r2.dot(v);

float spec2;

if(rDotv2< 0) spec2= 0.0;elsespec= pow(rDotv2, 10);

Now we can compute the sum of ambient, diffuse and specular components by

calling the function

col.phongLight(backgroundCol, lDotn2, spec2);

Return the above colour

**Refraction:**

the refraction ray is the ray inside the object. So the colors need to be combined.

From the lecture note, I got equation :

andare the refraction coefficients. Like in air is 1 and in water is 1.33

using these equations, I can get the ray vector when ray get in an object or out an object. Careful, need consider the direction of normal, n = n \* (-1).

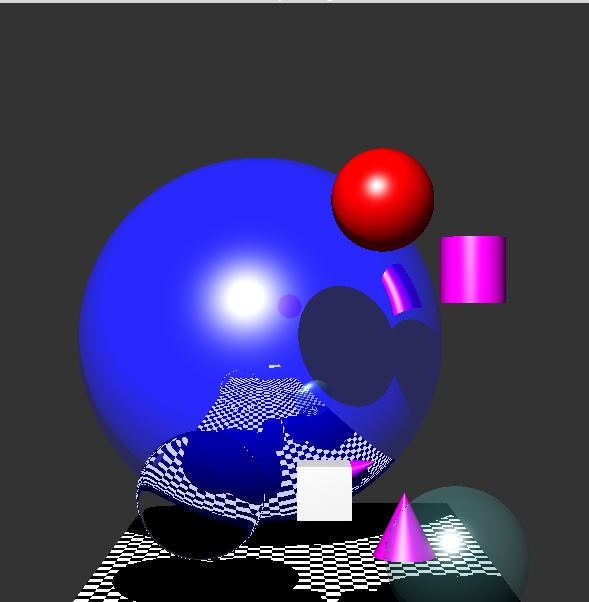
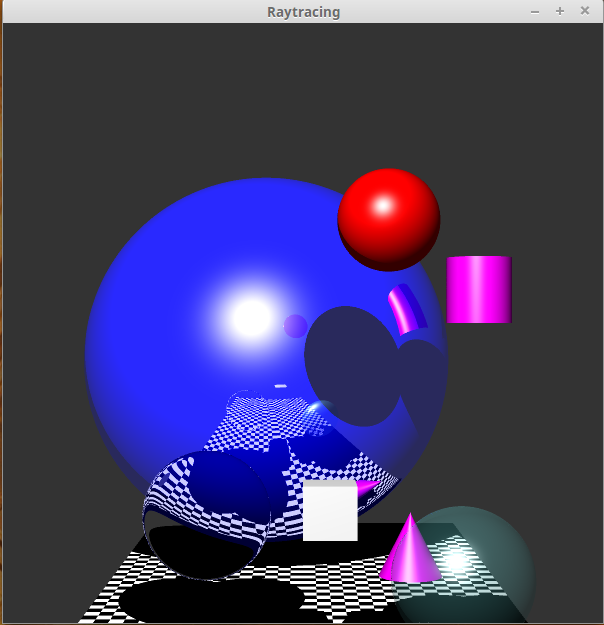
**Transparent object:**

It is almost the same as the refraction. Just change both and to 1. And I set a color for the transparent object otherwise it is really hard to find it. Then just combine the colors.

**Anti-aliasing :**

I use supersampling to do the anti-aliasing. Just divide pixel into 2\*2 grid, and trace 5 rays, 1 at centre and 4 at the grid's centre. My idea is if the colors are similar then get the average otherwise recursively subdivide the grids, keeping going until meet condition that colors are similar or reaches the limited steps.

But anti-aliasing do cost a lots of time on labs computers, so I comment the recursion part. The effect is still obviously. The jaggies and patterns are much better. Everything get smoother.



**Anti-aliasing off on**

**the shadows and the edge of the cylinder and the refracted sphere are obviously much better.**

**References :**

lecture notes, Cosc363, University of Canterbury, 2016

lab materials, Cosc363, University of Canterbury, 2016