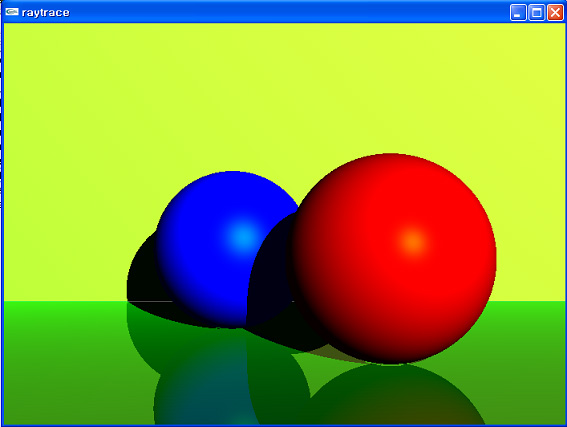
## Computer Graphics Final Homework: A Simple Ray Tracer



### Instructions

In this project, you will work on a simple ray tracer.

1. You are asked to implement all of the following features (refer to sample as a reference):

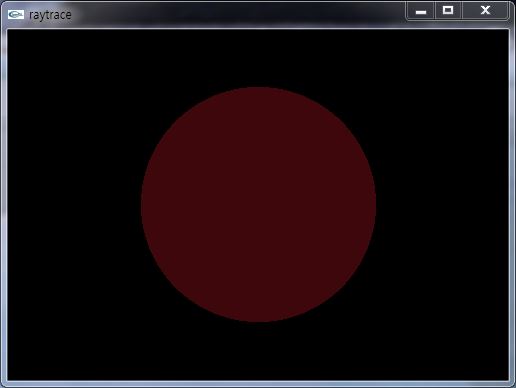
* Add a (point or directional) light source and diffuse/specular components to the lighting model.
* Add a new kind of object - your choice - to the scene
* Add reflection
* Cast shadow rays to make shadows

1. If you implement one of the following additional features, you get extra credits.

* Add refraction
* Add texture mapping
* Ray tracing acceleration techniques such as space subdivision

### Starting off

Here are the files to make a primitive raytracer that are zipped together under ~/codes:

* SimpleRayTracer.cpp
* raytrace.cpp
* raytrace.h
* raytraceData.h
* lowlevel.cpp
* lowlevel.h
* tracer.cpp
* tracer.h
* shader.cpp
* shader.h
* framebuffer.fs
* framebuffer.vs

When you compile it similarly like other OpenGL homeworks, you should get an executable that draws a picture of a dark red circle on a black background: a ray-traced picture of a red sphere in dim ambient light.

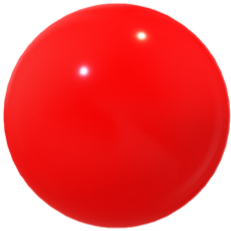
The simple ray tracer casts a ray through every pixel in the image plane. The image plane lies in the plane z=-1, and the viewpoint is at the origin. The sphere you see is centered at (0,0,-2) and has radius 1.

*y*

*z*



*x*



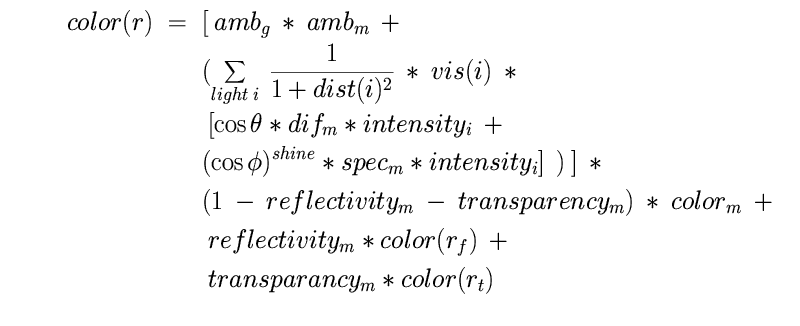
(0,0,-2)

*z=-1*

The raytracer uses OpenGL only to put up a window; it draws an array of pixels into the window, and YOU are responsible for coloring the pixels one by one to make the image. The function *drawScene* writes single pixels to the array.

### Lighting model

Here is the simple lighting model you are required to implement.



Here's what the symbols mean. Let p be the point on a surface seen by ray r.

* ambg is the general ambient lighting level (scalar)
* ambm is the ambient reflectance of the surface (scalar)
* vis(i) is zero if light source i cannot be seen from p, 1 otherwise
* θ is the angle between the ray from p to light source i and the surface normal
* difm is the diffuse reflectance of the surface (scalar)
* intensityi is the intensity of the light source (scalar)
* colorm is the color of the material (an RGB value)
* φ is the angle between r and the reflection of the ray from light source i at p
* specm is the specular reflectance of the surface (scalar)
* shine controls how sharp the specular highlight is (usually > 1)
* transparencym is the transparency of the surface (should be between 0 and 1)
* rt is the transmitted ray through the surface (a ray)
* rf is the refracted ray by the surface (a ray)
* reflectivitym is the reflectivity of the surface (should be between 0 and 1)

This required lighting model handles shadows, diffuse reflectance, a local specular reflectance term, and refraction.

You want each of the R,G and B components of the final pixel color to come out between zero and one, so all the material parameters above should be chosen between zero and one as well. When calculating cosines, set any value less than zero to be zero. Otherwise, for instance, strange things show up near the boundary of the sphere. The following formulae can be handy to compute rt and rf:

As usual, you should take it step by step.

### Step 1

Define a light source and get the diffuse/specular reflectance component of the lighting model working.

It's easier to use a directional light source (position is a point at infinity); extend it to handling point light sources at a finite point.

**DO NOT** define the light source in OpenGL; since you're not using OpenGL to draw any polygons, this will not have any effect! Make a data structure of your own to represent the light source, with a position and intensity. Then use that information to compute the diffuse component of the lighting at each pixel according to the lighting model equation above.

### Step 2

Make some more objects and add them to your scene. At least one new kind of object is required. A halfspace under the sphere is a good idea. Boxes are easier to implement than general intersections of halfspaces. Cylinders, cones and other quadric surfaces like paraboloids and hyperboloids are easy and fun. Try varying the material properties to get a nice looking scene. For every object you'll need a function to find the first intersection of a ray with the object, and a function to find the normal on the object at a point.

You may use the intersection software available at

<http://www.geometrictools.com/Source/Intersection3D.html>

You have to modify *tracer::trace()* whenever you add a new object - otherwise it doesn't test for intersections with the new object.

### Step 3

Add shadows by defining the function *Vis(i)*. You decide whether a point p on the object surface is in shadow by casting a ray from p to the light source i. If *trace* returns a point other than p, then the object lies between p and i, p is in shadow, and *Vis(i)=0*. If not, then p is visible from i, and *Vis(i)=1*.

You might run into this common bug: when p lies on the surface of an object, *trace* returns that object as the first thing hit, because of roundoff error. You can fix this by checking that the parameter *t* along the ray is actually somewhat greater than zero (define some small tolerance value) in *trace*.

### Step 4

Make one of the surfaces in the scene reflective. You find the reflection ray at a point p by reflecting the ray from p to the viewpoint (which is *(0,0,0)*) across the surface normal at p. Compute the color seen along the reflection ray by recursively calling *rayColor*. Be careful to limit the number of levels of recursion.

### Turning it in

* Source files and executable
* Image files to capture the raytraced image (the more, the better)
* Description of your project including the features that you implemented

Zip the above files into a single file and submit it to the cyber.