

**Fall 2004**  
**22C:151 Introduction to Computer Graphics**  
**Assignment 8**

**Due: Thursday November 11th at 12:01am (Wednesday at midnight)**

<b>Goal:</b> Write a basic raytracer.
---------------------------------------

**Problem 1 (5 points):** Think about your imaginary camera that is going to take a raytraced picture of your synthetic scene. You know the eye position, where you are looking at, and the up direction (just like in `gluLookAt(...)`). You also know some field of view and aspect ratio (just like in `gluPerspective(...)`). Using this information, you need to determine a ray leaving the eye that passes through each pixel in your image. Write code to compute a ray for each pixel.

*Hint: For C++ programmers, this makes a good class: a “Camera” class, with a method “generateRay(i,j)” which takes an (i,j) position on the screen and computes the ray running through the center of that pixel based on position/at/up/near/far that you specify in the constructor.*

**Problem 2 (5 points):** Now we need to determine if our rays see any objects. This means we need to intersect a ray with some objects. For this assignment, you’ll need to implement two different objects: a sphere and an infinite plane. Write code that determines if a ray intersects a sphere and write code that determines if a ray intersects an infinite plane.

*Hint: For C++ programmers, it might be a good idea to have an “Object” base class. Both spheres and infinite planes are types of objects. And all objects need to have a method to determine if a ray intersects them.*

**Problem 3 (5 points):** Now that we know if our eye sees any objects, we need to light them. To light an object, we need to have a normal. Write code that takes in an intersection point and returns the surface normal at that point. You’ll need to write 2 separate functions – one for spheres and one for planes.

*Hint: For C++ programmers, it might be a wise to add a “getNormal” method to your object class. You’ll need to be able to compute a normal for all objects.*

**Problem 4 (5 points):** Now we’re going to add simple lighting. Given an intersection position, a normal, and a light position, compute simple diffuse (or Lambertian) lighting. Remember, if  $\hat{N}$  is the unit-length surface normal at a point,  $\hat{L}$  is the unit-length direction to the light, and  $C_L$  and  $C_M$  are the colors of (respectively) the light and the object’s material, then the color at the point is:  $\max(0, (\hat{N} \cdot \hat{L}) * C_L * C_M)$ .

*Hint: For C++ programmers, you might want to define a “Material” class. Each object will have an associated material type, and “Lambertian” (or Diffuse) will be one type of material.*

**Problem 5 (10 points):** Put the pieces all together and create a very basic, simple raytracer. Render a couple example scenes, output them to PPM files, and put them on your web page. Also, I will specify a simple scene before next Tuesday (the 9th) that I’d like everyone to render. Remember, if you had trouble with PPMs, code is posted on the class web page that will output files for you.

**NOTE:** A “README” file is required in order to get full credit! Please tell us how to compile and run your program. If there are any special command-line parameters, make sure you specify them!

**NOTE:** Raytracing involves a number of small, trivial bits of code that can be cleaned up significantly by adding them to classes. If you know C++, I’d highly suggest writing the following classes: “Ray,” “Vector,” “Point,” “Camera,” “Object,” “Material,” and “Image.”