**CMPE 460 Project 2.**

**Due: March 17**

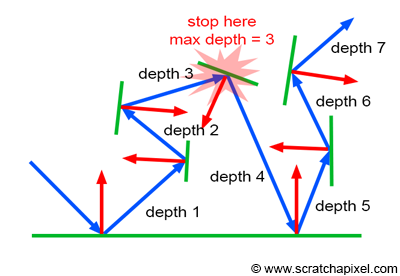
**Submission thru Moodle page**

Write a recursive ray tracer. The recursive procedure should first check depth against maxdepth and if it is greater, it should return local color. Local color consists of ambient and diffuse terms, also checking for shadows. If depth is smaller than or equal to maxdepth; it should also send reflection rays and find reflected color. There is no transmission (refraction). You may add a ground plane that is reflective; that makes it easier to have reflections.

Assume the eye point is at the origin (0,0,0); and the center of the screen is at (0,0,100).The screen extends from (-50,-50,100) to (50,50,100) and the resolution is to be 1000x1000 pixels. Assume there is only one light source and it is at (500,500,500).

For every ray which is cast through the eye and the scene, when a ray collides with an object, it should be reflected from the surface of this object and the “depth” counter should be incremented. For every collision with an object surface, two color components of the Phong shading model should be calculated for the collision point: The ambient color, and the diffuse color (no specular color to make it simpler). After the color calculation for that specific point, the ray should be reflected into the proper direction and new collisions with different objects should be checked. If “maxdepth” is reached or the reflected ray does not collide with any object, the calculated color should be printed on the current screen pixel. For each collision point, the calculated color may be stored into a container and an average color for every collision can be printed on the pixel. An example pseudo code would be like:

|  |
| --- |
| For each pixel p:  r = CalculateRay(p);  color = ambientColor;  depthCount = 0;  CastRay(r,&color,&depthCount);  finalColor = color / depthCount;  putColor(Color, finalColor);  void CastRay(Ray r, Color\* color, int\* depthCount)  {  o <- Check every object o, get the closest object o with a collision with r;  if (o != null and \*depthCount < maxDepth)  {  \*depthCount ++;  r\_new = CalculateReflectedRay(r,o);  diffuseColor = CalculateDiffuseColor(r,o);  // View Point is known, so view vector is also known  ~~specularColor = CalculateSpecularColor(r,o,view\_vector);~~  \*color += (o.diffuseComponent\*diffuseColor ~~+ o.specularComponent\*specularColor~~ + o.ambientComponent\*ambientColor);  CastRay(r\_new,color,depthCount);  }    } |



An example reflection scheme

Inputs to the program are:

number of spheres N

Diffuse and Ambient colors of the spheres i, i=1,..,N

Position (x,y,z) of sphere i, i=1,..,N

Radius of sphere i, i=1,..,N

(Note that nearest object should have z>200 and far objects should have z<1000. Choose x & y so that spheres are visible (x,y<z/2))

A fully reflective ground plane P

Ambient Color of the Scene

Output: 1000x1000 color image file representing screen.

Test your program with the following parameters, and with similar parameters of your choice:

number of spheres 2

color of sphere 1: (R,G,B)=(255,0,0);

color of sphere 2=(0,255,0);

Position (x,y,z) of sphere 1: (50,50,300)

Radius of sphere 1 = 20

Position (x,y,z) of sphere 2: (100,100,600)

Radius of sphere 2 = 60

Report: Your report should explain the method used, and display outputs with several parameter settings.

What to submit: You should submit your source code; and exe together with your report.