1 Scene Language

Scenes are described using xml files. The scene schema has the following constraints:

- Scene files have one top level node called a scene node
- a scene node contains 0 or more light nodes, 0 or more geometry nodes, exactly 1 camera node, exactly 1 settings node
- a light object must contain a color node, an intensity node, a position node and a look at node
- a geometry node contains one primitive node specifying the shape of the object, 1 translation node and a material node, a material node specifies a diffuse color, reflactance, transparency, and refraction values.
- a camera node has a position node a look at node, an up node, a focal length node, a width node, and a height node.
- a settings node contains a background color node, radiosity node (enabling or disabling radiosity) and radiosity accuracy node.

Scene parsing leverages libxml2 which made the process pretty painless and reads the scene file into a Scene t struct which contains an array of pointers for geometry and light objects (as well as how many of each we have) a pointer to a camera object a pointer to a settings object.

2 Primitives

My raytracer is pretty limited in termis of primitives right now, it only parses and stores boxes, tori, spheres and planes. And only planes and spheres have intersection functions written which are needed to really do anything. On the other hand primitives are really easy to add due to the geometry abstraction.

3 Intersection

The heart of the program is of course the intersection functionality. I've implemented an Intersect Scene which takes a ray (the same structure we used for the other projects) and a Scene t * and returns the first thing the ray hits as an Intersection t. An Intersection t struct contains, a pointer to the geometry object it hit, the normal of the intersection the point in world space at which the intersection occurred, the parameter of the ray at which the intersection occurred, and u and v parameters for the textures of the objects (used for radiosity).

4 Tracing

Using the intersection we can get a color value for each ray we shoot, then we can use that to calculate diffuse and specular intensity using the ideas we've used for fragment shading (as well as shadows). Then we can look at the geometry's material and if it has reflectance we reflect our original ray over the normal of the intersection, shoot again and blend the colors. Similarly if we have transparency we can do a snell's law refraction and reshoot the ray.

5 Radiosity Textures: the good stuff

If a scene enables the radiosity option, then each geometry objects has a Radiosity Textures record the illumination on the surface of the geometry using forward tracing. That is rays start at the lights and then we shoot them in to the scene and record what they hit, the light rays also cascade off of geometry. This allows for more accurate light simulations. In particular it allows for things that are obscured or not directly facing a light to still be illuminated by it and allows the geometry to pick up color from the geometry around it.

The paper I based my implentation on: `http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=C492EA2FEBCDC524D296C1C8F68F9353?doi=10.1.1.84.4862&rep=rep1&type=pdf`