Tyler Paulsen
CSCI-711-01 Global Illumination
Weekly report for Week #5

Summary:
At the time of publication, an efficient global illumination algorithm was a major goal, and most models implemented a single parameterized shading model. These shading models had to use many different types of tricks -- texture modeling, heuristic modeling, etc.. -- to have a realistic image produced. Images that were of any difficult type of material were hard to replicate, or the solution was complicated and had lots of overhead.

There were three major goals to the renderman spec: have an abstract shading model based on ray optics that could be used for local and global illumination, Ability to define the interface between shading and rendering, and provide a high level language that is easy to use.

The shader language had three types: light source, surface reflectance, and atmosphere shaders. The light source shaders, calculated the color and intensity of light from a source. The surface reflectance calculates the reflection of an object's surface with incoming light. The atmosphere shaders calculated the scattering light off of objects. The light source and surface reflectance shaders do not need to be aware of each other in order to do their calculations, The surface shader can be bound to any geometric primitive, and does not need to know anything about the incoming light while the light source shader only calculated the light emitted, and does not need take into consideration whether it is bound to a primitive or not. Local illumination can be done by just assuming the light is scattered only once throughout the scene.

The shader language is much like opengl or C syntax we have learned in class. To make the spec as abstract as possible, an object can have any number of materials added to it.

There is six steps in the pipeline to render a renderman scene: compilation, loading, instancing, binding, elaboration, and evaluation. Compilation uses all of the shading source files to create a program. Loading creates a type of reference for the shader file to be called later. Instancing creates a cache of shaders in memory to be referenced by objects. Binding is when the geometric primitive is bounded to a shader. Elaboration done to make the Binding step more efficient by moving data around to places where it can be accessed quicker. Evaluation is when the scene is created.

Questions:
What if you wanted to make your ray tracer conform to the RenderMan spec?
The first two points -- abstractions and an interface between the renderer and shader -- of the RenderMan spec is easily conformed to by object oriented principles. The last point -- ease of use -- all depends on the libraries used to draw the scene, and the language.

What would you have to change with regards to way your code is structured?
The renderman spec allowed an object to have multiple materials associated with it. Currently in my program, a material is represented as an illumination model, and it would be neat to incorporate a way to have multiple illumination models for one object.

Tyler Paulsen
CSCI-711-01 Global Illumination
Weekly report for Week #5

How would you have to change the overall architecture of your ray tracer code?
Currently there is a World object that renders the scene with one method by calling on all the other objects in the scene -- lights, geometric objects. The main program sets up the scene, and the renderer is used by the world to draw. The world object is instantiated by a main method, but I would like to have the world object somehow be tied to the renderer with the ability to modify the scene without having the current abstraction of Main->world->renderer.