

Comp236 – Spring 2005
Programming Assignment #4
Due 4/13/05 (before 11:59:59pm)

Objective: In this project you will extend the simple ray-tracing program given in class using the code from lectures 17 and 18 as a starting point. Unlike previous programming assignments you must use Python to implement this one, but you will be given the example code used in class.

Policies: Everyone must turn in their own assignment. You can collaborate with others, but any work that you turn in should be your own.

Minimal requirements (worth 80%):

- 1) Extend the surface shader to handle refraction. You will need to modify the supplied Sphere “Renderable” to properly handle refraction (Reminder: the given intersect() method assumes that the ray originates outside of the sphere. This is not always the case for refraction).
- 2) Implement a “Triangle” Renderable. You can use either of the methods discussed in lecture 18, or your own variant. Assume that the outward facing side of the triangle is the one where its vertices appear in a counterclockwise order. The given index-of-refraction value represents the ratio of the speed of light of the inside material relative to the outside material.
- 3) Implement any other Renderable primitive of your own choice. For example, a bicubic patch, a cylinder, or a cone. You should allow your primitives to be specified in general positions and orientations.

Extras (worth varying amounts):

- 1) Add an acceleration technique and demonstrate its performance improvement. Here are some possibilities in order of increasing difficulty:
 - a. Spherical bounding volumes
 - b. Axis-aligned bounding volumes
 - c. A 3-D grid or some other spatial partitioning approach
 - d. A light buffer
- 2) Add a randomized sampling method for enhanced rendering. Once more, here is a non-exhaustive list in order of increasing difficulty:
 - a. Antialiasing
 - b. Depth of field effects
 - c. Area light sources (jitter the shadow rays towards an area or volumetric light source)
 - d. Improved global illumination via randomized sampling of reflected and refracted directions
- 3) Add various interpolants to your triangle primitive
 - a. Add texture mapping. The most straight forward way to accomplish this is to add label-texture indices to each vertex, and interpolate them over the triangle’s interior
 - b. Add variable material properties for each vertex
 - c. Add normal interpolation (as in Phong shading) to improve the shaded appearance of your triangles
- 4) Make your raytracer as few lines as possible, and still work