CS6630 Assignment 3

Instructor: Steve Marschner

Due 19 October 2009

For this homework you'll implement a volumetric ray tracer that works for homogeneous and inhomogeneous media. You can work from the provided Java framework (available on CMS), or if you prefer you may implement a completely separate program in C++ or Java.

The input to your renderer comes in a simple XML format that supports triangle mesh surfaces and volumes of scattering medium that are bounded by boxes. The framework implements a basic Monte Carlo ray tracer that does direct illumination on Lambertian surfaces but not much else. It also contains code to read sampled volume data from raw data files like the example datasets provided with the assignment.

Your program just needs to add support for computing single scattering in volumes, paralleling the ray tracer's handling of direct illumination on surfaces. You should do this using the Monte Carlo approach outlined in lecture and in the lecture notes, including the ability to efficiently take several illumination samples along a path. Volumes and objects should cast shadows on one another, and volumes should self-shadow, making them appear like solid surfaces when there are sharply delineated regions of high density. Your program should support isotropic and Henyey-Greenstein phase functions.

Here are the cases you should hand in the output of your program for:

1. The example Cornell-box scenes available on the web page. They provide a basic visual test of correct behavior, and they include a number of heterogeneous datasets from CT scans. It's true, it doesn't make a lot of sense to render CT data with a physically based renderer, since the densities are not optical densities. But nonetheless it makes good test data.

2. Set up a scene to let you test the renderer against the plane-parallel analytic result of Homework 2 (with refractive index 1.0). This will provide much clearer evidence of correctness. (The `RayTracer` class includes a feature that will write out the numeric pixel value of a one-pixel image, to assist in comparing against known results.)

3. Set up test scenes to illustrate the effects of changing the phase function. In particular, try pointing the camera at a small source through a layer of medium, and try varying the phase function parameter to see the effects on the appearance of solid surfaces.

I will likely add more test cases as time goes on, and you're encouraged to share test inputs with one another (and with me). Be creative with your scenes! Developing renderers is much more fun when it results in cool pictures!

In addition to the basic requirements, there are special extras for students with more background:

4. If you have implemented a volume ray tracer before, you should also implement a hierarchical scheme for storing the volume, with adaptive ray marching for greater efficiency.

5. If you have implemented a Monte Carlo renderer before, you should implement multiple importance sampling to sample using both the phase function and the luminaires.

I can discuss these extra requirements on an individual basis in more detail.