1 Introduction

In this assignment, you will add photon mapping support to a Monte Carlo raytracer. Photon mapping is a global illumination technique that separates scene geometry from radiance calculation. The algorithm is completed in two passes. First, radiance from the light, split into discrete chunks called “photons,” is transported through the scene, and is saved in a 3D spatial acceleration data structure called a photon map. Second, the renderer queries the photon map for nearby photons, and uses them to compute an estimate for indirect illumination.

Photon mapping can be integrated with just about any rendering algorithm. When properly tuned, photon maps produce results that look visually similar to techniques like path tracing, without requiring as much of a wait. However, they do not converge to an exact solution, and will look wrong if not tuned properly.

2 Requirements

Your final image must support full global illumination, with caustics from reflection/refraction and color bleeding from diffuse interreflection. Specifically, you must have

- Recursive Monte Carlo raytracing, with support for reflection, refraction and soft shadows (already provided in support code)

- Photon map generation, by tracing photons through the scene using Monte Carlo techniques

- Final radiance calculation combining Monte Carlo raytracing and photon map estimate

To run the demo implementation of photon, type photon_demo in a terminal. Your results should match the demo when the same rendering parameters and scene file are used. [Caveat: some differences in materials such as the Cornell Box Sphere scene are to be expected since the demo was compiled with G3D9, which used a different BRDF implementation. This should manifest in noise. If your entire scene is a shade darker than the demo - you should double check your code!]

3 Getting Started

Support code is available in /course/cs2240/asgn/photon.

As previously mentioned, the support code already implements a Monte Carlo raytracer. You will implement the scatter() method, which shoots a photon into the scene, and diffuse(), which uses the photon map to estimate the radiance due to diffuse inter-object reflection (hence the name). Short hints have been left as comments in the support code. We also suggest reading and understanding trace(), the recursive raytracing function, as well as the subroutines it calls directly.

Rendering parameters used by the support code are provided as constants in app.h. The default parameters were chosen to balance between rendering speed and picture quality. You can get better-looking pictures by increasing the photon count and/or increasing the number of direct lighting samples taken per pixel. However, if you do this you'll be waiting a long time to get your picture!

By default, the support code shows you your photon map as it is built, and then shows the final rendering as the raytracer runs. To view just the photon map and the scene wireframe, press the 1 key on your keyboard; to view just the rendering, press 2.

As in the pathtracing assignment, we recommend referring to the “Rendering in Practice” chapter of the course text (Ch. 32). The example code given in the book uses G3D 8.x, whereas the support code uses G3D 10.x, but the concepts carry over. We also expect reading Henrik Wann Jensin’s paper on photon mapping (http://cs.brown.edu/courses/cs224/papers/photon_mapping.pdf) to be informative.

You may find the G3D10 Surfel::scatter() method useful in this assignment.

4 Extra Credit

For extra credit, you may consider implementing any of the following effects:

- Tone Mapping
- Anisotropic Reflection
- Participating Media
- Subsurface Scattering
- Shadow Photons
- Irradiance Caching
- Image-Space Photon Mapping
- Something else! (But check with a TA first)
5 Handing In

Hand in your assignment using the command `cs224_handin photon`. Remember to document design decisions, known bugs and other useful tidbits in your README. Do not hard-code any paths in your program!