CS667 Homework 4
(due 15 October)

For this homework you will implement direct illumination calculations in a Monte Carlo ray tracer written in Java. You can download the code from CMS. Some further details of what you're expected to implement are included in the comments of the classes mentioned below.

The provided code is a ray tracer that supports triangles, triangle meshes, and spheres, with rudimentary box-tree acceleration. It supports only Lambertian reflectors and emitters.

1. Implement the Microfacet material based on Walter et al. [1]. The only major wrinkle is that the Microfacet material has a diffuse component and a specular component, whereas the paper describes a two-sided model with no provision for diffuse reflection. Your BRDF should simply be the sum of a (colored) diffuse component and the (uncolored) microfacet BRDF. You should sample it by randomly choosing to sample either the specular or the diffuse component.

The renderer is implemented in an architecture that has Renderers that use different algorithms for computing ray radiance. The only Renderer (for now) is DirectOnlyRenderer, which computes direct illumination by calling a DirectIlluminator. The only working DirectIlluminator is ProjSolidAngleIlluminator, which does Monte Carlo integration over the incident hemisphere using uniform sampling with respect to projected solid angle.

This works fine, but it is terribly inefficient, requiring thousands of samples per pixel to get good results even with Lambertian materials and friendly illumination setups. Your job is to implement three other sampling strategies for direct illumination.

2. Implement LuminairesIlluminator, which does Monte Carlo integration over the area of the luminaires. This includes implementing area sampling for the different types of geometry.

3. Implement BRDFIlluminator, which does Monte Carlo integration over the incident hemisphere using importance sampling according to the BRDF. This includes implementing importance sampling for each material.

4. Implement MultipleIlluminator, which does multiple importance sampling for the luminaries and BRDF according to Veach and Guibas [2] using the balance heuristic. To take this a little farther, implement the power heuristic as well.

This last part requires little new code once the previous two parts are working, but some subtleties arise because of the need to unify the integration domains.

Implementation

The Microfacet class contains a main() method that will help you test your sampling code. It produces four images of the hemisphere, showing: (a) the BRDF value, (b) the reported probability density, (c) the actual probability density (computed using a histogram of many generated samples), and (d) the ratio between (a) and (b). If your sampling code is correct, then (b) and (c) should match exactly (save for some noise) and (d) should very pretty smoothly without a diffuse component.

For testing the illuminators, I'm providing a few scenes for which analytical solutions are available (for a one-pixel image, the provided outer loop will print the pixel value to the console). With enough samples you should see all your methods converging to the correct numbers.

You might find it easier to implement the illuminators first, since they can be tested with just diffuse materials.

I am also providing a set of other test scenes, with the output from my implementation, on the web page for this assignment. One of the scenes, *plates.xml*, resembles Veach & Guibas's Figure 2 test scene, and with multiple importance sampling you should be able to get nice results for all the sources reflected in all the plates.

References

[1] Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. "Microfacet Models for Refraction through Rough Surfaces." In proceedings of *Eurographics Symposium on Rendering 2007*.

[2] Eric Veach and Leonidas J. Guibas. "Optimally Combining Sampling Techniques for Monte Carlo Rendering." In proceedings of *SIGGRAPH 95*.