Assignment 2: Acceleration Structures and Direct Illumination from Area Lights

Due date: April 3rd, 2014, 12:00

The goal of this assignment is to add an acceleration structure based on binary space partitioning trees (BSP trees) to your ray tracer. In addition, you will add illumination from area light sources. You should extend your code from the previous assignment to implement this project. **You must turn in your work through Ilias**.

Required Features

Basic BSP Tree Construction (20 points)

In the first step, you should implement an algorithm to construct a BSP tree hierarchy. We suggest that you add a class `BSPAccelerator` and a class `BSPNode` to your project as discussed in class. `BSPAccelerator` should have a method `construct` that takes an aggregate object as its input and constructs a hierarchy based on the objects in the aggregate. Usually the aggregate will be a triangle mesh. Basic BSP tree construction splits BSP nodes in their geometric middle until each node includes fewer objects (e.g., triangles) than a user specified number, or the tree reaches a maximum depth. The PBRT book reports that a maximum depth of $8 + 1.3 * log(n)$ is reasonable, where n is the number of objects in the tree.

During BSP tree construction, you will have to test for each object whether it intersects the bounding box of the current BSP node. To do this, each of your objects should implement a method that returns its bounding box. Use axis aligned bounding boxes for simplicity. A method to test whether two axis aligned bounding boxes intersect is straightforward to implement: two axis aligned boxes are guaranteed *not* to intersect if any of their extents in x, y, or z coordinates do not overlap.

Your `BSPAccelerator` class should implement the `Intersectable` interface. This way you can include the accelerator like any other object in your scene. To debug and test your implementation, first write an `interesect` method that simply traverses the tree and intersects the ray with all objects stored in the leaves. This will not accelerate the intersection test, but it will be helpful to test whether your hierarchy is built correctly.

BSP Tree Traversal (30 points)

Write an `intersect` method that traverses the BSP tree as discussed in class. Render a few different scenes and record the time used with and without BSP tree acceleration. Test different parameters for the maximum number of objects in the leaves of the BSP tree. Save the images and write down the timings for all your experiments.

Area Light Sources (15 points)

Implement an integrator that can compute illumination from rectangular area light sources. Sample the light sources as discussed in class. You can simply shoot one shadow ray for each primary ray.

You will need to experiment with the exact number of rays needed to get nice soft shadows. The number depends on the size of the area light and your scene setup. Expect to use at least a few dozens rays per pixel.

Glossy Material (15 points)

Implement a glossy material using a normalized Phong model including importance sampling, as discussed in class.

Multiple Importance Sampling (10 points)

Extend your integrator for area light sources using multiple importance sampling as discussed in class. Demonstrate the effectiveness of multiple importance sampling by showing comparisons between sampling the light sources, sampling the BRDF, and multiple importance sampling.

Deliverables (10 points)

You will demonstrate your code during the exercise session. You should prepare several test scenes that demonstrate that your code works properly. Save example images beforehand, so that you can show them to us during the exercise session. Save the timings for all your performance experiments and explain them to us during the exercise session.

Hacker's Bonus (max. 40 points)

- Implement illumination from environment maps.
- Optimize your BSP tree using the surface area heuristics. Consider the slides for details and ask us for further tips.
- Implement additional BRDF models, such as Torrance-Sparrow. Instead of Schlick's approximation for Fresnel reflection, use a more accurate approach such as described in the PBRT book (Section 9.2). An other suggestion is to implement the color shift

described in the paper (link only accessible computers in unibe.ch domain) by Cook and Torrance.

- Implement any hacker's bonus suggestion from previous assignments, which you did not do yet!

Advice

Start programming early and make a time schedule!