Week 3

The key subject this week is Monte Carlo integration, that is, to solve integrals by sampling. With sampling comes the ability to render soft shadows and indirect illumination. We will focus on soft shadows in the following exercises.

Ray Tracing

In the exercises for Week 1, the Lambertian shading for area lights was only an approximation. Monte Carlo techniques can eliminate this inaccuracy.

- Load the Cornell box (`CornellBox.obj`) and the blocks inside it (`CornellBlocks.obj`) into your ray tracer.
- Instead of the simplified area light sampler used in Week 1, implement true sampling of a random position on the surface of the area light (first sample a random triangle, then sample a random position on that triangle). Make sure that your shader for Lambertian materials takes a number of samples and estimates the value of the direct lighting integral properly using Monte Carlo integration theory. (In the `pathtrace` project of the course framework, update the `sample` function in `AreaLight.cpp` and the `shade` function in `Lambertian.cpp`.)
- As a result you should obtain an image of the Cornell box where the blocks cast soft shadows. Pick a number of samples that results in smooth soft shadows and save the image. (The framework is set up to sample area lights 4 times per pixel sample by default. See line 183 of `pathtrace.cpp`.)
- Ambient occlusion is essentially the idea that the environment (the background) is an infinitely distant ambient area light. Compute ambient occlusion by tracing rays in directions sampled on the hemisphere over each surface point. Consider the background colour to be the incident illumination if the ray is not occluded. (In the framework, implement the `shade` function in `Ambient.cpp` and the `sample_cosine_weighted` function in `sampler.h`. Note that the ambient shader is used when you press '2' on the keyboard.)
- The result of ambient occlusion is soft and realistic illumination. To get a more interesting scene, replace the Cornell blocks by the Stanford bunny (`bunny.obj`) and an elephant (`justElephant.obj`). Pick a number of samples that results in smooth illumination and save the image. (The framework is set up to trace 5 occlusion rays per pixel sample by default. See line 105 of `pathtrace.cpp`.)

Real-Time Shading

Sampling is also essential in many real-time soft shadow rendering techniques. Sampling-based ambient occlusion on the GPU [Pharr and Green 2004, see below] is, however, a brute force approach which, in our opinion, requires too many passes. Finite element methods are better suited for dynamic ambient occlusion [Bunnell 2005, see below], but this would take us too far into the realm of real-time tricks. Consequently, you are only required to have a go at real-time soft shadows due to (finite) area lights.

- Load the Cornell box (`CornellBox.obj`) and the blocks inside it (`CornellBlocks.obj`) into your real-time shading program.
- Extend your shader implementing omnidirectional shadow mapping such that it iterates over a user specified number n of extra light samples (choosing 0 extra samples should result in hard shadows). Use a texture with n positions sampled randomly on the surface of the area light. For each position `sample_pos`, find a direction `sample_dir` using

```
sample_dir = reflect(sample_pos - pos, c_dir) + c_dir;
```

where `pos` is the fragment position and `c_dir` is the direction from the center of the source (used in the normal omnidirectional shadow mapping). Find the fraction of directions that are not occluded and use this as your visibility term for soft shadows. (In the `realtime` project of the course framework, implement the `sample` function in `AreaLight.cpp` class and the fragment program in `SoftShadow.cpp`.)

- As a result you should obtain an image of the Cornell box where the blocks cast soft shadows. Pick a number of samples that results in smooth soft shadows and save the image. (The framework is set up to use 5 samples by default. See line 82 of `realtime.cpp`.)

Week 3 Deliverables

Two Cornell box images with blocks that cast soft shadows (one ray traced, one real-time shaded). One Cornell box image with bunny and elephant shaded by ambient occlusion (ray traced). Explain the number of samples used per pixel for each image. Explain the formula that finds `sample_dir`. Discuss the differences between off-line and real-time rendering. Both differences in implementation difficulty, output quality, and generality. Include frame rates for the real-time results.

Reading Material

The curriculum for Week 3 is

P Chapter 14. *Monte Carlo Integration I: Basic Concepts*.
P Section 15.6.3. *Area Lights*. (Soft shadows.)
P Section 15.6.5. *Infinite Area Lights*. (Ambient occlusion.)
P Section 16.1. *Direct Lighting*.

Alternative literature available online or uploaded to CampusNet:

- Philip Dutré. *Global Illumination Compendium*. Lecture Notes, Katholieke Universiteit Leuven, September 2003. http://www.cs.kuleuven.ac.be/~phil/GI/
- P. Shirley, C. Wang, and K. Zimmerman. Monte Carlo techniques for direct lighting calculations. *ACM Transactions on Graphics*, Vol. 15, No. 1, pp. 1–36, 1996.

Additional resources:

- Matt Pharr and Simon Green. Ambient Occlusion. In *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*, Chapter 17, Addison-Wesley, 2004. http://http.developer.nvidia.com/GPUGems/gpugems_ch17.html
- Yury Uralsky. Efficient Soft-Edged Shadows Using Pixel Shader Branching. In *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, Chapter 17, Addison-Wesley 2005. http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter17.html
- Michael Bunnell. Dynamic Ambient Occlusion and Indirect Lighting. In *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, Chapter 14, Addison-Wesley 2005. http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter14.html