Week 1

This first exercise is primarily about getting a ray tracer and an OpenGL renderer up and running such that they produces the same results for local lighting models. If you have not previously implemented your own ray tracer (including an intersection acceleration data structure) and your own OpenGL renderer, we recommend that you use the course framework available at CampusNet File Sharing. Using the course framework will probably require the least amount of work.

Geometry and Materials

Throughout the course we expect you to be able to load triangle meshes defined by Wavefront .obj-files. This includes the .mtl-files which describe the materials attached to the different triangles of the meshes.

- Familiarize yourself with the Wavefront OBJ file format and especially with the Wavefront MTL file format. Make sure you know how to create a new material (in the .mtl-file) and how to change the material attached to an object (in the .obj-file). And make sure you understand the material parameters that you can set in the MTL format. Confer the links in the section "Reading Material" below.
- Find out how you load .obj-files into your renderer. (In the course framework, you simply provide them as command line arguments to the executables. In Visual Studio, command line arguments are provided in the project properties under Debugging.)

The models that you need for the exercises are available in the `models` folder of the course framework.

Ray Tracing

The first few weeks we will concentrate on the merits of off-line versus real-time rendering.

- Load the Stanford bunny (`bunny.obj`) into your ray tracer. Preferably this should result in some simple pre-visualization of the bunny. (If you are using the course framework, use the `pathtrace` project for this exercise. Take some time to understand what you can do with the keyboard in this program. Do this by investigating the `keyboard` function in `pathtrace.cpp`.)
- Illuminate the bunny by a directional light source[1] emitting the radiance $L_e = \pi$ in the direction $\vec{\omega}_e = (-1, -1, -1)/\sqrt{3}$ (this is the default light in the framework). Make sure that you shade the bunny correctly using the Lambertian BRDF. (In the framework, you need to implement the `sample` function in `Directional.cpp` and the `shade` function in `Lambertian.cpp`.)
- Compose a new material in the file `bunny.mtl` and attach it to the bunny.
- Turn on shadow rays and use a few extra jitter samples to get a nice anti-aliased bunny. Save the resulting image (take a screenshot).
- Now load the Cornell box (`CornellBox.obj`) and the blocks inside it (`CornellBlocks.obj`). The Cornell box has an area light source. Implement a simplified area light sampler which always samples the center of the area and returns the light emitted from the entire area. (In the framework, implement the `sample` function in `AreaLight.cpp`.[2]) Again save the resulting image.

Real-Time Shading

The purpose is now to do the same using real-time shading techniques. Chances are you already did something similar in your previous graphics courses. If so, think of the following exercises as a good way of getting your real-time shader output to correspond to your ray tracing output.

[1] Technically, a directional light is an infinitely distant collimated light source of infinite area. Very theoretical idea, but, from a human perspective, it somehow resembles the sun.

[2] Hint: Look at the `AreaLight` constructor in the `realtime` project for inspiration.

- Load the Stanford bunny (`bunny.obj`) into your real-time shading program. (If you are using the course framework, use the `realtime` project for this exercise. Again take some time to understand what you can do with the keyboard in this program. Do this by investigating the `keyboard` function in `realtime.cpp`.)
- Implement a fragment shader that, using the same directional light, produces the same output as you got with your ray tracer before you turned on the shadow rays. (In the framework, you need to implement the `set_light` function in `Directional.cpp` and the fragment shader in `Lambertian.cpp`.) Save the resulting image.
- Load the Cornell box (`CornellBox.obj`) and the blocks inside it (`CornellBlocks.obj`). Make sure your Lambertian shader handles the area light correctly. (In the framework, implement the `set_light` function in `AreaLight.cpp` and improve the fragment shader in `Lambertian.cpp` if necessary.) It should produce the same output as your ray tracer except for the shadows.
- Implement omnidirectional shadow mapping for the area light in the Cornell box. (In the framework, implement the distance fragment shader used for capturing the cube map and the fragment shader that uses the cube map in `Shadow.cpp`.) Again save the resulting image.

Week 1 Deliverables

Two bunny images (one ray traced, one real-time shaded), two Cornell box images (one ray traced, one real-time shaded). 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 1 is

P Sections 1–1.2. *Photorealistic Rendering and the Ray-Tracing Algorithm*.
P Chapter 5 except 5.1.1. *Color and Radiometry*.
P Intro. to Chapter 9 and Section 9.3. *Reflection models* and *Lambertian reflection* as a special case.
P Sections 13.1–13.4 except 13.2.1–13.2.3. *Light Sources*.

Alternative literature available online or uploaded to CampusNet:

- http://en.wikipedia.org/wiki/Ray_tracing_(graphics)
- Pat Hanrahan. Rendering Concepts. In *Radiosity and Realistic Image Synthesis*, Chapter 2, Morgan Kaufmann, 1993.

Additional resources:

- The Wavefront OBJ file format specification: http://local.wasp.uwa.edu.au/~pbourke/dataformats/obj/
- The Wavefront MTL file format specification: http://local.wasp.uwa.edu.au/~pbourke/dataformats/mtl/
- The Cornell box (data and history): http://www.graphics.cornell.edu/online/box/
- Andreas Bærentzen. *Lecture Notes on Real-Time Graphics*, DTU Informatics, 2009. http://www2.imm.dtu.dk/~jab/rasterization-pipeline.pdf
- Bent D. Larsen. Direct Illumination. In *Real-Time Global Illumination by Simulating Photon Mapping*, Chapter 3, PhD Thesis, IMM-PHD-2004-130, Informatics and Mathematical Modelling, Technical University of Denmark, September 2004. http://www.imm.dtu.dk/pubdb/p.php?4115.
- Philipp S. Gerasimov. Omnidirectional Shadow Mapping. In, R. Fernando (editor), *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*, Chapter 12, Addison-Wesley, 2004. Available online at http://http.developer.nvidia.com/GPUGems/gpugems_ch12.html