Worksheet 6

If you take a close look at your ray traced results, you might notice that shadow edges and geometry edges are jagged. The purpose of this set of exercises is to remove the jaggies such that you get nice, smooth, anti-aliased edges in your renderings. In addition, we will add indirect illumination using progressive unidirectional path tracing.

Learning Objectives

- Use simple sampling techniques for anti-aliasing and indirect illumination.
- Apply stratified sampling by subdividing pixels and jittering the sample point in each sub-pixel.
- Render anti-aliased edges using ray tracing.
- Do the first step of path tracing: stochastic sub-sampling of pixels.

Anti-Aliasing

To get anti-aliased rendering results, we employ stratified sampling. This means that each pixel is subdivided into equally sized sub-pixels, and a ray is traced through a randomly sampled position within each sub-pixel.

- The ray caster in the render framework has functions for incrementing and decrementing the pixel subdivision level (`subdivs`). Describe how the pixel subdivision level is changed while the program is running. Hint: Examine the `keyboard` function in the file `RenderEngine.cpp`.
- Whenever the pixel subdivision level is changed, the function `compute_jitters` in the file `RayCaster.cpp` is called to find a jitter sampled position for each sub-pixel. Explain what the function `compute_jitters` stores in the vector array `jitter`. Explain how many sub-pixels we get for each pixel when the pixel subdivision level is `subdivs` $= s$.
- Modify the function `compute_pixel` in the file `RayCaster.cpp` such that a ray is cast through each sub-pixel. For each sub-pixel, use the vector array `jitter` to get the jittered position through which the ray should be traced. Accumulate the result from each sub-pixel and divide by the number of sub-pixels (as in path tracing).
- Render the default scene using different pixel subdivision levels. Record the render times. Answer the following questions:

  What is the relationship between the pixel subdivision level and the render time?

  What is the relationship between the pixel subdivision level and the aliasing error in the render result?

Progressive Path Tracing

Progressive rendering is useful in the sense that we do not need to start over if the initial number of samples was insufficient to get the desired image quality. With a progressive technique, we can keep improving the image until the desired quality is obtained. We can also quickly spot a problem by inspecting the rendering as it progressively improves.

- Render the Cornell box with blocks (`CornellBox.obj` and `CornellBlocks.obj`) using progressive updates until the image has nice anti-aliased edges. Press ’t’ on the keyboard to start and stop a progressive rendering. Note the number of samples per pixel and explain how the progressive approach achieves anti-aliasing.
- Enable sampling of indirect illumination by implementing sampling of a cosine-weighted hemisphere. Do this by implementing the function `sample_cosine_weighted` in the file `sampler.h`.

- Render the Cornell box with blocks including path traced indirect illumination. Do this by implementing the function `shade` in the file `MCGlossy.cpp`. Press '3' on the keyboard to apply this shader to all diffuse surfaces during a rendering.

Worksheet 6 Deliverables

Renderings of the default scene using different pixel subdivision levels. Renderings of the Cornell box with anti-aliased edges and including indirect illumination. Include relevant code snippets, a table of render times, and list the number of samples per pixel. Provide the explanations and answers that we ask for in the assignment text. Please insert all this into your lab journal.

Reading Material

The curriculum for Worksheet 6 is (42 pages)

B 13.4. *Distribution Ray Tracing*.

B Chapter 14. *Sampling*.

B Chapter 23. *Global Illumination*.