Raster-based Global Illumination on the GPU Using Photon Maps

Jeremiah Darais
Advanced Shader Programming
Final Project

For my final project, I partnered with Anthony Cummings to develop a global-illumination renderer that uses the GPU for both the rendering and the global-illumination calculation. The global illumination was done using photon maps. The algorithm for photon map global illumination is typically divided into two passes. In the first pass, photons are traced through the scene, originating at the light source, to get an approximation of the lighting for the scene. In the second pass, called final-gathering, rays are cast from the visible points of the scene into the photon map to get an approximation of the indirect light at each point. Anthony was responsible for implementing the first pass, and I implemented the final-gather step. Both passes were eventually implemented on the GPU. The first pass was implemented with a method similar to that described by Tim Purcell in a 2002 Siggraph paper. The second pass implementation was based on an article in GPU Gems 2, "High-Quality Global Illumination Rendering Using Rasterization" by Toshiya Hachisuka. In this article, Toshiya describes a method that does the final gathering by casting rays from all visible points, one direction at a time rather than casting rays in all directions from each visible point at a time, as is done in typical ray-tracing methods. The method uses rasterization and depth peeling to simulate casting a large set of parallel rays through the scene. Because each pass adds one indirect lighting sample for each visible point, the user can watch the final gathering step converge as it is displayed on the screen.

In the GPU gems article, a finely tessellated mesh was used to store the photon map using vertex colors to store irradiance values. For the implementation of our global illumination renderer, we decided to use a grid photon map. Using a grid-based photon map can introduce more artifacts, and must be sampled with texture lookups, whereas a finely-tesselated mesh can simply be sampled by rasterization, however a grid photon map is much easier to construct, and can produce reasonable results.

After the photon map is constructed, the scene is rendered once for direct lighting, and the result is stored in a texture. Then, random directions are chosen for casting rays for the final-gather step. The rays are cast through the scene by first setting up an orthographic view that encompasses the entire scene. Then, layers of the scene are rendered back to front with several passes with depth-peeling. This is done by reversing the z-test so that only the farthest geometry is rendered. The z-buffer from the previous pass must also be stored, so fragments as far or farther away than previously rendered layers are discarded. After each depth-peeling pass, the scene is rendered from the eye's point of view, and the image rendered from the ray's point of view is projected into the scene in a way similar to what is done for shadow maps. The ray-view coordinates are found for the visible point, and if the visible point's z-value in ray-space is closer than the rendered point for the ray, (i.e. the ray hits the visible point going from the rendered ray point), then the color at the rendered ray point is added to the visible point in a buffer used for accumulating the ray's illumination in eye-space. After iterating through all the layers of the scene, the buffer used for accumulation contains all the indirect lighting along a particular ray direction for the visible points of a scene. In our implementation, this buffer is added to the accumulation buffer, and a counter for the number of final-gather samples is incremented. To produce the final image, the indirect lighting information is drawn from the accumulation buffer, dividing by the number of samples used, and combined with the direct lighting pass that was stored earlier in a texture. More details and diagrams on the final-gather step can be found in the GPU Gems article here:
http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter38.html

The advantage to doing photon mapping on the GPU is the ability to take advantage of the parallelism capabilities of the GPU, since both passes of the photon mapping algorithm can be parallelized fairly well. The photon-tracing was implemented first on the CPU, and later on the GPU, and the speed difference between the two was apparent. For the final-gather step, we did not have an alternate implementation that would give us an idea of speed advantages of implementing it on the GPU, however there are other advantages to doing final-gathering on the GPU as well. Since it is incremental, the user can move through the scene and change the camera view, and once the view is still, the global-illumination will gradually gather more samples and converge, until the camera is moved again, thus giving a convenient way to move around a scene, and quickly get an idea of what the global illumination will look like.

We decided to use the Cornell box as our scene for testing. An accurate global-illumination renderer should lighten the shadows, darken corners, and produce visible color bleeding from the colored walls onto the boxes. We were able to achieve these effects with our renderer. Sampling of the photon grid proved to be a challenge, however, and there are some visible artifacts that result from the sampling of the 3D texture photon map. One current improvement that could be made is to develop better sampling techniques for sampling the grid photon map to reduce these artifacts.