Wendy Chen
CPSC 490
September 7, 2016

Extended Path Tracing Exploration

Path tracing is an unbiased Monte Carlo rendering technique that can be used to create physically realistic images. In the path tracing algorithm, light rays are fired from the camera into the scene, where they are allowed to bounce around until they hit a light source. Using material information from each surface the light ray collides with and using the initial amount of energy provided by the light source, the color of the ray's corresponding pixel in the image can be calculated. Because path tracing allows for light rays to bounce around surfaces in the scene before finally hitting a light source, both light from direct sources and indirect sources are accounted for. Thus, path tracing naturally simulates global illumination. When enough samples are collected, path tracing can produce noise-free, physically accurate images that resemble photographs.

Path tracing is one algorithm that solves Kajiya's "rendering equation" (1986), which describes radiance emitted towards a view direction from a particular point, given functions for incoming light and bidirectional scattering. The rendering equation assumes first that the scene is globally illuminated. Second, reflected light is equivalent to emitted light. Third, light scatters in a particular direction. In path tracing, the light paths can be constructed in two ways: shooting rays from light sources, or gathering rays from surface points. The paths that the light rays follow are based off of the BSDFs (Bidirectional Scattering Distribution Functions) of the materials from each surface the rays encounter.

The most basic implementation of the path tracing algorithm is naïve one-directional path tracing. In bidirectional path tracing, rays are both shot and gathered, and the corresponding paths are joined together. While bidirectional path tracing must cast more rays, a bidirectional algorithm converges faster. Enhancement techniques like Importance Sampling or Metropolis Light Transport (MLT) also exist. Importance sampling casts more rays in the directions where the luminance of the scene is greater, thus, casting fewer rays overall. Metropolis Light Transport is a technique often used on scenes where light must pass through small corridors (ie. Creating caustics). MLT is able to more easily handle optical effects like caustics by remembering successful sampling paths, then using new paths that are mutations of previously found paths.

Project

In my 490, I propose to implement various methods of path tracing and compare their performance on rendering scenes of varying difficulty. I plan to build each version of the path tracer off of the previous iteration. Thus, I have selected the simplest implementation of path tracing and two ways of enhancing it. The methods I intend to explore are:

- **Naïve path tracing (one direction):** In this implementation, only the gathering path would be considered.
- **Bidirectional path tracing:** Considers both the shooting paths and the gathering paths.
- **Metropolis Light Transport:** This implementation considers bidirectional paths and can remember and mutate the paths.

To test the various implementations, I have selected scenes of varying difficulties to render:

- **Heavy Geometry Scenes:** Scenes that have dense geometry or geometry of massive scale.
- **Difficult optical phenomena:** Caustics, subsurface scattering, etc…
- **Motion**: Scenes that contain animation.

Finally, I would like to compare my renders with results from freely available unbiased renderers like LuxRender or Blender.

For future work (possibly next semester), I would like to add more enhancements to the path tracer. Possible enhancements include acceleration data structures and optimizations, and parallelization of the path tracer for GPU rendering.

Implementation details

The path tracer will be built in C++ following an object-oriented style. I will look to Physically Based Rendering Theory (PBRT) and their code samples for guidance. As I won't likely build a GUI, the renderer will work from the terminal. For sample 3D models, I can find free scenes online. In addition to rendering the classics like the bunny, dragon, and teapot, I would also like to artistically assemble and create some original scenes as well.

Previous Work

My 490 project on path tracing will build off of previous work I have done in graphics, specifically with ray tracers. Ray tracing is a rendering technique in which rays are fired from the camera into the scene, and then at the point of surface intersection, a ray is fired to each light source in the scene. Thus, ray tracing computes only direct illumination, whereas path tracing can automatically account for global illumination.

My previous work with ray tracers started with accelerating a JavaScript ray tracer using a kd-tree and producing a gallery of geometrically heavy images. Since, I have ported the ray tracer over into C, and I have written a parallelized version in CUDA that runs on the GPU.

Schedule (~12 weeks)

Tentatively, two to three weeks of development time should be set aside for the implementation of each method, with additional buffer time for refactoring and image rendering.

Proposal:
Week 1 (8/28 – 9/3)

Method 1, one-directional path tracing:
Week 2 (9/4 – 9/10)
Week 3 (9/11 – 9/17)
Week 4 (9/18 – 9/24)

Method 2, bidirectional path tracing:
Week 5 (9/25 – 10/1)
Week 6 (10/2 – 10/8)
Week 7 (10/9 – 10/15)

Refactoring/Data collection
Week 8 (10/16 – 10/22)
Week 9 (10/23 – 10/29)

Method 3, Metropolis Light Transport:
Week 10 (10/30 – 11/5)
Week 11 (11/6 – 11/12)
Week 12 (11/13 – 11/19)

An enhancement (?)
Week 13 (11/20 – 11/26) Thanksgiving break
Week 14 (11/27 – 12/3)

Afterwards, I will work on the final report up through reading period.

Deliverables

- **Path tracer**: The path tracer will have been built up in layers, so there should be three versions.
- **Gallery of images**: Renders of scenes of varying complexity, and comparison images generated by freely available renderers.
- **Final report**: Write up of implementation details of my path tracer.

Sources

I will follow the PBRT text for guidance. There may also be SIGGRAPH course notes on path tracing that I could look at.