CSE 274: Optional Assignment — Monte Carlo Path Tracing

Ravi Ramamoorthi

1 Introduction

This document lists an optional assignment to build a basic Monte Carlo global illumination rendering system using path tracing. Path tracing, which goes back to Kajiya's SIGGRAPH 86 paper on the Rendering Equation, is now the method of choice for physically-based production rendering. While there have been many improvements such as photon mapping, irradiance caching, bidirectional importance sampling, and metropolis light transport, it remains a core reference algorithm. In addition, it provides a basic framework to explore a variety of new algorithms, some of which are current subjects of great interest in rendering research. In fact, most rendering papers will include a comparison image with a basic path tracer to verify the accuracy of their algorithm and provide a baseline for assessing speedups.

As far as our course is concerned, this assignment is strictly optional and is not graded, nor is it a replacement in any form for the final project of the course. However, if you do end up doing the path tracer, please document it and show results in your final project submission. For many of the algorithms in the course, a path tracer is a basic building block and hence this provides documentation of how you might go about writing your own in preparation to implement some of the methods discussed in the course. Moreover, since CSE 168 is no longer taught, this provides the basics for the initial assignment there. Note however, that this assignment is strictly optional, and you are welcome and encouraged to instead use off the shelf efficient path (or ray) tracers for your final project (PBRT, Mitsuba, OptiX etc.) rather than building your own. We are also hopeful that even if you don't do the path tracer in this course, the material will be helpful to you if you wish to build one at a later time.

Since this assignment isn't graded, some of it will be slightly under-specified, and we do not specify a grading scheme. You may want to take a look online at open source renderers for reference (a good one is PBRT, `http://www.pbrt.org`). There is a wealth of information available online, but no skeleton code available. While there is considerable flexibility since this is not an official required assignment, the general intent of thse notes is to get you to write your own path tracer from scratch.

2 Raytracing

A basic building block of a path tracer is a ray tracer (or at least a ray-surface intersection routine). As such, if you have never written a raytracer, we provide some information below. If you have already written a raytracer, feel free to use that as a starting point and skip this section (though you may want to look at, especially the acceleration part, if you have not already implemented everything considered here).

For information on ray tracing, I recommend you to my MOOC course on edX, CSE 167x. Please register (free) for the edX course at `https://www.edx.org/course/computer-graphics`. Then navigate to the raytracing part (unit 3) and lectures 9 and 10 on Raytracing 1 and 2. This unit, along with its assignment contains all of the information on writing a basic raytracer. You may also want to take a look at the corresponding lecture slides from the local UCSD CSE 167 (when I taught it in Winter 2017; please find URL from my website) which may have a bit more detail, especially regarding the acceleration structures.

If you do want to use your path tracer in the context of a final project on adaptive sampling and reconstruction, you will need to do an acceleration structure to optimize the raytracer. We do not specify what acceleration scheme you should use, and any method is fine. Perhaps the simplest is to just implement a regular uniform grid; each grid cell is a small cube and holds any geometry that intersects that grid cell (so some geometry could be duplicated). When tracing a ray, you go to each grid cell in order, and intersect only with the geometry in that cell. If a ray doesn't intersect some grid cells, they can be avoided altogether.

The trick is to find out for each sphere/triangle, which grid cells it intersects, and store them appropriately, and then to augment the ray marcher to know when it hits the next grid cell. You will need to play with grid resolution, perhaps starting with something like $5 \times 5 \times 5$.

You may need new test scenes with thousands of objects to really test the speedup (the feedback scenes also include one or two examples for this purpose). Perhaps it is simplest to find .off files online which have a very simple geometry format. You could either convert them to your raytracer format, or get the the raytracer to read that format directly.

3 Path Tracer

The path tracer assignment will largely reference the lectures in class, also relying on online material (of which there is plenty) and perhaps a reference text like PBRT. The goal is to demonstrate a basic path tracing capability for global illumination. This involves randomly generating multiple rays for a pixel, deploying a termination mechanism like Russian Roulette, and generating one or two rays at an intersection to evaluate direct and indirect lighting, respectively. You should test (and document) on a variety of scenes, including at least a Cornell box, and a shiny sphere on a plane, showing the various effects. Some key components that we would grade, if this were a graded assignment, are:

1. *Basic:* Implementation of a Path Tracer, that works.
2. *Documentation/Scenes:* Adequate documentation with scenes that properly demonstrate the method
3. *Importance:* Proper importance sampling and Russian Roulette if appropriate.
4. *Direct/Indirect:* Proper handling of both direct and indirect illumination and separation of them for computation.

In addition, you should strive to make the path tracer robust, including effects like antialiasing, depth of field (sample the aperture), soft shadows and motion blur. A minimum of Lambertian plus Phong BRDFs with basic importance sampling should also be supported.

A variety of additional items (extra credit if this were a graded assignment) will be possible. One immediate extension is to implement BRDF importance sampling for general materials per Lawrence et al. 04. A second is to also include complex image-based lighting, on the lines of Agarwal et al. 03 (or the simpler PBRT book method). You may also want to try implementing photon mapping following the work of Jensen et al.

Finally (actually, you can get here without doing the robustness and extra credit parts), you can implement adaptive sampling and reconstruction on top of your path tracer, based on the paper you have chosen to implement for your project.

In terms of support, ray tracing and rendering are covered in many textbooks and online materials, although none is "the" textbook for the class. A good reference is Eric Veach's PhD thesis at Stanford. Of historical interest is the original Kajiya 86 rendering equation paper, and Cook's 84 paper on distribution ray tracing. A suitably encyclopedic work is Andrew Glassner's Principles of Digital Image Synthesis. The PBRT book on physically based rendering is another good source. A number of other textbooks and siggraph course notes can be used. Ask the instructor if needed for other suggestions.

4 What you need to implement: Basic Path Tracing

The first goal is to implement a basic Monte Carlo Path Tracing system. The basic requirements are as follows:

Trace multiple primary rays for each pixel: In standard Whitted Ray Tracing, only a single ray is traced for a pixel (excluding antialiasing). By contrast, in Monte Carlo methods, multiple samples are drawn, with their values being averaged. Thus, you will trace a certain number of rays for each pixel, averaging their contributions (in doing so, pay careful attention to the statistics; you must divide by the probability of

each path to not bias the results). The number of rays or paths at each pixel controls how much variance or noise the final image will have; it is a tradeoff between computation time and image quality. For all of your final results, please report the number of rays traced per pixel.

Trace rays into the scene: In what follows, we discuss what happens to a single ray. First, it will intersect with the scene, as in standard ray tracing. Where path tracing differs is what happens once a ray hits a surface.

In general, path tracing uses a probability to decide where the secondary ray should be cast. First, we must choose whether to sample direct lighting, indirect "diffuse + glossy", mirror reflections or emission. Emission is generally only from the light sources, and is probably not immediately relevant here.

Probabilities for type of ray: In general, we will use some probabilities to determine what type of secondary ray to use: direct, indirect, or mirror (where relevant... there is no need to initially implement mirror surfaces and mirror reflections for path tracing... those are more relevant for Whitted ray tracing that cannot handle general materials). The probability should be noted, as the final statistical Monte Carlo calculation divides the value by the probability (for example, if a probability of one-half computes indirect lighting, its value must be scaled by two for correctness, since only one-half the rays will sample it).

Once the type of ray has been determined, we still need to decide the exact direction. For direct lighting, we can pick one of the light sources (say, based on the intensity of the light; this is related to importance sampling), and shoot shadow rays to determine shading in the conventional way.

For indirect lighting, you could simply pick a random direction on the hemisphere. Cosine-weighting (to account for the Lambertian cosine factor) will improve the results. For later robustness, you should also importance sample the combination of diffuse Lambertian and specular Phong BRDF. Note that the weight of the photon is decreased by the reflectance of the material in question, so after several bounces, it will have low weight.

A common problem has been in choosing probabilities, weighting rays and dividing by the correct probability. This is especially true for rendering with an area light, where many students do not compute the multiplicative factors correctly. Please ask the instructor for help in your implementation.

Additional Features: Note that in the above formulation, a ray can be traced forever, though its contribution diminishes with more bounces. Cutting off at a certain number of bounces, as in Whitted ray tracing, can introduce bias. Instead, you may wish to implement Russian Roulette sampling, whereby a given photon is either accepted or rejected (if accepted, its weight should be increased to account for the rejection probability). The probability of rejecting photons should increase for low weight.

Finally, while "pure" path-tracing chooses between direct and indirect light at each step, you may want to simply use two rays, one to sample direct lighting, and one for the indirect to simplify the calculations. Since the direct lighting ray is not reflected further, this does not lead to an exponential increase in rays with the number of bounces, and the accuracy of the solution may improve.

5 Robustness and Multidimensional Integration

The ideas of Monte Carlo path tracing and distribution ray tracing make it easy to take into account a variety of effects including antialiasing, motion blur, soft shadows, and depth of field. To demonstrate robustness and show more interesting images in your solution, you should demonstrate at least one of the above effects.

Antialiasing is perhaps the simplest, just by for each ray, picking where in the image plane pixel it goes through; you can simply assign it a random location or use stratified sampling for lower variance. It will be important to find good examples to show the power of the technique. Soft shadows from an area light source are similar, in that the direct lighting rays just need to sample the light source. Again, they can pick a location randomly or you can stratify. Motion blur is handled by distributing rays through time, when you generate the initial rays through a pixel, while depth of field distributes by location on the lens aperture.

In addition, you should implement proper importance sampling of the Lambertian + Phong BRDF, so you can most efficiently compute the effects of multiple bounces of rays.