CS 283: Assignment 2 — Monte Carlo Path Tracing

Ravi Ramamoorthi

1 Introduction

The goal of this assignment is to build a basic Monte Carlo global illumination rendering system using path tracing. There have been many subsequent advances over path tracing, such as photon mapping, irradiance caching, bidirectional importance sampling, and metropolis light transport. However, 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.

This assignment should be completed in groups of two. Both partners will receive the same grade. If you absolutely want to work alone, you may do so, but similar requirements will apply. There is currently no source code for this assignment, and as noted below, it will be turned in by creating a website. You may find the website created by Eric Biggs (a former course student) helpful, at *http://pathray.ebiggs.com* (this is optional, and I have not fully verified his project). Some example images from the previous edition of the course will be shown to get an idea of what is possible, and you may also want to take a look online at open source renderers for reference (a good one is PBRT, *http://www.pbrt.org*). The best assignment from a few years ago (by Daniel Ritchie and Lita Cho) is at *http://path.30minuteslate.com.* However, please see the note below regarding not copying code from existing sources. Future editions of the course may provide more support, but the point is also to get you to work independently and communicate your results, given that this is an advanced graduate course.

Please note that you are expected to write your path tracer from scratch. (You may make use of any previous code you yourself have written). Copying of code from online sources, existing path tracers, or previous instances of the class, is not permitted.

Since the background of students may vary, I will provide several options below for completing the assignment, based on how much experience you already have with writing rendering systems. We leave it to your judgement and honesty to self-identify in the categories below.

- For those who have not previously written a ray tracer, 30% of the grade can be obtained for completing a basic ray-tracing assignment, which is given separately.

  Others can start with existing code they have access to for basic raytracing. In this case, you are not required to comply with the specific instructions in the basic raytracing assignment, although you may still want to take a look at it as a reference.

  In the event you have a basic raytracer available, but need to upgrade its capabilities, such as adding acceleration structures or additional primitives, up to 20% of the grade for the assignment can be obtained for these improvements (not available to those writing a basic raytracer from scratch, where the maximum is already 30%).

  For those who need to first build or modify an existing raytracer, they are expected to start very early on the assignment, writing the basic raytracer in the first two weeks of class (before homework 1). This will allow them to be up to par with students who begin directly with Monte Carlo path tracing.

- 75% (60% for those writing a raytracer from scratch) of the grade will be available for demonstrating a basic path tracing capability of 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. Your assignment should also properly document the results, including at least a Cornell box, and a shiny sphere on a plane, showing the various effects.

The grading scheme is subject to change, but tentatively assigns the following weights to different parts of the assignment:

1. *Basic:* Implementation of a Path Tracer, that works, 25 points
2. *Documentation/Scenes:* Adequate documentation with scenes that properly demonstrate the method, 15 points
3. *Importance:* Proper importance sampling and Russian Roulette if appropriate, 15 points
4. *Direct/Indirect:* Proper handling of both direct and indirect illumination and separation of them for computation, 20 points.

- 25% of the grade (10% if writing a raytracer from scratch) is for making 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 extra credit items will be possible, and should be discussed with the instructor before implementation. The main idea is to bring you up to date with current research in rendering. 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. A third suggestion is to speed up multidimensional rendering following Hachisuka et al. 08. You may also want to try implementing photon mapping following the work of Jensen et al. Any of these extra credit items can also substitute for making the path tracer robust above (but all groups must submit a working basic path tracer in any case).

2 Submitting the Assignment and Logistics

Since one of the goals of preparing for research is to also communicate clearly, the assignment must be turned in either by submitting a PDF by e-mail to the grader (as you would a scientific paper), or by creating a website and sending a pointer to it. Document your results properly, showing the various types of scenes your method can handle. Explain any interesting algorithmic aspects. It is likely you will need to create some test scenes, though you can get some inspiration from the Cornell Box (you can easily find the geometry for this online, or simply type it in...) and simple spheres on a table.

Failure to fully document the results of the program will likely lead to unhappiness from the grader and me. Be honest, and note what works and what doesn't; this will be the most fun, and make the assignment easiest to grade. Some part of the grade will be reserved for the clarity of the writeup.

If you make a website, do not modify it after the due date. In either case, please include links to the source code and executable, and again do not modify them after the due date. *It is critical the source code for the path tracer be made available, since we do look at this for grading. In general, please link it off the project website (or if submitting a PDF, include a link).*

Support: Feel free to contact me regarding questions. There is no code framework provided for this assignment. If needed, you are welcome to look at online code sources, like POVray or PBRT mentioned above for inspiration, but all code turned in must generally be your own. If in doubt, ask. (This does not apply to basic utility code, such as for reading and writing image files in different formats).

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. These are all linked in the reading resources from the class website. 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.

3 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. We will discuss this issue in class, but 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, it's 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.

4 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.

5 Extra Credit

There are certainly many possibilities to extend path tracing, and these can provide avenues for extra credit. Some possibilities are listed below. You may wish to speak to the instructor before tackling any of these. Since this is extra credit, it is specified only at a very high level. There are many other possibilities so talk to the instructor if you have other ideas.

- Implement one of the extensions like bidirectional path tracing, photon mapping, or Metropolis Light Transport.
- Include general BRDFs for materials, and extend your importance sampling scheme per Lawrence et al. 04
- Include image-based environment lighting per Agarwal et al. 03
- Explore ways of accelerating multidimensional rendering, for example the multidimensional adaptive sampling paper of Hachisuka et al. 08. There are also many more recent extensions on this that my group has been working on, so see me if you are interested.
- Optimize your path tracer for efficiency by optimizing the ray-tracing routines, including acceleration structures, and implementing many Monte Carlo importance sampling and stratification methods.