CSE 167: Assignment 4—Simple Raytracer

Ravi Ramamoorthi

1 Goals and Motivation

This assignment asks you to write a first simple raytracer. Raytracers can produce some of the most impressive renderings, with high quality shadows and reflections. With rapid advances in graphics hardware and compute power, today virtually all offline rendered images in production for any application are raytraced, and within a few years, we expect the majority of pixels in real-time applications like games to also be raytraced. (Unfortunately however, it is unlikely that you as a first-time coder of raytracing will be able to deliver a real-time product and in fact your raytracer will likely be really really slow. Anticipate this, and plan ahead by STARTING EARLY and giving yourself enough time to render final images).

While very rewarding, please note that many students have commented that *this is the hardest (but most rewarding) assignment they have done in their career at UCSD.* Please be aware of this and START EARLY. Also for this reason, while it is strictly optional, *please do try to work in a group of two to reduce workload.* The requirements are essentially the same, whether or not you work in a group of two.

In fact, you will be using much the same file format as for assignment 2 on the scene viewer, allowing you to make direct comparisons of images produced using standard OpenGL rasterization and with raytracing.

Raytracers are conceptually very simple. However, the actual implementation effort can be considerable. Therefore, you should start early and proceed through the assignment strictly in the order of the specifications provided. For this reason, we also include a milestone. The assignment can be fairly hard to debug. You should strive to make progress incrementally. Start with the simplest functionality (can you render an image with one triangle on the screen?), debug that fully and then proceed. Trying to write the whole thing at once will lead to a mess of undebuggable code.

The instructions for this homework are on UCSD Online as is the image-grader and feedback server (no code-grading or skeleton code for this assignment, no OpenGL). The purpose of this PDF is only to document a few additional topics, including the milestone submission, acceleration structures and extra credit.

2 Logistics

We do provide some test scene files. Download the file *testscenes.zip* which has three test scenes, that are documented and have multiple camera positions you should try. There are also images of these scenes with different camera specifications. Note that these images were created in an OpenGL previewer and are a useful guide, but do not have the sophisticated shading, shadows and reflections, which your raytracer will provide for the same images.

Beyond these initial examples, we do provide an image-based feedback server for this assignment through UCSD Online, which includes a set of more complicated and realistic scenes. As with previous assignments, you are required to provide a link to the feedback server output in your final submission (not milestone). Please also note that the feedback server sends rays through the center of a pixel (i.e., at locations 0.5, 1.5, 2.5 and so on rather than at integer values). This is useful for getting an exact match. *Please follow the file format and other instructions on UCSD Online.*

3 Submission

Submission will be using the standard mechanism in the assignment submission (both partners should follow the official CSE 167 submission procedures and submit separately, also noting their partner in the README. Of course, the code and links to feedback output will be the same). In addition, your submission can

optionally include the link to a website which has images and documentation of your raytracer (but **please do not post source code.**). This website is optional, but is recommended if you want to show off additional examples, and *is required if you want extra credit (in which case, it should document and show example images that justify the extra credit).* Your source code submission should also include a *README* as always. In this case, your *README* should also briefly explain what you did for your acceleration structure (if you implemented one), and point to any images or documentation/timings showcasing that it works.

4 Milestone

For this homework, we require that you submit a milestone to remain on track. Milestone submission will follow the same procedure as above, except that you would of course have done much less of the assignment and may not yet have a link to feedback server results. You also need only include a link to the website or a PDF document, not the source code. This year, we are proposing a more structured milestone than in the past, but please bear in mind that these are *minimal* requirements. This is a hard assignment, where the earlier you start and the more you do, the better for the final submission. The milestone will count for 10 points, and the final submission for 60 points (total of 70 points).

In partciular, I expect you to have completed the camera routine, and basic ray-surface intersections for both triangles and spheres. It would also be good if you had the basic transformations working. At this point, the milestone does not require lighting or shading, but getting started on them as soon as possible is still a good idea.

To demonstrate these, you should document the first two scenes in *testscenes.zip*. Specifically, *scene1.test* includes just a simple plane, and mainly tests that you can set up your camera correctly. You should show images of your raytracer generating images for each of the three camera positions given, and compare to the reference OpenGL snapshots. (To speed up the process for scene1, you could just implement a sort of hacked up ray-quad intersection.) It is not required that shading match exactly (or that you implement shading at all), but the geometry of the plane should match. Next, you should implement ray-triangle and ray-sphere intersections, and demonstrate them on the dice example in *scene2.test*. Again, for each of the reference images and camera positions, show a comparison of your result and the reference solution (which was rendered in OpenGL).

While not required by the milestone, the final submission will also require basic lighting and shading, shadow rays and recursive ray tracing (we recommend you do them in that order). *scene3.test* begins to test proper geometry transformations and lighting and shading, and is included in several parts of the final feedback scenes. As such, while not required for the milestone, you would benefit from also getting your program to work on *scene3.test*. Especially if you are working in a group of two, you really should be doing this. Please also see the detailed steps on UCSD Online; you should be able to almost immediately get some of the feedback server examples working if you can demonstrate the milestone.

If you are working alone, and you don't want extra credit or otherwise want to implement acceleration, you can ignore the next section. However, there's still a lot to do after the milestone, in terms of completing transformations, lighting and shading, recursive raytracing etc. You should still be able to pass all the image cases on the feedback system of UCSD Online, and do bear in mind that some of the scenes may render slowly in the absence of an acceleration structure.

5 Acceleration Structure (not required for students working alone)

Please note that the acceleration structure is a required part of the final assignment (not milestone) for students working in groups of two, and is documented here, rather than on the UCSD Online site.

Ray tracing has historically been a slow process, and the scenes you are provided therefore have pretty simple object geometry. To get raytracing to work on more complicated scenes, some type of ray tracing acceleration structure is required. Therefore, this assignment requires that you implement some sort of acceleration scheme. Note that this refers to a conventional geometric acceleration structure (such as axis-aligned bounding boxes or kd-trees or uniform/non-uniform grids), not optimizations such as parallelization or using hardware.

(This can be difficult; at least 90% of the credit will be given for the core components above, but if you want a perfect score, you must do this part; however, please do it last and you will lose only a very few points if you do not do it. This part is also only required for groups of two; not for students working alone.).

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 $5 \times 5 \times 5$.

If you do get this working, please document the algorithm and speedup in your writeup *README*. You may need new test scenes with thousands of objects to really document and 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 raytracer to read that format directly.

Again, note that this part will not be judged too harshly, and will lead to only a small deduction if not implemented. However, it is important to do a complete raytracer, and I hope that at least some groups will go forward. We will not be picky; we just require some effort at acceleration (it is also worth a small amount of extra credit for students working alone).

Just as one way of measuring speedups and improvements (for fun, not necessarily part of the homework), you could measure how many rays per second (on some set of scenes) your raytracer can achieve. For comparison, the latest NVIDIA Turing RT raytracing architecture reports more than 10 Billion rays per second!

Doing the ray tracer with acceleration structures overlaps a good bit with homework 1 of CSE 168 (at least when I have taught it the past couple of years). As such, if you've gotten this far, you may also want to consult UCSD Online's CSE 168 and look at the lecture and material on raytracing acceleration there.

6 Extra Credit

Please do the extra credit only after completing the regular assignment. The number of points of extra credit are small, especially relative to effort required. Up to 10 points of extra credit will be given for additional features and/or the best images produced by your ray tracer. Feel free to exercise your creativity here and go overboard. Obvious ideas are more primitives, better acceleration structures, soft shadows from area lights, Monte Carlo sampling for handling complex lighting and materials, and so on. If you are submitting for extra credit, your submission must include a link to a website, where you document these features and show images demonstrating the extra credit.