Ray Tracer

CS 354 Assignment V

Due 1:00 PM on Apr 22

1 Overview of Assignment

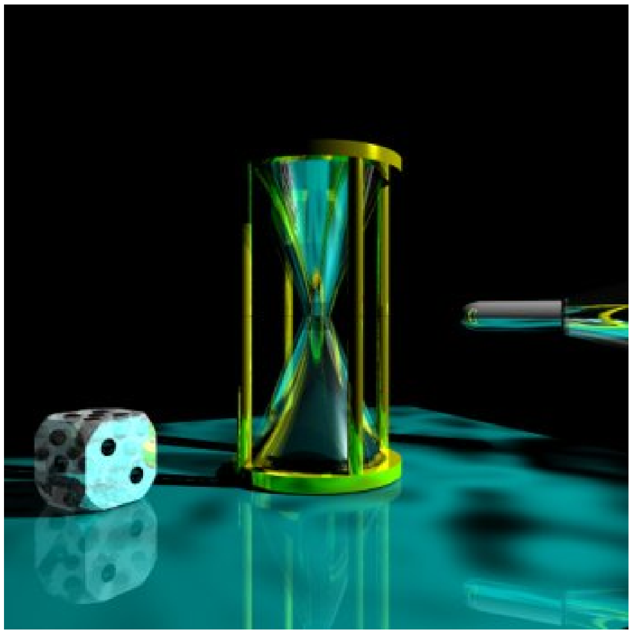

Figure 1: A scene ray-traced using a reference implementation of the ray tracer.

In this project, you will build a program that will generate ray-traced images of complex scenes using the Whitted illumination model.

1.1 Reminder about Teams

Per the syllabus, you may work on this project in self-selected teams of up to two. If you work in a team, all team members will get the same grade.

1.2 Starter Code

Unlike in previous assignments, the ray tracer will *not* use OpenGL. We have provided, as starter code, a C++ skeleton that includes functionality like reading in both `.bmp` and `.png` files for texture maps, and writing out images you produce as `.bmps`.

2 Building and Running the Starter Code

To install the starter code, unzip the files and type `make`. The executable resulting from a successful build generates extremely simple ray-traced images where the pixels are shaded only by the diffuse terms of the material at the ray intersections, and this only for non-polygon-mesh objects. For polygon meshes, it doesn't even compute intersections.

This project is a very large collection of files and object-oriented C++ code. Fortunately, you only need to work directly with a subset of it. However, you will probably want to spend a bit of time getting familiar with the layout and class hierarchy at first, so that when you code you know what classes and methods are available for your use.

To get started, look at comments about where to put your code in `RayTracer.cpp`, `trimesh.cpp`, `light.cpp`, and `material.cpp`. The starting point for where ray tracing begins, and where you will be needing to add a lot of functionality, is in the `RayTracer.cpp` file. This is a good file to start studying to explore what methods get called and what they do. In addition, the ray tracer features a debugging window that allows you to see individual rays bouncing around the scene. This window provides a lot of visual feedback that can be enormously useful when debugging your application. A more detailed explanation of how to use the debugging window is available online at `http://www.cs.utexas.edu/users/fussell/courses/cs384g/projects/raytracing/debug.html`.

The starter code can run in both text mode and gui mode. Running without any arguments will execute the program in the gui mode. For usage see `ray --help`.

3 Submission Instruction

All submissions must be uploaded to Canvas by the project due date and time. If you are working in a team, only one team member needs to upload the assignment. In exceptional cases where Canvas is not working, you may turn in the assignment by emailing the TA a Dropbox link, handing him a USB stick, etc, but there will be *no exceptions* to the project due date–late submissions will be penalize per the syllabus late policy.

Submit your project as a single .tgz file. This file should be free of temporary files, core dumps, etc. In addition to your complete source code, the .tgz file should include a `README` file containing, at minimum:

1. Your name and the name of your teammate (if any);
2. An explanation of any required features that you know are not working completely correctly, but would like to have considered for partial credit;
3. Anything the spec below asks you to include.

All code ***MUST* compile and run on the 3rd floor GDC computer lab Linux machines!!!**

In particular,

- Do not submit a 7zip of a Visual Studio .soln file;
- Do not include any absolute paths in your Makefiles that point to your home folder;
- If you use any external libraries (which must be approved in advance by the instructor) that are not installed by default on the GDC computers, you must include the library source files with your submission, and include building of the library in your build process.

Please triple-check that your submission adheres to these instructions. *Instructor or TA reserves the right to return, ungraded, any submissions that fail to compile or run, or that do not include a `README` file.*

Tip After uploading your files to Canvas, it is not a bad idea to download them onto a different computer and check that it still compiles and runs, in case you accidentally leave out some important files from the .tgz. It is also wise to try to upload to Canvas well ahead of the final deadline: again, there will be *no exceptions* to the lateness policy, not even for technical difficulties.

4 Grading

The assignment is broken down into several features described below. Each feature is worth a number of points and will be graded separately (although later feature may build on and require a working implementation of earlier features). Your code should be correct and free of segfaults, deadlocks, or other buggy behavior. You will *not* be graded on software engineering (use design patterns and unit tests if they help you) but unreadable code is less likely to receive partial credit. You should not worry too much about performance or optimizing your code, but egregiously inefficient implementations of any of the features will be penalized. In addition, extra credit will be offered for especially efficient implementations of the ray tracer (see below).

5 Required Features *(150 points)*

You will need to fill in the missing pieces of the ray tracer in the starter code.

- *(30 pts)* The starter code has no triangle-ray intersection implementation. Fill in the triangle intersection code so that your ray tracer can display triangle meshes.
- *(40 pts)* Implement the Whitted illumination model, which includes Phong *shading* (emissive, ambient, diffuse, and specular terms) as well as reflection and refraction terms. You only need to handle directional and point light sources, i.e. no area lights, but you should be able to handle multiple lights.

The first three terms in Whitted's model will require you to trace rays towards each light, and the last two will require you to recursively trace reflected and refracted rays. (Notice that the number of reflected and refracted rays that will be calculated is limited by the `depth` setting in the ray tracer. This means that to see reflections and refraction, you must set the depth to be greater than zero!)

When tracing rays toward lights, you should look for intersections with objects, thereby rendering shadows. If you intersect a semi-transparent object, you should attenuate the light, thereby rendering partial (color-filtered) shadows, but you may ignore refraction of the light source.

Some equations that will come in handy when writing your shading and ray tracing algorithms are available online at `http://www.cs.utexas.edu/users/fussell/courses/cs384g/projects/raytracing/equations.pdf`.

- *(10 pts)* The skeleton code doesn't implement Phong interpolation of normals. You need to add code for this (only for meshes with per-vertex normals.)
- *(20 pts)* Once you've implemented the shading model and can generate images, you will notice that the images you generated are filled with "jaggies." You should implement anti-aliasing by super-sampling and averaging down. You should provide a slider and an option to control the number of samples per pixel (1, 4, 9 or 16 samples). You need only implement a box filter for the averaging down step. More sophisticated anti-aliasing methods can be implemented for extra credits.
- *(50 pts)* Implement a KD tree or bounding volume hierarchy (BVH) to accelerate ray-object intersection.

Make sure that you design your acceleration module so that it is able to handle the current set of geometric primitives – that is, triangles, spheres, squares, boxes, and cones.

The starter code includes several sample scenes of varying complexity. The complex scenes are `trimesh1`, `trimesh2`, and `trimesh3`. You will notice that `trimesh1` has per-vertex normals and materials, and `trimesh2` has per-vertex materials but not normals. Per-vertex normals and materials imply interpolation of these quantities at the current ray-triangle intersection point (using barycentric coordinates).

6 Extra Credit *(up to 50 points)*

Many opportunities for extra credit are available for this project.

Important Any extra credit you attempt must obey the following two cardinal rules:

1. Your code **must** obey the above spec and match the reference solution. In other words, you can add new features for extra credit, but should *not* modify the way that existing features function. You *will* be deducted points if your extra credit work causes your program to violate the spec!
2. Include a brief explanation of any scenes/features you would like considered for extra credit in your README file. Undocument or poorly-documented extra credit features risk receiving no credit.

6.1 Performance Contest *(up to 15 points)*

During grading, the submission whose ray tracing code is the fastest will receive 15 points extra credit, and other high-efficiency submissions will receive partial extra credit. To measure rendering speed, a *secret scene* not included in the starter code (containing at least a few thousand triangles) will be traced using your code. Your code should trace one ray per pixel, and the rays should be traced with 5 levels of recursion, i.e. each ray should bounce 5 times. If during these bounces you strike surfaces with a zero specular reflectance and zero refraction, stop there. At each bounce, rays should be traced to all light sources, including shadow testing. The command line used for testing rendering speed looks like:

```
ray -w 400 -r 5 [in.ray] [out.bmp].
```

Only ray tracers that correctly render the secret scene will be eligible for performance extra credit.

You are welcome to use multiple threads when traing rays. You are also welcome to precompute scene-specific acceleration structures when your program starts, and make other time-memory tradeoffs, but your precomputation time and memory use should be reasonable and will be included in the timing of your ray tracer. Don't try to customize your ray tracer for the test scenes; we will use only the *secret scene* when running the performance contest.

If you have any questions about what constitutes a fair acceleration technique, ask us. Compiling with optimization enabled (e.g. the `-O3` flag in gcc) is allowed. Coding your inner loops in machine language is not allowed. Using multiple computers or the cloud is not allowed. In general, don't go overboard tuning aspects of your system that aren't related to tracing rays.

6.2 Creative Scene *(up to 10 points)*

You may also include ray tracing scenes of your own; obviously, your ray tracer should be capable of rendering these scenes. Complex and creative scenes will receive more extra credit. Please carefully describe any scenes you want considered for extra credit in your `README` file.

6.3 Additional Technical Features *(up to 15 points)*

You may add additional features to the ray tracer for extra credit. Particularly impressive submissions will receive more points, and you may be invited to demo an outstanding entry for the class. Some ideas include

- add a menu option that lets you specify a background image to replace the environment's ambient color during the rendering. That is, any ray that goes off into infinity behind the scene should return a color from the loaded image, instead of just black. The background should appear as the backplane of the rendered image with suitable reflections and refractions to it;
- add some new types of geometry to the ray tracer. Consider implementing torii or general quadrics. Many other objects are possible here;
- implement texture and/or normal/bump mapping;
- implement procedural textures based on Perlin noise;
- implement CSG, constructive solid geometry. This extension allows you to create very interesting models.

7 Reminder about Collaboration Policy and Academic Honesty

You are free to talk to other students about the algorithms and theory involved in this assignment, or to get help debugging, but all code you submit (except for the starter code, and the external libraries included in the starter code) must be your own. Please see the syllabus for the complete collaboration policy.

8 Special Thanks

This project is a lightly-modified version of the ray tracing project in Don Fussell's undergraduate and graduate graphics class. The starter code was written by Don.