DVA338 Fundamentals of Computer Graphics

Assignment 3

1. Implementing a Ray Tracer

Global illumination methods can produce rendered images with superior quality since they take into account both direct illumination (light coming directly from the light sources) and indirect illumination (light inter-reflected among surfaces in the scene). Ray tracing is the most well-known algorithm that is used in global illumination and it is well-suited to capture effects such as shadows and specular reflections. More advanced global illumination effects can also be produced using various ray shooting schemes, e.g., diffuse inter-reflections and caustics. In this project, you will first implement a simple ray tracer, which gives accurate shadows and specular reflections (according to Section 1.1). Then you will extend your ray tracer with more features by choosing freely from the suggestions given in Sections 1.2-1.13 below. You will need to do at least two of the items 1.2-1.13. How many of the extra features you want to support depends on your ambition, skills, and the time you want to put into this assignment.

TLDR: Solve 1.1 and two of 1.2-1-13.

1.1 Simple Classical Ray Tracer

Start by downloading the startup code from the course web page. The code includes some rudimentary functionality to render spheres using a single colour by following the eye rays from a fixed viewpoint. The resulting image is stored in an internal bitmap structure, and the rendered image is also shown on the screen by drawing the pixels using OpenGL. Your task is to extend this source code to a simple classical ray tracer. Here follows a list of things you need to do:

- Define a camera model using suitable parameters from which the eye rays can be derived conveniently. Note that it should be possible to position the camera arbitrarily in the scene.
- Extend the ray-sphere intersection test so that the ray parameter t is calculated for the closest hit point.
- Implement a local illumination model to calculate the colour at the closest hit points. For this, you also need to define a point light source and suitable material properties for the spheres in the scene.
- Include shadow rays and reflection rays to get hard shadows and specular reflections among the spheres.
- Measure the rendering time in seconds and the number of executed ray-sphere intersection tests. Show the results of to the user.

***NOTE**: The startup code only includes the C++ and h files. You should create your own project and handle library definitions for that as you did for the first assignment.*

1.2 Interactive Camera Positioning

Add support for interactive navigation in the scene by giving the user complete control of the camera positioning. You probably need to support a simpler real-time rendering method to be used during camera positioning, and then switch back to ray tracing when the final view has been found, since a computer graphics application needs a frame rate of at least 5-10 images per second to be considered as interactive.

1.3 Transparent Materials

To render transparent objects realistically, the bending of light as it travels through different media must be taken into account. Add support for tracing refraction rays so that realistic rendering of spheres made of e.g. glass becomes possible.

1.4 Geometric Primitives

Add support for a geometric primitive other than the sphere in your ray tracer. The most commonly used primitive in ray tracing is the triangle. Alternatively, you may consider supporting, e.g., planes, convex polygons, boxes, cylinders, cones, ellipsoids, or tori.

1.5 Texture Mapping

Extend your ray tracer with support for 2D texture mapping using bitmap images. In addition, implement support for procedural 3D texture mapping to imitate materials such as wood and marble.

1.6 Anti-Aliasing

Implement an anti-aliasing technique to improve the quality of the rendered images. A simple but computationally costly method for anti-aliasing is to trace a whole bunch of rays for each pixel and blend the resulting colours together. This is called super-sampling. Alternatively, you can trace fewer rays by using adaptive super-sampling, where you start by tracing e.g. four eye rays per pixel. Then depending on how much the four resulting colours differ, you determine if you need to trays more eye rays, or not, for this pixel. Another interesting approach is to use a stochastic sampling method to determine which rays to shoot for a given pixel location.

1.7 Soft Shadows

Implement support for triangular area lights in you ray tracer to make soft shadows possible. The simplest way to accomplish this is to treat the area light as if it was a set of point lights instead, which are placed in some suitable pattern over the triangle defining the area light.

1.8 Glossy Reflections

Glossy or blurred reflections arise on some surfaces, since the material is neither an ideal reflector, nor a purely diffuse surface. Simulate this effect in your ray tracer by randomly perturbing the ideal reflection ray in some suitable way.

1.9 Ambient Occlusion

Add support for computing ambient occlusion to increase the realism of your shading calculations. To determine the ambient occlusion for a certain point,

fire a bunch of shadow rays distributed over a hemisphere and collect the results.

1.10 Acceleration Data Structure

Brute force ray tracing is extremely computationally expensive. To make the rendering faster, select and implement an acceleration data structure, such as a bounding volume tree, a grid, or a kd-tree. Measure the rendering speed with and without your acceleration data structure for a number of scenes of varying complexity.

1.11 SIMD Computations

Use vectorised instructions, such as Intel's SSE or AVX, to speed-up the most frequently used computations in your ray tracer, such as the ray-primitive intersection test and the evaluation of the used shading equations. Measure the execution time with and without the SIMD computations. You are allowed (but not limited to) use Intel's Embree library for this.

1.12 Multi-threading

Parallelize the overall rendering process by using multi-threading (on a computer equipped with a multi-core processor). Define a suitable workload (sub-image to render) for each thread. To be clear, the balance of the workload should be somewhat equal for all threads. Measure the execution time with and without multi- threading.

Attn: Make sure to not miss any race condition that could occur while working with multiple threads.

1.13 Other Techniques

Feel free to implement any other interesting features in your ray tracer. Try to think creatively to come up with some nice ideas. Also, there are lots of nice proposals in the ray tracing literature that you can study to get some inspiration.

2. Cheating

Note that all sorts of plagiarism are strictly forbidden! Obviously, it is not acceptable to hand in someone else's work as your own. Make it a good habit to present only your own work (ideas, solutions, text, source code, etc.) as yours and attribute all others sources from which you have benefited, be it a person, book, web page, or something else. Also note that "helping" another student by giving them a copy of your own solution (or a part of it) is also considered as cheating. If you have any questions about what is, or is not, proper academic conduct, please ask the course leader.