Project II: Ray Tracing Engine

CSCI 155 Computer Graphics
Pitzer College
Spring 2019

Due: 10 p.m. on Wednesday, May 8

In this project you will implement an extendable ray tracing engine.

It is important to understand the setup. A scene to be rendered consists of *geometry* whose appearance is determined through *material*s which is a consequence of the *BRDF* of the material. The scene is lit by one or more *light* sources. An image is formed when the scene is *sample*d by a *camera* through a *view plane*.

The variety of objects of each of the above types is captured through class hierarchies in the accompanying `raytracer` directory. Each folder, e.g. `geometry`, defines an abstract class, e.g. `Geometry`, which is then inherited by different concrete classes, e.g. `Plane` and `Sphere`.

The `utilities` folder includes utility classes, notably `Image` and `ShadeInfo`. `Image` holds pixel colors and writes an image to file in PPM format. `ShadeInfo` contains all the information required for shading a point. These classes refer to classes from the `world` folder.

The `world` folder includes 2 classes. `ViewPlane` contains information on the view plane. `World` contains all the information required to render the scene–the geometry and associated materials, the light sources, the view plane, the camera and sampler, and the background color. `World::build` populates the world. You will reimplement this function each time to define a new image to be rendered. The `build` folder includes 2 implementations as described below.

Go over the provided files to make sure that you understand the overall structure. The provided classes are suggestions. In working with them for the problems below, you may extend or modify them as necessary. You will also add new classes to existing hierarchies, and create and populate new hierarchies.

1. Hello World

To get things started, a `Cosine` material is included in the `materials` folder which does not use any BRDF; `raytracer.cpp` includes a basic ray tracer; `buildHelloWorld.cpp` contains an implementation of `World::build` to define a simple scene; and `buildChapter14.cpp` contains an implementation of `World::build` that defines a scene similar (but not identical) to the title picture of Chapter 14. Implement the necessary files in order to render the 2 scenes and share the resulting images on Workplace. You may use exhaustive ray tracing for now.

2. Acceleration

Add a new folder called `acceleration` and populate it with a hierarchy of acceleration structures. Add an `Acceleration*` member to `World` and use it to compute ray intersections.

3. Ray Tracing

The ray tracer in `raytracer.cpp` is very basic–it shades based on primary rays only. Add a new folder called `tracers` and populate it with a hierarchy of `Tracer` classes. You can start the hierarchy by moving the ray tracer in `raytracer.cpp` to a `Basic` class derived from `Tracer`. Add other ray tracers that implement other ray tracing features like computation of shadows, recursive levels of reflections, and transparency. Add a `Tracer*` member to `World` and use it for ray tracing.

4. Appearance

Implement some `BRDF`s and correspondingly define new `Material` subclasses that use them.

5. Showcase

We now want to showcase your ray tracing engine–how good it is and how it can be used to create stunning images. Your task is to:

- **Create an original scene**. The scene should be *original*. You can get inspiration from past rendering competitions at other institutions, e.g. at Uni Saarland (2018, 2017) and at Stanford, or from the various ray-traced images available on the web, e.g. at Internet Raytracing Competition, but the final scene should be the product of your own imagination.

  You may use third-party assets, such as models or textures (e.g. from 3D Warehouse, Blend Swap, repositories at Stanford, Georgia Tech, the VisionAir project (occasionally down), and by Keenan Crane). Those assets must be publicly available for free. You can use those assets to build your original scene, but it is not allowed for the whole, or major part of the scene to be straight reused from somewhere. Alternatively, you can model everything yourself, e.g. using Blender, Maya, 3DS Max, or SketchUp.

- **Use your engine to render the scene**. You must render two images of your scene: a low quality at around 480x360 and high quality with resolution 1920x1080 or higher. You may use different aspect ratios if they better fit your scene.

  The images must be rendered by your engine. Any post-processing must be implemented within your framework. The images need not be realistic. You can use features that do not follow real world physics if your artistic concept demands it.

  The high quality image must render in less than 12 hours on a modern computer.

- **Create a web page to showcase your work**. The website should feature your image and its title. It should also
  - summarize your concept and how you arrived at it,
  - shortly describe how you built your scene,
  - highlight interesting parts or features of your image. Additional images may be included for this purpose.
  - list all the additional features and changes you have implemented on top of the code provided for the project. This includes changes made for the problems above.
  - include a table comparing rendering times with and without an acceleration structure. Supporting renderings must be included.
  - link for every included image to the corresponding implementation of `World::build`,
  - acknowledge all third party sources of used assets or resources, once where they are used, e.g. in the caption of a rendered image, and again as part of a master list toward the bottom of the page.
  - include the names of all team members and a photograph of your team.
  - include any other comments desired by the team.

Credits

This project is adapted from the rendering competition run by Philipp Slusallek in his Computer Graphics 1 course. The code is adapted from that provided by Kevin Suffern.