Automagic's Raytracing Renderer
Computer Graphics: Rendering, October 2024
Coursework 2

Sean Memery, Krzysztof Grykiel, Kartic Subr

1 Overview

Great news, *Automagic* were impressed by your previous work and have hired you as a creative consultant! Your next task is to bootstrap the core renderer for the startup's augmented reality toolkit. For this assignment you are expected to develop code for a full raytracing renderer along with more advanced features if you can. *Automagic* want a full report on your work including your methodology and evaluation.

Your expected contribution is as follows: A basic raytracer that implements Blinn-Phong rendering, implementation of reflective and refractive materials and a report outlining the steps taken to implement your raytracer. On top of this, there are some more advanced features that *Automagic* think would be a good inclusion, but aren't required for a basic submission.

2 Specifications

2.1 Raytracer

Goal: Code a `C++` raytracing renderer from scratch with the following features.

Features:

1. Image write (`ppm` format)
2. Camera implementation, with coordinate transformation
3. Intersection tests (sphere, triangles, cylinder)
4. Binary image writing (intersection/no intersection)
5. Blinn-Phong shading
6. Shadows
7. Tone mapping (linear)
8. Reflection
9. Refraction
10. Textures (on sphere, triangle, cylinder)
11. Bounding volume hierarchy as an acceleration structure

Constraints:

1. Make your program capable of loading a provided `JSON` scene file.
2. Output a `ppm` image of the scene.
3. Recreate the rendered images of the example scenes (not pixel perfect).

4. The code should be clean and readable, with indicative variable and function naming, and should contain ample comments describing function's operations and variable roles.

Example renders of a provided `JSON` scene are shown in figure 1. These images showcase the expected behaviour of your raytracer and include a render of a custom scene with the advanced submission features.

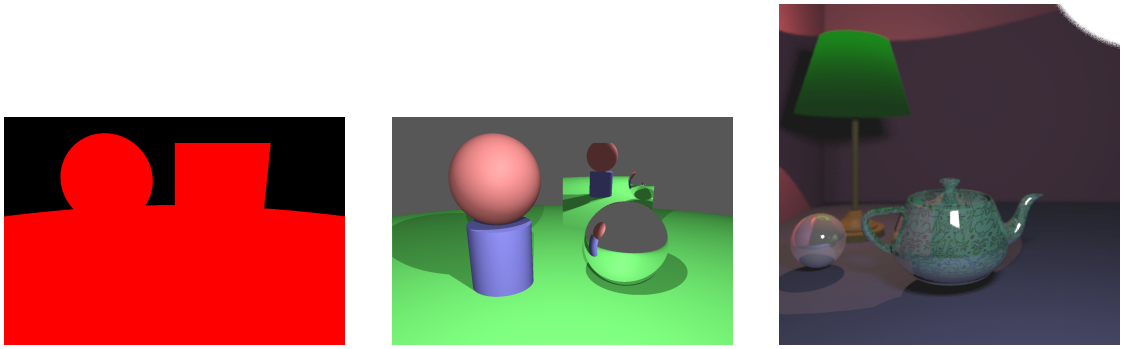
2.2 Pathtracer

Goal: Improve your raytracer with a path tracer with the following features.

Features:

1. Antialiasing via multi-sampling pixels
2. Defocus in finite-aperture cameras by sampling the camera's aperture
3. Render materials with BRDF's (e.g. microfacet)
4. Soft shadows via sampling area lights
5. Multi-bounce path tracing
6. An advanced feature of your choice (eg. Caustics, Volumetrics, fancy BRDFs, etc.)

Constraints: Same as those for the raytracer.



(a) Render Mode: **binary** (b) Render Mode: **phong** (c) Render Mode: **pathtracer**

Figure 1: Example renders of scene.json utilising the **basic Raytracer** features and of a custom scene utilising the **Pathtracer**. Additional scenes are provided for you to test your raytracer with.

3 Marking scheme

A total of 100 points are assigned for this project, which will then be halved, i.e. its final contribution to your grade will be at most 50%, depending on the marks. See the course website for clarification. The marking scheme is described below.

1. **Basic raytracer features** 35
   (a) Image writing (3)
   (b) Virtual pin-hole camera (5)
   (c) Intersection tests (5)
   (d) Blinn-Phong shading (5)
   (e) Shadows (5)
   (f) Tone mapping (2)
   (g) Reflection (5)
   (h) Refraction (5)
2. **Intermediate raytracer features** 15
   (a) Textures (5)
   (b) Acceleration hierarchy (10)
3. **Advanced raytracer features** 20
   (a) Pixel sampling (5)
   (b) Lens sampling (5)
   (c) BRDF sampling (5)
   (d) Light sampling (5)
4. **Report** 10
   (a) Conciseness (5)
   (b) Effectiveness (5)
5. **Exceptionalism** 20

This coursework is meant to be an *independent exercise* and so create your own code, scene, and video. **Do not copy** the examples provided, and **do not collaborate** with each other to actively write code. Indeed, we encourage you to discuss and brainstorm general concepts pertaining to the coursework or the use of programming assistants.

The use of generative AI is encouraged for this coursework, but not required. Bear in mind that the workload was designed assuming that students would use AI (as demonstrated in class), so avoid it at your own risk!

If you execute the basic raytracer, along with the report, you can expect a mark of about 45%. The expected time for students who have kept up with lectures and tutorials to attain this is about **16h**. The intermediate features are expected to take about **15h** and can allow you to score about 65% on this coursework. Be warned that the advanced submission is not required and could lead to you spending a large amount of time. We do not recommend that you attempt the advanced features if you have already spent more than 35h on the raytracer. Even so, we strongly recommend that you limit your time to at most **12h** on the advanced features.

Some important tips include: a) design and plan your implementation to be modular; b) use programming assistants to obtain code for individual modules; c) test individual modules; d) engage with the instructor/TA if some component is taking too long; and e) **start early**.

4 Deliverables

Submit a compressed .zip with your student ID as the name. e.g. "s123456.zip" via Learn. Any deviations from the following format will incur a penalty of 5% of the marks for impacted sections.

4.1 Folder structure

The structure of the zip should be as follows:

```
s123456
├─ FeatureList.txt
├─ Code
│  ├─ Makefile
│  ├─ raytracer.cpp
│  ├─ raytracer.h
│  └─ ...
├─ TestSuite
│  ├─ binary_scene.ppm
│  ├─ phong_scene.ppm
│  ├─ binary_primitives.ppm
│  ├─ mirror_image.ppm
│  └─ simple_phong.ppm
└─ Report
   └─ s123456.pdf
```

FeatureList.txt:
List all the features in this spec, and for each, whether you have implemented it and whether you have demonstrated it working fully.

Code:
The `Code` folder should simply contain all of the code used to create your program. This should include a `Makefile` used to compile the source code, and the individual `cpp` and header files.

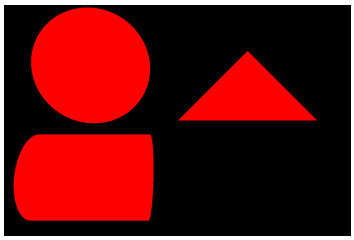
TestSuite:
Include images produced by your raytracer with each of the files provided as part of the test suite. For scene.json, include both the images (binary and phong modes).

Report:
The folder `Report` should contain only one file, which will use your student ID (or exam number) as the filename. This will be your report as a PDF. No other format will be accepted. Your report should explain the steps taken to implement your ray tracer, with detailed descriptions of feature implementations, and rendered images illustrating each of its abilities. Explain your work, step by step, with inline images. Figures should be numbered, annotated, referenced and clearly visible. Therefore, the report should include a section for each implemented feature (notated by its number in the specification section, e.g. for shadows: Basic Raytracer e - Shadows) and **an explanation of how it was implemented and an evaluation of the implementation**.

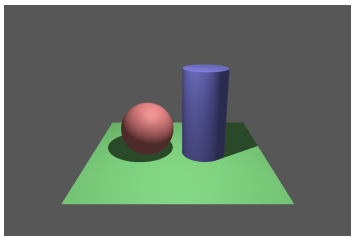
5 Tips

There are many moving pieces in a raytracer program and approaching it from scratch can be challenging. Programming assistants will be extremely helpful for code generation, especially if you aren't too familiar with `C++` programming but as explained in class they are only as effective as the design and prompts that are fed to them.

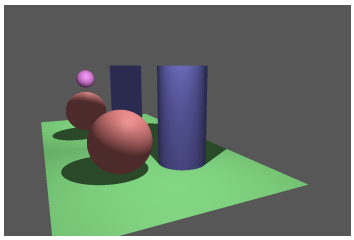
- we recommend familiarising yourself with the high level components that make up a raytracer.
- Make a high level plan of how your program will synchronise the components.
- Try to modularise your design for easy testing and flexibility (in case you want to implement advanced features).
- If you have implemented many of the advanced features (but not the acceleration hierarchy for example) you may find that your rendering could be slow.
- A good way to improve the efficiency of your code is using compiler optimisations such as the -03 flag or taking advantage of multi-threading with openMP! Simply import the openMP library and write `#pragma omp parallel` in front of your main loop.
- If you find yourself needing more samples than your computer can handle in a reasonable amount of time for your path tracer, try importance sampling.
- If you'd like to have custom shapes in your scenes to show off some aspects of your work, consider preparing the model in Blender and then writing a Blender script that translates that object into the JSON format your raytraces is using. Hint: Most meshes can be represented as a series of triangles.
- You may find it inconvenient to work with `ppm` images in Windows, there exist VSCode extensions that are very helpful for this.
- Your raytraced images of the scenes in the test suite should look like this:



binary_primitives.png



simple_phong.png



mirror_image.png

6 Plagiarism

Automagic is very concerned that they will be sued for infringement. This coursework needs to be your original work, up to sections provided by programming assistants. You are *not* allowed to directly import code from online tutorials, most notably the Raytracing in One Weekend series. Your code will be checked for similarities with publicly available codebases such as RTiOW. A high degree of similarity will lead to further investigations.

You may test the similarity (against RTioW) yourself by using JPlag. Simply follow the instructions provided on the github page! If you are concerned that the reported similarity is high, please check with the instructor immediately.