Automagic's Raytracing Renderer
Computer Graphics: Rendering, October 2023
Coursework 2

Sean Memery, Kartic Subr

1 Overview

Great news, *Automagic* were very impressed by your previous work and have hired you as a creative consultant! Their next task for you is to create the core renderer for their augmented reality software. In this assignment you are expected to develop code for a full raytracing renderer along with more advanced features if you can. But this is an AI company, why live in the past! You've decided to make use of new technologies like ChatGPT and similar LLM resources to construct your raytracer. *Automagic* want a full report on your work, including how you interact with the LLM resources, so they can learn how to best make use of these modern tools.

Your full expected contribution is as follows: A basic raytracer that implements Blinn-Phong rendering, a report outlining the steps taken to implement your raytracer including the LLM queries and responses, and a video showcasing your renderer on a custom scene. On top of this, there are some more advanced features that *Automagic* think would be a good inclusion, but aren't required to implement for a basic submission.

2 Specifications

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.

2.1 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. Textures (on sphere, triangle, cylinder)
8. Tone mapping (linear)
9. Reflection
10. Refraction
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. Use at least one of the linked LLM resources (i.e. ChatGPT or Copilot).
4. Report your queries, responses and your modifications.
5. Recreate the rendered images of the example scenes (not pixel perfect).
6. 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.

2.2 Pathtracer

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

Features:

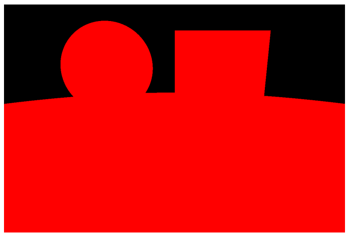
1. Antialiasing via multi-sampling pixels
2. Defocus in finite-aperture cameras by sampling the camera's aperture
3. Render materials with BRDF's (e.g. microfacet)
4. Soft shadows via sampling area lights
5. Multi-bounce path tracing

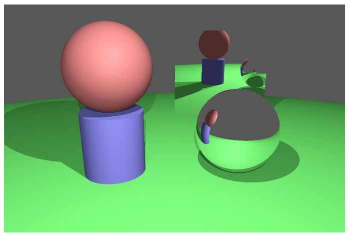
Constraints: Same as those for the raytracer.

2.3 Video

Goal: A video showcasing the abilities of your renderer, along with the input file used.
Constraints:

1. The video should be a minimum of 10 seconds long.
2. The camera should be in motion during the video.
3. The scene used for the video should be *interesting* i.e. it should include some features such as textures, moving shapes, moving lights, etc.
4. This should be in the .mp4 file format.
5. Render quality should be reasonable (resolution, sample count, frame-rate, etc.)


(a) Render Mode: **binary**


(b) Render Mode: **phong**

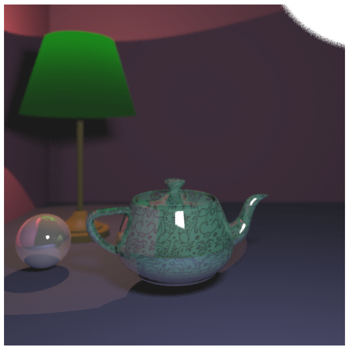

(c) Render Mode: **pathtracer**

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 demonstrated** 35
   - (a) Camera (5)
   - (b) Intersection tests (5)
   - (c) Blinn-Phong shading (5)
   - (d) Shadows (5)
   - (e) Tone mapping (5)
   - (f) Reflection (5)
   - (g) Refraction (5)
2. **Intermediate raytracer features** 20
   - (a) Textures (10)
   - (b) Acceleration hierarchy (10)
3. **Pathtracer** 20
   - (a) Pixel sampling (5)
   - (b) Lens sampling (5)
   - (c) BRDF sampling (5)
   - (d) Light sampling (5)
4. **Report** 15
   - (a) Formatted LLM queries (10)
   - (b) Explanation of code assembly for each features (total 5)
5. **Video** 10
   - (a) Conception and screenplay (2)
   - (b) Modelling (4)
   - (c) Rendering (4)

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 create your scenes. Indeed, we encourage you to discuss and brainstorm general concepts pertaining to this coursework or about the use of relevant tools.

The marking for each feature includes marks for querying the LLM, adapting/integrating its response in your code, debugging it and finally for demonstrating the feature. In addition, there are marks for explaining this in the report.

If you execute the basic raytracer well, along with the report and video, you can expect a mark of about 50-60%+. The expected time for students who have attended lectures and tutorials to attain this is about **24h**. The intermediate features are expected to take about **10h** and can allow you to score a total of 70-80% 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 path tracer 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 path tracer.

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
├─ Video
│  └─ video.mp4
├─ Report
│  └─ s123456.pdf
└─ LLM-responses
   └─ LLM-responses.json
```

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 files provided as part of the test suite provided. For scene.json, include both the images (binary and phong modes).

Video:
The `Video` folder should only contain your video file. The submitted video should be at least 10 seconds long and depict a moving camera traversing a custom scene. Part of your mark is the quality of the scene (i.e. the showcased features and creative aspects) while most of the marks for this

section is for the quality of the rendering of the video. The submitted video file should be a single `mp4` file made by combining many `ppm` images outputted by your program, see here for a possible method of creating the `mp4` file from `ppm` images.

Report:
The folder `Report` should contain only one file, which will use your student ID as your name. 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. The report should be in `.pdf` format (we recommend writing with LaTeX), and explain your work, step by step, with inline images. Figures should be numbered, annotated, referenced and clearly visible. How you make use of the LLM responses in this work is very important and should be included in the report, as they may need to be augmented to fit into your program. 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 6 - Shadows) and **an explanation of how and why you made any changes to the original LLM output**. The LLM responses will be included in a separate file for your submission, a `JSON` document with entries for each feature of your raytracer, outlined below.

LLM-responses:
The `LLM-responses` folder should contain a single file recording the responses of your chosen LLM method. The LLM responses are very important for this work and are to be included in their own `JSON` document containing a record of each implemented feature, containing five entries in each:

1. `"feature"`: The feature that was implemented by this LLM response, formatted using the numbers from the outline (e.g. for shadows this will be `"Basic Raytracer:6"`). It is very important to match this formatting.
2. `"method"`: The LLM method used to generate a response (i.e. `"chatgpt"` or `"copilot"`).
3. `"query"`: The exact text query used to generate a response (i.e. the input to ChatGPT or Copilot chat, or the comment used for Copilot auto-fill).
4. `"response"`: The exact response of the LLM, directly copied from the source with no changes made.
5. `"code"`: The location of the beginning and end of your final code for this feature, formatted like this: `"path/to/file.cpp:line_number_start-path/to/file.cpp:line_number_end"` (e.g. `"code/raytracer.cpp:10-code/raytracer.cpp:25"`)

5 Tips

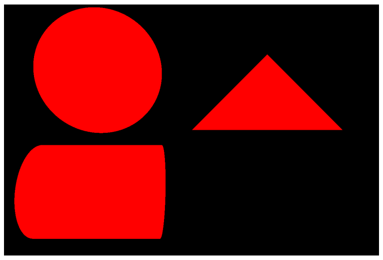
There are many moving pieces in a raytracer program and approaching it from scratch can be challenging. The LLM resources will be extremely helpful during this, especially if you aren't too familiar with `C++` programming, but hopefully you can make use of the following pieces of advice as well:

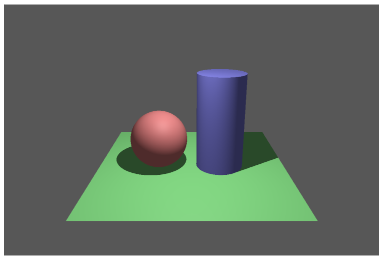
- Similar to what was discussed in the *Coding a raytracer* tutorial, we recommend familiarising yourself with the high level components that make up a raytracer and approach each one by one.

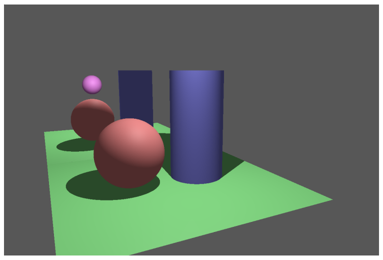
- Making a high level plan of how your program will execute will help you piece together the components.

- Be aware of optimisation, while not vital for this coursework it will help you while creating your video to able to iterate and improve quickly.

- There are many resources online for correct implementations of raytracers, if you doubt the output of the LLM resources, e.g. the Raytracing in One Weekend series.
- It can be very easy to created a highly coupled program when coding a raytracer, many different components need to communicate with eachother during rendering, so try to plan out your program with this is mind.
- The provided scene file is a great place to start when visualising what is needed in your program, on top of what is described in this document, as you can see every detail of the scene that your program needs to be able to handle.
- You may find it particularly difficult to build upon your basic raytracer to implement the advanced features, try to keep modularity in mind during development if you think you will attempt the advanced submission.
- If you have implemented many of the advanced features (but not the acceleration hierarchy for example) you may find that your rendering time is extremely high, making the creation of the video particularity difficult. We recommend making use of computing resources of the university if possible, keeping your scene as simple as required, or keeping your rendering resolution/sample count to a minimum level.
- You may find it inconvenient to work with `ppm` images in Windows, their 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