1/18/23[1]

Ray/Path tracing tracing assignment
COMP 371 – Winter 2023
Dr. Tiberiu Popa

Introduction

Ray tracing is a 3D rendering method that computes the color of each pixel by tracing a ray through the pixel, intersecting it with the scene and estimating the rendering equation at the intersection point. Ray tracers are widely used in the computer graphics. They are currently used for high-quality photorealistic rendering, in differential renderers, neural renderers and real-time ray tracers are currently supported by modern graphics hardware. There are plenty of resources for raytracing, including in your textbook. Some of my favorite ones are these[2][3][4] .

In this project you are asked to implement a complex ray tracer with global illumination support (i.e. path tracer) starting from a very basic provided code and only using the Eigen library. The command line takes one input argument, the name of a JSON file that contains information about the scene: geometry, cameras, lights, and other rendering parameters such as output file, resolution, etc. This assignment is individual.

Please read this document very carefully!

Json format

The entire scene as well as the output parameters are loaded using a "json" file format. Json format is a standard text-based format for representing structured data. You are given the code to parse a json file format and there are lots of example and documentation on the format online[5].

First, a few brief words on the json format. The json file format consists of a list of variable names and their values. For example (1):

```
{"name":"George", "age":21, "married":true}
```

Json format also supports more complex structures, or classes (2):

```
{"name":"George, "employment":{"employed":yes, "company":"UBISOFT",
"position":"software engineer", "years":5}}.
```

Json format also supports arrays:

[1] This document might change throughout the term, please check moodle periodically and download the latest version.
[2] https://raytracing.github.io/books/RayTracingInOneWeekend.html
[3] https://developer.nvidia.com/discover/ray-tracing
[4] https://github.com/dannyfritz/awesome-ray-tracing
[5] https://www.w3schools.com/js/js_json_datatypes.asp

```
{“name”:”George, "cars":["Ferarri", "Lamborghini", "Porsche"], "children":[
{"name":Nicoletta, "age":23},{"name":"Alberto","age":21}]}
```

The external/*json.hpp* contains an external implementation of the json parser[6]. An example is provided in the code base, and it will be covered during the tutorial.

Specifications

The ray tracing specifications are done using three array variables: *geometry, light, output.* The *geometry* variable describes the geometry in the scene, the *light* variable describes the light sources in the scene, and the *output* variable describes the output parameters (i.e. cameras, filenames, resolutions, etc.). Each of these variables have additional members as described in the following table:

Name	**Mandatory members:**	**Optional members**
geometry	type, ka, kd, ks, pc, ac, dc, sc	transform
light	type, id, is	transform areasampling
output	filename, size, fov, up, lookat, ai, bkc	raysperpixel, antialiasing, twosiderender, globalillum

Notes:

- Mandatory to have at least one geometry, one light and one output
- Each variable may have more mandatory members based on their specific type as described in the table below:

Type variable value	**When occurs in**	**Mandatory members**	**Notes**
sphere	geometry	radius, centre	
rectangle	geometry	p1, p2, p3, p4	The 4 corners of the rectangle in counterclockwise order with respect to the normal. The 4 points are assumed to be coplanar.
point	light	centre	Denotes a point area source. The centre variable denotes its 3D position.
area	light	p1, p2, p3, p4	Denotes an area light source (i.e. rectangle shape) – it uses the same conventions and specifications as for the rectangle geometry type.

[6] https://github.com/nlohmann/json

The following table contains descriptions of the rest of the variables:

Name	Type	Notes
type	string	See table above for values and other dependencies
centre	Array or 3 floating point values	Represents a point in 3D space
transform	Array of 16 floating point values	Represents a 4 by 4 affine transformation matrix in row major representation.
size	Array of 2 unsigned int values	Used in the output to specify the resolution of the image.
up	Array or 3 floating point values	The up vector of the camera. You may not assume that it is unit length. It is not necessarily perpendicular to the lookat vector.
lookat	Array or 3 floating point values	The lookat vector of the camera. You may not assume that it is unit length.
filename	std::string	A ppm file for the ray tracer output
antialiasing	boolean	use or not antialiasing. If raysperpixel is specified, use it for antialiasing. If raysperpixel is not specified, make your own decision on how many rays to send
raysperpixel	Array of one, two or three unsigned int values	For antialiasing and global illumination. If there is only one value, it represents how many rays per pixel to use using a uniform random sampling. If there are two values (a, b) you need to use stratified sampling using a grid of a by a and b uniform random samples for each grid cell. If three numbers are specified (a, b, c) you need to use stratified sampling on a grid of a by b and c is the number of samples per cell (i.e. rays per strata).
areasampling	Array of one, two or three unsigned int values	Same interpretation as the raysperpixel, but to be used only when sampling a light area source and it is a property of the light.
radius	float	Radius of the sphere
twosiderender	boolean	When true you should assume that the light bounces the same way of it touches the front or back of the object. If no value is specified, the default value is true.
ai	Array or 3 floating point values between 0 and 1 representing a rgb color	Ambient intensity of the scene, it is specified in the output object
bkc	Array or 3 floating point values between 0 and 1 representing a rgb color	Background color of the output https://learnopengl.com/Lighting/Basic-Lighting
ac, dc, sc	Array or 3 floating point values between 0 and 1 representing a rgb color	Ambient, diffuse and specular reflection color of a light source.
ka, kd, ks	Floating point number between 0 and 1	Ambient, diffuse and specular reflection coefficient

pc	Floating point number >1	Phong coefficient
id, is	Array or 3 floating point values between 0 and 1 representing an rgb color	Diffuse and specular light intensity
fov	A floating point value representing the angle in degrees of the field of view	
globalillum	Boolean variable	When true render with global illumination (path tracing using monte carlo integration). You can assume only diffuse lighting. When false (default value) rendering using the full Phong shader. Use the rayperpixel variable as a strategy for the initial ray cast. If raysperpixel is not specified, you can make your own choices in terms of how many rays to send and what type of sampling to use.

Notes:

- All variable names are lower case.
- For your project, if you decide to do a ray-tracer extension, you can augment these tables to support more options.
- More examples will be provided in class.

Modes not tested in parallel:

- global illumination and specular lighting → if both specified use global illumination and ignore the specular lighting component
- global illumination and anti-aliasing → if both are specified ignore the antialiasing
- anti-aliasing and area light source → if both are specified ignore the anti-aliasing
-
-

Features

Your ray-tracer must support the following features:

1) Local illumination (i.e. Blinn-Phong shading)
2) Global illumination via path-tracing
3) Anti-aliasing using stratified random subpixel sampling
4) Two-sided rendering
5) Sphere and rectangle primitives

What is given

You are given a very basic C++ project built using CMake that compiles on the Linux machines in the lab. You are provided code to save an image file in ppm format and code for parsing a JSON file. The CMake configuration

file *CMakeLists.txt* also provides the include and link folders for Eigen and a simple example with Eigen is provided. The README.txt file provides instructions on how to make and run the code.

This codebase is part of the course code repository available publicly in github. A link to this repository will be posted. We strongly advise to use *git clone* command rather than simply download it to pull updates if necessary.

The ray-tracer codebase is in the *COMP371_RaytracerBase/code/* sub-folder.

The *external/* sub-folder contains: (1) the code to parse json files, (2) code to save an image in a simple ppm format simple and (3) some example code that shows how to use the given code as well as how to use the Eigen library.

The *external/* sub-folder contains a few sample scenes.

By default, if you run the code with no arguments it simply calls some simple functions that show how to save images and how to use Eigen. If a json file is given as the first argument, a simple json parsing code is called that extracts some of the information from the json file.
When you start building your solution, you will create a *src/* sub-folder where you can put as many C/C++ files as you wish. **This folder you will submit as your solution and all code in this folder must be completely original with no external resources.**

When you start building your solution you must follow the instructions in the README.txt file. These instructions will also be presented in the lab.

In brief:

1. Enable in the CMakeLists.txt the definition of `STUDENT_SOLUTION`
2. The entry point to your code is a file *src/RayTracer.h* that implements a *RayTracer class* with the following specifications:
    a. The constructor takes one input j, an instance of a `nlohmann::json` class. You should use this variable to extract the scene information. Examples are provided.
    b. It has a *run()* member function that, as the names says, runs the ray-tracer code on the scene extracted from j.

Your code must compile and run on the Linux computers in the lab.

Personal Computers

The teaching team does not have the bandwidth to support any personal computers, **therefor we only support the setup in the lab**. However, this course uses open source, highly cross platform tools that should run easily on any of the 3 major platforms: Linux, Mac and Windows. For Linux and Mac using any reasonable package manager will easily install all the required dependencies: C/C++ compiler, CMake and Eigen. On Mac we recommend using Mac Ports[7].

[7] https://www.macports.org/

If you are not using the setup provided in the lab, you must make sure that your solution builds and runs correctly on the lab machines so you should familiarize yourself with ssh. You can always log in with your ENCS credentials to login.encs.concordia.ca. The code provided compiles and run on this machine.

Road map

Milestone	Steps	Testing strategies
Learn and understand the codebase provided	Compiling and running the codebase following the README file and help from the teaching team. Learn its structure. (1 week)	Test the image save with your own example. Test the Eigen examples and understand the vectors and matrix structures in Eigen.
Loading the scene	Use the example given to read the json file. At the same time create your own classes to hold the scene info. Think about the relationships between these objects. Certain reading operations that are repeated (i.e. reading vectors, matrices) should be factored out of the main code and delivered by some helper classes. Encapsulation, inheritance, polymorphism, and code factorization concepts that you learned in your soft-eng classes will come in very handy in writing your ray tracer. (1 week)	Load the examples provided. This will not be enough, the scenes provided will not cover the entire spectrum of possible scenes. So, write your own scenes and see if you can
Produce an image using Ray-casting (intersecting rays from the camera to the scene)	Create your ray class. Instantiate rays for every pixel. Intersect with sphere first. Intersect with rectangles. (2 weeks)	Use the scenes provided together with the final images to determine if your output is correct. You re encourage to create your own scenes, some simpler and some more complicated. You can check with us to verify the output of your own scenes against our implementation.
Local illumination with point and area light sources (i.e. soft shadows)	Implement a simple Phong shader to compute the shading using local illumination (i.e. sending rays towards the light sources) (1 weeks)	
Global illumination using path-tracing	Monte-Carlo integration to compute the rendering equation (2 weeks)	
Anti-aliasing	Use multiple rays per pixel (0.5 week)	Visual debugging

Notes:

- If you chose a ray-tracer project obviously you should do these steps in about half the time in ordewr to leave room for your project objectives.

- This roadmap was created to provide you with some structure to your assignment; but individual parts can take longer or shorter depending on the student.
- You really should start working on it from day 1 (even before day 1 if possible).

Submission

You should develop your solution in the *src/* sub-folder. Your code will be called from the *main.cpp* and your implementation should match the interface provided. The README.txt file provides more specific details.

You need to submit the entire *src* folder, which should contain your solution as a zip file: "<studentid>.zip". You should use the "zip" command on the lab machines (i.e. not tar, rar or other archiving software). For instance: "zip studentid.zip *src"*. This will archive the entire *src* folder into a file *studentid.zip*. When unzipped: *unzip raytracer.zip*, the *src* folder will be created and unpacked at the current location. A common mistake would be do type: "zip studentid.zip *src/"*. This will pack the contents of the folder and not the folder itself. This is incorrect.

For this assignment you are also asked to add a scene that you created. You should name it <studentid>.json and add it to the src folder.

You then submit the zip file using the EAS system as Assignment 1 before the deadline. You can have as many submissions as you want, we will always consider the last one before the deadline. Therefore, make use of this feature and submit several times.

Following these instructions is very important as your code will be build and ran automatically on the lab machines. Failure to build and run according to the specifications may result in a mark of 0.

To avoid such a situation, we provide three additional test submission deadlines: **alpha**, **beta1** and **beta2**. These submissions are not graded, but some feedback will be provided to make sure you are on the right path (pun intended ☺ . The deadlines for these test submissions are firm and posted on moodle. You will submit the exact same way as the final submission (in EAS you can have as many submissions as you wish for a given deliverable).

The final deadline is posted on moodle. No extensions possible except for medical reasons and with a medical note. Any late submission will receive a grade of 0.

Evaluation

The evaluation of the ray tracer is done in 3 parts:

1. The code is automatically compiled and run on the lab machines on several unknown test files. **There is a time limit for each case**. If your code exceeds the time limit a grade of 0 will be given for that specific test file.
2. The instructor will evaluate the images (half of the grade).
3. During the final project demos, a few minutes will be reserved for a short Q&A about the results and about course material on ray tracing (half of the grade).

Debugging

We will give you a set of json files and corresponding images for you to test your code. These are not exhaustive tests so you should write your own. You are permitted to exchange json files with other students for testing purposes. You can also ask us to run the solution on your files, we will try our best to answer to this requests, but provide no guarantee.

Errors and bugs

The codebase of this ray tracer assignment as well as our solution evolves over time, and it may contain occasional bugs. If you feel that an example is incorrect or differs from yours, but you think yours is correct, please contact the teaching team.

Nevertheless, it is important to remember that the evaluation is not using a pixel-by-pixel comparisons, so reasonable variations of the posted solution may still receive full marks.

Originality and Plagiarism

A ray tracer is an important assignment in every university level computer graphics course in the entire world. Therefore, a simple google search will yield thousands of ray tracer implementations out there. Particularly, Peter Shirley's link[8] is excellent and will help you step by step to build your ray-tracer. **Please look at the method and do not copy his code, this assignment must be original and individual.** We run plagiarism tests and you are expected to explain your solution in detail to the instructor.

[8] https://raytracing.github.io/books/RayTracingInOneWeekend.html