ELEC-A7151 - Object oriented programming with C++

Path tracer - project document

Fall 2022

Aleksi Kääriäinen 728971

Jukka Aho 100496022

Xie Yuting 100878187

1. Overview

The goal of this project is to produce a software that renders a 3-dimensional image that simulates real-world illumination. The software is used via command line. When running the program, the program asks for two arguments: the path to the input file and the name of the output file. The input file should be a JSON-file that follows the format specified in the document. After reading the input file, the program renders the scene and outputs a ppm-file on exit.

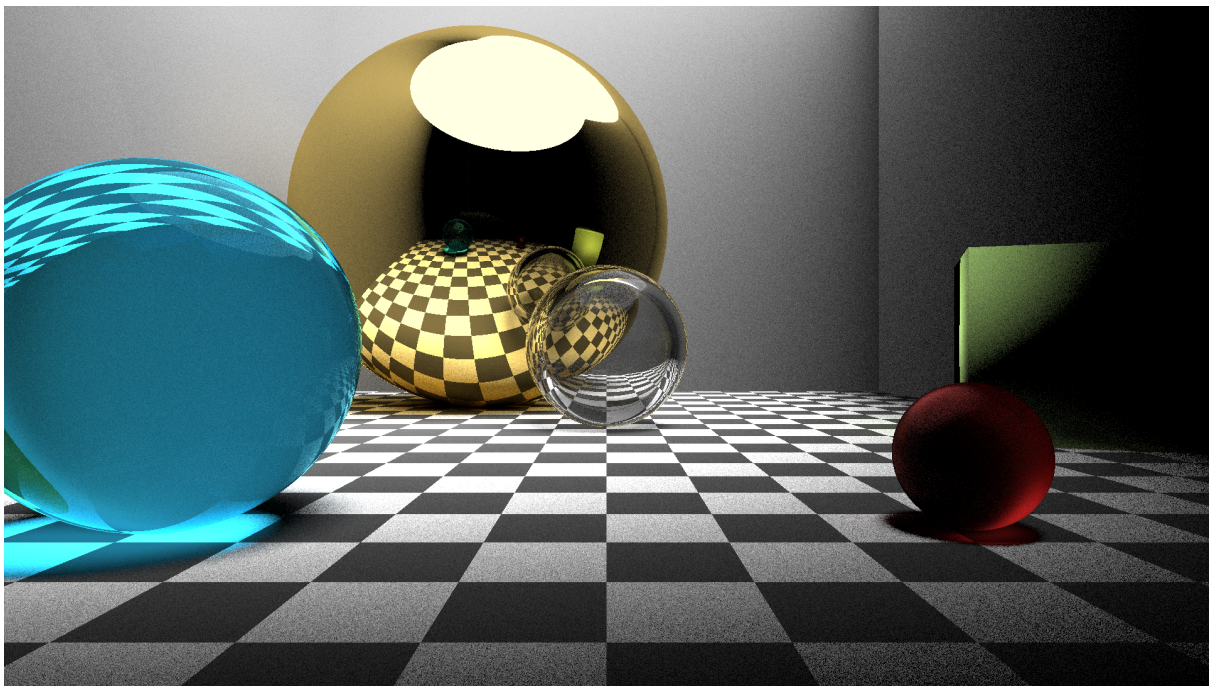

Figure 1. Example output image of software

The rendering algorithm is fairly simple. Contrary to a real-life camera, the light rays are shot from the camera, towards the image. A color is calculated, if and only if the ray shot from the camera hits the light source. Otherwise, the background color is returned. This way, an object is only rendered into the image if light rays deflect from the object into the light-source, resulting in a realistic illumination and shading. A form of anti-aliasing is implemented to create clearer images with fewer jagged edges. The amount of anti-aliasing is controllable by the user. The anti-aliasing functions by shooting multiple rays through each pixel, and then averaging the resulting color value by the number of rays shot through the pixel.

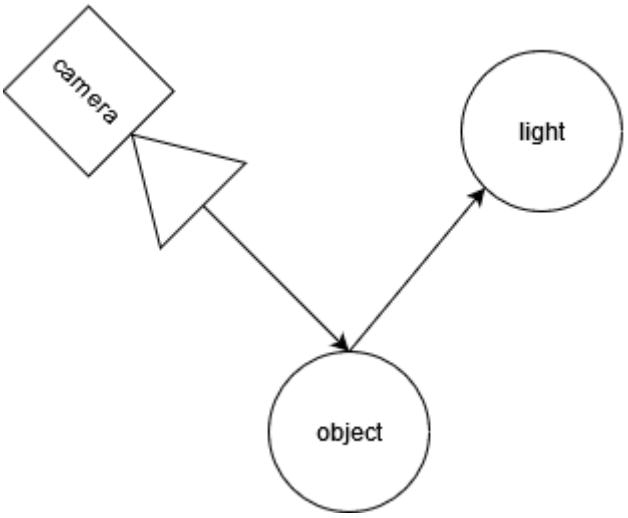

Figure 2. Visualization of algorithm

The objects the path tracer supports are physical geometries: spheres, boxes, and planes. These geometries can be rotated and given a specific size in the vector space in addition to its origin. Additionally, each geometry is colored, and is made of a single material.

Three materials were implemented in this project: lambertian, metal and glass. They each refract and scatter light rays differently. Additionally, objects can be assigned to emit light.

2. Software Structure

The main structure of the program is described in the UML-chart below.

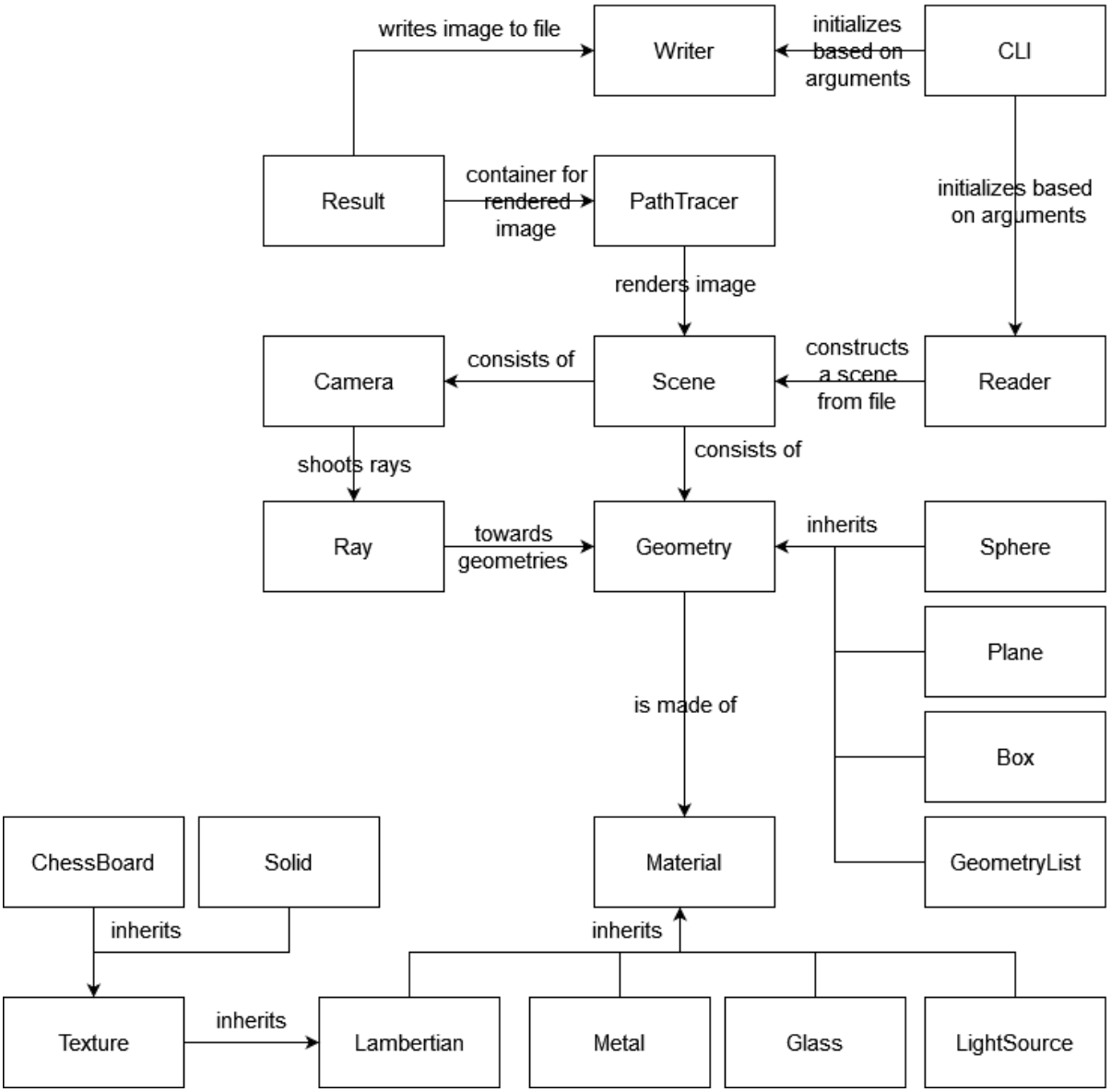

Figure 3. UML-chart depicting the structure of the software

The project was implemented as a header library. All source files are header (.hpp) files, and a library header file including all source files is located in the root of /include.

The C++ standard library is used extensively. Shared pointers were utilized to minimize the amount of memory management the developers have to work with. Containers, IO-functionality, and used constants were all from the standard library. Eigen3 was used mainly for its ready-made implementation of vectors and matrices. JSON for Modern C++ was used to parse the input JSON-files into a scene object. This proved to be quite easy to integrate into our project, since it is a single header file, and is included in the project by simply adding the file to the project. Additionally, CMake was used for linking the source files and libraries, and producing the executable files, such as the main program, unit tests, and examples that produce an output file. Finally, Doxygen was used to generate source-code documentation. The documentation is available at https://tracey-docs.netlify.app/.

3. Instructions for building and using the software

Install following libraries:

- CMake
- Eigen
- Doxygen for documentation
- clang-format for code formatting

Run following command to build the makefile

cmake -S . -B build

cmake --build build

...

4. Testing

There are five different test cpp for separate functions, including test_camera, test_cli, test_file_writer,test_from_json and test_material. Each of them could run independently, in case that the newly updated functions have an unexpected impact on one another. The selected targets can be independently run by selecting the proper target and pressing F7 in VS Code. And the majority of those testing was verifying visually that the output ppm image is appropriate.

5. Work log

Week	**Aleksi**		**Jukka**		**Xie**	
45	Project plan	5h	Project plan	5h	Project plan	5h
46	Geometry, Sphere, FileWriter	20h	Setting up cmake, camera implementation	20h	Ray, color, testing, Antialiasing	20h
47	Plane, Box	15h	Setting up Doxygen, Setting up Eigen	15h	diffuse, specular, and transparent materials	20h
48	FileReader, test renders	20h	Parallelized rendering	20h	Basic texture, Point light	15h
49						