CPP Project plan: Path tracer

Important Dates:

- ~~**Kick-off session: Wednesday, October 11, 2023 at 12:15**~~
- ~~**Project topic/group questionnaire (deadline): Friday, October 13, 2023 at 17:59**~~
- ~~**Project group distribution and confirmation: Wednesday, October 18, 2023 at 23:59**~~
- **Project plan submission (commit) to git (deadline): Friday, October 27, 2023 at 23:59**
- **Project plan review with your advisor: Week 44 (starting Monday, October 30)**
- **Project demo to your advisor: Weeks 49 and 50 (starting Monday, December 4)**
- **Project final commit to git (deadline): Sunday, December 10, 2023 at 23:59**
- **Project peer evaluation (deadline): Friday, December 15, 2023 at 23:59**

1. Scope of the work: what features and functionalities will be implemented, how is the program used, and how does it work

What features and functionalities will be implemented?

- Simple GUI: Our simple GUI will allow the user to choose an input file/files containing a geometric description of the environment and objects, specify parameters such as resolution, FOV, number of Monte Carlo samples etc. The GUI will contain a run-functionality, which will run the path tracing algorithm and finally output the obtained scene on the screen. In addition to displaying the output in the GUI, the user will be able to choose a file path where the output will be stored in a .png format.
- Proprietary input file format: We will first implement a proprietary input file format that will at least contain specifications for spheres and area lights (luminous spheres) in the scene.
- (Support for a standard triangle mesh format: After implementing the proprietary input format, we will implement functionality that will be able to read standard triangle meshes from a .obj file.)
- Output image file format: The output image will be stored in a .png file format.
- Geometric objects: The proprietary input file format will contain only spheres of different positions, sizes and materials. The standard triangle mesh format will be

able to represent any geometry. In any case, our path tracer will not be limited by the number of objects in the system.
- Light sources: Area lights in the form of spheres.
- Material models: We will implement at least diffuse and specular materials that can have different colours.
- Surface texture: Spheres can have a surface texture, which is specified in the proprietary input format.
- Multi-path light reflections.
- Monte Carlo integration: To make higher-quality images.
- Parallelization to multiple CPU cores using OpenMP.
- Geometry acceleration data structure: Once we implement triangle mesh support, we will also optimise the geometry back end using bounding volume hierarchy for example.

How is the program used?

- The front-end will run on a GUI based on DearImgui & OpenGL.
- The user can choose to load a file from a file explorer prompt.
- After the file has been loaded, the user can render the image by pressing the corresponding button in the GUI environment.
- The image is generated by our custom back-end and then displayed in the GUI. The user can save the image if they so wish.
- The GUI allows for the user to dictate the parameters of the render engine. For example, resolution and aspect ratio can be defined by the user.
- Additionally, camera position and some other simulation features (such as the position of the point light) can be adjusted in the GUI.

How does it work?

- When the user has loaded a data file and presses ‘render’, the GUI sends a call to our custom back-end along with the scene data
- The back-end uses path tracing to calculate the final color value for each pixel
- The resulting image is stored as a temporary file and displayed by the GUI
- The user can then choose to save the file which copies the image file into another location

2. The high-level structure of the software: main modules, main classes (according to current understanding)

Main modules:

Renderer: Handles the overall rendering process.

The renderer is the orchestration class. It coordinates the whole process.

1. It starts by asking the camera for a ray for the current pixel.
2. The ray is then passed to the scene to find intersections.
3. Once a hit is found, the renderer gathers the colour contribution from that hit using the object's material.
4. When path tracing is implemented, the renderer shoots additional rays and accumulates their contributions too.
5. The final colour is then set for the current pixel.

Scene: Manages scene objects, lights, and materials.

The scene contains all objects and light sources to be rendered.

1. When a ray enters the scene, the scene checks for intersections with all objects.
2. If multiple objects are hit, it'll determine the closest one (the first object the ray hits when moving forward from its origin).

Camera: Defines the viewpoint and settings for the simulation.

The camera is the viewer's eye into the scene. It defines from where and at which angle the scene will be rendered.

1. When the rendering process begins, for each pixel on the screen (or designated output resolution), the camera generates a ray. This ray represents a line of sight from the camera's position into the scene.
2. The generation of a ray will depend on the camera's properties: its position, orientation, field of view, and possibly depth of field (for effects like blur).

Ray: Represents a ray with an origin and direction.

A ray is a mathematical representation of a line with an origin (start point) and direction. In the context of path tracing:

1. The ray gets passed into the scene to check what it intersects.
2. If it hits an object, it'll gather information about that hit, such as the point of intersection, the normal at that point, and the material of the object it hit.

Material: Defines how rays interact with surfaces.

The material of an object dictates how light interacts with it.

1. Once an intersection point is known, the material's shading method is invoked to determine the color contribution from that hit point.
2. Different material properties can cause the ray to reflect, refract, or simply gather light. This might spawn new rays, starting the process again. For instance, a reflective material would spawn a new reflected ray, which then gets checked against the scene for further intersections.

Math: Contains mathematical utilities, like vectors, matrices, and intersection algorithms.

UI Module: Manages user interfaces, inputs, and outputs, using Dear ImGui for real-time control and feedback

Main classes:

Renderer: Orchestrates the rendering process, initiating the path tracing for each pixel.

Ray: Represents a ray with an origin and a direction, used for intersection tests and light transport calculations.

Scene: Contains and manages all objects and lights in the scene. Handles queries for intersections and lighting information.

Camera: Defines the viewpoint, orientation, and lens attributes for rendering the scene. Generates primary rays.

Object: An abstract class that represents a 3D object, with methods for intersection tests.

Sphere, Plane, Box: Derived classes from Object, representing specific geometric shapes and their unique intersection calculations.

Material: A class that defines the optical properties of a surface, including colour, reflectivity, transparency, and potentially texture maps.

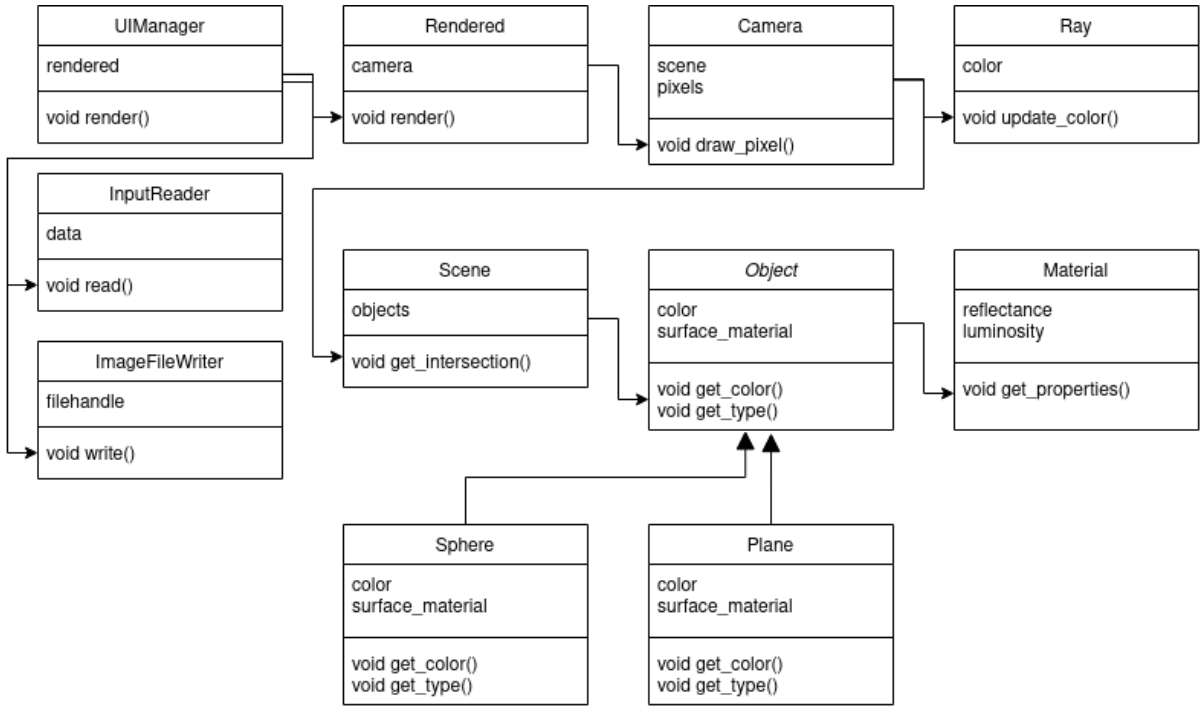
Light: An abstract base class for different light types, with general attributes like colour and intensity.

PointLight, DirectionalLight, SpotLight: Specific light types derived from Light, each with unique characteristics and effects on the scene.

Math: Includes common math operations not provided by standard libraries, or specific rendering calculations.

Random: Handles the generation of random numbers for techniques like Monte Carlo sampling.

UIManager: Handles the creation of UI elements, processes user inputs, and updates the rendering parameters or scene data accordingly.



https://drive.google.com/file/d/1goAb68M_QCvZS9TGAtRqFQQeVDwHcqRM/view?usp=sharing

The planned use of external libraries

Dear Imgui+OpenGL: For the Graphical User Interface, displaying the rendered images and adjusting the parameters dynamically. Not used in path tracing rendering - our custom back-end is kept sanitary.

Division of work and responsibilities between the group members

Aleksanteri: GUI environment

Mikael: Proprietary input format and routines for reading it

All: Read path tracing tutorials

Planned schedule and milestones before the final deadline of the project

3.11. Base GUI environment deployed. Proprietary input format and reader routines developed.