Ray Tracing Rendering

November 29, 2018

The objective of this project is to understand and practice the main elements of a ray tracing engine and then to implement an acceleration structure for the ray/scene intersection calculation, as well as to manage textures.

1 The Environment of the Project

1.1 Setup, Directory Structure and Configuration

In the *SythesesImage* folder, contains the following directories:

- `dependencies ima`: This directory contains libraries pre-compiled for the project: `SOIL, lib3ds, tbb, SDL ...`
- `Models`: This directory contains a set of sample models in 3DS format.
- `RayCasting`: This directory contains the sources of the ray tracing project. The sub-directories are organized as follows:
  - `Doxygen`: This directory contains the configuration file used to generate documentation in doxygen format.
  - `src`: This directory contains the source codes of the project. They are organized in such a fashion that each sub-directory corresponds to a namespace and that each file is named after the class it contains. If .cpp files are associated with .h files, they are in a src subdirectory.
  - `project 201x`: These directories correspond to various projects compatible with different versions of Visual Studio IDE, though the 2017 version is recommended.

You will need to copy the `RayCasting` and `Models` directories and place them in the same directory in order to undertake your project.

Attention: Before working on the Visual Studio project, you should run the script: `setup.bat` located in the `dependencies ima` directory for setting environment variables used in project configurations.

1.2 Main Files and Starting Points

`main.cpp`: In the src directory of the project, you will find the main.cpp file. This file allows you to configure the rendering by providing the scene to calculate (see the name of the included methods in the next section), the number of bounces to take into account when calculating reflections, the size of the rendering window, and so on. The configuration of all these settings are carried out the main function of the program.

You are advised to use a small rendering window during the debugging phases in order to limit calculation times while being able to observe the results, especially when you consider indirect lighting. Three sizes are available in the main function: 1000x1000, 500x500 and

300x300. Attention, you must have square rendering windows, the management of the form factor is not implemented at the moment.

`Geometry/scene.h`. The `Geometry::Scene` class is the starting point for the rendering engine. It proposes a method `compute` which calculates the rendering of the whole scene. The latter takes three parameters:

- `maxDepth`: is the maximum number of ray rebounds to consider.
- `subPixelSubdivision`: is the square root of the number of primary ray to be traced per-pixel (this corresponds to antialiasing).
- `passPerPixel`: is the number of rendering passes to apply per pixel. This parameter will not be useful unless you implement a Monte-Carlo rendering method. Otherwise, assign a value of 1.

The number of primary rays traced per-pixel equals $subPixelSubdivision^2 * passPerPixel$.

The `sendRay` method is called once by the `compute` per primary ray. This method is designed to return the color of the pixel associated with this primary ray. It takes 5 parameters:

- `ray:` The ray which you wish to perform your calculation.
- `depth:` The current depth for recursion (number of rebounds of the ray since the corresponding primary ray has been casted).
- `maxDepth:` The maximum recursion depth (maximum number of rebounds of the ray)
- `diffuseSamples:` Only useful when implementing the Monte Carlo method, to know the number of ray to cast for the estimation process of the indirect diffuse component.
- `specularSamples:` Only useful when implementing the Monte Carlo method, to know the number of ray to cast for the estimation process of the indirect diffuse component.

As you can see, this method is empty. This is the very starting point for this project and the method that you are going to complete. Attention, during this project, think about the functional decoupling, this is important if you want to find in your different calculations.

1.3 The provided scene

The table below summarizes the main features of the scenes that are offered in the application (they are initialized in the main function). The first column is the name of the method to call to initialize the corresponding scene:

2 The steps of the project

In this section, provided a set of steps to follow to complete your project. Parts a) and b) are mandatory (this is the basic algorithm for ray tracing and accelerating structure). Part c) is an extension that you are advised to program.

2.1 The algorithm of ray tracing

When implementing your algorithm, it is strongly recommended to think about functional division. A proposal is the following: a function to calculate the direct lighting, a function to calculate the indirect lighting and a function to check if a point is illuminated or not by a point light source. This division is not exhaustive but will allow you to better locate your code.

Initialisation function of the scene	Type of scene	Number of triangles	Usage of textures	Light weighted	Surface light
initDiffuse	The Cornell box (diffused)	36	No	Yes	No
initSpecular	The Cornell box (specular)	36	No	Yes	No
initDiffuseSpecular	The Cornell box (diffused & specular)	36	No	Yes	No
initGuitar	A guitar	45399	No	Yes	No
initDog	A dog toy	132034	No	Yes	No
initGarage	A garage	219152	No	Yes	No
initTemple	A temple	603111	No	Yes	Yes
initRobot	A robot	126944	No	Yes	No
initGraveStone	A tomb	540902	Yes	Yes	No
initBoat	A boat	389554	No	Yes	No
initSombrero	A Sombrero	2024	Yes	Yes	No
initTibetHouse	A Tibet house	22251	Yes	Yes	Yes
initTibetHouseInside	A Tibet house	22251	Yes	Yes	Yes
initMedievalCity	A medieval city	774644	Yes	Yes	No

Figure 1: Parameters for the different scene.

3 Supplement of the code and the problem

We provide a nearly complete ray tracing project which have all the functions except the $Scene :: sendRay()$ which you should implement. The framework enables the loading of a .3DS file, launching rays for all the pixels of the screen of the camera (including the subsampling). Also, classes to compute the intersection of a ray and a triangle (all .3DS files are only triangles) are provided.

We divided the project into three stages. The stage #1 is essential and the stage #2 is enough for the full mark. If you are interested in this problem, we recommend you to have a try on the stage #3.

3.1 Task #1: Basic Ray tracing Realization

To make you familiar with the code faster, we suggest you to finish this part along the following instructions:

1) initialize the project with the Cornell box scene initDiffuse (see main.cpp)
2) implement sendRay() so it returns the diffuse color of the intersected triangle
3) implement diffuse lighting (without shadows)
4) integrate shadows in the computation
5) switch to the project with CornellBox scene initDiffuseSpecular (some faces are now mirrors)
6) integrate specular reflexions
7) integrate emmisse materials (none in the example)
8) integrate ray bounces (using depth and depthmax)

3.2 Task #2: Speed up the tracing

After task #1, you have got a nice but slow work for the ray tracing problem. Thus, you are asked to optimize your algorithm to make it faster.

You can apply any methods to achieve this, but one of them is required. You are asked to use KD-trees of AABB trees to avoid all triangles for intersection (principle is to store triangles in volumes, and cast the rays on the volumes – if hit then only test the triangles inside the volume. Intersection between a volume and a ray is provided in class Bounding Box.)

3.3 Task #3 (optional): Better and better

To achieve a realistically rendering, you have a long way to go. One of the problem you have to face is that your current result is not very clean since you don't do interpolation between normals. To smooth the normals on the surface, you could use normal interpolation in class Triangle. Then render with textures.

4 Evaluation Metric

Your rendering image is the key point to evaluate your score. Also we will check your code to prevent the plagiarism.

The grades of all 3 tasks is distributed as follows:

- Task 1, 60%
- Task 2, 30%
- Task 3, 20%
- In addition, the report contains 10%.

We need to remind you the score is not the objective of this very experiment but the process of dealing with the problem and searching for your own way to solve it is.

Suggestion : Search engines and group communication is encouraged during this project however **copy is forbidden**.