CSE306 - Assigment I

Deivis Banys

1 Architecture

My whole project is distributed into the following parts:

- `Functions`- a folder of used functions. It contains two main functions, where the first one is called `FunctionsSmall.h` and it consists of some "small" functions (e.g. function that performs gamma correction or clips pixels such that they stay within range $[0, 255]$). The second one is called `FunctionsScene.h` and consists of functions that are independent from any classes/structures, but are used to build a scene (e.g. initialization of a scene, `randomCos`function, etc.)
- `Libraries` - a folder of used external libraries.
- `Images` - a folder used to gather generated images.
- `Models` - a folder of used objects of type `.obj` (a low poly cat mesh and a simple triangle used for testing).
- `Objects` - a folder of various objects (classes and structures) seen in the course (namely `Vector, Geometry`, `Intersection, Mesh, Ray, Scene, Sphere`) and gathered in different header files. Also, I introduced a structure `Image`, which consists of an image grid, width, height, etc.
- `main.cpp` - the main function.

Also, note that I did not separate header files into two separate ones (one for the definition of the structure/class and the functions and the other one for the implementations of those functions) as typically structures/classes do not have many functionalities and thus it should not be too hard to follow the code.

2 Diffuse and mirror surfaces, direct lighting and shadows for point light sources

First of all, I define function `initScene()`, which recreates the standard scene seen in lecture notes. It consists of walls, ground and a ceiling, which are are made from gigantic spheres approximating planes (Figure 1.). Then, I introduce functions that compute Ray-Sphere and Ray-Scene intersections as well as add shading and shadows (Figure 2.)

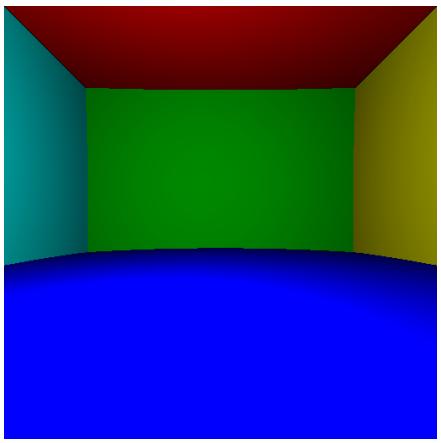

Figure 1: Empty scene

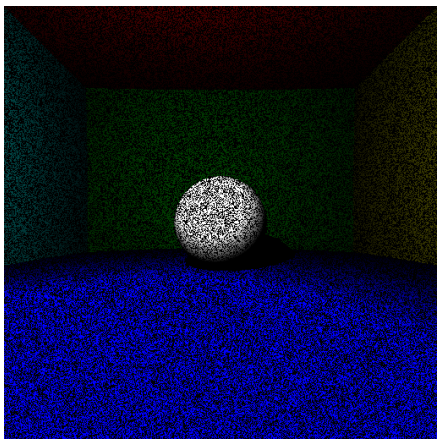

Figure 2: First scene with a sphere

At first image does not look realistic, as it appears pretty dark and with a lot of noise. To get rid of that, I firstly launch the ray from $P + \epsilon N$ rather than from P (Figure 3). Then, I apply gamma correction and we have the expected result (Figure 4.)

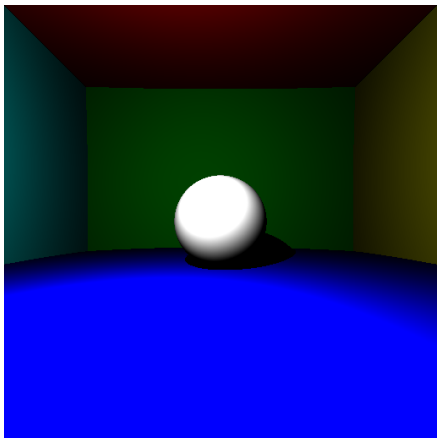

Figure 3: After changing ray's starting point

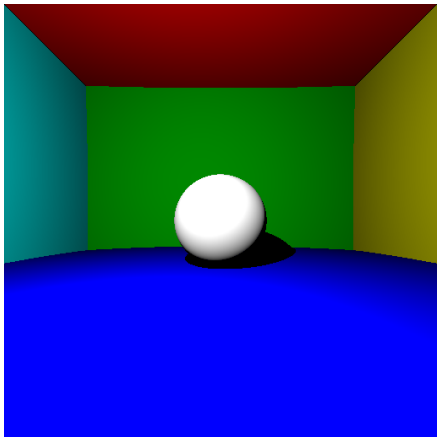

Figure 4: After gamma correction ($\gamma = 2.2$)

Lastly, I finish implementing the given functions in the lecture notes to obtain reflection functionality. The obtained results were rather realistic (Figures 5 and 6).

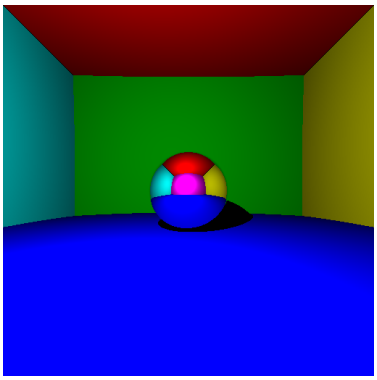

Figure 5: Reflective sphere

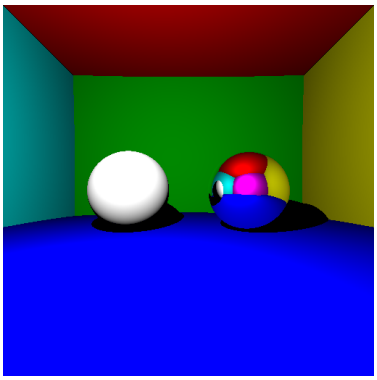

Figure 6: Diffuse and reflective spheres

Also, all the images shown above took ≤ 1 second to compute.

3 Indirect lighting for point light sources

Now, by modifying the `getColor` function, I added the indirect lighting for point light sources. The following results were observed with `ray_depth` = 10:

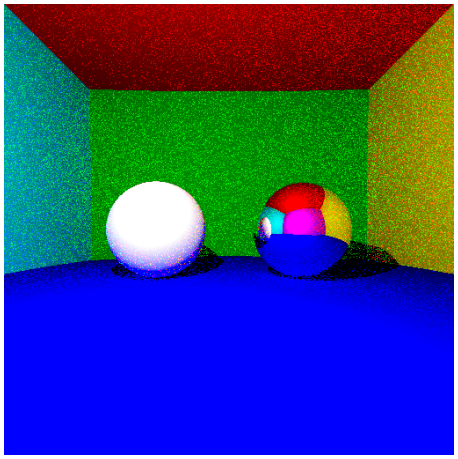

Figure 7: Indirect lightning (running time: $\sim 5s$)

4 Antialiasing

After adding antialiasing, we observe that images become much smoother (Figure 8 and 9) comparing to the images observed when only applying the indirect lightning.

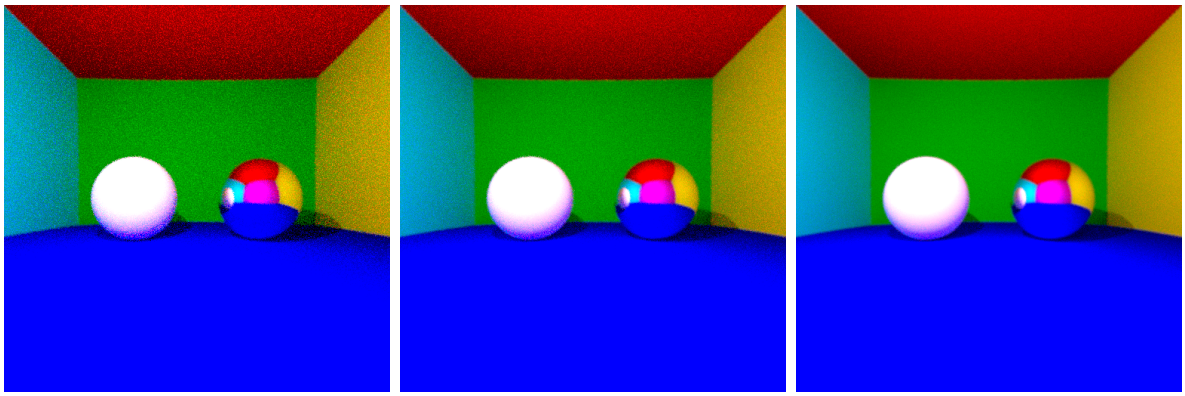

Figure 8: Aliasing with number of paths being 10, 30 and 100 (from the left to the right respectively)

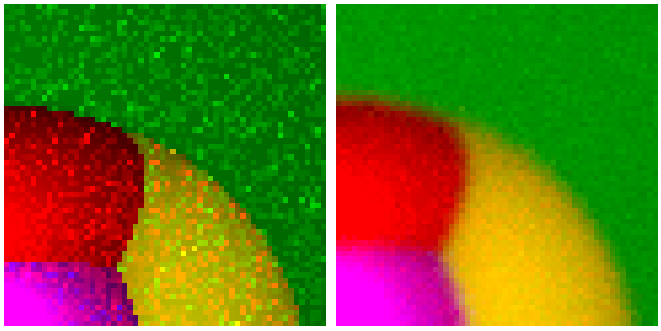

Figure 9: Before and after applying antialiasing

However, it highly affects the running times. The running times for renders with number of path being 10 (resp. 30 and 100) were $\sim$ 22s (resp. $\sim$ 66s and $\sim$ 232s) with parallelization.

5 Ray mesh intersection including BVH

For the Ray mesh intersection I implemented the Moller–Trumbore intersection algorithm. To do so, I designed a `Geometry` structure and inherited previously defined structures `Sphere` and `TriangleMesh` as well as made changes to my code such that now a structure `Scene` consists of pointers to `Geometry` objects. To test the functionality of the algorithm, I added a simple object `triangle.obj` so that I can more easily debug my code. I indeed made my code work on this example, however I did not have enough time to make it work on the `cat.obj` example.

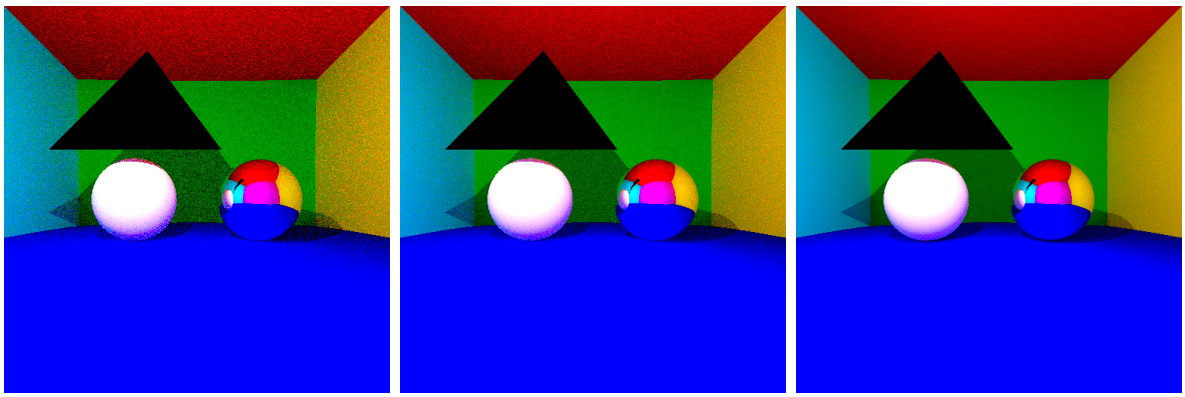

Figure 10: Moller–Trumbore intersection algorithm on one triangle

For the above images, the running times were $25s., 80s., 268s.$ respectively when using `ray_depth` $= 3$.