Ray Tracing : Report

Introduction

This project was create to the generate a ray tracing program that will generate a realism image of a scene. In this part we will explain how we decided to organize ourselves for this project.

Ideas

In the beginning we both were a bit lost, because we didn't have any clue of how to begin our project, so the first weeks we decided to see tutorials explaining the idea of ray tracing and how it's is used.

Afterwards we started looking for in formations of how we should organize our code, which classes a necessary and want our their purposes.

We started coding by following the YouTube tutorials but also by looking up projects in GitHub and only keeping the ideas that would serves us. This process was very difficult since we had get the idea of ray tracing but we still did not understand of majority of the algorithms that should be apply so we tried to understand codes that did not have many comments and were very long. For example https://www.scratchapixel.com/ had many lesson for the ray tracing but at the beginning we did not understand the program that well and it was very different of how we started writing our program.

The following days we started to understand better and be able to produce result but our program had a huge default we did not use matrix. In the end we still did not used because we had to change a lot of things and by a lack of time we had no choice but to leave of version of ray tracing without matrix.

At least we did not take much of a time to get our ideas then we decided to give ourselves tasks.

Tasks

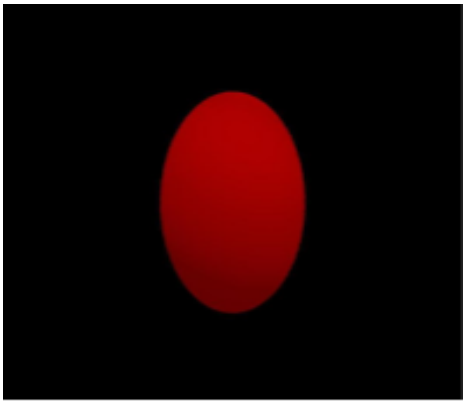
Firstly we decided to write our class *Parser* since it was the only part that did not really needed any notions of ray tracing only pure c++ programming followed by *StandardFigure.*

Then we created the basic classes *Vector* and *Color*. Followed by *Shape*, *Image, Light* and *Ray*. We started by testing if we could created a ppm file

and if the parser worked correctly before launching ourselves into the basics classes.

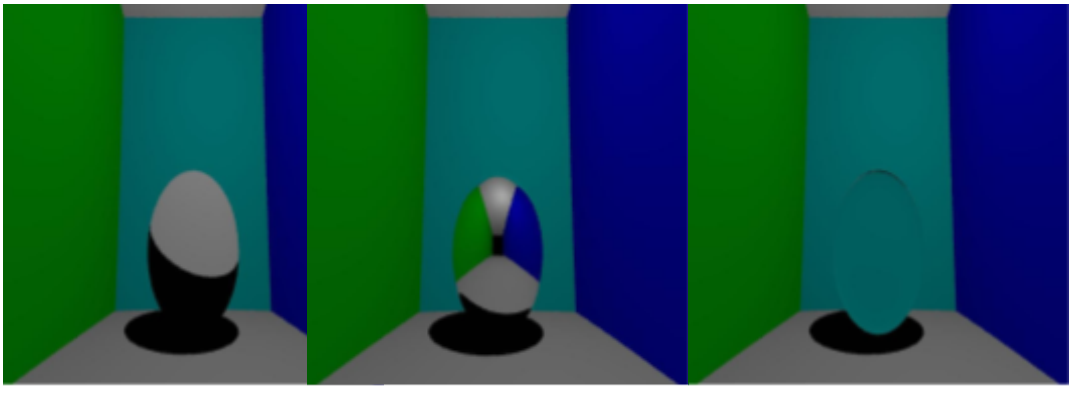
Afterwards we had our period where we started rewriting operator and writing the necessary method so we could actually test if we could create scenes.

This was the moment we created the *Sphere* class that implements *Shape,* the intersect method for the *Sphere* was not difficult to find since many people use the *Sphere* to explain how the ray tracing works and we finally had our first image.



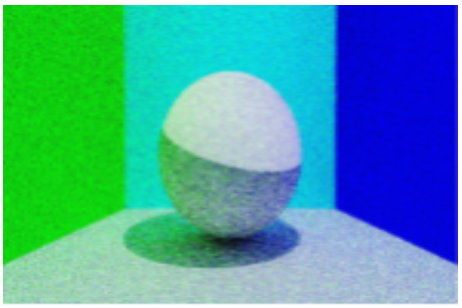
Sphere : Red

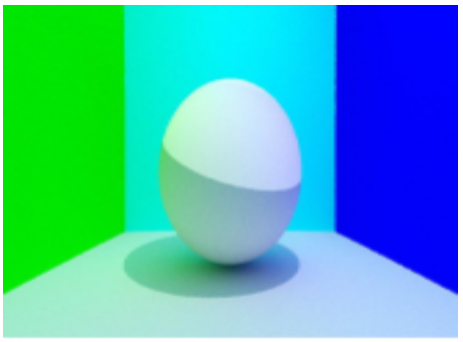
Before starting working on the other shapes we decided to do the maximum we could for this shape so we looked up not only the phong lighting but also the shadow and transparency and mirror algorithms.



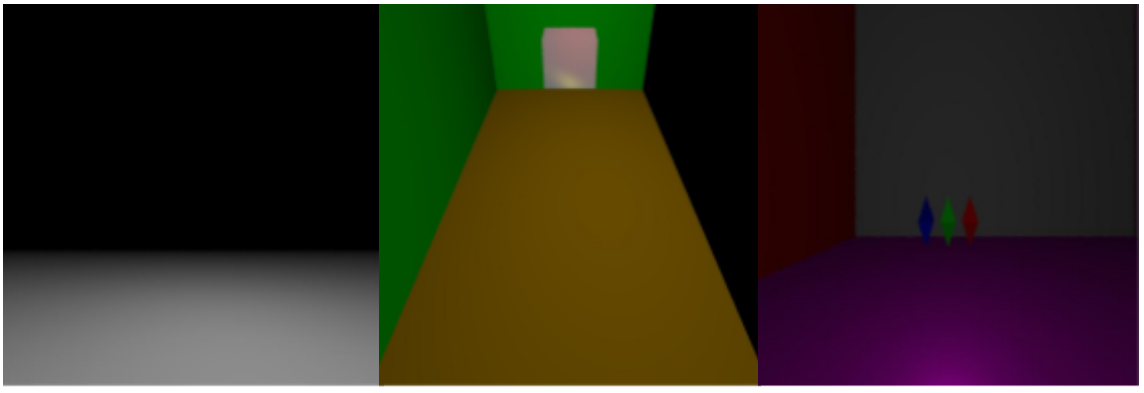
Sphere : shadow *Sphere : mirror* *Sphere : transparecy*

But we also decided to launch ourselves in the recursive interactive rays for at least the sphere.

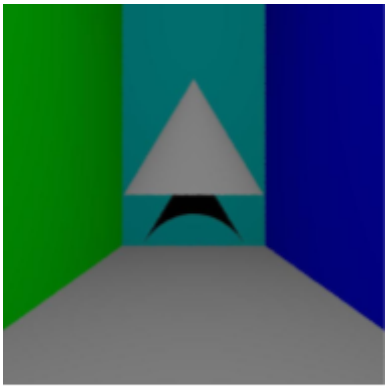




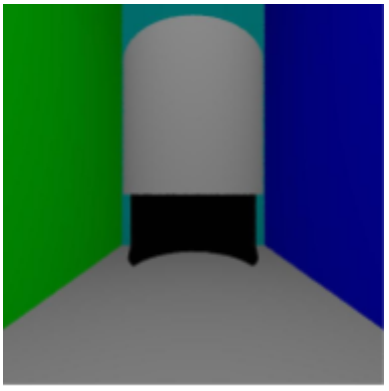
By completing every requirement with the sphere we decided to do the other shapes one by one.



Plane : White *Cube : diffuse and specular* *Triangle : With 2 light*



Cone : Shadow

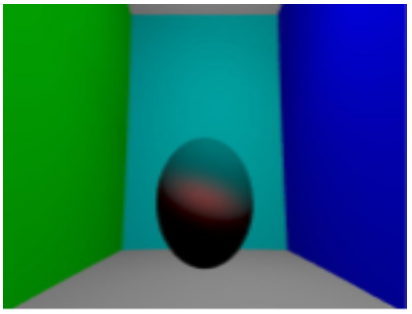


Cylinder : Shadow

Finally there was only the camera and the graphic window left. Since we did not understand how to work with opengl we decided to use sdl for this project. We spent a large amount of time trying to understand how we could place and change the viewpoint of the camera.

Remarks

Every shape has a material that is optional, the user can initialized of let the material by default further explained in the USER and DEV manual. But the 3 components ambiance, diffuse and specular each have their own color meaning if the color of one of them is different from the color object then by mixing itself with the object color the user will perceive a different color. For example:



Sphere : Manual ambiance, diffuse and specular

If the ambiance is black, the diffuse is cyan and the specular is red. The exponent will be 16 so it's very visible. Note the smaller the exponent the bigger the spot.

Bugs

More details in the USER and DEV manuals. The transparency for the cubes does not work for the cubes. The shadows are not perfect. If the command line is not written properly the program will not work correctly. The graphic window since it will render the image every time it can be very slow. The mirror and transparency will have shadows. The position of the camera cannot be 0 0 0 otherwise it cannot see the scene.

Conclusion

Even if we faced a lot difficulties all along the project and considering the situation of the year, we still were able to produce a satisfying result. We also found this project very interesting and fun to do.