Capucine COUDERT, Perrine PEYGOURDI, Ninon DELAMARRE-TRONEL,

SCAN group 73

Minimal Ray Tracing Report

1. Project Context and Objectives

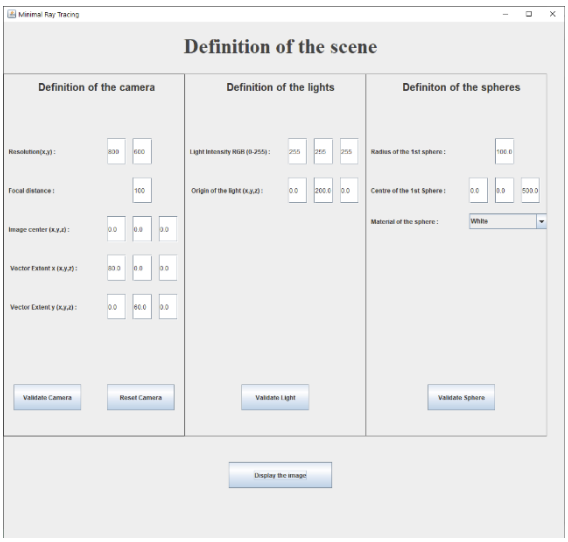
Ray tracing is a technique for calculating optics by computer, used for rendering in image synthesis or for studies of optical systems. It consists of simulating the reverse path of light: the lighting from the camera to the objects and then to the lights is calculated, whereas in reality, the light goes from the scene to the eye. This technique reproduces the physical phenomena (Fermat's principle of the inverse return of light, Snell-Descartes' laws) of reflection and refraction.

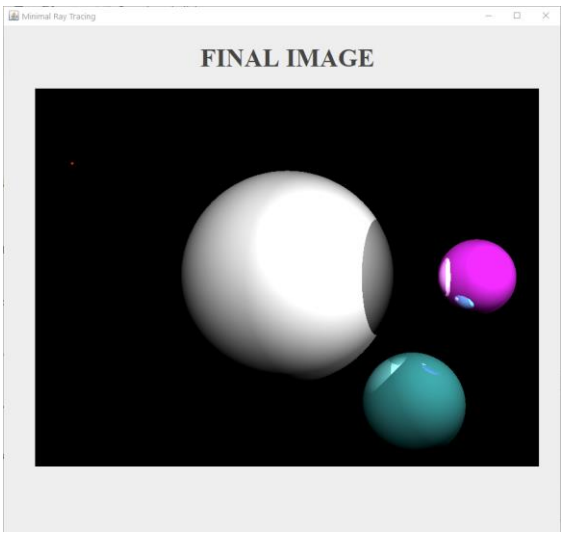
It consists, for each pixel of the image to be generated, of launching a ray from the point of view (the "camera") into the "3D scene". The first point of impact of the ray on an object defines the object concerned by the corresponding pixel. Rays are then fired from the point of impact towards each light source to determine its brightness (is it lit or in the shadow of other objects?). The final colour of the pixel can be defined according to the amount of light reaching it, to the surface properties of the object: its colour, roughness, etc, reflections notions, transmission, and transparency notions, and to other more complex elements.

A sophisticated and powerful ray tracer can define a scene containing different shapes, different lights, and different camera to allow to create very realistic images such the one underneath. For our simulation, we will work on a black background, with a pre-defined scene containing two light sources, one of which can be modified, and three spheres made of specific materials that can be defined, and similarly, only one can be modified.

2. GUI Description

In our ray-tracing program, the scene is composed of 3 spheres illuminated by two light sources and seen by one camera. Two spheres and one light are already defined in the program. However, the user can choose all the different parameters regarding the remaining sphere and light. The camera is also defined by the user, which is also able to reset it, to enter another version. All the elements which are selected by the user have already been predefined. This means that if no values are entered the initial version of the element will be considered.





The user can choose between nine different materials to create a sphere. The principal characteristic of a material is its reflection coefficient. Indeed, this coefficient is a number between 0 and 1. When this coefficient gets closer to 1 the reflection increases. On the opposite, when it is closed to 0 the material becomes opaque.

material	reflection coefficient (between 0 and 1)
white	0
orange	0
yellow	0,25
cyan	0,25
green	0,5
red	0,5
pink	0,8
magenta	1
blue	1

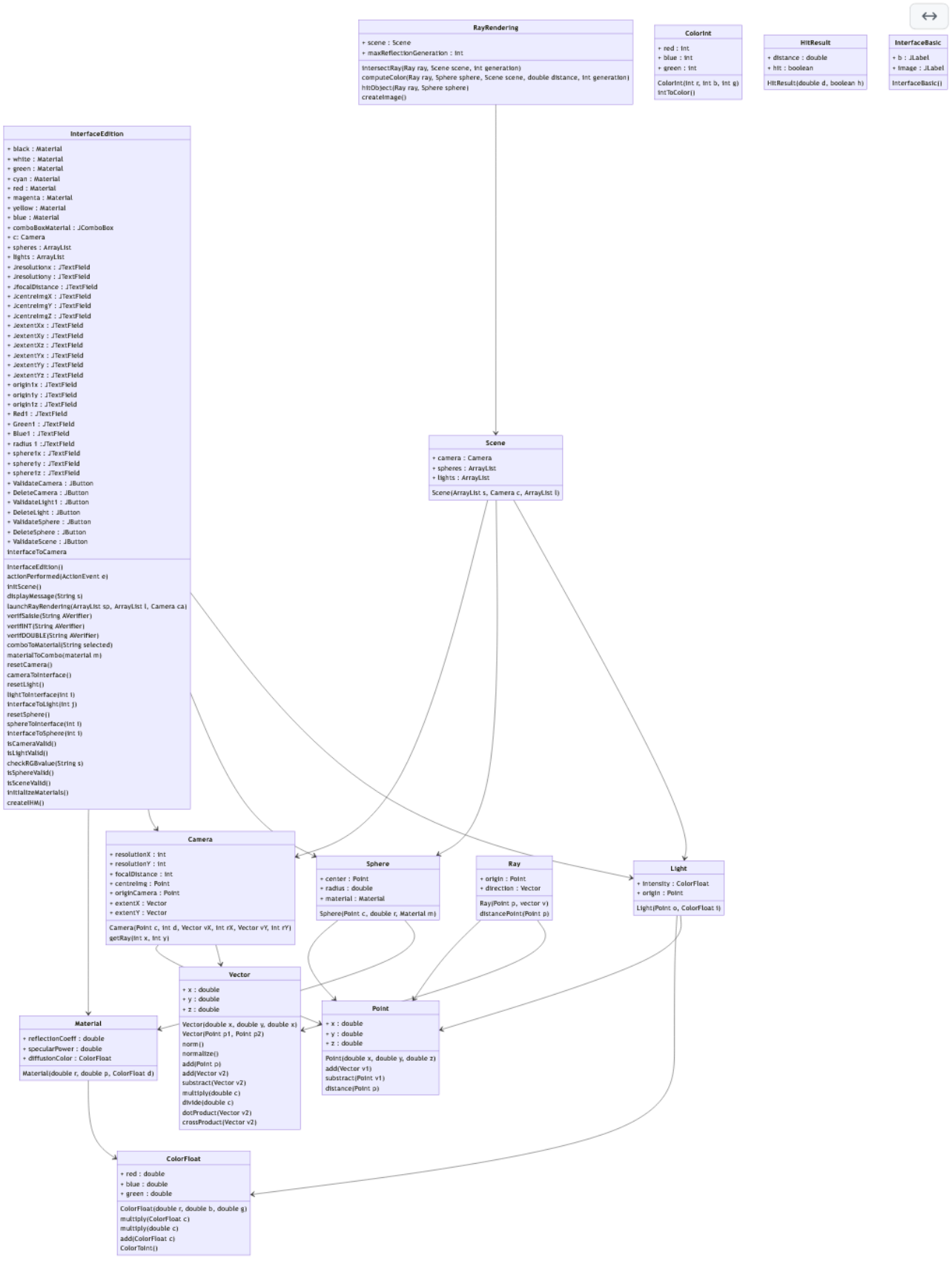
However, even if two materials have the same reflection coefficient, they are not identical. Indeed, two other parameters must be considered: the specular power and the diffusion color. The specular power also called “shininess factor" does not impact on the reflective capacity of the material but acts on its appearance. Here is the table of the different materials, order from the least to the most reflective material.

3. Algorithm Description

Our algorithm allows to determine an image for a scene containing 3 spheres (no other shapes) seen by one camera. Among the three spheres, two can’t be modified and one has been predefined, but the user can modify it. Regarding the lights, one has already been entered in the program and the other one is only predefined and can therefore be modified. As we explained in the first part, our algorithm aims to, for each pixel of the image to be generated, of launching a ray from the point of view (the "camera"), into the "3D scene". This is the opposite of what we are taught in physics courses (the rays leave the object and go in all directions, and some of these rays reach our eyes). We decide here to do the opposite path for the sake of simplicity. As we explained before, the first point of impact of the ray defines the object touched by the corresponding pixel. The rays are then extended from the point of impact to each light source to determine their brightness. The final color of the pixel is then defined by the "amount" of light from the light source reaching the point of intersection of the object, the color of the light and the color of the object. To understand in greater detail how the Ray-Tracing phenomenon works, we consulted several websites that explain the concept, which are listed in the bibliography in section B.

In our algorithm, we added the principle of reflection. For each ray hitting an object, we create a reflected ray according to the reflection laws of optics. Once created, this ray follows the same treatment as the “regular rays” (i.e, checking if it hits a sphere, if yes compute return the color). Then, the color of the initial pixel is influenced by the sphere around it and its reflection. To define the scene of the image, the user has to use the interface. Firstly, the user defines the position of the camera. He enters the focal distance of the camera, the position of the plane of the image in the 3D scene. To do so, he enters the central point of the frame, the two normal vectors defining the plane, and the resolutions in x and in y of the final image. Then, he defines the light and the sphere present in the scene that can be modified. For the light, he enters the point of origin of the light and its intensity color. As for the sphere, it has a 3D point defining its center, a radius and is made of a specific material. A material possesses a coefficient of reflection, a color of diffusion (“surface color”) and a specular power. For the sake of simplicity, the user can choose between a finite list of predefined material that we defined in the algorithm (by defining their diffusion color, their reflection coefficient, and their specular power) that we listed in a JComboBox, a tool that we learned to use during the project (resource listed in section C of our bibliography). Once the user has defined the global scene, he can launch the computation of the final image. We write our program so that we have default values for the camera, for the sphere and the light that we can modify, so that we can run the program without having to configure everything.

4. Class Diagram



5. Results and Discussions

A. Objectives achieved

This subject seemed quite intimidating at first, since we had already coded some games during the past two years, but we had never made an algorithm that looked like this ray tracing project.

At the beginning, the project was not very clear in our minds, we did not know where to start, but finally, thanks to certain websites helped us a lot (with different levels of explanations: see section A of the bibliography), we managed to tackle the difficulties one by one until obtaining our final code. This allowed us to see our progress during the simulations throughout the project.

B. Strongest points

One of our strengths, in our opinion, is that we were able to stand out from the other groups, who mostly coded games. Indeed, we found that our program was a little more original and moreover, we found that it was quite impressive to manage to code and to get a the end a 3D image. We are quite proud of the result, and it was very satisfying to be able to do this on our own (of course by using several resources to get an idea of how to proceed for some points).

Also, we were careful during the whole project to move ahead little by little. We really preferred to code all the steps one by one, rather than jumping into big parts without really knowing where to go. This allowed us to see our progress as we went along, and to add elements one by one without mixing everything up. This way, the project seemed less complex to tackle.

And finally, a big point of difficulty that we have managed to solve is the gap between skills in computer science. In our group, there are big differences in level, and this made us quite afraid at the beginning, but we finally succeeded in dividing the tasks so that each of us could do things at her level. We finally manage to divide the work in a quite equitable way.

C. Flaws of our work

As for any project, our simulation has flaws. The main weakness of the program is that our interface doesn’t allow the user to add or modify other lights and spheres. This is a pity because it prevents the user from playing with the possibilities of the ray tracing program. A nice improvement could be to be able to create and modify materials to increase the possibilities of different pictures.

Also, we find that our interface is not very elaborate. We didn't manage to make it fit the size of the window, and it's less aesthetic since it has fixed dimensions. We didn't manage to use the grid tool to optimize its size.

D. Lessons learnt and enhancement

For this project, we learned how to use the collaborative work platform GitHub, which allowed us to work on our files in parallel. And even though it took us a while to get used how it works, it was helpful eventually.
We also had to deal with other difficulties, such as translating physical phenomena into a programming language, but also with all the problems related to the realism of the final output, and how to make our final image the most realistic possible. We sometimes found ourselves facing problems that we couldn't find the origin of, and on which we struggled a lot, but then we learned from our mistakes and made sure to code little by little to see our progress in the most detailed way possible.

Now that our program is finished, and now that we have a final vision of our work, we can consider some improvements we could have made if we had had more time. The first improvement (and the one that we have really try to do) is the possibility of entering several spheres and lights in the scene. Indeed, the scenes can be

composed of several elements; this is what would be, in our opinion, the most interesting thing to add if we had more time. Moreover, we could have changed the black background of the scene into a patterned one to have a better view of the reflection phenomena, but we could also diversify the shapes of the objects, or add more cameras to observe the scene from several points of view, but we could also add the notion of transmission through some materials (the ones we created are complement opaque), and make it so that the user can create the materials as he wants.

6. Project Timeline and Members' Contributions

For the project, we used the collaborative work platform GitHub. We had used a similar platform for a project in Mechanical design (GrabCab) before, so it was not too complicated for us to get used to this system. It still took us one or two sessions to adapt to GitHub, as it has many features, and this took away some of the work time that we could have dedicated to the project. However, the skills we acquired by working with GitHub will certainly not be lost, and it can be an advantage for our future studies to know how to use this kind of system.

Concerning our organization, we started the project by making some research about Ray-Tracing and about the physics behind it, and we also looked at some websites that provide some explanations of it (see section A in the bibliography). We all made some research at first Then, we started coding. Perrine and Ninon started by creating the basic classes: Sphere, Vector, Point whereas Capucine started on working with the class Light. Then, we worked on the more complicated classes, with the use of more detailed resources on the Ray-Tracing concept (see section B). Ninon worked on the material classes whereas Capucine worked on the classes Camera and Scene, that were more difficult to tackle than the other. Capucine then dealt with the main class Ray Rendering which sends the rays and determines the color of each pixel, and Ninon joined Perrine who had already started to work a lot on the interface, to set up all the necessary elements. And of course, throughout the project, we helped each other when we encountered difficulties.

Throughout the project, we made sure to code as we went along to make sure we were doing the steps one by one, and not to get confused, so that each time we had an idea of what we were working on. For example, we first coded so that we could create an image. Then, we tried to work with the Vector class to detect when a ray coming from a point representing the camera touches a sphere: we displayed white if the sphere was touched, and black if not. Once this was working out well, we added the calculation of the pixel color, and once everything was clear, we finally added the reflection effects.

Eventually, Capucine did the biggest part of our algorithm. This comes from the fact that she is much more comfortable in coding than Ninon and Perrine as she wants to join the Informatics department, and hence has a certain appeal for this kind of project. Therefore, as we have unequal skills in coding, it was hard to divide the tasks equally at the beginning, but finally, we managed to work by making sure that each of us could do things according to her abilities. If we were to define our level of participation in the project in percentages, Capucine would have 40%, and Ninon and Perrine would both have 30%. It also was very instructive for us to work together even though we have very different levels in coding.

7. Bibliography

A. First approach resources

As a first approach we used these resources to better understand the subject and the ray-tracing technic.

- https://github.com/ssloy/tinyraytracer/wiki/Part-1:-understandable-raytracing
- https://fr.wikipedia.org/wiki/Ray_tracing
- https://tobias.isenberg.cc/graphics/LabSessions/RaytracingProject

B. More detailed resources

Then, when we had well-understood the ray-tracing technic, we used the articles and the video to have an idea of what an algorithm of raytracing looks like. It was very useful to start the project.

- https://raytracing.github.io/books/RayTracingInOneWeekend.html
- https://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes
- https://www.youtube.com/watch?v=SV-cgdobtTA

C. Documentation on Java's tools

We used the link to learn how to use a combo box widget in our interface, and the second to learn about the UML creation with the mermaid documentation.

- https://docs.oracle.com/javase/tutorial/uiswing/components/combobox.html
- https://github.com/mermaid-js/mermaid/blob/develop/docs/classDiagram.md