École Centrale Lyon

Graphic Informatics

Students :
Martin Guillerm

Professor :
Nicolas Bonneel

Contents

1 Link

Here is the link of my GitHub repository with all the resources I used.

If you got some questions or find that they are missing resources, you can send me an email at : martin.guillerm@etu.ec-lyon.fr

2 Implementation

2.1 First steps

Firstly, the mandatory parts are the creation of a closed scene (6 spheres which allow a closed space between each other), the creation of objects (sphere with center, radius and albedo of color) and Ray tracing in order to create the rendering.

Then, I also add shadows by looking if there is an object between the object detected and the light.Then, I add different types of spheres using refraction and reflection. At this point, you could get the following picture:

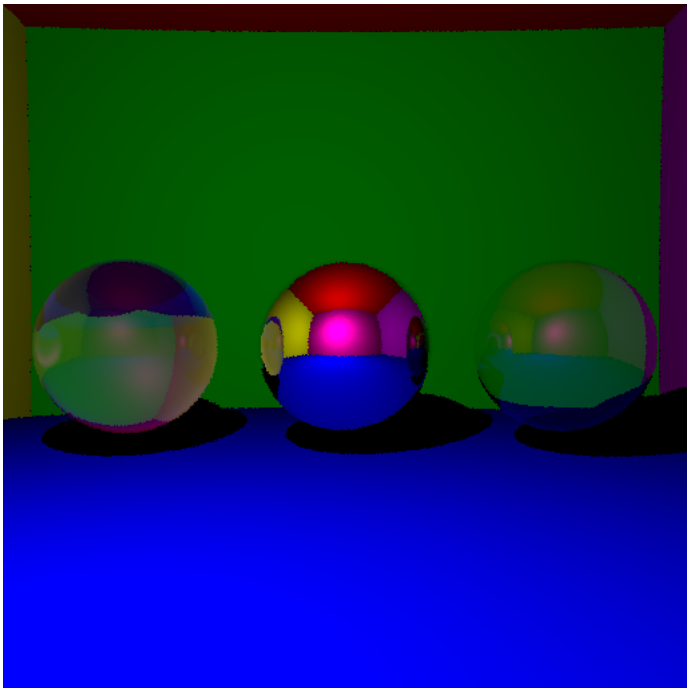


Figure 1: 3 spheres in a virtual world

As you can see, you can get some mirrored spheres and some transparent spheres. In the transparent spheres, you can get a glass-like one which will invert the image when you look through it. The other one is more like a plastic one (a hollow sphere in reality) which will not invert the image.

2.2 Indirect Light

The problem now is that the shadows are really dark, I've modified it in order to get some smooth shadows using indirect light. Basically, I launch 500 rays when I touch an object in order to get a mix of indirect light.

Here you can see the impact of indirect light:

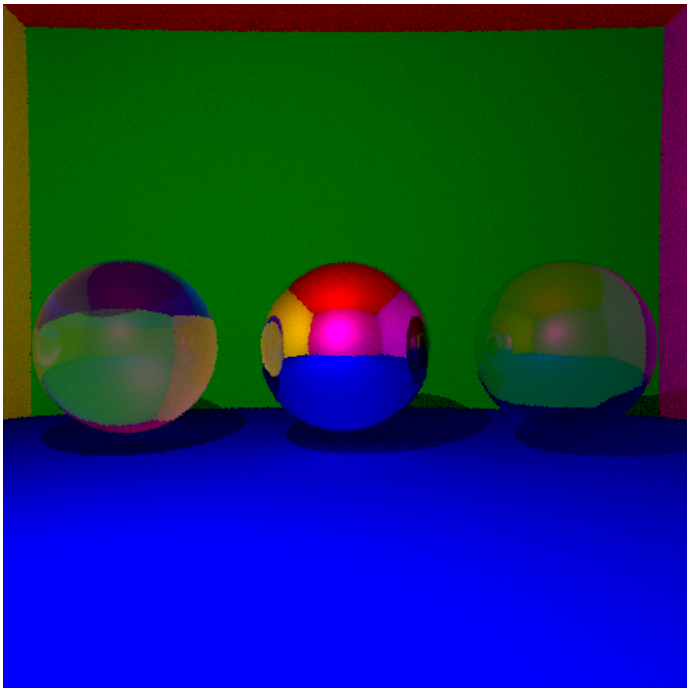


Figure 2: Add of indirect light

2.3 Source of light

I also changed the source of light from a point a sphere. This allows to get a more realistic image :

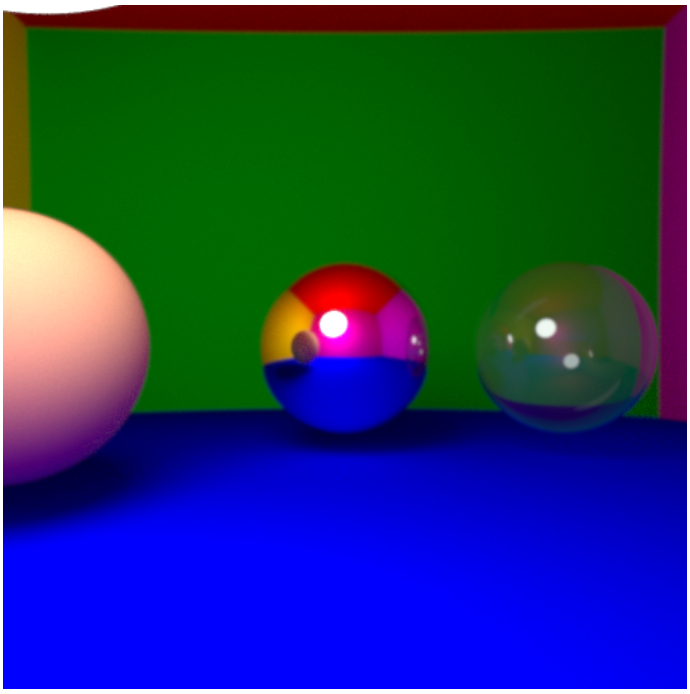
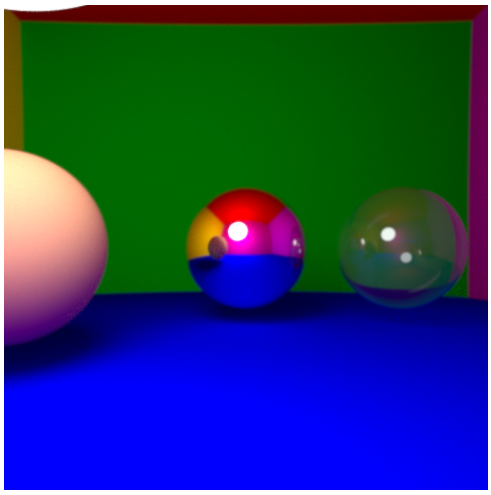


Figure 3: Change the source of light

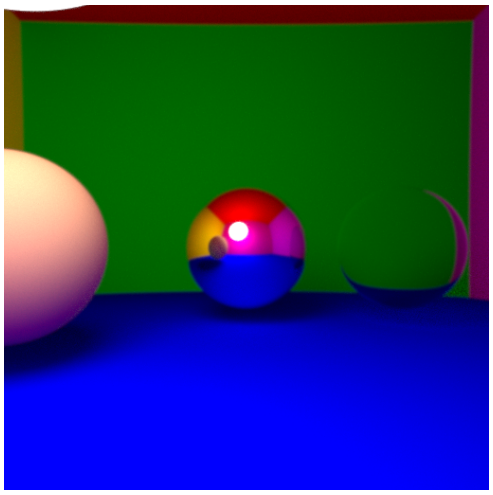
2.4 Fresnel

Note that you can see some pink reflections in the transparent spheres. This comes from the Fresnel equation. In fact, in reality, a part of the ray is reflected whereas the other part is refracted.

Here is the same image with/without the implementation of the Fresnel equation:



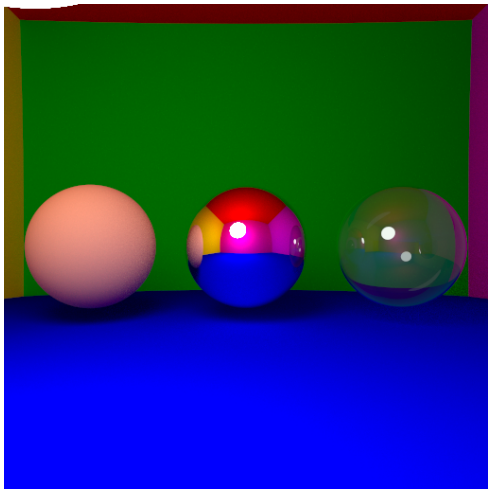
(a) With Fresnel



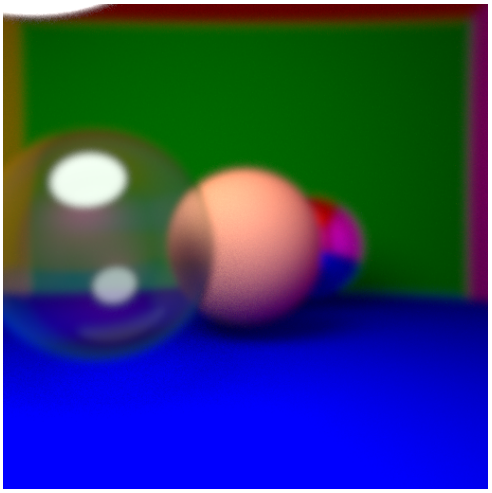
(b) Without Fresnel

2.5 Blur effect

Finally, I added the blur effect to the image linked to the aperture and focus. Therefore, we can focus on objects nearby or far away. I'm not very satisfied with this part, and I think it can be improved.



(a) Without blur effect



(b) With blur effect

2.6 Mesh Loading

Then, I added the mesh loading. It allows adding a mesh composed of vertices and triangles. (We also add a bounding box around the mesh in order to reduce the computation time.)

I also added the possibility to translate/rotate/scale the mesh, by just modifying the positions of the vertices and the triangles.

Here is the result with two meshes of a cat. I decided to try 2500 rays for 7 bounces; therefore, it takes a lot of time (more than 2 hours, I guess). Nevertheless, the picture is very nice.

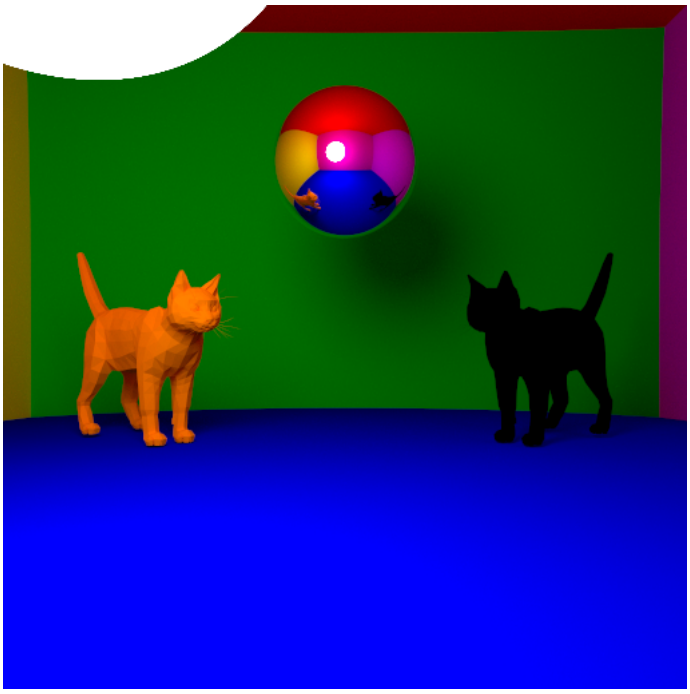


Figure 6: Mesh loading

Later, I implemented mesh smoothing (Phong Smoothing). Indeed, currently, meshes are composed of faces and we easily observe the change of faces (which results in a change of the normal). So we will smooth our mesh by using a field of normals rather than a single normal for a face :

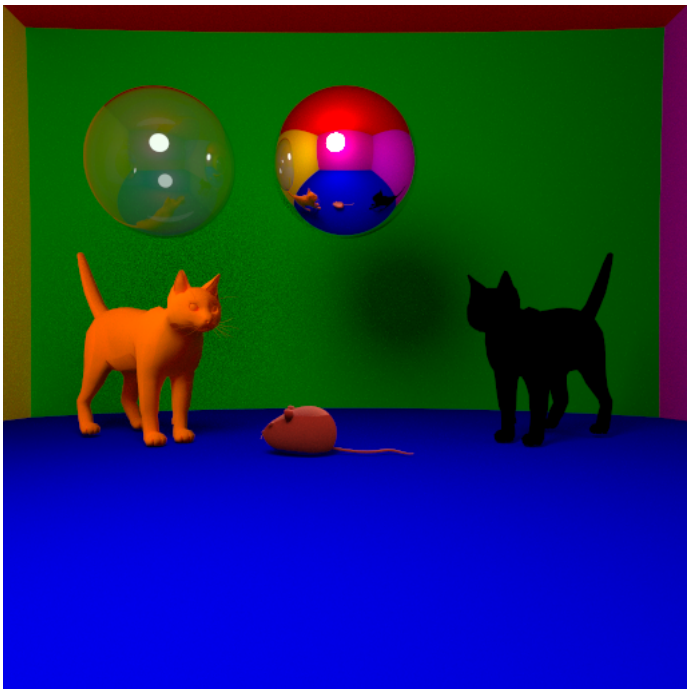


Figure 7: Mesh loading

I also added a mouse mesh to make it more fun and put the light 10 pixel higher in order to don't see it in the picture.

2.7 Texture

The last asset that I implemented is texturing. You can find tow types of texturing. The first one concerned the mesh. It simply associates a triangle of the mesh with a color on a picture. here is the result with our cat :

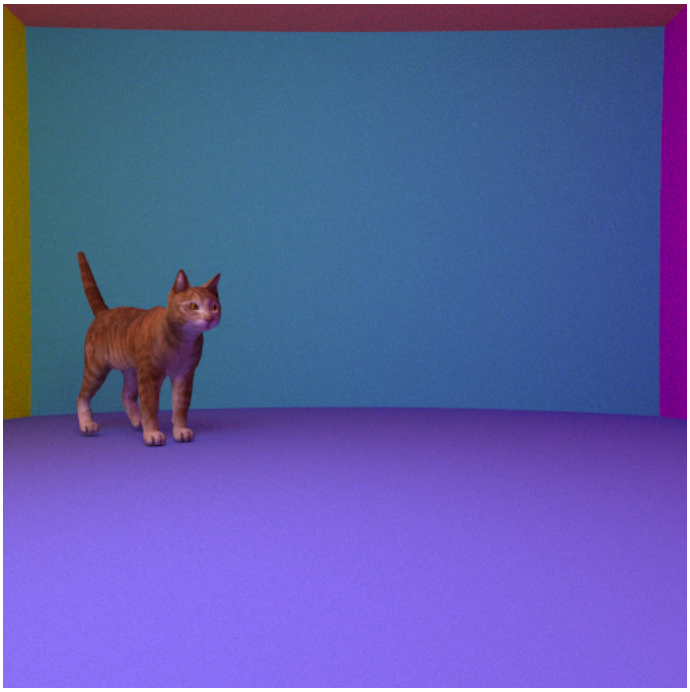


Figure 8: Mesh texturing

Then, the second texturing is procedural texturing for sphere. Here, you determine the texture thanks to a mathematical function (you can also use noise like Perlin noise if you want). Here I computed to procedural texture, one on the ground (don't worry, I solved the checkerboard problem later) and one on the right sphere :

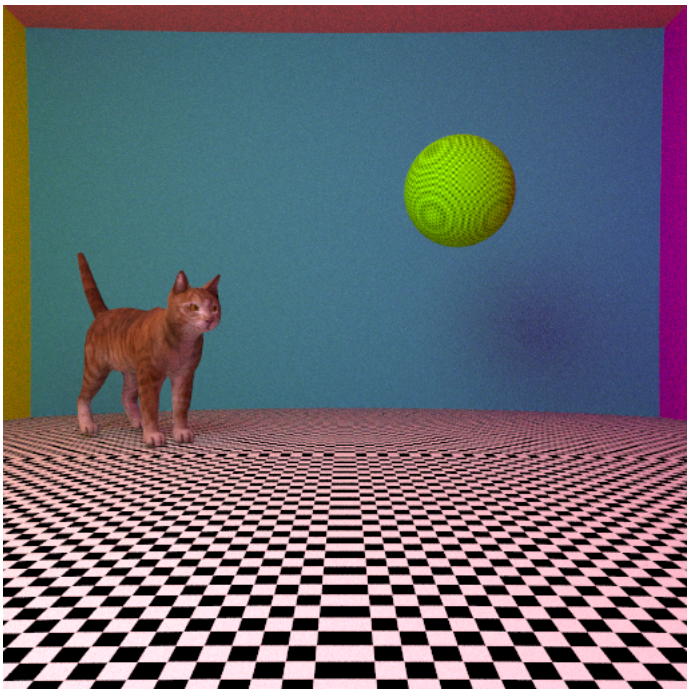


Figure 9: Procedural texturing

2.8 Animation

I added the fact that we can translate/rotate the camera (or the objects if you want). By combining multiple images across a mix of translation and rotation, we are able to display a mini-movie of our scene.

For this section, you can see the result in the presentation of the GitHub repository.

2.9 Performance

An important fact of the project was to reduce the computation time in order to be more efficient. For this, 3 steps were integrated.

First, the implementation of parallel execution which allows to compute each pixel in parallel and win a lot a of time. this is possible thanks to this line :

```
1 #pragma omp parallel for
```

Then, two more modules were implemented, the bounding box and the BVH. Basically, the bounding box (BBox) is a bounding box around the mesh. It helps save time by shortening the process when a ray doesn't intersect the mesh. As for the BVH (Bounding Volume Hierarchy), it divides the mesh into sub-samples to compare the ray with only a small area of the mesh rather than the entire mesh (in practice, we compare the ray with 10/100 points instead of 10,000).

The following table shows the impact of the implemented algorithms (please note that these time are approximated and taken for a picture 512x512 with 1000 rays and 7 bounces for three meshes and two spheres) :

	Only pragma 1	with Bbox	with BVH
Time (min)	240	60	10

Table 1: Impact of Implemented Algorithms

Finally, thanks to the BVH, we need only 10 minutes whereas we needed 4 hours at the beginning.

3 Final result

In this part, I decided to show you my final scene where I let my imagination run wild. I decided to compose a museum scene featuring some of the most well-known statues :

Figure 10: The museum (3000 rays/7 bounces/1h30)

If I had more time, I would have liked to fill the scene with paintings and other statues, and move the camera through my museum. Due to lack of available time, I simply performed a movement around this part of the scene as you can see in the GitHub repository presentation.