Hillel Pinto : 1475638

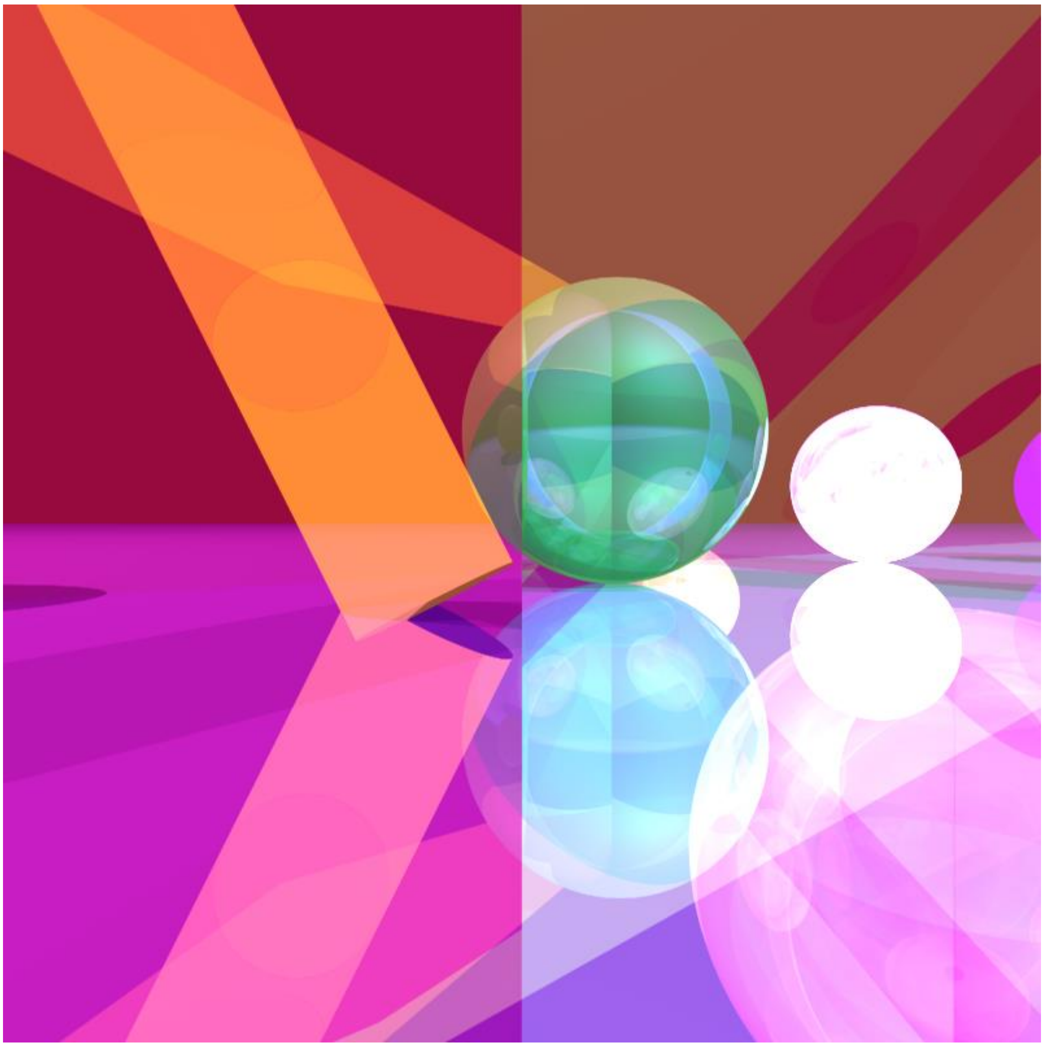
Yona Zsmerla :339622375

Explanation of Mini-project (Part 1 / 2) in Java , 2021,JCT Lev

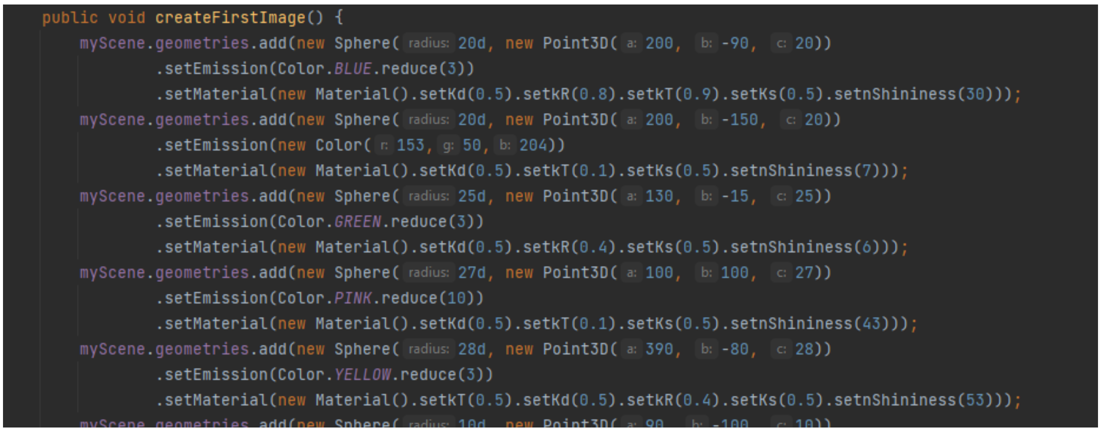
The goal of this part of the project is to finally use all the tools and classes created so far, to be able to generate images of all kinds and make improvements to this image.

So we first generated an image composed of 10 shapes and several different light sources in order to be able to obtain an image which represents the application of a good part of the functions created until then (in the exercises).

Here is the picture :

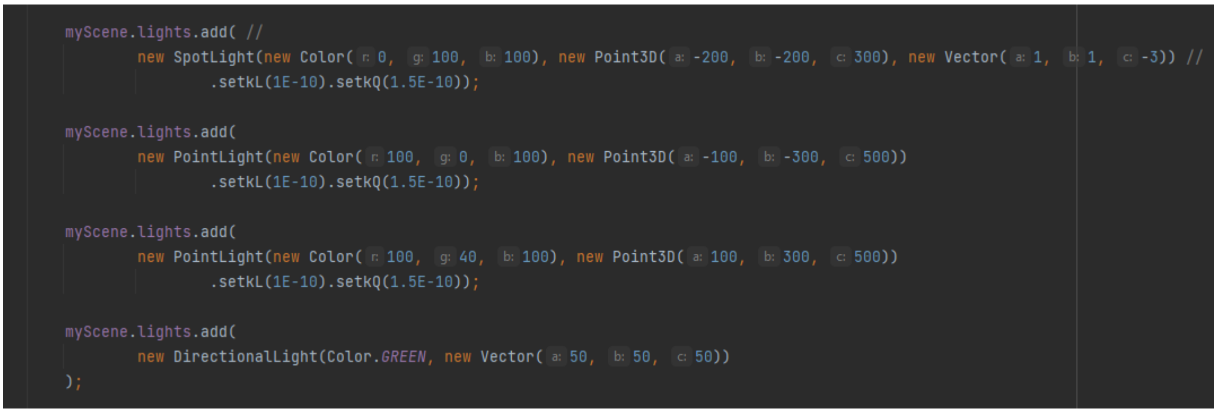


More precisely, in order to generate this picture we create a scene object which contains a camera object and all those geometries object,we add thos forms into a list,like this :



```
public void createFirstImage() {
    mySceneˌgeometries.add(new Sphere( radius: 20d, new Point3D( a: 200,  b: -90,  c: 20))
            .setEmission(Color.BLUE.reduce(3))
            .setMaterial(new Material().setKd(0.5).setkR(0.8).setkT(0.9).setKs(0.5).setnShininess(30)));
    myScene.geometries.add(new Sphere( radius: 20d, new Point3D( a: 200,  b: -150,  c: 20))
            .setEmission(new Color( r: 153, g: 50, b: 204))
            .setMaterial(new Material().setKd(0.5).setkT(0.1).setKs(0.5).setnShininess(7)));
    myScene.geometries.add(new Sphere( radius: 25d, new Point3D( a: 130,  b: -15,  c: 25))
            .setEmission(Color.GREEN.reduce(3))
            .setMaterial(new Material().setKd(0.5).setkR(0.4).setKs(0.5).setnShininess(6)));
    myScene.geometries.add(new Sphere( radius: 27d, new Point3D( a: 100,  b: 100,  c: 27))
            .setEmission(Color.PINK.reduce(10))
            .setMaterial(new Material().setKd(0.5).setkT(0.1).setKs(0.5).setnShininess(43)));
    myScene.geometries.add(new Sphere( radius: 28d, new Point3D( a: 390,  b: -80,  c: 28))
            .setEmission(Color.YELLOW.reduce(3))
            .setMaterial(new Material().setkT(0.5).setKd(0.5).setkR(0.4).setKs(0.5).setnShininess(53)));
    myScene.geometries.add(new Sphere( radius: 10d, new Point3D( a: 90,  b: -100,  c: 10))
```

But with all those geometries we had to ilgaine where can we set the light,which color and the distance ,so after a long time of trying we finnaly set those kind of lights in order to get the picture up here :



```
    myScene.lights.add( //
            new SpotLight(new Color( r: 0,  g: 100,  b: 100), new Point3D( a: -200,  b: -200,  c: 300), new Vector( a: 1,  b: 1,  c: -3)) //
                    .setkL(1E-10).setkQ(1.5E-10));

    myScene.lights.add(
            new PointLight(new Color( r: 100,  g: 0,  b: 100), new Point3D( a: -100,  b: -300,  c: 500))
                    .setkL(1E-10).setkQ(1.5E-10));

    myScene.lights.add(
            new PointLight(new Color( r: 100,  g: 40,  b: 100), new Point3D( a: 100,  b: 300,  c: 500))
                    .setkL(1E-10).setkQ(1.5E-10));

    myScene.lights.add(
            new DirectionalLight(Color.GREEN, new Vector( a: 50,  b: 50,  c: 50))
    );
```

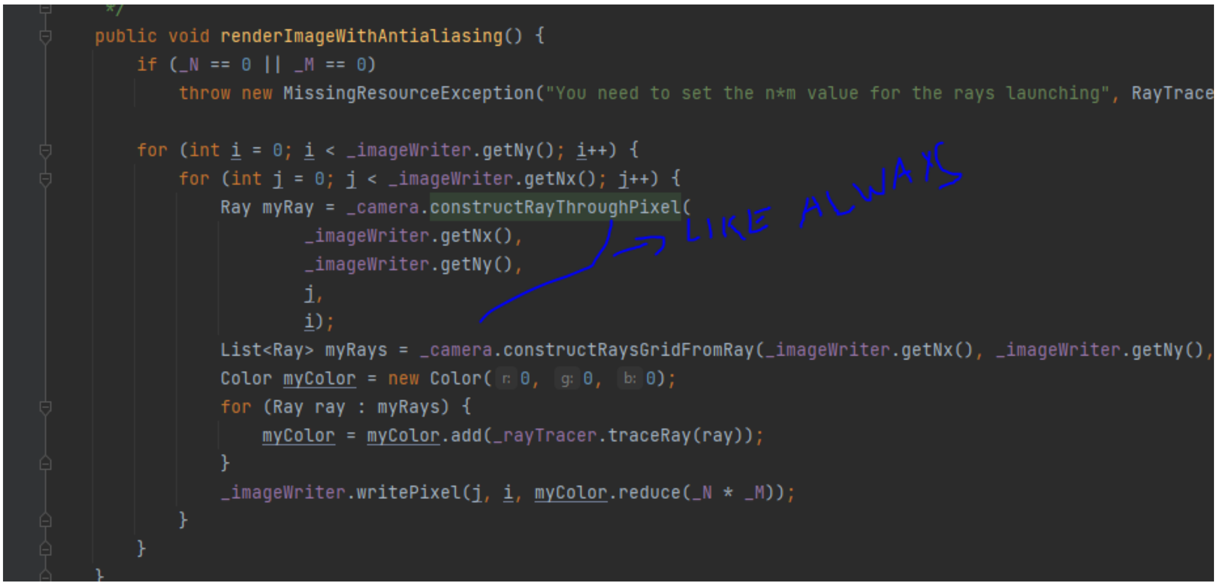
So the first part was to create a real very complex image composed of several touches of color and shapes, unlike the images generated during the exercises, which were simpler, sometimes even without several light sources. So far this has been the least difficult part, because now after having all the tools to generate the images you want, we will focus on its quality, how to improve it.

We have seen several known methods to improve image quality and for the purposes of the project we had to choose only one but we implemented two.

The first method is called super-sampling and the second is called soft shadows. When generating an image we have a view plane which is made up of pixels and in which we will send a ray from the center of the camera to the center of each existing pixel on the view plane!

The method of super sampling consists in not sending only one single ray, but several ....

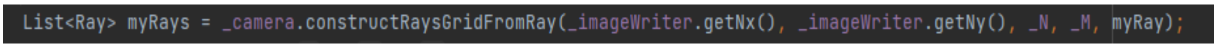
What do we gain? we gain color clarity because we greatly reduce the aliasing between the colors of the photo, and the tones are clearer, I will show you a before / after photo in a few moments. First, the code and the explanation. We always used the same function (createRayThroughPixel) that sends a ray from the center of the camera, called P0 in the course, to every pixel of the camera's view plane, so we took it and improved it according to our needs. have made it so that the user decides the total number of rays he wants to send in each pixel.



```
    */
    public void renderImageWithAntialiasing() {
        if (_N == 0 || _M == 0)
            throw new MissingResourceException("You need to set the n*m value for the rays launching", RayTrace

        for (int i = 0; i < _imageWriter.getNy(); i++) {
            for (int j = 0; j < _imageWriter.getNx(); j++) {
                Ray myRay = _camera.constructRayThroughPixel(
                        _imageWriter.getNx(),
                        _imageWriter.getNy(),
                        j,
                        i);
                List<Ray> myRays = _camera.constructRaysGridFromRay(_imageWriter.getNx(), _imageWriter.getNy(),
                Color myColor = new Color( r: 0,  g: 0,  b: 0);
                for (Ray ray : myRays) {
                    myColor = myColor.add(_rayTracer.traceRay(ray));
                }
                _imageWriter.writePixel(j, i, myColor.reduce(_N * _M));
            }
        }
    }
```

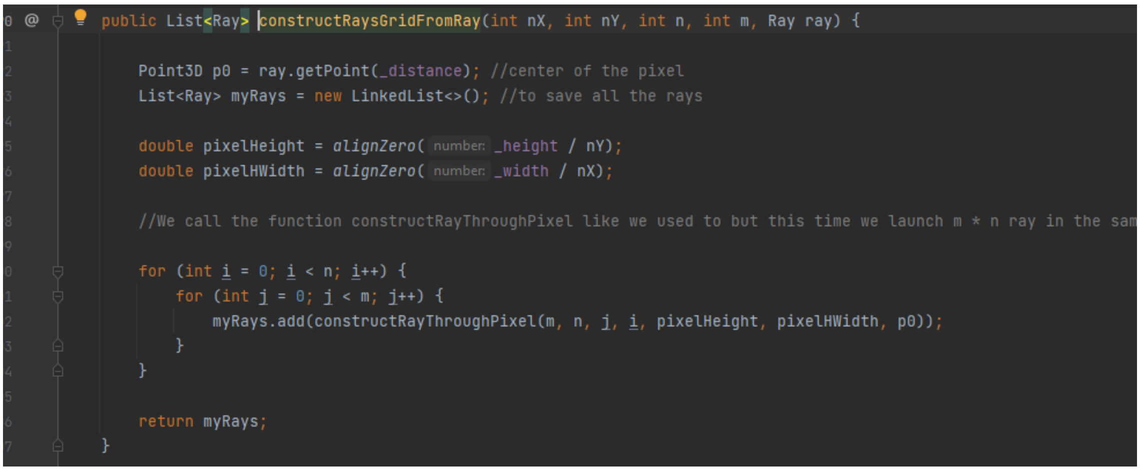
Here’s the function called when we choose to generate the poicture with antialiasing improvements.

The part marked in blue ,is not very special,but after this ,when we already launched a ray through the center of a pixel , we launch a set of rays in the pixel but in not the very same place exactly.



```
List<Ray> myRays = _camera.constructRaysGridFromRay(_imageWriter.getNx(), _imageWriter.getNy(), _N, _M, myRay);
```

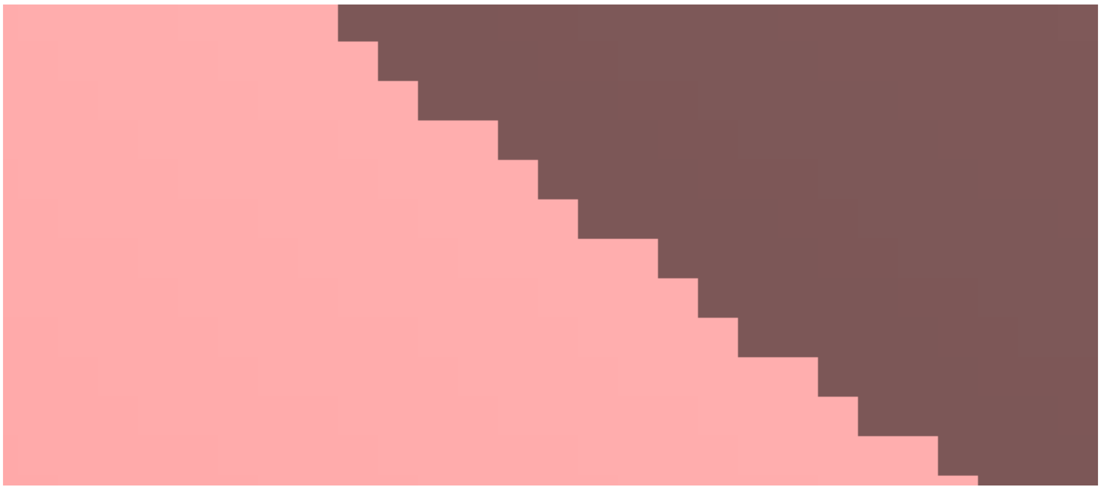
Now lets see how the set of rays will be launched in this particular pixel :

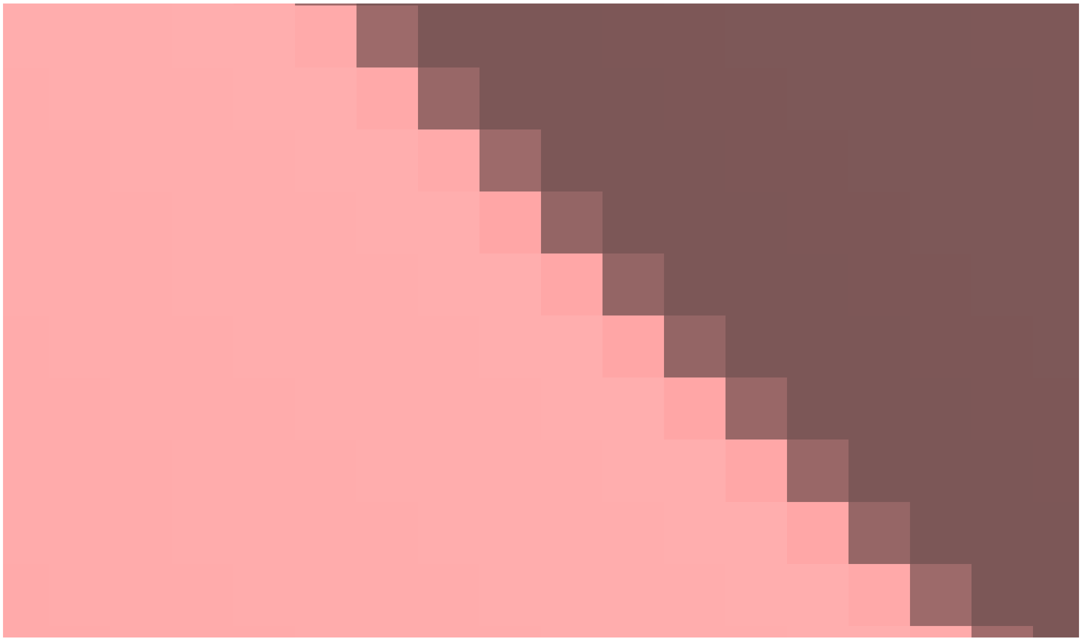


```
public List<Ray> constructRaysGridFromRay(int nX, int nY, int n, int m, Ray ray) {

    Point3D p0 = ray.getPoint(_distance); //center of the pixel
    List<Ray> myRays = new LinkedList<>(); //to save all the rays

    double pixelHeight = alignZero( number: _height / nY);
    double pixelHWidth = alignZero( number: _width / nX);

    //We call the function constructRayThroughPixel like we used to but this time we launch m * n ray in the sam

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            myRays.add(constructRayThroughPixel(m, n, j, i, pixelHeight, pixelHWidth, p0));
        }
    }

    return myRays;
}
```

Okay so we get in the parameters of our function ,the ray that we already launched ,in order to get the distance from the center of our camera to the center of the pixel that is passed by this ray and this time we launch n*m rays which is the total numbers of ray that the user wants to launch into a single pixel .But like we can see the function takes no longer the same number of arguments because we took the original function and update it in order to send the rays in random areas of the pixels.

So finally,after all those rays launch into a pixel ,we call a function that will return the color of the object intersected by the ray,and with all those colors we just make a sum and tada…we have a very much beter quality of a picture,here’s the picture before (with aliasing between the colors) and now (with much rays launched in a pixel to get a a more precise color without aliasing) :





In details,when we zoom we can see much better the improvement,there’s attenuation of the aliasing so the color is much clear.

Until now we saw just the first method,super sampling in order to avoid the aliasing,now we wil talk about the second improvement method called soft shadows