CmpE460 - Assignment1

Report

1. Input & Output

a. Config File

I've used a config file to get spheres. This is a text file. It's lines represent the spheres. Each line consists of 7 integers separated with white space. The first three represent the coordinates of the center of the sphere, x, y, z, respectively. The second three represents the RGB value of the sphere ranging from 0 to 255, red, green, blue, respectively. The last entry (integer) represents the radius of the sphere. All coordinates and radius are in screen space.

Example:

```
50 50 300 255 0 0 20
100 100 600 0 255 0 60
```

b. Running

```
python rayTracer.py --config_file config.txt
```

The output is named *result.png* in the working directory.

2. Requirements

Numpy = 1.19.2
Matplotlib = 3.3.2

3. Method Used

a. Converting from Raster Space to Screen Space

The image is represented as a three dimensional array in the program. The image has 1000 x 1000 resolution and the left-top corner has the coordinates (0, 0). We have to scale the coordinates from (0, 1000) to (0, 1) to convert NDC (Normalized Device Coordinates). Then we have to move the origin of the space from left-top to the middle of the screen. Then scale it back to screen coordinates, which is (-50, +50) in our case.

Besides, each pixel should represent the middle of the pixel in screen space so we add 0.5 to each coordinate for both x and y.

To convert from raster space to NDC space we add 0.5 to each pixel coordinate and scale it with the resolution, which is (1000x1000) in our case.

Then, to convert from NDC space to screen space, we subtract 2 times NDC x-value from one so that [0,1] range maps to [-1,1] range. Then we subtract 1 from 2 times NDC y-value so that we get [-1, 1] range. The difference between x and y axis is related to the coordinates of the old origin, top-left. Top-Left has negative x-value whereas its y-value is positive. This is why the computation is different.

Lastly, we multiply each screen space coordinates to get a scaled version of it.

b. Ray Generation

Each ray consists of two 3-vectors, the origin and the direction. The origin of the primary rays are the same since they all started from the camera (or the eye) located at point (0, 0, 0). The direction is to the screen space coordinates of each pixel computed using the procedure above.

On the other hand, the secondary rays (the rays used to compute shadows) have different origins and directions as well. The origin of each secondary ray is the intersection point of the object and a primary ray. The direction is from the intersection point to the light source.

c. Intersection

The intersection of a ray and an object is computed by solving their parametric equation together. (r(t) = e + t*d, (p - c)^2 = R^2) The combination of the two equations gives a quadratic equation.

We first calculate its discriminant to test whether there is a solution or not. If discriminant is negative, that means the ray and the object does not intersect. If discriminant is zero, that means the ray is tangent to the sphere. If the discriminant is positive, that means the ray passes through the sphere and intersects with it by two points, one for entering and the other for leaving the sphere. Since the color depends on the rays first intersection, we use the negative root of the discriminant while solving the equation to find t.

Note that we need to save the smallest t-value of a ray since the ray may intersect with multiple objects. We keep track of the smallest t because we are interested in only the object that is closest to the screen.

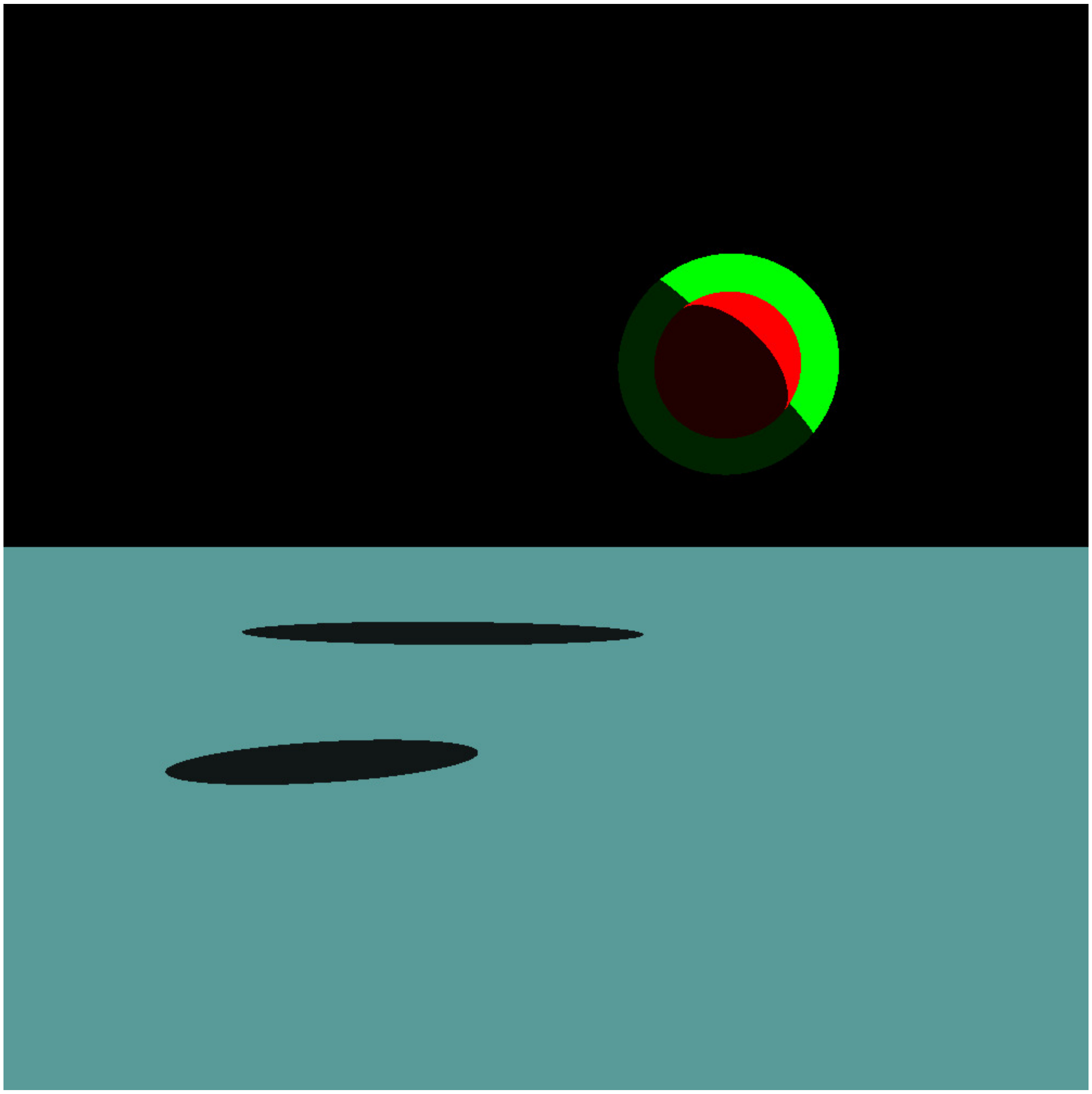
In my implementation, I have a plane to show the shadows more and to give a more 3D intuition. The ray tracing procedure is the same except the discriminant part because the solution of the ray and the plane does not give a quadratic equation.

d. Shadowing

A primary ray may intersect with an object but there may be another object between that intersection point and the light source. For those cases we introduce the secondary rays. Secondary rays start from the intersection point and move towards the light source. If this secondary ray intersects with an object (blocking object), that means the primary intersection point is in shadow. The blocking object may be another object or the object itself (self-shadowing)

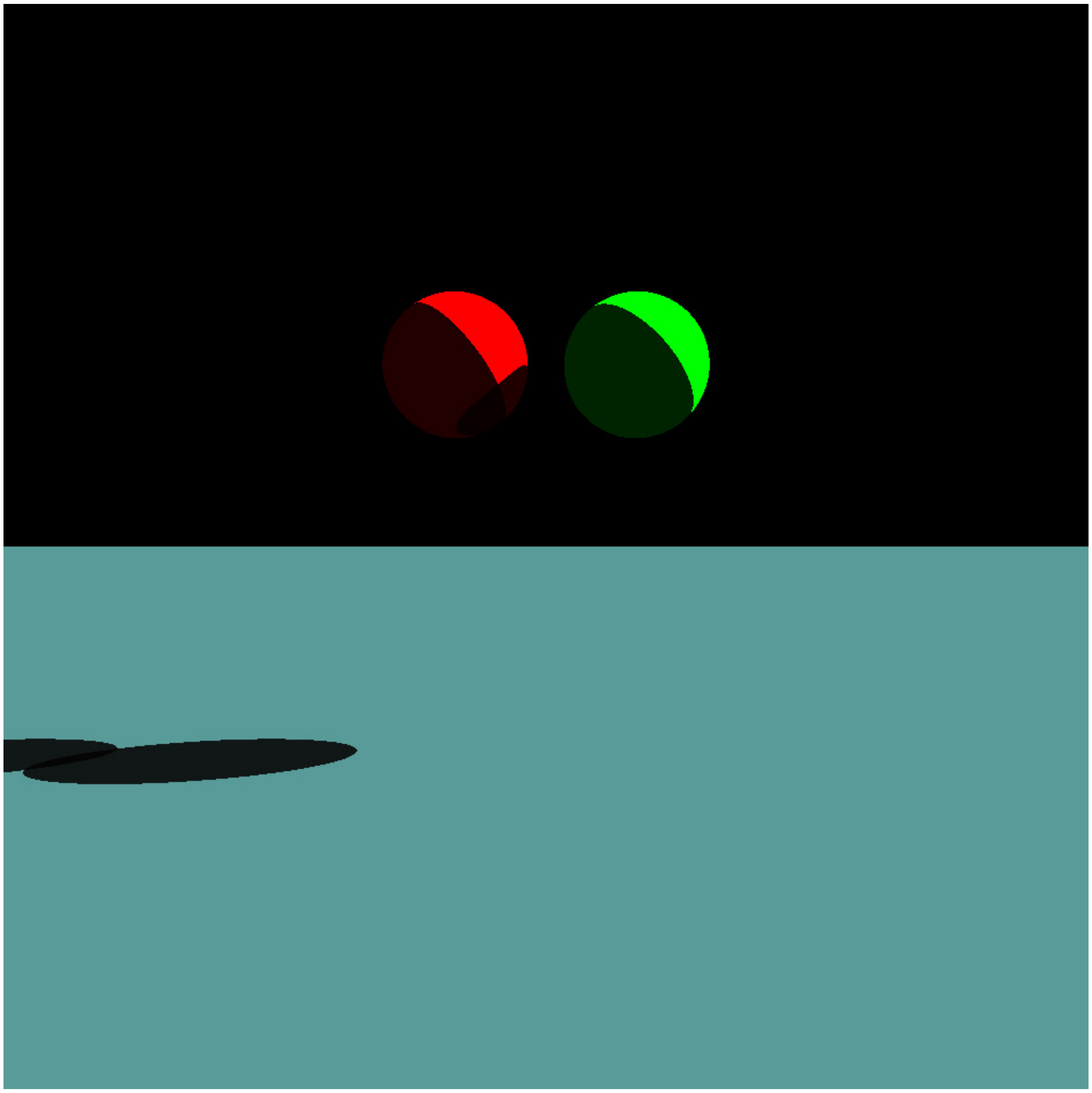
There are multiple types of shadowing techniques. The one we used in this project is the ambient shadowing. In ambient shadowing, if a point is in shadow then its RGB value is lowered by a constant, which is 0.1 in our case.

4. Example Outputs



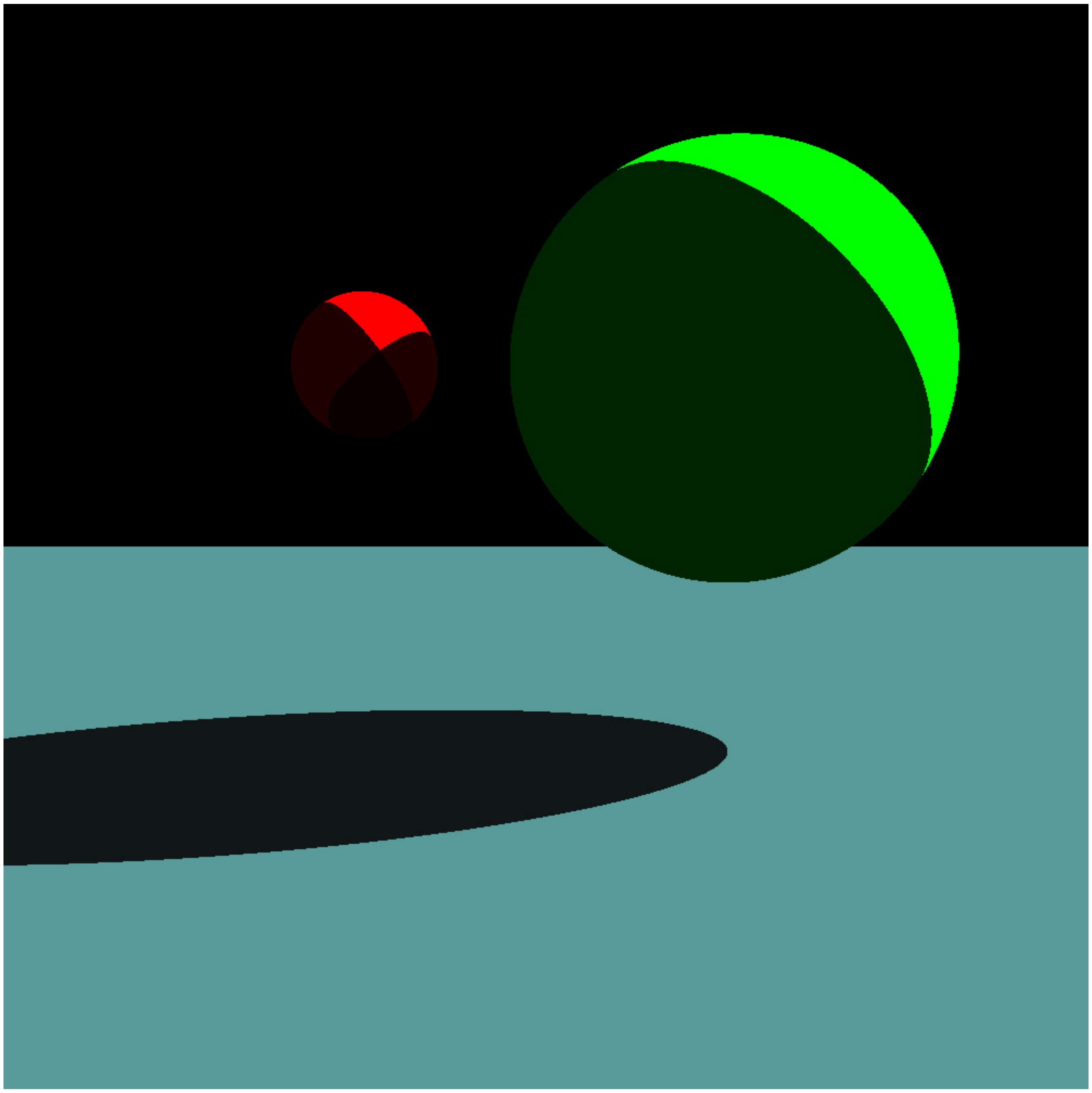
Config File:

```
50 50 300 255 0 0 20
100 100 600 0 255 0 60
```



Config File:

```
-25 50 300 255 0 0 20
25 50 300 0 255 0 20
```



Config File:

```
-50 50 300 255 0 0 20
50 50 300 0 255 0 60
```