

CMPE 460 - Spring 2020

# Homework 1

## Ray Tracing

---

Aysu Sayin

2016400051

## 1. Introduction

In this work, ray tracing is implemented to render the scenes with sphere objects. The space consists of the spheres that user gave as input. The eye point is at the origin and the screen extends from (-50, -50, 100) to (50, 50, 100). There is a point light source at (500, 500, 500) and the resolution is 1000x1000 pixels.

For this homework, the shading is ignored and only the shadows are showed by multiplying the color with 0.1.

## 2. Method

In this section, I will briefly explain the ray tracing method that is implemented. My program is implemented with Python in an object oriented manner. There is the Scene class where all the tracing algorithm is implemented.

Ray tracing is implemented in the following way: Each pixel in the screen is iterated. A ray is shot in the middle of a pixel. The (x,y,z) coordinates of the middle point is calculated with

$$x = (i / 10) - 50 + 0.05$$

$$y = ((999-j) / 10) - 50 + 0.05$$

$$z = 100$$

i,j are the row and column number respectively. The screen width and height is 100 so i and j values are divided by 10. Each pixel is 0.1x0.1. Thus, in order to find the middle point, 0.05 is added.

Then, a ray is generated where its start point is the eye and the direction is the (x,y,z) coordinates that we found - eye.

Then, each sphere in the scene is iterated and checked if it intersects with the ray. The sphere-ray intersection is calculated using simple geometry. The calculations are made as if the sphere is at the origin for the ease of calculation. Therefore, the starting point of the ray is shifted accordingly. The ray formula,  $p + t.d$ , is inserted in the sphere formula,  $x^2 + y^2 + z^2 + r^2 = 0$ , and solved for t.

The resulting equation is  $d \cdot d \cdot t^2 + 2 \cdot p \cdot d \cdot t + p \cdot p - r^2 = 0$ . The discriminant is calculated from this equation. If it is less than 0, that means ray and sphere don't intersect; otherwise they intersect at point with parameters  $t_0$  and  $t_1$ . If those  $t$  values are greater than 1, the objects are visible on the screen. Otherwise, they are behind screen so that we cannot see them.

A ray can intersect with multiple objects but only the foremost one must be visible on the screen. Thus, the sphere with minimum  $t$  value is stored. Then the illumination at the intersection point is calculated. In this homework, it is checked if any object is blocking this point. Similarly, a ray whose starting point is the intersection point and the direction is (light - starting point) is created. Then, all the objects are iterated again and checked if any of them intersects with the ray. In this case, only the objects with  $1 > t > 0.0001$  can block the light because it means that ray and object intersect at a point between the sphere and the light. Instead of 0, the threshold is set to 0.0001 to avoid from the errors occurring because of the objects' intersection with themselves. If something is blocking that point on the sphere, the color is multiplied by 0.1.

The generated image is saved as a jpg file using Python's PIL library.

### 3. Input-Output

Input of the program is given from the terminal by following the instructions. First, number of spheres is entered. Then, for each sphere color in R G B format, the (x, y, z) position of the center and the radius is given. Here is the example input of the program:

```
Welcome
Enter the number of spheres: 2
Color of the sphere 1 in R G B format: 255 0 0
Position (x, y, z) of the sphere 1: 50 50 300
Radius of the sphere 1: 20
-----
Color of the sphere 2 in R G B format: 0 255 0
Position (x, y, z) of the sphere 2: 100 100 600
Radius of the sphere 2: 60
-----
```

---

The output is a jpg file named “scene.jpg” saved in the same directory with the program. If user would like to display the image using Matplotlib, set “show=True” when calling *render* method.

## 4. Results

### Example 1:

The number of spheres: 2

Color of the sphere 1 in R G B format: 255 0 0

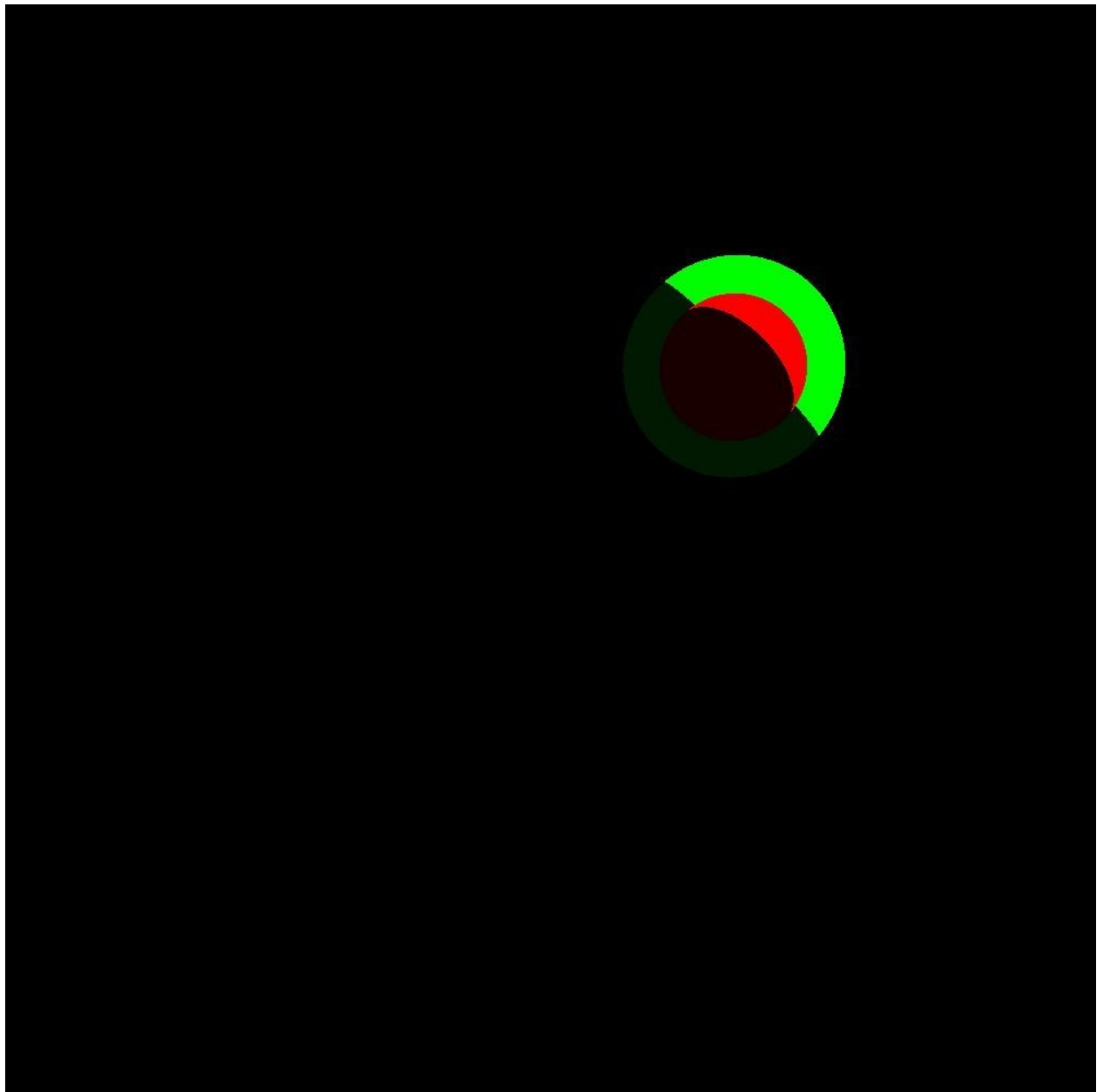
Position (x, y, z) of the sphere 1: 50 50 300

Radius of the sphere 1: 20

Color of the sphere 2 in R G B format: 0 255 0

Position (x, y, z) of the sphere 2: 100 100 600

Radius of the sphere 2: 60




### Example 2:

The number of spheres: 3

Color of the sphere 1 in R G B format: 200 200 200

Position (x, y, z) of the sphere 1: 100 100 500



Radius of the sphere 1: 70

-----

Color of the sphere 2 in R G B format: 255 0 0

Position (x, y, z) of the sphere 2: 200 200 400

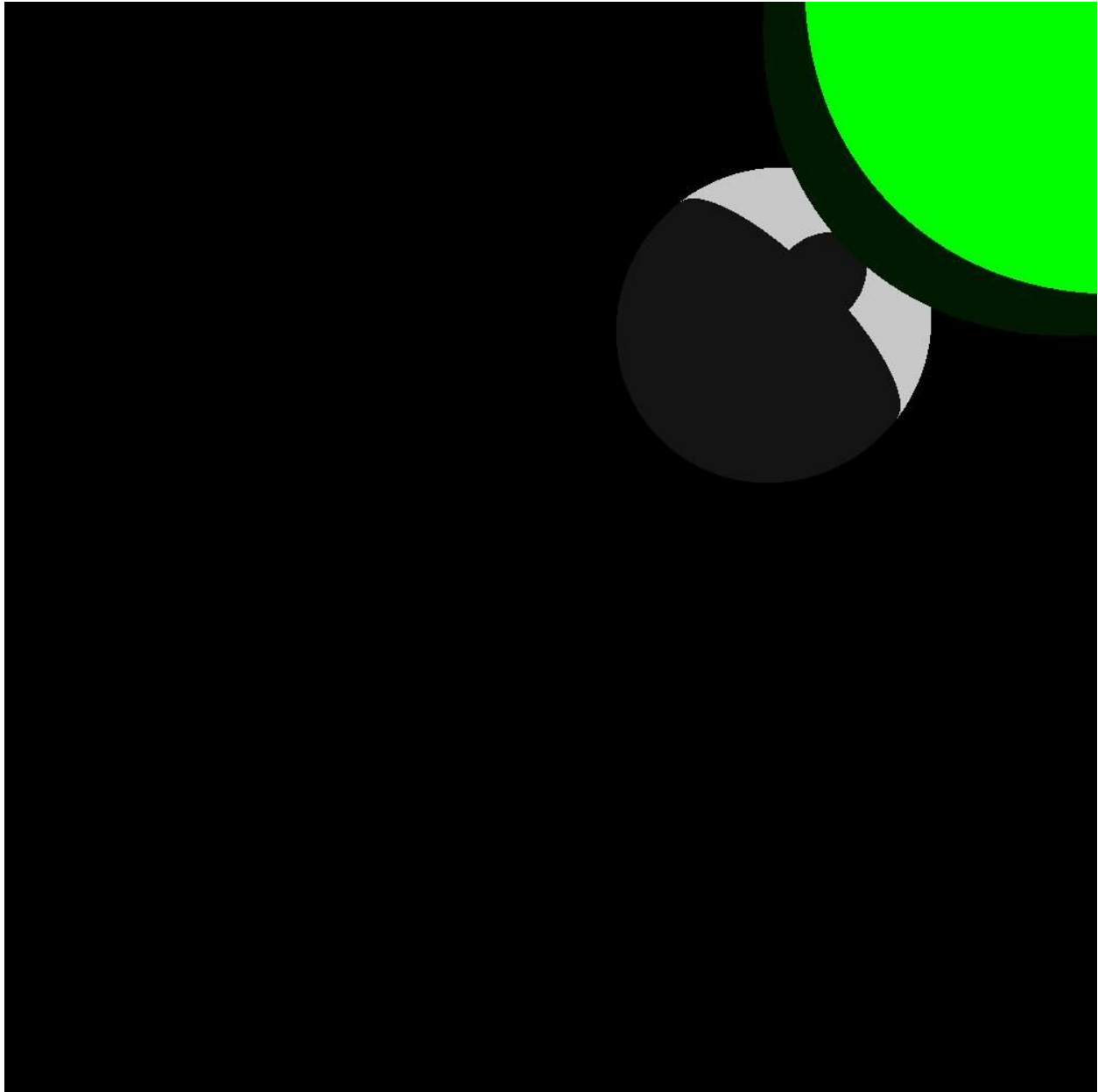
Radius of the sphere 2: 50

-----

Color of the sphere 3 in R G B format: 0 255 0

Position (x, y, z) of the sphere 3: 50 50 100

Radius of the sphere 3: 30



### Example 3:

Enter the number of spheres: 3

Color of the sphere 1 in R G B format: 0 0 255

Position (x, y, z) of the sphere 1: -100 -100 400

Radius of the sphere 1: 30

-----

Color of the sphere 2 in R G B format: 0 255 0

Position (x, y, z) of the sphere 2: -200 -200 -800

Radius of the sphere 2: 70

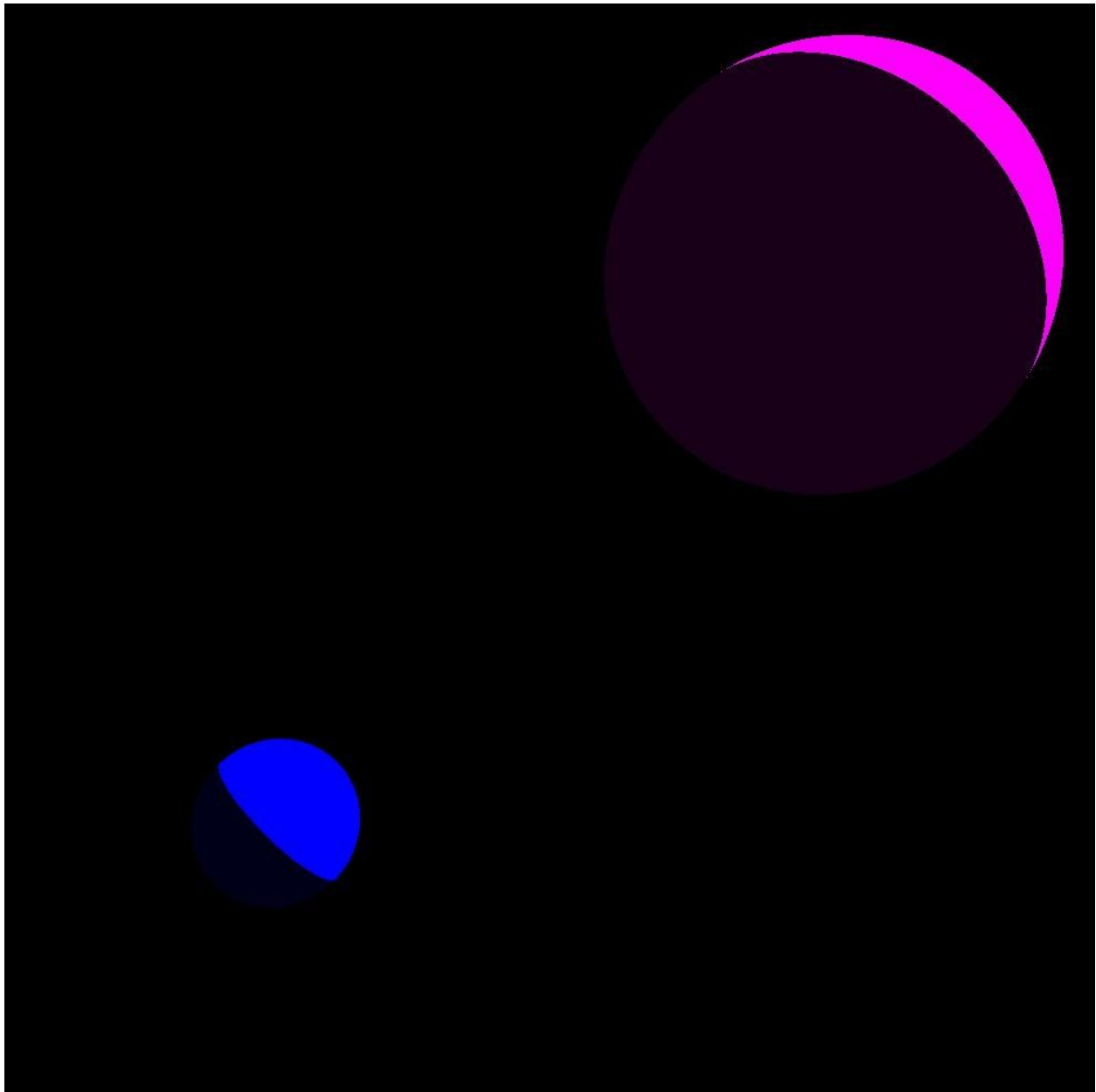
-----

Color of the sphere 3 in R G B format: 255 0 255

Position (x, y, z) of the sphere 3: 50 50 200

Radius of the sphere 3: 40






#### Example 4:

The number of spheres: 4

Color of the sphere 1 in R G B format: 255 255 0

Position (x, y, z) of the sphere 1: 50 50 300

Radius of the sphere 1: 20



---

Color of the sphere 2 in R G B format: 255 0 255

Position (x, y, z) of the sphere 2: -50 50 300

Radius of the sphere 2: 20

---

Color of the sphere 3 in R G B format: 0 255 255

Position (x, y, z) of the sphere 3: 50 -50 300

Radius of the sphere 3: 20

---

Color of the sphere 4 in R G B format: 0 0 255

Position (x, y, z) of the sphere 4: -50 -50 300

Radius of the sphere 4: 20

