**Ray Tracing Report:**

I used the method that is “for each pixel of the screen”, so in this method, all pixels in the image are controlled and processed.

I usually used numpy library because it makes array controls and mathematical processes easy. Therefore, I usually use numpy library in my main code and the methods.

In main code, I use a tricky\_point to find true intersection point and do not to find same previous nearest intersection point, so I use an eligible value (10^-5) for tricky\_point to shift.

Methods:

* create\_sphere(N):

It provides to create spheres by using information that taken from users. Return array of spheres.

* normalVector(vector):

It provides to find the normal of the vector.Return the normal of the vector.

* reflectionVector(vector, normal):

It provides to find reflected vector of the vector by using the normal (normal of surface). Return the reflection vector of the vector from the surface.

* intersectionSphere(center, radius, ray\_origin, ray\_direction):

It provides to detects intersections between a ray and a sphere. Return the distance from the origin of the ray.

* nearest\_intersected\_object(objects, ray\_origin, ray):

It provides to find the nearest object that a ray intersects. Return nearest object and its distance.

* illumination\_calculator(nearestObject, light\_intersection, normal\_to\_surface):

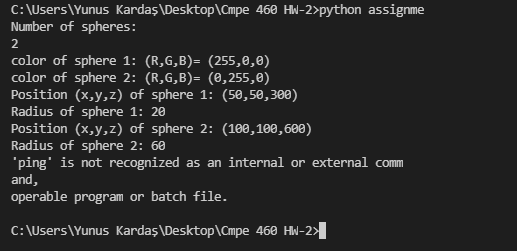
It provides calculation of ambient and diffuse illumination, sum them and return the sum.

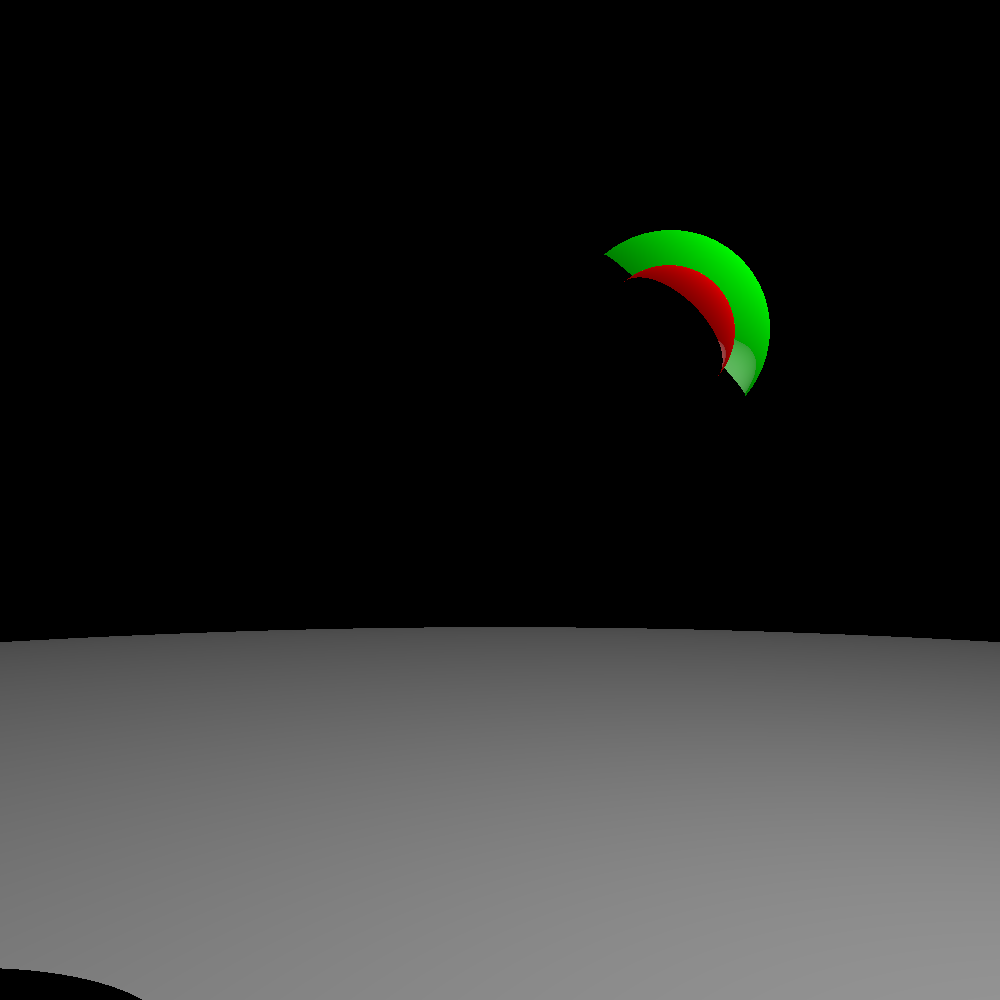
* cast\_ray(direction, color, depth, origin, reflection):

It provides the recursion of the ray tracing by usihn directions of the reflected rays and intersection with nearest object calculations. So, calculations of the colors and directions are made in this function. The depth is checked and if depth is smaller than or equal the max\_depth, then the function make recursion. Return the color value divided by the depth.

Tests:

Test1





Test2:

