Gebze Technical University

Engineering Department

Computer Engineering

Homework #4

CSE 462

Applied Augmented Reality and 3D User Interfaces

Ahmet Yigit

200104004066

1. Abstract

a. This report outlines the development and implementation of a ray-casting renderer in Unity that incorporates the unique influence of black hole physics. The aim is to simulate how light behaves in a gravitationally distorted space while satisfying the assignment requirements, including creating a 3D scene, modeling Lambertian materials, and rendering an image of size 640x480 pixels. The methodology involves utilizing Unity's scripting tools and implementing custom physics for ray trajectories under black hole influence. The results section will present rendered images with and without black holes, illustrating the differences in ray behavior.

2. Abstract

a. Ray casting is a fundamental technique in computer graphics, used to simulate light transport and generate realistic images. In this project, a unique variation of ray casting is implemented to model the effect of black hole physics on light rays. Unlike conventional ray casting, where light travels in straight lines, this project introduces curved ray paths influenced by gravitational forces.

b. The primary objectives of this project are:

i. Build a 3D world with at least 10,000 triangles and Lambertian materials.
ii. Incorporate adjustable light sources and a pinhole camera model.
iii. Implement and render ray paths influenced by black holes.
iv. Compare results between conventional ray casting and black hole ray casting.

3. Methodology

a. Environment Setup

i. The 3D environment consists of:

1. Renderable Objects:
    a. Multiple objects tagged as "Renderable" with Lambertian materials.

2. Light Sources:
    a. Three distinct light sources tagged as "LightSource," each with adjustable intensity and position.

3. Black Hole:
    a. A single black hole object tagged as "blackHole," acting as the gravitational source.

ii. Code

```
private void InitializeComponents()
{
    // Find all object with thags / rend. ls. blackHOLE
    objects = new List<GameObject>(GameObject.FindGameObjectsWithTag("Renderable"));
    if (objects.Count == 0) Debug.LogError("No objects found with 'Renderable' tag!");

    GameObject[] lightObjects = GameObject.FindGameObjectsWithTag("LightSource");
    lightSources = new Light[lightObjects.Length];
    for (int i = 0; i < lightObjects.Length; i++)
    {
        lightSources[i] = lightObjects[i].GetComponent<Light>();
    }
    if (lightSources.Length == 0) Debug.LogError("No lights found with 'LightSource' tag!");

    myBlackHole = GameObject.FindGameObjectWithTag("blackHole");
    if (myBlackHole == null && blackhole) Debug.LogError("No black hole found with 'blackHole' tag!");

    image.texture = texture;
}
```

1. This ensures that all scene elements are dynamically loaded and accessible during runtime.

b. Ray-Casting Algorithm

i. The ray-casting process involves:

1. Generating rays from a pinhole camera with adjustable field of view.
2. Simulating ray trajectories based on the presence or absence of a black hole.
3. Handling intersections with Lambertian surfaces to calculate pixel colors.

ii. Black Hole Influence

1. Under the influence of a black hole, rays follow a quadratic curve, modeled as:

```
// get ray position and direction based on black hole influence -- every update works until my var true => false
currentPos += currentDir * stepSize;
Vector3 toBlackHole = (myBlackHole.transform.position - currentPos).normalized;
float distanceToBlackHole = Vector3.Distance(currentPos, myBlackHole.transform.position);
float influenceFactor = blackHoleInfluence / (distanceToBlackHole * distanceToBlackHole);
currentDir = Vector3.Lerp(currentDir, toBlackHole, influenceFactor).normalized;
```

   a. This code snippet adjusts the direction of each ray incrementally based on its proximity to the black hole, simulating gravitational lensing effects.

iii. Image Rendering

1. The final image is rendered pixel-by-pixel:
   a. Rays intersecting objects contribute color based on Lambertian reflection.

   b. Rays influenced by the black hole curve towards its center.

   c. The resulting image is saved using Unity's Texture2D and file I/O utilities.

2. The image generation logic is as follows:

```
private void RenderWithoutBlackHole()
{
    for (int i = 0; i < imageWidth; i++)
    {
        for (int j = 0; j < imageHeight; j++)
        {
            Ray ray = cam.ViewportPointToRay(new Vector3((float)i / imageWidth, (float)j / imageHeight, 0));
            RaycastHit hit;

            if (Physics.Raycast(ray, out hit, Mathf.Infinity))
            {
                SetPixelColor(i, j, hit);
            }
            else
            {
                texture.SetPixel(i, j, Color.black);
            }
        }
    }

    texture.Apply();
    SaveImage("_no_hole");
}
```

iv. Test Cases for Black Hole Influence

1. A key parameter influencing the ray behavior is blackHoleInfluence. Testing was conducted by varying this parameter from 1 to 10 and observing the resulting distortion in ray trajectories:

    a. blackHoleInfluence = 1: Minimal distortion; rays deviate slightly near the black hole.

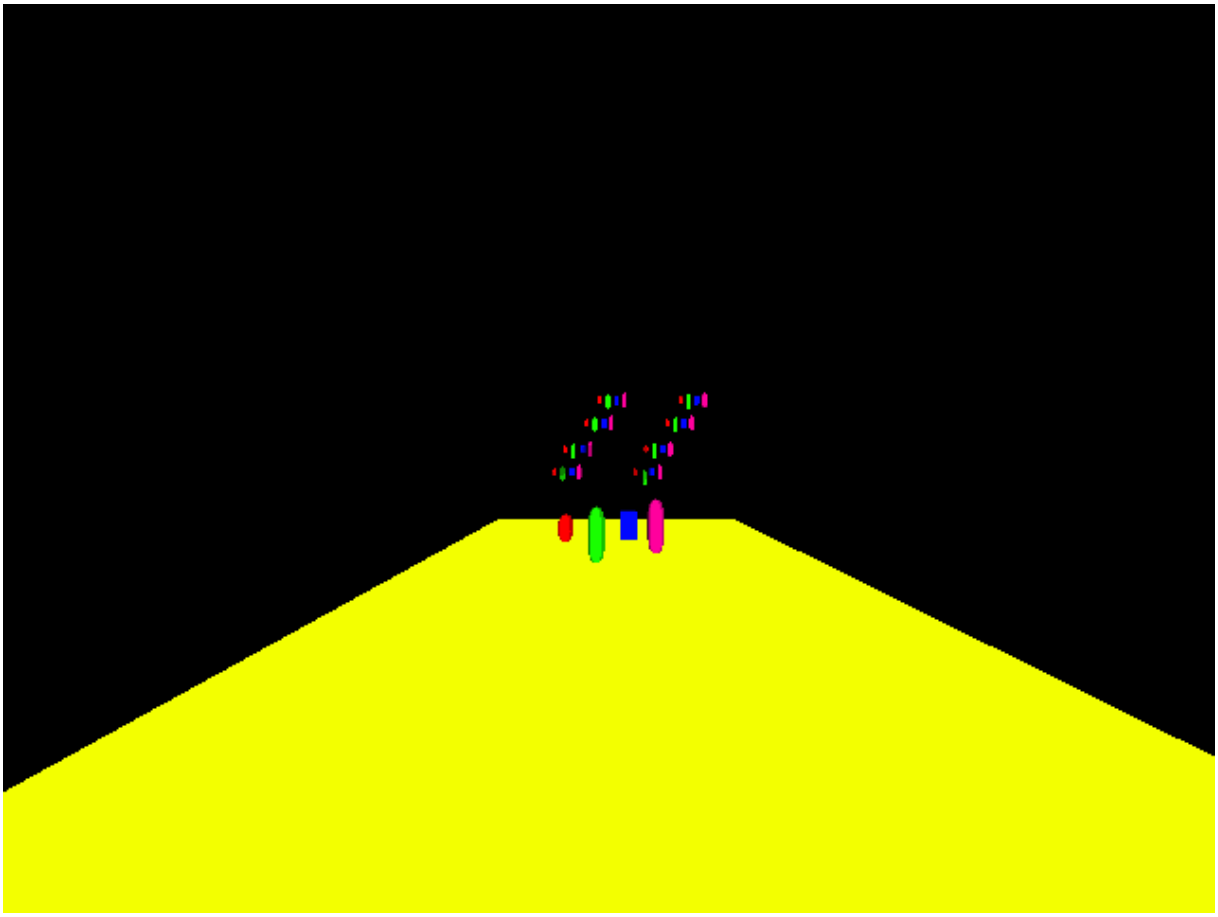
    b. blackHoleInfluence = 5: Noticeable distortion; rays exhibit a pronounced curve.

    c. blackHoleInfluence = 10: Extreme distortion; most rays are captured by the black hole.

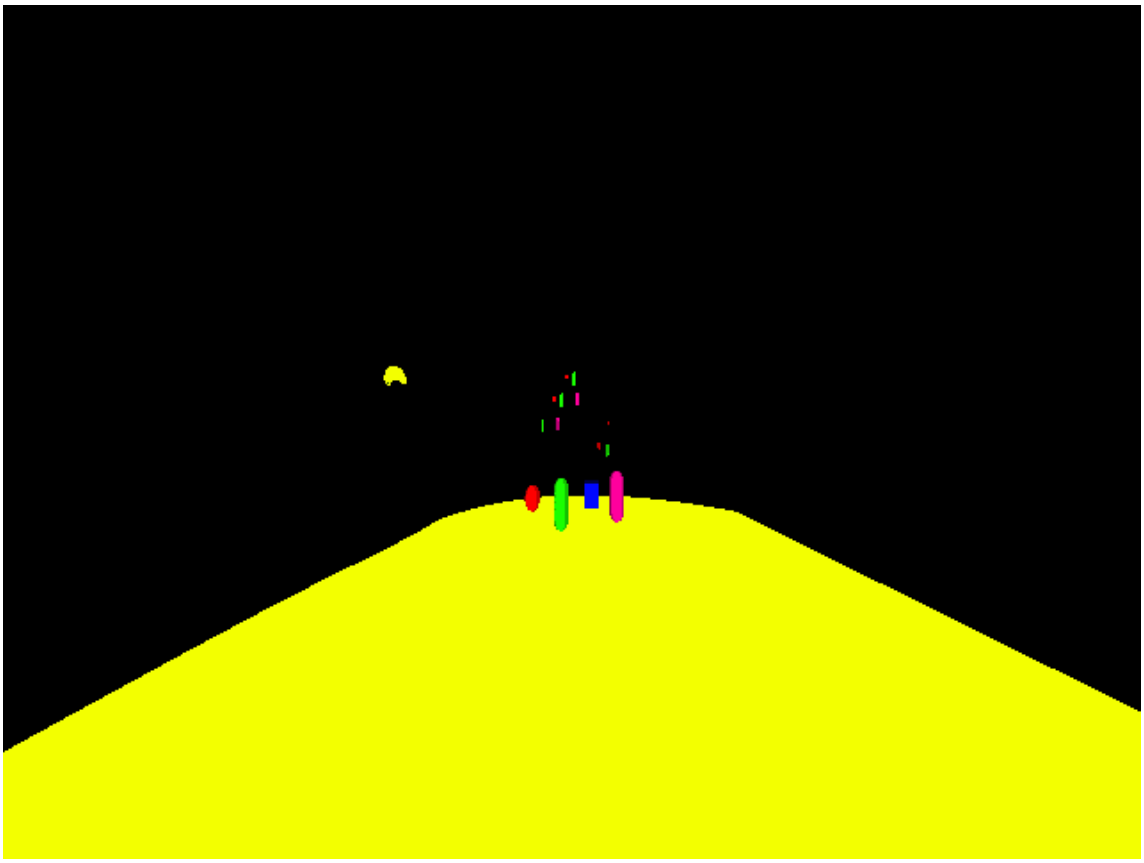
2. This variation directly influences the image output, as shown in the results section.

c. Results

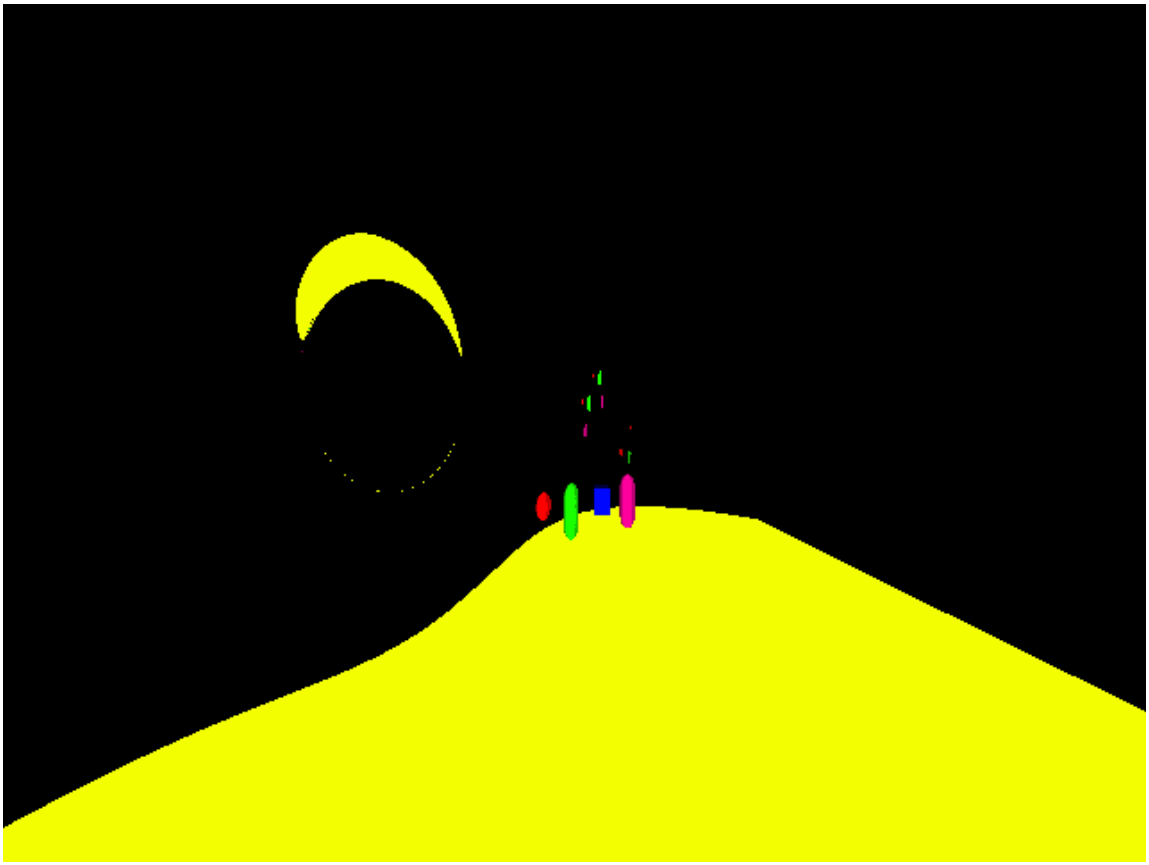
i. Image Without Black Hole



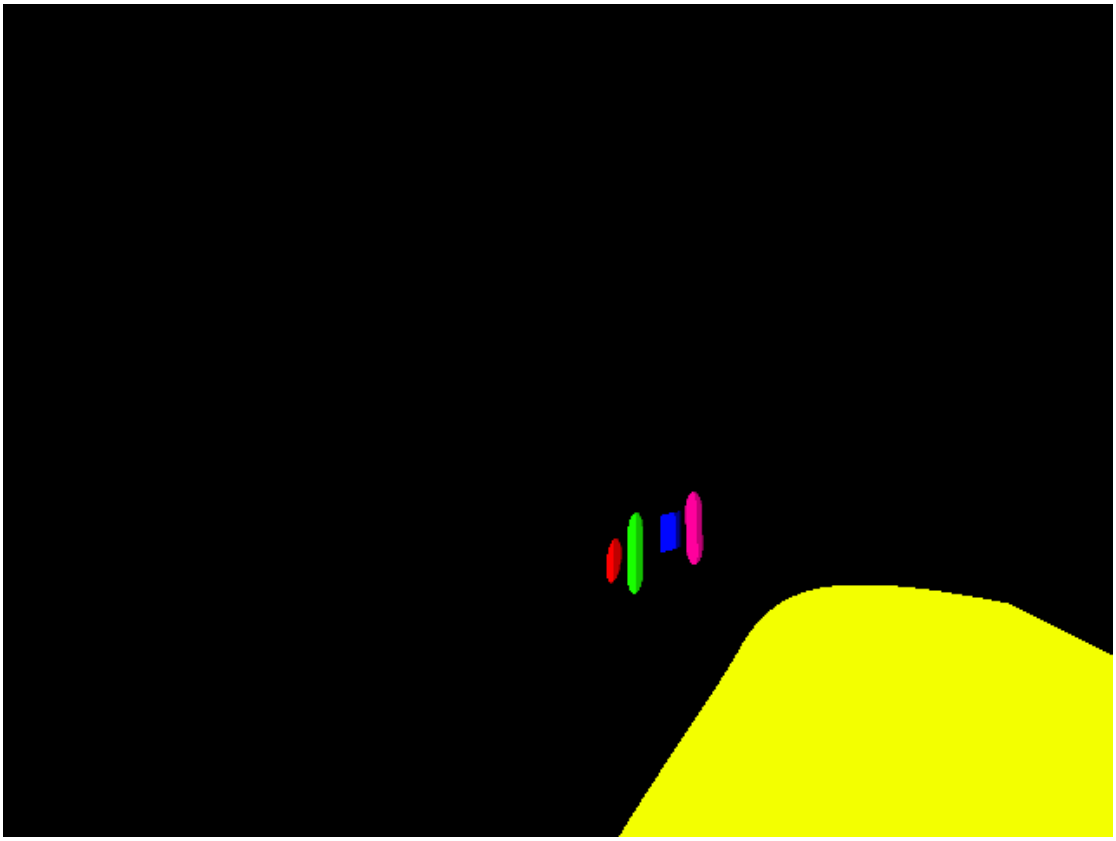
ii. Image With Black Hole (blackHoleInfluence = 0.1)



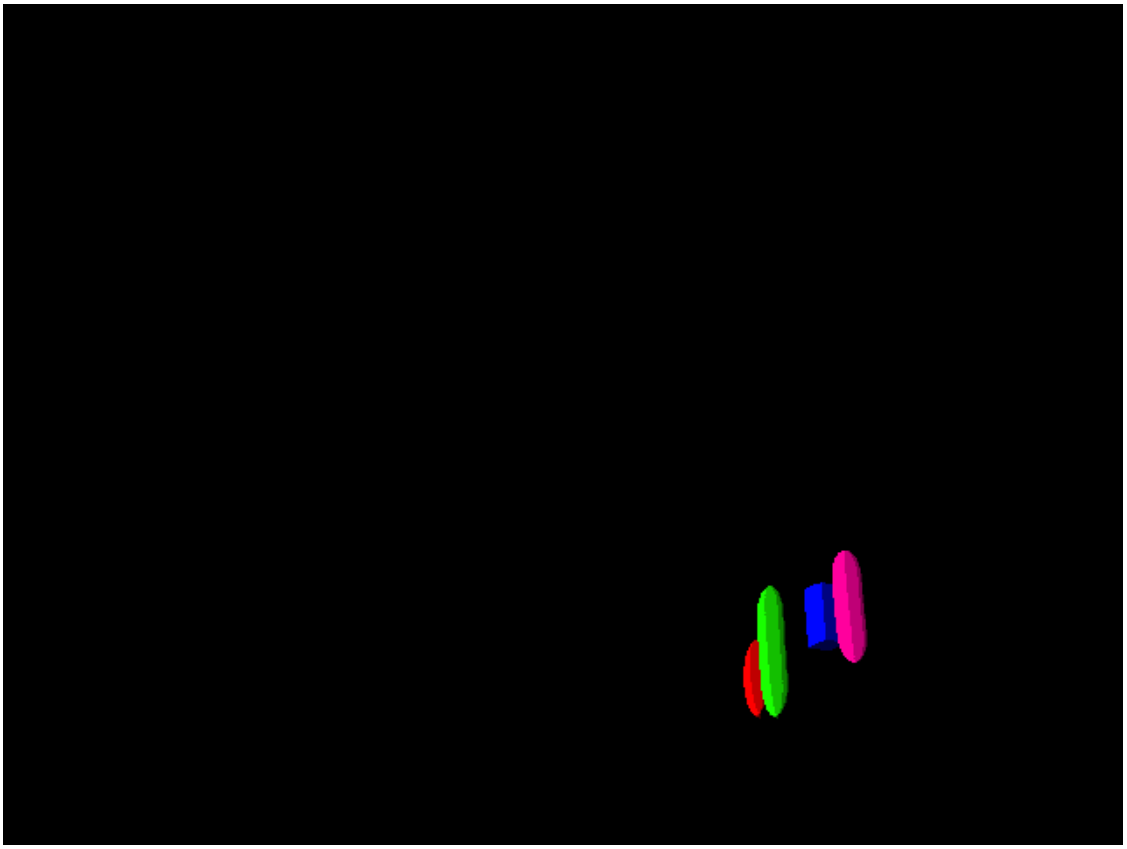
iii. Image With Black Hole (blackHoleInfluence = 1)



iv. Image With Black Hole (blackHoleInfluence = 10)

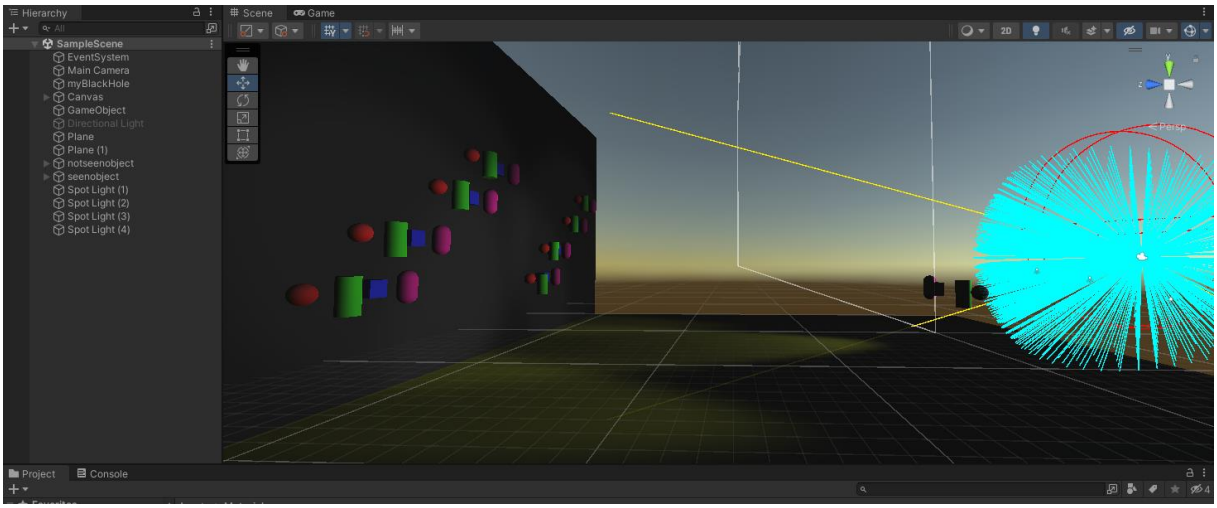


v. Image With Black Hole (blackHoleInfluence = 20)

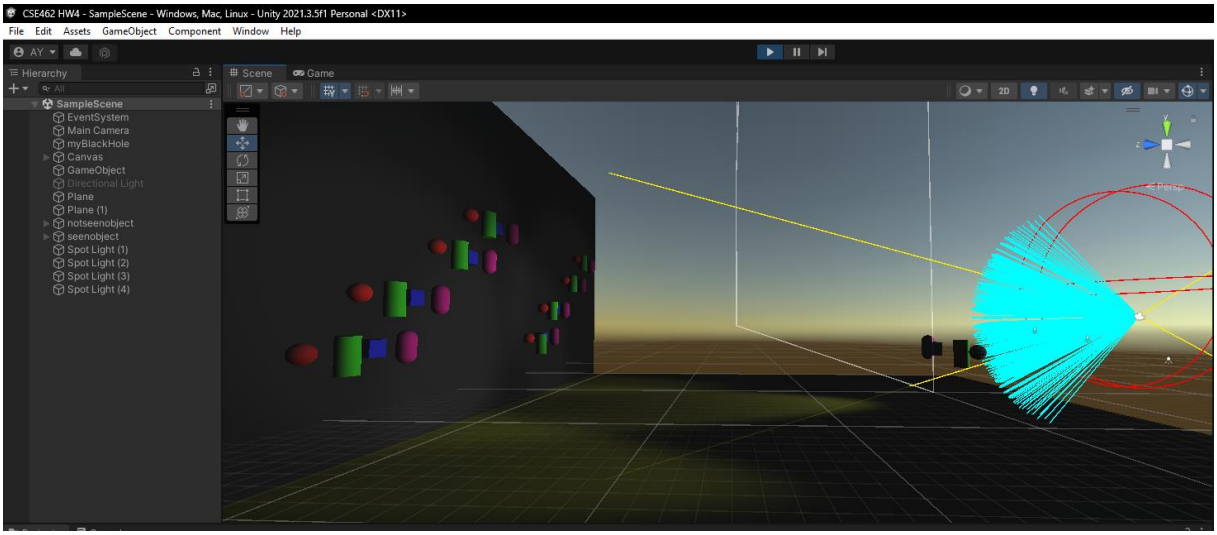


d. Processing of Ray – It depends on CPU/GPU

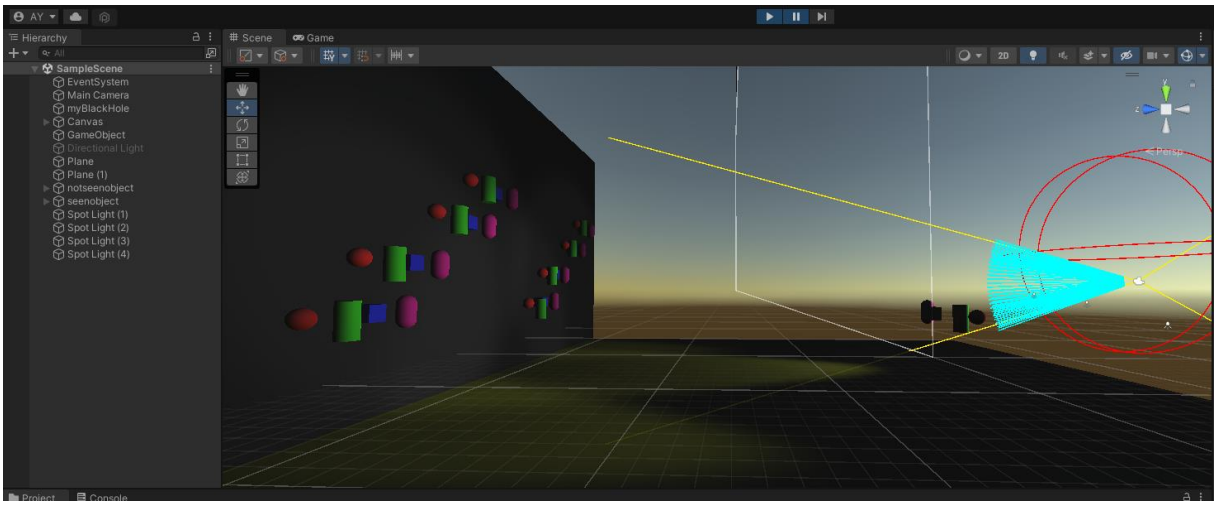
i. Initial [0-1s]



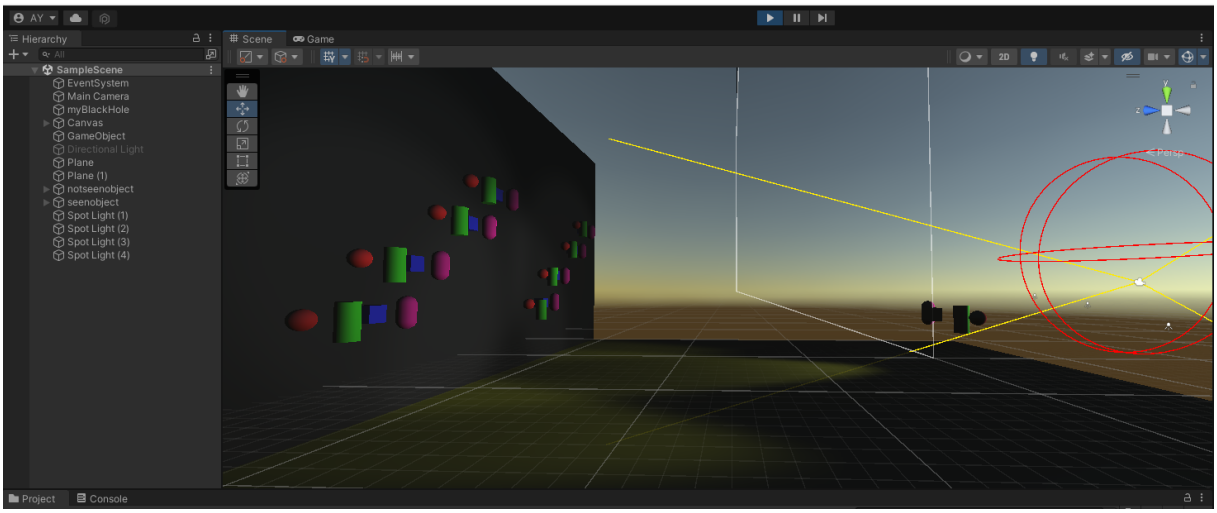
ii. After [1-3] Seconds



iii. After [3-5] Seconds



iv. Final



e. Discussion

i. Observations

1. The images successfully demonstrate the difference in ray trajectories caused by the black hole's influence.

2. As blackHoleInfluence increases, the curvature of ray paths becomes more pronounced, leading to greater pixel distortions near the black hole.

ii. Limitations

1. The images successfully demonstrate the difference in ray trajectories caused by the black hole's influence.

2. As blackHoleInfluence increases, the curvature of ray paths becomes more pronounced, leading to greater pixel distortions near the black hole.

f. Conclusion

i. This project successfully implements a ray-casting renderer that simulates black hole physics in Unity. The results highlight the impact of gravitational forces on light trajectories, providing a compelling visualization of astrophysical phenomena. Testing variations in blackHoleInfluence demonstrated its direct effect on image output, emphasizing the importance of parameter tuning. Future work could include exploring more complex gravitational models and optimizing performance for real-time applications.