GEBZE TECHNICAL UNIVERSITY
COMPUTER ENGINEERING
DEPARTMENT

CSE462 Augmented Reality Fall 2024
Homework 4 Report

Osmancan Bozali
220104004011

Introduction:

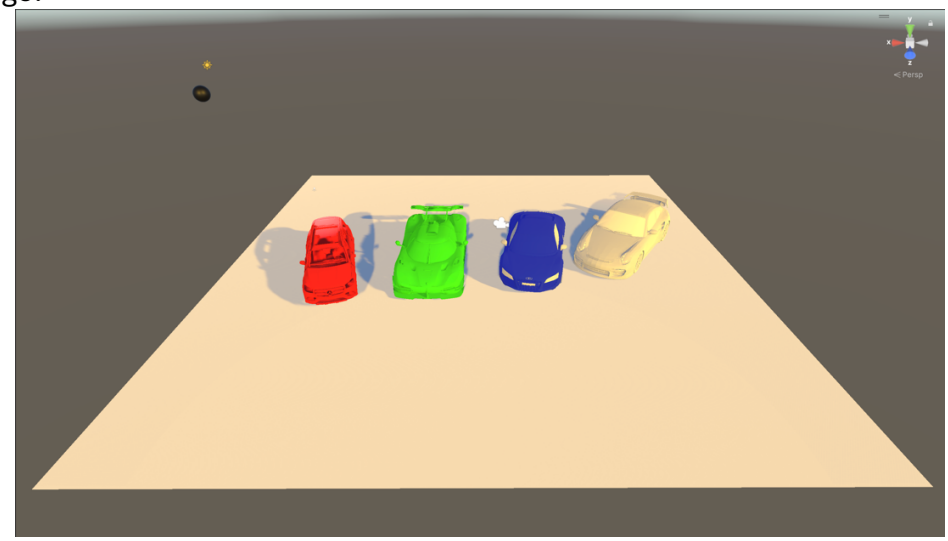
Ray casting is a fundamental technique used in computer graphics to simulate the interaction of light rays with objects in a 3D scene. In this project, we extend traditional ray casting by incorporating the physics of black holes, creating a unique rendering scenario where light rays are curved by gravitational forces. The primary objective of this project is to render a 3D scene consisting of multiple objects with Lambertian materials under two different conditions: one with standard raycasting (straight-line rays) and another influenced by a black hole's gravitational field. The black hole introduces distortion by curving light rays along a quadratic trajectory as they approach the event horizon, creating visually compelling and physically accurate results. To accomplish this, a pinhole camera model with adjustable parameters such as field of view and orientation is used to capture the scene. The project includes at least 10,000 triangles (4 car models), multiple light sources, and adjustable properties to simulate different scenarios.

Methodology:

Scene Setup:

The 3D scene for this project was created in Unity and consists of four distinct objects, each modeled as cars with Lambertian materials. These objects collectively have over 10,000 triangles, meeting the assignment's requirements for geometric complexity. The materials were chosen to reflect light diffusely, providing realistic shading across all objects. The scene is illuminated by three adjustable light sources strategically placed to ensure consistent and well-lit renders. The light intensity and position were made configurable to allow for experimentation with different lighting setups.

Scene Image:



Camera Setup:

The rendering process utilized a pinhole camera model implemented within Unity. The camera's field of view (FoV), orientation, and position were adjustable, offering flexibility in capturing the scene from various perspectives. The camera resolution was fixed at 640x480 pixels, aligning with the assignment's requirements. Rays were cast from the camera's position through a virtual image plane, with each pixel corresponding to a unique ray direction. The camera's parameters ensured that the rendering accurately simulated a realistic perspective projection.

Raycasting Process:

Without Black Hole:

In the *NoBlackHoleRayCaster* script, rays were cast from the camera through each pixel's position in the image plane using Unity's *Physics.Raycast*. Intersections determined pixel colors based on Lambertian shading, producing realistic lighting and shadows without distortion.

```
void RenderScene() {

        for (int y = 0; y < imageHeight; y++) {
            for (int x = 0; x < imageWidth; x++) {
                float viewportX = (float)x / (imageWidth - 1);
                float viewportY = (float)y / (imageHeight - 1);

                Vector3 rayOrigin = mainCamera.transform.position;
                Vector3 rayTarget = mainCamera.ViewportToWorldPoint(new
Vector3(viewportX, viewportY, 1.0f));
                Vector3 rayDirection = (rayTarget - rayOrigin).normalized;

                Color pixelColor = backgroundColor;

                if (Physics.Raycast(rayOrigin, rayDirection, out RaycastHit hit,
maxRayDistance)) {
                    Renderer renderer = hit.collider.GetComponent<Renderer>();
                    if (renderer != null && renderer.material != null) {
                        float diffuse = Mathf.Max(0.2f, Vector3.Dot(hit.normal, -
rayDirection));
                        pixelColor = renderer.material.color * diffuse;
```

```
                    pixelColor += new Color(0.1f, 0.1f, 0.1f, 0f);
                }
            }
            renderedImage.SetPixel(x, y, pixelColor);
        }
    }
    renderedImage.Apply();
    SaveRenderedImage();
}
```

With Black Hole:

The *BlackHoleRayCaster* script introduced gravitational bending, curving rays 2 degrees toward the black hole. Rays were iteratively adjusted to follow quadratic paths, with termination at the event horizon or maximum distance. Object intersections used Lambertian shading, while rays reaching the event horizon resulted in black pixels.

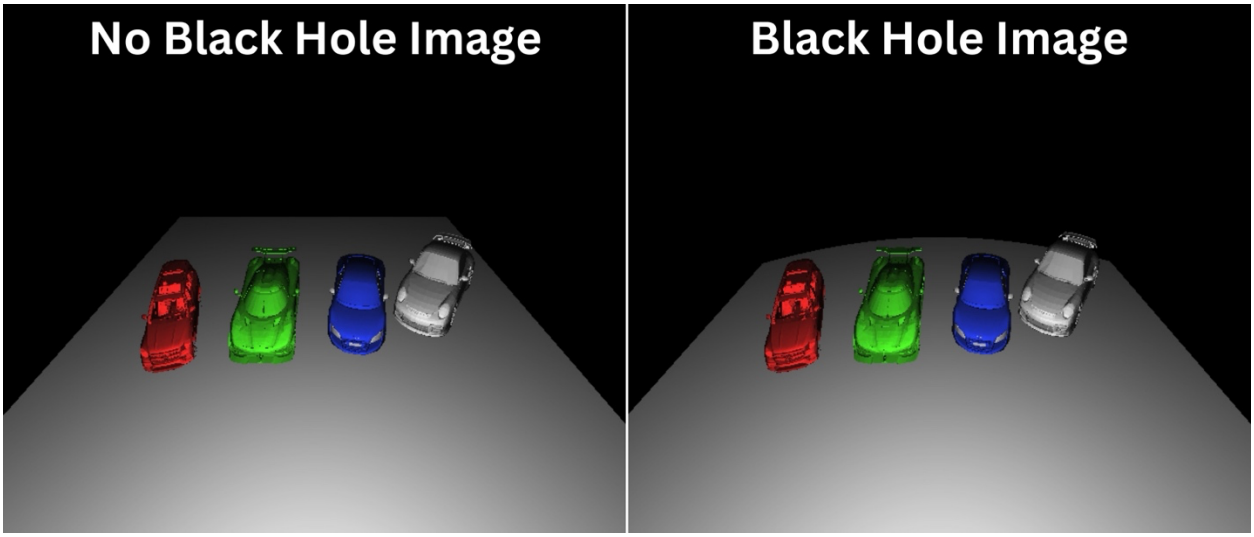
```
void RenderScene() {
    if (blackHole == null) {
        Debug.LogError("Black hole reference not set!");
        return;
    }

    shouldRender = false;
    Texture2D outputTexture = new Texture2D(imageWidth, imageHeight);

    float fov = mainCamera.fieldOfView;
    float aspect = mainCamera.aspect;
    Vector3 cameraPosition = mainCamera.transform.position;
    Vector3 cameraForward = mainCamera.transform.forward;
    Vector3 cameraRight = mainCamera.transform.right;
    Vector3 cameraUp = mainCamera.transform.up;

    for (int y = 0; y < imageHeight; y++) {
        for (int x = 0; x < imageWidth; x++) {
            float normalizedX = (2.0f * x / imageWidth - 1.0f) * aspect;
            float normalizedY = 2.0f * y / imageHeight - 1.0f;
            float tanFov = Mathf.Tan(fov * 0.5f * Mathf.Deg2Rad);
```

```
                Vector3 rayDirection = cameraForward + cameraRight * (normalizedX *
tanFov) + cameraUp * (normalizedY * tanFov);
                rayDirection.Normalize();

                Color pixelColor = backgroundColor;
                Vector3 lastPos = Vector3.zero;

                if (CastCurvedRay(cameraPosition, rayDirection, out RaycastHit hit,
out lastPos)) {
                    Renderer renderer = hit.collider.GetComponent<Renderer>();
                    if (renderer != null && renderer.material != null) {
                        float diffuse = Mathf.Max(0.2f, Vector3.Dot(hit.normal, -
rayDirection));
                        pixelColor = renderer.material.color * diffuse;
                        float distToBlackHole = Vector3.Distance(hit.point,
blackHole.position);
                        float darkening = Mathf.Clamp01(distToBlackHole /
(gravitationalStrength * 2));
                        pixelColor = Color.Lerp(Color.black, pixelColor, darkening);
                        pixelColor += new Color(0.1f, 0.1f, 0.1f, 0f);
                    }
                }
                outputTexture.SetPixel(x, y, pixelColor);
            }
        }

        outputTexture.Apply();
        SaveRenderedImage(outputTexture);

        Destroy(outputTexture);
        shouldRender = true;
    }
```

Rendering Pipeline:

The rendering pipeline cast rays per pixel, calculating colors based on material properties and lighting. Near the black hole, additional darkening effects simulated gravitational intensity. Final images were saved as PNG files for comparison between standard and black hole raycasting.

Results:



The comparison between the two images highlights the differences in a scene rendered with and without the influence of a black hole. In the left image, which uses standard raycasting, the objects retain their original proportions and appear undistorted. The lighting, based on Lambertian shading, provides realistic illumination, and the light paths remain straight, resulting in a uniform and natural perspective.

In contrast, the right image includes a black hole, causing significant visual distortions. (curve of the plane can be seen easily) Light rays bend as they approach the black hole, leading to warped appearances for objects located near its center. This effect is a result of gravitational lensing.

Conclusion:

This project demonstrates the impact of a black hole on raycasting in a 3D scene. By simulating gravitational lensing, the rendered images highlight how rays bend under the influence of a black hole, creating distortions. The comparison between standard raycasting and black hole-influenced rendering illustrates the significant effects of gravitational physics on visual perception.