CSE462 – HW4
Muhammed Sefa Cahyir - 1801042686

Overview

The goal of this assignment is to develop a ray caster in Unity that simulates light-ray physics in a 3D environment with unusual properties, such as the influence of black holes that bend rays along a quadratic curve. Additionally, we aim to create a 3D world with realistic lighting and materials.

Implementation Details

Ray Physics and Black Hole Influence

- **Ray Casting Logic:** Rays are cast from the starting points. If a ray encounters a black hole's gravitational influence, its trajectory is altered by applying a quadratic curve.
- **Key Script Highlights:**
  - `MultiRayBouncer.cs`: This script manages ray tracing, black hole interaction, and rendering.
  - **Gravity Effect:** The method `ApplyBlackHoleGravity` calculates the influence of the black hole on the ray's trajectory using custom physics.

3D World Setup

- Created with Unity editor tools:
  - Four distinct objects with a total triangle count exceeding 10,000.
  - Lambertian material applied to each object.
  - Three directional light sources were placed, with positions and intensities adjustable in the Unity Inspector.

Camera System

- Implemented a pinhole camera with the following adjustable parameters:
  - Field of View (FoV).
  - Center and viewing direction.

```
for (int i = 0; i < maxBounces; i++)
{
    if (applyGravity && physics.blackHole != null)
    {
        currentDirection = ApplyBlackHoleGravity(currentPosition, currentDirection, rayIndex);
    }

    Ray ray = new Ray(currentPosition, currentDirection);

    if (Physics.Raycast(ray, out RaycastHit hit, rayLength))
    {
        positions.Add(hit.point);
        currentDirection = Vector3.Reflect(currentDirection, hit.normal);
        currentPosition = hit.point;

        if (isBlackHoleActive)
        {
            positions.Add(physics.blackHole.position);
            break;
        }
    }
    else
    {
        if (isBlackHoleActive)
        {
            positions.Add(physics.blackHole.position);
            break;
        }

        Vector3 nextPosition = currentPosition + currentDirection * rayLength;
        positions.Add(nextPosition);
        break;
    }
```

Ray Bouncing and Rendering

- Configured rays to:
  - Bounce a maximum of 5 times.
  - Stop upon hitting a black hole or reaching maximum length.
  - Reflect based on the hit surface's normal vector.

Line Rendering

- Used Unity's `LineRenderer` to visually represent the rays in the scene.
- Prefabs were instantiated for each ray, and their positions were updated in real-time based on the ray's calculated trajectory.

Black Hole

```
using UnityEngine;

3 references
public class BlackHolePhysics : MonoBehaviour
{
    5 references
    public Transform blackHole;
    3 references
    public float blackHoleRadius = 5f;

    3 references
    public Vector3 CalculateRay(Vector3 origin, Vector3 direction)
    {
        // Kara deliğe olan mesafe
        Vector3 toBlackHole = blackHole.position - origin;
        float distance = toBlackHole.magnitude;

        if (distance < blackHoleRadius)
        {
            // Kara deliğe ulaşıldı
            return blackHole.position;
        }

        // Eğrilik etkisi (derece 2 eğri)
        Vector3 curve = direction + toBlackHole.normalized * Mathf.Pow(distance / blackHoleRadius, 2);
        return origin + curve.normalized * distance;
    }
}

```

Screenshot and Video Link (https://youtu.be/gVxnqlJncmo