Yakup Talha Yolcu
1801042609
CSE462 AR
HW4

Ray casting involves sending rays from the camera through each pixel of the image, and checking if any of the objects in the scene intersect with the ray. If an object intersects with the ray, the pixel is colored based on the material and lighting of the object. If no objects intersect with the ray, the pixel is colored black.

objects is an array of GameObjects, which are the objects in the scene that will be rendered in the image.

cam is a Camera object that represents the camera from which the image will be rendered.

imageWidth and imageHeight are integers that specify the dimensions of the image in pixels.

fov is a float that represents the field of view of the camera.

aspectRatio is a float that represents the aspect ratio of the image (the ratio of the width to the height).

lightSources is an array of Light objects, which represent the light sources in the scene.

image is a RawImage UI element, which is a display element that can display a texture.

allRays is an array of all rays used in blackhole mode

allHits is an array of all bool variables that represents hits in blackhole mode

blackHole is a bool variable that represents blackhole mode

myBlackHole is blackhole object

In the **Start()** function, which is called when the script starts, the following steps are taken to render the image:

First calculate triangle counts

```
        //cube 12 triangle
        //capsule 832
        //cylinder 80
        //plane 200
        //sphere 768
        int totalTrianglecount=0;
        foreach(GameObject obj in objects) {

if(obj.GetComponent<MeshFilter>().sharedMesh.name.Equals("Cube")) {
                totalTrianglecount+=12;
            }
            else
if(obj.GetComponent<MeshFilter>().sharedMesh.name.Equals("Capsule")) {
                totalTrianglecount+=832;
            }
            else
if(obj.GetComponent<MeshFilter>().sharedMesh.name.Equals("Cylinder")) {
                totalTrianglecount+=80;
            }
            else
if(obj.GetComponent<MeshFilter>().sharedMesh.name.Equals("Sphere")) {
                totalTrianglecount+=768;
            }
        }
        Debug.Log("Calculated triangle Count = "+totalTrianglecount);

        int totalTriangles = 0;
        foreach(MeshFilter mf in
FindObjectsOfType(typeof(MeshFilter))){
            totalTriangles += mf.mesh.triangles.Length;
        }
        Debug.Log("Other triangle Couns?="+totalTriangles);
        Debug.Log("Other triangle
Couns?="+UnityEditor.UnityStats.triangles);
```

A new **Texture2D** object called texture is created with the dimensions specified by **imageWidth and imageHeight.**
The image object's texture property is set to the texture object.

if blackhole mode is on, we first initialize the allRays and allHits arrays.
if blackhole mode is off, then we call withoutBlackHole() function and do the followings:

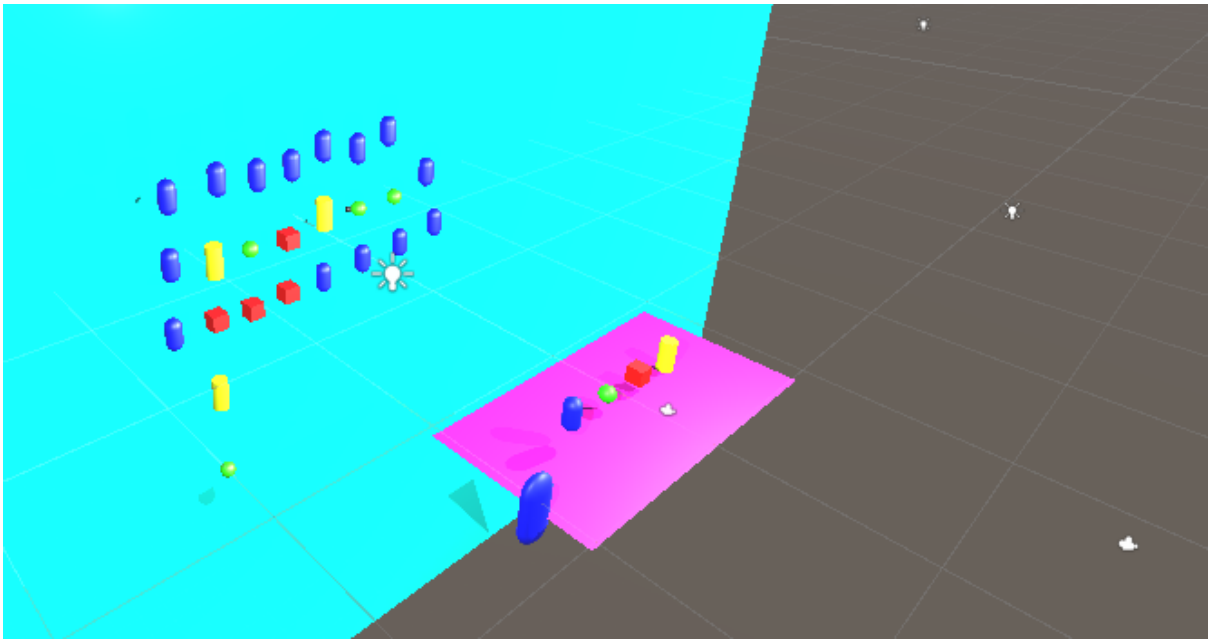
```
    void Start()
    {
        //init texture and image
        texture = new Texture2D(640, 480);
        image.texture = texture;
        checkTriangleCount();
        //black hole mode off
        if(!blackhole) {
            //run normal raycasting
            withoutBlackHole();
        }
        else {
            //black hole mode on, first initialize rays and possible
hits
            for (int i=0;i<imageWidth; i++){
                for(int j = 0; j < imageHeight; j++){
                    allRays[i,j]=cam.ViewportPointToRay(new
Vector3((float)i/imageWidth, (float)j/imageHeight, 0));
                    allHits[i,j]=false;
                }
            }
        }
    }
```

A nested loop iterates over each pixel in the image.
For each pixel, a Ray object called ray is created by calling the **ViewportPointToRay** function on the cam object and passing it a Vector3 with the normalized x and y coordinates of the pixel and a z coordinate of 0. This creates a ray that passes through the center of the pixel.
The **Physics.Raycast** function is called with the ray object and a **RaycastHit** object called hit as arguments. This function shoots the ray and checks if it intersects with any colliders in the scene. If the ray intersects with an object, the hit object is populated with information about the intersection point, the normal of the surface at the intersection point, and the collider that was hit. If the ray does not intersect with any objects, the hit object is not modified.

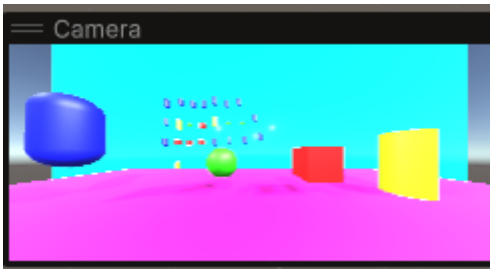
If the **Physics.Raycast** function returns true, meaning that the ray intersected with an object, the pixel is colored based on the material and lighting of the object. First, a variable called **hitCount** is initialized to 0. This variable will be used to keep track of how many light sources are hitting the intersection point. Then, a loop iterates over each Light object in the lightSources array. For each light source, the sendRay function is called with the light source and the intersection point as arguments. If the **sendRay** function returns 1, it means that the light source is hitting the intersection point, so **hitCount** is incremented. After the loop finishes, the **newColor** variable is set to the color of the object's material. If hitCount is 0, meaning that no light

After that, we create file to save the png file to given directory

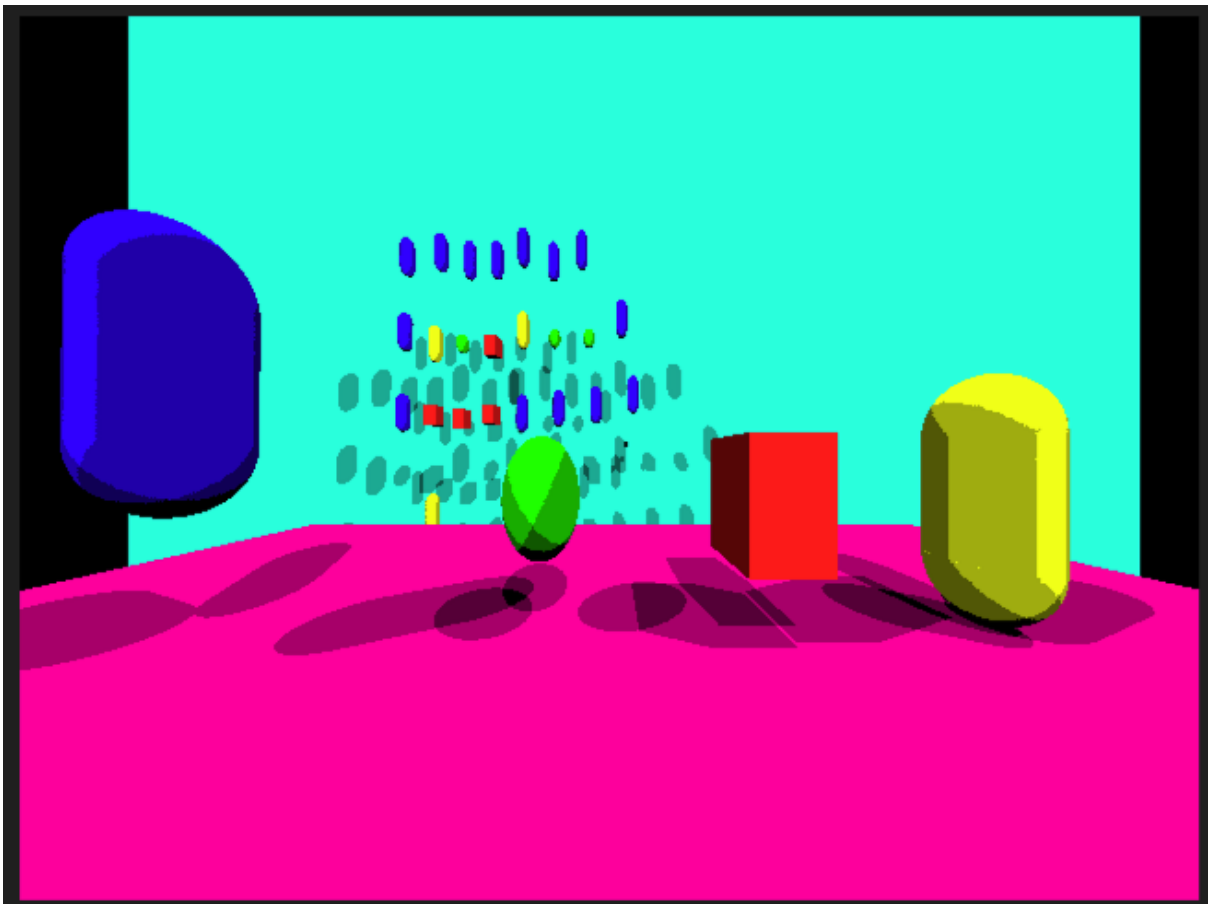
Here is an example of scene and generated image:
Scene:



That's what camera sees:



Generated image (without black hole mode):



```
    private void withoutBlackHole() {
        //in x coordinates
        for (int i=0;i<imageWidth; i++)
        {
            //in y coordinates
            for(int j = 0; j < imageHeight; j++)
            {
                //camera sends a ray to the direction
                Ray ray=cam.ViewportPointToRay(new
Vector3((float)i/imageWidth, (float)j/imageHeight, 0));

                //there is a hit on that ray
                RaycastHit hit;
                if (Physics.Raycast(ray, out hit, Mathf.Infinity))
                {
                    //lights will send the rays to that hit point
                    int hitCount=0;
                    Vector3 hitP=hit.point;
                    //get color the change the color intensity
```

```
                    Color
newColor=hit.collider.GetComponent<Renderer>().material.color;
                    //each light sends a ray to the pointa
                    foreach(Light myLight in lightSources) {
                        hitCount+=sendRay(myLight,hitP);
                    }
                    //ratio the color and hitcount depend on the light
source count
                    newColor.r=(newColor.r*hitCount)/3;
                    newColor.g=(newColor.g*hitCount)/3;
                    newColor.b=(newColor.b*hitCount)/3;

                    //set the pixel color
                    texture.SetPixel(i, j, newColor);
                }
                else
                {
                    texture.SetPixel(i,j,Color.black);
                }
            }
        }
        //apply texture, ray casting is done
        texture.Apply();
        createImageFile("_without_black_hole");
    }
```

```
private void createImageFile(string s) {
        //if directory does not exists, create it
        if(!Directory.Exists(Application.dataPath+"/GeneratedImages"))
{

Directory.CreateDirectory(Application.dataPath+"/GeneratedImages");
        }

        //refresh database
        AssetDatabase.Refresh();

        //take filepath
        string filePath =
Application.dataPath+"/GeneratedImages/image"+s+".png";

        // Encode the texture into a .png file
        byte[] bytes = texture.EncodeToPNG();
```

```
        // Create a new file at the specified location, overwriting any
file with the same name
        FileStream fileStream = File.Create(filePath);

        // Write the .png file to the file stream
        fileStream.Write(bytes, 0, bytes.Length);

        // Close the file stream
        fileStream.Close();

        //refresh database
        AssetDatabase.Refresh();

        Debug.Log("Image is generated");
    }


    private int sendRay(Light light,Vector3 hitP) {

        RaycastHit hit;
        int hit_flag=0;
        //take direction
        Vector3 dir = hitP-light.transform.position;
        dir.Normalize();
        Ray ray = new Ray(light.transform.position,dir);
        if (Physics.Raycast(ray , out hit, Mathf.Infinity)){
            if(Vector3.Distance(hit.point,hitP)<=0.001) {
                hit_flag=1;
            }
            if(hit.point==hitP) {
                hit_flag=1;
            }
        }
        return hit_flag;
    }
```

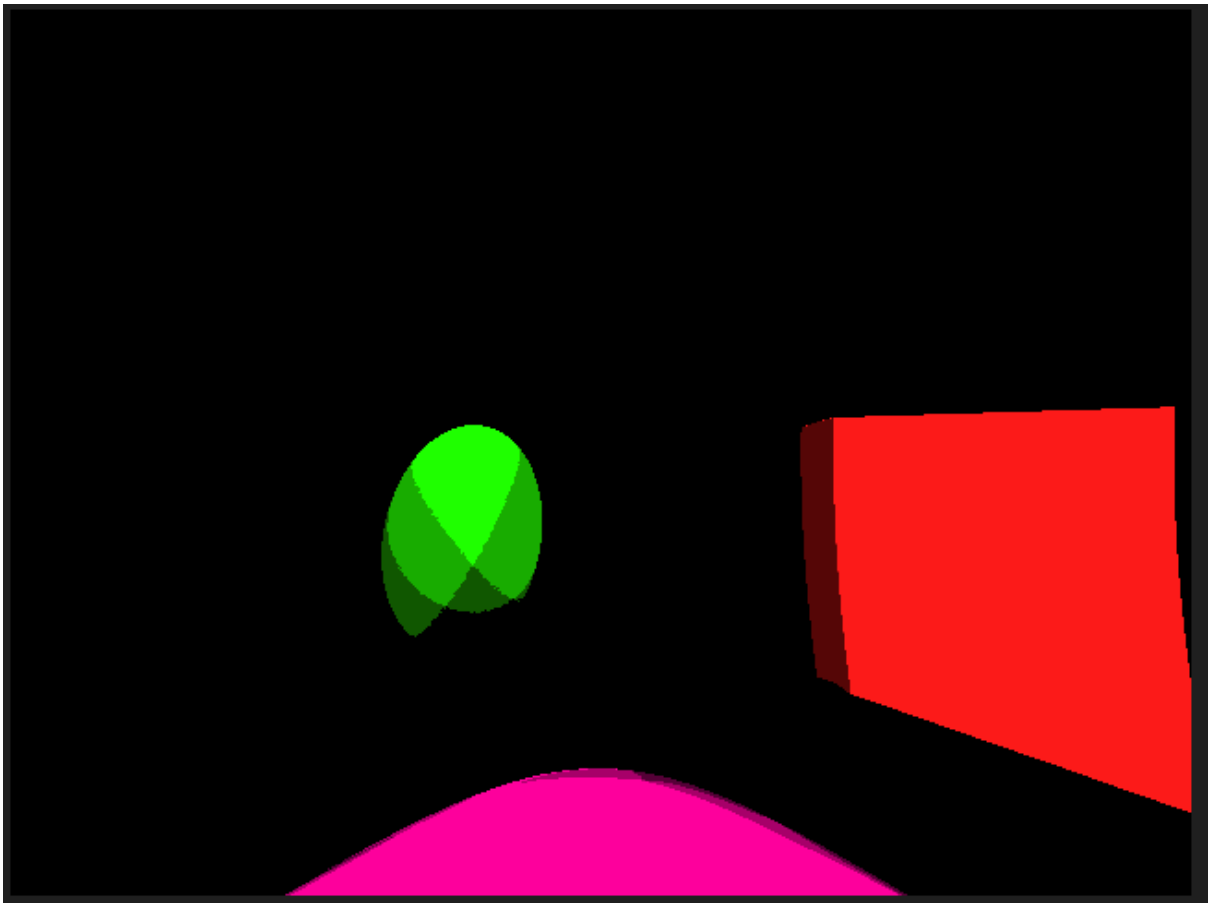
If black hole mode is on, after initializing arrays, we use FixedUpdate() function to iterate rays.

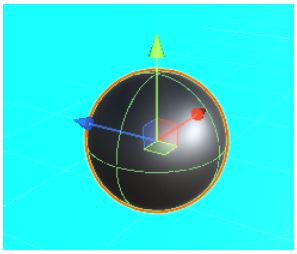
```
    void FixedUpdate()
    {
        //if black hole mode is no, then send rays to create frame
        if(blackhole) {
            if(withBlackHole()) {
                createImageFile("_with_black_hole");
                blackhole=false;
            }
        }
    }
```

```
private bool withBlackHole() {
        bool Finito=true;
        for (int x = 0; x < imageWidth; x++){
            for (int y = 0; y < imageHeight; y++){
                //if there is no hit in that pixel
                if (!allHits[x,y]){
                    RaycastHit hit;
                    //send ray
                    if (Physics.Raycast(allRays[x,y], out hit,1)){
                        //hit
                        allHits[x,y]=true;
                        int hitCount=0;
                        Color color=
hit.collider.GetComponent<Renderer>().material.color;
                        for (int k = 0; k < 3; k++){

hitCount+=sendRay(lightSources[k],hit.point);
                        }
                        Color newColor= new
Color(hitCount*(color.r/3),hitCount*(color.g/3),hitCount*(color.b/3));
                        texture.SetPixel(x, y,newColor);
                    }
                    else{
                        Finito=false;

allRays[x,y].origin=allRays[x,y].origin+(allRays[x,y].direction.normali
zed);
                        Vector3
newDirection=(myBlackHole.transform.position-allRays[x,y].origin);
```

```
                        Vector3 dir=newDirection.normalized+
(allRays[x,y].direction);
                        allRays[x,y].direction=dir.normalized;
                    }
                }
            }
        }
        texture.Apply();
        return Finito;
    }
```

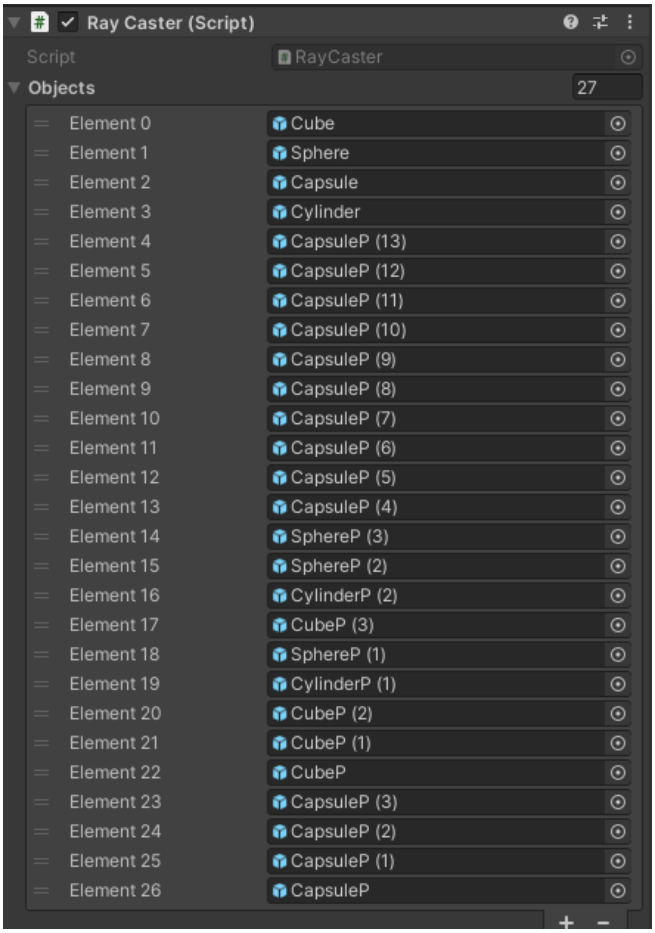
My image with black hole mode on:





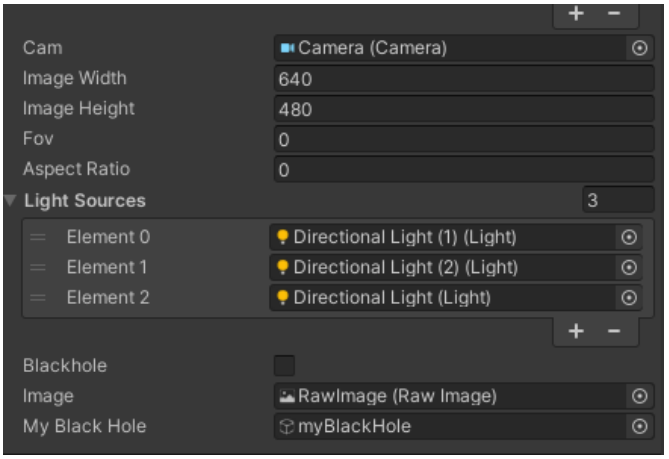
My blackhole object is in space but it is very small object

In blackhole mode, first we check the pixel hit, then send a ray from camera to opposite direction depend on the pixels. if it hits,then we get the color and send rays from light sources to that point. and set color, but if it does not hits, then we regenerate the ray with the curved shape and go to the next pixel

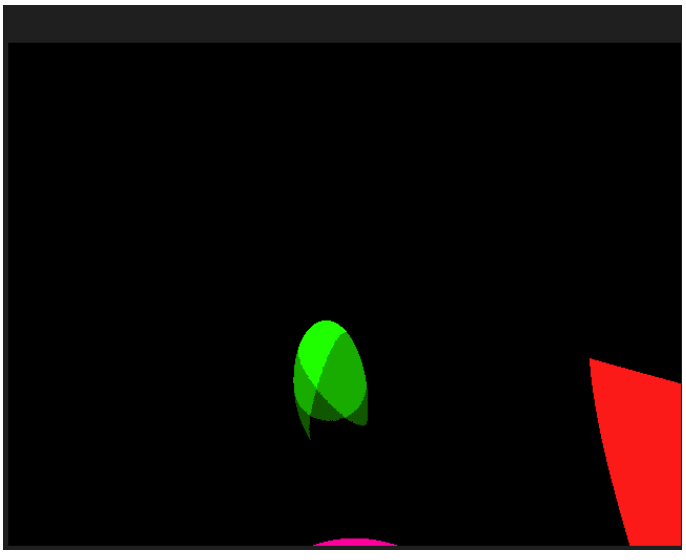


There are more than 4 objects with total triangle more than 10k, Camera's fov,center, and can be adjusted from the unity editor, there are 3 lightsources with adjustable intensity and position.

Other references:



Other test results:



Blackhole place changed

Camera place changed