1. potrebne strukture u shaderu

```
// materijal od kojeg se sastoji pojedini objekt
struct Material {
    float4 color;
    float emmission;
    float4 emmissionColor;
};

// kugla
struct Sphere {
    float3 position;
    float radius;
    Material material;
};

// zraka
struct Ray {
    float3 position;
    float3 direction;
    float4 color;
};

// podatci o tocki koju je zraka pogodila
struct HitInfo {
    bool didHit;
    float3 position;
    float distance;
    float3 normal;
    Material material;
};
```

2. potrebne strukture u C# program

```
// klasa koja sluzi samo za sucelje koje korisnik vidi
[Serializable] // oznacava da hocemo editirati u inspektoru
public class Sphere
{
    public Transform transform;
    public Color color;
    public float emmission;
    public Color emmissionColor;
}

// strukture ekvivalentne onima iz shadera
private struct RayTraceMaterial
{
    public RayTraceMaterial(Color color, float emmission, Color
emmissionColor)
    {
        this.color = color;
        this.emmission = emmission;
        this.emmissionColor = emmissionColor;
    }

    Color color;
    float emmission;
    Color emmissionColor;
};

private struct SphereStruct
```

```
{
    public SphereStruct(Sphere s)
    {
        position = s.transform.position;
        radius = s.transform.lossyScale.x / 2;
        material = new RayTraceMaterial(s.color, s.emmission,
s.emmissionColor);
    }

    Vector3 position;
    float radius;
    RayTraceMaterial material;
};
```

3. pozicija piksela u prostoru

   C#

```
// izracunamo dimenzije near clip plane-a
float planeHeight = Camera.main.nearClipPlane *
Mathf.Tan(Camera.main.fieldOfView * 0.5f * Mathf.Deg2Rad) * 2f;
float planeWidth = planeHeight * Camera.main.aspect;
mat.SetVector("_NearPlane", new Vector3(planeWidth, planeHeight,
Camera.main.nearClipPlane));

// proslijedimo object to world space matricu kamere
mat.SetMatrix("_CameraObjectToWorldMat",
Camera.main.transform.localToWorldMatrix);
```

   HLSL

```
// izracunamo poziciju koja odgovara trenutnom pikselu na near ravnini
float3 rayPoint = float3(i.uv - 0.5, 1) * _NearPlane;
rayPoint = mul(_CameraObjectToWorldMat, float4(rayPoint, 1));
```

4. napišimo funkciju *castRay* koja pronalazi točku koju zraka pogodi

```
HitInfo raySphereIntersection(Ray ray, Sphere s)
{
    HitInfo hit;

    float3 L = s.position - ray.position; // vektor od izvora zrake do
centra kugle
    float Tca = dot(L, ray.direction); // udaljenost od izvora zrake do
tocke na zraki koji je pod pravim kutom od centra

    // gleda na komplet drugu stranu ray
    if (Tca < 0.0)
    {
        hit.didHit = false;
        return hit;
    }

    float d = sqrt(length(L) * length(L) - Tca * Tca);
    if (d > s.radius)
    {
        hit.didHit = false;
        return hit;
    }
```

```
    float Thc = sqrt(s.radius * s.radius - d * d); // udaljenost od
intersectiona do tocke na zraki koja je pod pravim kutom od centra
    float3 intersection = ray.position + ray.direction * (Tca - Thc);

    hit.didHit = true;
    hit.position = intersection;
    hit.distance = length(ray.position - intersection);
    hit.normal = normalize(intersection - s.position);
    hit.material = s.material;
    return hit;
}
```

5. prebacimo kugle s CPU-a u shader
6. PRIKAŽEMO KUGLE
7. generiranje slučajnih brojeva (**zašto?**)
    a. generiranje slučajnog broja 0-1
    b. generiranje slučajnog smjera

```
// funkcija koja generira pseudonasumican broj od 0 do 0xffffffff (2^32-
1) za pojedini state
// inout oznacava da ce promijena na state biti odrazena i u pozivajucoj
funkciji (kao da koristimo pointer)
uint nextRand(inout uint state)
{
	state = state * 747796405 + 2891336453;
	uint result = ((state >> ((state >> 28) + 4)) ^ state) *
277803737;
	result = (result >> 22) ^ result;
	return result;
}
```

8. napišemo *trace* funkciju :)
9. strukture za trokute

```
struct Triangle
{
    float3 posA;
    float3 posB;
    float3 posC;
    float3 normalA;
    float3 normalB;
    float3 normalC;
};

struct Mesh
{
    int firstTriangleIdx;
    int numTriangles;
    Material material;
};
```

```
private struct MeshStruct
{
    public MeshStruct(RayTracingMesh mesh, int firstTriangleIdx, int
numTriangles)
    {
```

```
        this.firstTriangleIdx = firstTriangleIdx;
        this.numTriangles = numTriangles;
        this.material = new RayTraceMaterial(mesh.color, mesh.emmission,
mesh.emmissionColor);
    }

    int firstTriangleIdx;
    int numTriangles;
    RayTraceMaterial material;
}

private struct TriangleStruct
{
    public TriangleStruct(Vector3 posA, Vector3 posB, Vector3 posC,
Vector3 normalA, Vector3 normalB, Vector3 normalC)
    {
        this.posA = posA;
        this.posB = posB;
        this.posC = posC;
        this.normalA = normalA;
        this.normalB = normalB;
        this.normalC = normalC;
    }

    Vector3 posA;
    Vector3 posB;
    Vector3 posC;
    Vector3 normalA;
    Vector3 normalB;
    Vector3 normalC;
}
```

```
        HitInfo rayTriangleIntersection(Ray ray, Triangle tri, Material mat)
        {
          float3 edgeAB = tri.posB - tri.posA;
          float3 edgeAC = tri.posC - tri.posA;
          float3 normalVector = cross(edgeAB, edgeAC);
          float3 ao = ray.position - tri.posA;
          float3 dao = cross(ao, ray.direction);

          float determinant = -dot(ray.direction, normalVector);
          float invDet = 1 / determinant;

                // Calculate dst to triangle & barycentric coordinates of
        intersection point
          float dst = dot(ao, normalVector) * invDet;
          float u = dot(edgeAC, dao) * invDet;
          float v = -dot(edgeAB, dao) * invDet;
          float w = 1 - u - v;

                // Initialize hit info
          HitInfo hit;
          // for backface culling use: (determinant >= 1E-6)
```

```
            // for no backface culling use: ((abs(determinant) >= 1E-6) -> careful
        ka zbog float precision ti zavrsi ray nutra v objektu i onda je stuck
            hit.didHit = ((determinant >= 1E-6) && (dst >= 0) && (u >= 0) && (v >= 0)
        && (w >= 0));
            hit.position = ray.position + ray.direction * dst;
            hit.distance = dst;
            hit.material = mat;

            float3 normal = normalize(tri.normalA * w + tri.normalB * u +
        tri.normalC * v);
            hit.normal = normal;

            return hit;
}
```