HW1 Simple Ray Tracing Work

Jiayao Zhang
Department of Computer Science
King Abdullah University of Science and Technology
jiayao.zhang@kaust.edu.sa

For further references see Shadertoy Example

1 Requirement 1

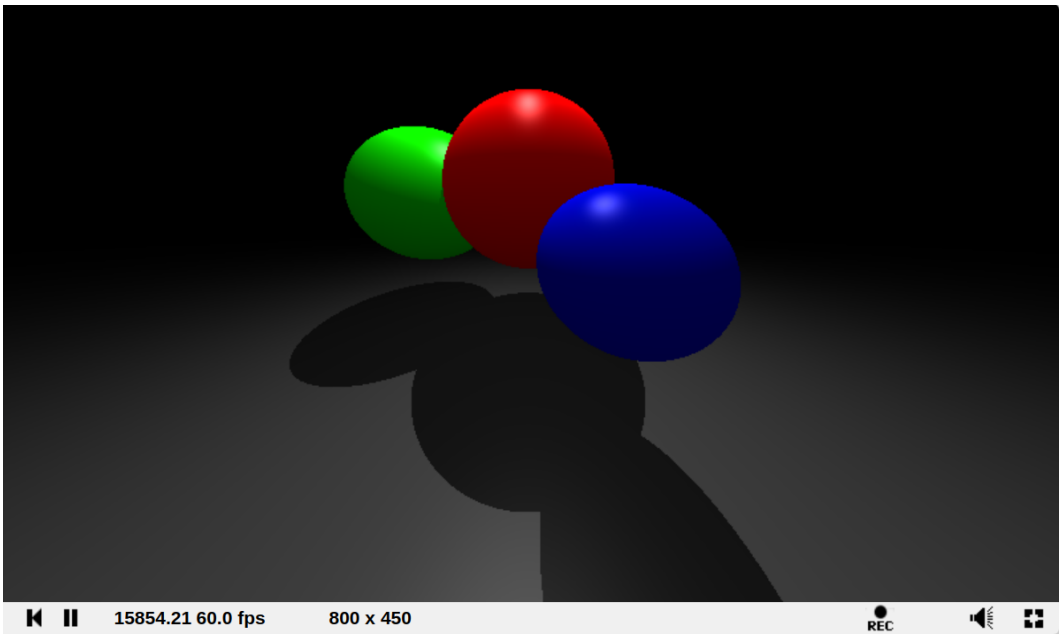

Figure 1: The ray tracing

Implement simple ray tracing algorithm that computes rays for a parallel or perspective projection for a given fixed viewpoint and viewing plane of a resolution at least 640x480. Viewing plane should be defined by equation z = 0 and its extents are [-2,2] in the x direction and [-1.5,1.5] in the y direction with the viewing direction in the positive z direction. The viewpoint should be at the position (0,0,-1).

As we can observe from the Fig.1, the resolution of the viewing plane is 800×450, which is higher than the minimum requirement. And the Viewing Plane as we can see that $x \in [-1.5, 1.5], y \in [-1, 2], z = 0$, and the viewpoint is at (0,0,-1).

2 Requirement 2

Define three geometric objects quadric surfaces, through their analytical formula. Two of them should at least partially be defined within the range x:[-2,2], y:[-1.5,1.5], z:R+. These two should be of color (1,0,0) and (0,1,0). One additional object should be in the same range but with z:R- colored with

Kaust ID 173368.

(0,0,1). A gray (0.5,0.5,0.5) plane should be the last object in the scene and it will be located in y = -1.4.

```
// objects in the scene
int num_objects = 4;

vec3 obj_pos[]   = vec3[](vec3(0, -1.4, 0), vec3(0,1,1), vec3(-1.5, 1.0, 1.5), vec3(0.5,0.3,-0.1));
vec4 obj_prop[]  = vec4[](vec4(0), vec4(1.0), vec4(1), vec4(0.45));
vec3 obj_color[] = vec3[](vec3(0.5), vec3(1,0,0), vec3(0,1,0), vec3(0,0,1) );
int  obj_type[]  =  int[](0, 1, 1, 1); // 2 ellipse ...

vec3 light = vec3(0,5,0);
vec3 light_color = vec3(20);
```

Figure 2: The first vector is the plane and the last three are sphere respectively according to the requirements.

3 Requirement 3

Perform a simple intersection test for all pixels iterating over all objects. For each pixel define the closest point of intersection and calculate the surface normal at this location.

```
vec3 normalSphere(in vec3 pos, in vec4 pr)
{
    return (pos - pr.xyz)/pr.w;
}

float intersectPlane(in vec3 ro, in vec3 rd, vec3 op)
{
    return ( (-ro.y + op.y) / rd.y);
}

vec3 normalPlane(in vec3 pos)
{
    return vec3(0.0, 1.0,  0.0);
}
```

Figure 3: Here we can get the normal vector of the sphere and the plane, also the intersection with Plane.

```
float intersectSphere(in vec3 ro, in vec3 rd, in vec4 pr)
{
    vec3 v_surface_camera = ro - pr.xyz;
    float sphere_rad = pr.w;

    float b = 2.0 * dot(v_surface_camera, rd);
    float c = dot(v_surface_camera, v_surface_camera) - sphere_rad * sphere_rad;
    float h = b*b - 4.0 * c;

    if(h < 0.0)
    {
        return -1.0;    y   t
    }
    return (-b - sqrt(h)) / 2.0; //Again a = 1.
}
```

Figure 4: Here i define the intersection of sphere

4 Requirement 4

Then perform the shading with the light positioned at the location (0,5,0) with light color (1,1,1). Calculate a simple Lambertian Shading Model at each intersection with ka = 0.3, kd=0.7. Keep in mind that each shadow ray (ray from surface point to the light source point) you need to calculate whether it intersects another object or not. If it does, the object is in shadow and only the ambient component will be used in the illumination calculation.

```
// shading parameters
float Ka = 0.3; // ambient
float Kd = 0.7; // diffuse
float Ks = 0.35; // specular
```

Figure 5: Here I illustrate the diffuse coefficient and the ambient coefficient factors and the specular shading.

```
// ray towards objects
for(int i=0; i<num_objects; i++)
{
    float od;
    vec3 op, on;
    // ro = ray origin, rd = ray dir, od = object dist, op = object pos, on = object normal
    if(RayIntersectsObject(i, ro, rd, od, op, on))
    {
        if(intersection_dist < 0.0)
        {
            intersection_dist = od;
        }
        else
        {
            if(od > intersection_dist) continue;   // if the object distance is larger than the intersection
            else intersection_dist = od;           // then the object is in shadow
        }

        // lambert shading
        vec3 L = normalize(light - op);
        float NdotL = dot(on, L);
        vec3 H = normalize( -rd + L);
        float NdotH = pow(dot(on, H), 46.0);
        vec3 I = light_color * 1.0 / pow(length(light - op), 2.0);
        vec3 col_ambient  = Ka * I * obj_color[i];
        vec3 col_diffuse  = Kd * I * obj_color[i] * max(0.0, NdotL);
        vec3 col_specular = Ks * I * vec3(1) * max(0.0, NdotH);
        // ray twds light source
        bool not_in_shadow = true;
        for(int j=0; j<num_objects; j++) // here we use the 3 spheres and one plane.
        {
            if(i == j) continue;

            float nothing_f;
            vec3 nothing_v;
            if(RayIntersectsObject(j, op, normalize(light-op), nothing_f, nothing_v, nothing_v))
            {
                not_in_shadow = false;
                break;  // when the object is in shadow , break.
            }
        }
        col = col_ambient + col_diffuse * float(not_in_shadow) + col_specular;
        // if not in shadow, we calculate the illumination parts
    }

}

fragColor = vec4(col,1.0);
}
```

Figure 6: Here I illustrate the diffuse coefficient and the ambient coefficient factors and the specular shading.

5 Requirement 5

For each calculated pixel value crop the floating point values to the range of [0..255] for each R,G,B channel and save the image in the TARGA file format. (More appreciated of course would be to create an image and show it in the application window.)

https://www.shadertoy.com/view/ws33D2

I have put my image on the shadertoy website, hence we can easily see the image from this website and add more items freely if we like to.

6 Calculated the time it took to perform the raytracing

AS we can see that the time is 60 fps. Next time, I will show this demo using Visual Studio Professional Version with OpenGL. Hopefully I will much more higher fps. Since this demo runs at shadertoy, so it limited at 60fps.