Visual Computing - Point Cloud Rendering

Kranzl, Manuel
ai22m038@technikum-wien.at

June 14, 2023

1 Specification of the task

This paper represents the analysis of the solution of the first topic - Ray Casting in Shader Toy. The main task was the elaboration of a complex shader, which implements Ray Casting of a simple scene as PixelShader. It was recommended to use ShaderToy for this exercise.

2 Requirements for scene and lightning

The requirements for this scene were the following:

- at least one ground level
- at least one box and one sphere
- lighting based on diffuse lighting model
- calculation of shadows using Ray Cast to the light source

3 Solution

3.1 Basics

As recommended, ShaderToy was used for this exercise. The first twelve lectures of the linked tutorial (*https://inspirnathan.com/posts/47-shadertoy-tutorial-part-1/*) explained every detail needed to solve this exercise. The solution is based on the twelfth tutorial in the linked tutorial. The box and the sphere are laying next to each other on the ground level, with the light source circling over them.

To create reflections of light a simple form of Ray Casting was implemented, using Ray Marching as a tool. The Ray Casting part of the code can be seen in a later chapter of this paper. The implementation of Ray Marching and the whole code can be seen in the submission. For the implementation of shadows a simple form of Soft Shadows ways added. This part of the code can also be seen in the submission.

3.2 Ground, box and sphere

As described in the tutorial, it is very simple to write the function for creating boxes or spheres. The following part of the code was used for the creation of the box, with the functions for the floor and sphere being very similar.

```
Surface sdBox( vec3 p, vec3 b, vec3 offset, vec3 col)
{
  p = p - offset;
  vec3 q = abs(p) - b;
  float d = length(max(q,0.0)) + min(max(q.x,max(q.y,q.z)),0.0);
  return Surface(d, col);
}
```

To build the scene itself these functions were loaded one after another using an union-function.

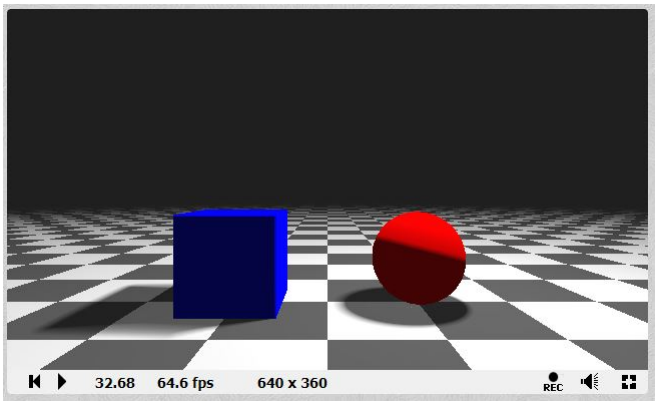
3.3 Creation of main image

After the implementation of a simple RayMarching and a SoftShadow function the main image creation looks like this:

```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
  vec2 uv = (fragCoord-.5*iResolution.xy)/iResolution.y;
  vec3 backgroundColor = vec3(0.01, .01, .01);

  vec3 col = vec3(0);
  vec3 ro = vec3(0, .25, 3); // ray origin that represents camera position
  vec3 rd = normalize(vec3(uv, -1)); // ray direction

  Surface co = rayMarch(ro, rd); // closest object

  if (co.sd > MAX_DIST) {
    col = backgroundColor; // ray didn't hit anything
  } else {
    vec3 p = ro + rd * co.sd; // point discovered from ray marching
    vec3 normal = calcNormal(p);

    vec3 lightPosition = vec3(cos(.75*iTime)*2., 2, sin(.75*iTime)*2.);
    vec3 lightDirection = normalize(lightPosition - p);

    float dif = clamp(dot(normal, lightDirection), 0., 1.) + 0.5; // diffuse reflection

    float softShadow = clamp(softShadow(p, lightDirection, 0.02, 2.5), 0.1, 1.0);

    col = dif * co.col * softShadow;
  }

  col = mix(col, backgroundColor, 1.0 - exp(-0.0002 * co.sd * co.sd * co.sd)); // fog
  col = pow(col, vec3(1.0/2.2)); // Gamma correction
  fragColor = vec4(col, 1.0); // Output to screen
}
```

All the subfunctions can be seen in the main submission file.

4 Results



5 Discussion

5.1 Difference between Ray Casting and Ray Tracing

In Ray Casting a ray gets sent from the camera through every pixel of the screen. When encountering an object the ray gets stopped and the according color will be projected on the screen. There are also ways to include light and shadows, as seen later in this paper.

Ray Tracing uses this concept, but improves it by casting rays in other directions from the interception point (e.g. to the light for shadowing, reflecting the ray for realistic reflections, straight through the object for rendering transparent objects, ...). Ray Tracing therefore is slower than Ray Casting, but leads to more realistic results.

5.2 Lightning model

The used lightning model is a basic diffuse reflection model. For every pixel we trace a ray until Ray Marching detects an intersection point p. We then calculate the surface normal of this point using the calcNormal-function as well as the direction to the light source (which itself is changing its position depending on the time). The dot product between these two vectors indirectly represents the amount of reflection of this point. We also add a factor for shadows, calculated by softShadow(). We calculate the value of shadow added by wandering along the ray pointing from p to the light source. When we hit an object, we now we have to add a shadow. By changing the amount of shadow depending on the distance from p to the object we create the illusion of soft shadows.