Danmarks
Tekniske
Universitet

Planar Reflector

Authors

Jiaqi WU - s222398
Yao Xiao - s222505

December 18, 2022

Contents

1 Introduction

The purpose of this project is to further develop the 3D teapot program in my worksheet 9. My original drawing program was implemented with javascript/HTML and WebGL shaders. WebGL is an API for rendering interactive 2D and 3D graphics to most browsers. Compared with the original project, the following features are added:

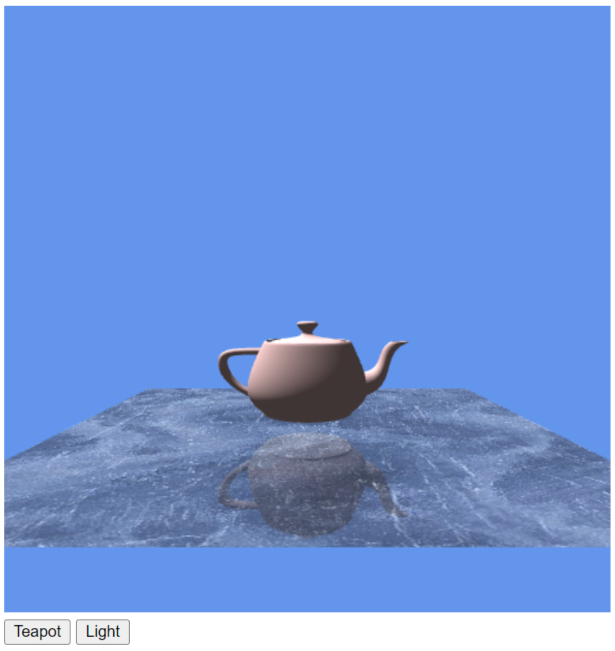
Use a reflection matrix to draw a reflected version of the scene.

Blend reflector and the reflected scene drawn behind it.

Use a stencil buffer to ensure that reflected objects are only drawn where the reflector will be drawn afterwards.

Use oblique near plane clipping to ensure that submerged objects do not result in reflected objects appearing in front of the reflector.

The following figure shows the final effect of our project. Click teapot and light to control the up and down movement of the teapot and the movement of the light.



This report will focus on the methods and math behind program as just described.

2 Methods

Several techniques must be developed in order to draw interactive images on an HTML canvas and have them run immediately in a browser. In this part, I will present an abstracted summary of the various strategies used. Section will provide in-depth explanations of various strategies.

2.1 HTML

HTML (HyperText Markup Language) conceptually explains the structure of a webpage and can embed text, images, sounds, videos, and other things to be presented to the browser. HTML works well with the scripting language javascript. We define shaders in HTML and add buttons to control teapot and light.

2.2 SHADERS

Shaders is a program that runs on a graphics processor unit (GPU), a processor specifically designed to accelerate graphics rendering. GPUs are very good at vector and matrix manipulation, which is an important part of webgl programming, so vertex and fragment shaders are a natural choice for creating such drawing programs to render in the browser. Vertex shaders are responsible for manipulating the properties of vertices. These can be operations such as transforming vertex positions, generating texture coordinates and illuminating vertices. The result of these operations is a set of pixels, called fragments, which can be manipulated by the fragment shader. The fragment shader knows the position of the fragments and can therefore handle depth, colour interpolation etc. between fragments.

2.3 WEBGL

As mentioned earlier, this project uses the WebGL library, which compiles shader instructions into GPU code. webGL defines this method in the javascript file, and the methods of this library are described in detail in the next section.

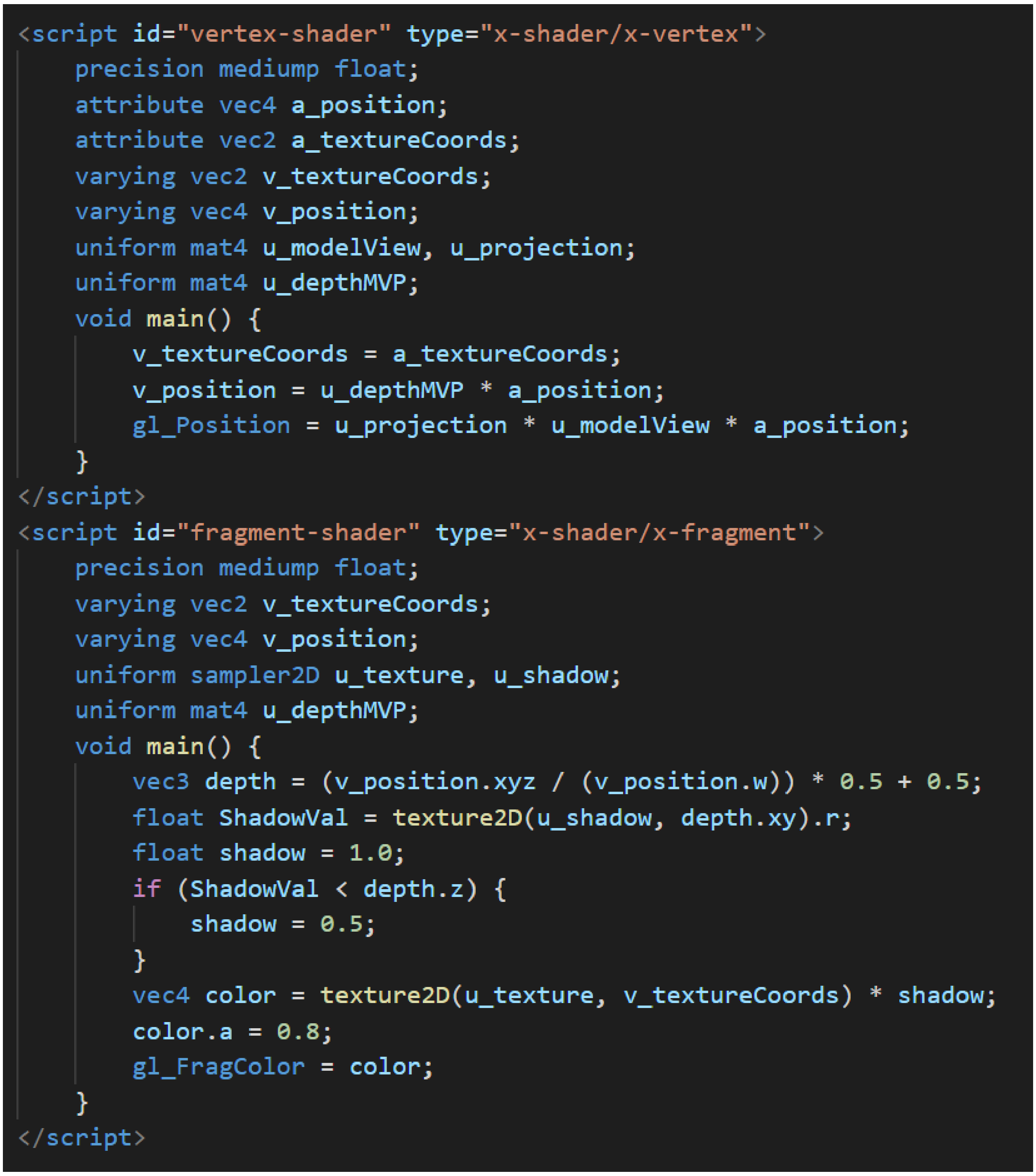
3 Implementation

3.1 HTML

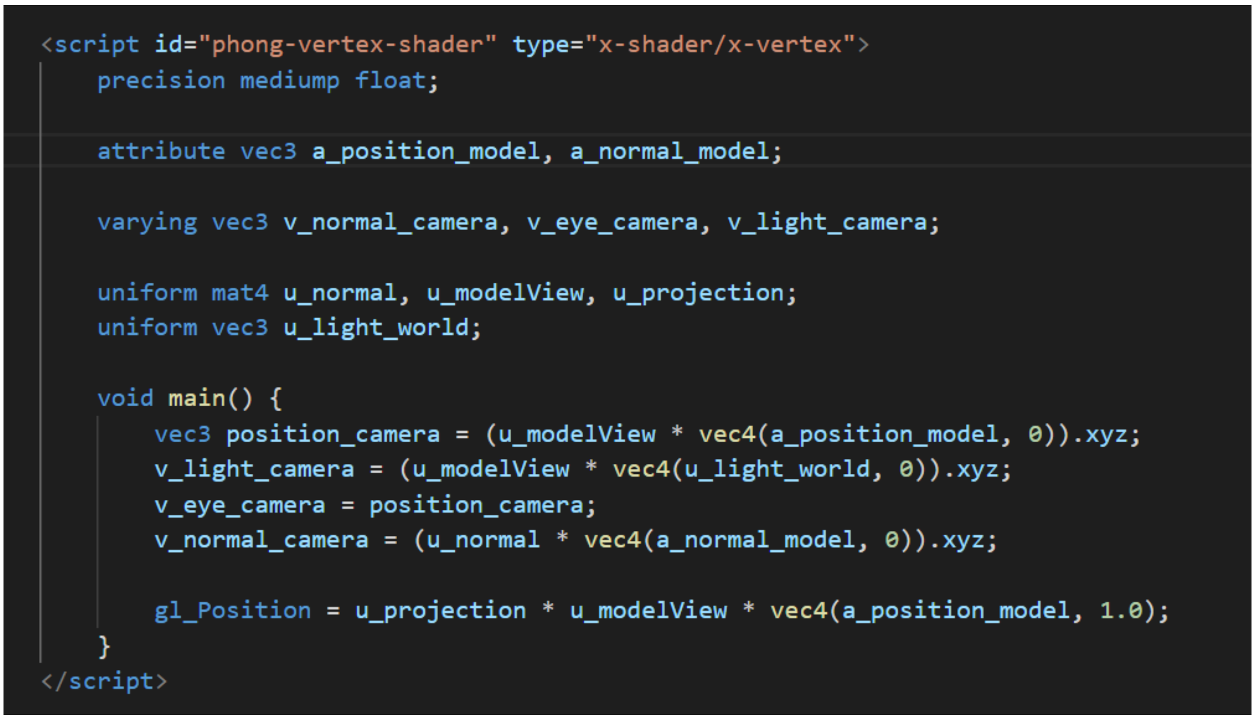
We implemented vertex shader, fragment shader, phong vertex shader, phong fragment shader, and depth vertex shader in HTML. These shaders can also be defined in other files, such as JavaScript. The vertex shader acts on each vertex to generate the final position of each vertex. It will be executed once for each vertex. Once the final position of each vertex is determined, GPU can assemble the set of visible vertices into points, lines and triangles, thus improving the speed of rendering scenes and models.

The fragment shader is used to calculate the color and other attributes of the fragment. The simplest clip shader only outputs color values; Complex clip shaders can have multiple inputs and outputs. Clip shaders can not only output the same color forever, but also consider lighting, bump mapping, shadow generation and highlight generation, as well as translucency and other effects. Clip shaders can also modify the depth of clips to output multiple colors for multiple render targets.

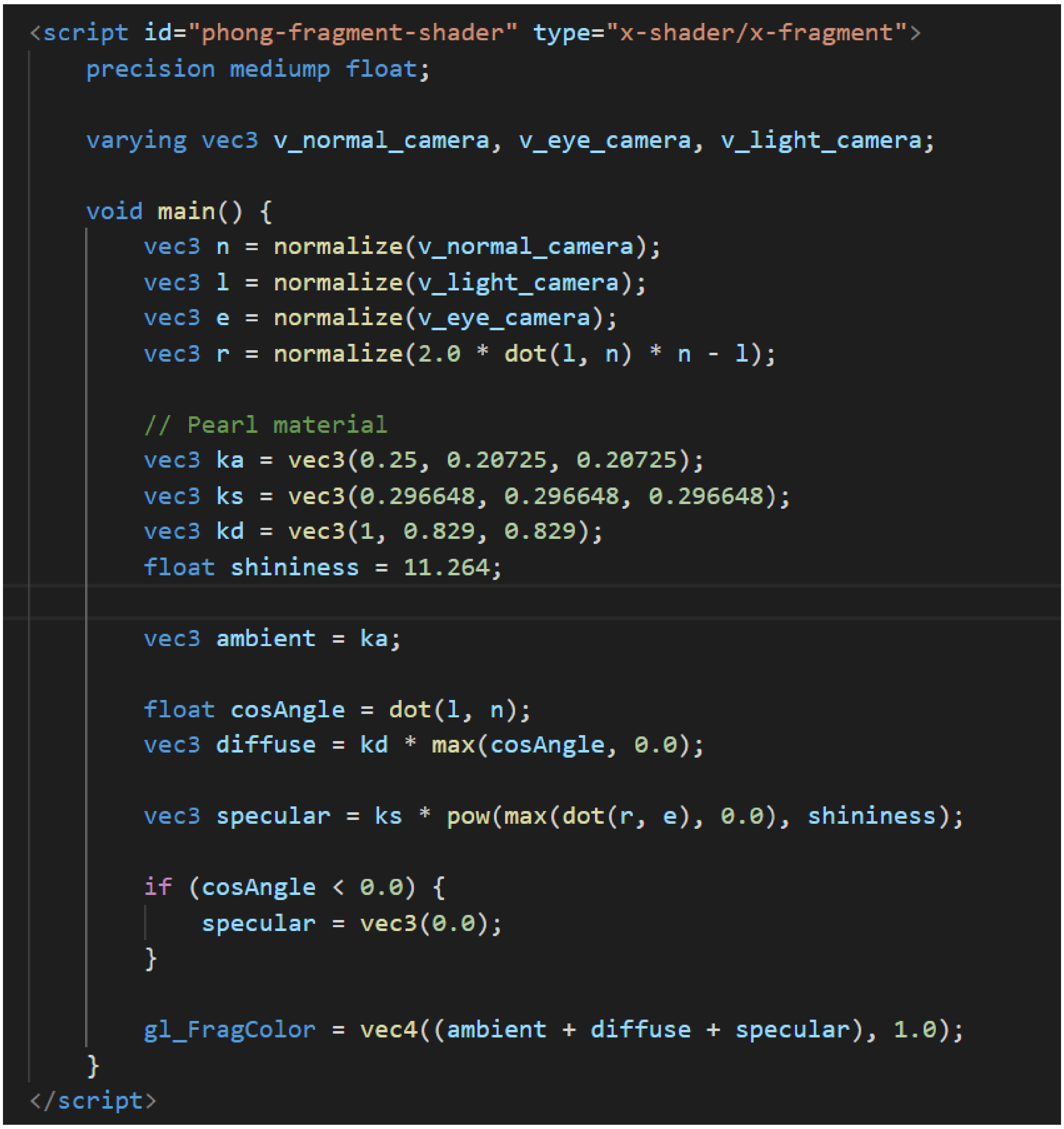
We use phong lighting to achieve light effects, and depth to achieve depth effects.



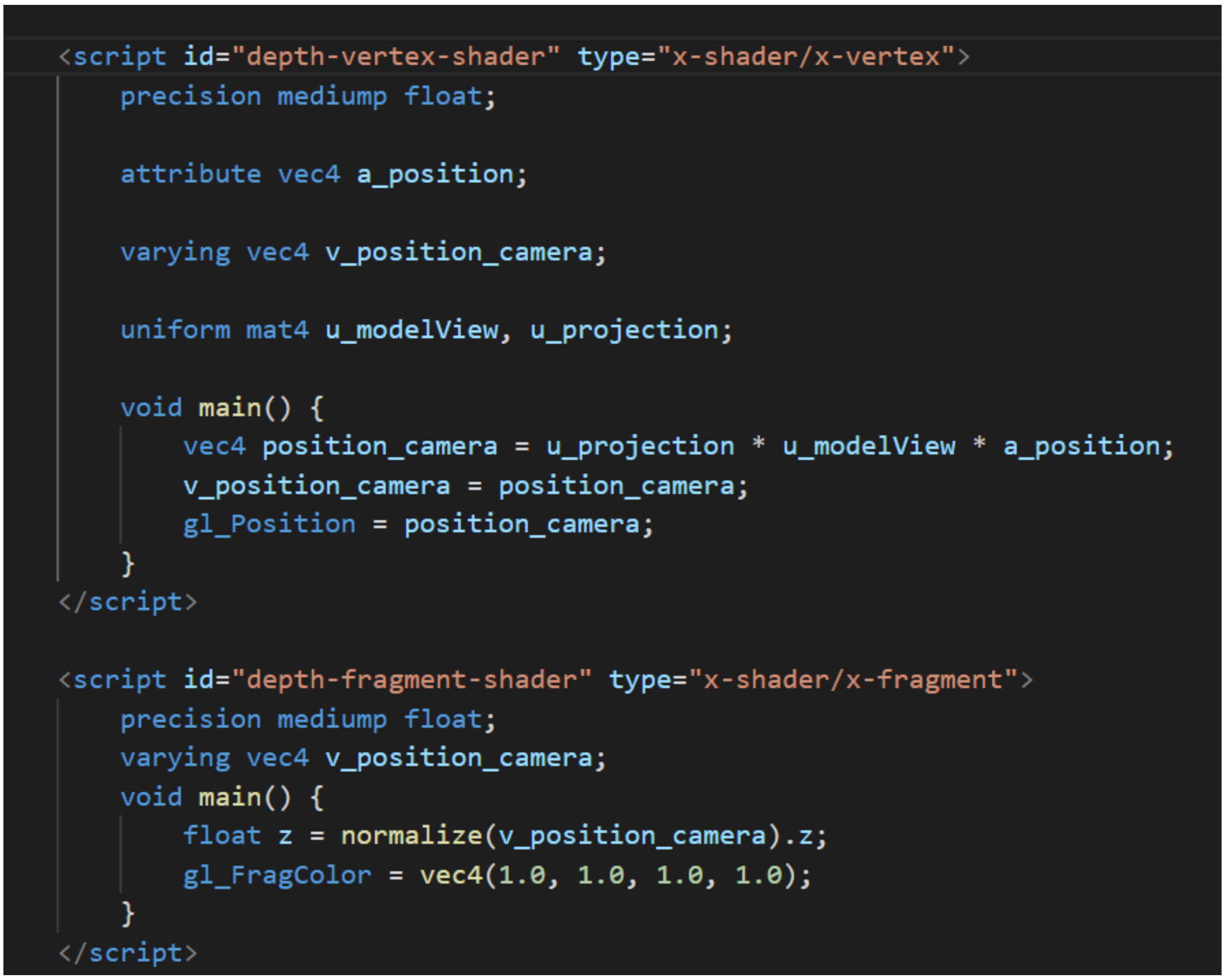
```
<script id="vertex-shader" type="x-shader/x-vertex">
    precision mediump float;
    attribute vec4 a_position;
    attribute vec2 a_textureCoords;
    varying vec2 v_textureCoords;
    varying vec4 v_position;
    uniform mat4 u_modelView, u_projection;
    uniform mat4 u_depthMVP;
    void main() {
        v_textureCoords = a_textureCoords;
        v_position = u_depthMVP * a_position;
        gl_Position = u_projection * u_modelView * a_position;
    }
</script>
<script id="fragment-shader" type="x-shader/x-fragment">
    precision mediump float;
    varying vec2 v_textureCoords;
    varying vec4 v_position;
    uniform sampler2D u_texture, u_shadow;
    uniform mat4 u_depthMVP;
    void main() {
        vec3 depth = (v_position.xyz / (v_position.w)) * 0.5 + 0.5;
        float ShadowVal = texture2D(u_shadow, depth.xy).r;
        float shadow = 1.0;
        if (ShadowVal < depth.z) {
            shadow = 0.5;
        }
        vec4 color = texture2D(u_texture, v_textureCoords) * shadow;
        color.a = 0.8;
        gl_FragColor = color;
    }
</script>
```



```
<script id="phong-vertex-shader" type="x-shader/x-vertex">
    precision mediump float;

    attribute vec3 a_position_model, a_normal_model;

    varying vec3 v_normal_camera, v_eye_camera, v_light_camera;

    uniform mat4 u_normal, u_modelView, u_projection;
    uniform vec3 u_light_world;

    void main() {
        vec3 position_camera = (u_modelView * vec4(a_position_model, 0)).xyz;
        v_light_camera = (u_modelView * vec4(u_light_world, 0)).xyz;
        v_eye_camera = position_camera;
        v_normal_camera = (u_normal * vec4(a_normal_model, 0)).xyz;

        gl_Position = u_projection * u_modelView * vec4(a_position_model, 1.0);
    }
</script>
```



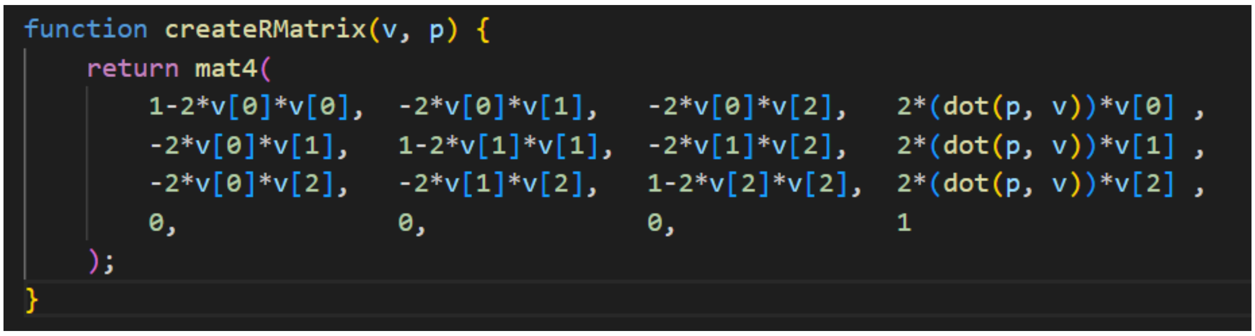
```
<script id="phong-fragment-shader" type="x-shader/x-fragment">
    precision mediump float;

    varying vec3 v_normal_camera, v_eye_camera, v_light_camera;

    void main() {
        vec3 n = normalize(v_normal_camera);
        vec3 l = normalize(v_light_camera);
        vec3 e = normalize(v_eye_camera);
        vec3 r = normalize(2.0 * dot(l, n) * n - l);

        // Pearl material
        vec3 ka = vec3(0.25, 0.20725, 0.20725);
        vec3 ks = vec3(0.296648, 0.296648, 0.296648);
        vec3 kd = vec3(1, 0.829, 0.829);
        float shininess = 11.264;

        vec3 ambient = ka;

        float cosAngle = dot(l, n);
        vec3 diffuse = kd * max(cosAngle, 0.0);

        vec3 specular = ks * pow(max(dot(r, e), 0.0), shininess);

        if (cosAngle < 0.0) {
            specular = vec3(0.0);
        }

        gl_FragColor = vec4((ambient + diffuse + specular), 1.0);
    }
</script>
```



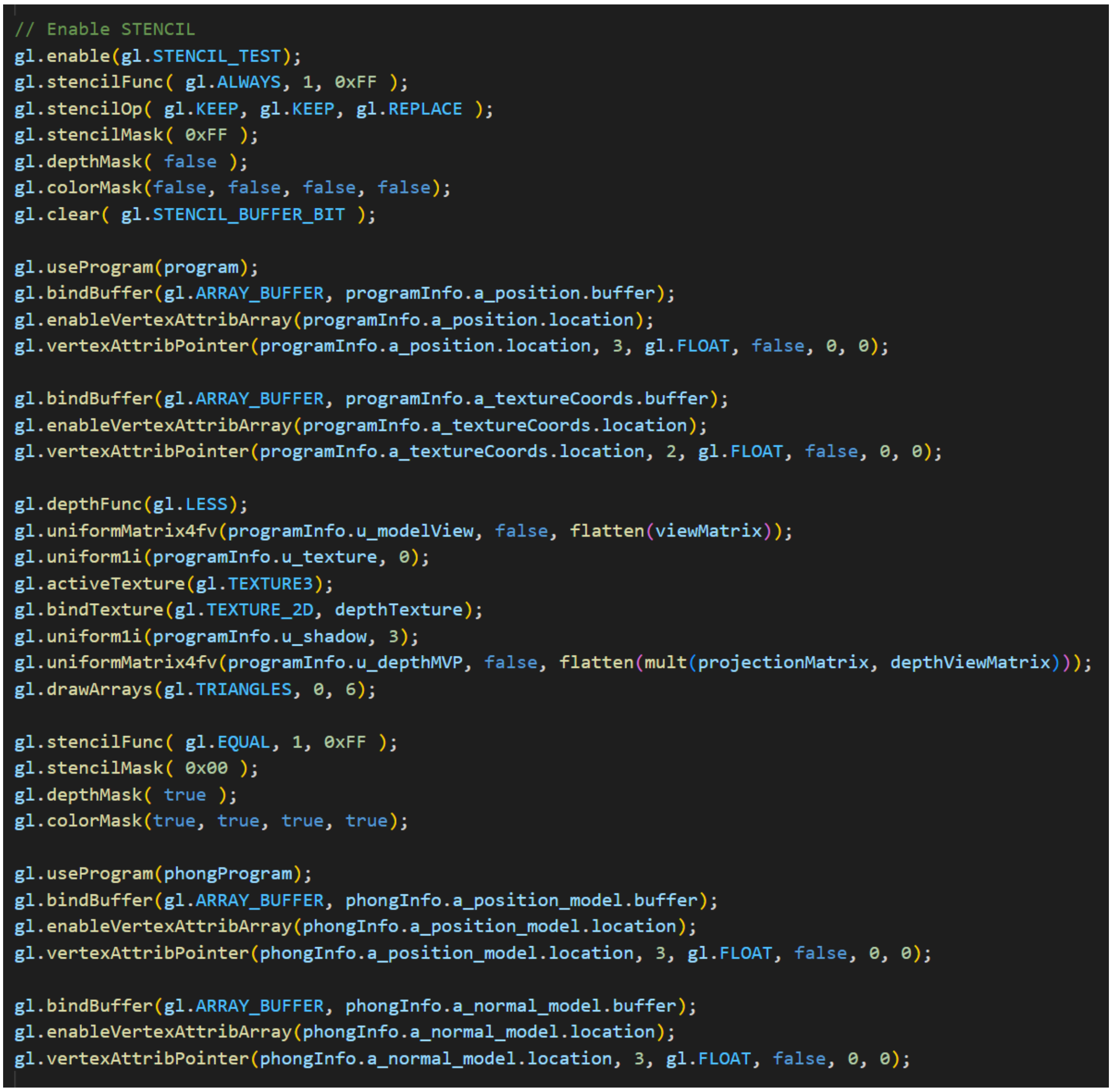
```
<script id="depth-vertex-shader" type="x-shader/x-vertex">
    precision mediump float;

    attribute vec4 a_position;

    varying vec4 v_position_camera;

    uniform mat4 u_modelView, u_projection;

    void main() {
        vec4 position_camera = u_projection * u_modelView * a_position;
        v_position_camera = position_camera;
        gl_Position = position_camera;
    }
</script>

<script id="depth-fragment-shader" type="x-shader/x-fragment">
    precision mediump float;
    varying vec4 v_position_camera;
    void main() {
        float z = normalize(v_position_camera).z;
        gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
    }
</script>
```

3.2 JavaScript

In JavaScript, we realized the drawing and lighting effect of the teapot, which is consistent with the requirements in worksheet 9. The main new content of the project is reflection. We have established a reflection matrix to realize the reflection function. We have established a reflection matrix by referring to the method in Reflections (by McReynolds et al.). However, we can not only realize reflection in the specified plane, so we use STENCIL_ TEST to ensure that the reflection only appears in the plane.

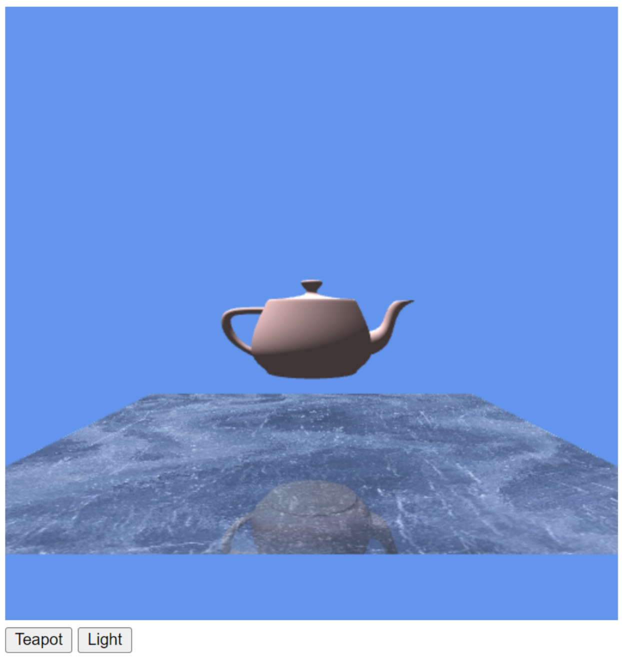


```
function createRMatrix(v, p) {
    return mat4(
        1-2*v[0]*v[0],  -2*v[0]*v[1],   -2*v[0]*v[2],   2*(dot(p, v))*v[0] ,
        -2*v[0]*v[1],   1-2*v[1]*v[1],  -2*v[1]*v[2],   2*(dot(p, v))*v[1] ,
        -2*v[0]*v[2],   -2*v[1]*v[2],   1-2*v[2]*v[2],  2*(dot(p, v))*v[2] ,
        0,              0,              0,              1
    );
}
```



```
// Enable STENCIL
gl.enable(gl.STENCIL_TEST);
gl.stencilFunc( gl.ALWAYS, 1, 0xFF );
gl.stencilOp( gl.KEEP, gl.KEEP, gl.REPLACE );
gl.stencilMask( 0xFF );
gl.depthMask( false );
gl.colorMask(false, false, false, false);
gl.clear( gl.STENCIL_BUFFER_BIT );

gl.useProgram(program);
gl.bindBuffer(gl.ARRAY_BUFFER, programInfo.a_position.buffer);
gl.enableVertexAttribArray(programInfo.a_position.location);
gl.vertexAttribPointer(programInfo.a_position.location, 3, gl.FLOAT, false, 0, 0);

gl.bindBuffer(gl.ARRAY_BUFFER, programInfo.a_textureCoords.buffer);
gl.enableVertexAttribArray(programInfo.a_textureCoords.location);
gl.vertexAttribPointer(programInfo.a_textureCoords.location, 2, gl.FLOAT, false, 0, 0);

gl.depthFunc(gl.LESS);
gl.uniformMatrix4fv(programInfo.u_modelView, false, flatten(viewMatrix));
gl.uniform1i(programInfo.u_texture, 0);
gl.activeTexture(gl.TEXTURE3);
gl.bindTexture(gl.TEXTURE_2D, depthTexture);
gl.uniform1i(programInfo.u_shadow, 3);
gl.uniformMatrix4fv(programInfo.u_depthMVP, false, flatten(mult(projectionMatrix, depthViewMatrix)));
gl.drawArrays(gl.TRIANGLES, 0, 6);

gl.stencilFunc( gl.EQUAL, 1, 0xFF );
gl.stencilMask( 0x00 );
gl.depthMask( true );
gl.colorMask(true, true, true, true);

gl.useProgram(phongProgram);
gl.bindBuffer(gl.ARRAY_BUFFER, phongInfo.a_position_model.buffer);
gl.enableVertexAttribArray(phongInfo.a_position_model.location);
gl.vertexAttribPointer(phongInfo.a_position_model.location, 3, gl.FLOAT, false, 0, 0);

gl.bindBuffer(gl.ARRAY_BUFFER, phongInfo.a_normal_model.buffer);
gl.enableVertexAttribArray(phongInfo.a_normal_model.location);
gl.vertexAttribPointer(phongInfo.a_normal_model.location, 3, gl.FLOAT, false, 0, 0);
```

4 Result



As you can see, our program can display the reflection of the teapot and ensure that the reflection does not appear outside the specified plane, meeting the requirements of the task.

5 Discussion

In general, we have successfully established a reflection matrix to realize the reflection function and use STENCIL_TEST to ensure that the reflection only appears in the plane. We didn't encounter too much trouble in the development process. Everything was very smooth when establishing the reflection function, but in the STENCIL_TEST part, we had to go online to consult relevant materials to learn how to use it. Fortunately, the problem was solved quickly and did not affect our project progress. For possible future work, we can try to complete some other difficult tasks, such as implementing bezier curves in worksheet2.

6 Link to online project

You can see it in our student page at: `www.student.dtu.dk/~s222398`

7 CREDIT

Jiaqi work for the STENCIL_ TEST and the Introduction methods parts in report.Yao work for the reflection matrix and the rest part in the report.