BBM414 Computer Graphics Lab.
Assignment 5 - Report

Berk Bubuş
21945939
Department of Computer Engineering
Hacettepe University
Ankara, Turkey
b21945939@cs.hacettepe.edu.tr

Overview

In this assignment, we have 2 parts. At the first part, we have integrated light and texture to our code. At second, we made skybox with lots of cubes which has the seagull and sea textures, and camera controls.

1 Part 1

In Part 1, I have modified these variables to get the correct color.

```
var lightPosition = vec4(1.0, 1.0, 1.0, 0.0 );
var lightAmbient = vec4(0.2, 0.2, 0.2, 1.0 );
var lightDiffuse = vec4( 1.0, 1.0, 1.0, 1.0 );
var lightSpecular = vec4( 1.0, 1.0, 1.0, 1.0 );

var materialAmbient = vec4( 1.0, 1.0, 1.0, 1.0 );
var materialDiffuse = vec4( 1.0, 1.0, 1.0, 1.0);
var materialSpecular = vec4( 1.0, 1.0, 1.0, 1.0 );
var materialShininess = 100.0;
```

And I added these lines to use buttons to change light position etc.

```
    document.getElementById("ButtonX").onclick = function(){axis = xAxis;};
    document.getElementById("ButtonY").onclick = function(){axis = yAxis;};
    document.getElementById("ButtonZ").onclick = function(){axis = zAxis;};
    document.getElementById("ButtonT").onclick = function(){flag = !flag;};
  document.getElementById("LightX+").onclick = function(){lightPosition[0]+=0.1;}
  document.getElementById("LightX-").onclick = function(){lightPosition[0]-=0.1;}
  document.getElementById("LightY+").onclick = function(){lightPosition[1]+=0.1;}
  document.getElementById("LightY-").onclick = function(){lightPosition[1]-=0.1;}
  document.getElementById("LightZ+").onclick = function(){lightPosition[2]+=0.1;}
  document.getElementById("LightZ-").onclick = function(){lightPosition[2]-=0.1;}
```

And I added these lines for textures.

```
      gl.vertexAttribPointer(
        texcoordAttributeLocation, size, type, normalize, stride, offset);

      var texture = gl.createTexture();
```

```
        // use texture unit 0
        gl.activeTexture(gl.TEXTURE0 + 0);

        // bind to the TEXTURE_2D bind point of texture unit 0
        gl.bindTexture(gl.TEXTURE_2D, texture);
        gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 1, 1, 0, gl.RGBA, gl.UNSIGNED_BYTE,
              new Uint8Array([0, 0, 255, 255]));
        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_LINEAR);

        // Asynchronously load an image
        var image = new Image();
        image.src = "images/snowflake.png";
        image.addEventListener('load', function() {
        // Now that the image has loaded make copy it to the texture.
        gl.bindTexture(gl.TEXTURE_2D, texture);
        gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
        gl.generateMipmap(gl.TEXTURE_2D);
        });
```

2 Part 2 - HTML

I added a little part to HTML to add nocursor class to the style.

```
<html>
    <head>
        <title>Assignment 4 - Berk Bubu </title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <script src="js/initialize.js" defer></script>
        <script src="js/shaders.js" defer></script>
        <script src="js/app.js" defer></script>
        <style>
            .nocursor {
                position: relative;
                overflow: hidden;
                cursor: none;
            }
        </style>
    </head>
    <body>
        <canvas class="nocursor" id="glCanvas" width="512" height="512">
            Oops ... your browser doesn't support the HTML5 canvas element
␣␣␣␣␣␣␣␣␣␣␣␣</canvas>
␣␣␣␣</div>
␣␣␣␣</body>
</html>
```

And I got these lines on fragment shader to create textures.

```
precision mediump float;

in vec4 fColor;

in vec2 v_texcoord;
uniform sampler2D u_texture;
out vec4 oColor;

void
main()
{
    oColor = fColor * texture(u_texture, v_texcoord);
}
```

3 Part 2

In this part, I used my assignment 4 files and added codes to them. And my cursor options same as in assignment 4. If we click to the canvas, our cursor will lock. But even if we don't click, we don't see our cursor in the canvas boundaries.

4 Part 2 - shaders.js

In shaders.js, we have lots of shader sources. 3 vertex and 3 fragment shaders are what we got. I have one pair for skybox, one pair for per-vertex lighting cubes and floor, and one pair for per-pixel one. per-vertex and per-pixel ones are similar. Skybox one is more basic. I used WebGL2 Fundamentals a lot to code my spotlight and texture parts.

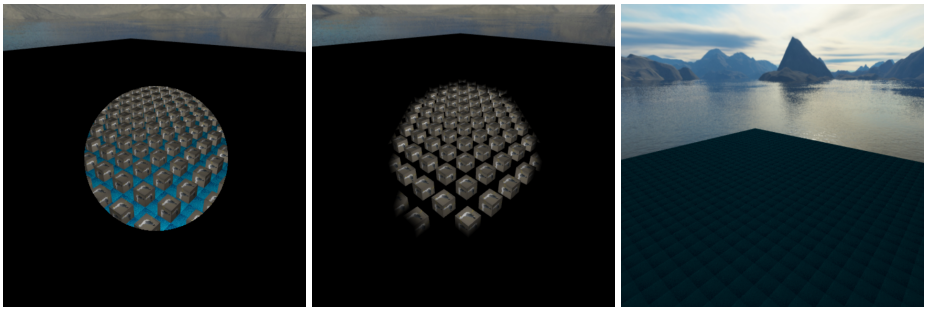
```
const vsSource = `#version 300 es
    #define PI 3.1415926538

    in vec4 aPosition;
    in vec2 aTexcoord;
    in vec3 aNormal;

    uniform mat4 projection;
    uniform mat4 view;
    uniform mat4 model;
    out vec2 vTexcoord;
    out float isSea;

    out vec3 vNormal;
    uniform vec3 uLightWorldPosition;
    uniform vec3 viewWorldPosition;
    uniform mat4 uWorldInverseTranspose;
    out vec3 vSurfaceToLight;
    out vec3 vSurfaceToView;

    void main()
    {
        vNormal = - mat3(uWorldInverseTranspose) * aNormal;
        vec3 surfaceWorldPosition = (aPosition).xyz;
        vSurfaceToLight = uLightWorldPosition - surfaceWorldPosition;
        vSurfaceToView = viewWorldPosition - surfaceWorldPosition;
        gl_Position = projection * view * model * aPosition;
        if(aPosition.y <= 0.0){
            isSea = 1.0;
        }
        else{
            isSea = 0.0;
        }

        vTexcoord = aTexcoord;
    }
`;

const fsSource =  `#version 300 es
    precision highp float;
    in vec2 vTexcoord;
    in float isSea;

    uniform sampler2D seagullTexture;
    uniform sampler2D seaTexture;
    out vec4 oColor;

    in vec3 vNormal;
    in vec3 vSurfaceToLight;
    in vec3 vSurfaceToView;
```

```
    uniform vec4 uColor;
    uniform float uShininess;
    uniform vec3 uLightDirection;
    uniform float uLimit;
    void main(){
        vec3 normal = normalize(vNormal);
        vec3 surfaceToLightDirection = normalize(vSurfaceToLight);
        vec3 surfaceToViewDirection = normalize(vSurfaceToView);
        vec3 halfVector = normalize(surfaceToLightDirection + surfaceToViewDirection);
        float dotFromDirection = dot(surfaceToLightDirection, -uLightDirection);
        float inLight = step(uLimit, dotFromDirection);
        float light = dot(normal, surfaceToLightDirection) * inLight;
        float specular = inLight * pow(dot(normal, halfVector), uShininess);
        if(isSea==1.0){
            oColor = uColor * texture(seaTexture, vTexcoord);
        }
        else{
            oColor = uColor * texture(seagullTexture, vTexcoord);
        }

        oColor.rgb *= light;
        oColor.rgb += specular;

    }
`;

const vsSource1 = `#version 300 es
    #define PI 3.1415926538

    in vec4 aPosition;
    in vec2 aTexcoord;
    in vec3 aNormal;
    uniform float uShininess;
    uniform vec3 uLightDirection;
    uniform float uLimit;
    uniform mat4 projection;
    uniform mat4 view;
    uniform mat4 model;
    out vec2 vTexcoord;
    out float isSea;
    out vec4 vColor;
    out float light;
    out float specular;
    uniform vec3 uLightWorldPosition;
    uniform vec3 viewWorldPosition;
    uniform mat4 uWorldInverseTranspose;

    void main()
    {
        vec3 vNormal = - mat3(uWorldInverseTranspose) * aNormal;
        vec3 surfaceWorldPosition = (aPosition).xyz;
        vec3 vSurfaceToLight = uLightWorldPosition - surfaceWorldPosition;
        vec3 vSurfaceToView = viewWorldPosition - surfaceWorldPosition;
        gl_Position = projection * view * model * aPosition;
        if(aPosition.y <= 0.0){
            isSea = 1.0;
        }
        else{
            isSea = 0.0;
        }

        vTexcoord = aTexcoord;
        vec3 normal = normalize(vNormal);
        vec3 surfaceToLightDirection = normalize(vSurfaceToLight);
        vec3 surfaceToViewDirection = normalize(vSurfaceToView);
```

```
        vec3 halfVector = normalize(surfaceToLightDirection + surfaceToViewDirection);
        float dotFromDirection = dot(surfaceToLightDirection, -uLightDirection);
        float inLight = step(uLimit, dotFromDirection);
        light = dot(normal, surfaceToLightDirection) * inLight;
        specular = inLight * pow(dot(normal, halfVector), uShininess);
    }
`;

const fsSource1 =  `#version 300 es
    precision highp float;
    in vec2 vTexcoord;
    in float isSea;
    uniform vec4 uColor;
    uniform sampler2D seagullTexture;
    uniform sampler2D seaTexture;
    out vec4 oColor;
    in float light;
    in float specular;
    in vec4 vColor;

    void main(){

        if(isSea==1.0){
            oColor = uColor * texture(seaTexture, vTexcoord);
        }
        else{
            oColor = uColor * texture(seagullTexture, vTexcoord);
        }

        oColor.rgb *= light;
        oColor.rgb += specular;

    }
`;

const vsSourceSkybox = `#version 300 es
    in vec4 a_position;
    out vec4 v_position;
    void main() {
        v_position = a_position;
        gl_Position = a_position;
        gl_Position.z = 1.0;
    }
`;

const fsSourceSkybox = `#version 300 es
    precision highp float;

    uniform samplerCube u_skybox;
    uniform mat4 u_viewDirectionProjectionInverse;

    in vec4 v_position;

    out vec4 outColor;

    void main() {
        vec4 t = u_viewDirectionProjectionInverse * v_position;
        outColor = texture(u_skybox, normalize(t.xyz / t.w));
    }
`;
```

5 Part 2 - app.js

First of all, I have managed all the parts correct but In my per-vertex part. I have a tiny problem about creating the floor. I created the floor with 8 edges. So it doesn't look nice with lighting because I don't have enough vertex for it. So there is a difference in the video and my code. First one is per-pixel, second one is per-vertex, and as you can see, if we point our camera to the edge of the floor which is a vertex point, there is light but low (because it is just one point of three). This lighting difference is about the floor creation.



And I have 2 init functions with 2 draw functions. And one of my init function which is named initOther have parameters to switch pervertex and perpixel shaders. After pushing "p", our initOther function is called and the new initialization activated. And there are special vertex arrays to bind at the right time.

After all, I have lots of functions to help me to create the code. One of them is rectanglePrism. This helps me to create lots of cubes with given parameters. And the floor of course, with given parameters.

```
function rectanglePrism(a,b,c, rectangleCount,betweenX , starterX,starterY,starterZ){
    let positions = [];
    let indices = [];
    let surfaces = [];
    indices.push([starterX,starterY,starterZ],[starterX+a,starterY,starterZ],[starterX+a,s
        [starterX,starterY+b,starterZ],[starterX,starterY,starterZ+c],[starterX+a,starterY
        [starterX+a,starterY+b,starterZ+c],[starterX,starterY+b,starterZ+c]);
    surfaces.push([[indices[0][0],indices[0][1],indices[0][2]],[indices[1][0],indices[1][1
        [indices[2][0],indices[2][1],indices[2][2]],[indices[2][0],indices[2][1],indices[2
        [indices[3][0],indices[3][1],indices[3][2]],[indices[0][0],indices[0][1],indices[0
    surfaces.push([[indices[0][0],indices[0][1],indices[0][2]],[indices[1][0],indices[1][1
        [indices[5][0],indices[5][1],indices[5][2]],[indices[5][0],indices[5][1],indices[5
        [indices[4][0],indices[4][1],indices[4][2]],[indices[0][0],indices[0][1],indices[0
    surfaces.push([[indices[0][0],indices[0][1],indices[0][2]],[indices[3][0],indices[3][1
        [indices[7][0],indices[7][1],indices[7][2]],[indices[7][0],indices[7][1],indices[7
        [indices[4][0],indices[4][1],indices[4][2]],[indices[0][0],indices[0][1],indices[0
    surfaces.push([[indices[3][0],indices[3][1],indices[3][2]],[indices[2][0],indices[2][1
        [indices[6][0],indices[6][1],indices[6][2]],[indices[6][0],indices[6][1],indices[6
        [indices[7][0],indices[7][1],indices[7][2]],[indices[3][0],indices[3][1],indices[3
    surfaces.push([[indices[1][0],indices[1][1],indices[1][2]],[indices[5][0],indices[5][1
        [indices[6][0],indices[6][1],indices[6][2]],[indices[6][0],indices[6][1],indices[6
        [indices[2][0],indices[2][1],indices[2][2]],[indices[1][0],indices[1][1],indices[1
    surfaces.push([[indices[7][0],indices[7][1],indices[7][2]],[indices[6][0],indices[6][1
        [indices[5][0],indices[5][1],indices[5][2]],[indices[5][0],indices[5][1],indices[5
        [indices[4][0],indices[4][1],indices[4][2]],[indices[7][0],indices[7][1],indices[7
    let adderX = 0;
    let adderZ = 0;
    let toRectangleCount = 0;
    let total = rectangleCount*rectangleCount;
    while(toRectangleCount < total){
        for(let j = 0; j < 6; j++){
```

```
            for(let k = 0; k < 6; k++){
                positions.push(surfaces[j][k][0]+adderX);
                positions.push(surfaces[j][k][1]);
                positions.push(surfaces[j][k][2]+adderZ);
            }
        }
        adderX+=betweenX;
        toRectangleCount++;
        if(toRectangleCount%rectangleCount===0){
            adderX=0;
            adderZ+=betweenX;
        }
    }
    return positions;
}
```

And I have setTexcoords and getNormals functions to calculate these variables. And I have lots of mathematical function.

```
function getNormals(positions) {
    const normals = [];

    for (let i = 0; i < positions.length; i += 9) {

      const v1 = [positions[i], positions[i + 1], positions[i + 2]];
      const v2 = [positions[i + 3], positions[i + 4], positions[i + 5]];
      const v3 = [positions[i + 6], positions[i + 7], positions[i + 8]];


      const edge1 = [
        v2[0] - v1[0],
        v2[1] - v1[1],
        v2[2] - v1[2],
      ];
      const edge2 = [
        v3[0] - v1[0],
        v3[1] - v1[1],
        v3[2] - v1[2],
      ];


      const normal = [
        edge1[1] * edge2[2] - edge1[2] * edge2[1],
        edge1[2] * edge2[0] - edge1[0] * edge2[2],
        edge1[0] * edge2[1] - edge1[1] * edge2[0],
      ];


      const length = Math.sqrt(normal[0] * normal[0] + normal[1] * normal[1] + normal[2] *
      normal[0] /= length;
      normal[1] /= length;
      normal[2] /= length;


      normals.push(normal[0], normal[1], normal[2]);
      normals.push(normal[0], normal[1], normal[2]);
      normals.push(normal[0], normal[1], normal[2]);
    }

    return normals;
  }

function setTexcoords(gl) {
    let temp = rectangleCount*rectangleCount;
```

```
    const array = new Float32Array((temp+1)*6*12);

    for(let i = 0; i < (temp)*6*12; i+=12){
        array[i] = 0.0;
        array[i+1] = 0.0;
        array[i+2] = 1.0;
        array[i+3] = 0.0;
        array[i+4] = 1.0;
        array[i+5] = 1.0;
        array[i+6] = 1.0;
        array[i+7] = 1.0;
        array[i+8] = 0.0;
        array[i+9] = 1.0;
        array[i+10] = 0.0;
        array[i+11] = 0.0;
    }

    for(let i = (temp)*6*12; i < (temp+1)*6*12; i+=12){
        array[i] = 0.0;
        array[i+1] = 0.0;
        array[i+2] = temp/5;
        array[i+3] = 0.0;
        array[i+4] = temp/5;
        array[i+5] = temp/5;
        array[i+6] = temp/5;
        array[i+7] = temp/5;
        array[i+8] = 0.0;
        array[i+9] = temp/5;
        array[i+10] = 0.0;
        array[i+11] = 0.0;
    }

    gl.bufferData( gl.ARRAY_BUFFER,array,gl.STATIC_DRAW);
}
```