ANKARA UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

Computer Graphics COM3037-B
2021-2022 Fall
Homework 2 Assigment Report

Elif Çakaloğlu
18290086

The Task is designing a simple 3D geometric model which is can switch between two projections and also able to rotate in all three axes by using WebGL, HTML and Javascript.

How its work;

1-When the program is run for the first time, model's wireframe version is displayed.
2-You can rotate the model with WASD and right and left arrow keys.
3-You can access the mesh view of the model by using wires button.
4-For the illuminated version of the model, the lighting button is used.
5-You can see the textured version of the model with lighting and without lighting view. For this, texture without lighting and texture with lighting buttons are used.

Functionality	How it's done	Type of input device
Rotating right	C key	Keyboard(D and right arrow)
Rotating left	C key	Keyboard(A and left arrow)
Rotating up	C key	Keyboard(W)
Rotating down	C key	Keyboard(S)
Change view with wires	Button	Mouse
Change view with texture(lighting)	Button	Mouse
Change view with texture(non-lighting)	Button	Mouse
Lighting	Button	Mouse

First webgl was initialized.

```
window.onload = function init() {
    const canvas = document.querySelector("#glcanvas");
    gl = WebGLUtils.setupWebGL(canvas);
    if (!gl) {
        alert("Unable to initialize WebGL. Your browser or machine may not support it.");
        return;
    }

    var program = initShaders(gl, "vertex-shader", "fragment-shader");
    gl.useProgram(program);

    gl.enable(gl.DEPTH_TEST);
```

The getElementById() method returns the element that has the ID attribute with the specified value. And I use addEventListener for attaching a click event to the document.
I used the "mode" variable to specify the buttons. E.g: mode=1 used for wires button, mode=2 for lighting button,..

```
document.getElementById("wiresBtn").addEventListener("click", function () {
    mode = 1;
    gl.uniform1i(modeLoc, mode);
    render();
});
document.getElementById("lightingBtn").addEventListener("click", function () {
    mode = 2;
    gl.uniform1i(modeLoc, mode);
    render();
});
document.getElementById("texturesNoLightBtn").addEventListener("click", function () {
    mode = 3;
    gl.uniform1i(modeLoc, mode);
    render();
});
document.getElementById("texturesLightBtn").addEventListener("click", function () {
    mode = 4;
    gl.uniform1i(modeLoc, mode);
    render();
});
```

The following codes in the above have been written to rotate the model with WASD and arrow keys.
To find the codes for the key combinations I used the site https://keycode.info. When we looked a little deeper into the code, mdelViewMatrix used for passing to the shader and calculate the position of the object.
The commands gl.uniformMatrix4fv and gl.uniformMatrix3fv are used to modify a matrix or an array of matrices. The numbers in the command name are interpreted as the dimensionality of the matrix.The number 4 indicates 4x4 matrix and the 3 indicates 3x3 matrix.

```
switch (event.keyCode) {
    case 83:  //KEY S
        modelViewMatrix = mult(translate(-midX, -midY, -midZ), modelViewMatrix);
        modelViewMatrix = mult(rotateX(5), modelViewMatrix);
        modelViewMatrix = mult(translate(midX, midY, midZ), modelViewMatrix);

        normalMatrix = [
            vec3(modelViewMatrix[0][0], modelViewMatrix[0][1], modelViewMatrix[0]
            vec3(modelViewMatrix[1][0], modelViewMatrix[1][1], modelViewMatrix[1]
            vec3(modelViewMatrix[2][0], modelViewMatrix[2][1], modelViewMatrix[2]
        ];
        gl.uniformMatrix4fv(mVMatLoc, false, flatten(modelViewMatrix));
        gl.uniformMatrix3fv(normalMatrixLoc, false, flatten(normalMatrix));
        render();
        break;
    case 65: //KEY A
        modelViewMatrix = mult(translate(-midX, -midY, -midZ), modelViewMatrix);
        modelViewMatrix = mult(rotateY(-5), modelViewMatrix);
        modelViewMatrix = mult(translate(midX, midY, midZ), modelViewMatrix);

        normalMatrix = [
            vec3(modelViewMatrix[0][0], modelViewMatrix[0][1], modelViewMatrix[0]
            vec3(modelViewMatrix[1][0], modelViewMatrix[1][1], modelViewMatrix[1]
            vec3(modelViewMatrix[2][0], modelViewMatrix[2][1], modelViewMatrix[2]
        ];
        gl.uniformMatrix4fv(mVMatLoc, false, flatten(modelViewMatrix));
        gl.uniformMatrix3fv(normalMatrixLoc, false, flatten(normalMatrix));
        render();
        break;
    case 68: //KEY D
        modelViewMatrix = mult(translate(-midX, -midY, -midZ), modelViewMatrix);
        modelViewMatrix = mult(rotateY(5), modelViewMatrix);
        modelViewMatrix = mult(translate(midX, midY, midZ), modelViewMatrix);

        normalMatrix = [
            vec3(modelViewMatrix[0][0], modelViewMatrix[0][1], modelViewMatrix[0]
            vec3(modelViewMatrix[1][0], modelViewMatrix[1][1], modelViewMatrix[1]
            vec3(modelViewMatrix[2][0], modelViewMatrix[2][1], modelViewMatrix[2]
        ];
        gl.uniformMatrix4fv(mVMatLoc, false, flatten(modelViewMatrix));
        gl.uniformMatrix3fv(normalMatrixLoc, false, flatten(normalMatrix));
        render();
        break;
    case 87: //KEY W
        modelViewMatrix = mult(translate(-midX, -midY, -midZ), modelViewMatrix);
        modelViewMatrix = mult(rotateX(-5), modelViewMatrix);
        modelViewMatrix = mult(translate(midX, midY, midZ), modelViewMatrix);
        normalMatrix = [
            vec3(modelViewMatrix[0][0], modelViewMatrix[0][1], modelViewMatrix[0]
            vec3(modelViewMatrix[1][0], modelViewMatrix[1][1], modelViewMatrix[1]
            vec3(modelViewMatrix[2][0], modelViewMatrix[2][1], modelViewMatrix[2]
        ];
        gl.uniformMatrix4fv(mVMatLoc, false, flatten(modelViewMatrix));
```

```
        break;
    case 38: //Arrowup
        modelViewMatrix = mult(translate(-midX, -midY, -midZ), modelViewMatrix);
        modelViewMatrix = mult(rotateX(-5), modelViewMatrix);
        modelViewMatrix = mult(translate(midX, midY, midZ), modelViewMatrix);
        normalMatrix = [
            vec3(modelViewMatrix[0][0], modelViewMatrix[0][1], modelViewMatrix[0][2]),
            vec3(modelViewMatrix[1][0], modelViewMatrix[1][1], modelViewMatrix[1][2]),
            vec3(modelViewMatrix[2][0], modelViewMatrix[2][1], modelViewMatrix[2][2])
        ];
        gl.uniformMatrix4fv(mVMatLoc, false, flatten(modelViewMatrix));
        gl.uniformMatrix3fv(normalMatrixLoc, false, flatten(normalMatrix));
        render();
        break;
    case 40:  //Arrowdown
        modelViewMatrix = mult(translate(-midX, -midY, -midZ), modelViewMatrix);
        modelViewMatrix = mult(rotateX(5), modelViewMatrix);
        modelViewMatrix = mult(translate(midX, midY, midZ), modelViewMatrix);

        normalMatrix = [
            vec3(modelViewMatrix[0][0], modelViewMatrix[0][1], modelViewMatrix[0][2]),
            vec3(modelViewMatrix[1][0], modelViewMatrix[1][1], modelViewMatrix[1][2]),
            vec3(modelViewMatrix[2][0], modelViewMatrix[2][1], modelViewMatrix[2][2])
        ];
        gl.uniformMatrix4fv(mVMatLoc, false, flatten(modelViewMatrix));
        gl.uniformMatrix3fv(normalMatrixLoc, false, flatten(normalMatrix));
        render();
        break;
    case 39: //KEY RIGHT
        modelViewMatrix = mult(translate(-midX, -midY, -midZ), modelViewMatrix);
        modelViewMatrix = mult(rotateZ(-5), modelViewMatrix);
        modelViewMatrix = mult(translate(midX, midY, midZ), modelViewMatrix);
        normalMatrix = [
            vec3(modelViewMatrix[0][0], modelViewMatrix[0][1], modelViewMatrix[0][2]),
            vec3(modelViewMatrix[1][0], modelViewMatrix[1][1], modelViewMatrix[1][2]),
            vec3(modelViewMatrix[2][0], modelViewMatrix[2][1], modelViewMatrix[2][2])
        ];
        gl.uniformMatrix4fv(mVMatLoc, false, flatten(modelViewMatrix));
        gl.uniformMatrix3fv(normalMatrixLoc, false, flatten(normalMatrix));
        render();
        break;
    case 37: //KEY LEFT
        modelViewMatrix = mult(translate(-midX, -midY, -midZ), modelViewMatrix);
        modelViewMatrix = mult(rotateZ(5), modelViewMatrix);
        modelViewMatrix = mult(translate(midX, midY, midZ), modelViewMatrix);
        normalMatrix = [
            vec3(modelViewMatrix[0][0], modelViewMatrix[0][1], modelViewMatrix[0][2]),
            vec3(modelViewMatrix[1][0], modelViewMatrix[1][1], modelViewMatrix[1][2]),
            vec3(modelViewMatrix[2][0], modelViewMatrix[2][1], modelViewMatrix[2][2])
        ];
        gl.uniformMatrix4fv(mVMatLoc, false, flatten(modelViewMatrix));
        gl.uniformMatrix3fv(normalMatrixLoc, false, flatten(normalMatrix));
        render();
```

Function preapareObject used for adding right quads and also triangles as an attributes.

```
function prepareObject() {
    for (var i = 0; i < quads.length; i += 12) {
        addAttributes(quads[i] - 1, quads[i + 1] - 1, quads[i + 2] - 1);
        addAttributes(quads[i + 3] - 1, quads[i + 4] - 1, quads[i + 5] - 1);
        addAttributes(quads[i + 6] - 1, quads[i + 7] - 1, quads[i + 8] - 1);

        addAttributes(quads[i] - 1, quads[i + 1] - 1, quads[i + 2] - 1);
        addAttributes(quads[i + 6] - 1, quads[i + 7] - 1, quads[i + 8] - 1);
        addAttributes(quads[i + 9] - 1, quads[i + 10] - 1, quads[i + 11] - 1);
    }

    for (var i = 0; i < triangles.length; i += 9) {
        addAttributes(triangles[i] - 1, triangles[i + 1] - 1, triangles[i + 2] - 1);
        addAttributes(triangles[i + 3] - 1, triangles[i + 4] - 1, triangles[i + 5] - 1);
        addAttributes(triangles[i + 6] - 1, triangles[i + 7] - 1, triangles[i + 8] - 1);
    }
}
```

Function findMids used for finding middle axes for the x,y and z axes. We will use these midaxes for rotating the model in 3-dimension.

```
function findMids() {
    var minX, maxX;
    var minY, maxY;
    var minZ, maxZ;
    for (var i = 0; i < vertices.length; i += 3) {
        if (i == 0) {
            minX = vertices[i];
            maxX = vertices[i];
        }
        else {
            if (maxX < vertices[i]) {
                maxX = vertices[i];
            }
            if (minX > vertices[i]) {
                minX = vertices[i];
            }
        }
    }
    for (var i = 1; i < vertices.length; i += 3) {
        if (i == 1) {
            minY = vertices[i];
            maxY = vertices[i];
        }
        else {
            if (maxY < vertices[i]) {
                maxY = vertices[i];
            }
            if (minY > vertices[i]) {
                minY = vertices[i];
            }
        }
```

```
for (var i = 2; i < vertices.length; i += 3) {
    if (i == 2) {
        minZ = vertices[i];
        maxZ = vertices[i];
    }
    else {
        if (maxZ < vertices[i]) {
            maxZ = vertices[i];
        }
        if (minZ > vertices[i]) {
            minZ = vertices[i];
        }
    }
}
midX = (minX + maxX) / 2;
midY = (minY + maxY) / 2;
midZ = (minZ + maxZ) / 2;
```

The following lines of code create buffers in which normal, vertices and triangles will be kept in it.For example; nBuffer for normals, vBuffer for vertices ,tBuffer for triangels.

```
var nBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, nBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(meshNormals), gl.STATIC_DRAW);
var vNormal = gl.getAttribLocation(program, "vNormal");
gl.vertexAttribPointer(vNormal, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vNormal);

var vBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(meshVertices), gl.STATIC_DRAW);
var vPosition = gl.getAttribLocation(program, "vPosition");
gl.vertexAttribPointer(vPosition, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vPosition);

var tBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, tBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(meshTexCoords), gl.STATIC_DRAW);
var vTexCoords = gl.getAttribLocation(program, "vTexCoords");
gl.vertexAttribPointer(vTexCoords, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vTexCoords);
```

In here,using ortho; we set where the model should stand when the program is first opened.
The other lines used for getting location for projectionMatrix and modelViewMatrix.

```
projectionMatrix = ortho(-3.0, 3.0, -5.0, 5.0, -3.0, 3.0);

pMatLoc = gl.getUniformLocation(program, "projectionMatrix");
gl.uniformMatrix4fv(pMatLoc, false, flatten(projectionMatrix));

mVMatLoc = gl.getUniformLocation(program, "modelViewMatrix");
gl.uniformMatrix4fv(mVMatLoc, false, flatten(modelViewMatrix));
```

Here we enter the ambient,diffuse,specular, light and shiness variables of our model.
I used these because that I want to get more realistic model.
Diffuse: for lighting that directly strikes an object and then reflects in all directions
Specular: A light ray that reflects off of a surface at an equal but opposite angle to its incoming angle.
Ambient: To explain briefly “background light”.

```
ambientProduct = mult(ambientLight, materialAmbient);
diffuseProduct = mult(lightColor, materialDiffuse);
specularProduct = mult(lightColor, materialSpecular);

gl.uniform4fv(gl.getUniformLocation(program, "ambientProduct"), flatten(ambientProduct));
gl.uniform4fv(gl.getUniformLocation(program, "diffuseProduct"), flatten(diffuseProduct));
gl.uniform4fv(gl.getUniformLocation(program, "specularProduct"), flatten(specularProduct));
gl.uniform4fv(gl.getUniformLocation(program, "lightDirection"), flatten(lightDirection));
gl.uniform1f(gl.getUniformLocation(program, "shininess"), materialShininess);
```

We set these variables at the beginning of our code.We declare these variables at the beginning of our code.

```
var ambientLight = vec4(0.2, 0.2, 0.2, 1.0);
var lightColor = vec4(0.0, 1.0, 0.0, 1.0);
var lightDirection = vec4(1.0, 1.0, 1.0, 0.0);

var materialAmbient = vec4(0.5, 0.5, 0.5, 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 = 10.0;

var ambientProduct, diffuseProduct, specularProduct;
```

The codes in the screenshots below are for texture.The Uint8Array typed array represents an array of 8-bit unsigned integers.I use the site https://lvgl.io/tools/imageconverter convert to C array format of my texture.

```
var texHeight = 256;
var texWidth = 256;

var myTexels = new Uint8Array([ 0x87, 0x55,0x3e, 0xff, 0x85, 0x53, 0x3a, 0xff, 0x87,
    0x3e, 0xff, 0x82, 0x52, 0x3b, 0xff, 0x85, 0x53, 0x3a, 0xff, 0x87, 0x58, 0x3c, 0x
    0x3f, 0xff, 0x82, 0x58, 0x42, 0xff, 0x83, 0x56, 0x3f, 0xff, 0x85, 0x59, 0x40, 0x
    0x3b, 0xff, 0x7f, 0x59, 0x44, 0xff, 0x7c, 0x53, 0x3d, 0xff, 0x7c, 0x54, 0x3b, 0x
    0x3b, 0xff, 0x78, 0x55, 0x3f, 0xff, 0x75, 0x51, 0x3b, 0xff, 0x76, 0x52, 0x3c, 0x
    0x33, 0xff, 0x67, 0x47, 0x30, 0xff, 0x66, 0x43, 0x2d, 0xff, 0x68, 0x45, 0x2f, 0x
    0x43, 0xff, 0x77, 0x55, 0x3c, 0xff, 0x78, 0x56, 0x3d, 0xff, 0x74, 0x51, 0x3b, 0x
    0x46, 0xff, 0x81, 0x5e, 0x42, 0xff, 0x86, 0x62, 0x48, 0xff, 0x7c, 0x5a, 0x41, 0x
    0x46, 0xff, 0x8a, 0x63, 0x44, 0xff, 0x89, 0x64, 0x47, 0xff, 0x8b, 0x66, 0x4b, 0x
    0x49, 0xff, 0x93, 0x6b, 0x48, 0xff, 0x94, 0x6b, 0x4b, 0xff, 0x94, 0x6d, 0x4e, 0x
    0x4b, 0xff, 0x9b, 0x6f, 0x4c, 0xff, 0x9b, 0x70, 0x4e, 0xff, 0x9a, 0x71, 0x51, 0x
```

```
var texture = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, texture);
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, texWidth, texHeight, 0, gl.RGBA, gl.UNSIGNED_BYTE, myTexels);
gl.generateMipmap(gl.TEXTURE_2D);


gl.uniform1i(gl.getUniformLocation(program, "texmap"), 0);
```

And the last thing I want to mention in the javascript file is how I declare the vertices, quads, normals, triangles and textureCoords. I used MeshLab for converting the .obj file coordinates the format which I want.

```
var vertices=[
    -0.005440, -0.809443, -0.939428,
    0.019167, 3.227034, 0.022684,
    0.637348, -0.809443, -0.705472,
    0.028745, 3.227034, 0.026170,
    0.979368, -1.003520, -0.113076,
    0.033842, 3.227034, 0.034997,
```

```
var normals = [
    0.323800, 0.322100, -0.889600,
    0.873000, 0.302200, -0.382700,
    0.953600, 0.297100, 0.047700,
    0.610900, 0.311200, 0.728000,
    -0.116400, 0.296500, 0.947900,
    -0.531300, 0.301500, 0.791700,
```

```
var triangles = [
    213, 321, 171, 152, 322, 161, 164, 323, 165,
    213, 321, 171, 164, 323, 165, 162, 338, 177,
    213, 321, 171, 174, 356, 188, 184, 357, 192,
    213, 321, 171, 184, 357, 192, 152, 322, 161,
    213, 321, 171, 194, 384, 210, 204, 385, 214,
    213, 321, 171, 204, 385, 214, 174, 356, 188,
    213, 321, 171, 215, 412, 232, 225, 413, 236,
    213, 321, 171, 225, 413, 236, 194, 384, 210,
```

```
var quads = [
    64,,1, 2,,1, 4,,1, 65,,1,
    65,,2, 4,,2, 6,,2, 66,,2,
    66,,3, 6,,3, 8,,3, 67,,3,
    67,,4, 8,,4, 10,,4, 68,,4,
    68,,5, 10,,5, 12,,5, 69,,5,
    69,,6, 12,,6, 14,,6, 70,,6,
    70,,7, 14,,7, 16,,7, 71,,7,
    72,,9, 18,,9, 2,,9, 64,,9,
    71,,10, 16,,10, 18,,10, 72,,10,
    115,,12, 26,,12, 27,,12, 116,,12,
```

```
var textureCoords = [
    0.842209, 0.673918,
    0.842209, 0.700261,
    0.815866, 0.700261,
    0.815866, 0.673918,
    0.709298, 0.606840,
    0.736021, 0.606840,
```

In HTML file;

Firstly buttons were added in the body.

```
<div>
    <button id="wiresBtn">WIRES</button>
    <button id="lightingBtn">LIGHTING</button>
    <button id="texturesNoLightBtn">TEXTURE WITHOUT LIGHTING</button>
    <button id="texturesLightBtn">TEXTURE WITH LIGHTING</button>

</div>
```

In here, we add the vertex shaders that are given to us. Then position, normal and texture coordinate attributes were defined in the vertex shader for repositioning.

```
<script id="vertex-shader" type="x-shader/x-vertex">
    attribute vec4 vPosition;
    attribute vec4 vNormal;
    attribute vec2 vTexCoords;

    uniform mat4 modelViewMatrix;
    uniform mat4 projectionMatrix;

    uniform mat3 normalMatrix;

    varying vec3 vertPos, normalInterp;
    varying vec2 fTexCoords;

    void main() {
        vec4 pos = modelViewMatrix * vPosition;

        vertPos = vec3(pos) / pos.w;

        normalInterp = vec3(normalMatrix * vNormal.xyz);

        fTexCoords = vTexCoords;

        gl_Position = projectionMatrix * pos;
    }
</script>
```

In the last part, we add the fragment shader. And ambient, diffuse,specular variables are added for correct shading, lighting etc. during the rotation.

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

    uniform vec4 ambientProduct;
    uniform vec4 diffuseProduct;
    uniform vec4 specularProduct;
    uniform float shininess;
    uniform int mode;

    uniform vec4 lightDirection;

    varying vec3 vertPos, normalInterp;
    varying vec2 fTexCoords;

    uniform sampler2D texMap;

    void main() {
        if(mode == 1) {
            gl_FragColor = vec4(0.0, 0.0, 0.0, 1.0);
        }
        else {
            vec4 fColor;

            vec3 N = normalize(normalInterp);
            vec3 L = normalize(lightDirection.xyz);

            float lambertian = max(dot(N, L), 0.0);
            vec4 specular = vec4(0.0, 0.0, 0.0, 1.0);

            vec4 ambient = ambientProduct;

            vec4 diffuse = lambertian * diffuseProduct;

            if(lambertian > 0.0) {

                vec3 V = normalize(-vertPos);
                vec3 H = normalize(L + V);
                float specAngle = max(dot(N, H), 0.0);
                specular = specularProduct * pow(specAngle, shininess);
            }
            fColor = ambient + diffuse + specular;
            fColor.a = 1.0;
            if(mode == 2){
                gl_FragColor = fColor;
            }
            else if (mode == 3) {
                gl_FragColor = texture2D(texMap, fTexCoords);
            }
            else if (mode == 4) {
                gl_FragColor = fColor * texture2D(texMap, fTexCoords);
            }
        }
    }
</script>
```

REFERENCES

https://developer.mozilla.org/en-US/

https://webglfundamentals.org/webgl/lessons/

https://www.meshlab.net

https://learnopengl.com/Getting-started/Textures

https://drive.google.com/drive/folders/1g_Ct-kWPTnW3uHjq1ft9wNYOMHwKq1h-?usp=sharing

https://lvgl.io/tools/imageconverter

https://keycode.info.

https://free3d.com/tr/3d-models/lowpoly