



Politechnika Wrocławska

Sprawozdanie 7

Ćwiczenie 7.WebGL

Krzysztof Zalewa

18.1.2025

Spis treści

1 Wstęp teoretyczny

1.1

2 Zadanie laboratoryjne

2.1 Treść zadania

W ramach zadania laboratoryjnego należało napisać program korzystający z WebGL. Program ten miał wyświetlić dwa obiekty sześcia z brakującą ścianką i czworościan. Powinna być możliwość obrotu obiektów, wyboru który obiekt obracamy oraz zwiększenie lub zmniejszenie prędkości obrotu.

2.2 Opis działania programu

Program rysuje obiekty zgodnie z treścią zadania. W prawym górnym rogu znajdują się przyciski które pozwalają na kontrolę obracania. Poniżej przycisków znajduje się suwak który pozwala na przyspieszenie lub zwolnienie animacji.

2.3 Kod programu

```
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8" />
5     <title>Lab7 WebGL</title>
6     <link rel="stylesheet" href="css/style.css">
7     <script
8       src="https://cdnjs.cloudflare.com/ajax/libs/gl-matrix/2.8.1/gl-matrix-min.js"
9       integrity="sha512-zhHQRO/H5SEBL3Wn6yYSaTTZej12z0hVZK0v3TwCUXT1z5qeqGcXJLLrbE_
10      ↪ RYRScEDDpYIJhPC1fk31gqR783iQ=="
11     crossorigin="anonymous"
12     defer></script>
13 <script src="js/init-buffers.js"></script>
14 <script src="js/draw-scene.js"></script>
15 <script src="js/Lab7.js"></script>
16 <script>
17   function loadTexture(gl,shape) {
18     const texture = gl.createTexture();
19     gl.bindTexture(gl.TEXTURE_2D, texture);
20     const level = 0;
21     const internalFormat = gl.RGBA;
22     const width = 1;
23     const height = 1;
24     const border = 0;
25     const srcFormat = gl.RGBA;
26     const srcType = gl.UNSIGNED_BYTE;
27     const pixel = new Uint8Array([0, 0, 255, 255]);
28     gl.texImage2D(
29       gl.TEXTURE_2D,
30       level,
```

```

30         internalFormat,
31         width,
32         height,
33         border,
34         srcFormat,
35         srcType,
36         pixel,
37     );
38     const base64Image1 = "..."
39     const base64Image2 = "...";
40     const image = new Image();
41     image.onload = () => {
42         gl.bindTexture(gl.TEXTURE_2D, texture);
43         gl.texImage2D(
44             gl.TEXTURE_2D,
45             level,
46             internalFormat,
47             srcFormat,
48             srcType,
49             image,
50         );
51         if (isPowerOf2(image.width) && isPowerOf2(image.height)) {
52             gl.generateMipmap(gl.TEXTURE_2D);
53         } else {
54             gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
55             gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
56             gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
57         }
58     };
59     if(shape == "cube"){
60         image.src = base64Image1;
61     }else if(shape == "tetra"){
62         image.src = base64Image2;
63     }
64     return texture;
65 }
66 function isPowerOf2(value) {
67     return (value & (value - 1)) === 0;
68 }
69 </script>
70 </head>
71
72 <body onload="main()">
73     <div class="container">
74         <form>
75             <input type="checkbox" id="cube"> Sześcian
76             <input type="checkbox" id="tetrahedron"> Czworoscian
77         </form>
78         <form>
79             <input type="checkbox" id="rotateX"> Rotacja X
80             <input type="checkbox" id="rotateY"> Rotacja Y
81             <input type="checkbox" id="rotateZ"> Rotacja Z
82         </form>
83         <div class="slidecontainer">
84             <input type="range" min="1" max="100" value="50" class="slider"
85                 ↵ id="speedRange">

```

```

85     </div>
86     <canvas id="gl-canvas"></canvas>
87     <script>
88         document.getElementById("rotateX").checked = false;
89         document.getElementById("rotateX").addEventListener("change", (event) => {
90             rotateX = event.target.checked;
91             console.log("Rotate X:", rotateX);
92         });
93         document.getElementById("rotateY").checked = false;
94         document.getElementById("rotateY").addEventListener("change", (event) => {
95             rotateY = event.target.checked;
96             console.log("Rotate Y:", rotateY);
97         });
98         document.getElementById("rotateZ").checked = false;
99         document.getElementById("rotateZ").addEventListener("change", (event) => {
100             rotateZ = event.target.checked;
101             console.log("Rotate Z:", rotateZ);
102         });
103         document.getElementById("cube").checked = true;
104         document.getElementById("cube").addEventListener("change", (event) => {
105             if (event.target.checked) {
106                 selectedShape = "cube";
107                 document.getElementById("tetrahedron").checked = false;
108                 [prevX, rotationX] = [rotationX, prevX];
109                 [prevY, rotationY] = [rotationY, prevY];
110                 [prevZ, rotationZ] = [rotationZ, prevZ];
111                 console.log("Selected shape:", selectedShape);
112             }
113         });
114         document.getElementById("tetrahedron").checked = false;
115         document.getElementById("tetrahedron").addEventListener("change", (event)
116             ↵ => {
117             if (event.target.checked) {
118                 selectedShape = "tetrahedron";
119                 document.getElementById("cube").checked = false;
120                 [prevX, rotationX] = [rotationX, prevX];
121                 [prevY, rotationY] = [rotationY, prevY];
122                 [prevZ, rotationZ] = [rotationZ, prevZ];
123                 console.log("Selected shape:", selectedShape);
124             }
125         });
126         document.getElementById("speedRange").value = speed;
127         document.getElementById("speedRange").addEventListener("input", (event) =>
128             ↵ {
129             speed = event.target.value;
130             console.log("Rotation speed:", speed);
131         });
132     </script>
133 </div>
</body>
</html>

```

Fragment kodu 1: Fragment kodu z programu

```

1  let speed = 50;
2
3  let rotateX = false;
4  let rotateY = false;
5  let rotateZ = false;
6  let selectedShape = "cube";
7  let rotationX = 0;
8  let rotationY = 0;
9  let rotationZ = 0;
10 let prevX = 0;
11 let prevY = 0;
12 let prevZ = 0;
13
14 function main() {
15     const canvas = document.querySelector("#gl-canvas");
16     canvas.width = canvas.clientWidth;
17     canvas.height = canvas.clientHeight;
18     const gl = canvas.getContext("webgl");
19     if (gl === null) {
20         alert(
21             "Unable to initialize WebGL. Your browser or machine may not support it.",
22         );
23         return;
24     }
25     gl.viewport(0, 0, gl.drawingBufferWidth, gl.drawingBufferHeight);
26     gl.clearColor(0.0, 0.0, 0.0, 1.0);
27     gl.clear(gl.COLOR_BUFFER_BIT);
28     const vsSource = `
29         attribute vec4 aVertexPosition;
30         attribute vec2 aTextureCoord;
31         uniform mat4 uModelViewMatrix;
32         uniform mat4 uProjectionMatrix;
33         varying highp vec2 vTextureCoord;
34         void main(void) {
35             gl_Position = uProjectionMatrix * uModelViewMatrix * aVertexPosition;
36             vTextureCoord = aTextureCoord;
37         }
38     `;
39     const fsSource = `
40         varying highp vec2 vTextureCoord;
41         uniform sampler2D uSampler;
42         void main(void) {
43             gl_FragColor = texture2D(uSampler, vTextureCoord);
44         }
45     `;
46
47     const shaderProgram = initShaderProgram(gl, vsSource, fsSource);
48     const programInfo = {
49         program: shaderProgram,
50         attribLocations: {
51             vertexPosition: gl.getAttribLocation(shaderProgram, "aVertexPosition"),
52             textureCoord: gl.getAttribLocation(shaderProgram, "aTextureCoord"),
53         },
54         uniformLocations: {
55             projectionMatrix: gl.getUniformLocation(shaderProgram,
56                 ↵ "uProjectionMatrix"),

```

```

56     modelViewMatrix: gl.getUniformLocation(shaderProgram, "uModelViewMatrix"),
57     uSampler: gl.getUniformLocation(shaderProgram, "uSampler"),
58 },
59 };
60 const buffers = initBuffers(gl);
61 const texture1 = loadTexture(gl, "cube");
62 const texture2 = loadTexture(gl, "tetr");
63 gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
64 function render(now) {
65     rotationX = rotateX ? rotationX + speed * 0.002 : rotationX;
66     rotationY = rotateY ? rotationY + speed * 0.002 : rotationY;
67     rotationZ = rotateZ ? rotationZ + speed * 0.002 : rotationZ;
68     drawScene(gl, programInfo, buffers, rotationX, rotationY, rotationZ, prevX, prevY,
        ↪ prevZ, texture1, texture2, selectedShape);
69     requestAnimationFrame(render);
70 }
71 requestAnimationFrame(render);
72 }
73 function initShaderProgram(gl, vsSource, fsSource) {
74     const vertexShader = loadShader(gl, gl.VERTEX_SHADER, vsSource);
75     const fragmentShader = loadShader(gl, gl.FRAGMENT_SHADER, fsSource);
76     const shaderProgram = gl.createProgram();
77     gl.attachShader(shaderProgram, vertexShader);
78     gl.attachShader(shaderProgram, fragmentShader);
79     gl.linkProgram(shaderProgram);
80     if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
81         alert(
82             `Unable to initialize the shader program: ${gl.getProgramInfoLog(
83                 shaderProgram,
84             )}`);
85     );
86     return null;
87 }
88 return shaderProgram;
89 }
90 function loadShader(gl, type, source) {
91     const shader = gl.createShader(type);
92     gl.shaderSource(shader, source);
93     gl.compileShader(shader);
94     if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
95         alert(
96             `An error occurred compiling the shaders: ${gl.getShaderInfoLog(shader)}`,
97         );
98         gl.deleteShader(shader);
99         return null;
100     }
101     return shader;
102 }

```

Fragment kodu 2: Fragment kodu z programu

```

1 function initBuffers(gl) {
2     const cubeBuffers = {
3         position: initCubePositionBuffer(gl),

```

```

4         textureCoord: initCubeTextureBuffer(gl),
5         indices: initCubeIndexBuffer(gl),
6     }
7     const tetrBuffers = {
8         position: initTetrPositionBuffer(gl),
9         textureCoord: initTetrTextureBuffer(gl),
10        indices: initTetrIndexBuffer(gl),
11    }
12    return {
13        cube: cubeBuffers,
14        tetr: tetrBuffers,
15    };
16 }
17 function initCubePositionBuffer(gl) {
18     const positionBuffer = gl.createBuffer();
19     gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
20     const positions = [
21         -1.0, -1.0, 1.0, 1.0, -1.0, 1.0, 1.0, 1.0, 1.0, -1.0, 1.0, 1.0, //
22         ↪ front
23         -1.0, -1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, -1.0, 1.0, -1.0, -1.0, //
24         ↪ back
25         -1.0, 1.0, -1.0, -1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, -1.0, //
26         ↪ top
27         -1.0, -1.0, -1.0, 1.0, -1.0, -1.0, 1.0, -1.0, 1.0, -1.0, -1.0, 1.0, //
28         ↪ bottom
29         1.0, -1.0, -1.0, 1.0, 1.0, -1.0, 1.0, 1.0, 1.0, 1.0, -1.0, 1.0, //
30         ↪ right
31     ];
32     gl.bufferData(
33         gl.ARRAY_BUFFER,
34         new Float32Array(positions),
35         gl.STATIC_DRAW
36     );
37     return positionBuffer;
38 }
39 function initCubeIndexBuffer(gl) {
40     const indexBuffer = gl.createBuffer();
41     gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
42     const indices = [
43         0,1,2,0,2,3, // front
44         4,5,6,4,6,7, // back
45         8,9,10,8,10,11, // top
46         12,13,14,12,14,15, // bottom
47         16,17,18,16,18,19, // right
48     ];
49     gl.bufferData(
50         gl.ELEMENT_ARRAY_BUFFER,
51         new Uint16Array(indices),
52         gl.STATIC_DRAW,
53     );
54     return indexBuffer;
55 }
56 function initCubeTextureBuffer(gl) {
57     const textureCoordBuffer = gl.createBuffer();
58     gl.bindBuffer(gl.ARRAY_BUFFER, textureCoordBuffer);
59     const textureCoordinates = [

```

```

55         0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,    // front
56         0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,    // back
57         0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,    // top
58         0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,    // bottom
59         0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,    // right
60     ];
61     gl.bufferData(
62         gl.ARRAY_BUFFER,
63         new Float32Array(textureCoordinates),
64         gl.STATIC_DRAW,
65     );
66     return textureCoordBuffer;
67 }
68 function initTetrPositionBuffer(gl) {
69     const positionBuffer = gl.createBuffer();
70     gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
71     const positions = [
72         0.0, 1.0, 0.0,    // Top vertex
73         -1.0, -1.0, 1.0,  // Front-left
74         1.0, -1.0, 1.0,   // Front-right
75         0.0, -1.0, -1.0,  // Back
76     ];
77     gl.bufferData(
78         gl.ARRAY_BUFFER,
79         new Float32Array(positions),
80         gl.STATIC_DRAW
81     );
82     return positionBuffer;
83 }
84 function initTetrIndexBuffer(gl) {
85     const indexBuffer = gl.createBuffer();
86     gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
87     const indices = [
88         0, 1, 2,  // Front face
89         0, 2, 3,  // Right face
90         0, 3, 1,  // Left face
91         1, 3, 2,  // Bottom face
92     ];
93     gl.bufferData(
94         gl.ELEMENT_ARRAY_BUFFER,
95         new Uint16Array(indices),
96         gl.STATIC_DRAW
97     );
98     return indexBuffer;
99 }
100 function initTetrTextureBuffer(gl) {
101     const textureCoordBuffer = gl.createBuffer();
102     gl.bindBuffer(gl.ARRAY_BUFFER, textureCoordBuffer);
103     const textureCoordinates = [
104         0.5, 1.0,    // Top vertex
105         0.0, 0.0,    // Front-left
106         1.0, 0.0,    // Front-right
107         0.5, 0.0,    // Back
108     ];
109     gl.bufferData(
110         gl.ARRAY_BUFFER,

```



```

111         new Float32Array(textureCoordinates),
112         gl.STATIC_DRAW
113     );
114     return textureCoordBuffer;
115 }

```

Fragment kodu 3: Fragment kodu z programu

```

1  function drawScene(gl, programInfo, buffers, rotationX, rotationY, rotationZ, prevX, pr
↪   evY, prevZ, texture1, texture2, selectedShape) {
2      gl.clearColor(0.0, 0.0, 0.0, 1.0); // Clear to black, fully opaque
3      gl.clearDepth(1.0); // Clear everything
4      gl.enable(gl.DEPTH_TEST); // Enable depth testing
5      gl.depthFunc(gl.LEQUAL); // Near things obscure far things
6      gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
7      const fieldOfView = (45 * Math.PI) / 180; // in radians
8      const aspect = gl.canvas.clientWidth / gl.canvas.clientHeight;
9      const zNear = 0.1;
10     const zFar = 100.0;
11     const projectionMatrix = mat4.create();
12     mat4.perspective(projectionMatrix, fieldOfView, aspect, zNear, zFar);
13     let cubeX; let tetrX;
14     let cubeY; let tetrY;
15     let cubeZ; let tetrZ;
16     if(selectedShape == "cube"){
17         cubeX = rotationX; tetrX = prevX;
18         cubeY = rotationY; tetrY = prevY;
19         cubeZ = rotationZ; tetrZ = prevZ;
20     } else if(selectedShape == "tetrahedron"){
21         tetrX = rotationX; cubeX = prevX;
22         tetrY = rotationY; cubeY = prevY;
23         tetrZ = rotationZ; cubeZ = prevZ;
24     }
25     drawFigure(gl, programInfo, buffers.cube, cubeX, cubeY, cubeZ, texture1, projectionMa
↪   trix, 30, -2);
26     drawFigure(gl, programInfo, buffers.tetr, tetrX, tetrY, tetrZ, texture2, projectionMa
↪   trix, 12, 2);
27 }
28 function drawFigure(gl, programInfo, figure, rotationX, rotationY, rotationZ, texture, pr
↪   ojectionMatrix, vertNum, move){
29     const modelViewMatrix = mat4.create();
30     mat4.translate(
31         modelViewMatrix, // destination matrix
32         modelViewMatrix, // matrix to translate
33         [move, 0.0, -6.0],
34     ); // amount to translate
35     mat4.rotate(
36         modelViewMatrix, // destination matrix
37         modelViewMatrix, // matrix to rotate
38         rotationZ, // amount to rotate in radians
39         [0, 0, 1],
40     ); // axis to rotate around (Z)
41     mat4.rotate(
42         modelViewMatrix, // destination matrix

```

```

43     modelViewMatrix, // matrix to rotate
44     rotationY , // amount to rotate in radians
45     [0, 1, 0],
46 ); // axis to rotate around (Y)
47 mat4.rotate(
48     modelViewMatrix, // destination matrix
49     modelViewMatrix, // matrix to rotate
50     rotationX , // amount to rotate in radians
51     [1, 0, 0],
52 );
53 setPositionAttribute(gl, figure, programInfo);
54 setTextureAttribute(gl, figure, programInfo);
55 gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, figure.indices);
56 gl.useProgram(programInfo.program);
57 gl.uniformMatrix4fv(
58     programInfo.uniformLocations.projectionMatrix,
59     false,
60     projectionMatrix,
61 );
62 gl.uniformMatrix4fv(
63     programInfo.uniformLocations.modelViewMatrix,
64     false,
65     modelViewMatrix,
66 );
67 gl.activeTexture(gl.TEXTURE0);
68 gl.bindTexture(gl.TEXTURE_2D, texture);
69 gl.uniform1i(programInfo.uniformLocations.uSampler, 0);
70 {
71     const vertexCount = vertNum;
72     const type = gl.UNSIGNED_SHORT;
73     const offset = 0;
74     gl.drawElements(gl.TRIANGLES, vertexCount, type, offset);
75 }
76 }
77 function setPositionAttribute(gl, buffers, programInfo) {
78     const numComponents = 3;
79     const type = gl.FLOAT;
80     const normalize = false;
81     const stride = 0;
82     const offset = 0;
83     gl.bindBuffer(gl.ARRAY_BUFFER, buffers.position);
84     gl.vertexAttribPointer(
85         programInfo.attribLocations.vertexPosition,
86         numComponents,
87         type,
88         normalize,
89         stride,
90         offset,
91     );
92     gl.enableVertexAttribArray(programInfo.attribLocations.vertexPosition);
93 }
94 function setTextureAttribute(gl, buffers, programInfo) {
95     const num = 2;
96     const type = gl.FLOAT;
97     const normalize = false;
98     const stride = 0;

```

```
99     const offset = 0;
100    gl.bindBuffer(gl.ARRAY_BUFFER, buffers.textureCoord);
101    gl.vertexAttribPointer(
102        programInfo.attribLocations.textureCoord,
103        num,
104        type,
105        normalize,
106        stride,
107        offset,
108    );
109    gl.enableVertexAttribArray(programInfo.attribLocations.textureCoord);
110 }
111
```

Fragment kodu 4: Fragment kodu z programu

3 Źródła

- 1.