



Politechnika Wrocławska

Sprawozdanie 7

Ćwiczenie 7.WebGL

Krzysztof Zalewa

20.1.2025

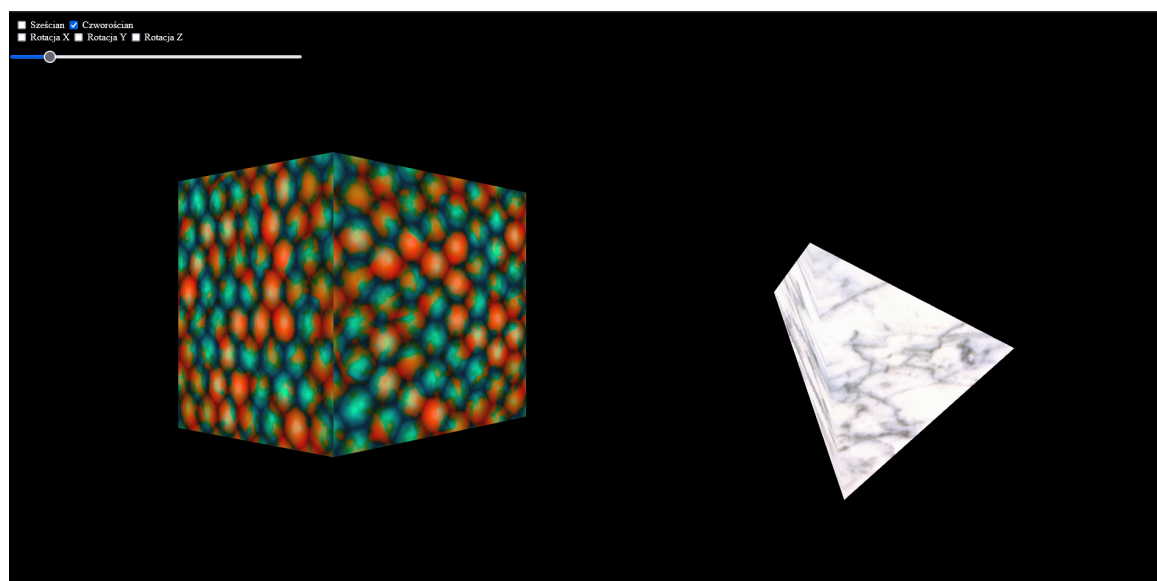
Spis treści

1	Wstęp teoretyczny	2
1.1	WebGL	2
1.2	GLSL	2
1.3	Użyte funkcje	3
2	Zadanie laboratoryjne	3
2.1	Treść zadania	3
2.2	Opis działania programu	3
2.3	Kod programu	3
3	Źródła	12

1 Wstęp teoretyczny

1.1 WebGL

WebGL jest Api pozwalającym na renderowanie 2D i 3D w obiekcie canvas z html (W przeglądarkach które na to pozwalają). Programy używające WebGL składają się z kodu głównego napisanego w JavaScript i kodu shaderów napisanego w GLSL. Kod ten jest następnie wykonywany przez kartę graficzną urządzenia.



Rysunek 1: Przykład programu z wykorzystaniem WebGL

1.2 GLSL

Język programowania potoku graficznego (ang. OpenGL Shading Language). Język ten jest używany w nowszych wersjach OpenGL zamiast niektórych funkcji wbudowanych ponieważ pozwala on na znacznie większą elastyczność modelu programowania potoku graficznego. W moim programie zaimplementowałem dwa shadery:

1. **Shader wierzchołków** - Za każdym razem gdy kształt jest renderowany dla każdego wierzchołka uruchamiany jest shader wierzchołków. Jego zadaniem jest transformacja oryginalnych kordynatów na kordynaty używane przez WebGL (W zakresie -1.0 do 1.0).

Shader ten może mieć wiele atrybutów które min. pozwalają określić kordynaty tekstury. Na koniec przetransformowana pozycja przekazywana jest do WebGL przez specjalną zmienną `gl_Position`.

```
1      const vsSource = `
2      attribute vec4 aVertexPosition;
3      attribute vec2 aTextureCoord;
4      uniform mat4 uModelViewMatrix;
5      uniform mat4 uProjectionMatrix;
6      varying highp vec2 vTextureCoord;
7      void main(void) {
8          gl_Position = uProjectionMatrix * uModelViewMatrix
9              ↪ * aVertexPosition;
10         vTextureCoord = aTextureCoord;
11     }
12     `;
```

Fragment kodu 1: Przykładowy shader wierzchołków

2. **Shader fragmetów** - Za każdym gdy shader wierzchołków wykona swoje zadanie i narysuje wierzchołki zaczynają być rysowane pojedyncze piksele. Shader fragmetów jest wywoływany dla każdego piksela. Jego zadaniem jest wybranie koloru danego piskela na podstawie tekseli(piksel w teksturze). Następnie kolor jest przekazywany do WebGL przez specjalną zmienną `gl_FragColor`. Następnie odpowiedni kolor jest rysowany w miejscu piksela.

```
1      const fsSource = `
2      varying highp vec2 vTextureCoord;
3      uniform sampler2D uSampler;
4      void main(void) {
5          gl_FragColor = texture2D(uSampler, vTextureCoord);
6      }
7      `;
```

Fragment kodu 2: Przykładowy shader fragmentów

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-zhHQR0/H5SEBL3Wn6yYSaTTZej12z0hVZK0v3TwCUXT1z5qqGcXJLLrbE_
   ↪  RYRScEDDpYIJhPC1fk31gqR783iQ=="
10       crossorigin="anonymous"
11       defer></script>
12      <script src="js/init-buffers.js"></script>
13      <script src="js/draw-scene.js"></script>
14      <script src="js/Lab7.js"></script>
15      <script>
16        function loadTexture(gl,shape) {
17          const texture = gl.createTexture();
18          gl.bindTexture(gl.TEXTURE_2D, texture);
19          const level = 0;
20          const internalFormat = gl.RGBA;
21          const width = 1;
22          const height = 1;
23          const border = 0;
24          const srcFormat = gl.RGBA;
25          const srcType = gl.UNSIGNED_BYTE;
26          const pixel = new Uint8Array([0, 0, 255, 255]);
27          gl.texImage2D(
28            gl.TEXTURE_2D,
29            level,
30            internalFormat,
31            width,
32            height,
33            border,
34            srcFormat,
35            srcType,
36            pixel,
37          );
38          const base64Image1 = "data:image/jpeg;base64,iVBORwOKGgoAAAANSUh...";
39          const base64Image2 = "data:image/jpeg;base64,iVBORwOKGgoAAAANSUh...";
40          //Te dwie zmienne w orginale zawierają tekstury zakodowane w base64 żeby
   ↪  nie zaśmiecać
41          //sprawozdania losowymi znakami uciąłem ciągi danych
42          const image = new Image();
43          image.onload = () => {
44            gl.bindTexture(gl.TEXTURE_2D, texture);
45            gl.texImage2D(
46              gl.TEXTURE_2D,
47              level,
48              internalFormat,
49              srcFormat,
50              srcType,
51              image,
52            );
53            if (isPowerOf2(image.width) && isPowerOf2(image.height)) {
54              gl.generateMipmap(gl.TEXTURE_2D);

```

```

55     } else {
56         gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
57         gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
58         gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
59     }
60 };
61 if(shape == "cube"){
62     image.src = base64Image1;
63 }else if(shape == "tetr"){
64     image.src = base64Image2;
65 }
66 return texture;
67 }
68 function isPowerOf2(value) {
69     return (value & (value - 1)) === 0;
70 }
71 </script>
72 </head>
73
74 <body onload="main()">
75     <div class="container">
76         <form>
77             <input type="checkbox" id="cube"> Sześcian
78             <input type="checkbox" id="tetrahedron"> Czworościan
79         </form>
80         <form>
81             <input type="checkbox" id="rotateX"> Rotacja X
82             <input type="checkbox" id="rotateY"> Rotacja Y
83             <input type="checkbox" id="rotateZ"> Rotacja Z
84         </form>
85         <div class="slidecontainer">
86             <input type="range" min="1" max="100" value="50" class="slider"
87                 ↵ id="speedRange">
88         </div>
89         <canvas id="gl-canvas"></canvas>
90         <script>
91             document.getElementById("rotateX").checked = false;
92             document.getElementById("rotateX").addEventListener("change", (event) => {
93                 rotateX = event.target.checked;
94                 console.log("Rotate X:", rotateX);
95             });
96             document.getElementById("rotateY").checked = false;
97             document.getElementById("rotateY").addEventListener("change", (event) => {
98                 rotateY = event.target.checked;
99                 console.log("Rotate Y:", rotateY);
100             });
101             document.getElementById("rotateZ").checked = false;
102             document.getElementById("rotateZ").addEventListener("change", (event) => {
103                 rotateZ = event.target.checked;
104                 console.log("Rotate Z:", rotateZ);
105             });
106             document.getElementById("cube").checked = true;
107             document.getElementById("cube").addEventListener("change", (event) => {
108                 if (event.target.checked) {
109                     selectedShape = "cube";
110                     document.getElementById("tetrahedron").checked = false;

```

```

110         [prevX, rotationX] = [rotationX, prevX];
111         [prevY, rotationY] = [rotationY, prevY];
112         [prevZ, rotationZ] = [rotationZ, prevZ];
113         console.log("Selected shape:", selectedShape);
114     }
115 });
116 document.getElementById("tetrahedron").checked = false;
117 document.getElementById("tetrahedron").addEventListener("change", (event)
↪ => {
118     if (event.target.checked) {
119         selectedShape = "tetrahedron";
120         document.getElementById("cube").checked = false;
121         [prevX, rotationX] = [rotationX, prevX];
122         [prevY, rotationY] = [rotationY, prevY];
123         [prevZ, rotationZ] = [rotationZ, prevZ];
124         console.log("Selected shape:", selectedShape);
125     }
126 });
127 document.getElementById("speedRange").value = speed;
128 document.getElementById("speedRange").addEventListener("input", (event) =>
↪ {
129     speed = event.target.value;
130     console.log("Rotation speed:", speed);
131 });
132 </script>
133 </div>
134 </body>
135 </html>

```

Fragment kodu 3: 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"),
57             modelViewMatrix: gl.getUniformLocation(shaderProgram, "uModelViewMatrix"),
58             uSampler: gl.getUniformLocation(shaderProgram, "uSampler"),
59         },
60     };
61     const buffers = initBuffers(gl);
62     const texture1 = loadTexture(gl, "cube");
63     const texture2 = loadTexture(gl, "tetr");
64     gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
65     function render(now) {
66         rotationX = rotateX ? rotationX + speed * 0.002 : rotationX;
67         rotationY = rotateY ? rotationY + speed * 0.002 : rotationY;
68         rotationZ = rotateZ ? rotationZ + speed * 0.002 : rotationZ;
69         drawScene(gl, programInfo, buffers, rotationX, rotationY, rotationZ, prevX, prevY,
70             ↪ prevZ, texture1, texture2, selectedShape);
71         requestAnimationFrame(render);
72     }
73     requestAnimationFrame(render);
74 }
75
76 function initShaderProgram(gl, vsSource, fsSource) {
77     const vertexShader = loadShader(gl, gl.VERTEX_SHADER, vsSource);
78     const fragmentShader = loadShader(gl, gl.FRAGMENT_SHADER, fsSource);
79     const shaderProgram = gl.createProgram();
80     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 4: 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, //
26         ↪ top

```



```

24         -1.0, -1.0, -1.0, 1.0, -1.0, -1.0, 1.0, -1.0, 1.0, -1.0, -1.0, 1.0,    //
           ↪ bottom
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,    //
           ↪ right
26     ];
27     gl.bufferData(
28         gl.ARRAY_BUFFER,
29         new Float32Array(positions),
30         gl.STATIC_DRAW
31     );
32     return positionBuffer;
33 }
34 function initCubeIndexBuffer(gl) {
35     const indexBuffer = gl.createBuffer();
36     gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
37     const indices = [
38         0,1,2,0,2,3,                // front
39         4,5,6,4,6,7,                // back
40         8,9,10,8,10,11,             // top
41         12,13,14,12,14,15,          // bottom
42         16,17,18,16,18,19,          // right
43     ];
44     gl.bufferData(
45         gl.ELEMENT_ARRAY_BUFFER,
46         new Uint16Array(indices),
47         gl.STATIC_DRAW,
48     );
49     return indexBuffer;
50 }
51 function initCubeTextureBuffer(gl) {
52     const textureCoordBuffer = gl.createBuffer();
53     gl.bindBuffer(gl.ARRAY_BUFFER, textureCoordBuffer);
54     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 5: 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 6: Fragment kodu z programu

3 Źródła

1. https://pl.wikipedia.org/wiki/OpenGL_Shading_Language
2. [https://www.khronos.org/opengl/wiki/Core_Language_\(GLSL\)](https://www.khronos.org/opengl/wiki/Core_Language_(GLSL))

3. https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial/Getting_started_with_WebGL
4. https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial/Adding_2D_content_to_a_WebGL_context
5. https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial/Animating_objects_with_WebGL
6. https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial/Creating_3D_objects_using_WebGL
7. https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial/Using_textures_in_WebGL