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 kożystający 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

.

```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Lab7 WebGL</title>
    <link rel="stylesheet" href="css/style.css">
    <script
      src="https://cdnjs.cloudflare.com/ajax/libs/gl-matrix/2.8.1/gl-matrix-min.js"
      integrity="sha512-zhHQR0/H5SEBL3Wn6yYSaTTZej12z0hVZKOv3TwCUXT1z5qeqGcXJLLrbERYRScEDDpYIJhPC1fk31gqR783iQ=="
      crossorigin="anonymous"
      defer></script>
    <script src="js/init-buffers.js"></script>
    <script src="js/draw-scene.js"></script>
    <script src="js/Lab7.js"></script>
    <script>
      function loadTexture(gl,shape) {
        const texture = gl.createTexture();
        gl.bindTexture(gl.TEXTURE_2D, texture);
        const level = 0;
        const internalFormat = gl.RGBA;
        const width = 1;
        const height = 1;
        const border = 0;
        const srcFormat = gl.RGBA;
        const srcType = gl.UNSIGNED_BYTE;
        const pixel = new Uint8Array([0, 0, 255, 255]);
        gl.texImage2D(
          gl.TEXTURE_2D,
          level,
        internalFormat,
        width,
        height,
        border,
        srcFormat,
        srcType,
        pixel,
      );
      const base64Image1 = "data:image/jpeg;base64,iVBORw0KGgoAAAANSUh..."
      const base64Image2 = "data:image/jpeg;base64,iVBORw0KGgoAAAANSUh...";
    const image = new Image();
    image.onload = () => {
      gl.bindTexture(gl.TEXTURE_2D, texture);
      gl.texImage2D(
        gl.TEXTURE_2D,
        level,
        internalFormat,
        srcFormat,
        srcType,
        image,
      );
      if (isPowerOf2(image.width) && isPowerOf2(image.height)) {
        gl.generateMipmap(gl.TEXTURE_2D);
      } else {
        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
      }
    };
    if(shape == "cube"){
      image.src = base64Image1;
    }else if(shape == "tetr"){
      image.src = base64Image2;
    }
    return texture;
  }
  function isPowerOf2(value) {
    return (value & (value - 1)) === 0;
  }
  </script>
</head>

<body onload="main()">
  <div class="container">
    <form>
      <input type="checkbox" id="cube"> Sześcian
      <input type="checkbox" id="tetrahedron"> Czworościan
    </form>
    <form>
      <input type="checkbox" id="rotateX"> Rotacja X
      <input type="checkbox" id="rotateY"> Rotacja Y
      <input type="checkbox" id="rotateZ"> Rotacja Z
    </form>
    <div class="slidecontainer">
        <input type="range" min="1" max="100" value="50" class="slider"
        ↪  id="speedRange">
```

```
   </div>
   <canvas id="gl-canvas"></canvas>
   <script>
    document.getElementById("rotateX").checked = false;
    document.getElementById("rotateX").addEventListener("change", (event) => {
     rotateX = event.target.checked;
     console.log("Rotate X:", rotateX);
    });
    document.getElementById("rotateY").checked = false;
    document.getElementById("rotateY").addEventListener("change", (event) => {
     rotateY = event.target.checked;
     console.log("Rotate Y:", rotateY);
    });
    document.getElementById("rotateZ").checked = false;
    document.getElementById("rotateZ").addEventListener("change", (event) => {
     rotateZ = event.target.checked;
     console.log("Rotate Z:", rotateZ);
    });
    document.getElementById("cube").checked = true;
    document.getElementById("cube").addEventListener("change", (event) => {
     if (event.target.checked) {
       selectedShape = "cube";
       document.getElementById("tetrahedron").checked = false;
       [prevX, rotationX] = [rotationX, prevX];
       [prevY, rotationY] = [rotationY, prevY];
       [prevZ, rotationZ] = [rotationZ, prevZ];
       console.log("Selected shape:", selectedShape);
     }
    });
    document.getElementById("tetrahedron").checked = false;
    document.getElementById("tetrahedron").addEventListener("change", (event) => {
     if (event.target.checked) {
       selectedShape = "tetrahedron";
       document.getElementById("cube").checked = false;
       [prevX, rotationX] = [rotationX, prevX];
       [prevY, rotationY] = [rotationY, prevY];
       [prevZ, rotationZ] = [rotationZ, prevZ];
       console.log("Selected shape:", selectedShape);
     }
    });
    document.getElementById("speedRange").value = speed;
    document.getElementById("speedRange").addEventListener("input", (event) => {
     speed = event.target.value;
     console.log("Rotation speed:", speed);
    });
   </script>
  </div>
 </body>
</html>
```

Fragment kodu 1: Fragment kodu z programu

.

```
let speed = 50;

let rotateX = false;
let rotateY = false;
let rotateZ = false;
let selectedShape = "cube";
let rotationX = 0;
let rotationY = 0;
let rotationZ = 0;
let prevX = 0;
let prevY = 0;
let prevZ = 0;

function main() {
    const canvas = document.querySelector("#gl-canvas");
    canvas.width = canvas.clientWidth;
    canvas.height = canvas.clientHeight;
    const gl = canvas.getContext("webgl");
    if (gl === null) {
        alert(
        "Unable to initialize WebGL. Your browser or machine may not support it.",
        );
        return;
    }
    gl.viewport(0, 0, gl.drawingBufferWidth, gl.drawingBufferHeight);
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);
    const vsSource = `
      attribute vec4 aVertexPosition;
      attribute vec2 aTextureCoord;
      uniform mat4 uModelViewMatrix;
      uniform mat4 uProjectionMatrix;
      varying highp vec2 vTextureCoord;
      void main(void) {
        gl_Position = uProjectionMatrix * uModelViewMatrix * aVertexPosition;
        vTextureCoord = aTextureCoord;
      }
    `;
    const fsSource = `
      varying highp vec2 vTextureCoord;
      uniform sampler2D uSampler;
      void main(void) {
        gl_FragColor = texture2D(uSampler, vTextureCoord);
      }
    `;

    const shaderProgram = initShaderProgram(gl, vsSource, fsSource);
    const programInfo = {
      program: shaderProgram,
      attribLocations: {
        vertexPosition: gl.getAttribLocation(shaderProgram, "aVertexPosition"),
        textureCoord: gl.getAttribLocation(shaderProgram, "aTextureCoord"),
      },
      uniformLocations: {
        projectionMatrix: gl.getUniformLocation(shaderProgram,
         ↪  "uProjectionMatrix"),
        modelViewMatrix: gl.getUniformLocation(shaderProgram, "uModelViewMatrix"),
        uSampler: gl.getUniformLocation(shaderProgram, "uSampler"),
      },
    };
    const buffers = initBuffers(gl);
    const texture1 = loadTexture(gl, "cube");
    const texture2 = loadTexture(gl, "tetr");
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
    function render(now) {
      rotationX = rotateX ? rotationX + speed * 0.002 : rotationX;
      rotationY = rotateY ? rotationY + speed * 0.002 : rotationY;
      rotationZ = rotateZ ? rotationZ + speed * 0.002 : rotationZ;
      drawScene(gl, programInfo, buffers, rotationX,rotationY,rotationZ,prevX,prev
       ↪  Y,prevZ,texture1,texture2,selectedShape);
      requestAnimationFrame(render);
    }
    requestAnimationFrame(render);
}
function initShaderProgram(gl, vsSource, fsSource) {
    const vertexShader = loadShader(gl, gl.VERTEX_SHADER, vsSource);
    const fragmentShader = loadShader(gl, gl.FRAGMENT_SHADER, fsSource);
    const shaderProgram = gl.createProgram();
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    gl.linkProgram(shaderProgram);
    if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
      alert(
        `Unable to initialize the shader program: ${gl.getProgramInfoLog(
          shaderProgram,
        )}`,
      );
      return null;
    }
    return shaderProgram;
}
function loadShader(gl, type, source) {
    const shader = gl.createShader(type);
    gl.shaderSource(shader, source);
    gl.compileShader(shader);
    if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
      alert(
        `An error occurred compiling the shaders: ${gl.getShaderInfoLog(shader)}`,
      );
      gl.deleteShader(shader);
      return null;
    }
    return shader;
}
```

Fragment kodu 2: Fragment kodu z programu

.

```
function initBuffers(gl) {
    const cubeBuffers = {
        position: initCubePositionBuffer(gl),
        textureCoord: initCubeTextureBuffer(gl),
        indices: initCubeIndexBuffer(gl),
    }
    const tetrBuffers ={
        position: initTetrPositionBuffer(gl),
        textureCoord: initTetrTextureBuffer(gl),
        indices: initTetrIndexBuffer(gl),
    }
    return {
        cube: cubeBuffers,
        tetr: tetrBuffers,
    };
}
function initCubePositionBuffer(gl) {
    const positionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
    const positions = [
        -1.0, -1.0, 1.0, 1.0, -1.0, 1.0, 1.0, 1.0, 1.0, -1.0, 1.0, 1.0,         //
        ↪ front
        -1.0, -1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, -1.0, 1.0, -1.0, -1.0,     //
        ↪ back
        -1.0, 1.0, -1.0, -1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, -1.0,         //
        ↪ top
        -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
        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
      ];
    gl.bufferData(
        gl.ARRAY_BUFFER,
        new Float32Array(positions),
        gl.STATIC_DRAW
    );
    return positionBuffer;
}
function initCubeIndexBuffer(gl) {
    const indexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
    const indices = [
      0,1,2,0,2,3,          // front
      4,5,6,4,6,7,          // back
      8,9,10,8,10,11,       // top
      12,13,14,12,14,15,    // bottom
      16,17,18,16,18,19,    // right
    ];
    gl.bufferData(
      gl.ELEMENT_ARRAY_BUFFER,
      new Uint16Array(indices),
      gl.STATIC_DRAW,
    );
    return indexBuffer;
  }
function initCubeTextureBuffer(gl) {
    const textureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, textureCoordBuffer);
    const textureCoordinates = [
      0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,   // front
      0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,   // back
      0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,   // top
      0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,   // bottom
      0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,   // right
    ];
    gl.bufferData(
      gl.ARRAY_BUFFER,
      new Float32Array(textureCoordinates),
      gl.STATIC_DRAW,
    );
    return textureCoordBuffer;
  }
function initTetrPositionBuffer(gl) {
    const positionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
    const positions = [
        0.0, 1.0, 0.0,       // Top vertex
        -1.0, -1.0, 1.0,     // Front-left
        1.0, -1.0, 1.0,      // Front-right
        0.0, -1.0, -1.0,     // Back
    ];
    gl.bufferData(
        gl.ARRAY_BUFFER,
        new Float32Array(positions),
        gl.STATIC_DRAW
    );
    return positionBuffer;
}
function initTetrIndexBuffer(gl) {
    const indexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
    const indices = [
        0, 1, 2,  // Front face
        0, 2, 3,  // Right face
        0, 3, 1,  // Left face
        1, 3, 2,  // Bottom face
    ];
    gl.bufferData(
        gl.ELEMENT_ARRAY_BUFFER,
        new Uint16Array(indices),
        gl.STATIC_DRAW
    );
    return indexBuffer;
}
function initTetrTextureBuffer(gl) {
    const textureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, textureCoordBuffer);
    const textureCoordinates = [
        0.5, 1.0,   // Top vertex
        0.0, 0.0,   // Front-left
        1.0, 0.0,   // Front-right
        0.5, 0.0,   // Back
    ];
    gl.bufferData(
        gl.ARRAY_BUFFER,
        new Float32Array(textureCoordinates),
        gl.STATIC_DRAW
    );
    return textureCoordBuffer;
}
```

Fragment kodu 3: Fragment kodu z programu

.

```
function drawScene(gl, programInfo, buffers,rotationX,rotationY,rotationZ,prevX,prevY,prevZ,texture1,texture2,selectedShape) {
    gl.clearColor(0.0, 0.0, 0.0, 1.0); // Clear to black, fully opaque
    gl.clearDepth(1.0); // Clear everything
    gl.enable(gl.DEPTH_TEST); // Enable depth testing
    gl.depthFunc(gl.LEQUAL); // Near things obscure far things
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    const fieldOfView = (45 * Math.PI) / 180; // in radians
    const aspect = gl.canvas.clientWidth / gl.canvas.clientHeight;
    const zNear = 0.1;
    const zFar = 100.0;
    const projectionMatrix = mat4.create();
    mat4.perspective(projectionMatrix, fieldOfView, aspect, zNear, zFar);
    let cubeX;let tetrX;
    let cubeY;let tetrY;
    let cubeZ;let tetrZ;
    if(selectedShape == "cube"){
        cubeX = rotationX;tetrX = prevX;
        cubeY = rotationY;tetrY = prevY;
        cubeZ = rotationZ;tetrZ = prevZ;
    }else if(selectedShape == "tetrahedron"){
        tetrX = rotationX;cubeX = prevX;
        tetrY = rotationY;cubeY = prevY;
        tetrZ = rotationZ;cubeZ = prevZ;
    }
    drawFigure(gl,programInfo,buffers.cube,cubeX,cubeY,cubeZ,texture1,projectionMatrix,30,-2);
    drawFigure(gl,programInfo,buffers.tetr,tetrX,tetrY,tetrZ,texture2,projectionMatrix,12,2);
  }
function drawFigure(gl,programInfo,figure,rotationX,rotationY,rotationZ,texture,projectionMatrix,vertNum,move){
    const modelViewMatrix = mat4.create();
    mat4.translate(
        modelViewMatrix, // destination matrix
        modelViewMatrix, // matrix to translate
        [move, 0.0, -6.0],
    ); // amount to translate
    mat4.rotate(
        modelViewMatrix, // destination matrix
        modelViewMatrix, // matrix to rotate
        rotationZ, // amount to rotate in radians
        [0, 0, 1],
      ); // axis to rotate around (Z)
    mat4.rotate(
        modelViewMatrix, // destination matrix
```

```
            modelViewMatrix, // matrix to rotate
            rotationY , // amount to rotate in radians
            [0, 1, 0],
        ); // axis to rotate around (Y)
    mat4.rotate(
            modelViewMatrix, // destination matrix
            modelViewMatrix, // matrix to rotate
            rotationX , // amount to rotate in radians
            [1, 0, 0],
        );
    setPositionAttribute(gl, figure, programInfo);
    setTextureAttribute(gl, figure, programInfo);
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, figure.indices);
    gl.useProgram(programInfo.program);
    gl.uniformMatrix4fv(
        programInfo.uniformLocations.projectionMatrix,
        false,
        projectionMatrix,
    );
    gl.uniformMatrix4fv(
        programInfo.uniformLocations.modelViewMatrix,
        false,
        modelViewMatrix,
    );
    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.uniform1i(programInfo.uniformLocations.uSampler, 0);
    {
        const vertexCount = vertNum;
        const type = gl.UNSIGNED_SHORT;
        const offset = 0;
        gl.drawElements(gl.TRIANGLES, vertexCount, type, offset);
    }
}
function setPositionAttribute(gl, buffers, programInfo) {
    const numComponents = 3;
    const type = gl.FLOAT;
    const normalize = false;
    const stride = 0;
    const offset = 0;
    gl.bindBuffer(gl.ARRAY_BUFFER, buffers.position);
    gl.vertexAttribPointer(
        programInfo.attribLocations.vertexPosition,
        numComponents,
        type,
        normalize,
        stride,
        offset,
    );
    gl.enableVertexAttribArray(programInfo.attribLocations.vertexPosition);
}
function setTextureAttribute(gl, buffers, programInfo) {
    const num = 2;
    const type = gl.FLOAT;
    const normalize = false;
    const stride = 0;
    const offset = 0;
    gl.bindBuffer(gl.ARRAY_BUFFER, buffers.textureCoord);
    gl.vertexAttribPointer(
      programInfo.attribLocations.textureCoord,
      num,
      type,
      normalize,
      stride,
      offset,
    );
    gl.enableVertexAttribArray(programInfo.attribLocations.textureCoord);
}

```

Fragment kodu 4: Fragment kodu z programu

3 Źródła

1.