Sprawozdanie z ćwiczenia 7
WebGL - podstawy

Maksymilian Iwanow 209946
Poniedziałek 10.15

Celem ćwiczenia było wprowadzenie do grafiki trójwymiarowej opartej na technologii bazującej na elemencie HTML5 canvas. Canvas jest elementem, który może być użyty do rysowania grafik przy użyciu skryptów JavaScript (rysowanie wykresów, tworzenie kompozycji fotografii bądź do animacji).

WebGL to rozszerzenie języka JavaScript, zapewnia dostęp do trójwymiarowego API w przeglądarce internetowej. Aktualnie zaimplementowany jest w większości współczesnych przeglądarek internetowych.

Zadanie polegało na wytworzeniu czworościanu, bazując na kodzie zadanym na stronie internetowej ZSK. Należało też zmodyfikować program tak, aby za każdym kliknięciem przycisku „URUCHOM” prędkość obrotu bryły była jednakowa.

Animacja kolorowego czworościanu

Pierwszą rzeczą, którą należało zrobić było utworzenie nowego projektu. W tym celu stworzono folder o nazwie WebGL_Project. Wewnątrz utworzono plik *index.html* oraz dwa dodatkowe podkatalogi: js i css. W podkatalogu js umiejscowiono skrypt o nazwie *main.js* oraz drugi – *matrix.js*, a w css, plik kaskadowych arkuszy stylów – *style.css*.

Zawartość pliku *index.html*:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Grafika komputerowa - WebGL</title>
  <link rel="stylesheet" href="css/style.css">
  <script src="js/matrix.js"></script>
  <script src="js/main.js"></script>
</head>
<body>
        <div>
                <form class="rotationCheckboxes">
                        <input type="checkbox" id="rotateX"> Rotacja X
                        <input type="checkbox" id="rotateY"> Rotacja Y
                        <input type="checkbox" id="rotateZ"> Rotacja Z
                </form>
                <canvas id="glcanvas" width="500" height="300">Brak wsparcia dla elementu HTML
canvas.</canvas>
                <br />
                <input type="button" value="Uruchom" onclick="runWebGL()"/>
                <input type="button" value="Zamknij" onclick="self.close()"/>
  </div>
</body>
</html>
```

Zawartość pliku w zasadzie nie różni się w żadnym stopniu od wykorzystywanego w przykładzie obracającego się sześcianu. Jest to standardowy szablon dokumentu HTML5 wzbogacony o element canvas, posiada 3 *checkboxy* do wyboru osi rotacji. W sytuacji gdy okaże się, że nasza przeglądarka nie obsługuje elementu canvas, dostaniemy komunikat który umieściliśmy wewnątrz komponentu.

Zawartość pliku *style.css*:

```
canvas#glcanvas {
   border: 1px solid #66666D;
   background-color: #545469;
}
```

W przypadku pliku kaskadowych arkuszy stylów także nie zmieniamy nic - ustawiamy kolor tła elementu canvas oraz dodajemy ramkę aby uwidocznić obszar naszego elementu canvas.

Zawartość pliku *main.js*:

```
var gl_canvas;
var gl_ctx;
var _triangleVertexBuffer;
var _triangleFacesBuffer;
var _position;
var _color;
var _PosMatrix;
var _MovMatrix;
var _ViewMatrix;
var _matrixProjection;
var _matrixMovement;
var _matrixView;
var rotationSpeed = 0.005;
var zoomRatio = -6;

var uruchomiony =0;
var X, Y, Z;

function runWebGL () {
   getRotation();
   gl_canvas = document.getElementById("glcanvas");
   gl_ctx = gl_getContext(gl_canvas);
   gl_initShaders();
   gl_initBuffers();
   gl_setMatrix();
   gl_draw();
   uruchomiony=1;
}

function getRotation() {
   X = document.getElementById('rotateX').checked;
   Y = document.getElementById('rotateY').checked;
   Z = document.getElementById('rotateZ').checked;
}

function gl_getContext (canvas) {
   try {
      var ctx = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
      ctx.viewportWidth = canvas.width;
      ctx.viewportHeight = canvas.height;
   } catch (e) {}

   if (!ctx) {
      document.write('Unable to initialize WebGL. Your browser may not support it.')
   }
   return ctx;
}

// Declare the shaders. They are pieces of code compiled by WebGL and
// executed on the graphics device. They are written in GLSL.
```

```
function gl_initShaders () {
  // position of the point - 0. is Z and 1. is W
  // PosMatrix is uniform variable - its value is constant while rendering an object
  // MovMatrix is the movement matrix of the triangle
  // gl_position -> we move position with MovMatrix before projecting it
  var vertexShader = "\n\
    attribute vec3 position;\n\
    uniform mat4 PosMatrix;\n\
    uniform mat4 MovMatrix;\n\
    uniform mat4 ViewMatrix; \n\
    attribute vec3 color;\n\
    varying vec3 vColor;\n\
    void main(void) {\n\
      gl_Position = PosMatrix * ViewMatrix * MovMatrix * vec4(position, 1.);\n\
      vColor = color;\n\
    }";

  // set black color
  var fragmentShader = "\n\
    precision mediump float;\n\
    varying vec3 vColor;\n\
    void main(void) {\n\
      gl_FragColor = vec4(vColor, 1.);\n\
    }";

  // this function is used to compile a shader
  var getShader = function(source, type, typeString) {
    var shader = gl_ctx.createShader(type);
    gl_ctx.shaderSource(shader, source);
    gl_ctx.compileShader(shader);

    if (!gl_ctx.getShaderParameter(shader, gl_ctx.COMPILE_STATUS)) {
      alert('error in' + typeString);
      return false;
    }
    return shader;
  };

  // Compile the vertex and fragment shaders
  var shader_vertex = getShader(vertexShader, gl_ctx.VERTEX_SHADER, "VERTEX");
  var shader_fragment = getShader(fragmentShader, gl_ctx.FRAGMENT_SHADER, "FRAGMENT");


  // Create the Shader program.
  // Shader program is a combination of a vertex and fragment shaders.
  var SHADER_PROGRAM = gl_ctx.createProgram();
  gl_ctx.attachShader(SHADER_PROGRAM, shader_vertex);
  gl_ctx.attachShader(SHADER_PROGRAM, shader_fragment);


  // Linking of the shader program to the WebGL context - gl_ctx,
  // in order to match the shader variables to javascript variables
  gl_ctx.linkProgram(SHADER_PROGRAM);

  // Link PosMatrix\MovMatrix\ViewMatrix GLSL variables to
  // _PosMatrix\_MovMatrix\_ViewMatrix javascript variables
  // Uniforms do not need to be enabled like attributes
  _PosMatrix = gl_ctx.getUniformLocation(SHADER_PROGRAM, "PosMatrix");
  _MovMatrix = gl_ctx.getUniformLocation(SHADER_PROGRAM, "MovMatrix");
  _ViewMatrix = gl_ctx.getUniformLocation(SHADER_PROGRAM, "ViewMatrix");
```

```
 // position GLSL variable links to _position variable
   _position = gl_ctx.getAttribLocation(SHADER_PROGRAM, "position");    // *******
   // color GLSL variable links to _color variable
   _color = gl_ctx.getAttribLocation(SHADER_PROGRAM, "color");
   // enable GLSL attributes variables
   gl_ctx.enableVertexAttribArray(_position);
   gl_ctx.enableVertexAttribArray(_color);
   // linking is over - tells WebGL context to use SHADER_PROGRAM for rendering.
   gl_ctx.useProgram(SHADER_PROGRAM);
}

function gl_initBuffers () {
   var triangleVertices = [
      1,1,1,
      0,0,1,
      -1,-1,1,
      1,0,0,
      -1,1,-1,
      1,1,0,
      1,-1,-1,
      0,1,0,
   ];


   // Building Vertex Buffer Object - WebGL vertex array
   _triangleVertexBuffer = gl_ctx.createBuffer();              // *******
   gl_ctx.bindBuffer(gl_ctx.ARRAY_BUFFER, _triangleVertexBuffer);
   gl_ctx.bufferData(gl_ctx.ARRAY_BUFFER, new Float32Array(triangleVertices), gl_ctx.STATIC_DRAW);


  // Triangle faces array
   var triangleFaces = [
      0,1,2,
      0,1,3,
      0,2,3,
      1,2,3,
   ];

   _triangleFacesBuffer = gl_ctx.createBuffer();
gl_ctx.bindBuffer(gl_ctx.ELEMENT_ARRAY_BUFFER, _triangleFacesBuffer);
   gl_ctx.bufferData(gl_ctx.ELEMENT_ARRAY_BUFFER, new Uint16Array(triangleFaces),
gl_ctx.STATIC_DRAW);
}

function gl_setMatrix () {
   _matrixProjection = MATRIX.getProjection(40, gl_canvas.width/gl_canvas.height, 1, 100);
   _matrixMovement = MATRIX.getIdentityMatrix();
   _matrixView = MATRIX.getIdentityMatrix();

   MATRIX.translateZ(_matrixView, zoomRatio);
}

function gl_draw() {
   // set the color to transparent
   gl_ctx.clearColor(0.0, 0.0, 0.0, 0.0);
   // enable Depth buffer test and set depth buffer comparison function
   gl_ctx.enable(gl_ctx.DEPTH_TEST);
   gl_ctx.depthFunc(gl_ctx.LEQUAL);

   // set the clear value for the depth buffer to 1
   gl_ctx.clearDepth(1.0);

   var timeOld = 0;
```

```
   var animate = function (time) {
      var dAngle = rotationSpeed *(time - timeOld);

      if (X) {
         MATRIX.rotateX(_matrixMovement, dAngle);
      }
      if (Y) {
         MATRIX.rotateY(_matrixMovement, dAngle);
      }
      if (Z) {
         MATRIX.rotateZ(_matrixMovement, dAngle);
      }

     timeOld = time;

      // set the drawing area on the canvas and clear it
      gl_ctx.viewport(0.0, 0.0, gl_canvas.width, gl_canvas.height);
      gl_ctx.clear(gl_ctx.COLOR_BUFFER_BIT | gl_ctx.DEPTH_BUFFER_BIT);

      // set projection matrix. _matrixProjection is not set yet.
      // It is a javascript array of 1 dimension with 16 floats
      gl_ctx.uniformMatrix4fv(_PosMatrix, false, _matrixProjection);
      gl_ctx.uniformMatrix4fv(_MovMatrix, false, _matrixMovement);
      gl_ctx.uniformMatrix4fv(_ViewMatrix, false, _matrixView);
      // drawing is here - use these points for next drawing
      gl_ctx.vertexAttribPointer(_position, 3, gl_ctx.FLOAT, false, 4*(3+3), 0);
      gl_ctx.vertexAttribPointer(_color, 3, gl_ctx.FLOAT, false, 4*(3+3), 3*4);

      gl_ctx.bindBuffer(gl_ctx.ARRAY_BUFFER, _triangleVertexBuffer);
      gl_ctx.bindBuffer(gl_ctx.ELEMENT_ARRAY_BUFFER, _triangleFacesBuffer);

      // rysuj ostrosłup
      gl_ctx.drawElements(gl_ctx.TRIANGLES, 4*3, gl_ctx.UNSIGNED_SHORT, 0);

      // drawing is finished - show the render
      gl_ctx.flush();

      // redraws the scene as soon as ready
      window.requestAnimationFrame(animate);
   };

   // launch animate for the first time
   if(uruchomiony==0){
   animate(0);}
}
```

Plik *main.js* musiał zostać zmodyfikowany. Przede wszystkim – współrzędne wierzchołków w przestrzeni 3D. W przykładowym programie tworzony był sześcian, więc tablica wierzchołków wyglądała następująco:

```
var triangleVertices = [
-1,-1,-1, // wierzchołek #1
 0, 0, 0, // kolor: czarny
 1,-1,-1, // wierzchołek #2
 1, 0, 0, // kolor: czerwony
 1, 1,-1, // wierzchołek #3
 1, 1, 0, // kolor: zółty
-1, 1,-1, // wierzchołek #4
 0, 1, 0, // kolor: zielony
-1,-1, 1, // wierzchołek #5
 0, 0, 1, // kolor: niebieski
 1,-1, 1, // wierzchołek #6
 1, 0, 1, // kolor: fioletowy
 1, 1, 1, // wierzchołek #7
 1, 1, 1, // kolor: biały
-1, 1, 1, // wierzchołek #8
 0, 1, 1 // kolor: błękitny
];
```

A po modyfikacji:

```
var triangleVertices = [
    1,1,1,
    0,0,1,
    -1,-1,1,
    1,0,0,
    -1,1,-1,
    1,1,0,
    1,-1,-1,
    0,1,0,
  ];
```

Podobny zabieg należało zastosować w przypadku tablicy *triangleFaces[]* przechowującej kombinacje wierzchołków, z których tworzone były trójkąty w przestrzeni 3D.

Należało wprowadzić także funkcję warunkową zapobiegającą przyśpieszaniu obrotowi bryły. W tym celu została stworzona zmienna *uruchomiony* z początkową wartością 0. Przy wywoływaniu funkcji *animate* sprawdzane jest, czy została zmieniona jej wartość. Jeżeli nie – funkcja zostaje wywołana.

Zawartość pliku *matrix.js*:

```
var MATRIX = {
  degToRad: function(angle) {
    return (angle*Math.PI/180);
  },

  getProjection: function(angle, a, zMin, zMax) {
    var tan = Math.tan(MATRIX.degToRad(0.5*angle)),
      A=-(zMax+zMin)/(zMax-zMin),
        B=(-2*zMax*zMin)/(zMax-zMin);

    return [
      .5/tan,         0,   0,    0,
          0,   .5*a/tan,   0,    0,
          0,         0,   A,   -1,
          0,         0,   B,    0
    ]
  },

  getIdentityMatrix: function () {
    return [
      1, 0, 0, 0,
      0, 1, 0, 0,
      0, 0, 1, 0,
      0, 0, 0, 1
    ];
  },

  // rotate movement matrix with angle around X axis
  rotateX: function(movMat, angle) {
    var sin = Math.sin(angle);
    var cos = Math.cos(angle);
    var matElem1 = movMat[1],
        matElem5 = movMat[5],
        matElem9 = movMat[9];

    movMat[1] = movMat[1]*cos - movMat[2]*sin;
    movMat[5] = movMat[5]*cos - movMat[6]*sin;
    movMat[9] = movMat[9]*cos - movMat[10]*sin;

    movMat[2] = movMat[2]*cos + matElem1*sin;
    movMat[6] = movMat[6]*cos + matElem5*sin;
    movMat[10] = movMat[10]*cos + matElem9*sin;
  },
```

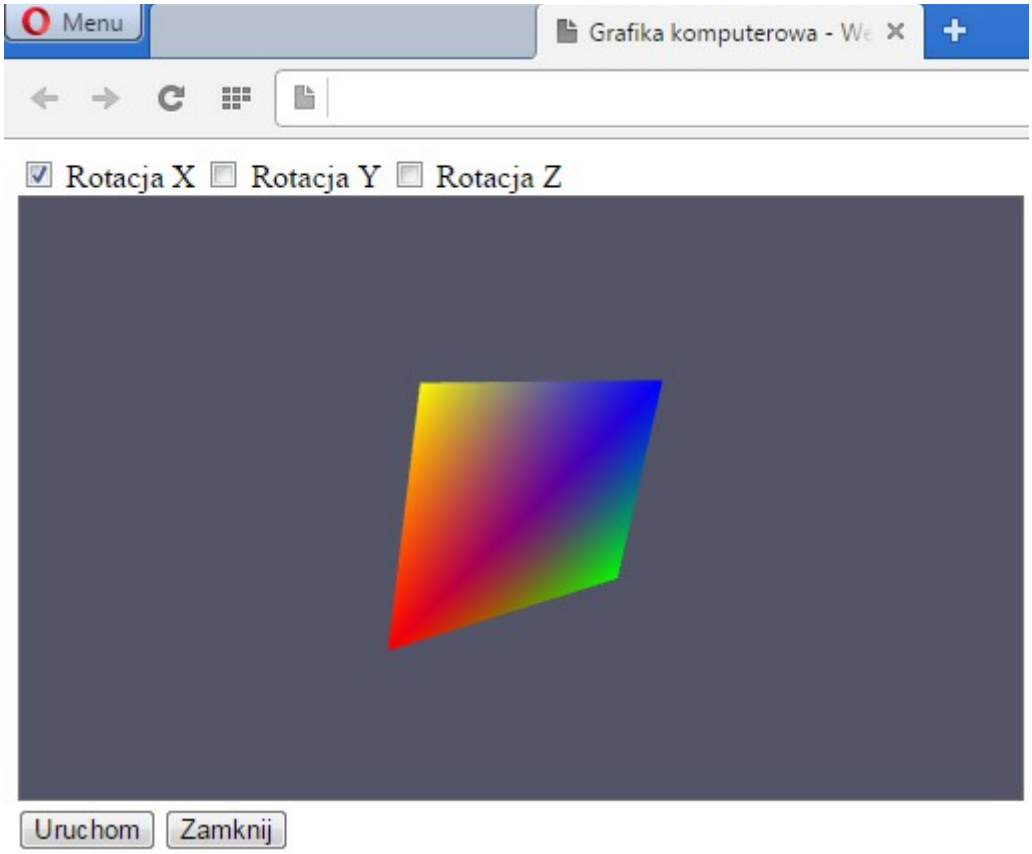
```
  // rotate movement matrix with angle around Y axis
  rotateY: function(movMat, angle) {
    var sin = Math.sin(angle);
    var cos = Math.cos(angle);
    var matElem0 = movMat[0],
        matElem4 = movMat[4],
        matElem8 = movMat[8];

    movMat[0] = movMat[0]*cos + movMat[2]*sin;
    movMat[4] = movMat[4]*cos + movMat[6]*sin;
    movMat[8] = movMat[8]*cos + movMat[10]*sin;

    movMat[2] = movMat[2]*cos - matElem0*sin;
    movMat[6] = movMat[6]*cos - matElem4*sin;
    movMat[10] = movMat[10]*cos - matElem8*sin;
  },
  // rotate movement matrix with angle around Z axis
  rotateZ: function(movMat, angle) {
    var sin = Math.sin(angle);
    var cos = Math.cos(angle);
    var matElem0 = movMat[0],
        matElem4 = movMat[4],
        matElem8 = movMat[8];

    movMat[0] = movMat[0]*cos - movMat[1]*sin;
    movMat[4] = movMat[4]*cos - movMat[5]*sin;
    movMat[8] = movMat[8]*cos - movMat[9]*sin;

    movMat[1] = movMat[1]*cos + matElem0*sin;
    movMat[5] = movMat[5]*cos + matElem4*sin;
    movMat[9] = movMat[9]*cos + matElem8*sin;
  },

  // translate movement matrix by trans along Z axis
  translateZ: function (movMat, trans) {
    movMat[14] += trans;
  }
};
```

W pliku matrix.js zaimplementowane zostały funkcje realizujące m.in. operacje obrotu obiektu. Wszystkie operacje dotyczą macierzy widoku modelu. Aby przesunąc dany wierzchołek, używana jest metoda translateZ(movMat, trans), gdzie drugim parametrem jest wartość dodawana do współrzędnych obiektu. Zawartość matrix.js nie różni się praktycznie od pliku wykorzystywanego w przykładowym programie z obracającym się sześcianem.

Podsumowanie, wnioski

1. Największą trudnością w wykonaniu zadania była zmiana współrzędnych wierzchołków 3D. Postać kanonicznych współrzędnych wierzchołków została zaczerpnięta z encyklopedii Wikipedia.
2. Kolejnym problemem okazało się stworzenie funkcji warunkowej zapobiegającej przyśpieszaniu obrotowi bryły. Powstała ona za pomocą metody *prób i błędów.*

Wynik działania programu



Obraz 1. Wynik działania programu – obracający się czworościan foremny