```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<script type="text/javascript">
var gl;
var shaderProgram;
var uPMatrix;
var vertexPositionBuffer;
var vertexColorBuffer;
function MatrixMul(a,b) //Mnożenie macierzy
{
  c = [
  0,0,0,0,
  0,0,0,0,
  0,0,0,0,
  0,0,0,0
  ]
  for(let i=0;i<4;i++)
  {
    for(let j=0;j<4;j++)
    {
      c[i*4+j] = 0.0;
      for(let k=0;k<4;k++)
      {
        c[i*4+j]+= a[i*4+k] * b[k*4+j];
      }
    }
  }
  return c;
}

function createRect2(p1x,p1y,p1z,p2x,p2y,p2z,p3x,p3y,p3z,p4x,p4y,p4z)
{
  let vertexPosition = [p1x,p1y,p1z, p2x,p2y,p2z, p4x,p4y,p4z,  //Pierwszy trójkąt
                        p1x,p1y,p1z, p4x,p4y,p4z, p3x,p3y,p3z]; //Drugi trójkąt

  return vertexPosition;
}

function createRectCoords(mu,mv,dau,dav,dbu,dbv)
{
  p1u = mu;            p1v = mv;
  p2u = mu + dau;      p2v = mv + dav;
  p3u = mu + dbu;      p3v = mv + dbv;
  p4u = mu + dau + dbu; p4v = mv + dav + dbv;
```

```
  let vertexCoord = [p1u,p1v, p2u,p2v, p4u,p4v,  //Pierwszy trójkąt
                     p1u,p1v, p4u,p4v, p3u,p3v]; //Drugi trójkąt

  return vertexCoord;
}

function createRectCoords2(p1u,p1v,p2u,p2v,p3u,p3v,p4u,p4v)
{
  let vertexCoord = [p1u,p1v, p2u,p2v, p4u,p4v,  //Pierwszy trójkąt
                     p1u,p1v, p4u,p4v, p3u,p3v]; //Drugi trójkąt

  return vertexCoord;
}

function createRectColor(r,g,b)
{
  let vertexColor = [r,g,b, r,g,b, r,g,b,  //Pierwszy trójkąt
                     r,g,b, r,g,b, r,g,b]; //Drugi trójkąt

  return vertexColor;
}

function startGL()
{
  alert("StartGL");
  let canvas = document.getElementById("canvas3D"); //wyszukanie obiektu w strukturze
strony
  gl = canvas.getContext("experimental-webgl"); //pobranie kontekstu OpenGL'u z obiektu
canvas
  gl.viewportWidth = canvas.width; //przypisanie wybranej przez nas rozdzielczości do
systemu OpenGL
  gl.viewportHeight = canvas.height;

  //Kod shaderów
  const vertextShaderSource = ` //Znak akcentu z przycisku tyldy - na lewo od przycisku 1 na
klawiaturze
    precision highp float;
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;
    attribute vec2 aVertexCoords;
    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    varying vec3 vColor;
    varying vec2 vTexUV;
    void main(void) {
```

```
    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0); //Dokonanie transformacji położenia punktów z przestrzeni 3D do przestrzeni obrazu (2D)
    vColor = aVertexColor;
    vTexUV = aVertexCoords;
   }
  `;
  const fragmentShaderSource = `
   precision highp float;
   varying vec3 vColor;
   varying vec2 vTexUV;
   uniform sampler2D uSampler;
   void main(void) {
    //gl_FragColor = vec4(vColor,1.0); //Ustalenie stałego koloru wszystkich punktów sceny
    gl_FragColor = texture2D(uSampler,vTexUV)*vec4(vColor,1.0); //Odczytanie punktu tekstury i przypisanie go jako koloru danego punktu renderowaniej figury
   }
  `;
  let fragmentShader = gl.createShader(gl.FRAGMENT_SHADER); //Stworzenie obiektu shadera
  let vertexShader   = gl.createShader(gl.VERTEX_SHADER);
  gl.shaderSource(fragmentShader, fragmentShaderSource); //Podpięcie źródła kodu shader
  gl.shaderSource(vertexShader, vertextShaderSource);
  gl.compileShader(fragmentShader); //Kompilacja kodu shader
  gl.compileShader(vertexShader);
  if (!gl.getShaderParameter(fragmentShader, gl.COMPILE_STATUS)) { //Sprawdzenie ewentualnych błedów kompilacji
   alert(gl.getShaderInfoLog(fragmentShader));
   return null;
  }
  if (!gl.getShaderParameter(vertexShader, gl.COMPILE_STATUS)) {
   alert(gl.getShaderInfoLog(vertexShader));
   return null;
  }

  shaderProgram = gl.createProgram(); //Stworzenie obiektu programu
  gl.attachShader(shaderProgram, vertexShader); //Podpięcie obu shaderów do naszego programu wykonywanego na karcie graficznej
  gl.attachShader(shaderProgram, fragmentShader);
  gl.linkProgram(shaderProgram);
  if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) alert("Could not initialise shaders");  //Sprawdzenie ewentualnych błedów

  //Opis sceny 3D, położenie punktów w przestrzeni 3D w formacie X,Y,Z
  let vertexPosition = []; //3 punkty po 3 składowe - X1,Y1,Z1, X2,Y2,Z2, X3,Y3,Z3 - 1 trójkąt
  let stepElevation = 90/12;
  let stepAngle = 360/24;
```

```
  let radius = 2.0;
  for(let elevation=0; elevation< 360; elevation+= stepElevation)
  {
    let radiusXZ = 5+radius*Math.cos(elevation*Math.PI/180);
    let radiusY  = radius*Math.sin(elevation*Math.PI/180);

    let radiusXZ2 = 5+radius*Math.cos((elevation+stepElevation)*Math.PI/180);
    let radiusY2  = radius*Math.sin((elevation+stepElevation)*Math.PI/180);

    for(let angle = 0; angle < 360; angle+= stepAngle)
    {

      let px1 = radiusXZ*Math.cos(angle*Math.PI/180);
      let py1 = radiusY;
      let pz1 = radiusXZ*Math.sin(angle*Math.PI/180);

      let px2 = radiusXZ*Math.cos((angle+stepAngle)*Math.PI/180);
      let py2 = radiusY;
      let pz2 = radiusXZ*Math.sin((angle+stepAngle)*Math.PI/180);

      let px3 = radiusXZ2*Math.cos(angle*Math.PI/180);
      let py3 = radiusY2;
      let pz3 = radiusXZ2*Math.sin(angle*Math.PI/180);

      let px4 = radiusXZ2*Math.cos((angle+stepAngle)*Math.PI/180);
      let py4 = radiusY2;
      let pz4 = radiusXZ2*Math.sin((angle+stepAngle)*Math.PI/180);

      vertexPosition.push(...createRect2(px1,py1,pz1,px2,py2,pz2,px3,py3,pz3,px4,py4,pz4));
// Ściana XZ
    }
  }

  vertexPositionBuffer = gl.createBuffer(); //Stworzenie tablicy w pamieci karty graficznej
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexPositionBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexPosition), gl.STATIC_DRAW);
  vertexPositionBuffer.itemSize = 3; //zdefiniowanie liczby współrzednych per wierzchołek
  vertexPositionBuffer.numItems = vertexPosition.length/9; //Zdefiniowanie liczby trójkątów w
naszym buforze

  //Opis sceny 3D, kolor każdego z wierzchołków
  let vertexColor = []; //3 punkty po 3 składowe - R1,G1,B1, R2,G2,B2, R3,G3,B3 - 1 trójkąt
  for(let elevation=135; elevation< 180; elevation+= stepElevation)
  {
    for(let angle = 0; angle < 360; angle+= stepAngle)
    {
```

```
    vertexColor.push(...createRectColor(1.0,0.9,0.7));
  }
}
for(let elevation=0; elevation< 135; elevation+= stepElevation)
{
  for(let angle = 0; angle < 360; angle+= stepAngle)
  {
    vertexColor.push(...createRectColor(1.0,0.4,0.4));
  }
}
for(let elevation=190; elevation< 360; elevation+= stepElevation)
{
  for(let angle = 0; angle < 360; angle+= stepAngle)
  {
    vertexColor.push(...createRectColor(1.0,0.9,0.7));
  }
}for(let elevation=180; elevation< 190; elevation+= stepElevation)
{
  for(let angle = 0; angle < 360; angle+= stepAngle)
  {
    vertexColor.push(...createRectColor(1.0,1.0,1.0));
  }
}
vertexColorBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vertexColorBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexColor), gl.STATIC_DRAW);
vertexColorBuffer.itemSize = 3;
vertexColorBuffer.numItems = vertexColor.length/9;

let vertexCoords = []; //3 punkty po 2 składowe - U1,V1, U2,V2, U3,V3 - 1 trójkąt
for(let elevation=-45; elevation< 0; elevation+= stepElevation)
{
  for(let angle = 0; angle < 360; angle+= stepAngle)
  {
    vertexCoords.push(...createRectCoords(0,0.0,0.0,0,0,0.0));
  }
}
for(let elevation=0; elevation< 135; elevation+= stepElevation)
{
  for(let angle = 0; angle < 360; angle+= stepAngle)
  {
    vertexCoords.push(...createRectCoords(0,1.0,1.0,0,0,1.0));
  }
}

vertexCoordsBuffer = gl.createBuffer();
```

```
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexCoordsBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexCoords), gl.STATIC_DRAW);
  vertexCoordsBuffer.itemSize = 2;
  vertexCoordsBuffer.numItems = vertexCoords.length/6;

  textureBuffer = gl.createTexture();
  var textureImg = new Image();
  textureImg.onload = function() { //Wykonanie kodu automatycznie po załadowaniu obrazka
    gl.bindTexture(gl.TEXTURE_2D, textureBuffer);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE,
textureImg); //Faktyczne załadowanie danych obrazu do pamieci karty graficznej
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
//Ustawienie parametrów próbkowania tekstury
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
  }
  textureImg.src="marshmallow.png"; //Nazwa obrazka

  //Macierze opisujące położenie wirtualnej kamery w przestrzenie 3D
  let aspect = gl.viewportWidth/gl.viewportHeight;
  let fov = 45.0 * Math.PI / 180.0; //Określenie pola widzenia kamery
  let zFar = 100.0; //Ustalenie zakresów renderowania sceny 3D (od obiektu najbliższego
zNear do najdalszego zFar)
  let zNear = 0.1;
  uPMatrix = [
   1.0/(aspect*Math.tan(fov/2)),0                    ,0                    ,0                    ,
   0                    ,1.0/(Math.tan(fov/2))       ,0                    ,0                    ,
   0                    ,0                    ,-(zFar+zNear)/(zFar-zNear)  , -1,
   0                    ,0                    ,-(2*zFar*zNear)/(zFar-zNear) ,0.0,
  ];
  Tick();
}
//let angle = 45.0; //Macierz transformacji świata - określenie położenia kamery
var angleZ = 0.0;
var angleY = 0.0;
var angleX = 0.0;
var tz = -20.0;
function Tick()
{
  let uMVMatrix = [
  1,0,0,0, //Macierz jednostkowa
  0,1,0,0,
  0,0,1,0,
  0,0,0,1
  ];
```

```
 let uMVRotZ = [
 +Math.cos(angleZ*Math.PI/180.0),+Math.sin(angleZ*Math.PI/180.0),0,0,
 -Math.sin(angleZ*Math.PI/180.0),+Math.cos(angleZ*Math.PI/180.0),0,0,
 0,0,1,0,
 0,0,0,1
 ];

 let uMVRotY = [
 +Math.cos(angleY*Math.PI/180.0),0,-Math.sin(angleY*Math.PI/180.0),0,
 0,1,0,0,
 +Math.sin(angleY*Math.PI/180.0),0,+Math.cos(angleY*Math.PI/180.0),0,
 0,0,0,1
 ];

 let uMVRotX = [
 1,0,0,0,
 0,+Math.cos(angleX*Math.PI/180.0),+Math.sin(angleX*Math.PI/180.0),0,
 0,-Math.sin(angleX*Math.PI/180.0),+Math.cos(angleX*Math.PI/180.0),0,
 0,0,0,1
 ];

 let uMVTranslateZ = [
 1,0,0,0,
 0,1,0,0,
 0,0,1,0,
 0,0,tz,1
 ];

 uMVMatrix = MatrixMul(uMVMatrix,uMVRotX);
 uMVMatrix = MatrixMul(uMVMatrix,uMVRotY);
 uMVMatrix = MatrixMul(uMVMatrix,uMVRotZ);
 uMVMatrix = MatrixMul(uMVMatrix,uMVTranslateZ);
 //alert(uPMatrix);

 //Render Scene
 gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
 gl.clearColor(1.0,0.0,0.0,1.0); //Wyczyszczenie obrazu kolorem czerwonym
 gl.clearDepth(1.0);            //Wyczyścienie bufora głebi najdalszym planem
 gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
 gl.useProgram(shaderProgram)   //Użycie przygotowanego programu shaderowego

 gl.enable(gl.DEPTH_TEST);          // Włączenie testu głębi - obiekty bliższe mają
przykrywać obiekty dalsze
 gl.depthFunc(gl.LEQUAL);           //
```

```
  gl.uniformMatrix4fv(gl.getUniformLocation(shaderProgram, "uPMatrix"), false, new
Float32Array(uPMatrix)); //Wgranie macierzy kamery do pamięci karty graficznej
  gl.uniformMatrix4fv(gl.getUniformLocation(shaderProgram, "uMVMatrix"), false, new
Float32Array(uMVMatrix));

  gl.enableVertexAttribArray(gl.getAttribLocation(shaderProgram, "aVertexPosition"));
//Przekazanie położenia
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexPositionBuffer);
  gl.vertexAttribPointer(gl.getAttribLocation(shaderProgram, "aVertexPosition"),
vertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

  gl.enableVertexAttribArray(gl.getAttribLocation(shaderProgram, "aVertexColor"));
//Przekazanie kolorów
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexColorBuffer);
  gl.vertexAttribPointer(gl.getAttribLocation(shaderProgram, "aVertexColor"),
vertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

  gl.enableVertexAttribArray(gl.getAttribLocation(shaderProgram, "aVertexCoords"));  //Pass
the geometry
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexCoordsBuffer);
  gl.vertexAttribPointer(gl.getAttribLocation(shaderProgram, "aVertexCoords"),
vertexCoordsBuffer.itemSize, gl.FLOAT, false, 0, 0);

  gl.activeTexture(gl.TEXTURE0);
  gl.bindTexture(gl.TEXTURE_2D, textureBuffer);
  gl.uniform1i(gl.getUniformLocation(shaderProgram, "uSampler"), 0);

  gl.drawArrays(gl.TRIANGLES, 0,
vertexPositionBuffer.numItems*vertexPositionBuffer.itemSize); //Faktyczne wywołanie
rendrowania

  setTimeout(Tick,100);
}

function handlekeydown(e)
{
 if(e.keyCode==87) angleX=angleX+1.0; //W
 if(e.keyCode==83) angleX=angleX-1.0; //S
 if(e.keyCode==68) angleY=angleY+1.0;
 if(e.keyCode==65) angleY=angleY-1.0;
 if(e.keyCode==81) angleZ=angleZ+1.0;
 if(e.keyCode==69) angleZ=angleZ-1.0;
 //alert(e.keyCode);
 //alert(angleX);
}
</script>
```

```
</head>
<body onload="startGL()" onkeydown="handlekeydown(event)">
<canvas id="canvas3D" width="640" height="480" style="border: solid black 1px"></canvas>
<img id="textureImg" src="Tex.png">
</body>
</html>
```