```
<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;
var vertexCoordsBuffer;
var vertexNormalBuffer;

function MatrixMul(a,b) //Mnożenie macierzy

{
  let 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)
{
  let p1u = mu;             p1v = mv;
```

```
  let p2u = mu + dau;       p2v = mv + dav;
  let p3u = mu + dbu;       p3v = mv + dbv;
  let 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 createNormal(p1x,p1y,p1z,p2x,p2y,p2z,p3x,p3y,p3z) //Wyznaczenie wektora
normalnego dla trójkąta
{
  let v1x = p2x - p1x;
  let v1y = p2y - p1y;
  let v1z = p2z - p1z;

  let v2x = p3x - p1x;
  let v2y = p3y - p1y;
  let v2z = p3z - p1z;

  let v3x =  v1y*v2z - v1z*v2y;
  let v3y =  v1z*v2x - v1x*v2z;
  let v3z =  v1x*v2y - v1y*v2x;

  vl = Math.sqrt(v3x*v3x+v3y*v3y+v3z*v3z); //Obliczenie długości wektora

  v3x/=vl; //Normalizacja na zakreś -1 1
  v3y/=vl;
  v3z/=vl;
```

```
  let vertexNormal = [v3x,v3y,v3z, v3x,v3y,v3z, v3x,v3y,v3z];
  return vertexNormal;
}

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;
    attribute vec3 aVertexNormal;
    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    varying vec3 vPos;
    varying vec3 vColor;
    varying vec2 vTexUV;
    varying vec3 vNormal;
    void main(void) {
      gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0); //Dokonanie transformacji położenia punktów z przestrzeni 3D do przestrzeni obrazu (2D)
      vPos = aVertexPosition;
      vColor = aVertexColor;
      vTexUV = aVertexCoords;
      vNormal = aVertexNormal;
    }
  `;
  const fragmentShaderSource = `
    precision highp float;
    varying vec3 vPos;
    varying vec3 vColor;
    varying vec2 vTexUV;
    varying vec3 vNormal;
    uniform sampler2D uSampler;
```

```
    uniform vec3 uLightPosition;
    void main(void) {
       vec3 lightDirection = normalize(uLightPosition - vPos);
       float brightness = max(dot(vNormal,lightDirection), 0.0);
      //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
      //gl_FragColor = vec4((vNormal+vec3(1.0,1.0,1.0))/2.0,1.0);
      gl_FragColor = clamp(texture2D(uSampler,vTexUV) *
vec4(brightness,brightness,brightness,1.0),0.0,1.0);
    }
  `;
  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 vertexNormal = [];
  let stepElevation = 90/6;
  let stepAngle = 360/20;

  let radius = [100.0, 1.0,    2.0,   0.3, 2.0, 1.0, 25.0, 20.0, 8.0, 8.0];
  let distance= [10.0, 110.0, 120.0, 130.0,132.0, 140.0, 230.0, 290.0, 390.0, 490.0];
```

```
for(let i=0;i<10;i++){
  for(let elevation=-90; elevation< 90; elevation+= stepElevation)
    {
    let radiusXZ = radius[i]*Math.cos(elevation*Math.PI/180);
    let radiusY  = radius[i]*Math.sin(elevation*Math.PI/180);

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



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

        let px1 = distance[i]+radiusXZ*Math.cos(angle*Math.PI/180);
        let py1 = radiusY;
        let pz1 = distance[i]+radiusXZ*Math.sin(angle*Math.PI/180);

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

        let px3 = distance[i]+radiusXZ2*Math.cos(angle*Math.PI/180);
        let py3 = radiusY2;
        let pz3 = distance[i]+radiusXZ2*Math.sin(angle*Math.PI/180);

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


        px1 -= distance[i];
        py1 -= 0;
        pz1 -= distance[i];

        px2 -= distance[i];
        py2 -= 0;
        pz2 -= distance[i];
```

```
      px3 -= distance[i];
      py3 -= 0;
      pz3 -= distance[i];

      px4 -= distance[i];
      py4 -= 0;
      pz4 -= distance[i];

            let p1 = Math.sqrt(px1*px1+py1*py1+pz1*pz1);
      let p2 = Math.sqrt(px2*px2+py2*py2+pz2*pz2);
      let p3 = Math.sqrt(px3*px3+py3*py3+pz3*pz3);
      let p4 = Math.sqrt(px4*px4+py4*py4+pz4*pz4);

        if(i==0)
        {
         p1 *= -1;
         p2 *= -1;
         p3 *= -1;
         p4 *= -1;
        }

     px1 /= p1;
     py1 /= p1;
     pz1 /= p1;

     px2 /= p2;
     py2 /= p2;
     pz2 /= p2;

     px3 /= p3;
     py3 /= p3;
     pz3 /= p3;

     px4 /= p4;
     py4 /= p4;
     pz4 /= p4;


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

  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/8; //Zdefinoiowanie 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 i=0; i<10; i++){
  for(let elevation=-90; elevation< 90; elevation+= stepElevation)
  {
    for(let angle = 0; angle < 360; angle+= stepAngle)
    {
      vertexColor.push(...createRectColor(0.0,1.0,1.0));
    }
  }
  }
  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/8;

  let vertexCoords = []; //3 punkty po 2 składowe - U1,V1, U2,V2, U3,V3 - 1 trójkąt



    for(let elevation=-90; elevation< 90; elevation+= stepElevation)
    {
      for(let angle = 0; angle < 360; angle+= stepAngle)
      {
        vertexCoords.push(...createRectCoords2(0.8, 0.0, 0.9, 0.0, 0.8, 1.0, 0.9, 1.0));
      }
    }


    for(let elevation=-90; elevation< 90; elevation+= stepElevation)
    {
      for(let angle = 0; angle < 360; angle+= stepAngle)
      {
        vertexCoords.push(...createRectCoords2(0.0, 0.0, 0.1, 0.0, 0.0, 1.0, 0.1, 1.0));
      }
    }


    for(let elevation=-90; elevation< 90; elevation+= stepElevation)
    {
```

```
  for(let angle = 0; angle < 360; angle+= stepAngle)
  {
    vertexCoords.push(...createRectCoords2(0.1, 0.0, 0.2, 0.0, 0.1, 1.0, 0.2, 1.0));
  }
}


for(let elevation=-90; elevation< 90; elevation+= stepElevation)
{
  for(let angle = 0; angle < 360; angle+= stepAngle)
  {
    vertexCoords.push(...createRectCoords2(0.9, 0.0, 1.0, 0.0, 0.9, 1.0, 1.0, 1.0));
  }
}


for(let elevation=-90; elevation< 90; elevation+= stepElevation)
{
  for(let angle = 0; angle < 360; angle+= stepAngle)
  {
    vertexCoords.push(...createRectCoords2(0.2, 0.0, 0.3, 0.0, 0.2, 1.0, 0.3, 1.0));
  }
}


for(let elevation=-90; elevation< 90; elevation+= 90)
{
  for(let angle = 0; angle < 360; angle+= 180)
  {
    vertexCoords.push(...createRectCoords2(0.3, 0.0, 0.4, 0.0, 0.3, 1.0, 0.4, 1.0));
  }
}


for(let elevation=-90; elevation< 90; elevation+= stepElevation)
{
  for(let angle = 0; angle < 360; angle+= stepAngle)
  {
    vertexCoords.push(...createRectCoords2(0.3, 0.0, 0.4, 0.0, 0.3, 1.0, 0.4, 1.0));
  }
}


for(let elevation=-90; elevation< 90; elevation+= stepElevation)
{
  for(let angle = 0; angle < 360; angle+= stepAngle)
```

```
    {
      vertexCoords.push(...createRectCoords2(0.4, 0.0, 0.5, 0.0, 0.4, 1.0, 0.5, 1.0));
    }
   }


   for(let elevation=-90; elevation< 90; elevation+= stepElevation)
   {
    for(let angle = 0; angle < 360; angle+= stepAngle)
    {
      vertexCoords.push(...createRectCoords2(0.5, 0.0, 0.6, 0.0, 0.5, 1.0, 0.6, 1.0));
    }
   }


   for(let elevation=-90; elevation< 90; elevation+= stepElevation)
   {
    for(let angle = 0; angle < 360; angle+= stepAngle)
    {
      vertexCoords.push(...createRectCoords2(0.6,0.0,0.7,1.0,0.6,0.0,0.7,1.0));


//vertexCoords.push(...createRectCoords(x/mapSizeX,y/mapSizeY,1.0/mapSizeX,0,0,1.0/ma
pSizeY));
    }
   }
  //}

  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/8;

  /*
  for(let i=0;i<vertexPosition.length/9; i++)
  {

vertexNormal.push(...createNormal(vertexPosition[i*9+0],vertexPosition[i*9+1],vertexPosition
[i*9+2],
                                  vertexPosition[i*9+3],vertexPosition[i*9+4],vertexPosition[i*9+5],
                                  vertexPosition[i*9+6],vertexPosition[i*9+7],vertexPosition[i*9+8]));
  }//*/
```

```
  vertexNormalBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexNormalBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexNormal), gl.STATIC_DRAW);
  vertexNormalBuffer.itemSize = 3;
  vertexNormalBuffer.numItems = vertexNormal.length/9;


  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="planety.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 = 600.0; //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 = 92.0;
var angleX = 0.0;

var lightX = 6;
var lightY = 0;
var lightZ = 6;
var tz = -150.0;
let tx = 100.0;
 let ty =0.0;
```

```
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
  ];

    let uMVTranslateX = [
  1,0,0,0,
  0,1,0,0,
  0,0,1,0,
  tx,0,0,1
  ];
  let uMVTranslateY = [
  1,0,0,0,
  0,1,0,0,
  0,0,1,0,
```

```
  0,ty,0,1
  ];

   uMVMatrix = MatrixMul(uMVMatrix,uMVTranslateX);
   uMVMatrix = MatrixMul(uMVMatrix,uMVTranslateY);
   uMVMatrix = MatrixMul(uMVMatrix,uMVTranslateZ);
  uMVMatrix = MatrixMul(uMVMatrix,uMVRotX);
  uMVMatrix = MatrixMul(uMVMatrix,uMVRotY);
  uMVMatrix = MatrixMul(uMVMatrix,uMVRotZ);


  //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.enableVertexAttribArray(gl.getAttribLocation(shaderProgram, "aVertexNormal"));
//Przekazywanie wektorów normalnych
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexNormalBuffer);
  gl.vertexAttribPointer(gl.getAttribLocation(shaderProgram, "aVertexNormal"),
vertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);

  gl.uniform3f(gl.getUniformLocation(shaderProgram, "uLightPosition"),lightX,lightY,lightZ);

  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)
{
 // Q W E A S D
 if(e.keyCode==87) tx=tx+5.0; //W
 if(e.keyCode==83) tx=tx-5.0; //S
 if(e.keyCode==68) angleY=angleY+1.0;
 if(e.keyCode==65) angleY=angleY-1.0;
 if(e.keyCode==81) tz=tz+5.0;
 if(e.keyCode==69) tz=tz-5.0;
 //alert(e.keyCode);
 //alert(angleX);

 //U I O J K L
 if(e.keyCode==76) lightX=lightX+0.1;
 if(e.keyCode==74) lightX=lightX-0.1;
 if(e.keyCode==73) lightY=lightY+0.1;
 if(e.keyCode==75) lightY=lightY-0.1;
 if(e.keyCode==85) lightZ=lightZ+0.1;
 if(e.keyCode==79) lightZ=lightZ-0.1;
}
</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>
```