```
<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
{
 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 MatrixTransposeInverse(m)
{
  let r = [
      0, 0, 0, 0,
      0, 0, 0, 0,
      0, 0, 0, 0,
      0, 0, 0, 0
   ];
  r[0] = m[5]*m[10]*m[15] - m[5]*m[14]*m[11] - m[6]*m[9]*m[15] + m[6]*m[13]*m[11] +
m[7]*m[9]*m[14] - m[7]*m[13]*m[10];
  r[1] = -m[1]*m[10]*m[15] + m[1]*m[14]*m[11] + m[2]*m[9]*m[15] - m[2]*m[13]*m[11] -
m[3]*m[9]*m[14] + m[3]*m[13]*m[10];
  r[2] = m[1]*m[6]*m[15] - m[1]*m[14]*m[7] - m[2]*m[5]*m[15] + m[2]*m[13]*m[7] +
m[3]*m[5]*m[14] - m[3]*m[13]*m[6];
  r[3] = -m[1]*m[6]*m[11] + m[1]*m[10]*m[7] + m[2]*m[5]*m[11] - m[2]*m[9]*m[7] -
m[3]*m[5]*m[10] + m[3]*m[9]*m[6];
  r[4] = -m[4]*m[10]*m[15] + m[4]*m[14]*m[11] + m[6]*m[8]*m[15] - m[6]*m[12]*m[11] -
m[7]*m[8]*m[14] + m[7]*m[12]*m[10];
  r[5] = m[0]*m[10]*m[15] - m[0]*m[14]*m[11] - m[2]*m[8]*m[15] + m[2]*m[12]*m[11] +
m[3]*m[8]*m[14] - m[3]*m[12]*m[10];
  r[6] = -m[0]*m[6]*m[15] + m[0]*m[14]*m[7] + m[2]*m[4]*m[15] - m[2]*m[12]*m[7] -
m[3]*m[4]*m[14] + m[3]*m[12]*m[6];
  r[7] = m[0]*m[6]*m[11] - m[0]*m[10]*m[7] - m[2]*m[4]*m[11] + m[2]*m[8]*m[7] +
m[3]*m[4]*m[10] - m[3]*m[8]*m[6];
   r[8] = m[4]*m[9]*m[15] - m[4]*m[13]*m[11] - m[5]*m[8]*m[15] + m[5]*m[12]*m[11] +
m[7]*m[8]*m[13] - m[7]*m[12]*m[9];
  r[9] = -m[0]*m[9]*m[15] + m[0]*m[13]*m[11] + m[1]*m[8]*m[15] - m[1]*m[12]*m[11] -
m[3]*m[8]*m[13] + m[3]*m[12]*m[9];
  r[10] = m[0]*m[5]*m[15] - m[0]*m[13]*m[7] - m[1]*m[4]*m[15] + m[1]*m[12]*m[7] +
m[3]*m[4]*m[13] - m[3]*m[12]*m[5];
  r[11] = -m[0]*m[5]*m[11] + m[0]*m[9]*m[7] + m[1]*m[4]*m[11] - m[1]*m[8]*m[7] -
m[3]*m[4]*m[9] + m[3]*m[8]*m[5];
   r[12] = -m[4]*m[9]*m[14] + m[4]*m[13]*m[10] + m[5]*m[8]*m[14] - m[5]*m[12]*m[10] -
m[6]*m[8]*m[13] + m[6]*m[12]*m[9];
  r[13] = m[0]*m[9]*m[14] - m[0]*m[13]*m[10] - m[1]*m[8]*m[14] + m[1]*m[12]*m[10] +
m[2]*m[8]*m[13] - m[2]*m[12]*m[9];
```

```
  r[14] = -m[0]*m[5]*m[14] + m[0]*m[13]*m[6] + m[1]*m[4]*m[14] - m[1]*m[12]*m[6] - m[2]*m[4]*m[13] + m[2]*m[12]*m[5];
  r[15] = m[0]*m[5]*m[10] - m[0]*m[9]*m[6] - m[1]*m[4]*m[10] + m[1]*m[8]*m[6] + m[2]*m[4]*m[9] - m[2]*m[8]*m[5];

  var det = m[0]*r[0] + m[1]*r[4] + m[2]*r[8] + m[3]*r[12];
  for (var i = 0; i < 16; i++) r[i] /= det;

  let rt = [ r[0], r[4], r[8], r[12],
            r[1], r[5], r[9], r[13],
            r[2], r[6], r[10], r[14],
            r[3], r[7], r[11], r[15]
            ];

  return rt;
}

function CreateIdentytyMatrix()
{
  return [
  1,0,0,0, //Macierz jednostkowa
  0,1,0,0,
  0,0,1,0,
  0,0,0,1
  ];
}
function CreateTranslationMatrix(tx,ty,tz)
{
  return  [
  1,0,0,0,
  0,1,0,0,
  0,0,1,0,
  tx,ty,tz,1
  ];
}
function CreateScaleMatrix(sx,sy,sz)
{
  return [
  sx,0,0,0,
  0,sy,0,0,
  0,0,sz,0,
  0,0,0,1
  ];
}
function CreateRotationZMatrix(angleZ)
{
  return [
  +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
  ];
}
function CreateRotationYMatrix(angleY)
{
  return [
  +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
  ];
}
function CreateRotationXMatrix(angleX)
```

```
{
  return [
  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
  ];
}
function CreatePointCloud(sx,ex,sy,ey,sz,ez,svx,evx,svy,evy,svz,evz,sa,ea,n)
{
  let vertexPos   = [];
  let vertexVelocity = [];
  let vertexColor = [];
  let vertexAge = [];
  for(let i=0;i<n;i++)
  {
    //Position randomization
    let px = sx + Math.random()*(ex-sx);
    let py = sy + Math.random()*(ey-sy);
    let pz = sz + Math.random()*(ez-sz);
    //Velocity randomization
    let vx = svx + Math.random()*(evx-svx);
    let vy = svy + Math.random()*(evy-svy);
    let vz = svz + Math.random()*(evz-svz);

    let age = sa + Math.random()*(ea-sa);

    vertexPos.push(...[px,py,pz]);
    vertexVelocity.push(...[vx,vy,vz]);
    vertexColor.push(...[1.0,1.0,1.0]);
    vertexAge.push(age);
  }
  return [vertexPos,vertexVelocity,vertexColor,vertexAge];
}

function createRect(mx,my,mz,dax,day,daz,dbx,dby,dbz)
{
  p1x = mx;            p1y = my;            p1z = mz;
  p2x = mx + dax;      p2y = my + day;      p2z = mz + daz;
  p3x = mx + dbx;      p3y = my + dby;      p3z = mz + dbz;
  p4x = mx + dax + dbx; p4y = my + day + dby; p4z = mz + daz + dbz;

  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 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 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 CreateWood(x,y,z,dx,dy,dz)
{
  //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 vertexColor = [];
  let vertexCoords = [];

  vertexPosition.push(...createRect(-0.2+x,-1.5+y,-0.2+z, 0.4,0.0,0.0, 0.0,3.0,0.0)); // Ściana XY
  vertexPosition.push(...createRect(-0.2+x,-1.5+y,-0.2+z, 0.4,0.0,0.0, 0.0,0.0,0.4)); // Ściana XZ
  vertexPosition.push(...createRect(-0.2+x,-1.5+y,-0.2+z, 0.0,3.0,0.0, 0.0,0.0,0.4)); // Ściana YZ

  vertexPosition.push(...createRect(0.2+x,1.5+y,0.2+z, -0.4, 0.0,0.0, 0.0,-3.0, 0.0)); // Ściana XY
  vertexPosition.push(...createRect(0.2+x,1.5+y,0.2+z, -0.4, 0.0,0.0, 0.0, 0.0,-0.4)); // Ściana XZ
  vertexPosition.push(...createRect(0.2+x,1.5+y,0.2+z,  0.0,-3.0,0.0, 0.0, 0.0,-0.4)); // Ściana YZ

  for(let i=0; i<vertexPosition.length; i+=3){
          vertexColor.push(...[1.0, 1.0, 1.0]);
  }

  vertexCoords.push(...createRectCoords(0,0,1,0,0,1));
  vertexCoords.push(...createRectCoords(0,0,1,0,0,1));
  vertexCoords.push(...createRectCoords(0,0,1,0,0,1));

  vertexCoords.push(...createRectCoords(0,0,1,0,0,1));
  vertexCoords.push(...createRectCoords(0,0,1,0,0,1));
  vertexCoords.push(...createRectCoords(0,0,1,0,0,1));

  return [vertexPosition, vertexColor, vertexCoords];
}

//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 vertexVelocity;
let vertexColor;
let vertexAge;
let vertexPosition2;
let vertexColor2;
let vertexCoords;
let sx = -1.0; //Położenie cząsteczek
let ex =  1.0;
let sy = -1.0;
let ey =  2.0;
let sz = -1.0;
let ez =  1.0;
let svx = -2.0; //Predkości cząsteczek
let evx =  2.0;
let svy = -2.0;
let evy =  2.0;
let svz = -2.0;
let evz =  2.0;
let sa = 0.1; //Czas życia nowych cząsteczek
let ea = 0.5;
function updatePointCloud(vertexPosition,vertexVelocity,vertexColor,vertexAge,n,dt,
sx,ex,sy,ey,sz,ez,svx,evx,svy,evy,svz,evz,sa,ea)
{
  for(let i=0;i<n;i++)
  {
    vertexPosition[i*3+0] = vertexPosition[i*3+0] + vertexVelocity[i*3+0]*dt;
    vertexPosition[i*3+1] = vertexPosition[i*3+1] + vertexVelocity[i*3+1]*dt;
    vertexPosition[i*3+2] = vertexPosition[i*3+2] + vertexVelocity[i*3+2]*dt;

    vertexColor[i*3+0] = (vertexAge[i]/ea)*8;
            vertexColor[i*3+1] = vertexAge[i]/ea;
    vertexColor[i*3+2] = 0;

    vertexAge[i] = vertexAge[i] - dt;
    if(vertexAge[i]<0)
    {
      vertexPosition[i*3+0] = sx + Math.random()*(ex-sx);
      vertexPosition[i*3+1] = sy + Math.random()*(ey-sy);
      vertexPosition[i*3+2] = sz + Math.random()*(ez-sz);
      //Velocity randomization
      vertexVelocity[i*3+0] = svx + Math.random()*(evx-svx);
      vertexVelocity[i*3+1] = svy + Math.random()*(evy-svy);
      vertexVelocity[i*3+2] = svz + Math.random()*(evz-svz);

      vertexAge[i] = sa + Math.random()*(ea-sa);
    }
  }
  return [vertexPosition,vertexVelocity,vertexColor,vertexAge];
}
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 uMMatrix;
    uniform mat4 uVMatrix;
    uniform mat4 uPMatrix;
              uniform mat4 uInvMMatrix;
    varying vec3 vPos;
    varying vec3 vColor;
              varying vec2 vTexUV;
    void main(void) {
      vPos = vec3(uMMatrix * vec4(aVertexPosition, 1.0));
      gl_Position = uPMatrix * uVMatrix * vec4(vPos,1.0); //Dokonanie transformacji położenia punktów z przestrzeni 3D do przestrzeni obrazu (2D)
      vColor = aVertexColor;
                vTexUV = aVertexCoords;
                gl_PointSize = 5.0;
    }
  `;
  const fragmentShaderSource = `
    precision highp float;
    varying vec3 vPos;
    varying vec3 vColor;
              varying vec2 vTexUV;
    uniform sampler2D uSampler;
    uniform float EnableWoodTexture;


    void main(void) {
      if(EnableWoodTexture == 1.0){
                            gl_FragColor = texture2D(uSampler,vTexUV)*vec4(vColor,1.0);
                }
    else{
                            gl_FragColor = vec4(vColor,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

  //[vertexPosition, vertexColor, vertexCoords, vertexNormal] = CreateShpere(0,0,0,2, 6, 12);
  [vertexPosition, vertexVelocity, vertexColor, vertexAge] = CreatePointCloud(sx,ex,sy,ey,sz,ez,svx,evx,svy,evy,svz,evz,sa,ea,1500);

  //console.log(vertexCoords);
  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; //Zdefinoiowanie liczby punktów w naszym buforze

  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;

  [vertexPosition2, vertexColor2, vertexCoords] = CreateWood(0,0,0,0,0,0);

  vertexPositionBuffer2 = gl.createBuffer(); //Stworzenie tablicy w pamieci karty graficznej
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexPositionBuffer2);
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexPosition2), gl.STATIC_DRAW);
  vertexPositionBuffer2.itemSize = 3; //zdefiniowanie liczby współrzednych per wierzchołek
  vertexPositionBuffer2.numItems = vertexPosition2.length/9; //Zdefinoiowanie liczby punktów w naszym buforze

  vertexColorBuffer2 = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexColorBuffer2);
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexColor2), gl.STATIC_DRAW);
  vertexColorBuffer2.itemSize = 3;
  vertexColorBuffer2.numItems = vertexColor2.length/9;

  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="texture.jpg"; //Nazwa obrazka


  console.log(vertexPosition);

  //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 KameraPositionX =  0.0;
var KameraPositionY =  0.0;
var KameraPositionZ = -8.0;
var CloudPositionX = 0.0;
var CloudPositionY = 0.0;
var CloudPositionZ = 0.0;

var Object2PositionX = 0.0;
var Object2PositionY = 0.0;
var Object2PositionZ = 0.0;
var Object2AngleZ = 40.0;

var Object3PositionX = 0.0;
var Object3PositionY = 0.0;
var Object3PositionZ = 0.0;
var Object3AngleX = 90.0;
var Object3AngleX2 = -50.0;

var Object4PositionX = 0.0;
var Object4PositionY = 0.2;
var Object4PositionZ = 0.0;
var Object4AngleX = 180.0;

var Object5PositionX = 0.0;
var Object5PositionY = 0.0;
var Object5PositionZ = 0.0;
var Object5AngleZ = -40.0;

var Object6PositionX = 0.0;
var Object6PositionY = 0.0;
var Object6PositionZ = 0.0;
var Object6AngleX = 90.0;
var Object6AngleX2 = 50.0;

function Tick()
{
  [vertexPosition, vertexVelocity, vertexColor, vertexAge] = updatePointCloud(vertexPosition,vertexVelocity,vertexColor,vertexAge,1500,0.01, sx,ex,sy,ey,sz,ez,svx,evx,svy,evy,svz,evz,sa,ea);
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexPositionBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexPosition), gl.STATIC_DRAW);

  gl.bindBuffer(gl.ARRAY_BUFFER, vertexColorBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexColor), gl.STATIC_DRAW);
  let uMMatrix0 = CreateIdentytyMatrix();
  let uMMatrix1 = CreateIdentytyMatrix();
  let uMMatrix2 = CreateIdentytyMatrix();
  let uMMatrix3 = CreateIdentytyMatrix();
  let uMMatrix4 = CreateIdentytyMatrix();
  let uMMatrix5 = CreateIdentytyMatrix();
  let uMMatrix6 = CreateIdentytyMatrix();

  let uVMatrix = CreateIdentytyMatrix();

  uVMatrix = MatrixMul(uVMatrix,CreateRotationXMatrix(angleX));
  uVMatrix = MatrixMul(uVMatrix,CreateRotationYMatrix(angleY));
  uVMatrix = MatrixMul(uVMatrix,CreateRotationZMatrix(angleZ));
  uVMatrix = MatrixMul(uVMatrix,CreateTranslationMatrix(KameraPositionX,KameraPositionY,KameraPositionZ));
  uMMatrix1 = MatrixMul(uMMatrix1,CreateTranslationMatrix(CloudPositionX,CloudPositionY,CloudPositionZ));
  uMMatrix2 = MatrixMul(uMMatrix2,CreateRotationZMatrix(Object2AngleZ));
  uMMatrix2 = MatrixMul(uMMatrix2,CreateTranslationMatrix(Object2PositionX,Object2PositionY,Object2PositionZ));

  uMMatrix3 = MatrixMul(uMMatrix3,CreateRotationXMatrix(Object3AngleX));
  uMMatrix3 = MatrixMul(uMMatrix3,CreateRotationXMatrix(Object3AngleX2));
  uMMatrix3 = MatrixMul(uMMatrix3,CreateTranslationMatrix(Object3PositionX,Object3PositionY,Object3PositionZ));

  uMMatrix4 = MatrixMul(uMMatrix4,CreateRotationXMatrix(Object4AngleX));
  uMMatrix4 = MatrixMul(uMMatrix4,CreateTranslationMatrix(Object4PositionX,Object4PositionY,Object4PositionZ));

  uMMatrix5 = MatrixMul(uMMatrix5,CreateRotationZMatrix(Object5AngleZ));
  uMMatrix5 = MatrixMul(uMMatrix5,CreateTranslationMatrix(Object5PositionX,Object5PositionY,Object5PositionZ));

  uMMatrix6 = MatrixMul(uMMatrix6,CreateRotationXMatrix(Object6AngleX));
  uMMatrix6 = MatrixMul(uMMatrix6,CreateRotationXMatrix(Object6AngleX2));
  uMMatrix6 = MatrixMul(uMMatrix6,CreateTranslationMatrix(Object6PositionX,Object6PositionY,Object6PositionZ));

  //alert(uPMatrix);

  //Render Scene
  gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
  gl.clearColor(0.0,0.0,0.0,1.0); //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, "uVMatrix"), false, new Float32Array(uVMatrix));
  gl.uniformMatrix4fv(gl.getUniformLocation(shaderProgram, "uMMatrix"), false, new Float32Array(uMMatrix1));

  // CLOUD POINTS
  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"));  //Przekazywanie wektorów colorów
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexColorBuffer);
  gl.vertexAttribPointer(gl.getAttribLocation(shaderProgram, "aVertexColor"), vertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

  gl.uniform1f(gl.getUniformLocation(shaderProgram, "EnableWoodTexture"),0.0);
  gl.drawArrays(gl.POINTS, 0, vertexPositionBuffer.numItems*vertexPositionBuffer.itemSize); //Faktyczne wywołanie rendrowania

   // WOOD

  gl.activeTexture(gl.TEXTURE0);
  gl.bindTexture(gl.TEXTURE_2D, textureBuffer);
  gl.uniform1i(gl.getUniformLocation(shaderProgram, "uSampler"), 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, "aVertexPosition"));  //Przekazanie położenia
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexPositionBuffer2);
  gl.vertexAttribPointer(gl.getAttribLocation(shaderProgram, "aVertexPosition"), vertexPositionBuffer2.itemSize, gl.FLOAT, false, 0, 0);

  gl.enableVertexAttribArray(gl.getAttribLocation(shaderProgram, "aVertexColor"));  //Przekazywanie wektorów colorów
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexColorBuffer2);
  gl.vertexAttribPointer(gl.getAttribLocation(shaderProgram, "aVertexColor"), vertexColorBuffer2.itemSize, gl.FLOAT, false, 0, 0);

  gl.uniformMatrix4fv(gl.getUniformLocation(shaderProgram, "uMMatrix"), false, new Float32Array(uMMatrix2));
  gl.uniformMatrix4fv(gl.getUniformLocation(shaderProgram, "uInvMMatrix"), false, new Float32Array(MatrixTransposeInverse(uMMatrix2)));
  gl.uniform1f(gl.getUniformLocation(shaderProgram, "EnableWoodTexture"),1.0);
  gl.drawArrays(gl.TRIANGLES, 0, vertexPositionBuffer2.numItems*vertexPositionBuffer2.itemSize); //Faktyczne wywołanie rendrowania

  gl.uniformMatrix4fv(gl.getUniformLocation(shaderProgram, "uMMatrix"), false, new Float32Array(uMMatrix3));
  gl.uniformMatrix4fv(gl.getUniformLocation(shaderProgram, "uInvMMatrix"), false, new Float32Array(MatrixTransposeInverse(uMMatrix3)));
  gl.uniform1f(gl.getUniformLocation(shaderProgram, "EnableWoodTexture"),1.0);
  gl.drawArrays(gl.TRIANGLES, 0, vertexPositionBuffer2.numItems*vertexPositionBuffer2.itemSize); //Faktyczne wywołanie rendrowania
  gl.uniformMatrix4fv(gl.getUniformLocation(shaderProgram, "uMMatrix"), false, new Float32Array(uMMatrix4));

  gl.uniformMatrix4fv(gl.getUniformLocation(shaderProgram, "uInvMMatrix"), false, new Float32Array(MatrixTransposeInverse(uMMatrix4)));

  gl.uniform1f(gl.getUniformLocation(shaderProgram, "EnableWoodTexture"),1.0);

  gl.drawArrays(gl.TRIANGLES, 0, vertexPositionBuffer2.numItems*vertexPositionBuffer2.itemSize); //Faktyczne wywołanie rendrowania


  gl.uniformMatrix4fv(gl.getUniformLocation(shaderProgram, "uMMatrix"), false, new Float32Array(uMMatrix5));

  gl.uniformMatrix4fv(gl.getUniformLocation(shaderProgram, "uInvMMatrix"), false, new Float32Array(MatrixTransposeInverse(uMMatrix5)));

  gl.uniform1f(gl.getUniformLocation(shaderProgram, "EnableWoodTexture"),1.0);

  gl.drawArrays(gl.TRIANGLES, 0, vertexPositionBuffer2.numItems*vertexPositionBuffer2.itemSize); //Faktyczne wywołanie rendrowania


  gl.uniformMatrix4fv(gl.getUniformLocation(shaderProgram, "uMMatrix"), false, new Float32Array(uMMatrix6));

  gl.uniformMatrix4fv(gl.getUniformLocation(shaderProgram, "uInvMMatrix"), false, new Float32Array(MatrixTransposeInverse(uMMatrix6)));

  gl.uniform1f(gl.getUniformLocation(shaderProgram, "EnableWoodTexture"),1.0);

  gl.drawArrays(gl.TRIANGLES, 0, vertexPositionBuffer2.numItems*vertexPositionBuffer2.itemSize); //Faktyczne wywołanie rendrowania


  setTimeout(Tick,100);

}

function handlekeydown(e)

{

 // Q W E A S D

 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);

 //U I O J K L
 if(e.keyCode==76) KameraPositionX=KameraPositionX+0.1;
 if(e.keyCode==74) KameraPositionX=KameraPositionX-0.1;
 if(e.keyCode==73) KameraPositionY=KameraPositionY+0.1;
 if(e.keyCode==75) KameraPositionY=KameraPositionY-0.1;
 if(e.keyCode==85) KameraPositionZ=KameraPositionZ+0.1;
 if(e.keyCode==79) KameraPositionZ=KameraPositionZ-0.1;
}
</script>
</head>
<body onload="startGL()" onkeydown="handlekeydown(event)">
<canvas id="canvas3D" width="640" height="480" style="border: solid black 1px"></canvas>
</body>
</html>
```