# Proste webGL
# obracanie i zoom skrzyni

# Segment główny strony

```
<body onload="webGLStart();">
    <canvas id="proste" style="border: none;" width="500"
     height="500"></canvas>
    <h2>Przyciski:</h2>
    <ul>
   <li><code>Page Up</code>/<code>Page Down</code>
     oddala/przybliża
   <li>Kursory wpływają na prędkość obracania
   <li><code>F</code> przełącza trzy różne rodzaje filtrów tekstur
    </ul>
</body>
</html>
```

# Inicjowanie webGL

```
 function webGLStart() {
   var canvas = document.getElementById(„proste");
   initGL(canvas);
   initShaders();
   initBuffers();
   initTexture();
   gl.clearColor(0.0, 0.0, 0.0, 1.0);
   gl.enable(gl.DEPTH_TEST);
   document.onkeydown = handleKeyDown;
   document.onkeyup = handleKeyUp;
   tick();
    }
</script>
</head>
```

# Wnętrze głównej pętli

```
 var lastTime = 0;
    function animate() {
   var timeNow = new Date().getTime();
   if (lastTime != 0) {
       var elapsed = timeNow - lastTime;
       xRot += (xSpeed * elapsed) / 1000.0;
       yRot += (ySpeed * elapsed) / 1000.0;
   }
   lastTime = timeNow;
    }
function tick() {
   requestAnimFrame(tick);
   handleKeys();
   drawScene();
   animate();
}
```

# Rysowanie pojedynczej klatki

```
function drawScene() {
  gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0, pMatrix);
  mat4.identity(mvMatrix);
  mat4.translate(mvMatrix, [0.0, 0.0, z]);
  mat4.rotate(mvMatrix, degToRad(xRot), [1, 0, 0]);
  mat4.rotate(mvMatrix, degToRad(yRot), [0, 1, 0]);
  gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
  gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
  gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
  gl.vertexAttribPointer(shaderProgram.textureCoordAttribute, cubeVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0,0);
  gl.activeTexture(gl.TEXTURE0);
  gl.bindTexture(gl.TEXTURE_2D, crateTextures[filter]);
  gl.uniform1i(shaderProgram.samplerUniform, 0);
  gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
  setMatrixUniforms();
  gl.drawElements(gl.TRIANGLES, cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
}
```

# Inicjowanie bufora – wierzchołki 3D

```
var cubeVertexPositionBuffer;
  var cubeVertexTextureCoordBuffer;
  var cubeVertexIndexBuffer;
  function initBuffers() {
 cubeVertexPositionBuffer = gl.createBuffer();
 gl.bindBuffer(gl.ARRAY_BUFFER,
   cubeVertexPositionBuffer);
 vertices = [
    // Front face
    -1.0, -1.0,  1.0,
   1.0, -1.0,  1.0,
   1.0,  1.0,  1.0,
    -1.0,  1.0,  1.0,
```

# Inicjowanie bufora – wsp.2D tekstur

```
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
    gl.STATIC_DRAW);
  cubeVertexPositionBuffer.itemSize = 3;
  cubeVertexPositionBuffer.numItems = 24;
  cubeVertexTextureCoordBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
  var textureCoords = [
    // Front face
    0.0, 0.0,
    1.0, 0.0,
    1.0, 1.0,
    0.0, 1.0,
```

# Inicjowanie bufora - trójkąty

```
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textureCoords), gl.STATIC_DRAW);
  cubeVertexTextureCoordBuffer.itemSize = 2;
  cubeVertexTextureCoordBuffer.numItems = 24;

  cubeVertexIndexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
  var cubeVertexIndices = [
     0, 1, 2,    0, 2, 3,    // Front face
     4, 5, 6,    4, 6, 7,    // Back face
     8, 9, 10,   8, 10, 11,  // Top face
     12, 13, 14,   12, 14, 15, // Bottom face
     16, 17, 18,   16, 18, 19, // Right face
     20, 21, 22,   20, 22, 23  // Left face
  ]
  gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(cubeVertexIndices),
     gl.STATIC_DRAW);
  cubeVertexIndexBuffer.itemSize = 1;
  cubeVertexIndexBuffer.numItems = 36;
 }
```

# Obsługa klawiatury 1

```
function handleKeys() {
  if (currentlyPressedKeys[33]) {
     // Page Up
     z -= 0.05;
  }
  if (currentlyPressedKeys[34]) {
     // Page Down
     z += 0.05;
  }
  if (currentlyPressedKeys[37]) {
     // Left cursor key
     ySpeed -= 1;
  }
  if (currentlyPressedKeys[39]) {
     // Right cursor key
     ySpeed += 1;
  }
  if (currentlyPressedKeys[38]) {
     // Up cursor key
     xSpeed -= 1;
  }
  if (currentlyPressedKeys[40]) {
     // Down cursor key
     xSpeed += 1;
  }
   }
```

# Obsługa klawiatury 2

```
var xRot = 0;
   var xSpeed = 0;
   var yRot = 0;
   var ySpeed = 0;
   var z = -5.0;
   var filter = 0;
   var currentlyPressedKeys = {};
   function handleKeyDown(event) {
      currentlyPressedKeys[event.keyCode] = true;
       if (String.fromCharCode(event.keyCode) == "F") {
                filter += 1;
                 if (filter == 3) {
                                filter = 0;
                }
       }
}
function handleKeyUp(event) {
      currentlyPressedKeys[event.keyCode] = false;
 }
```

# Grafika z macierzą

```
 var mvMatrix = mat4.create();
    var mvMatrixStack = [];
    var pMatrix = mat4.create();
function mvPushMatrix() {
   var copy = mat4.create();
   mat4.set(mvMatrix, copy);
   mvMatrixStack.push(copy);
}
function mvPopMatrix() {
   if (mvMatrixStack.length == 0) {
      throw "Invalid popMatrix!";
   }
         mvMatrix = mvMatrixStack.pop();
 }
function setMatrixUniforms() {
   gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
   gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
 }
 function degToRad(degrees) {
   return degrees * Math.PI / 180;
 }
```

# Tekstury

```
function handleLoadedTexture(textures) {
  gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
  gl.bindTexture(gl.TEXTURE_2D, textures[0]);
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, textures[0].image);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
  gl.bindTexture(gl.TEXTURE_2D, textures[1]);
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, textures[1].image);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
  gl.bindTexture(gl.TEXTURE_2D, textures[2]);
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, textures[2].image);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_NEAREST);
  gl.generateMipmap(gl.TEXTURE_2D);
  gl.bindTexture(gl.TEXTURE_2D, null);
   }
   var crateTextures = Array();
   function initTexture() {
  var crateImage = new Image();
  for (var i=0; i < 3; i++) {
     var texture = gl.createTexture();
     texture.image = crateImage;
     crateTextures.push(texture);
  }
  crateImage.onload = function () {
     handleLoadedTexture(crateTextures)
  }
  crateImage.src = "crate.gif";
}
```

# Inicjacja programu karty graficznej

```
 var shaderProgram;
   function initShaders() {
  var fragmentShader = getShader(gl, "shader-fs");
  var vertexShader = getShader(gl, "shader-vs");
  shaderProgram = gl.createProgram();
  gl.attachShader(shaderProgram, vertexShader);
  gl.attachShader(shaderProgram, fragmentShader);
  gl.linkProgram(shaderProgram);
  if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
     alert("Could not initialise shaders");
  }
  gl.useProgram(shaderProgram);
  shaderProgram.vertexPositionAttribute = gl.getAttribLocation(shaderProgram, "aVertexPosition");
  gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
  shaderProgram.textureCoordAttribute = gl.getAttribLocation(shaderProgram, "aTextureCoord");
  gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);
  shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram, "uPMatrix");
  shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram, "uMVMatrix");
  shaderProgram.samplerUniform = gl.getUniformLocation(shaderProgram, "uSampler");
}
```

# Ładowanie karty graficznej

```
function getShader(gl, id) {
  var shaderScript =
    document.getElementById(id)
    ;
  if (!shaderScript) {
     return null;
  }
  var str = "";
  var k = shaderScript.firstChild;
  while (k) {
     if (k.nodeType == 3) {
    str += k.textContent;
     }
     k = k.nextSibling;
  }
```

```
var shader;
  if (shaderScript.type == "x-shader/x-
       fragment") {
     shader =
       gl.createShader(gl.FRAGMENT_SHADER);
  } else if (shaderScript.type == "x-shader/x-
       vertex") {
     shader =
       gl.createShader(gl.VERTEX_SHADER);
  } else {
     return null;
  }
  gl.shaderSource(shader, str);
  gl.compileShader(shader);
  if (!gl.getShaderParameter(shader,
       gl.COMPILE_STATUS)) {
     alert(gl.getShaderInfoLog(shader));
     return null;
  }
  return shader;
   }
```

# Inicjowanie webGL

```
var gl;
   function initGL(canvas) {
  try {
      gl = canvas.getContext("experimental-webgl");
      gl.viewportWidth = canvas.width;
      gl.viewportHeight = canvas.height;
  } catch (e) {
  }
  if (!gl) {
      alert("Could not initialise WebGL, sorry :-(");
  }
   }
```

# Programy karty graficznej

```
<script id="shader-fs" type="x-shader/x-fragment">
    #ifdef GL_ES
    precision highp float;
    #endif
    varying vec2 vTextureCoord;
    uniform sampler2D uSampler;
    void main(void) {
   gl_FragColor = texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
    }
</script>
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec2 aTextureCoord;
    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    varying vec2 vTextureCoord;
    void main(void) {
   gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
   vTextureCoord = aTextureCoord;
    }
</script>
```

# Zrzut ekranu