WebGL

Željko Ružak

Uvod

- JavaScript API
- Interaktivna 3D grafika unutar preglednika
- Koristi grafički procesor
  - JavaScript kod
  - Shader kod

Povijest

- Vladimir Vukićević – 2006. Canvas 3D prototip
- 2007. – Mozilla i Opera imaju vlastite implementacije
- 2009. – Khronos Group – WebGL Working Group
  - Apple, Google, Mozilla, Opera i drugi
- 2011. – Verzija 1.0 WebGL specifikacije

Korištenje

- Pronaći preglednik koji ga podržava
  - Mozilla Firefox – verzija 4.0. i novije
  - Google Chrome –verzija 9 i novije
  - Safari – vezija 5.1 i novije
  - Opera – verzije 11 i 12
- Dovoljno jaka grafička kartica i najnoviji driveri
- DirectX Runtime
- Safari i opera – potrebno omogućiti

Korištenje

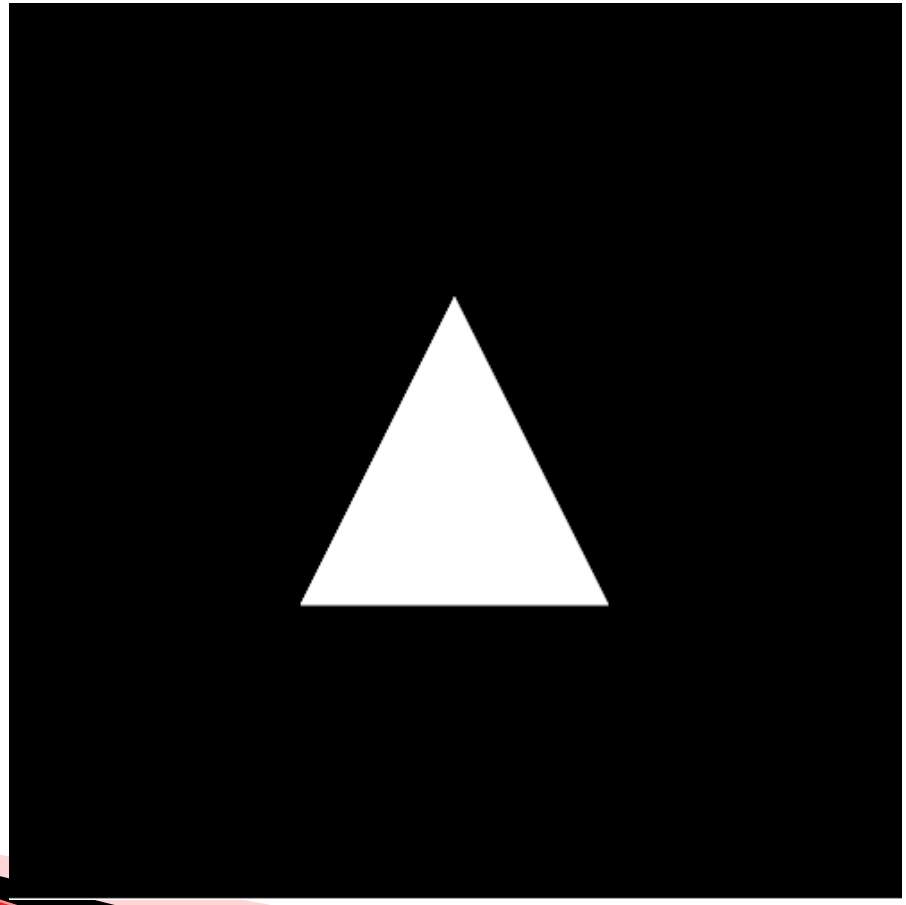
- Internet Explorer – nema podrške
  - Sigurnosni propusti
  - Još nije W3C standard
- Moguće dodati pomoću pluginova
  - Chrome Frame
  - IEWebGL
- Dostupno i na nekim mobilnim web preglednicima

Programiranje

- OpenGL – podudaranje koncepata
- Potrebno pisanje koda za mnoge mogučnosti prisutne u OpenGL-u
  - Kamera,
  - svjetla,
  - transformacijske matrice…
- Programiranje u JavaScriptu
- Programiranje shadera – kod koji se izvodi u grafičkom procesoru

Programiranje – primjer

- Jednostavan primjer



Primjer – canvas

- Canvas element za 3D scenu
- Pri učitavanju html datoteke se pokreće *webGLStart()*

```
<body onload="webGLStart();">
    <canvas id="primjer0" style="border: none;"
     width="500" height="500"></canvas>
</body>
```

Primjer – početak

```
function webGLStart() {
    var canvas = document.getElementById("primjer0");
    initGL(canvas);
    initShaders();
    initBuffers();

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);

    drawScene();
}
```

Primjer – postavljanje konteksta

```
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 :-(");
    }
}
```

Primjer – priprema međuspremnika

```
var triangleVertexPositionBuffer;

function initBuffers() {
    triangleVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    var vertices = [
         0.0,  1.0,  0.0,
        -1.0, -1.0,  0.0,
         1.0, -1.0,  0.0
    ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices)
                  , gl.STATIC_DRAW);
    triangleVertexPositionBuffer.itemSize = 3;
    triangleVertexPositionBuffer.numItems = 3;
}
```

Primjer – crtanje

```
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, -7.0]);
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
                           triangleVertexPositionBuffer.itemSize,
                           gl.FLOAT, false, 0, 0);
    setMatrixUniforms();
    gl.drawArrays(gl.TRIANGLES, 0, triangleVertexPositionBuffer.numItems);

}
```

Primjer – inicijalizacija shadera

```
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.pMatrixUniform = gl.getUniformLocation(shaderProgram, "uPMatrix");
    shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram, "uMVMatrix");
}
```

Primjer – dohvaćanje shadera

- Pronalazi shader
- Nije previše bitno na koji način

```
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;
}
```

Primjer – vertex shader

- Pretvaraju 3D koordinate u 2D oblik prikladan zaslonu
- Određuju koordinate ali sami ništa ne iscrtavaju

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
    }
</script>
```

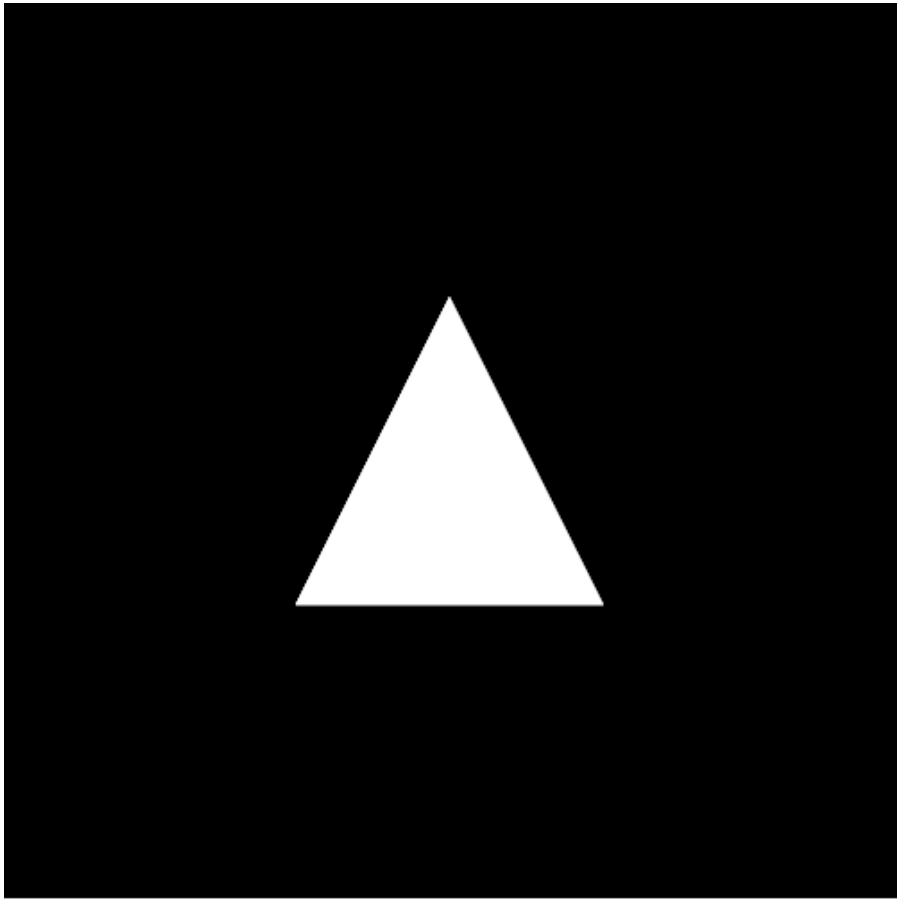
Primjer – fragment shader

- Poznati i kao pixel shaderi
- Promjena boje svakog piksela

```
<script id="shader-fs" type="x-shader/x-fragment">
    precision mediump float;

    void main(void) {
        gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
    }
</script>
```

Primjer – Gotovo



Zaključak

- Nova tehnologija – još u razvoju
  - Relativno malo primjera
- Složeno za koristiti u odnosu na OpenGL
  - Korištenje shadera obavezno
  - Potrebno programiranje mnogih osnovnih funkcija
- Polako se javljaju razne biblioteke, pluginovi i alati koji mogu znatno olakšati programiranje u WebGL-u.