Sección de Informática Gráfica | Computer Graphics Group
VALENCIA

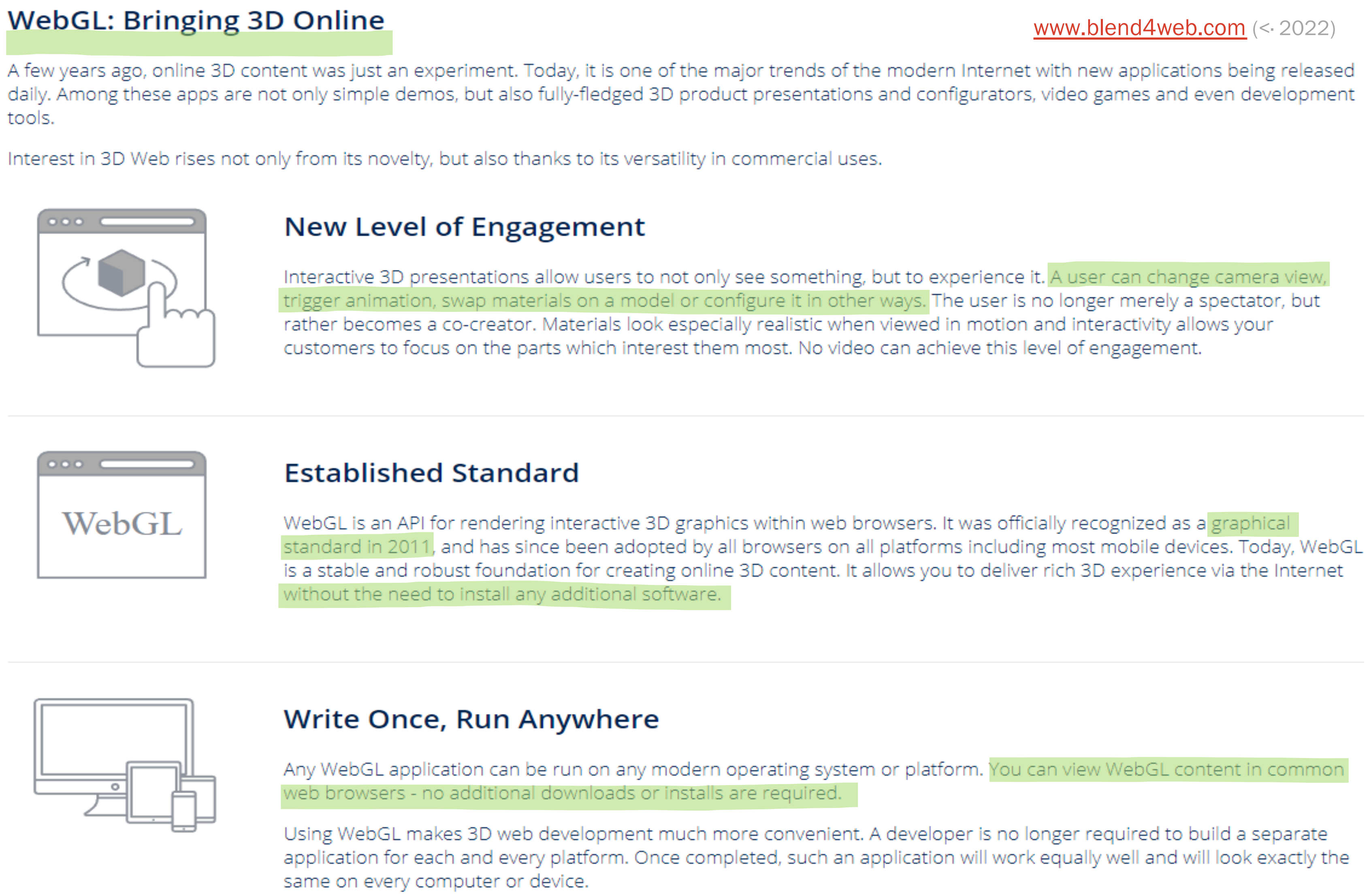
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Introducción

WebGL™

Gráficos 3D en la web

WebGL?



www.blend4web.com (<- 2022)

WebGL: Bringing 3D Online

A few years ago, online 3D content was just an experiment. Today, it is one of the major trends of the modern Internet with new applications being released daily. Among these apps are not only simple demos, but also fully-fledged 3D product presentations and configurators, video games and even development tools.

Interest in 3D Web rises not only from its novelty, but also thanks to its versatility in commercial uses.

New Level of Engagement

Interactive 3D presentations allow users to not only see something, but to experience it. A user can change camera view, trigger animation, swap materials on a model or configure it in other ways. The user is no longer merely a spectator, but rather becomes a co-creator. Materials look especially realistic when viewed in motion and interactivity allows your customers to focus on the parts which interest them most. No video can achieve this level of engagement.

Established Standard

WebGL is an API for rendering interactive 3D graphics within web browsers. It was officially recognized as a graphical standard in 2011, and has since been adopted by all browsers on all platforms including most mobile devices. Today, WebGL is a stable and robust foundation for creating online 3D content. It allows you to deliver rich 3D experience via the Internet without the need to install any additional software.

Write Once, Run Anywhere

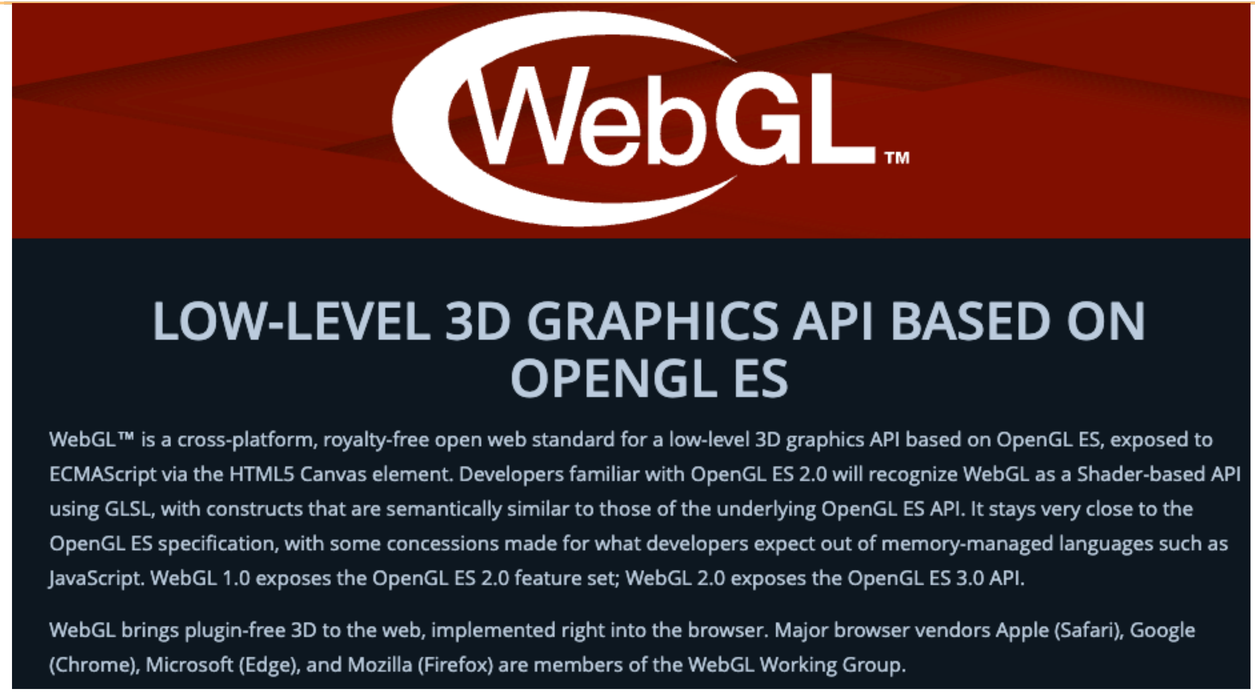
Any WebGL application can be run on any modern operating system or platform. You can view WebGL content in common web browsers - no additional downloads or installs are required.

Using WebGL makes 3D web development much more convenient. A developer is no longer required to build a separate application for each and every platform. Once completed, such an application will work equally well and will look exactly the same on every computer or device.

Algunos ejemplos

- Matemáticas: Google search, p.e. sin(x)+sin(y)
- Personalización/presentación de producto: ring boutique , calzado , automoción , aparatos
- Visualización de datos: population , noise, bookcase
- Modelado: snappytree, sketchfab
- Videojuegos: keepout, endlessTruck

WebGL



LOW-LEVEL 3D GRAPHICS API BASED ON OPENGL ES

WebGL™ is a cross-platform, royalty-free open web standard for a low-level 3D graphics API based on OpenGL ES, exposed to ECMAScript via the HTML5 Canvas element. Developers familiar with OpenGL ES 2.0 will recognize WebGL as a Shader-based API using GLSL, with constructs that are semantically similar to those of the underlying OpenGL ES API. It stays very close to the OpenGL ES specification, with some concessions made for what developers expect out of memory-managed languages such as JavaScript. WebGL 1.0 exposes the OpenGL ES 2.0 feature set; WebGL 2.0 exposes the OpenGL ES 3.0 API.

WebGL brings plugin-free 3D to the web, implemented right into the browser. Major browser vendors Apple (Safari), Google (Chrome), Microsoft (Edge), and Mozilla (Firefox) are members of the WebGL Working Group.

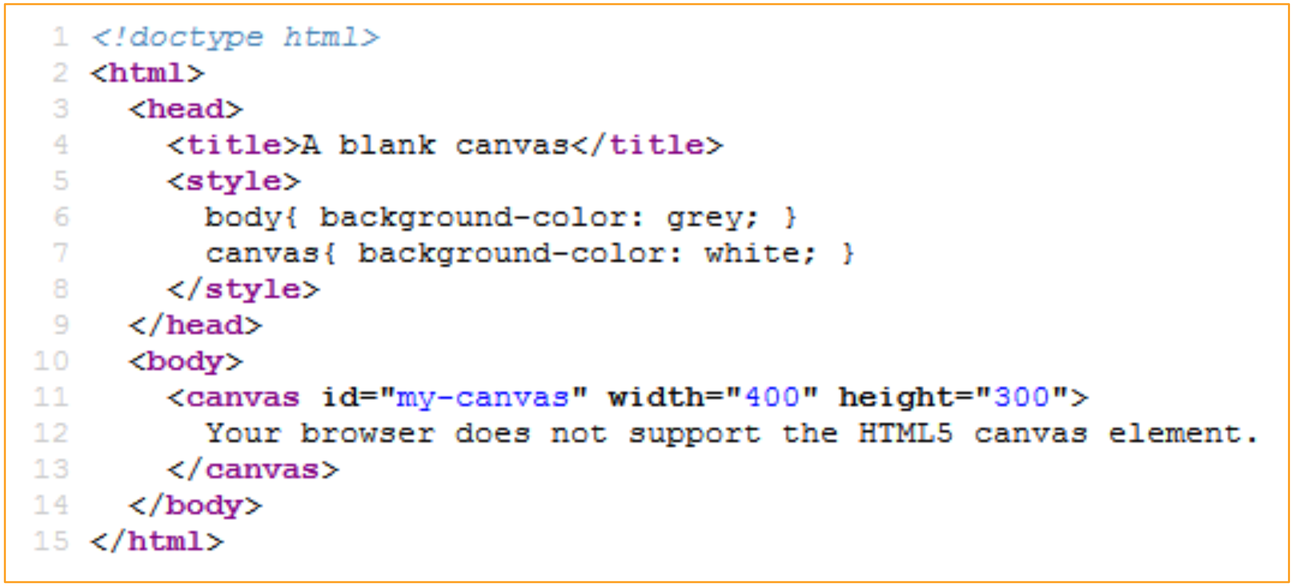
www.khronos.org

WebGL spec

- Gráficos 3D en el navegador
- Tecnología OpenGL
- Integración en documento HTML5 / javascript
- Gratuito
- Aval de Khronos Group

Canvas en HTML5

- tag en HTML5 <canvas> soporte de WebGL
- Área gráfica dentro de la página web (lienzo)
- Para situarlo usar un contenedor <div>
- Atributos: color de fondo, ancho y alto



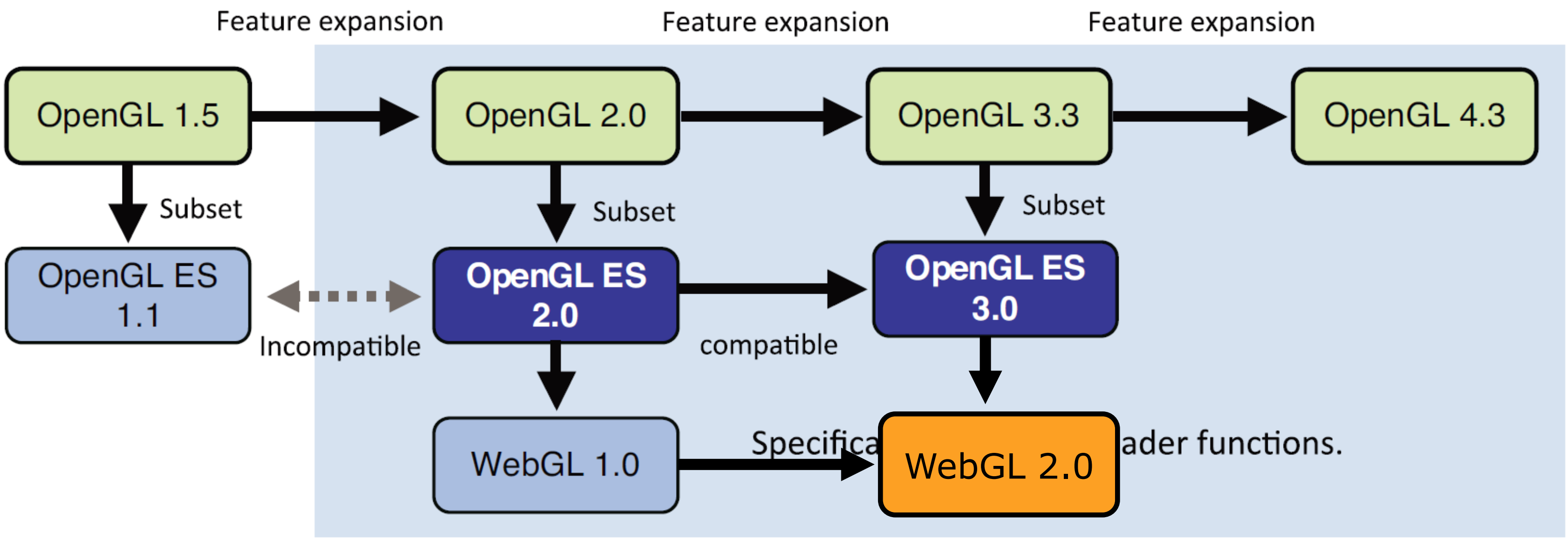
```
<!doctype html>
<html>
  <head>
    <title>A blank canvas</title>
    <style>
      body{ background-color: grey; }
      canvas{ background-color: white; }
    </style>
  </head>
  <body>
    <canvas id="my-canvas" width="400" height="300">
      Your browser does not support the HTML5 canvas element.
    </canvas>
  </body>
</html>
```

Contexto gráfico

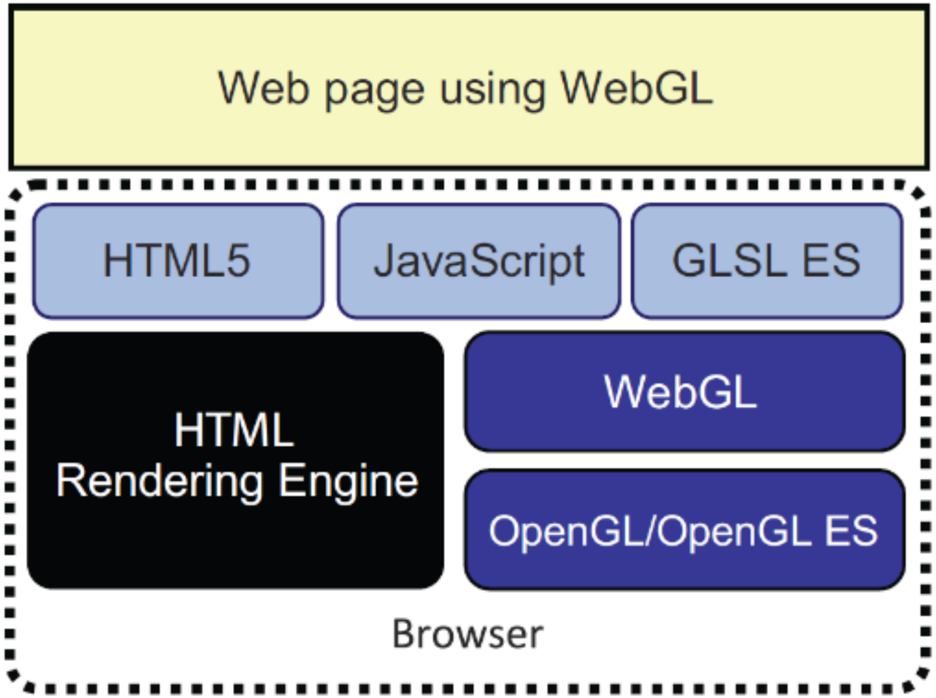
- Al área gráfica <canvas> debe asociársele un contexto gráfico
- Si el área gráfica es el lienzo, el contexto gráfico es el pincel y la pintura
- Contextos gráficos hoy:
  - “2d”
  - “webgl”
  - “webgl2”
- El contexto gráfico se maneja usando javascript

última versión webgl 2.0

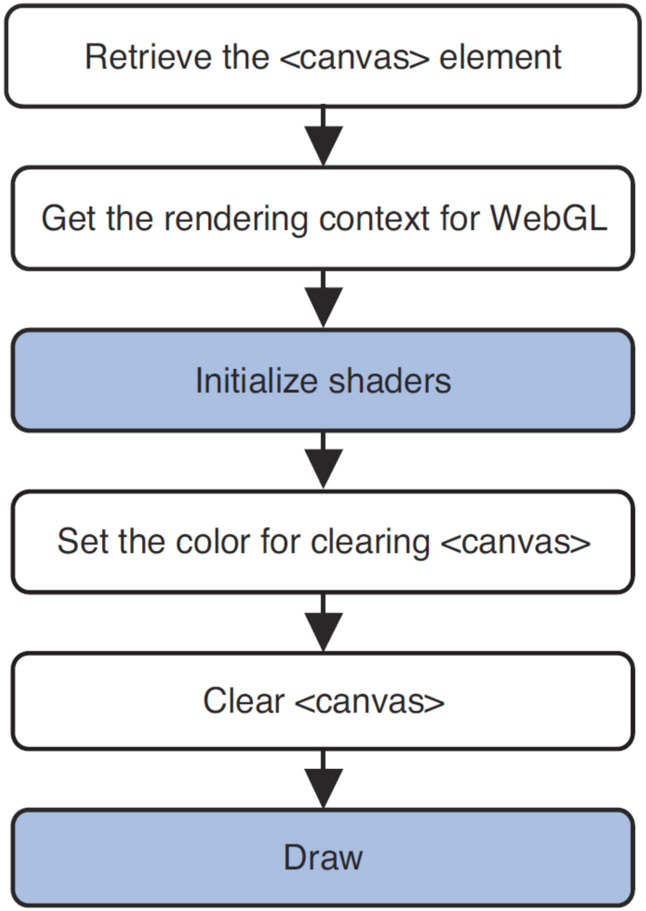
WebGL/OpenGL



WebGL: estructura de una aplicación

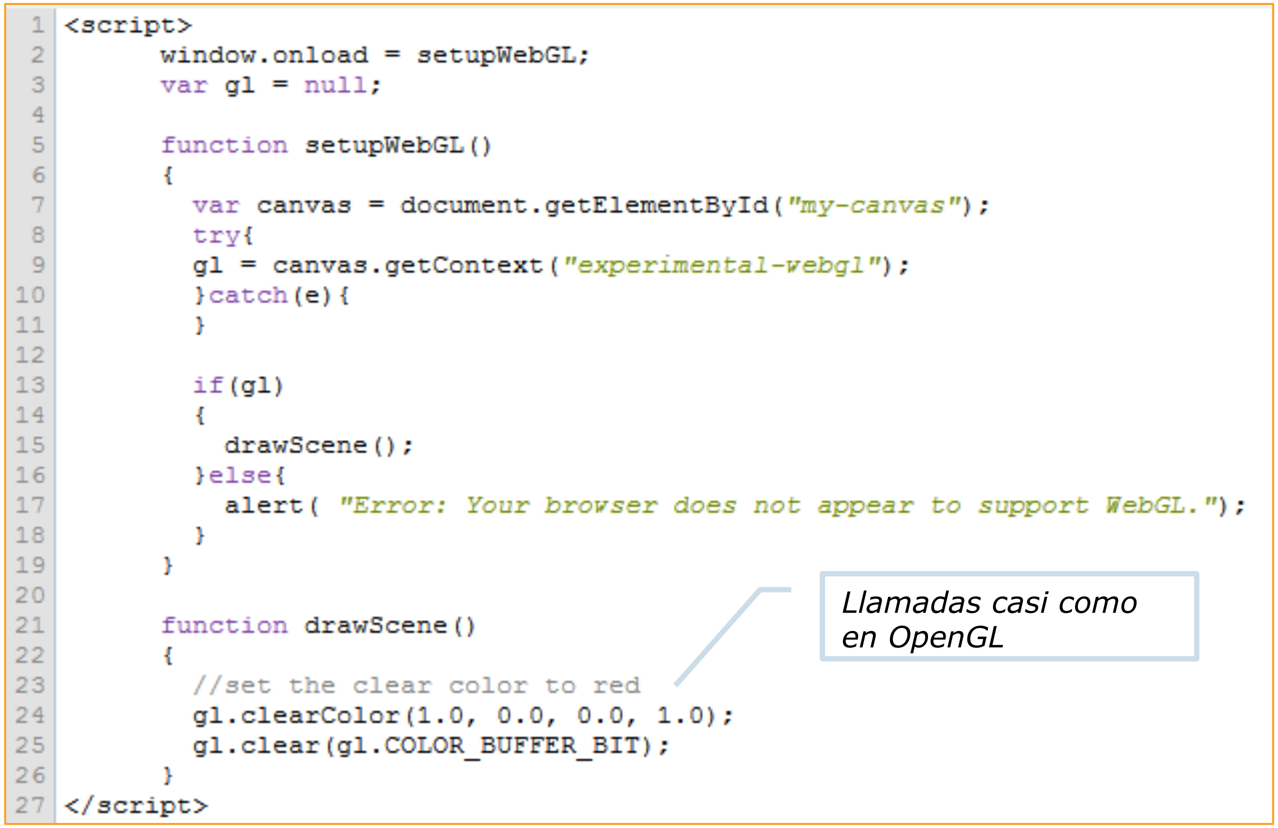


WebGL: flujo de un programa


Retrieve the <canvas> element
Get the rendering context for WebGL
Initialize shaders
Set the color for clearing <canvas>
Clear <canvas>
Draw

WebGL: ejemplo mínimo

- Se asocia al área gráfica (canvas). Contexto en var *gl*
- Chrome, Firefox, Safari, Opera, Edge, … http://get.webgl.org



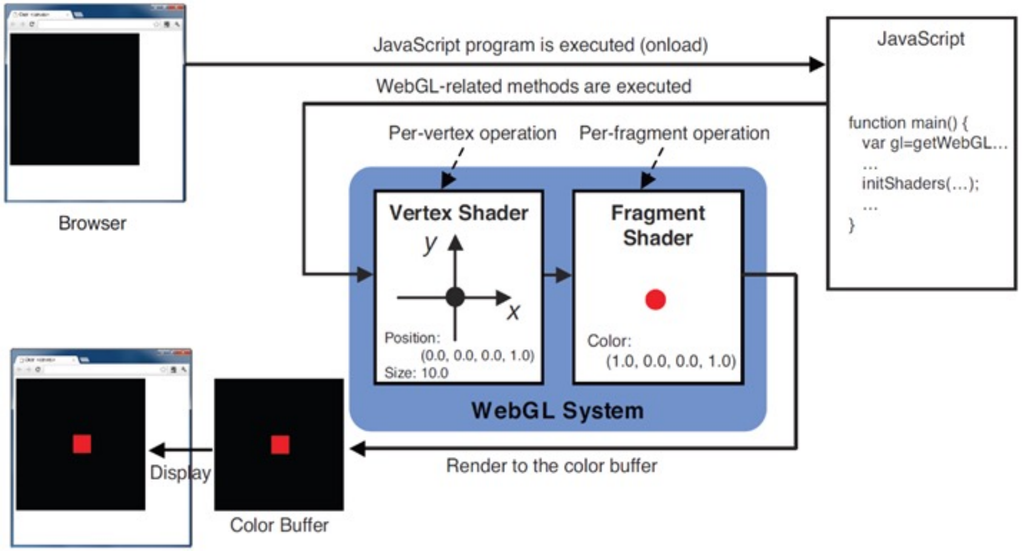
```
<script>
      window.onload = setupWebGL;
      var gl = null;

      function setupWebGL()
      {
        var canvas = document.getElementById("my-canvas");
        try{
        gl = canvas.getContext("experimental-webgl");
        }catch(e){
        }

        if(gl)
        {
          drawScene();
        }else{
          alert( "Error: Your browser does not appear to support WebGL.");
        }
      }

      function drawScene()
      {
        //set the clear color to red
        gl.clearColor(1.0, 0.0, 0.0, 1.0);
        gl.clear(gl.COLOR_BUFFER_BIT);
      }
</script>
```

Llamadas casi como en OpenGL

https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API

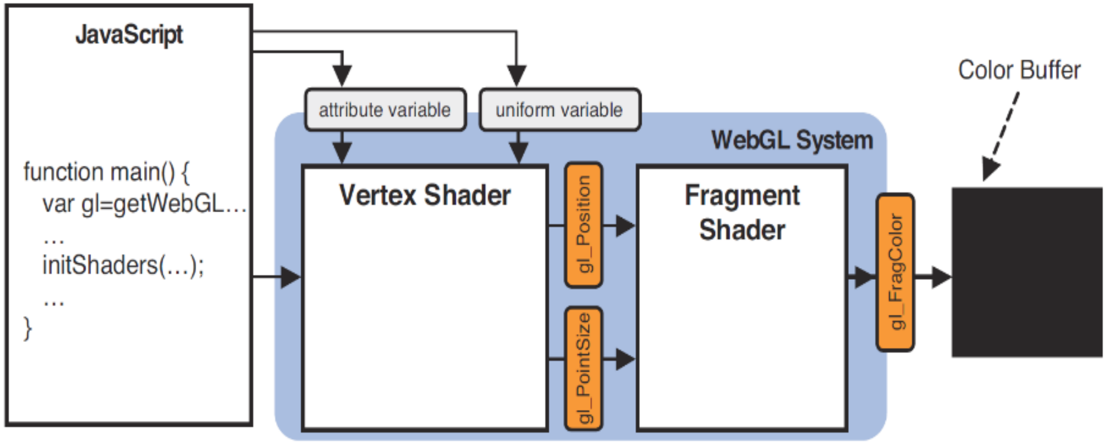
Shaders

- Programa tarjeta gráfica
- Una vez instalado afecta a lo que se dibuje a continuación
- Instalación en tiempo de ejecución
- Tipos de shaders en webgl:
  - vértices
  - fragmentos



```
function main() {
  var gl=getWebGL...
  ...
  initShaders(...);
  ...
}
```

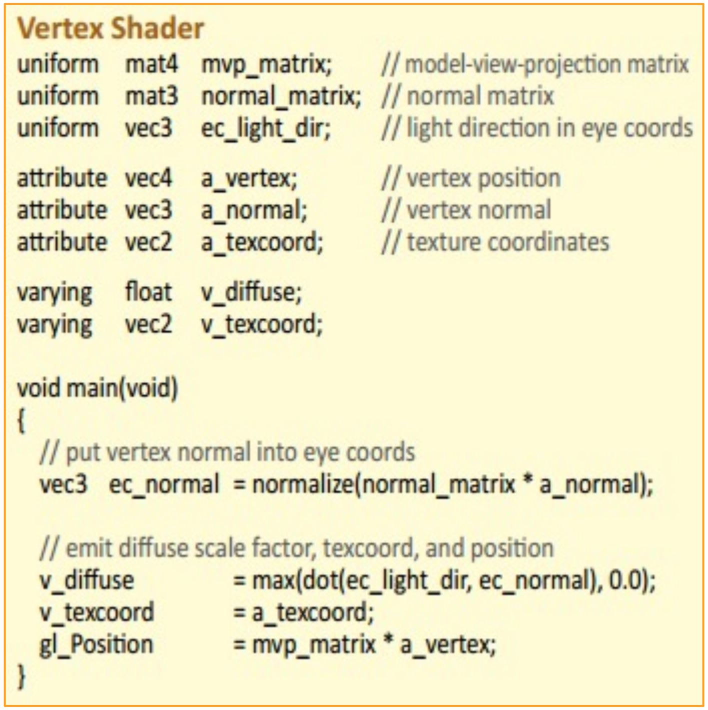
Shaders y aplicación: comunicación



Shaders

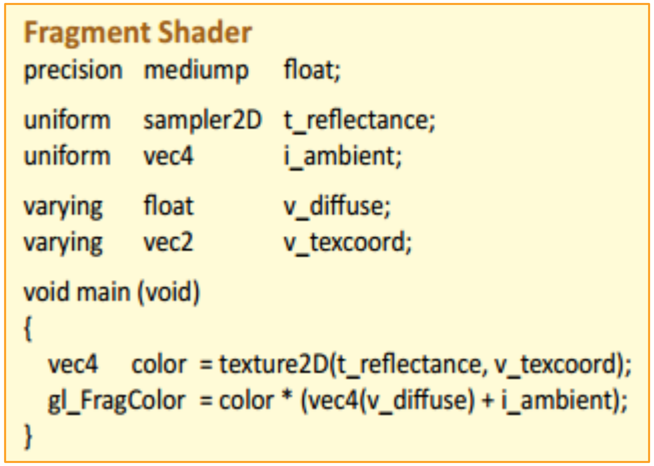
- Imprescindibles en WebGL
- GLSL: OpenGL Shading Language

Referencia rápida



Vertex Shader

```
uniform   mat4  mvp_matrix;     // model-view-projection matrix
uniform   mat3  normal_matrix;  // normal matrix
uniform   vec3  ec_light_dir;   // light direction in eye coords

attribute vec4  a_vertex;       // vertex position
attribute vec3  a_normal;       // vertex normal
attribute vec2  a_texcoord;     // texture coordinates

varying   float v_diffuse;
varying   vec2  v_texcoord;

void main(void)
{
  // put vertex normal into eye coords
  vec3  ec_normal  = normalize(normal_matrix * a_normal);

  // emit diffuse scale factor, texcoord, and position
  v_diffuse        = max(dot(ec_light_dir, ec_normal), 0.0);
  v_texcoord       = a_texcoord;
  gl_Position      = mvp_matrix * a_vertex;
}
```

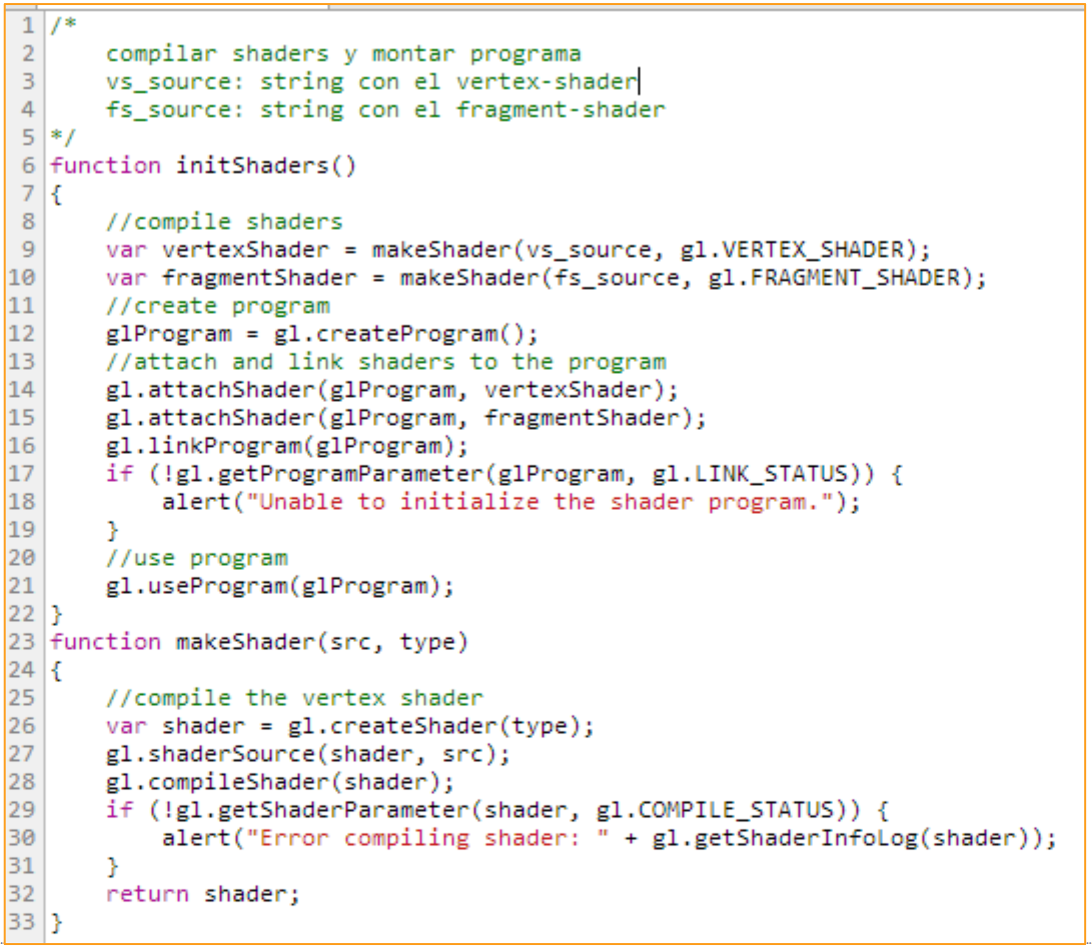


Fragment Shader

```
precision mediump   float;

uniform   sampler2D t_reflectance;
uniform   vec4      i_ambient;

varying   float     v_diffuse;
varying   vec2      v_texcoord;

void main (void)
{
  vec4    color  = texture2D(t_reflectance, v_texcoord);
  gl_FragColor   = color * (vec4(v_diffuse) + i_ambient);
}
```

Program

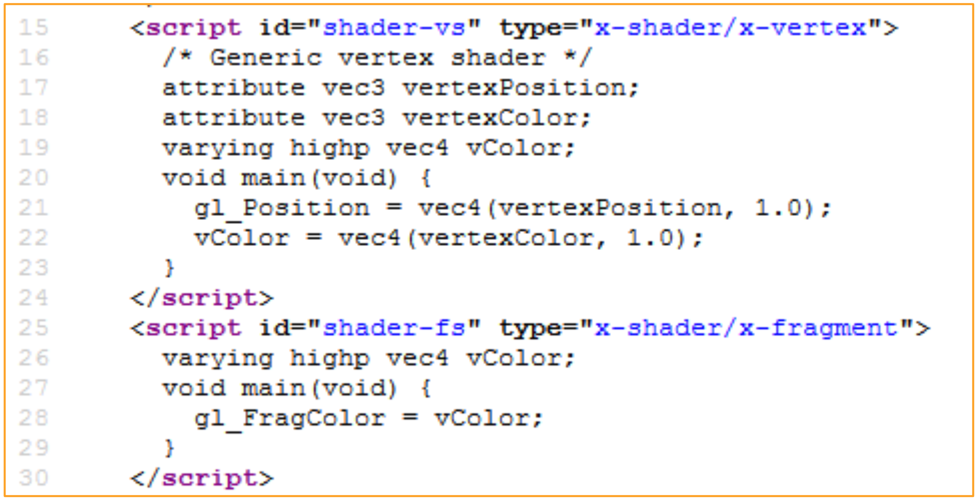
- program : código de *shaders* compilado y montado (en ejecución)
- Corre en GPU .... web en GPU !



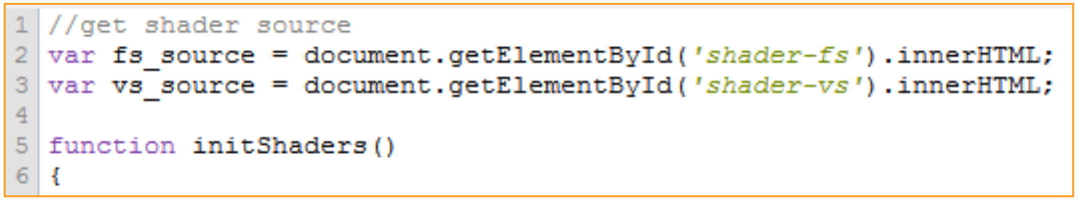
```
/*
    compilar shaders y montar programa
    vs_source: string con el vertex-shader
    fs_source: string con el fragment-shader
*/
function initShaders()
{
    //compile shaders
    var vertexShader = makeShader(vs_source, gl.VERTEX_SHADER);
    var fragmentShader = makeShader(fs_source, gl.FRAGMENT_SHADER);
    //create program
    glProgram = gl.createProgram();
    //attach and link shaders to the program
    gl.attachShader(glProgram, vertexShader);
    gl.attachShader(glProgram, fragmentShader);
    gl.linkProgram(glProgram);
    if (!gl.getProgramParameter(glProgram, gl.LINK_STATUS)) {
        alert("Unable to initialize the shader program.");
    }
    //use program
    gl.useProgram(glProgram);
}
function makeShader(src, type)
{
    //compile the vertex shader
    var shader = gl.createShader(type);
    gl.shaderSource(shader, src);
    gl.compileShader(shader);
    if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
        alert("Error compiling shader: " + gl.getShaderInfoLog(shader));
    }
    return shader;
}
```

Shaders en documento

- scripts en propio documento
- Usar id y recuperar shader con getElementById(‘shader_id’).innerHTML



```
<script id="shader-vs" type="x-shader/x-vertex">
  /* Generic vertex shader */
  attribute vec3 vertexPosition;
  attribute vec3 vertexColor;
  varying highp vec4 vColor;
  void main(void) {
    gl_Position = vec4(vertexPosition, 1.0);
    vColor = vec4(vertexColor, 1.0);
  }
</script>
<script id="shader-fs" type="x-shader/x-fragment">
  varying highp vec4 vColor;
  void main(void) {
    gl_FragColor = vColor;
  }
</script>
```



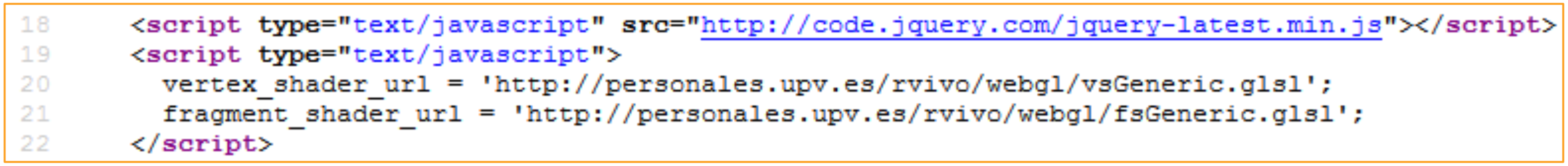
```
//get shader source
var fs_source = document.getElementById('shader-fs').innerHTML;
var vs_source = document.getElementById('shader-vs').innerHTML;

function initShaders()
{
```

setshaders.js

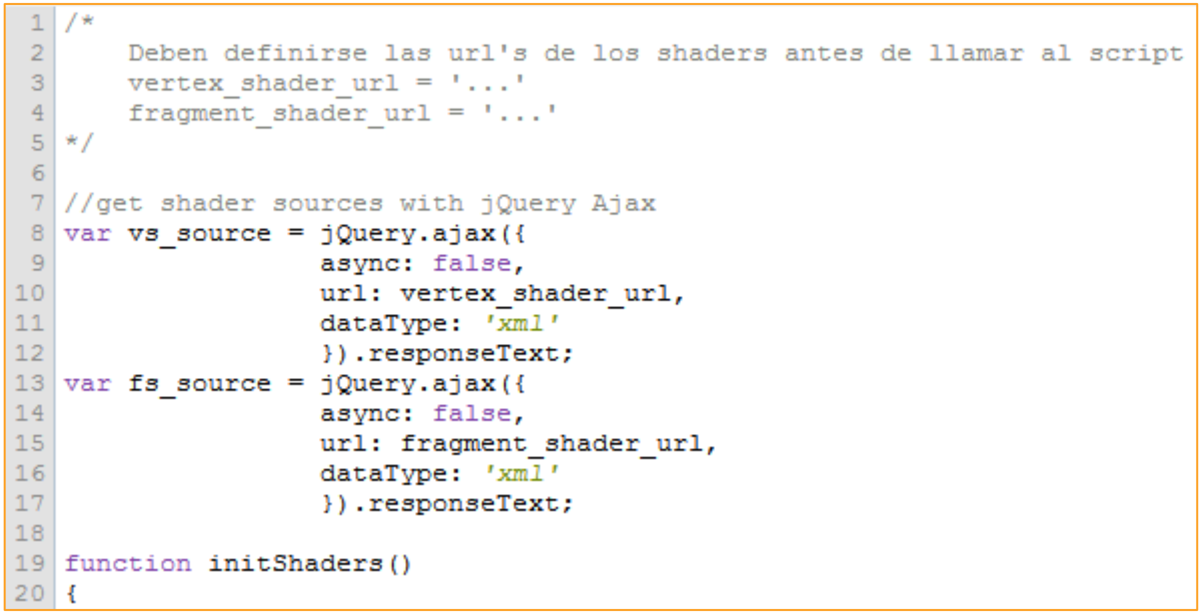
http://personales.upv.es/rvivo/webgl/innershaders.html

Shaders en ficheros en el servidor

- Shader en url del servidor como fichero de texto
- Recuperar con jQuery.ajax().responseText
- Cargar jQuery



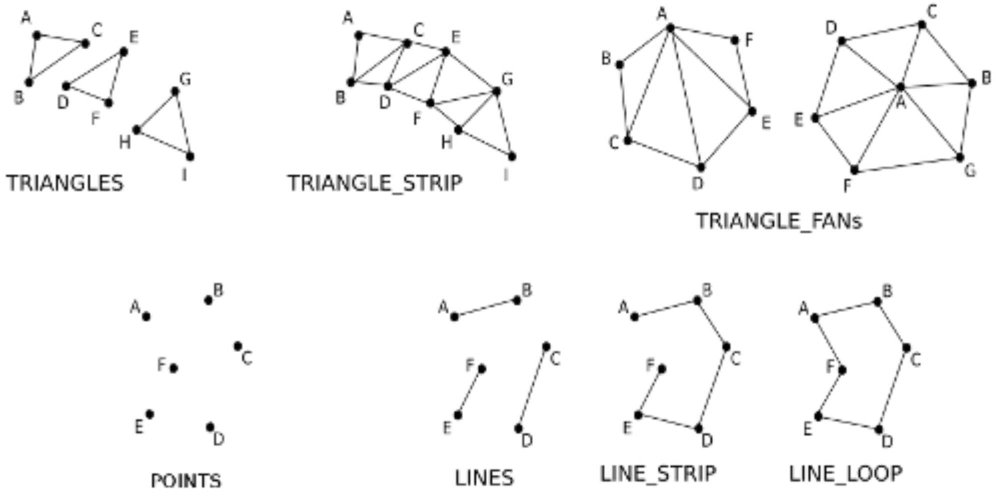
```
<script type="text/javascript" src="http://code.jquery.com/jquery-latest.min.js"></script>
<script type="text/javascript">
  vertex_shader_url = 'http://personales.upv.es/rvivo/webgl/vsGeneric.glsl';
  fragment_shader_url = 'http://personales.upv.es/rvivo/webgl/fsGeneric.glsl';
</script>
```



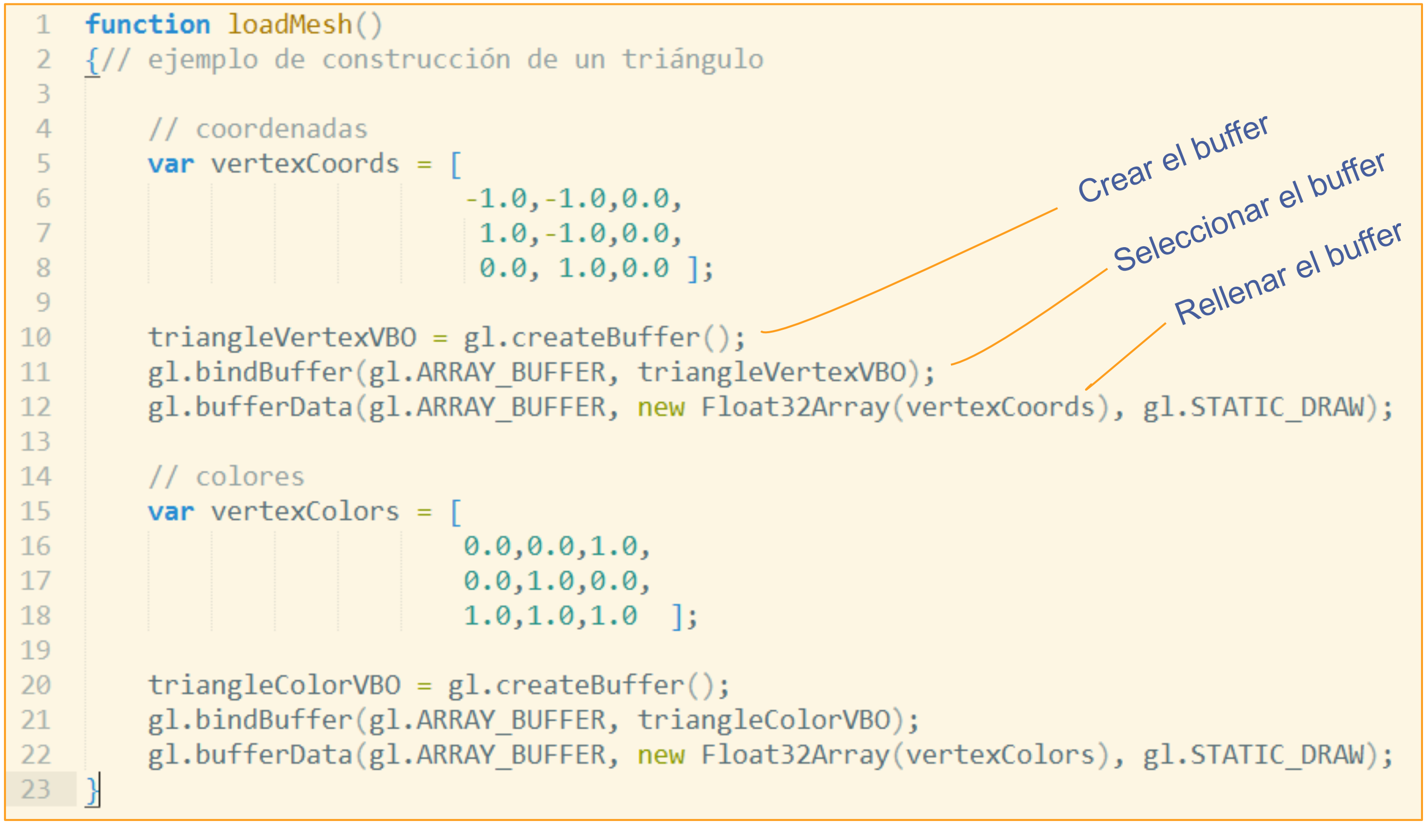
```
/*
    Deben definirse las url's de los shaders antes de llamar al script
    vertex_shader_url = '...'
    fragment_shader_url = '...'
*/

//get shader sources with jQuery Ajax
var vs_source = jQuery.ajax({
                async: false,
                url: vertex_shader_url,
                dataType: 'xml'
                }).responseText;
var fs_source = jQuery.ajax({
                async: false,
                url: fragment_shader_url,
                dataType: 'xml'
                }).responseText;

function initShaders()
{
```

Carga de geometría

- Mallas poligonales
- Uso de Vertex Buffer Objects (VBO)
  - Topología implícita en el orden: TRIANGLE_STRIP, TRIANGLE_FAN
  - Topología explicita mediante vector de índices: TRIANGLES
  - Otras primitivas: POINTS, LINES, LINE_STRIP, LINE_LOOP



Creación de VBO's



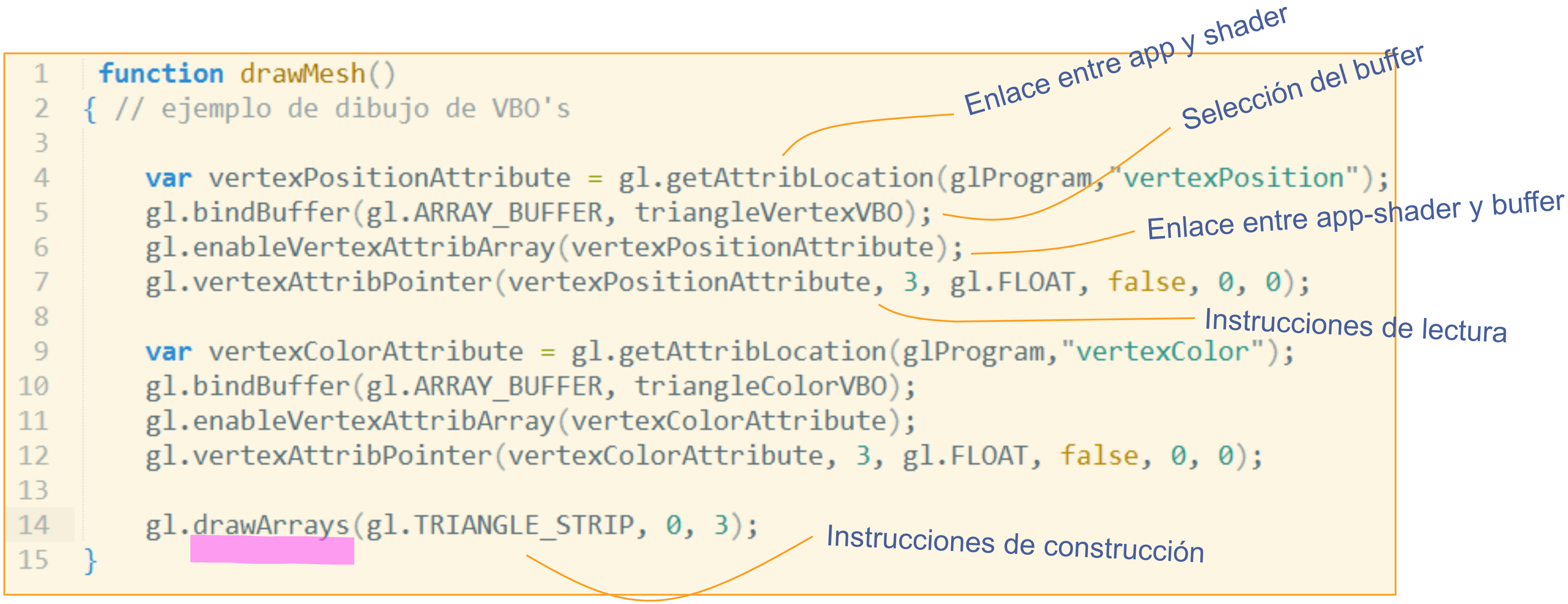
```
function loadMesh()
{// ejemplo de construcción de un triángulo

    // coordenadas
    var vertexCoords = [
                        -1.0,-1.0,0.0,
                         1.0,-1.0,0.0,
                         0.0, 1.0,0.0 ];

    triangleVertexVBO = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexVBO);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexCoords), gl.STATIC_DRAW);

    // colores
    var vertexColors = [
                        0.0,0.0,1.0,
                        0.0,1.0,0.0,
                        1.0,1.0,1.0  ];

    triangleColorVBO = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleColorVBO);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexColors), gl.STATIC_DRAW);
}
```

Crear el buffer

Seleccionar el buffer

Rellenar el buffer

Dibujo de VBO’s implícito en el orden



```
 function drawMesh()
{ // ejemplo de dibujo de VBO's

    var vertexPositionAttribute = gl.getAttribLocation(glProgram,"vertexPosition");
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexVBO);
    gl.enableVertexAttribArray(vertexPositionAttribute);
    gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);

    var vertexColorAttribute = gl.getAttribLocation(glProgram,"vertexColor");
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleColorVBO);
    gl.enableVertexAttribArray(vertexColorAttribute);
    gl.vertexAttribPointer(vertexColorAttribute, 3, gl.FLOAT, false, 0, 0);

    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 3);
}
```

Enlace entre app y shader

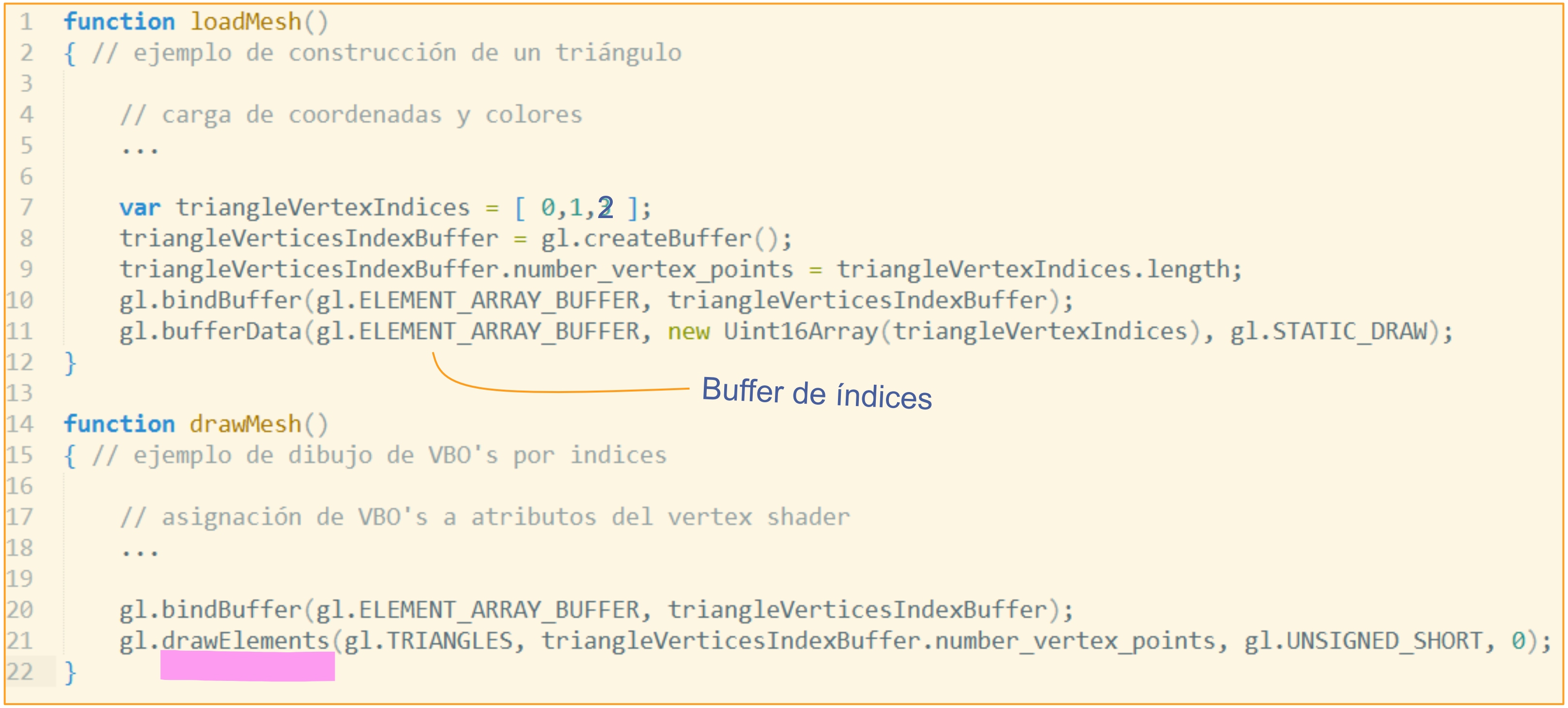
Selección del buffer

Enlace entre app-shader y buffer

Instrucciones de lectura

Instrucciones de construcción

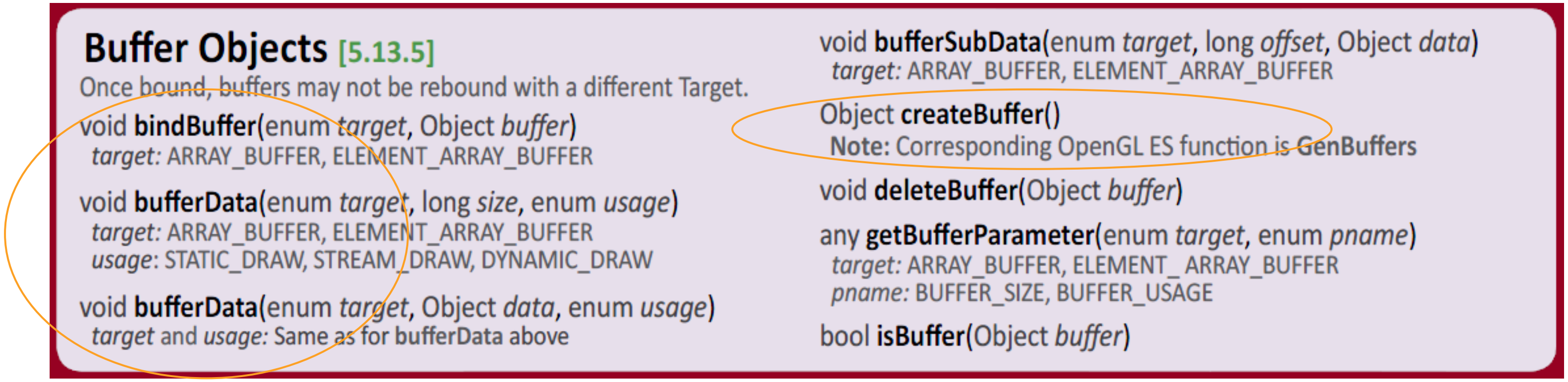
Dibujo de VBO’s mediante índices



```
function loadMesh()
{ // ejemplo de construcción de un triángulo

    // carga de coordenadas y colores
    ...

    var triangleVertexIndices = [ 0,1,2 ];
    triangleVerticesIndexBuffer = gl.createBuffer();
    triangleVerticesIndexBuffer.number_vertex_points = triangleVertexIndices.length;
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, triangleVerticesIndexBuffer);
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(triangleVertexIndices), gl.STATIC_DRAW);
}

function drawMesh()
{ // ejemplo de dibujo de VBO's por indices

    // asignación de VBO's a atributos del vertex shader
    ...

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, triangleVerticesIndexBuffer);
    gl.drawElements(gl.TRIANGLES, triangleVerticesIndexBuffer.number_vertex_points, gl.UNSIGNED_SHORT, 0);
}
```

Buffer de índices

Resumen VBO's



Buffer Objects [5.13.5]

Once bound, buffers may not be rebound with a different Target.

void **bindBuffer**(enum *target*, Object *buffer*)
target: ARRAY_BUFFER, ELEMENT_ARRAY_BUFFER

void **bufferData**(enum *target*, long *size*, enum *usage*)
target: ARRAY_BUFFER, ELEMENT_ARRAY_BUFFER
usage: STATIC_DRAW, STREAM_DRAW, DYNAMIC_DRAW

void **bufferData**(enum *target*, Object *data*, enum *usage*)
target and *usage:* Same as for bufferData above

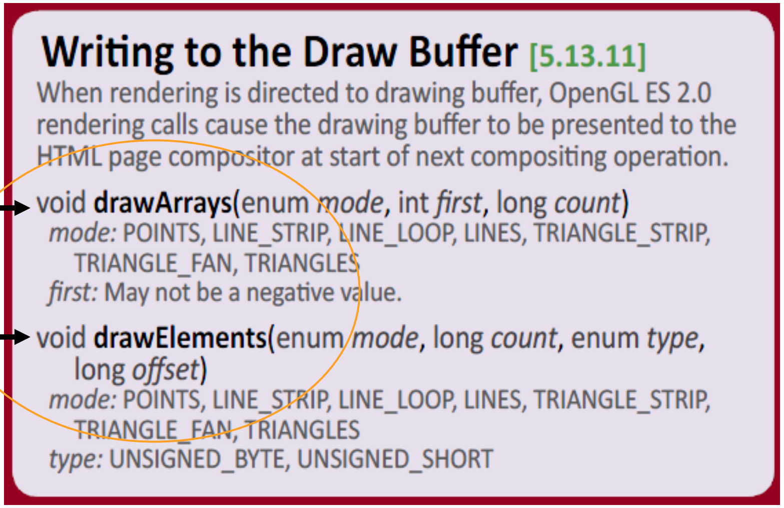
void **bufferSubData**(enum *target*, long *offset*, Object *data*)
target: ARRAY_BUFFER, ELEMENT_ARRAY_BUFFER

Object **createBuffer**()
Note: Corresponding OpenGL ES function is **GenBuffers**

void **deleteBuffer**(Object *buffer*)

any **getBufferParameter**(enum *target*, enum *pname*)
target: ARRAY_BUFFER, ELEMENT_ ARRAY_BUFFER
pname: BUFFER_SIZE, BUFFER_USAGE

bool **isBuffer**(Object *buffer*)



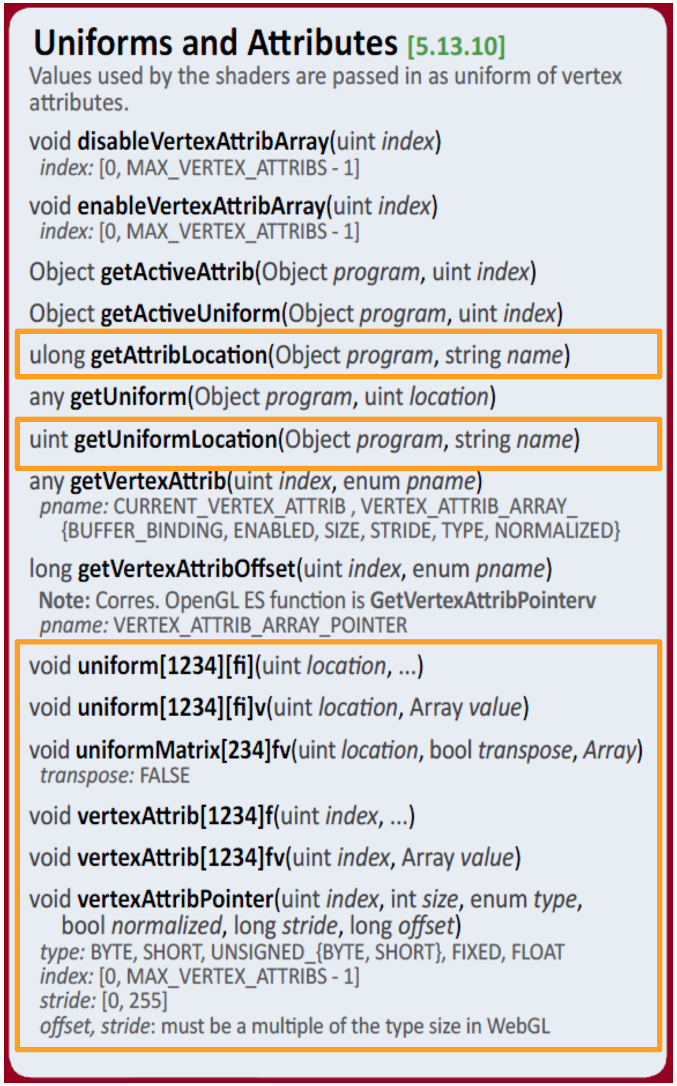
Writing to the Draw Buffer [5.13.11]

When rendering is directed to drawing buffer, OpenGL ES 2.0 rendering calls cause the drawing buffer to be presented to the HTML page compositor at start of next compositing operation.

implícita → void **drawArrays**(enum *mode*, int *first*, long *count*)
mode: POINTS, LINE_STRIP, LINE_LOOP, LINES, TRIANGLE_STRIP, TRIANGLE_FAN, TRIANGLES
first: May not be a negative value.

explícita → void **drawElements**(enum *mode*, long *count*, enum *type*, long *offset*)
mode: POINTS, LINE_STRIP, LINE_LOOP, LINES, TRIANGLE_STRIP, TRIANGLE_FAN, TRIANGLES
type: UNSIGNED_BYTE, UNSIGNED_SHORT

Interfaz Shader – App



Uniforms and Attributes [5.13.10]

Values used by the shaders are passed in as uniform of vertex attributes.

void **disableVertexAttribArray**(uint *index*)
index: [0, MAX_VERTEX_ATTRIBS - 1]

void **enableVertexAttribArray**(uint *index*)
index: [0, MAX_VERTEX_ATTRIBS - 1]

Object **getActiveAttrib**(Object *program*, uint *index*)

Object **getActiveUniform**(Object *program*, uint *index*)

ulong **getAttribLocation**(Object *program*, string *name*)

any **getUniform**(Object *program*, uint *location*)

uint **getUniformLocation**(Object *program*, string *name*)

any **getVertexAttrib**(uint *index*, enum *pname*)
pname: CURRENT_VERTEX_ATTRIB , VERTEX_ATTRIB_ARRAY_ {BUFFER_BINDING, ENABLED, SIZE, STRIDE, TYPE, NORMALIZED}

long **getVertexAttribOffset**(uint *index*, enum *pname*)
Note: Corres. OpenGL ES function is **GetVertexAttribPointerv**
pname: VERTEX_ATTRIB_ARRAY_POINTER

void **uniform[1234][fi]**(uint *location*, ...)

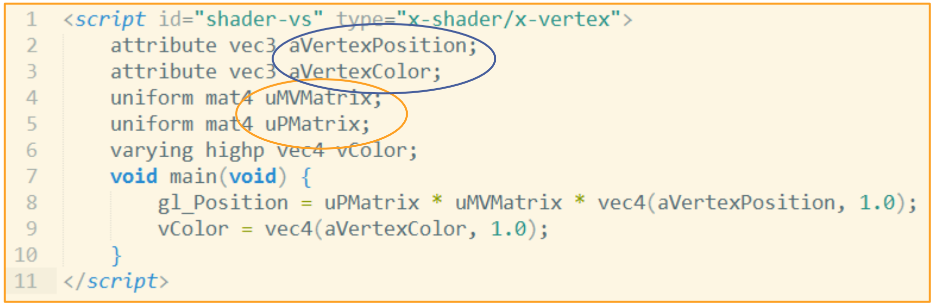
void **uniform[1234][fi]v**(uint *location*, Array *value*)

void **uniformMatrix[234]fv**(uint *location*, bool *transpose*, Array)
transpose: FALSE

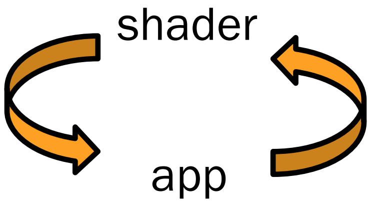
void **vertexAttrib[1234]f**(uint *index*, ...)

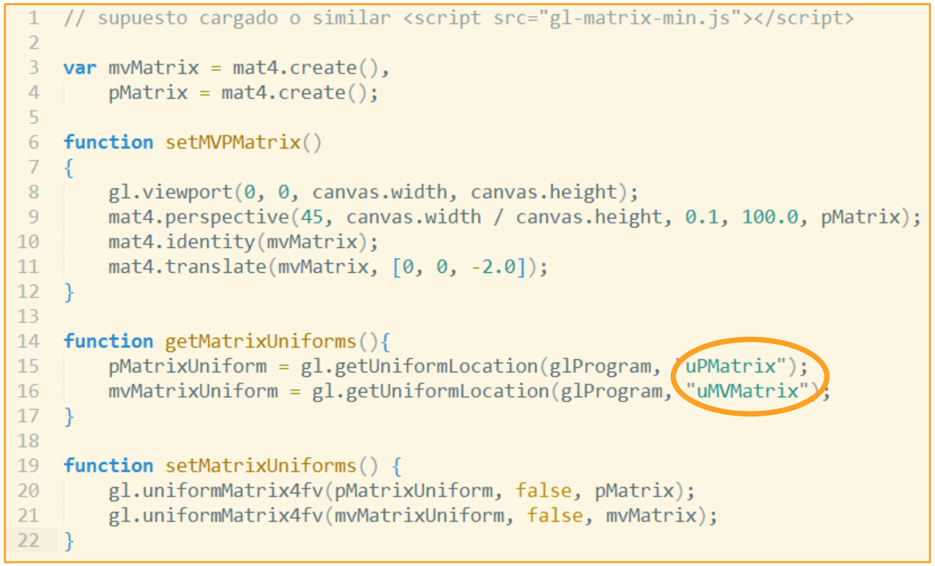
void **vertexAttrib[1234]fv**(uint *index*, Array *value*)

void **vertexAttribPointer**(uint *index*, int *size*, enum *type*, bool *normalized*, long *stride*, long *offset*)
type: BYTE, SHORT, UNSIGNED_{BYTE, SHORT}, FIXED, FLOAT
index: [0, MAX_VERTEX_ATTRIBS - 1]
stride: [0, 255]
offset, stride: must be a multiple of the type size in WebGL



```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexColor;
    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    varying highp vec4 vColor;
    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = vec4(aVertexColor, 1.0);
    }
</script>
```

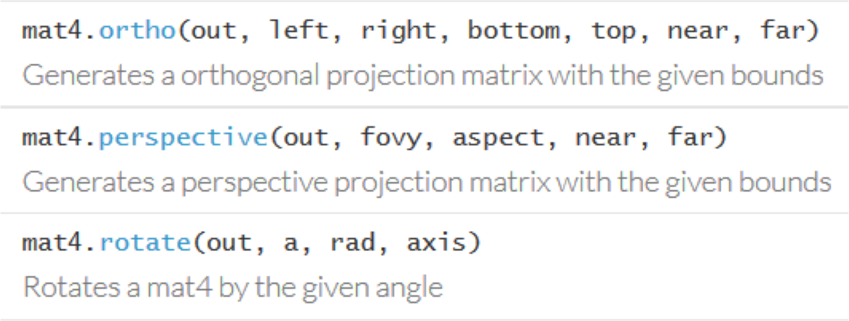




```
// supuesto cargado o similar <script src="gl-matrix-min.js"></script>

var mvMatrix = mat4.create(),
    pMatrix = mat4.create();

function setMVPMatrix()
{
    gl.viewport(0, 0, canvas.width, canvas.height);
    mat4.perspective(45, canvas.width / canvas.height, 0.1, 100.0, pMatrix);
    mat4.identity(mvMatrix);
    mat4.translate(mvMatrix, [0, 0, -2.0]);
}

function getMatrixUniforms(){
    pMatrixUniform = gl.getUniformLocation(glProgram, "uPMatrix");
    mvMatrixUniform = gl.getUniformLocation(glProgram, "uMVMatrix");
}

function setMatrixUniforms() {
    gl.uniformMatrix4fv(pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(mvMatrixUniform, false, mvMatrix);
}
```

Matrices [Model – View – Projection]

- ○ A diferencia de OpenGL (2.1) no existe manejo de matrices en WebGL ☹
- ○ Se debe hacer la gestión de las matrices y pasarlas como uniform
- ○ Existen librerías ya desarrolladas, p.e. http://glmatrix.net/
- ○ viewport()



`mat4.ortho(out, left, right, bottom, top, near, far)`

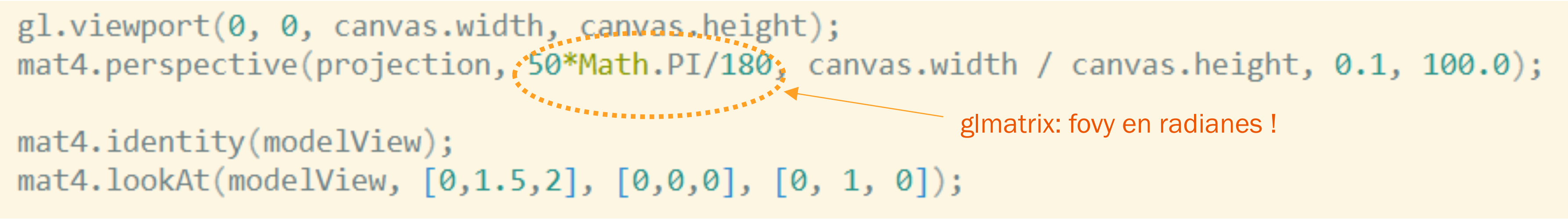
Generates a orthogonal projection matrix with the given bounds

`mat4.perspective(out, fovy, aspect, near, far)`

Generates a perspective projection matrix with the given bounds

`mat4.rotate(out, a, rad, axis)`

Rotates a mat4 by the given angle

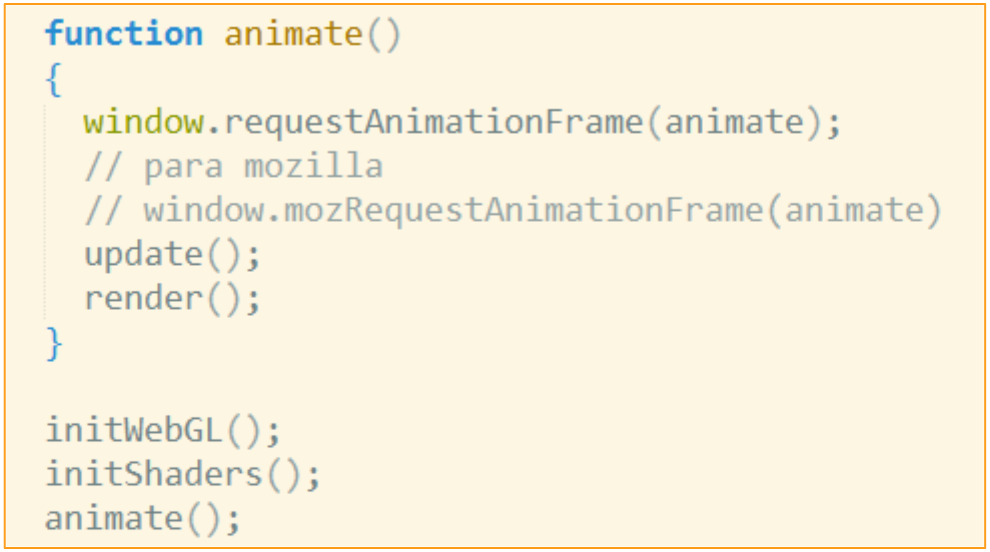


```
gl.viewport(0, 0, canvas.width, canvas.height);
mat4.perspective(projection, 50*Math.PI/180, canvas.width / canvas.height, 0.1, 100.0);

mat4.identity(modelView);
mat4.lookAt(modelView, [0,1.5,2], [0,0,0], [0, 1, 0]);
```

glmatrix: fovy en radianes !

Animación

- window.requestAnimationFrame(callback)
  - Encola el evento de animación con la *callback*
  - Se debe incluir dentro de la propia *callback*
  - Cada browser implementa su propia llamada: doc
    - Chrome requestAnimationFrame()
    - Firefox mozRequestAnimationFrame()



```
function animate()
{
   window.requestAnimationFrame(animate);
   // para mozilla
   // window.mozRequestAnimationFrame(animate)
   update();
   render();
}

initWebGL();
initShaders();
animate();
```

Bibliotecas

- El uso de WebGL directto
  - Es tedioso y difícil
  - Permite control de bajo nivel

- Bibliotecas sobre WebGL
  - Three.js
  - Babylon
  - GLGE
  - CopperLicht
  - blend4web
  - ……

Ejercicio cuboRGB

- http://personales.upv.es/rvivo/webgl/ejercicioCuboRGB.html
  1. Construir el VBO de colores por vértice
  2. Construir la *model view*
  3. Asociar los *uniforms* a la aplicación
  4. Modificar el *shader* de vértices
  5. Crear el bucle de animación