Universidad
Rey Juan Carlos
Escuela Técnica Superior
Ingeniería de Telecomunicación

Gráficos y visualización 3D

6. Texturas con WebGL

JOSÉ MIGUEL GUERRERO HERNÁNDEZ

EMAIL: JOSEMIGUEL.GUERRERO@URJC.ES

Índice de contenidos

Índice de contenidos

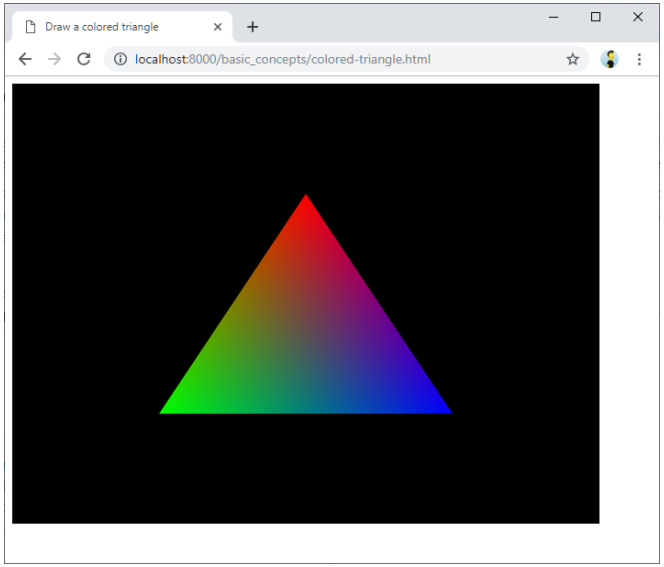
1. Introducción - Rasterización

- En temas anteriores hemos visto como se generan gráficos en WebGL mediante el procesado de dos programas GLSL:
  1. *Vertex shader*: Procesado de los **vértices** (coordenadas del modelo 3D)
  2. *Fragment shader*: Procesado de los **fragmentos** (píxeles en la pantalla)

1. Introducción - Rasterización

- Repaso del ejemplo `colored-triangle.html`

```
<script id="shaderVs" type="x-shader/x-vertex">
    attribute vec4 a_Position;
    attribute vec4 a_Color;
    varying highp vec4 v_Color;
    void main() {
        gl_Position = a_Position;
        v_Color = a_Color;
    }
</script>

<script id="shaderFs" type="x-shader/x-fragment">
    varying highp vec4 v_Color;
    void main() {
        gl_FragColor = v_Color;
    }
</script>
```



Las variables de tipo `attribute` (`a_Position` y `a_Color`) serán escritas desde JavaScript a través de un buffer (`gl.ARRAY_BUFFER`)

Las variable de tipo `varying` (`v_Color`) se calcula en el vertex shader por interpolación y se pasa el valor por fragmento al fragment shader

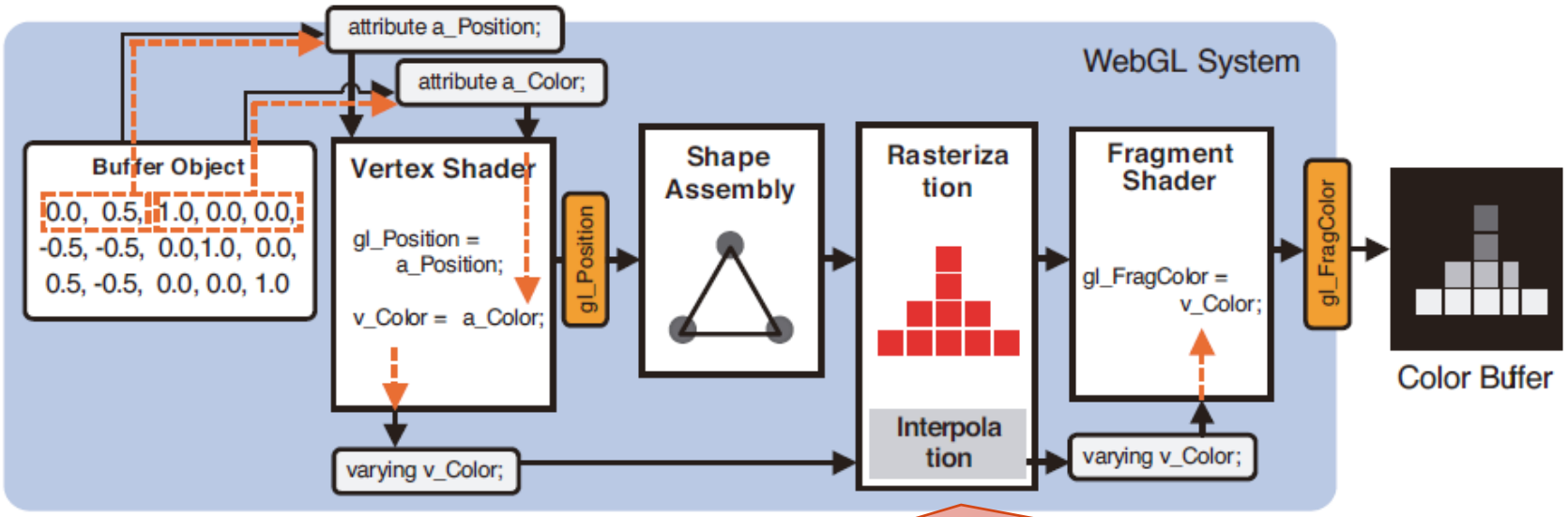
1. Introducción - Rasterización

- Repaso del ejemplo `colored-triangle.html`

```
function initVertexBuffers(gl) {
      var dim = 3;
      var vertices = new Float32Array([
          0, 0.5, 0, // Vertice #1
          -0.5, -0.5, 0, // Vertice #2
          0.5, -0.5, 0 // Vertice #3
      ]);
      var colors = new Float32Array([
          1.0, 0.0, 0.0, // Color #1 (red)
          0.0, 1.0, 0.0, // Color #2 (green)
          0.0, 0.0, 1.0, // Color #3 (blue)
      ]);
      // Create a buffer object for vertices and assign to a_Position variable
      gl.bindBuffer(gl.ARRAY_BUFFER, gl.createBuffer());
      gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
      var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
      gl.vertexAttribPointer(a_Position, dim, gl.FLOAT, false, 0, 0);
      gl.enableVertexAttribArray(a_Position);
      // Create colors buffer
      gl.bindBuffer(gl.ARRAY_BUFFER, gl.createBuffer());
      gl.bufferData(gl.ARRAY_BUFFER, colors, gl.STATIC_DRAW);
      var a_Color = gl.getAttribLocation(gl.program, 'a_Color');
      gl.vertexAttribPointer(a_Color, dim, gl.FLOAT, false, 0, 0);
      gl.enableVertexAttribArray(a_Color);

      return vertices.length / dim;
  }
```

1. Introducción - Rasterización

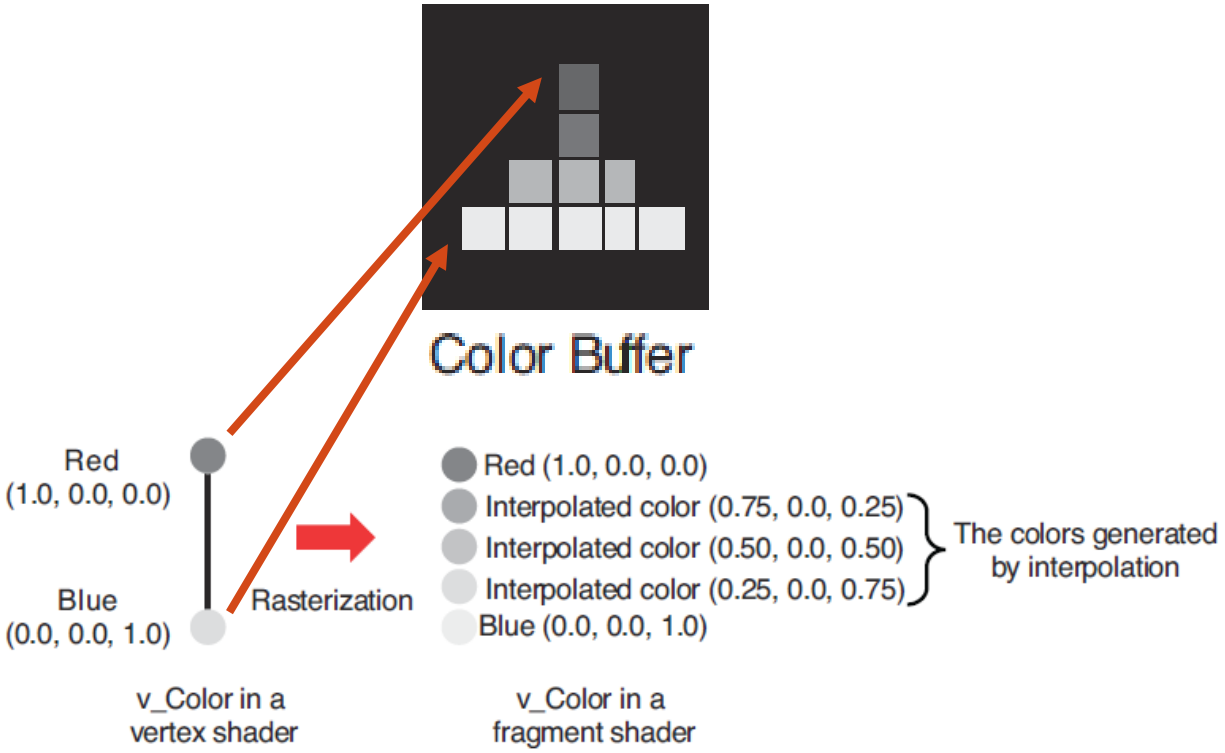
- Repaso del ejemplo `colored-triangle.html`



En el proceso de **rasterización**, la forma geométrica formada por los vértices y la primitiva de pintado se transforma en un conjunto de fragmentos (que se corresponderá con un conjunto de pixeles en la pantalla)

1. Introducción - Rasterización

- Repaso del ejemplo `colored-triangle.html`

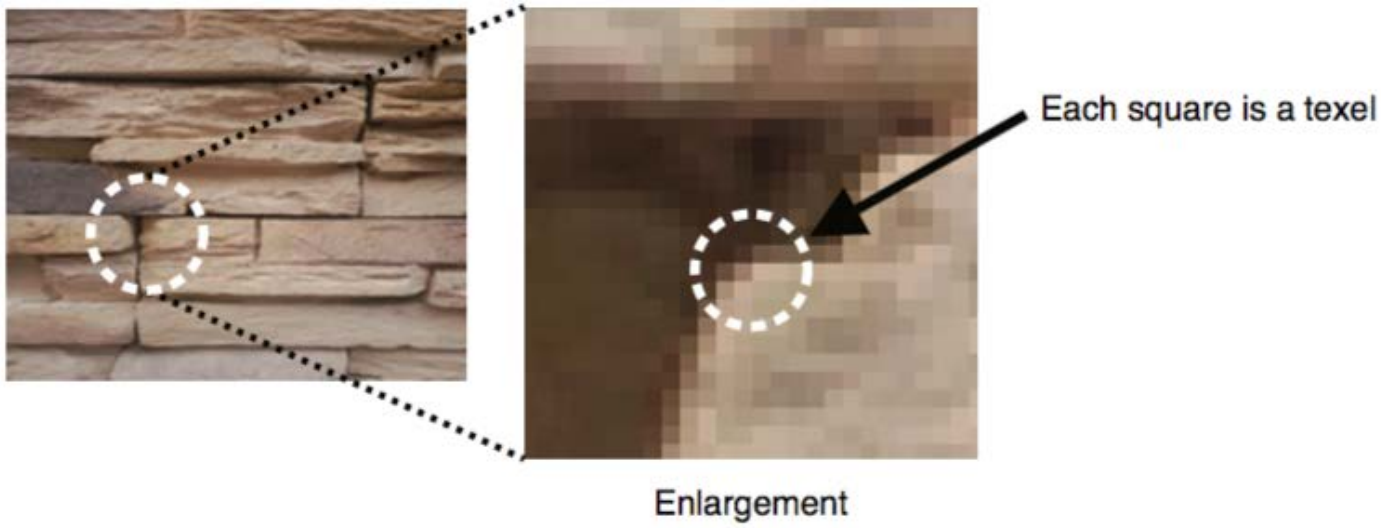


1. Introducción - Texturización

- En lugar de mediante colores planos, el proceso de rasterización se puede realizar mediante imágenes de trama (también llamadas mapa de bits o *rasterizadas*)
  - En este caso hablamos de **texturización**
- En algunas referencias (como la Wikipedia) se denomina rasterización al proceso por el cuál imágenes descritas como gráficos vectoriales se transforman en un conjunto de píxeles

1. Introducción - Texturización

- Los píxeles que forman la textura se denominan **texels** (elementos de textura) y cada uno codifica la información de su color en el formato RGB o RGBA



Índice de contenidos

2. Texturización en WebGL

- En WebGL, el proceso de texturización está formador por 4 pasos:
  1. Preparar la imagen que será mapeada en una forma geométrica
  2. Especificar el método de mapeo
  3. Cargar la imagen y configuración (filtrado)
  4. Extraer los pixeles de la imagen y asignarlos al fragmento correspondiente (texel) en el fragment shader

2. Texturización en WebGL

- La carga de la imagen (paso 1) se hará usando objetos de tipo `Image()` en JavaScript
- La carga de estas imágenes se hará de forma asíncrona (`image.src` … `image.onload`)
- Esta carga de imágenes está sujeta a la política del mismo origen (*same-origin policy*), por lo que nuestra página web deberá ser servida por un servidor (no se puede cargar directamente en local):

```
python -m SimpleHTTPServer
```

2. Texturización en WebGL

```
<script>
    function initTexturesAndDraw(gl) {
        var image = new Image();
        image.src = 'sky.jpg';
        image.onload = function () {
            loadTexture(gl, "u_Sampler", image, 0, true);
            drawScene(gl);
        };
    }
</script>
```

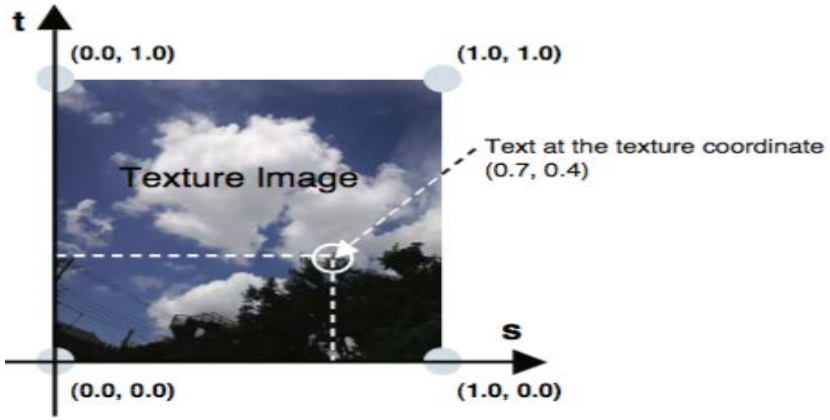
Ruta en donde se encuentra la imagen

Reserva de memoria para la imagen

Funciones para cargar la textura y dibujar la escena (carga y dibujado de los vertex)

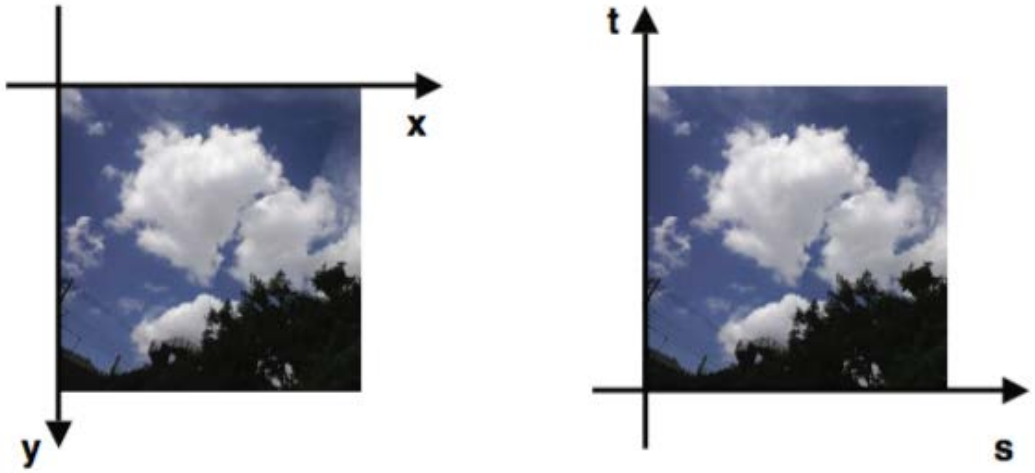
2. Texturización en WebGL

- El sistema de coordenadas de textura utilizado por WebGL es de dos dimensiones
- Para no confundir las coordenadas de textura con los ejes **x** e **y**, WebGL utiliza denomina las coordenadas de textura como **s** y **t**
- El área de la imagen original va desde (0.0, 0.0) hasta (1.0, 1.0), aunque se pueden usar valores de **s** y **t** fuera de ese rango



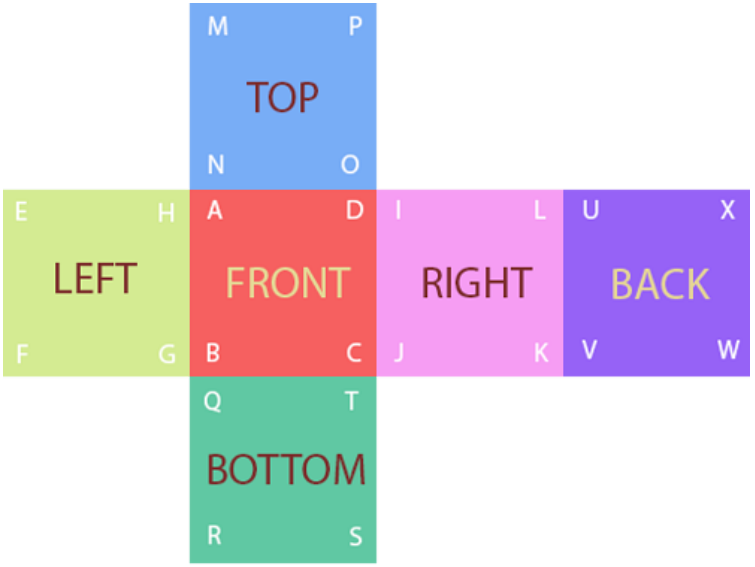
2. Texturización en WebGL

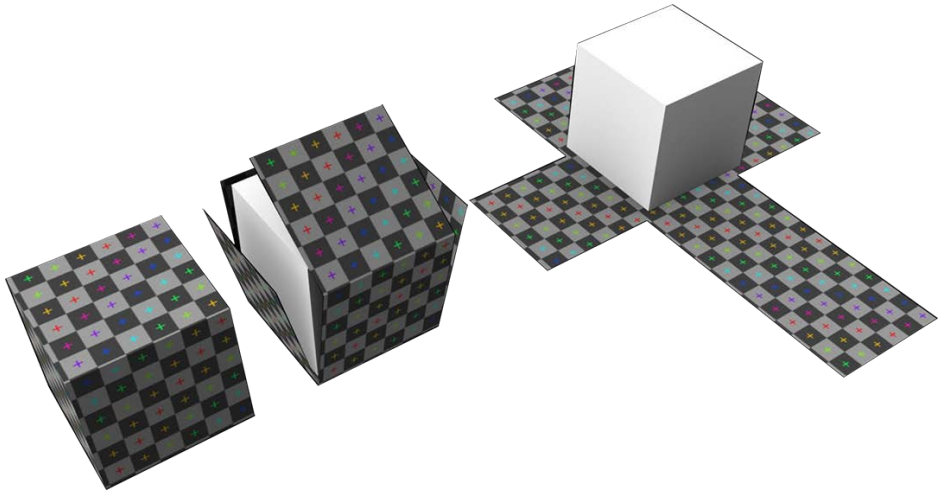
- El eje **t** del sistema de coordenadas de texturas en WebGL es el inverso con relación a las coordenadas usadas por algunos gráficos de trama (png, jpg, gif, bmp)
- Por esta razón, a veces, antes de usar ciertas imágenes de tramas en WebGL, hay que girar el eje Y de imagen



2. Texturización en WebGL

- Hay 2 tipos de textura en WebGL:
  1. `gl.TEXTURE_2D`: Basadas en una imagen de 4 lados
  2. `gl.TEXTURE_CUBE_MAP`: Basadas en un imagen con 6 caras diferentes





Índice de contenidos

3. Referencia API WebGL

```
<script>
    function initTexturesAndDraw(gl) {
        var image = new Image();
        image.src = 'sky.jpg';
        image.onload = function () {
            loadTexture(gl, "u_Sampler", image, 0, true);
            drawScene(gl);
        };
    }
    function loadTexture(gl, samplerUniform, image, unitNumber, flip) {
        // Flip the image's y axis
        gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, flip);
        // Activate and configure texture
        gl.activeTexture(gl.TEXTURE0 + unitNumber);
        gl.bindTexture(gl.TEXTURE_2D, gl.createTexture());
        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
        gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
        // Set texture to the sampler
        var fragmentSamplerUniform = gl.getUniformLocation(gl.program, samplerUniform);
        gl.uniform1i(fragmentSamplerUniform, unitNumber);
    }
</script>
```

Gira la imagen

Activa la textura X

Indica la forma de mapeo

Habilita la textura según su tipo

Asigna la imagen JavaScript a la textura creada

Como hemos hecho siempre con las variables a las que serán asignadas a los shaders, debemos preguntar al fragment shader dónde se encuentra y después asignarla

3. Referencia API WebGL

`gl.pixelStorei(pname, param)`

Perform the process defined by *pname* and *param* after loading an image.

Parameters	pname	Specifies any of the following:
	`gl.UNPACK_FLIP_Y_WEBGL`	Flips an image's Y-axis after loading the image. The default value is `false`.
	`gl.UNPACK_PREMULTIPLY_ALPHA_WEBGL`	Multiplies each component of RGB in an image by *A* in the image. The default value is `false`.
	param	Specifies none-zero (means `true`) or zero (means `false`). It must be specified in the integer.
Return value	None	
Errors	INVALID_ENUM	*pname* is none of these values.

3. Referencia API WebGL

`gl.activeTexture(texUnit)`

Make the texture unit specified by *texUnit* active.

Parameters	texUnit	Specifies the texture unit to be made active: `gl.TEXTURE0`, `gl.TEXTURE1`, ..., or `gl.TEXTURE7`. The tailing number indicates the texture unit number.
Return value	None	
Errors	INVALID_ENUM:	*texUnit* is none of these values

`gl.createTexture()`

Create a texture object to hold a texture image.

Parameters	None	
Return value	non-null	The newly created texture object.
	null	Failed to create a texture object.
Errors	None	

3. Referencia API WebGL

`gl.bindTexture(target, texture)`

Enable the texture object specified by *texture* and bind it to the *target*. In addition, if a texture unit was made active by `gl.activeTexture()`, the texture object is also bound to the texture unit.

Parameters	target	Specifies `gl.TEXTURE_2D` or `gl.TEXTURE_CUBE_MAP`.
	texture	Specifies the texture object to be bound.
Return value	None	
Errors	INVALID_ENUM	*target* is none of these values.

3. Referencia API WebGL

`gl.texParameteri(target, pname, param)`

Set the value specified by *param* to the texture parameter specified by *pname* in the texture object bound to *target*.

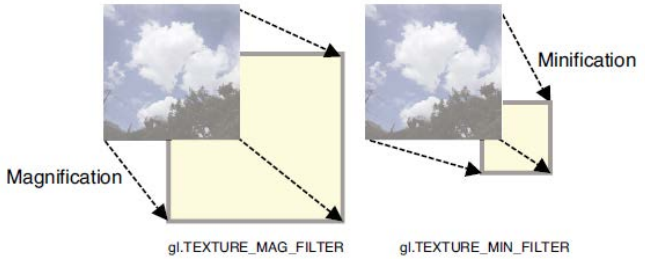
Parameters	target	Specifies `gl.TEXTURE_2D` or `gl.TEXTURE_CUBE_MAP`.
	pname	Specifies the name of the texture parameter (Table 5.3).
	param	Specifies the value set to the texture parameter *pname* (Table 5.4, Table 5.5).
Return value	None	
Errors	INVALID_ENUM	*target* is none of the preceding values
	INVALID_OPERATION	no texture object is bound to *target*

3. Referencia API WebGL

Table 5.3 Texture Parameters and Their Default Values

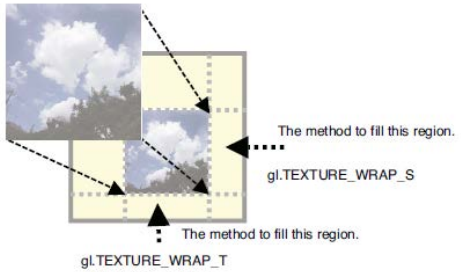
Texture Parameter	Description	Default Value
`gl.TEXTURE_MAG_FILTER`	Texture magnification	`gl.LINEAR`
`gl.TEXTURE_MIN_FILTER`	Texture minification	`gl.NEAREST_MIPMAP_LINEAR`
`gl.TEXTURE_WRAP_S`	Texture wrapping in s-axis	`gl.REPEAT`
`gl.TEXTURE_WRAP_T`	Texture wrapping in t-axis	`gl.REPEAT`

Mapear una textura en una figura con un área mayor



Mapear una textura en una figura con un área menor

Mapea la textura en el eje t (rellena la regiones arriba y abajo)



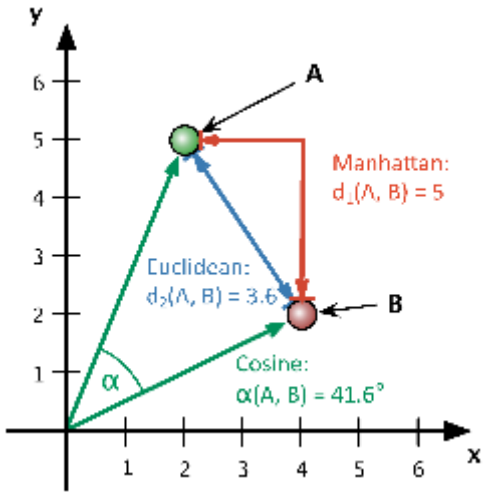
Mapea la textura en el eje s (rellena la regiones a la izquierda y a la derecha)

3. Referencia API WebGL

Table 5.4 Non-Mipmapped Values, Which Can be Specified to gl.TEXTURE_MAG_FILTER and gl.TEXTURE_MIN_FILTER[3]

Value	Description
`gl.NEAREST`	Uses the value of the texel that is nearest (in Manhattan distance) the center of the pixel being textured.
`gl.LINEAR`	Uses the weighted average of the four texels that are nearest the center of the pixel being textured. (The quality of the result is clearer than that of `gl.NEAREST`, but it takes more time.)

`gl.NEAREST` : utiliza el valor del texel más cercano al centro del pixel que se va a texturizar



`gl.LINEAR` : utiliza la media ponderada de los cuatro téxeles que están más cerca del centro del píxel que se va a texturizar

3. Referencia API WebGL

`gl.REPEAT` : usa la imagen de textura repetida

`gl.MIRRORED_REPEAT` : usa la imagen de textura reflejada repetida

`gl.CLAMP_TO_EDGE` : usa el color del borde de la imagen de textura

GL_REPEAT

GL_MIRRORED_REPEAT

GL_CLAMP_TO_EDGE

Table 5.5 Values that Can be Specified to gl.TEXTURE_WRAP_S and gl.TEXTURE_WRAP_T

Value	Description
`gl.REPEAT`	Use a texture image repeatedly
`gl.MIRRORED_REPEAT`	Use a texture image mirrored-repeatedly
`gl.CLAMP_TO_EDGE`	Use the edge color of a texture image

3. Referencia API WebGL

`gl.texImage2D(target, level, internalformat, format, type, image)`

Set the image specified by *image* to the texture object bound to *target*.

Parameters	target	Specifies `gl.TEXTURE_2D` or `gl.TEXTURE_CUBE_MAP`.
	level	Specified as 0. (Actually, this parameter is used for a MIPMAP texture, which is not covered in this book.)
	internalformat	Specifies the internal format of the image (Table 5.6).
	format	Specifies the format of the texel data. This must be specified using the same value as *internalformat*.
	type	Specifies the data type of the texel data (Table 5.7).
	image	Specifies an Image object containing an image to be used as a texture.
Return value	None	
Errors	INVALID_ENUM	*target* is none of the above values.
	INVALID_OPERATION	No texture object is bound to *target*

3. Referencia API WebGL

Table 5.6 The Format of the Texel Data

Format	Components in a Texel
`gl.RGB`	Red, green, blue
`gl.RGBA`	Red, green, blue, alpha
`gl.ALPHA`	(0.0, 0.0, 0.0, alpha)
`gl.LUMINANCE`	L, L, L, 1 L: Luminance
`gl.LUMINANCE_ALPHA`	L, L. L. alpha

Las componentes de luminancia (L) son percibidas como efectos de brillo para imágenes en escala de grises

Table 5.7 The Data Type of Texel Data

Type	Description
`gl.UNSIGNED_BYTE`	Unsigned byte format. Each color component has 1 byte.
`gl.UNSIGNED_SHORT_5_6_5`	RGB: Each component has 5, 6, and 5 bits, respectively.
`gl.UNSIGNED_SHORT_4_4_4_4`	RGBA: Each component has 4, 4, 4, and 4 bits, respectively.
`gl.UNSIGNED_SHORT_5_5_5_1`	RGBA: Each RGB component has 5 bits, and A has 1 bit.

Índice de contenidos

4. Referencia API GLSL ES

`vec4 texture2D(sampler2D sampler, vec2 coord)`

Retrieve a texel color at the texture coordinates specified by *coord* from the texture image specified by *sampler*.

Parameters	sampler	Specifies the texture unit number.
	coord	Specifies the texture coordinates.
Return value	The texel color (`vec4`) for the coordinates. The color format changes according to the *internalformat* specified by `gl.texImage2D()`. Table 5.9 shows the differences. If the texture image is not available for some reason, this function returns (0.0, 0.0, 0.0, 1.0).	

```
<script id="shaderVs" type="x-shader/x-vertex">
    attribute vec4 a_Position;
    attribute vec2 a_TexCoord;
    varying vec2 v_TexCoord;
    void main(void) {
        gl_Position = a_Position;
        v_TexCoord = a_TexCoord;
    }
</script>
```

```
<script id="shaderFs" type="x-shader/x-fragment">
    precision highp float;
    precision mediump float;
    uniform sampler2D u_Sampler;
    varying vec2 v_TexCoord;
    void main(void) {
        gl_FragColor = texture2D(u_Sampler, v_TexCoord);
    }
</script>
```

4. Referencia API GLSL ES

Table 5.8 Special Data Types for Accessing a Texture

Type	Description
sampler2D	Data type for accessing the texture bound to `gl.TEXTURE_2D`
samplerCube	Data type for accessing the texture bound to `gl.TEXTURE_CUBE_MAP`

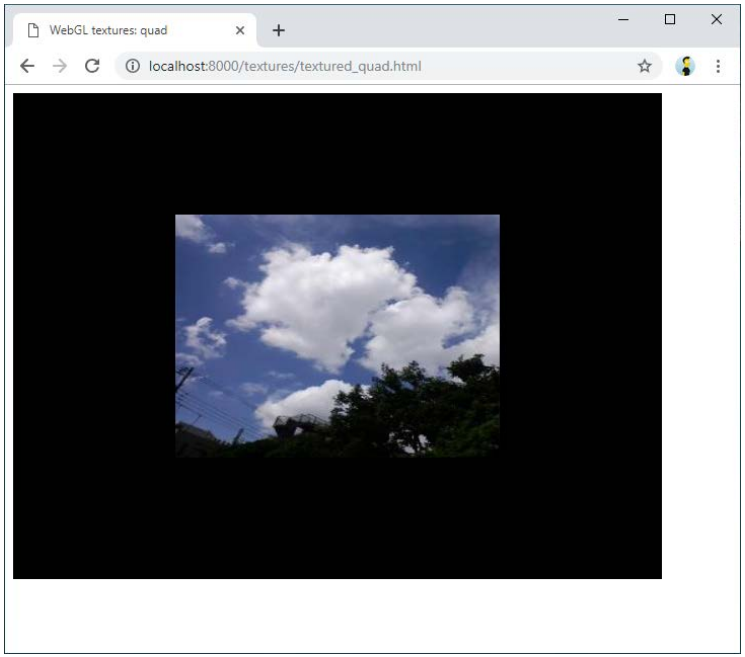
Table 5.9 Return Value of texture2D()

Internalformat	Return Value	
`gl.RGB`	(R, G, B, 1.0)	
`gl.RGBA`	(R, G, B, A)	
`gl.ALPHA`	(0.0, 0.0, 0.0, A)	
`gl.LUMINANCE`	(L, L, L, 1.0)	L indicates luminance
`gl.LUMINANCE_ALPHA`	(L, L, L, A)	

Índice de contenidos

5. Ejemplo: cuadrado con textura

```
<!DOCTYPE html>
<html>
<head><title>WebGL textures: quad</title></head>
<script id="shaderVs" type="x-shader/x-vertex">
    attribute vec4 a_Position;
    attribute vec2 a_TexCoord;
    varying vec2 v_TexCoord;
    void main(void) {
        gl_Position = a_Position;
        v_TexCoord = a_TexCoord;
    }
</script>
<script id="shaderFs" type="x-shader/x-fragment">
    precision highp float;
    precision mediump float;
    uniform sampler2D u_Sampler;
    varying vec2 v_TexCoord;
    void main(void) {
        gl_FragColor = texture2D(u_Sampler, v_TexCoord);
    }
</script>

<script>
    // ...
</script>

<body onload="init()">
    <canvas id="myCanvas" width="640" height="480"></canvas>
</body>
</html>
```



5. Ejemplo: cuadrado con textura

```
<!DOCTYPE html>
<html>
<head><title>WebGL textures: quad</title></head>
<script id="shaderVs" type="x-shader/x-vertex">
    attribute vec4 a_Position;
    attribute vec2 a_TexCoord;
    varying vec2 v_TexCoord;
    void main(void) {
        gl_Position = a_Position;
        v_TexCoord = a_TexCoord;
    }
</script>
<script id="shaderFs" type="x-shader/x-fragment">
    precision highp float;
    precision mediump float;
    uniform sampler2D u_Sampler;
    varying vec2 v_TexCoord;
    void main(void) {
        gl_FragColor = texture2D(u_Sampler, v_TexCoord);
    }
</script>

<script>
    // ...
</script>

<body onload="init()">
    <canvas id="myCanvas" width="640" height="480"></canvas>
</body>
</html>
```

Se reciben las coordenadas de la textura mediante la variable `a_TexCoord` de tipo `attribute`

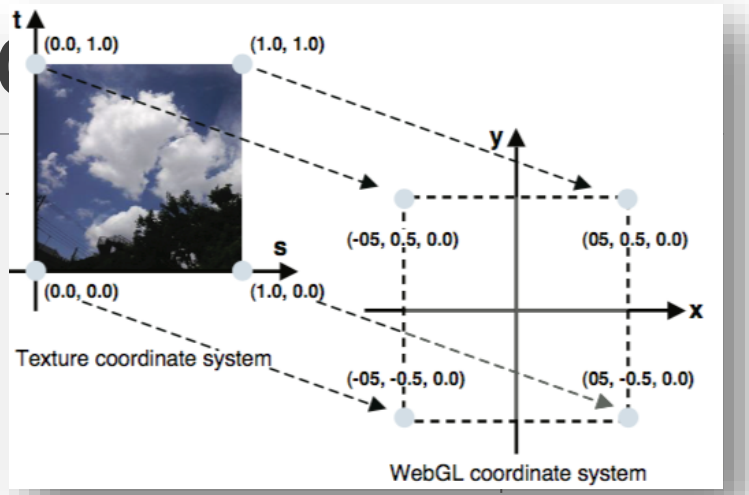
La textura se configura desde JavaScript (variable `u_Sampler`) y se procesa en el fragment shader usando las coordenadas calculas por vértice (`v_TexCoord`)

5. Ejemplo: cuadrado con textura

```
<script>
    function init() {
        // Get canvas object from the DOM
        var canvas = document.getElementById("myCanvas");
        // Init WebGL context
        var gl = canvas.getContext("webgl");
        if (!gl) {
            console.log("Failed to get the rendering context for WebGL");
            return;
        }
        // Init shaders
        var vs = document.getElementById('shaderVs').innerHTML;
        var fs = document.getElementById('shaderFs').innerHTML;
        if (!initShaders(gl, vs, fs)) {
            console.log('Failed to intialize shaders.');
            return;
        }

        // Init vertices and texture coordinates
        initVertexBuffers(gl);

        // Set clear color (black)
        gl.clearColor(0.0, 0.0, 0.0, 1.0);

        // Init textures
        initTexturesAndDraw(gl);
    }

</script>
```

El método `init()` es muy parecido a lo que hemos hecho en ejemplos anteriores, con la novedad de la función `initTexturesAndDraw()`, en la que se inicializarán las texturas y se dibujará la escena

5. Ejemplo: cuadra




```
<script>
   function initVertexBuffers(gl) {
        // Vertices (x, y) and texture coordinates (s, t)
        var verticesAndTextures = [
            -0.5,  0.5,   0.0, 1.0, // top-left image corner
            -0.5, -0.5,   0.0, 0.0, // bottom-left image corner
             0.5,  0.5,   1.0, 1.0, // top-right image corner
             0.5, -0.5,   1.0, 0.0, // bottom-right image corner
        ];

        gl.bindBuffer(gl.ARRAY_BUFFER, gl.createBuffer());
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(verticesAndTextures), gl.STATIC_DRAW);

        var vertexPositionAttribute = gl.getAttribLocation(gl.program, "a_Position");
        gl.enableVertexAttribArray(vertexPositionAttribute);
        gl.vertexAttribPointer(vertexPositionAttribute, 2, gl.FLOAT, false, 4 * 4, 0);

        var vertexTextureAttribute = gl.getAttribLocation(gl.program, "a_TexCoord");
        gl.enableVertexAttribArray(vertexTextureAttribute);
        gl.vertexAttribPointer(vertexTextureAttribute, 2, gl.FLOAT, false, 4 * 4, 4 * 2);
    }
</script>
```

El método `initVertexBuffers()` inicializa las coordenadas de los vértices (x, y) y las texturas (s, t) usando un buffer (`gl.ARRAY_BUFFER`) que será escrito en la variable `a_Position` del vertex shader

5. Ejemplo: cuadrado con textura

```
<script>
    function initTexturesAndDraw(gl) {
        var image = new Image();
        image.src = 'sky.jpg';
        image.onload = function () {
            loadTexture(gl, "u_Sampler", image, 0, true);
            drawScene(gl);
        };
    }
    function loadTexture(gl, samplerUniform, image, unitNumber, flip) {
        // Flip the image's y axis
        gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, flip);
        // Activate and configure texture
        gl.activeTexture(gl.TEXTURE0 + unitNumber);
        gl.bindTexture(gl.TEXTURE_2D, gl.createTexture());
        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
        gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
        // Set texture to the sampler
        var fragmentSamplerUniform = gl.getUniformLocation(gl.program, samplerUniform);
        gl.uniform1i(fragmentSamplerUniform, unitNumber);
    }
    function drawScene(gl) {
        gl.clear(gl.COLOR_BUFFER_BIT);
        gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
    }
</script>
```

Carga asíncrona de la imagen

Giro de la imagen sobre el eje Y

Configuración de la textura

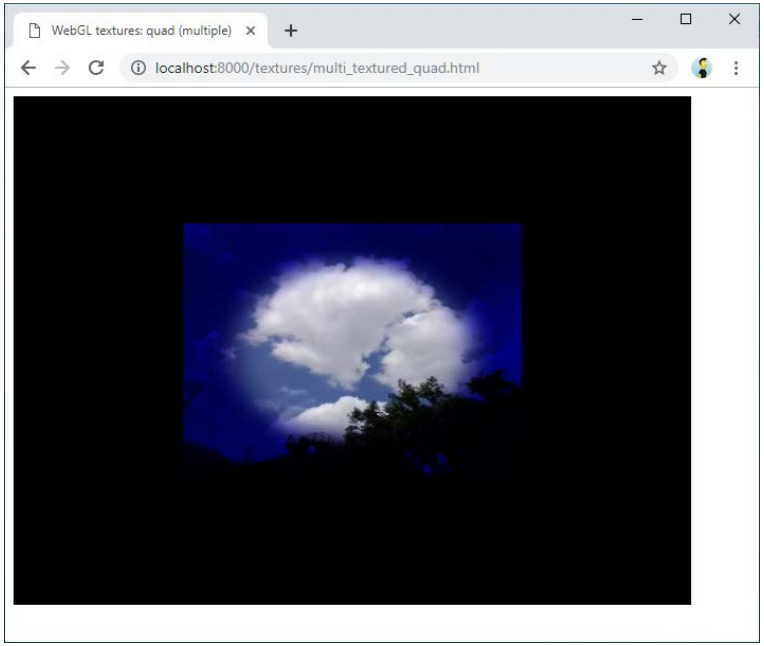
Carga de la textura en el fragment shader

Dibujamos 4 vértices usando la primitiva `gl.TRIANGLE_STRIP`

Índice de contenidos

6. Ejemplo: cuadrado con textura múltiple

```
<!DOCTYPE html>
<html>
<head><title>WebGL textures: quad (multiple)</title></head>
<script id="shaderVs" type="x-shader/x-vertex">
    attribute vec4 a_Position;
    attribute vec2 a_TexCoord;
    varying vec2 v_TexCoord;
    void main(void) {
        gl_Position = a_Position;
        v_TexCoord = a_TexCoord;
    }
</script>
<script id="shaderFs" type="x-shader/x-fragment">
 precision highp float;
    precision mediump float;
    uniform sampler2D u_Sampler0;
    uniform sampler2D u_Sampler1;
    varying vec2 v_TexCoord;
    void main(void) {
        vec4 color0 = texture2D(u_Sampler0, v_TexCoord);
        vec4 color1 = texture2D(u_Sampler1, v_TexCoord);
        gl_FragColor = color0 * color1;
    }
</script>
<script>
    // ...
</script>
<body onload="init()">
    <canvas id="myCanvas" width="640" height="480"></canvas>
</body>
</html>
```



En este ejemplo el fragment shader está preparado para recibir dos texturas

tiple

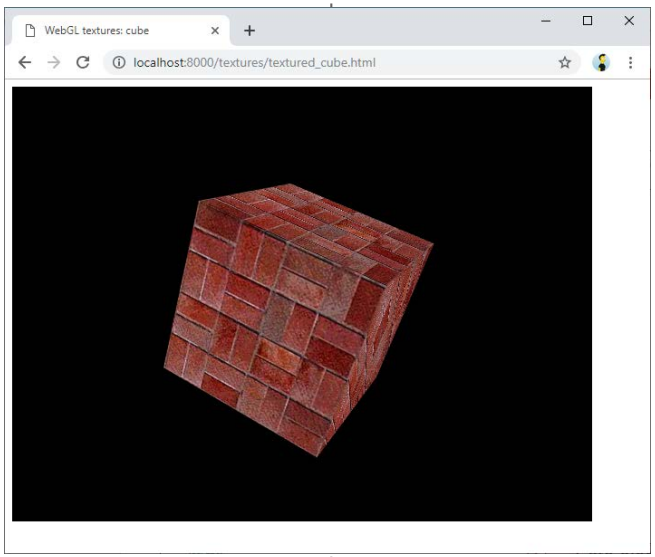
```
<script>
    function initTexturesAndDraw(gl) {
        var image0 = new Image();
        image0.src = 'sky.jpg';
        var image1 = new Image();
        image1.src = 'circle.gif';
        image0.onload = function () {
            loadTexture(gl, "u_Sampler0", image0, 0, true);
            image1.onload = function () {
                loadTexture(gl, "u_Sampler1", image1, 1, true);
                drawScene(gl);
            };
        };
    }

    function loadTexture(gl, samplerUniform, image, unitNumber, flip) {
        // Flip the image's y axis
        gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, flip);

        // Activate and configure texture
        gl.activeTexture(gl.TEXTURE0 + unitNumber);
        gl.bindTexture(gl.TEXTURE_2D, gl.createTexture());
        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
        gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);

        // Set texture to the sampler
        var fragmentSamplerUniform = gl.getUniformLocation(gl.program, samplerUniform);
        gl.uniform1i(fragmentSamplerUniform, unitNumber);
    }
</script>
```

Partiendo del ejemplo anterior, usamos diferentes imágenes y diferentes unidades de textura (`gl.TEXTURE0` y `gl.TEXTURE1`)

Índice de contenidos

7. Ejemplo: cubo con textura

```
<!DOCTYPE html>
<html>
<head>
    <title>WebGL textures: cube</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/gl-matrix/2.8.1/gl-matrix-min.js"></script>
    <script id="shaderVs" type="x-shader/x-vertex">
        attribute vec4 a_Position;
        attribute vec2 a_TexCoord;
        varying vec2 v_TexCoord;

        uniform mat4 u_pMatrix;
        uniform mat4 u_vMatrix;
        uniform mat4 u_mvMatrix;
        varying highp vec4 v_Color;
        void main() {
            gl_Position = u_pMatrix * u_vMatrix * u_mvMatrix * a_Position;
            v_TexCoord = a_TexCoord;
        }
    </script>
    <script id="shaderFs" type="x-shader/x-fragment">
        precision highp float;
        precision mediump float;
        uniform sampler2D u_Sampler;
        varying vec2 v_TexCoord;

        void main(void) {
            gl_FragColor = texture2D(u_Sampler, v_TexCoord);
        }
    </script>
    <script>
       // ...
    </script>
<body onload="init()">
    <canvas id="myCanvas" width="640" height="480"></canvas>
</body>
</html>
```



Este ejemplo es una mezcla del cubo en perspectiva visto en el tema anterior usando texturas para cada una de las caras

7. Ejemplo: cubo con textura

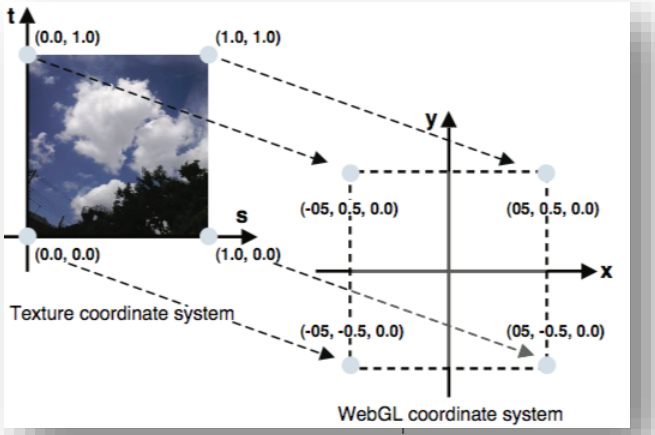
```
<script>
 function initTexturesAndDraw(gl) {
        var image = new Image();
        image.src = 'sky.jpg';
        image.onload = function () {
            loadTexture(gl, "u_Sampler", image, 0, true);
            drawScene(gl);
        };
    }

    function loadTexture(gl, samplerUniform, image, unitNumber, flip) {
        // Flip the image's y axis
        gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, flip);

        // Activate and configure texture
        gl.activeTexture(gl.TEXTURE0 + unitNumber);
        gl.bindTexture(gl.TEXTURE_2D, gl.createTexture());
        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
        gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);

        // Set texture to the sampler
        var fragmentSamplerUniform = gl.getUniformLocation(gl.program, samplerUniform);
        gl.uniform1i(fragmentSamplerUniform, unitNumber);
    }
</script>
```

Usamos la misma lógica que en el ejemplo anterior para inicializar las texturas y dibujar la escena, cambiando únicamente la imagen usada como textura

7. Ejemplo: cubo con textura

```
<script>
    function initVertexShader(gl) {
        var vertexesAndTextures = [ // Vertexes and texturecoordinates  (X, Y, Z, S, T)

            -0.5, -0.5, -0.5,        0.0, 0.0,
             0.5, -0.5, -0.5,        1.0, 0.0,
             0.5,  0.5, -0.5,        1.0, 1.0,
            -0.5,  0.5, -0.5,        0.0, 1.0,

            -0.5, -0.5, 0.5,         0.0, 1.0,
             0.5, -0.5, 0.5,         0.0, 0.0,
             0.5,  0.5, 0.5,         1.0, 0.0,
            -0.5,  0.5, 0.5,         1.0, 1.0,

            -0.5, -0.5, -0.5,        0.0, 0.0,
            -0.5,  0.5, -0.5,        0.0, 1.0,
            -0.5,  0.5,  0.5,        1.0, 1.0,
            -0.5, -0.5,  0.5,        1.0, 0.0,

             0.5, -0.5, -0.5,        0.0, 0.0,
             0.5,  0.5, -0.5,        0.0, 1.0,
             0.5,  0.5,  0.5,        1.0, 1.0,
             0.5, -0.5,  0.5,        1.0, 0.0,

            -0.5, -0.5, -0.5,        0.0, 0.0,
            -0.5, -0.5,  0.5,        0.0, 1.0,
             0.5, -0.5,  0.5,        1.0, 1.0,
             0.5, -0.5, -0.5,        1.0, 0.0,

            -0.5, 0.5, -0.5,         0.0, 1.0,
            -0.5, 0.5,  0.5,         0.0, 0.0,
             0.5, 0.5,  0.5,         1.0, 0.0,
             0.5, 0.5, -0.5,         1.0, 1.0,
        ];
        // ...
```

En este ejemplo se definen las posiciones de los vértices (x, y, z) en el mismo array las coordenadas de texturas en el sistema de coordenadas de WebGL por cada una de las caras



7. Ejemplo: cubo con textura

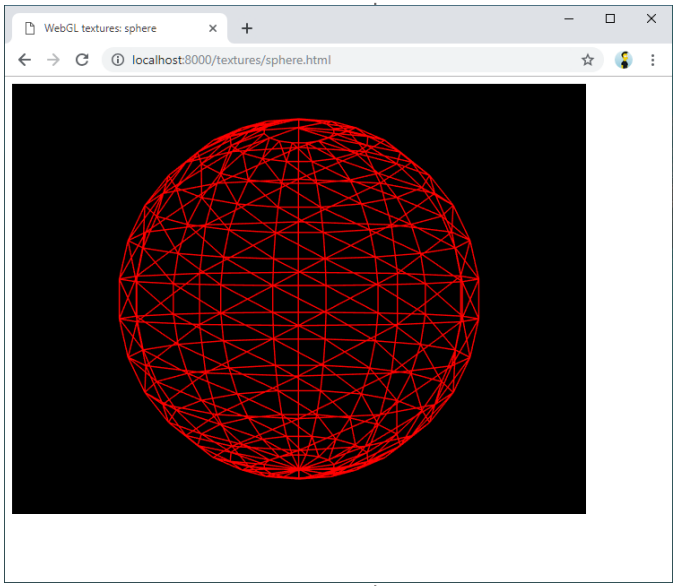
```
<script>
       function drawScene() {
            // Clear
            gl.clear(gl.COLOR_BUFFER_BIT);

            // Rotate
            var mvMatrix = mat4.fromRotation(mat4.create(), count, [0.5, 0.5, 0.5]);
            var uMvMatrix = gl.getUniformLocation(gl.program, "u_mvMatrix");
            gl.uniformMatrix4fv(uMvMatrix, false, mvMatrix);

            // Draw
            gl.drawElements(gl.TRIANGLES, 6 * 2 * 3, gl.UNSIGNED_SHORT, 0);

            // Call drawScene again in the next browser repaint
            count += 0.01;
            requestAnimationFrame(drawScene);
        }
</script>
```

La función `drawScene()` dibuja la escena, previo cálculo de la matriz de transformación (para girar el cubo) y matriz de vista (para la proyección en perspectiva)

Índice de contenidos

8. Ejemplo: esfera

```
<!DOCTYPE html>
<html>
 <title>Sphere</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/gl-matrix/2.8.1/gl-matrix-min.js"></script>
    <script id="shaderVs" type="x-shader/x-vertex">
        attribute vec4 a_Position;

        uniform mat4 u_pMatrix;
        uniform mat4 u_vMatrix;
        uniform mat4 u_mvMatrix;
        void main() {
            gl_Position = u_pMatrix * u_vMatrix * u_mvMatrix * a_Position;
        }
    </script>
    <script id="shaderFs" type="x-shader/x-fragment">
        void main(void) {
            gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
        }
    </script>
    <script>
        // ...
    </script>
<body onload="init()">
    <canvas id="myCanvas" width="640" height="480"></canvas>
</body>
</html>
```

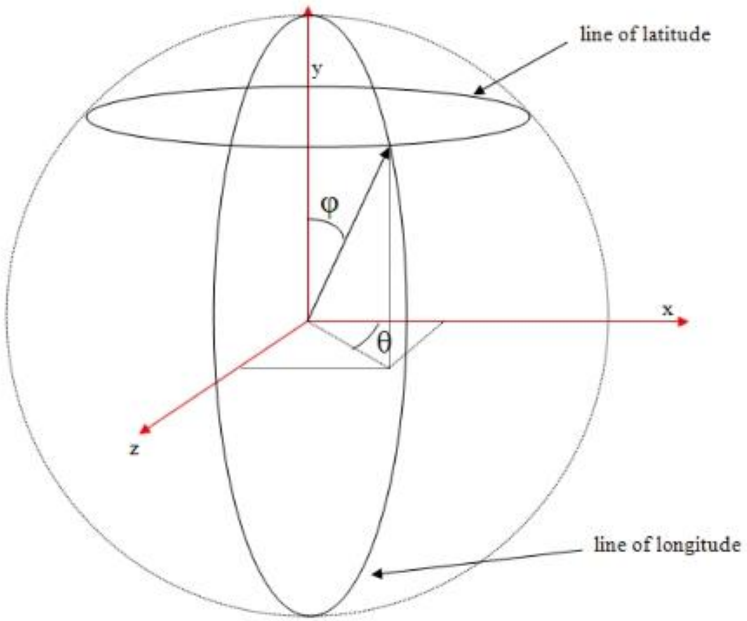


8. Ejemplo: esfera

- La superficie de una esfera se aproxima mediante polígonos
- Hay varios métodos para realizar esta aproximación, nosotros vamos a usar el método en el que se convierten coordenadas polares (r, θ, ϕ) a cartesianas (x, y, z)

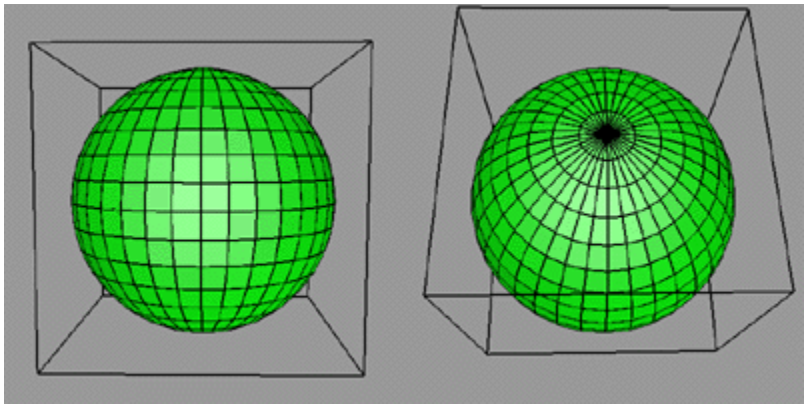
8. Ejemplo: esfera

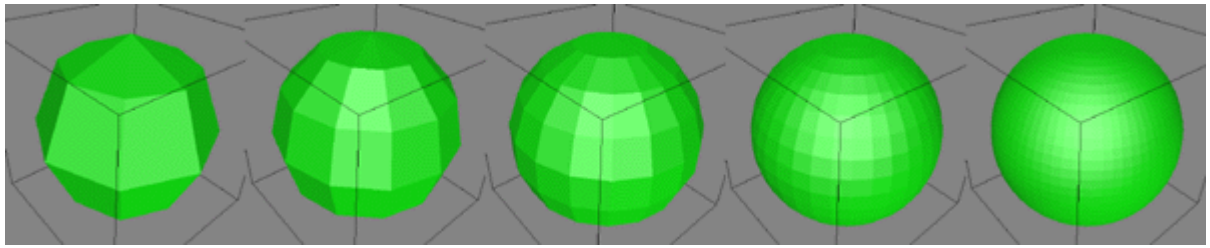
- La idea consiste en dividir la esfera en un número de secciones horizontales (latitud) y verticales (longitud)



8. Ejemplo: esfera

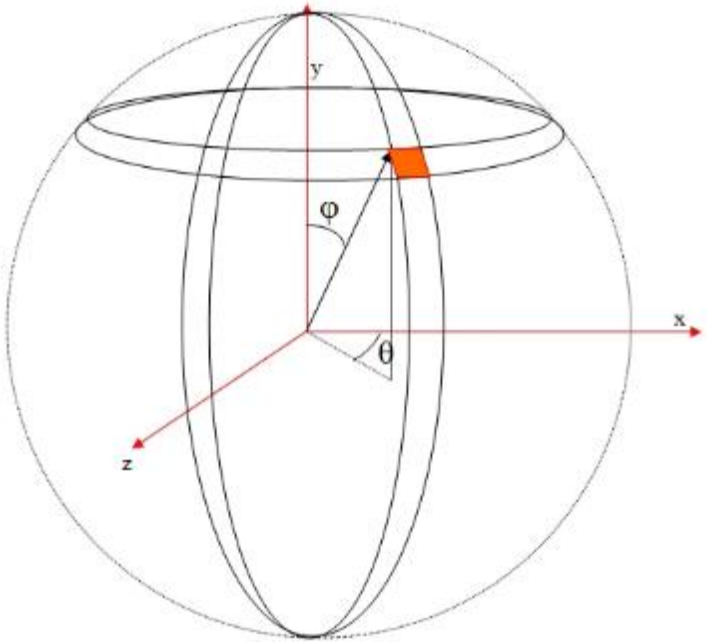
- En función del número de longitudes y latitudes, la esfera será más o menos aproximada





8. Ejemplo: esfera

- Dado un radio r, variamos los ángulos θ y ϕ
- Aproximamos una parte de la superficie de la esfera mediante un área de 4 lados



8. Ejemplo: esfera

- La conversión de coordenadas polares (r, θ, ϕ) a cartesianas (x, y, z) se hace usando estas fórmulas:

$$x = \sin\varphi \cdot \sin\theta$$
$$y = \cos\theta$$
$$z = \cos\varphi \cdot \sin\theta$$

$$-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$$
$$0 \leq \varphi \leq 2\pi$$

```
<script>
        var gl;
        var count = 0.0;
        const LATITUDE_BANDS = 15;
        const LONGITUDE_BANDS = 15;
        const RADIUS = 1;

         function initVertexShader(gl) {
            // Vertexes coordinates
            var vertexes = [];
            for (var i = 0; i <= LATITUDE_BANDS; i++) {
                var theta = i * Math.PI / LATITUDE_BANDS;
                var sinTheta = Math.sin(theta);
                var cosTheta = Math.cos(theta);

                for (var j = 0; j <= LONGITUDE_BANDS; j++) {
                    var phi = j * 2 * Math.PI / LONGITUDE_BANDS;
                    var sinPhi = Math.sin(phi);
                    var cosPhi = Math.cos(phi);

                    var x = sinPhi * sinTheta;
                    var y = cosTheta;
                    var z = cosPhi * sinTheta;

                    vertexes.push(RADIUS * x);
                    vertexes.push(RADIUS * y);
                    vertexes.push(RADIUS * z);
                }
            }

            // ...
</script>
```

Implementamos esta conversión de coordenadas en JavaScript usando dos bucles anidados en función de 3 constantes: RADIUS, LATITUDE_BANDS y LONGITUDE_BANDS

8. Ejemplo: esfera

```
<script>
            // ...
            // Indexes
            var indexes = [];
            for (var i = 0; i < LATITUDE_BANDS; i++) {
                for (var j = 0; j < LONGITUDE_BANDS; j++) {
                    var first = i * (LONGITUDE_BANDS + 1) + j;
                    var second = first + LONGITUDE_BANDS + 1;

                    indexes.push(first);
                    indexes.push(second);
                    indexes.push(first + 1);

                    indexes.push(second);
                    indexes.push(second + 1);
                    indexes.push(first + 1);
                }
            }
            // Write a_Position and using gl.ARRAY_BUFFER
            gl.bindBuffer(gl.ARRAY_BUFFER, gl.createBuffer());
            gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexes), gl.STATIC_DRAW);

            var vertexPositionAttribute = gl.getAttribLocation(gl.program, "a_Position");
            gl.enableVertexAttribArray(vertexPositionAttribute);
            gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);

            // Write indexes in gl.ELEMENT_ARRAY_BUFFER
            gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, gl.createBuffer());
            gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(indexes), gl.STATIC_DRAW);
        }
</script>
```

Se calculan los índices de los vértices iterando nuevamente las longitudes y latitudes

Como en otros ejemplos, usamos el buffer `gl.ARRAY_BUFFER` para escribir las coordenadas de los vértices y `gl.ELEMENT_ARRAY_BUFFER` para los índices

8. Ejemplo: esfera

```
<script>
        function drawScene() {
            // Clear
            gl.clear(gl.COLOR_BUFFER_BIT);

            // Rotate
            var mvMatrix = mat4.fromRotation(mat4.create(), count, [0, 0.5, 0]);
            var uMvMatrix = gl.getUniformLocation(gl.program, "u_mvMatrix");
            gl.uniformMatrix4fv(uMvMatrix, false, mvMatrix);

            // Draw
            gl.drawElements(gl.LINE_STRIP, LATITUDE_BANDS * LONGITUDE_BANDS * 3 * 2, gl.UNSIGNED_SHORT, 0);

            // Call drawScene again in the next browser repaint
            count += 0.01;
            requestAnimationFrame(drawScene);
        }
</script>
```

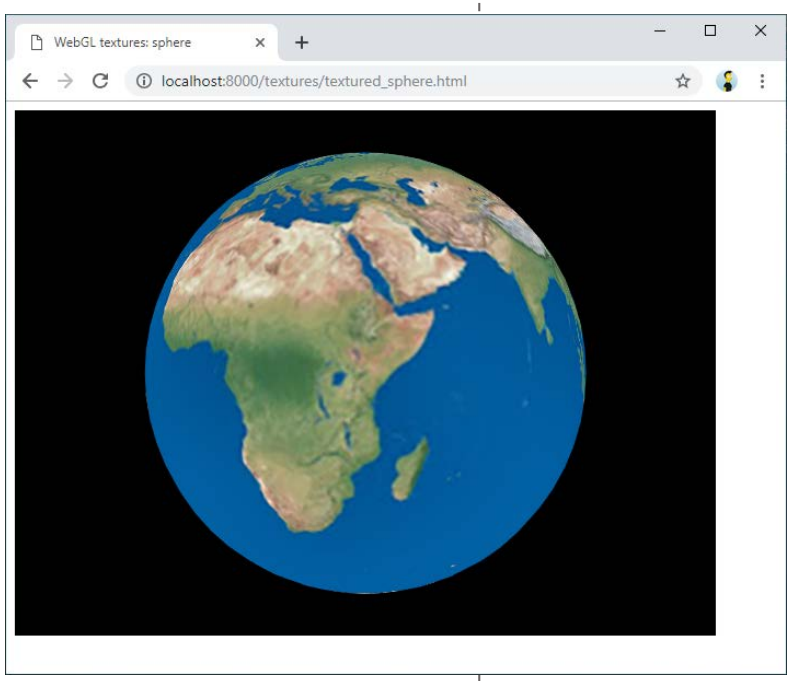
Número de cuadrados que hay, formados por 2 triángulos de 3 vértices cada uno

Por último se dibuja la escena usando la primitiva `gl.LINE_STRIP`. El número de vértices será función de `LATITUDE_BANDS` y `LONGITUDE_BANDS`

Índice de contenidos

9. Ejemplo: esfera con textura

```
<!DOCTYPE html>
<html>
    <title>WebGL textures: sphere</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/gl-matrix/2.8.1/gl-matrix-min.js"></script>
    <script id="shaderVs" type="x-shader/x-vertex">
        attribute vec4 a_Position;
        attribute vec2 a_TexCoord;
        varying vec2 v_TexCoord;

        uniform mat4 u_pMatrix;
        uniform mat4 u_vMatrix;
        uniform mat4 u_mvMatrix;
        varying highp vec4 v_Color;
        void main() {
            gl_Position = u_pMatrix * u_vMatrix * u_mvMatrix * a_Position;
            v_TexCoord = a_TexCoord;
        }
    </script>
    <script id="shaderFs" type="x-shader/x-fragment">
        precision highp float;
        precision mediump float;
        uniform sampler2D u_Sampler;
        varying vec2 v_TexCoord;

        void main(void) {
            gl_FragColor = texture2D(u_Sampler, v_TexCoord);
        }
    </script>
    <script>
        // ...
    </script>
<body onload="init()">
    <canvas id="myCanvas" width="640" height="480"></canvas>
</body>
</html>
```



```
<script>
        function initVertexShader(gl) {
           // Vertexes and textures coordinates
           var vertexesAndTextures = [];
           for (var i = 0; i <= LATITUDE_BANDS; i++) {
               var theta = i * Math.PI / LATITUDE_BANDS;
               var sinTheta = Math.sin(theta);
               var cosTheta = Math.cos(theta);

               for (var j = 0; j <= LONGITUDE_BANDS; j++) {
                   var phi = j * 2 * Math.PI / LONGITUDE_BANDS;
                   var sinPhi = Math.sin(phi);
                   var cosPhi = Math.cos(phi);

                   var x = sinPhi * sinTheta;
                   var y = cosTheta;
                   var z = cosPhi * sinTheta;

                   vertexesAndTextures.push(RADIUS * x);
                   vertexesAndTextures.push(RADIUS * y);
                   vertexesAndTextures.push(RADIUS * z);

                   var u = 1 - (j / LONGITUDE_BANDS);
                   var v = 1 - (i / LATITUDE_BANDS);

                   vertexesAndTextures.push(u);
                   vertexesAndTextures.push(v);
               }
           }

           // ...

</script>
```

Partiendo del ejemplo anterior, añadimos las coordenadas de las texturas dentro del array de las coordenadas de las texturas

```
<script>
        function initVertexShader(gl) {
            // ...

            // Indexes
            var indexes = [];
            for (var i = 0; i < LATITUDE_BANDS; i++) {
                for (var j = 0; j < LONGITUDE_BANDS; j++) {
                    var first = i * (LONGITUDE_BANDS + 1) + j;
                    var second = first + LONGITUDE_BANDS + 1;

                    indexes.push(first);
                    indexes.push(second);
                    indexes.push(first + 1);

                    indexes.push(second + 1);
                    indexes.push(first + 1);
                    indexes.push(second);
                }
            }

            // Write a_Position and a_TexCoord using gl.ARRAY_BUFFER
            gl.bindBuffer(gl.ARRAY_BUFFER, gl.createBuffer());
            gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexesAndTextures), gl.STATIC_DRAW);

            var vertexPositionAttribute = gl.getAttribLocation(gl.program, "a_Position");
            gl.enableVertexAttribArray(vertexPositionAttribute);
            gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 4 * (3 + 2), 0);

            var vertexColorAttribute = gl.getAttribLocation(gl.program, "a_TexCoord");
            gl.enableVertexAttribArray(vertexColorAttribute);
            gl.vertexAttribPointer(vertexColorAttribute, 2, gl.FLOAT, false, 4 * (3 + 2), 4 * 3);

            // Write indexes in gl.ELEMENT_ARRAY_BUFFER
            gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, gl.createBuffer());
            gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(indexes), gl.STATIC_DRAW);
        }
</script>
```

Las coordenadas de las texturas se escriben en el buffer `gl.ARRAY_BUFFER`

9. Ejemplo: esfera con textura

```
<script>
        function initTexturesAndDraw(gl) {
            var image = new Image();
            image.src = 'earth.png';
            image.onload = function () {
                loadTexture(gl, "u_Sampler", image, 0, false);
                drawScene();
            };
        }

        function loadTexture(gl, samplerUniform, image, unitNumber, flip) {
            // Flip the image's y axis
            gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, flip);

            // Activate and configure texture
            gl.activeTexture(gl.TEXTURE0 + unitNumber);
            gl.bindTexture(gl.TEXTURE_2D, gl.createTexture());
            gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
            gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);

            // Set texture to the sampler
            var fragmentSamplerUniform = gl.getUniformLocation(gl.program, samplerUniform);
            gl.uniform1i(fragmentSamplerUniform, unitNumber);
        }
</script>
```

Usamos la misma lógica que en ejemplos anteriores para inicializar las texturas y dibujar la escena, cambiando únicamente la imagen usada como textura

9. Ejemplo: esfera con textura

```
<script>
        function drawScene() {
             // Clear
             gl.clear(gl.COLOR_BUFFER_BIT);

             // Rotate
             var mvMatrix = mat4.fromRotation(mat4.create(), count, [0, 0.5, 0]);
             var uMvMatrix = gl.getUniformLocation(gl.program, "u_mvMatrix");
             gl.uniformMatrix4fv(uMvMatrix, false, mvMatrix);

             // Draw
             gl.drawElements(gl.TRIANGLES, LATITUDE_BANDS * LONGITUDE_BANDS * 3 * 2, gl.UNSIGNED_SHORT, 0);

             // Call drawScene again in the next browser repaint
             count += 0.01;
             requestAnimationFrame(drawScene);
         }
</script>
```

Se dibuja la escena usando la primitiva en función de LATITUDE_BANDS y LONGITUDE_BANDS. Previamente se hace una rotación en el eje Y