Sistemas Gráficos en la Web

Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Sistemas Gráficos

Grado en Ingeniería Informática

Curso 2016-2017

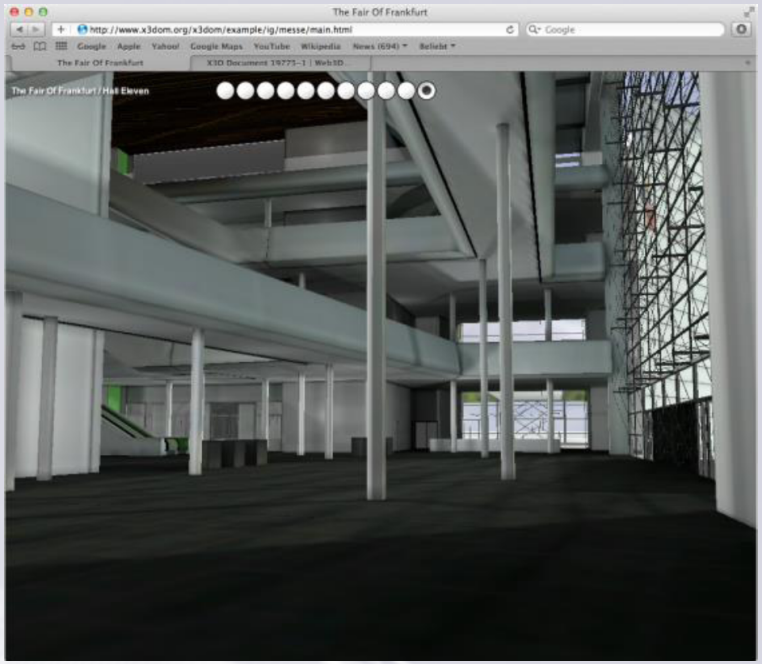
Contenidos

Objetivos

- Detectar los contextos en los que la visualización 3D puede ser interesante en un entorno web
- Comprender la problemática de la visualización 3D en los navegadores web
- Saber organizar un sistema gráfico 3D embebido en una página web

Introducción

- Desarrollar sistemas gráficos para la web es una tarea compleja
- Supone ejecutar algoritmos complejos en plataformas con recursos hardware muy dispares
- En este tema no se tratará sobre:
  - Renderización remota de gráficos
  - Descarga de modelos de un servidor
- Se tratará la visualización 3D en el marco de un navegador web



Escena 3D sobre Safari

©Fraunhofer IGD, Darmstadt

Visualización 3D en web

Problemática específica (1)

- Volumen de información y velocidad de transferencia
  - Arquitectura cliente / servidor
  - Separados cientos o miles de kilómetros
  - Con una velocidad muy variable. A veces, muy limitada
  - Un modelo 3D puede tener miles o millones de polígonos
- Altos requerimientos computacionales
  - La visualización 3D requiere bastantes recursos hardware
  - No todo usuario de un navegador dispone de ellos

Visualización 3D en web

Problemática específica (y 2)

- Entorno de ejecución restringido al navegador web
  - Memoria limitada
  - Sin acceso directo al sistema de ficheros,
  - Ni a las bibliotecas existentes en el sistema
- Diversidad de plataformas
  - Sistemas operativos
  - Navegadores

Tecnologías

- En los últimos años
  - VRML
    - Estándar no aceptado nativamente por todos los navegadores
    - Requiere de plugins, normalmente de pago, para su funcionamiento
  - Applets Java
    - Se ejecuta sobre la máquina virtual Java, se evitan las restricciones que impone el navegador
    - Pero está limitado a 64 MB y no soporta más de 250.000 polígonos
- Últimamente se han desarrollado diversos estándares Web
  - Motivado por la pérdida del monopolio de Internet Explorer y Safari
  - Facilitado por:
    - La aparición de HTML5, incluye nativamente un elemento canvas
    - La definición de WebGL, biblioteca JavaScript que accede a las capacidades OpenGL del cliente

WebGL

- https://www.khronos.org/webgl/
- Aparece con el apoyo de Google, Apple y Mozilla
  - Microsoft se sumó más tarde
- Es una API
  - Se accede mediante interfaces en JavaScript, no mediante etiquetas HTML
- Está basado en OpenGL ES 2.0 (OpenGL Embedded System)
- Utiliza GLSL
  - El uso de shaders es obligatorio
- Puede convivir con otro contenido HTML
- Es multiplataforma
- Es gratis

Estructura de un programa WebGL

1. Crear un elemento canvas
2. Obtener su contexto gráfico
3. Inicializar el viewport
4. Crear buffers que contengan la geometría
5. Crear matrices que transformen la geometría al espacio del dispositivo
6. Crear los shaders encargados de la visualización
7. Inicializar los shaders
8. Dibujar

Ejemplo de programa: Hola Mundo WebGL

Código HTML

Ejemplo extraído de [Parisi'2014]

Ejemplo de programa WebGL: Código HTML

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset='utf-8'/>
    <script type="text/javascript" src="gl-matrix.js"></script>
    <script type="text/javascript" src="script.js"></script>
  </head>
  <body style='margin:0px' onload='main()'>
    <canvas id='theCanvas' width=500 height=500
        style='position: absolute; background-color: black;'>
    </canvas>
  </body>
</html>
```

Hola Mundo WebGL

Función main

Hola Mundo WebGL: Función main

```
function main () {
  // Obtención del canvas y del contexto gráfico
  var canvas = document.getElementById ("theCanvas");
  var gl = initWebGL (canvas);

  // Inicialización del viewport
  initViewport (gl, canvas);

  // Creación de la geometría y las transformaciones
  var cube = createCube (gl);
  initMatrices (canvas);

  // Creación e inicialización de shaders y texturas
  initShader (gl);
  initTexture (gl);

  // Visualización
  run (gl, cube);
}
```

Hola Mundo WebGL

Contexto gráfico del canvas y Viewport

Hola Mundo WebGL: Contexto gráfico del canvas y Viewport

```
function initWebGL (canvas) {
  var gl = null;
  var msg = "WebGL not supported or not enabled";
  try {
    gl = canvas.getContext("experimental-webgl");
  } catch (e) {
    msg = "Error creating WebGL Context!: " + e.toString();
  }
  if (!gl) {
    alert(msg);
    throw new Error(msg);
  }
  return gl;
}

function initViewport (gl, canvas) {
  gl.viewport (0, 0, canvas.width, canvas.height);
}
```

Hola Mundo WebGL

Creación de la geometría

Hola Mundo WebGL: Creación de la geometría

```
function createCube (gl) {
  // Varios buffers para vértices, colores y coordenadas de textura
  var vertexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  var verts = [ -1.0, -1.0,  1.0, . . . , -1.0, 1.0, -1.0 ];
  gl.bufferData(gl.ARRAY_BUFFER,
      new Float32Array(verts), gl.STATIC_DRAW);
  // Similar con los colores y las coordenadas de textura
  // Otro buffer para los índices
  var cubeIndexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeIndexBuffer);
  var cubeIndices = [ 0, 1, 2, . . . , 20, 22, 23 ];
  gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
      new Uint16Array(cubeIndices), gl.STATIC_DRAW);
  // Se compone todo en un único objeto
  var cube = { vertex: vertexBuffer,  vertSize: 3,  nVerts: 24,
    color: colorBuffer,                colorSize: 3,  nColors: 24,
    texCoord: texCoordBuffer,      texCoordSize: 2,  nTexCoords: 24,
    indices: cubeIndexBuffer, nIndices: 36, primtype: gl.TRIANGLES };
  return cube;    }
```

Hola Mundo WebGL

Creación de las transformaciones

Hola Mundo WebGL: Creación de las transformaciones

```
var projectionMatrix, modelViewMatrix, rotationAxis;

function initMatrices (canvas)
{
  // Create a model view matrix with object at 0, 0, -8
  modelViewMatrix = mat4.create();
  mat4.translate(modelViewMatrix, modelViewMatrix, [0, 0, -8]);

  // Create a project matrix with 45 degree field of view
  projectionMatrix = mat4.create();
  mat4.perspective(projectionMatrix, Math.PI / 4,
      canvas.width / canvas.height, 1, 10000);

  rotationAxis = vec3.create();
  vec3.normalize(rotationAxis, [1, 1, 1]);
}
```

Hola Mundo WebGL

Shaders (1)

Hola Mundo WebGL: Código fuente del vertex shader

```
var vertexShaderSource =
"    attribute vec3 vertexPos; \n" +
"    attribute vec3 colorVert; \n" +
"    attribute vec2 texCoord; \n" +
"    uniform mat4 modelViewMatrix; \n" +
"    uniform mat4 projectionMatrix; \n" +
"    varying vec3 vColor; \n" +
"    varying vec2 vTexCoord; \n" +
"    void main(void) { \n" +
"        gl_Position = projectionMatrix * \n" +
"            modelViewMatrix * vec4(vertexPos, 1.0); \n" +
"        vColor = colorVert; \n" +
"        vTexCoord = texCoord; \n" +
"    } \n";
```

- Integrado en JavaScript como una cadena de texto

Hola Mundo WebGL

Shaders (2)

Hola Mundo WebGL: Código fuente del fragment shader

```
<script type="x-shader/x-fragment" id="fragment">
  varying vec3 vColor;
  varying vec2 vTexCoord;
  uniform sampler2D uSampler;
  void main(void) {
    gl_FragColor = texture2D(uSampler,
        vec2(vTexCoord.s, vTexCoord.t)) * vec4(vColor, 1.0);
  }
</script>
```

- Embebido en el código HTML de la página
- En el código JavaScript se lee así

Hola Mundo WebGL: Lectura del código fuente del vertex shader

```
var fragmentShaderSource =
    document.getElementById ("fragment").textContent;
```

Hola Mundo WebGL

Shaders (3)

Hola Mundo WebGL: Lectura y compilación de los shaders

```
function createShader (gl, str, type) {
  var shader;
  if (type == "fragment") {
    shader = gl.createShader(gl.FRAGMENT_SHADER);
  } else if (type == "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;
}
```

Hola Mundo WebGL

Shaders (4)

Hola Mundo WebGL: Creación del programa shader

```
var shaderProgram,    // Programa shader y varios locations
    shaderVertexPositionAttribute, shaderVertexColorAttribute,
    shaderProjectionMatrixUniform, shaderModelViewMatrixUniform,
    shaderSamplerUniform;

function initShader (gl) {
  // Cargar y compilar fragment y vertex shader
  var fragmentShader = createShader(gl, fragmentShaderSource,
      "fragment");
  var vertexShader = createShader(gl, vertexShaderSource, "vertex");

  // Enlazarlos en un programa shader
  shaderProgram = gl.createProgram();
  gl.attachShader(shaderProgram, vertexShader);
  gl.attachShader(shaderProgram, fragmentShader);
  gl.linkProgram(shaderProgram);

  . . .
```

Hola Mundo WebGL

Shaders (y 5)

Hola Mundo WebGL: Obtención de los locations de los parámetros

```
    . . .
    // Obtención de punteros a los parámetros, locations
    // Atributos de los vértices
    shaderVertexPositionAttribute = gl.getAttribLocation
        (shaderProgram, "vertexPos");
    gl.enableVertexAttribArray (shaderVertexPositionAttribute);
    // Similar para colores y coordenadas de textura ...

    // Parámetros uniform: matrices y textura
    shaderProjectionMatrixUniform = gl.getUniformLocation
        (shaderProgram, "projectionMatrix");
    shaderModelViewMatrixUniform = gl.getUniformLocation
        (shaderProgram, "modelViewMatrix");
    shaderSamplerUniform = gl.getUniformLocation
        (shaderProgram, "uSampler");
}
```

Hola Mundo WebGL

Texturas

Hola Mundo WebGL: Texturas

```
var okToRun = false;
function handleTextureLoaded (gl, texture) {
  gl.bindTexture (gl.TEXTURE_2D, texture);
  gl.pixelStorei (gl.UNPACK_FLIP_Y_WEBGL, true);
  gl.texImage2D (gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
      gl.UNSIGNED_BYTE, texture.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, null);
  okToRun = true;
}

var webGLTexture;
function initTexture (gl) {
  webGLTexture = gl.createTexture();
  webGLTexture.image = new Image();
  webGLTexture.image.src = "webgl-logo-256-bn.jpg";
  webGLTexture.image.onload = function () {
    handleTextureLoaded (gl, webGLTexture); }
}
```

Hola Mundo WebGL

Dibujado (1)

Hola Mundo WebGL: Dibujado (1)

```
function draw (gl, obj) {
  // Borrado del frame y del z-buffer
  gl.clearColor (0.0, 0.0, 0.0, 1.0);
  gl.enable (gl.DEPTH_TEST);
  gl.clear (gl.COLOR_BUFFER_BIT  | gl.DEPTH_BUFFER_BIT);

  // Shader a usar
  gl.useProgram (shaderProgram);

  // Conexión con los parámetros del shader
  gl.bindBuffer (gl.ARRAY_BUFFER, obj.vertex);
  gl.vertexAttribPointer (shaderVertexPositionAttribute,
      obj.vertSize, gl.FLOAT, false, 0, 0);
  // Similar con colores y coordenadas de textura ...

  // Índices
  gl.bindBuffer (gl.ELEMENT_ARRAY_BUFFER, obj.indices);
  ...
```

Hola Mundo WebGL

Dibujado (y 2)

Hola Mundo WebGL: Dibujado (y 2)

```
    ...
    // Transformaciones
    gl.uniformMatrix4fv(shaderProjectionMatrixUniform, false,
        projectionMatrix);
    gl.uniformMatrix4fv(shaderModelViewMatrixUniform, false,
        modelViewMatrix);

    // Texturas
    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, webGLTexture);
    gl.uniform1i(shaderSamplerUniform, 0);

    // Dibujado
    gl.drawElements(obj.primtype, obj.nIndices, gl.UNSIGNED_SHORT, 0);
}
```

Hola Mundo WebGL

Animación

Hola Mundo WebGL: Animación

```
var duration = 5000; // ms
var currentTime = Date.now();

function animate () {
  var now = Date.now ();
  var deltat = now - currentTime;
  var fract = deltat / duration;
  var angle = 2.0 * Math.PI * fract;
  mat4.rotate(modelViewMatrix, modelViewMatrix, angle, rotationAxis);
  currentTime = now;
}

function run (gl, cube) {
  requestAnimationFrame( function() { run (gl, cube); } );
  if (okToRun) {
    draw (gl, cube);
    animate ();
  }
}
```

Ventajas y desventajas de WebGL

- Proporciona flexibilidad al desarrollar sistemas gráficos
- Y eficiencia gracias al uso de shaders
- Sin embargo ...
  - Exige escribir muchas líneas de código
  - No se tiene el cálculo matricial para las transformaciones
  - No se tienen primitivas geométricas, materiales, luces, etc.
- Se tienen dos opciones:
  - Escribir tu propia biblioteca de alto nivel, definiendo clases para representar geometría, materiales, etc.
  - Usar una biblioteca de las ya existentes
- La biblioteca líder en este campo es **Three.js** (`threejs.org`)

Three.js

Características (1)

- `threejs.org`
- Creada y liberada en 2010 por el español Ricardo Cabello
  - Es Open Source y está bien mantenida
  - En la actualidad cuenta con unos 90 codificadores
- Oculta los detalles de bajo nivel de WebGL
  - Se representa una escena 3D usando *mallas, materiales, luces*
  - Incluye las clases matemáticas necesarias (matrices y vectores)
- Potente
  - No es solo un wrapper de WebGL
  - Contiene múltiples objetos predefinidos útiles en el desarrollo de juegos, animaciones, presentaciones, etc.
  - Se disponen igualmente de numerosos ejemplos para usar en nuestros proyectos

Three.js

Características (2)

- Rápida
  - Usando buenas prácticas de programación, con un alto rendimiento
- Robusta
  - Dispone de chequeo de errores, excepciones y warnings en consola que facilita el desarrollo
- Soporta interacción (picking)
- Soporta formatos de archivo 3D
  - Formatos ASCII de los programas de modelado 3D
  - Un formato específico de Three.js, JSON
- Orientada a objetos, extensible
- Fácil de usar, y de aprender

Ejemplos realizados con Three.js

- Video clip

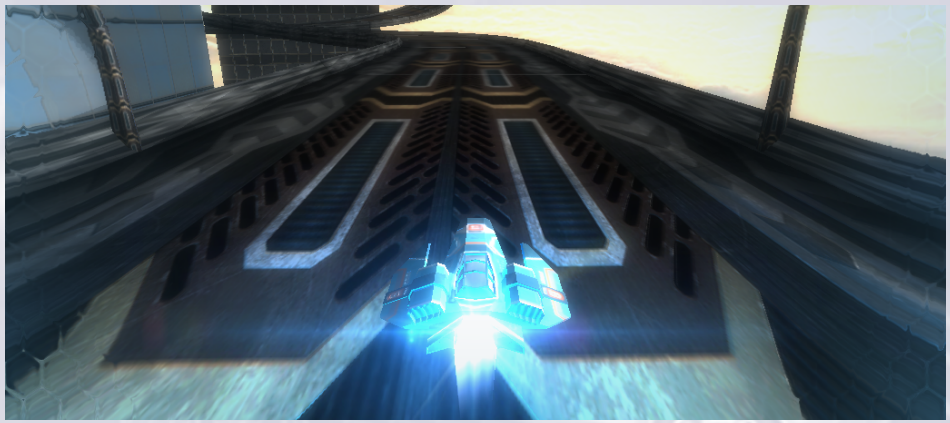
http://www.ro.me

- Configurador de coches

http://carvisualizer.plus360degrees.com/threejs

- Juegos

http://cwar.de/pinball/simpleball.html



http://hexgl.bkcore.com

Three.js

Recursos

- **Página oficial:** `http://threejs.org/`
  Incluye la biblioteca para descargársela, la documentación y bastantes ejemplos
- **Bibliografía**
  - J. Dirksen;
    Learning Three.js: The JavaScript Library for WebGL;
    recurso electrónico en `biblioteca.ugr.es`
    Ejemplos en:
    `https://github.com/josdirksen/learning-threejs`
  - J. Dirksen;
    Three.js Essential;
    recurso electrónico en `biblioteca.ugr.es`
    Ejemplos en:
    `https://github.com/josdirksen/essential-threejs`

Hola Mundo Three.js

Código HTML

Ejemplo extraído de [Parisi'2014]

Ejemplo de programa Three.js: Código HTML

```
<html>
  <head>
    <meta http-equiv="Content-Type"
        content="text/html; charset=UTF-8">
    <title>Three.js Cube with Lighting</title>
    <script src="jquery-1.9.1.js"></script>
    <script src="three.js"></script>
    <script src="RequestAnimationFrame.js"></script>
    <script type="text/javascript" src="script.js"></script>
  </head>
  <body>
    <canvas id="webglcanvas" style="border:none"
        width="500" height="500">
    </canvas>
  </body>
</html>
```

Hola Mundo Three.js

Función main

Ejemplo de programa Three.js: Función main

```
$(document).ready( function () {
    // Se obtiene el canvas y se construye el renderer
    var canvas = document.getElementById("webglcanvas");
    renderer = createRenderer (canvas);
    // Se crea una escena, a la que se le añade lo demás
    scene = new THREE.Scene ();
    // Se crea una cámara y se añade a la escena
    camera = createCamera (canvas, scene);
    scene.add ( camera );
    // Se crea una luz y se añade a la escena
    var light = createLight ();
    scene.add ( light );
    // Se crea la geometría y se añade a la escena
    geometry = createCube ();
    scene.add ( geometry );
    // Se visualiza
    run();
  } );
```

Hola Mundo Three.js

Renderer

Ejemplo de programa Three.js: Renderer

```
function createRenderer (canvas) {
  // Se crea un renderer basado en WebGL asociado al canvas
  var renderer = new THREE.WebGLRenderer( { canvas: canvas,
      antialias: true } );

  // Se establece el tamaño del viewport
  renderer.setSize (canvas.width, canvas.height);

  return renderer;
}
```

Hola Mundo Three.js

Cámara

Ejemplo de programa Three.js: Cámara

```
function createCamera (canvas, scene) {

  // Una cámara en perspectiva
  var camera = new THREE.PerspectiveCamera( 45,
           canvas.width / canvas.height, 1, 4000 );

  // En una determinada posición
  camera.position.x = 2;
  camera.position.y = 4;
  camera.position.z = 6;

  // Mirando al centro de la escena
  camera.lookAt (scene.position);

  return camera;
}
```

Hola Mundo Three.js

Luz

Ejemplo de programa Three.js: Luz

```
function createLight () {

  // Luz direccional amarillenta  de intensidad 1.5
  var light = new THREE.DirectionalLight( 0xffff77 , 1.5);

  // Posición de la luz direccional, mirando al origen
  light.position.set(0, 1, 1);

  return light;
}
```

Hola Mundo Three.js

Geometría

Ejemplo de programa Three.js: Geometría

```
function createCube () {
  // Se crea un material con textura difusa
  // Se carga la textura
  var map = THREE.ImageUtils.loadTexture("webgl-logo-256.jpg");

  // Se crea un material basado en Phong con textura difusa
  var material = new THREE.MeshPhongMaterial({ map: map,
    color: 0xffffff });

  // Se crea la geometría de un cubo
  var geometry = new THREE.CubeGeometry(2, 2, 2);

  // Se crea un Mesh con la geometría y el material
  var cube = new THREE.Mesh(geometry, material);

  return cube;
}
```

Hola Mundo Three.js

Visualizado y Animación

Ejemplo de programa Three.js: Visualizado y Animación

```
function animate() {
  var now = Date.now();
  var deltat = now - currentTime;
  var fract = deltat / duration;
  var angle = Math.PI * 2 * fract;
  geometry.rotation.y += angle;
  currentTime = now;
}

function run() {
  requestAnimationFrame(function() { run(); });

  // Remder de la escena
  renderer.render( scene, camera );

  // Modificaciones para la animación
  animate();
}
```

Sistemas Gráficos en la Web

Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Sistemas Gráficos

Grado en Ingeniería Informática

Curso 2016-2017

Parte de este material ha sido realizado en colaboración con Francisco Javier Melero Rus