Three.js-ESP Documentation

Publicación latest

31 de August de 2015

Índice general

Tutorial en español que explica como utilizar la libreria Three.js, para crear y mostrar gráficos animados por ordenador en 3D en un navegador Web

Introducción:

CAPÍTULO 1

Presentación

- Que es una librería en Programación
- • Es un fragmento de codigo que ya está hecho y puedes utilizar en tu proyecto, añadiendo funciones o mejorando las existentes.
- Que nos ofrece Three.js
- • Nos permite crear y mostrar gráficos animados por ordenador en 3D en un navegador Web.
- • Entre sus características, están las siguientes:
- • ○ Efectos: anaglifo, bizco y la barrera de paralaje.
- • ○ Escenas: añadir y eliminar objetos en tiempo de ejecución
- • ○ Cámaras: perspectiva y ortográfica
- • ○ Animación: armaduras, cinemática directa, cinemática inversa, morphing y fotogramas clave.
- • ○ Luces: ambiente, dirección, luz de puntos y espacios, sombras: emite y recibe.
- • ○ Shaders:reflejos en la lente, pase profundo y una extensa biblioteca de post-procesamiento
- • ○ Objetos: mallas, partículas, sprites, líneas, cintas, huesos y otros.
- • ○ Geometría: plano, cubo, esfera, toroide, texto en 3D y otras
- • ○ Exportación e importación: Los gráficos 3D se pueden crear desde Blender.
- ¿Para que sirve?
- • Es capaz de generar escenas 3D con WebGL*, Canvas (2D) y SVG, y utilizando la potencia de JavaScript, podemos interactuar con la animación desde el telcado, el ratón o incluso desde una WebCam.
- WebGL1 es una especificación estándar que está siendo desarrollada actualmente para mostrar gráficos en 3D en navegadores web. El WebGL permite mostrar gráficos en 3D acelerados por hardware (GPU) en páginas web, sin la necesidad de plug-ins.

CAPÍTULO 2

Preparando el terreno

Para poder utilizar Javascript o cualquiera de sus librerías, no necesitamos instalar nada, pero si habilitar en el navegador la utilización de WebGL.

2.1 Google Chrome

- **Paso 1:** Abre el navegador y en la barra de direcciones escribe: chrome://flags/
- **Paso 2:** Desde esta página podemos habilitar soporte adicional al navegador web, en nuestro caso para WebGL, necesitas habilitar los siguientes experimentos:
- • WebGL (Importante)
- • Ignorar la lista de renderización por software (Opcional si no funciona con el primero)
- • Canvas 2D acelerado por GPU (opcional)
- **Paso 3:** Reinicia el navegador.

2.2 Firefox

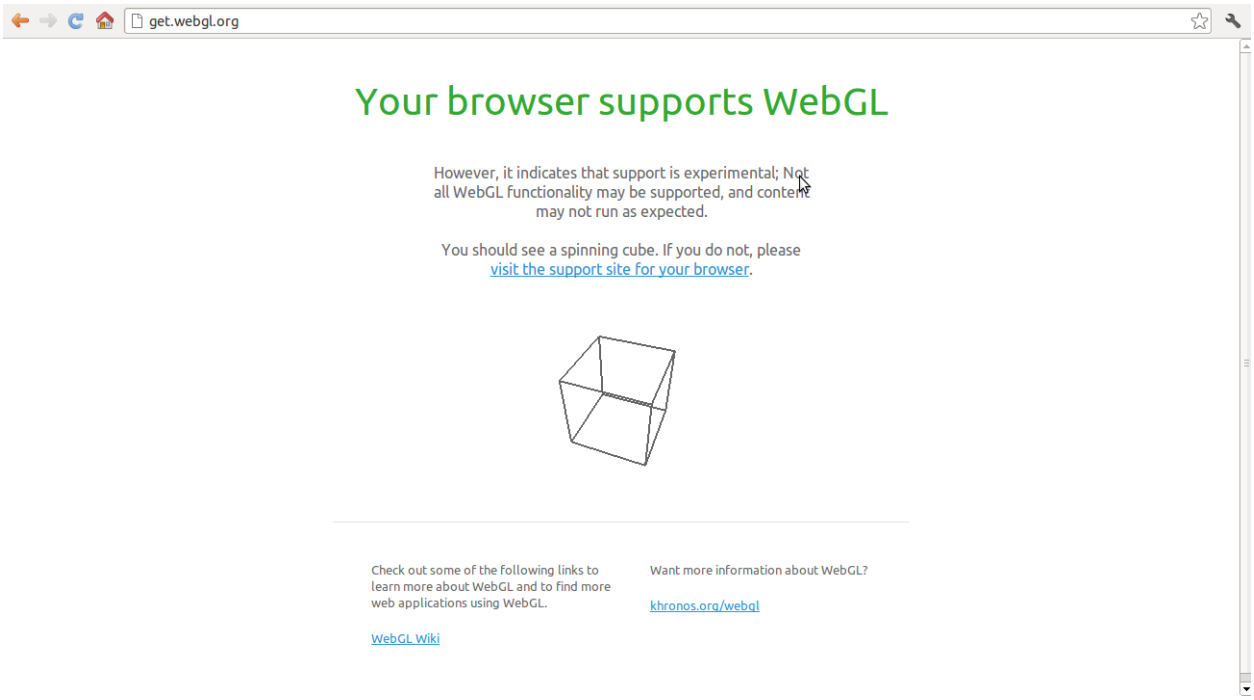
- **Paso 1:** Abre el navegador y en la barra de direcciones escribe: about:config
- **Paso 2:** Desde esta página podemos habilitar soporte adicional al navegador web, en nuestro caso para WebGL, necesitas habilitar los siguientes experimentos:
- • webgl.force-enabled. (importante)
- • layers.acceleration.force-enabled **–> solo en windows**
- • gfx.direct2d.force-enabled **–> solo en windows**
- **Paso 3:** Reinicia el navegador.

Una vez habilitado, abre esta página. get.webgl.org , te debería salir algo así:

2.3 Three.js

Para poder utilizar la librería de Three.js, hace falta descargarla y añadirla en la carpeta del proyecto.

Descargar la última versión .



Una vez descargado, descomprimir en la carpeta en la que tengamos el proyecto.

Solo utilizaremos el archivo **three.min.js**, que está en la carpeta build, hay que ponerlo en una carpeta que llamaremos "js", en la que guardaremos todos los archivos ".js".

CAPÍTULO 3

Conceptos básicos

Empecemos a diseñar nuestro proyecto

3.1 Que vamos a hacer

Para este tutorial haremos un proyecto en el cual aparecerá un cubo que podremos mover por la escena.

3.2 Partes del proyecto

Vamos a desgranar el proyecto en diferentes objetos, explicando cada uno:

- **Canvas** –> Canvas (lienzo en inglés) es un elemento HTML que permite la generación de gráficos dinámicamente. El objeto canvas puede ser accedido a través de JavaScript, permitiendo generar gráficos 2D, juegos, animaciones y composición de imágenes.
- **Escena** –> Una escena es un espacio tridimensional en el que puedes añadir objetos, interactuar con ellos y moverte por el.
- **Render** –> Un render es el objeto WebGL donde la tarjeta gráfica pintará todos los gráficos. Un mundo 3D (escena) puede ser enorme, pero en pantalla sólo se pintará aquello que quede dentro del encuadre de la "cámara". Ésto es lo que llamaremos render.
- **Camara** –> Una escena puede contener tantas cámaras como deseemos, pero sólo se podrá utilizar una de ellas para renderizar la escena en un momento dado. Una cámara puede ser posicionada y rotada tantas veces como queramos.Para definir una cámara hay que tener en cuenta qué tipo de proyección queremos tener. Three.js nos proporciona tres tipos de proyecciones:
  - Proyección paralela, cámara del tipo OrthographicCamera
  - Proyección cónica, cámara del tipo PerspectiveCamera
  - Proyección combinada, cámara del tipo CombinedCamera, y que permite cambiar entre la proyección cónica y paralela en tiempo de ejecución.

  La OrthographicCamera es una perspectiva que respeta el tamaño de los objetos, independientemente de la distancia a la que se encuentren de la cámara.

  La PerspectiveCamera es la que deforma los objetos según la distancia y posición que se encuentren con respecto a la cámara, tal y como ocurre en el mundo real.
- **Figura** –> Una figura es la cosa que quieres pintar en la pantalla. Un punto, Una línea, un triángulo, Un cuadrado. Un cubo. Un coche. Three.js llama a este tipo de objeto como Mesh.

Para crear una figura, Three.js necesita dos cosas: Una geometría y un material.

- La **geometría** es la lista de vértices y caras que forman una figura determinada. Three.js permite crear geometrías de muy diversas formas, como puedes encontrar en su documentación. Hay clases para definir figuras vértice a vértice, a mano, pero también hay clases que nos dibujan cubos, esferas o conos.
- Los **materiales** son "la piel" de las figuras, sirven para definir de qué color es cada cara de una figura y cómo la luz actúa en dicha cara. Al igual que con las geometrías, Three.js nos proporciona un montón de clases para crear distintos tipos de materiales según el efecto que queramos tener.

Introducción:

CAPÍTULO 4

Creando una escena

Separaremos la parte estética, que contiene el canvas con la parte lógica que contendrá la manipulación del cubo.

4.1 Parte estética

Estará en el archivo index.html, y será el archivo "ejecutable".

```
<!DOCTYPE html>
<html>
<head>
<title>Cubo en three.js</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<script src="js/three.min.js"></script>
<script src="js/javascript.js"></script> <!-- archivo que contendrá la parte lógica -->
</head>
<body onload="webGLStart();">
<div id="canvas" align="center"></div>
</body>
</html>
```

4.2 Parte lógica

Estará en el archivo javascript.js, está dentro de la carpeta js.

```
var escena;
var camara;
var render;

function startEscena(){
  //Render
  render = new THREE.WebGLRenderer(); // definimos el renderizador

  render.setClearColor(0x000000, 1); // el colorde limpieza, negro

  var canvasWidth = 500; // tamaño del canvas
  var canvasHeight = 500; // tamaño del canvas
  render.setSize(canvasWidth, canvasHeight);

  document.getElementById("canvas").appendChild(render.domElement); // indicamos que el render pinte
```

```
  //Escena
  escena = new THREE.Scene(); // definimos la escena

  //Camara
  camara = new THREE.PerspectiveCamera(45, canvasWidth / canvasHeight, 0.2, 150); // definimos la cam
  camara.position.set(0, 0, 0); // situamos la cámara en las coordenadas absolutas
  camara.lookAt(escena.position); // le indicamos a la cámara que mire al centro de la escena.
  escena.add(camara); // añadimos la cámara a la escena


}
function renderEscena(){
  render.render(escena, camara); // para dibujar la escena
}

function webGLStart() { //función llamada cuando se carga la página (onload)
   startEscena();
   renderEscena();
}
```

Respecto a la cámara, para crear una PerspectiveCamera necesitaremos poner cuatro opciones:

- Ángulo del campo de visión en grados
- Ratio de aspecto, que normalmente es la relación entre el WIDTH y el HEIGHT del canvas donde se va a renderizar la imagen
- Distancia mínima de dibujado
- Distancia máxima de dibujado

4.3 Resultado

Si abrimos el archivo index.html en el navegador, veremos algo como esto:

El canvas negro es la escena, y se ve negro ya que no hemos añadido ningún objeto.

CAPÍTULO 5

Añadiendo objetos

Ahora ya tenemos la escena, pero vacía. En three.js hay dos grandes tipos de figuras, las geométricas o vectoriales, que las puedes generar desde el propio three.js, y las que generas con un editor de gráficos 3D, como blender.

5.1 Figuras geométricas

Las figuras geometrícas que genera three.js son de las más fáciles de usar.

Vamos a dibujar un cubo:

Material

Crear un material, necesitamos utilizar la clase MeshBasicMaterial, que necesita que le indiquemos el color hexadecimal con el que pintar la figura. Vamos a hacer 6 materiales, uno por cara.

```
var cuboMateriales = [
   new THREE.MeshBasicMaterial({color:0x33FF00}),
   new THREE.MeshBasicMaterial({color:0x00CCFF}),
   new THREE.MeshBasicMaterial({color:0xFF0000}),
   new THREE.MeshBasicMaterial({color:0xFFCC00}),
   new THREE.MeshBasicMaterial({color:0x99FFFF}),
   new THREE.MeshBasicMaterial({color:0xFFFFFF})
];
var cuboMaterial = new THREE.MeshFaceMaterial(cuboMateriales);
```

Creamos un THREE.MeshFaceMaterial,que contiene los 6 materiales.

Figura

```
var cuboGeometria = new THREE.BoxGeometry(2, 2, 2);

cubo = new THREE.Mesh(cuboGeometria, cuboMaterial);
cubo.position.set(0, 0, -7.0);
escena.add(cubo);
```

Creamos la geometría del cubo, usando CubeGeometry. Hay que pasar tres valores: Dimensiones de su ancho (x) Dimensiones de su alto (y) Dimensiones de su profundidad (z)

A la clase new THREE.Mesh, hay que añadir la geometría, y el material. Según el orden de los colores puesto en el material, cambiará la disposición de estos. La distribución de colores sigue este patrón: - cara izquierda - cara derecha - cara superior - cara inferior - cara frontal - cara posterior

con *position.set* cambiamos las coordenadas del cubo, y con *escena.add* lo añadimos a ella.

El cubo lo tenemos que añadir dentro de la función startEscena, quedando así:

```
function startEscena(){
//Render
render = new THREE.WebGLRenderer(); // definimos el renderizador

render.setClearColor(0x000000, 1); // el colorde limpieza, negro

var canvasWidth = 500; // tamaño del canvas
var canvasHeight = 500; // tamaño del canvas
render.setSize(canvasWidth, canvasHeight);

document.getElementById("canvas").appendChild(render.domElement); // indicamos que el render pinte la

//Escena
escena = new THREE.Scene(); // definimos la escena

//Camara
camara = new THREE.PerspectiveCamera(45, canvasWidth / canvasHeight, 0.2, 150); // definimos la camar
camara.position.set(0, 0, 0); // situamos la cámara en las coordenadas absolutas
camara.lookAt(escena.position); // le indicamos a la cámara que mire al centro de la escena.
escena.add(camara); // añadimos la cámara a la escena

//Cubo

var cuboMateriales = [
 new THREE.MeshBasicMaterial({color:0x33FF00}),
 new THREE.MeshBasicMaterial({color:0x00CCFF}),
 new THREE.MeshBasicMaterial({color:0xFF0000}),
 new THREE.MeshBasicMaterial({color:0xFFCC00}),
 new THREE.MeshBasicMaterial({color:0x99FFFF}),
 new THREE.MeshBasicMaterial({color:0xFFFFFF})
];
var cuboMaterial = new THREE.MeshFaceMaterial(cuboMateriales);

var cuboGeometria = new THREE.BoxGeometry(2, 2, 2);

cubo = new THREE.Mesh(cuboGeometria, cuboMaterial);
cubo.position.set(0, 0, -7.0);
escena.add(cubo);
```

5.2 Resultado

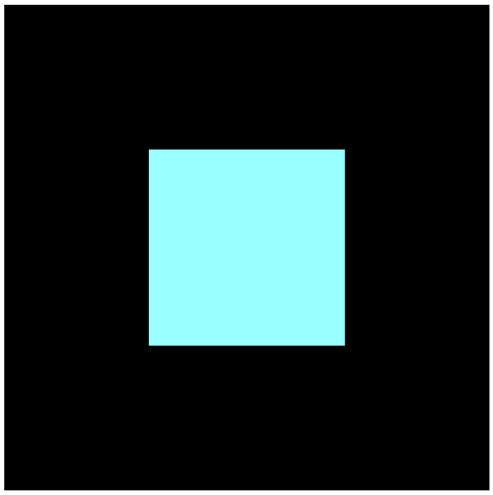
Si abrimos el archivo index.html en el navegador, veremos algo como esto:

5.3 Figuras vectoriales

Consiste en indicar todos los vectores y caras que contendrá la figura. Esta opción es util para hacer objetos no regulares.

Nosotros haremos un cuadrado. tenemos que indicar la cordenada de inicio y de final, de cada cara:

```
var cuadradoMaterial = new THREE.MeshBasicMaterial({
    color:0x8080FF,
    side:THREE.DoubleSide
```



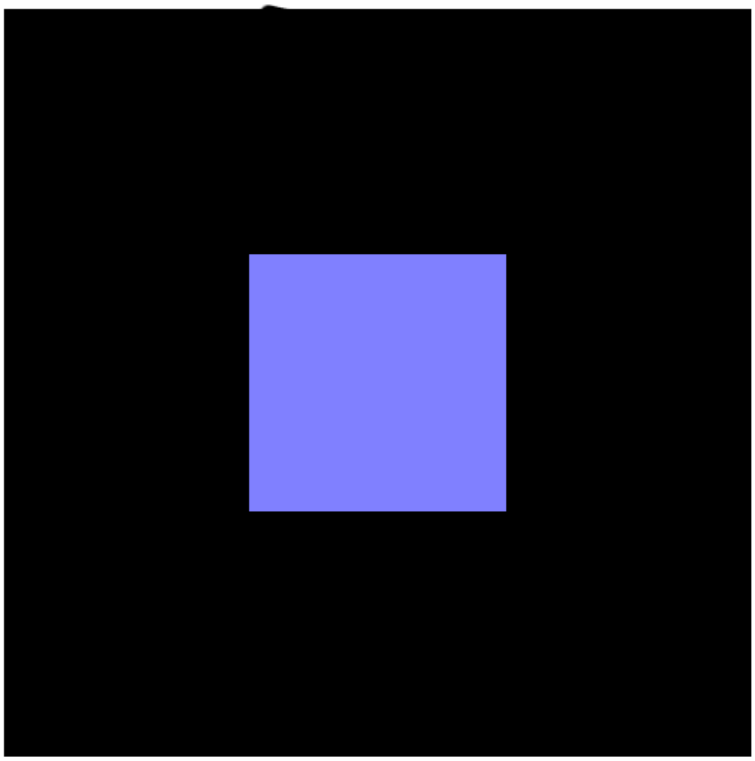
```
});

var cuadradoGeometria = new THREE.Geometry();
cuadradoGeometria.vertices.push(new THREE.Vector3(-1.0,  1.0, 0.0));
cuadradoGeometria.vertices.push(new THREE.Vector3( 1.0,  1.0, 0.0));
cuadradoGeometria.vertices.push(new THREE.Vector3( 1.0, -1.0, 0.0));
cuadradoGeometria.vertices.push(new THREE.Vector3(-1.0, -1.0, 0.0));
cuadradoGeometria.faces.push(new THREE.Face4(0, 1, 2, 3));

cubo = new THREE.Mesh(cuadradoGeometria, cuadradoMaterial);
cubo.position.set(0, 0, -7.0);
escena.add(cubo);
```

5.4 Resultado

Si abrimos el archivo index.html en el navegador, veremos algo como esto:



CAPÍTULO 6

Movimiento

Ya tenemos la escena, y un objeto. Ahora toca hacer que se mueva el cubo.

6.1 Creamos la función animarEscena()

Primero añadiremos una función, que se ejecutará cada pocas milésimas de segundo, en la que indicaremos que nuestro cubo rote.

Tenemos que añadir la variable cubo al principio del archivo javascript.js, para poder utilizarlo en cualquier parte de nuestro código. Antes de la función startEscena(), pondremos esta función:

```
window.requestAnimFrame = (
     function(){
             return  window.requestAnimationFrame       ||
             window.webkitRequestAnimationFrame ||
             window.mozRequestAnimationFrame    ||
             function( callback ){
                     window.setTimeout(callback, 1000 / 60);
             };
}
)();
```

Con esta función, conseguiremos que se ejecute la función animarEscena() ciclicamente.

```
function animarEscena(){
        requestAnimFrame(animarEscena); // para ejecutar ciclicamente

        cubo.rotation.y+= 0.002; // rotamos el cubo en el eje y 0.002 unidades de medida.

    renderEscena();
}
```

Solo nos falta añadir a la función webGLStart() una llamada a nuestra nueva función.

Tiene que quedar así:

```
var escena;
var camara;
var render;
var cubo;

window.requestAnimFrame = (
     function(){
```

```
            return  window.requestAnimationFrame       ||
            window.webkitRequestAnimationFrame ||
            window.mozRequestAnimationFrame    ||
            function( callback ){
                    window.setTimeout(callback, 1000 / 60);
            };
}
)();

function animarEscena(){
        requestAnimFrame(animarEscena); // para ejecutar ciclicamente

        cubo.rotation.y+= 0.02; // rotamos el cubo en el eje y 0.02 unidades de medida.

    renderEscena();
}


function startEscena(){
  //Render
  render = new THREE.WebGLRenderer(); // definimos el renderizador

  render.setClearColor(0x000000, 1); // el colorde limpieza, negro

  var canvasWidth = 500; // tamaño del canvas
  var canvasHeight = 500; // tamaño del canvas
  render.setSize(canvasWidth, canvasHeight);

  document.getElementById("canvas").appendChild(render.domElement); // indicamos que el render pinte

  //Escena
  escena = new THREE.Scene(); // definimos la escena

  //Camara
  camara = new THREE.PerspectiveCamera(45, canvasWidth / canvasHeight, 0.2, 150); // definimos la ca
  camara.position.set(0, 0, 0); // situamos la cámara en las coordenadas absolutas
  camara.lookAt(escena.position); // le indicamos a la cámara que mire al centro de la escena.
  escena.add(camara); // añadimos la cámara a la escena

  //cubo
     var cuboMateriales = [
   new THREE.MeshBasicMaterial({color:0x33FF00}),
   new THREE.MeshBasicMaterial({color:0x00CCFF}),
   new THREE.MeshBasicMaterial({color:0xFF0000}),
   new THREE.MeshBasicMaterial({color:0xFFCC00}),
   new THREE.MeshBasicMaterial({color:0x99FFFF}),
   new THREE.MeshBasicMaterial({color:0xFFFFFF})
];
var cuboMaterial = new THREE.MeshFaceMaterial(cuboMateriales);


var cuboGeometria = new THREE.BoxGeometry(2, 2, 2);

cubo = new THREE.Mesh(cuboGeometria, cuboMaterial);
cubo.position.set(0, 0, -7.0);
escena.add(cubo);
```

```
}
function renderEscena(){
  render.render(escena, camara); // para dibujar la escena
}

function webGLStart() { //función llamada cuando se carga la página (onload)
   startEscena();
   animarEscena();
   renderEscena();
}
```

6.2 Resultado

Si abrimos el archivo index.html en el navegador, veremos como el cubo rota: