# Three.js

A grandes rasgos three.js es una librería que nos permite crear WebGL (3D en el navegador), básicamente la diferencia cabe en que si queremos crear un cubo con three.js basta con algunas líneas en cambio con JavaScript la cantidad de líneas sería mayor.

Al igual que la mayoría de librerías (frameworks) podemos descargar la librería completa desde la página web oficial o acceder a una versión más ligera a través de su cdn (content delivery network).



Figura x.x - cdn three.js

Para mostrar gráficos en 3D en el navegador (WebGL) con three.js debemos crear una serie de elementos y funciones.

## Creando la escena

Para lograr un código más legible y manejable he realizado durante todo el proyecto una programación modular, así cada módulo resuelve un problema particular.

Para visualizar gráficos en 3D con three.js como elementos básicos necesitamos una escena, una cámara y un renderizado de la escena con la cámara. La página *html* contiene las etiquetas básicas de cualquier página, si nos hemos descargado la librería debemos indicar cada fichero javaScript que usemos durante la programación.

Para un código más ordenado indicaremos las librerías javaScript entre la etiqueta *<head>* y en la etiqueta *<body>* llamaremos al método *onload* indicando la función con la cual cargaremos todos los elementos de la página.

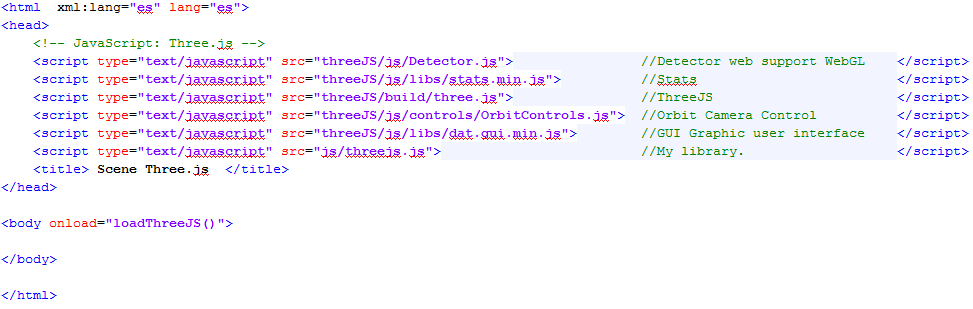
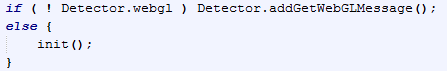


Figura x.x - PFCMyScene.html

Como se puede apreciar la página inicial carece de elementos ya que todos los elementos serán creados mediante javaScript modificando el DOM (Document object model), el fichero *threejs.js* es aquel que creara la escena.

Antes de crear la escena es muy útil llamar a la librería *Detector* ubicada en /*js/Detector.js* para indicarnos si nuestro navegador soporta WebGL, si todo va bien llamaremos a la función *init* que se encargará de cargar todos los elementos antes descritos más otros para lograr una escena más completa.



Si nuestro navegador permite el uso de WebGL llamamos a la función *init* que como su nombre lo indica se encarga de inicializar todos los elementos necesarios para crear una escena. Crearemos un objeto contenedor donde vamos a *renderizar* la escena, este será una etiqueta *div* (bloque de contenido o sección de página) y la añadimos a nuestra página.



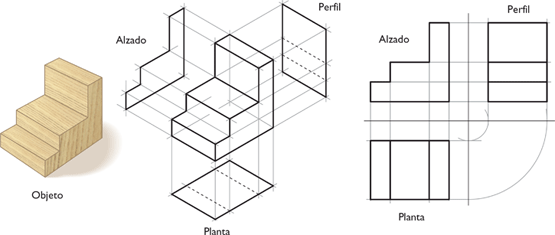
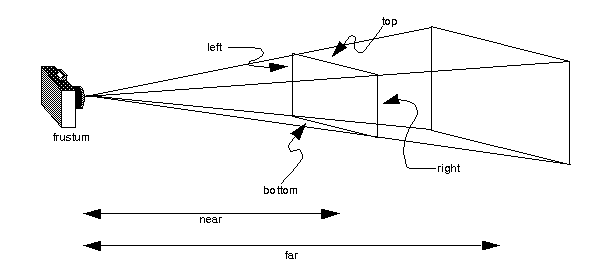
### Escena

La creación de la escena es bastante simple, solamente debemos crear el objeto y para añadir objetos usaremos la función *add*.



### Cámara

Three.js nos ofrece cuatro tipos de cámaras:

* Camera: esta es una clase abstracta de la cual heredan las otras.
* CubeCamera: Crea 6 cámaras para renderizar un tipos especiales de gráficos 3D, es una cámara que no use durante mi proyecto.
* OrthographicCamera: Cámara con una proyección ortográfica. Una proyección ortográfica es un sistema de representación gráfica que consiste en representar elementos geométricos o volúmenes en un plano mediante proyección ortogonal.
* PerspectiveCamera: Cámara con una proyección perspectiva. Este método consiste en proyectar puntos hacia el plano de visión, con este tipo de proyección parece todo más real ya que es la manera de formar las imágenes en el lente de las cámaras, como lo ve el ojo humano.  
  

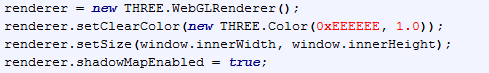
 Las cámaras tienes atributos similares como *near* y *far*,tal como lo indica su traducción es la posición de la cámara hacia el plano cercano y lejano respectivamente y nos indica que en ese intervalo serán mostrados los objetos en la escena. Para la cámara en perspectiva se necesitan dos valores más para los atributos *fov* y *aspect*, el primero es el campo de visión desde el fondo hasta el inicio en grados y el segundo es el radio de aspecto que siempre tendrá el valor del ancho de la ventana entre la altura de la misma.

Con la cámara creada debemos establecer su posición y hacia donde apunta, en este caso apuntará hacia la escena.



### Renderizado

En este punto ya hemos creado la cámara y la escena, ahora debemos crear el objeto de renderizado y añadirlo al DOM, es decir, añadirlo a nuestra página.



Como se puede ver en el fragmento de código solo hemos creado el objeto de renderizado añadiendo algunos valores a los atributos color de fondo, tamaño y muestra de sombra habilitada.

Para que nuestra escena sea dinámica he añadido un control orbital para la cámara, la idea es que podamos manejar a nuestro antojo la cámara por la escena. Necesitaremos añadir la librería ubicada en /*js/controls/OrbitControls.js* y crear el objeto *OrbitControls* asociando la cámara y el renderizado.



Una vez creado todos los objetos debemos añadir al DOM él renderizado y luego asociar la escena y la cámara, para esta acción crearemos una función llamada *render* la cual además de asociar el renderizado con la escena y la cámara nos proporcionará una escena más dinámica.

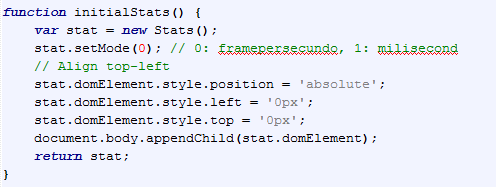


### Función *render*

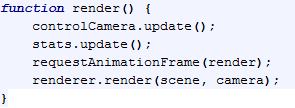
Como objetivo principal esta función asocia al objeto de renderizado la escena y cámara. Si asociamos los elementos descritos anteriormente y añadimos algún objeto como un cubo no veríamos nada ya que además de asociar los objetos debemos actualizar la escena, para ello usaremos la función *requestAnimationFrame* pasando como argumento la función *render*. La función requestAnimationFrame dibujara lo renderizado en la escena 60 veces por segundo (60 fotogramas por segundo), como se actualizará el renderizado también debemos actualizar el valor del objeto *OrbitControls* para lograr el efecto de control de la cámara.

Para ver la cantidad de fotogramas que nuestro navegador trabaja vamos a añadir un objeto *Stats* cuya librería está ubicada en */js/libs/stats.min.js*, este también debe ser actualizado en la función *render*. Crearemos un objeto y lo asociaremos a una etiqueta *div* creada dinámicamente como se hizo anteriormente, para ello usaremos la función *initialStats* que nos creará el bloque en el DOM y nos devolverá el objeto con las estadísticas.



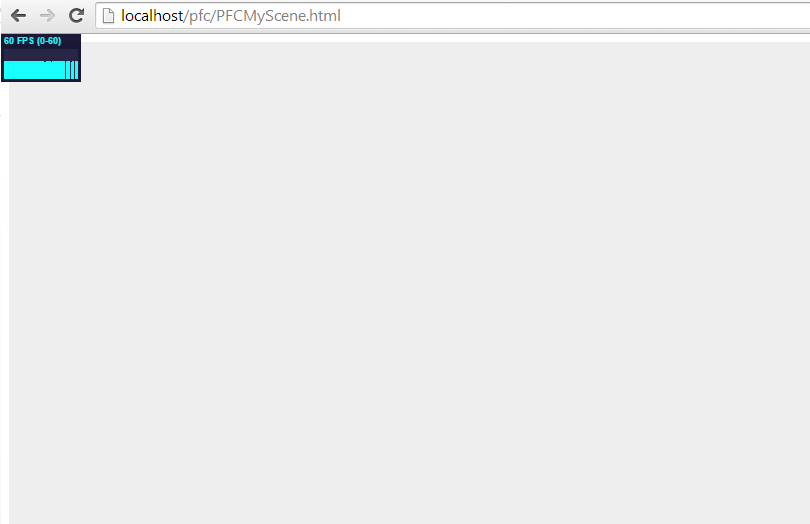


Ya creados los objetos básicos para la escena creamos la función *render*:



Cabe mencionar que todas las variables descritas son variables globales de javasScript, gracias a ello podemos modificar sus valores fácilmente en las funciones.

Con todo lo descrito anteriormente deberíamos crear una escena sin ningún elemento teniendo un aspecto que se puede apreciar en la siguiente imagen.



Crear unas escena: <http://threejs.org/docs/index.html#Manual/Introduction/Creating_a_scene>

WebBL: <http://es.wikipedia.org/wiki/WebGL>

Cdn: <http://es.wikipedia.org/wiki/Red_de_entrega_de_contenidos>

Página web: <http://threejs.org/>

<http://es.wikipedia.org/wiki/Proyecci%C3%B3n_ortogr%C3%A1fica>

<http://es.wikipedia.org/wiki/Proyecci%C3%B3n_ortogonal>

<http://sabia.tic.udc.es/gc/teoria/Proyecciones/Punto5.htm>

Ejemplos

<http://www.smartjava.org/content/all-109-examples-my-book-threejs-threejs-version-r63>

<http://stemkoski.github.io/Three.js/>

<http://learningthreejs.com/blog/2011/08/14/dat-gui-simple-ui-for-demos/>

<http://mrdoob.github.io/three.js/examples/canvas_interactive_cubes.html>

<http://srchea.com/apps/terrain-generation-diamond-square-threejs-webgl/>

Idea principal

<http://www.instructables.com/id/3D-Print-Your-Trek-in-color/?ALLSTEPS>