Funcionamiento de distintas Librerías gráficas en JS

Desarrollo Web En Entorno Cliente

22/02/2019

Luis Ángel Santos y Jesús Martínez

Índice del proyecto

1. FICHA DE PLANIFICACIÓN DEL PROYECTO

OBJETIVO DEL PROYECTO	Estudiar las librerías gráficas P5, Parallax y Three y mostrar mediante ejemplos prácticos su funcionamiento.
MIEMBROS	Luis Ángel Santos y Jesús Mártínez
PLANIFICACIÓN	La planificación por día se va a realizar a través de la herramienta de gestión de proyectos que nos ofrece la plataforma **GitHub,** la cual además usaremos como repositorio del proyecto. Los avances realizados serán documentados en un fichero .doc aparte.

2. Descripción y objetivo del proyecto

El objetivo del proyecto es mostrar el funcionamiento de las siguientes librerías gráficas en JavaScript mediante ejemplos prácticos:

- P5 → Se centra en el dibujo de gráficos bidimensionales aunque permite el uso de gráficos tridimensionales utilizando las herramientas de WebGL y Canvas
- Parallax → Permite la reacción del contenido con respecto a la orientación de un dispositivo móvil o con respecto a la posición del cursor en dispositivos de escritorio.
- Three →Orientada a la creación de contenido 3D en la web además de otro tipo de contenidos como de VR

3. Tareas realizadas y funcionamiento de librerías

Librería P5

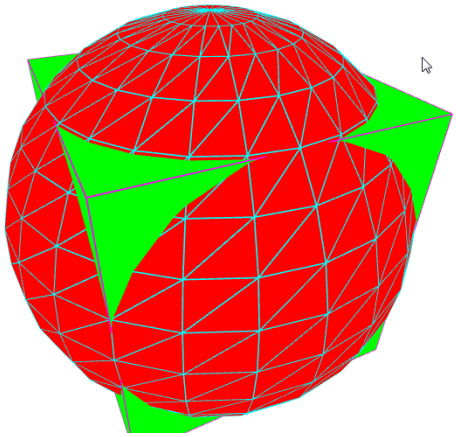
Introducción

P5.JS Es una librería que adapta la funcionalidad que ofrecía Processing (Un software y lenguaje de diseño por código) a la WEB, permitiendo dibujar sobre un canvas de una forma sencilla e intuitiva. Nos permite utilizar elementos de dos y 3 dimensiones sobre un canvas o utilizando WebGL e interactuar con elementos de la página misma (DOM).

Todos los sketches realizados con esta librería se estructuran con un archivo sketch.js que consta de dos funciones principales: setup() -> Su contenido se ejecuta una sola vez al comenzar el programa, draw() -> Se ejecuta en bucle hasta salir de la página. Es una estructura muy similar a la de processing.

Escena 3D Básica

Esta escena se inicia en un tamaño de 600 por 600 en un espacio WebGL, esto se indica en el setup() con la función createCanvas(width, height, render);

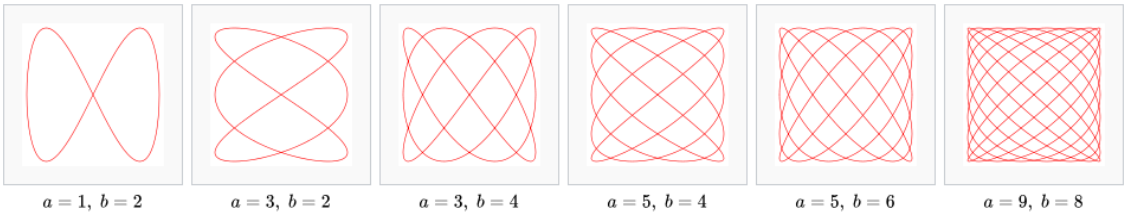


El render puede ser P2D (Bidimensional) o WEBGL (Tridimensional)

Cada vez que se dibuja sobre la pantalla se llama a la función box(altura), la cual dibuja una caja en 3D; y sphere(radio) que dibuja una esfera del radio dado. También se modifica la posición de la cámara basándose en la posición del ratón. SI el ratón se queda quieto la posición de la cámara queda fija.

Curvas de Lissajous

Son curvas generadas por la superposición de dos movimientos armónicos. Para este caso utilizaremos dos circunferencias que se calculan con distinta variación de ángulo, cogiendo de una el punto X calculado y de la otra el punto Y. Si la variación de ambas fuera la misma se dibujaría un círculo perfecto en la pantalla pero al cambiar se crean figuras de lo más curiosas.



En estas imágenes se muestran las curvas generadas por distintos conjuntos de variaciones, ***a*** es la variación del primer ángulo, y ***b*** la variación del segundo.

Para implementar esto por código se calcula punto por punto cada posición de la circunferencia y se dibuja. Para calcular tanto la posición horizontal como la vertical he creado las siguientes funciones que devuelven el punto dándole un ángulo y un radio utilizando trigonometría:

```
function pX( ang, rad )
{
    return Math.cos( PI * ang / 180 ) * rad + ( WIDTH / 2 );
}

function pY( ang, rad )
{
    return Math.sin( PI * ang / 180 ) * rad + ( HEIGHT / 2 );
}
```

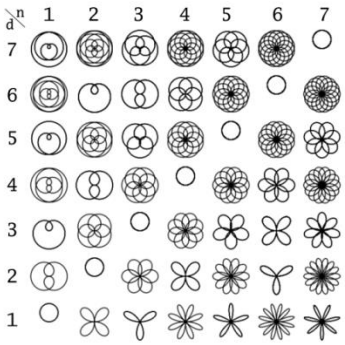
Para dibujarlo se calcula un punto específico, se almacena y se dibuja una línea entre el punto almacenado y uno nuevo:

```
function draw()
{
    var initialA = pX( a, radius );
    var initialB = pY( b, radius );
    if( isDrawing )
    {
        initialA = pX( a, radius );
        initialB = pY( b, radius );
        a += vA;
        b += vB;
        line( initialA, initialB, pX( a, radius ), pY( b, radius ) );
        loops++;
    }
    if( loops >= 360 ) isDrawing = false;
}
```

Entre el almacenaje y el nuevo cálculo se varía el ángulo (a += vA, b+= vB)

Rosas Polares

Es una fórumla que define el radio que tiene que tener un punto en un ángulo determinado según un número de “pétalos” constante:



La fórumla es la siguiente:

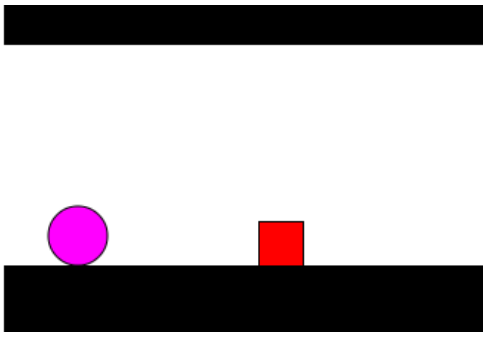
Radio(Θ) = longitudPetalos * coseno(numeroPetalos * Θ);

Θ Es el ángulo que se está dibujando, es lo mismo que una iteración desde 0 hasta 360 grados.

```
function drawRose( n, d )
{
    clear();
    var k = n / d;

    beginShape();
    stroke(0);
    noFill();
    strokeWeight(1);
    for (var a = 0; a < TWO_PI * d; a += 0.02) {
      var r = 200 * cos(k * a);
      var x = r * cos(a);
      var y = r * sin(a);
      vertex(x, y);
    }
    endShape(CLOSE);
}
```

Calculamos el número de pétalos utilizando la división de los dos valores dados y recorremos la rosa desde 0 hasta **τ (2*PI o Tau),** esto lo englobamos todo en una forma compuesta por "vertex" y una vez calculados todos los puntos lo dibujamos.

Juego Sencillo



Es un juego muy básico, se dibuja un círculo en la pantalla que tan solo puede estar en dos posiciones, o arriba (techo) o abajo (suelo). Se genera un cuadrado (enemigo) que va a acercándose a tú lado de la pantalla poco a poco y simplemente cuando llegue a la posición x,y en la que se encuentra el círculo tienes que estar en el lado contrario al del enemigo para que no te dé. Si lo saltas, la posición del enemigo vuelve a la inicial, su velocidad aumenta, su color cambia y tu puntuación sube.

En esta escena lo único que se mueven son los cuadrados de un lado de la pantalla al otro, y vuelven a aparecer al final si terminan su recorrido dando la sensación de que lo que se mueve es el personaje.

Funciones del lenguaje utilizadas

Función	Descripción
Clear()	Limpia todo el contenido del canvas
Fill()	Rellena del color que se pase como parámetro la forma a dibujar
Stroke()	Establece un trazado de la forma a dibujar
Circle()	Dibuja un círculo con los parámetros dados
Square()	Dibuja un cuadrado con los parámetros dados
Random()	Genera un número aleatorio
Cos()	Calcula el coseno de un número
Sin()	Calcula el seno de un número
Text()	Dibuja un texto en el canvas con los parámetros dados
CreateCanvas()	Crea un nuevo canvas con las propiedades especificadas
Translate()	Mueve un objeto a la posición deseada
beginShape()	Comienza una nueva figura formada por vertex()
Vertex()	Crea un nuevo vértice dentro de una figura personalizada
endShape()	Termina una figura personalizada formada por vertex()
noFill()	Hace que no se coloree el interior del próximo dibujo
strokeWeight()	Grosor del trazo de la siguiente figura
Line()	Dibuja una línea entre los puntos dados
Smooth()	Aplica suavizado sobre el canvas para no ver demasiados píxeles
Background()	Colorea el fondo de la escena
rotateX()	Rota la cámara tridimensional en el eje X
rotateY()	Rota la cámara tridimensional en el eje Y
Box()	Dibuja un cubo en una escena 3D
Sphere()	Dibuja una esfera en una escena 3D

Librería Parallax

Introducción

Parallax es una librería que nos va a permitir animar el contenido de nuestras páginas web de manera que estos reaccionen a la orientación de nuestro dispositivo o a la posición de nuestro cursor en caso de que estemos trabajando en un ordenador.

Su funcionamiento es más sencillo de lo que parece ya que básicamente vamos a trabajar con distintas capas las cuales van a tener siempre un atributo definido → **la profundidad (1)** y dependiendo de su profundidad dentro de una **escena (2)** la animación será más o menos visible.

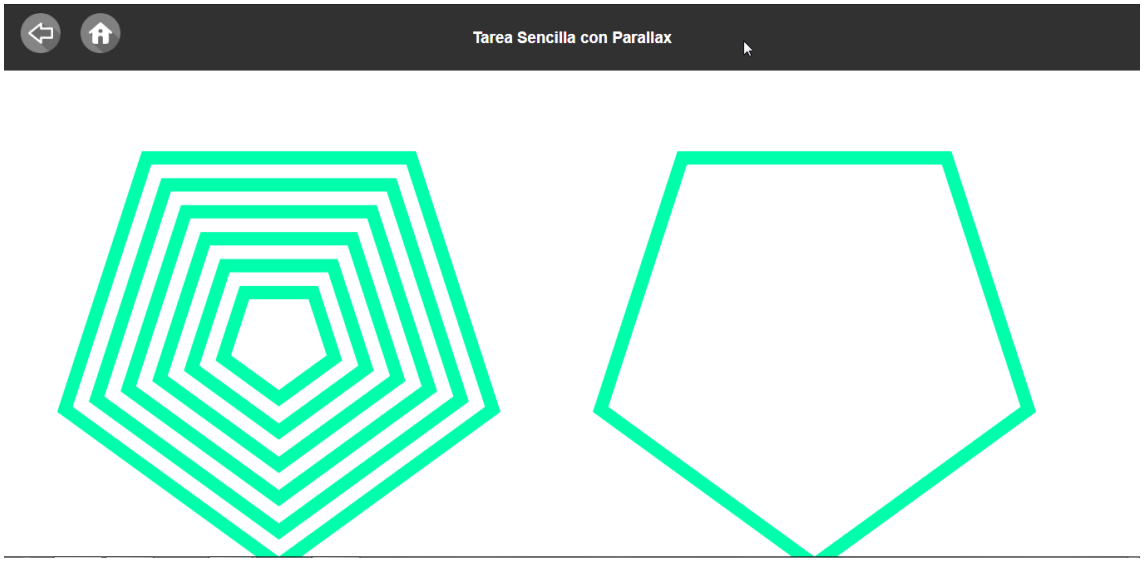
```
<ul class="scene" id="scene">
    <li class="layer estrellas" data-depth="0.20">
```
(1)

```
var escena = document.getElementById('scene');

var parallaxInstance = new Parallax(escena, {
  relativeInput: true,
  hoverOnly: true,
});
```
(2)

Siendo este el funcionamiento básico de los elementos de la librería, vamos a pasar a ver qué hemos realizado en los dos tipos de escenas de este apartado del proyecto:

Escena Sencilla

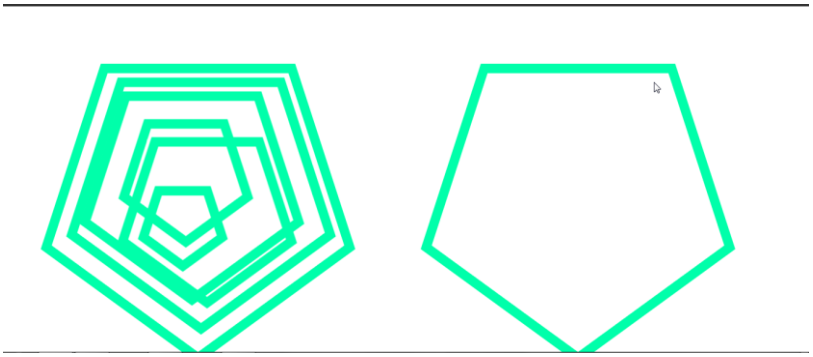
Para la escena más sencilla lo que hemos querido demostrar es el funcionamiento básico de la librería utilizando varias capas y las distintas posibilidades de configuración que las podemos aplicar.



El funcionamiento de esta escena se resume en que cuando se realice algún tipo de interacción con el elemento de la derecha de la página, las capas de la escena de la izquierda reaccionen y en función de su profundidad se muevan de una manera distinta.

```
<div class="content">
    <div class="container">
      <div id="scene" class="scene border" data-input-element="#scene-input">
        <div data-depth="1.00"><img src="../../../IMG/parallaxIMGS/simple Scene/layer1.png"></div>
        <div data-depth-x="0.80" data-depth-y="-0.80"><img src="../../../IMG/parallaxIMGS/simple
        Scene/layer2.png"></div>
        <div data-depth-x="-0.60" data-depth-y="0.60"><img src="../../../IMG/parallaxIMGS/simple
        Scene/layer3.png"></div>
        <div data-depth-x="0.40" data-depth-y="-0.40"><img src="../../../IMG/parallaxIMGS/simple
        Scene/layer4.png"></div>
        <div data-depth-x="-0.20" data-depth-y="-0.20"><img src="../../../IMG/parallaxIMGS/simple
        Scene/layer5.png"></div>
        <div data-depth-x="0.00" data-depth-y="-0.00"><img src="../../../IMG/parallaxIMGS/simple
        Scene/layer6.png"></div>
      </div>
    </div>

    <div class="container">
      <div id="scene-input" class="scene border">
        <div><img src="../../../IMG/parallaxIMGS/simple Scene/layer6.png"></div>
      </div>
    </div>

</div>
```

Como se puede ver, el elemento de la izquierda está compuesto por varias capas con diferente profundidad y en cada una de ellas hay una imagen, a su vez para que esta escena reaccione a la interacción que se haga con el elemento de la derecha, añadimos el atributo **data-input-element** que apunta al ID de este.



Escena Compleja

Para esta escena, hemos optado por demostrar el funcionamiento de la librería creando una escena bastante habitual en muchos sitios de la web hoy en día

- Usando diferentes capas entre las que vamos a tener un fondo y vamos a crear un efecto de superposición entre ellas
- Animaciones constantes que se ejecutan aunque el usuario no realice una interacción
- Efecto Parallax de la propia librería

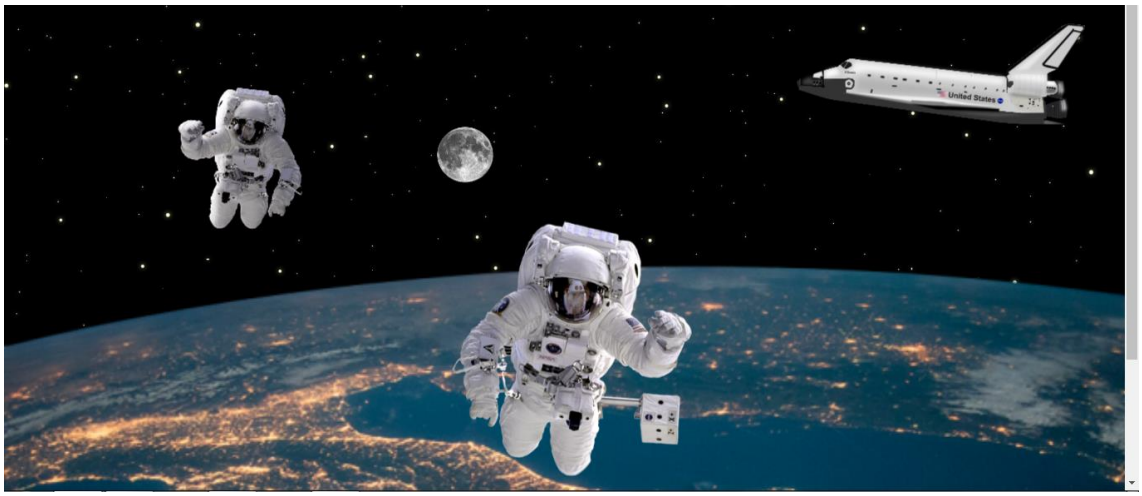
Para crear el efecto de reacción frente al cursor o movimiento del dispositivo hemos vuelto a crear una nueva escena y a determinar que solamente tenga reacción cuando se produzca el efecto hoverOnly y que sea un desplazamiento relativo al movimiento o posición del cursor: (*no tiene más complicación que esto para lograr el efecto deseado*)

```
var parallaxInstance = new Parallax(escena, {
  relativeInput: true,
  hoverOnly: true,
});
```

En este caso hemos optado por crear una estructura diferente a la de la escena sencilla, ya que en ella utilizábamos un conjunto de elementos anidados dentro de la “escena”, pero en este caso hemos utilizado una lista <ul> como “escena”, lo cual ha facilitado bastante más el trabajo que suponía crearla:

```
<ul class="scene" id="scene">
    <li class="layer estrellas" data-depth="0.20">
        <img src="../../../IMG/parallaxIMGS/complexScene/stars1.png">
    </li>
    <li class="layer estrellas" data-depth="0.20"><img src="../../../IMG/parallaxIMGS/
    complexScene/stars2.png"></li>
    <li class="layer tierra" data-depth="0.40"><img src="../../../IMG/parallaxIMGS/
    complexScene/tierra.png"></li>
    <li class="layer luna" data-depth="0.20"><img src="../../../IMG/parallaxIMGS/complexScene/
    luna.png"></li>
    <li class="layer astronauta" data-depth="0.30"><img src="../../../IMG/parallaxIMGS/
    complexScene/astronauta.png"></li>
    <li class="layer astronauta2" data-depth="0.70"><img src="../../../IMG/parallaxIMGS/
    complexScene/astronauta2.png"></li>
    <li class="layer nave" data-depth="0.30"><img src="../../../IMG/parallaxIMGS/complexScene/
    naveEspacial.png"></li>
</ul>
```

Lo siguiente ha sido aplicar estilos y animaciones las cuales se pueden ver en la propia presentación y con esto logramos una escena más compleja y más trabajada:



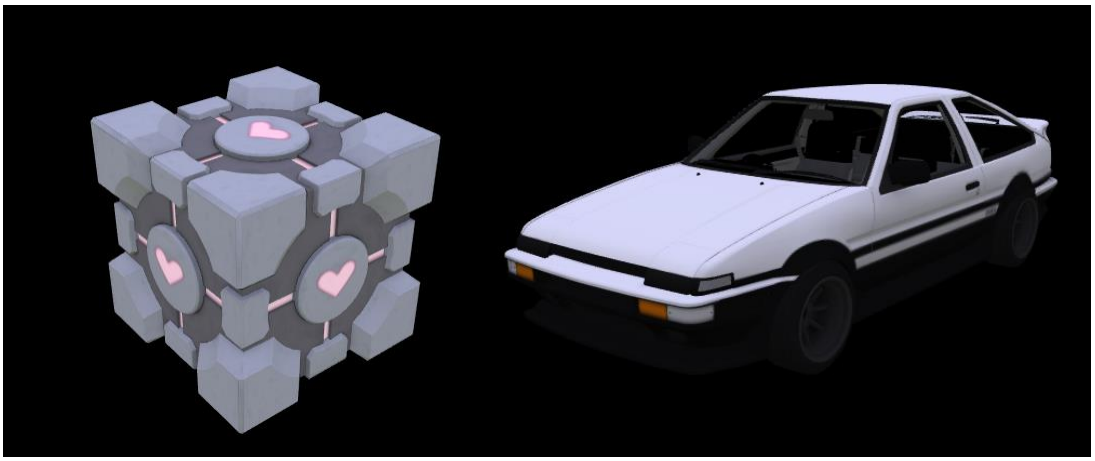
Librería Three

Introducción

Usada para crear y mostrar gráficos animados por ordenador en 3D en un navegador Web y puede ser utilizada en conjunción con el elemento canvas de HTML5, SVG o WebGL. Se ha popularizado como una de las más importantes para la creación de las animaciones en WebGL.

three.js

Cubo 3D / Coche 3D



Tanto el cubo tridimensional como el Toyota están dibujados en pantalla de la misma manera. Para mostrarlos lo primero es descargar los modelos de internet. La librería THREE.JS acepta una gran variedad de formatos tridimensionales pero en este caso hemos usado el formato GLTF adaptado a la web. Estos modelos vienen con un archivo de descripción, un archivo binario y las texturas del modelo.

Para cargar un modelo en el formato que sea necesitamos un “Loader” apropiado. Un loader es un módulo de THREE creado para interpretar un formato de modelo específico y dárselo a THREE de forma que este lo entienda. Ya que son modelos GLTF utilizamos el cargador GLTFLoader.js que se incluye con THREE.js.

Si queremos crear cualquiera de estas dos escenas necesitamos crear un contenedor, un controlador de cámara, una cámara, una escena, un renderizador y un punto de luz. Estos se definen al principio del script y posteriormente se inicializan en la función init.

```
var container, controls;
var camera, scene, renderer, light;
```

```
container = document.createElement( 'div' );
document.body.appendChild( container );

camera = new THREE.PerspectiveCamera( 45, window.innerWidth
camera.position.set( - 1.8, 0.9, 2.7 );

controls = new THREE.OrbitControls( camera );
controls.target.set( 0, - 0.2, - 0.2 );
controls.update();

scene = new THREE.Scene();

light = new THREE.HemisphereLight( 0xbbbbff, 0x444422 );
light.position.set( 0, 1, 0 );
scene.add( light );
```

El contenedor se establece como un nuevo div creado dentro del documento, para la cámara tenemos un objeto llamado PerspectiveCamera dentro de THREE al que le pasamos el tamaño de la cámara, posteriormente definimos su posición.

Los controles serán orbitales, lo que significa que el movimiento de nuestro ratón se traducirá como una rotación orbital con centro nuestra escena. Le deberemos de indicar la cámara y la posición en la que se tiene que centrar.

Para la escena nos basta con inicializarla como un objeto de tipo Scene y agregarle un punto de luz de tipo HemisphereLight al que le especificamos el color, la intensidad y la posición.

```
var loader = new THREE.GLTFLoader().setPath( 'AE86/' );
loader.load( 'scene.gltf', function ( gltf )
{
    scene.add( gltf.scene );
}, undefined, function ( e )
{
    console.error( e );
} );
```

Para utilizar el modelo 3d descargado lo primero que tenemos que hacer es inicializar un objeto de tipo GLTFLoader dentro del espacio de nombres de THREE, y le indicamos la carpeta en la que se encuentra tanto el modelo como las texturas.

Posteriormente llamamos a la función load con el nombre del descriptor gltf y una función que añada el contenido cargado a la escena.

```
renderer = new THREE.WebGLRenderer( { antialias: true } );
renderer.setPixelRatio( window.devicePixelRatio );
renderer.setSize( window.innerWidth, window.innerHeight );
renderer.gammaOutput = true;
container.appendChild( renderer.domElement );

window.addEventListener( 'resize', onWindowResize, false );
```

Por último tendremos que inicializar el motor de renderizado, utilizaremos WebGLRenderer con antialiasing activado como opción (Para suavizar la escena), le indicamos ratios, colores y tamaños de la escena y lo añadimos a nuestro contenedor, el cual hemos creado al principio

del script. También creamos una función para que se redefinan los parámetros en caso de que la ventana cambie de tamaño. Con esto tenemos todo lo necesario para mostrar nuestro objeto.

```
function animate()
{
    requestAnimationFrame( animate );
    scene.rotation.y += 0.005;
    renderer.render( scene, camera );
}
```

También necesitaremos una función “anímate” la cual entra en bucle, aumenta su rotación a cada frame y renderiza el resultado para poder visualizarlo.

Escena 3D compleja

Para esta tarea lo que hemos querido realizar ha sido crear una situación en la que se da el efecto de que nos encontramos dentro de un "cubo" y que a su vez una serie de elementos generados por la librería se encuentran dentro con nosotros y se mueven libremente.

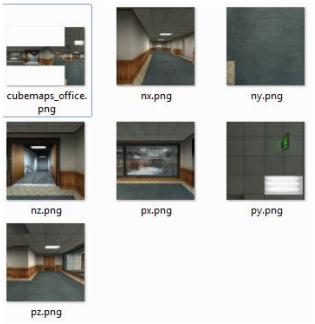
Eso sería la descripción básica de esta tarea, pero a su vez hemos implementado otras funcionalidades como por ejemplo:

- Tener una sensación de perspectiva y profundidad a través de las capas que componen el cubo
- Poder hacer un desplazamiento con el propio ratón que nos permita ver cómo se generan los elementos

Los pasos que hemos seguido han sido los siguientes:

1. **Creación del "cube map"** → Para esta tarea hemos utilizado la conocida técnica de cubeMapping (*)
   Lo primero que hemos tenido que realizar para poder implementar el aspecto dentro del "cubo" ha sido crearnos un mapa a partir de una serie de imágenes para después colocarlas en cada una de sus caras mediante código. Lo que hemos hecho ha sido encontrar un "cubeMap" que nos gustase:

   Lo siguiente ha sido recortar esta imagen usando Photosop para obtener las diferentes capas que van a ir sobre las caras del cubo:



   Una vez recortada, podemos avanzar al siguiente paso.

(*)*En los gráficos por ordenador, el mapeo de cubos es un método de mapeo del entorno que utiliza las seis caras de un cubo como forma del mapa. El ambiente se proyecta en los lados de un cubo y se almacena como seis texturas cuadradas, o se despliega en seis regiones de una sola textura.*

2. **Crear un contenedor, un controlador de cámara, una cámara, una escena, un renderizador, …** → Igual que en la tarea anterior pero con variaciones para crear el nuevo efecto
    a. **Creación de la escena** → Para esta escena hemos tenido que establecer un nuevo cargador de texturas que nos permita coger las diferentes capas que hemos creado a partir del cubeMap anterior.

```
scene = new THREE.Scene();
scene.background = new THREE.CubeTextureLoader()
    .setPath( 'texturas/' )
    .load( [ 'px.png', 'nx.png', 'py.png', 'ny.png', 'pz.png', 'nz.png' ] );
```

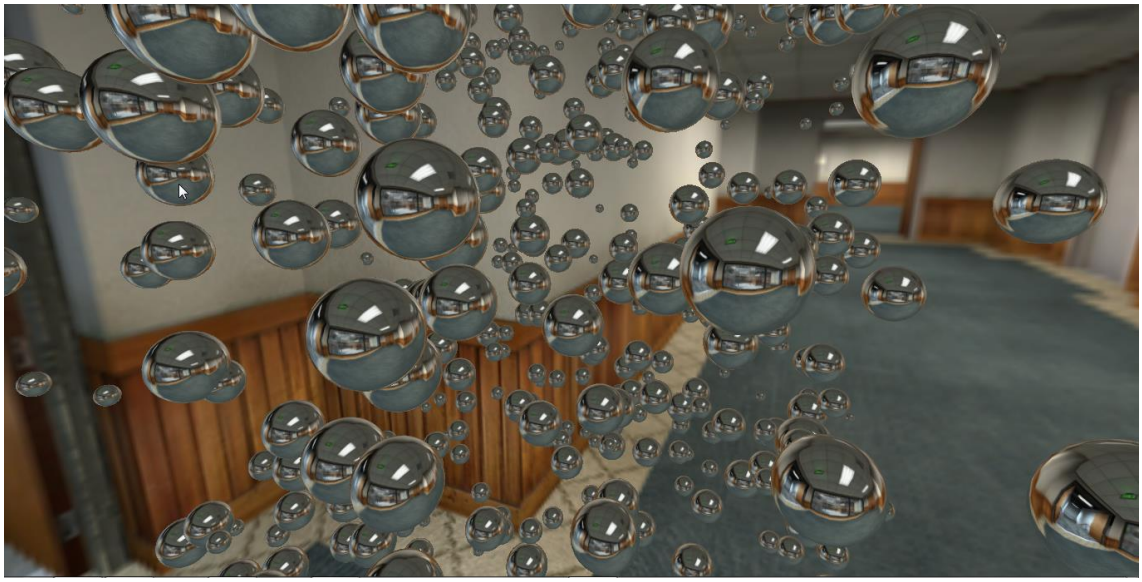
    b. Creación de los elementos (esferas) → Para crear los elementos hemos tenido que establecerles una serie de propiedades como el color, material, … e indicar el "mapa" en el que se van a generar (creado anteriormente). Finalmente a partir de un bucle generar el número de esferas que queramos e ir añadiéndolas a la escena.

```
var geometry = new THREE.SphereBufferGeometry( 100, 32, 16 );
var material = new THREE.MeshBasicMaterial( {
    color: 0xffffff,
    envMap: scene.background
});

for ( var i = 0; i < 500; i ++ )
{
    var mesh = new THREE.Mesh( geometry, material );

    mesh.position.x = Math.random() * 10000 - 5000;
    mesh.position.y = Math.random() * 10000 - 5000;
    mesh.position.z = Math.random() * 10000 - 5000;

    mesh.scale.x = mesh.scale.y = mesh.scale.z = Math.random() * 3 + 1;

    scene.add( mesh );
    spheres.push( mesh );

}
```

3. Establecer eventos → Además de la creación del renderizador, escena, … hemos establecido eventos para el desplazamiento del ratón o la redimensión de la ventana que permiten que el efecto creado no se destruya (*mostrado en la presentación*)

```
function onWindowResize()
{

    windowHalfX = window.innerWidth / 2;
    windowHalfY = window.innerHeight / 2;

    camera.aspect = window.innerWidth / window.innerHeight;
    camera.updateProjectionMatrix();

    renderer.setSize( window.innerWidth, window.innerHeight );

}

function onDocumentMouseMove( event )
{
    mouseX = ( event.clientX - windowHalfX ) * 15;
    mouseY = ( event.clientY - windowHalfY ) * 15;

}
```

4. **El efecto conseguido:**



4. Bibliografía y repositorio

- Nuestro repositorio, aquí podéis ver nuestro código:
  - https://github.com/1337luis/ProyectoDC

- Librerías utilizadas:
  - https://p5js.org/
  - http://matthew.wagerfield.com/parallax/
  - https://threejs.org/