

Futurama virtual city

vrandkode.github.io/w2vcity

Álvaro · López



Esqueleto del proyecto

El proyecto esta estructurado como un código html, código javascript (city.js) con la implementación de la ciudad y una carpeta de librerías relacionadas con Three.js. Hemos necesitado de las siguientes herramientas:

- Three.js
- Extensiones de Three.js
 - Controllers. Controladores de dispositivos de tipo ratón para el manejo de la cámara.
 - Data.gui. Controlador y vista de controles.
 - Object loader. Carga de objetos generados y geometrías complejas.

El **index.html** es la página base del proyecto y contiene la estructura de la página que pedirá acceso a las librerías javascript y donde se dibujará sobre un objeto "canvas". El canvas como lienzo es generado y añadido al arbol DOM del cuerpo de la página de forma programática, como veremos.

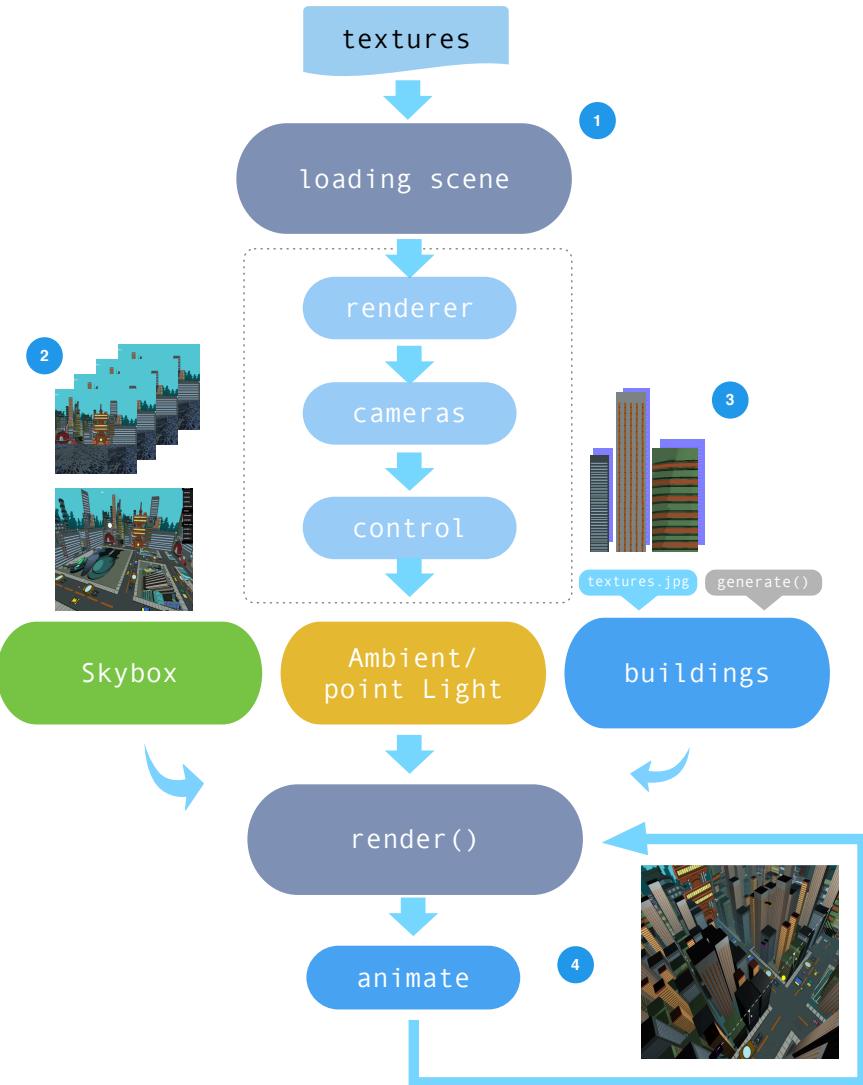
En el cuerpo se realizará la llamada para la inicialización de la rutina en city.js para la carga de las escenas sobre el canvas.

```
<html>
  <head>
    <title>Virtual city</title>
  </head>
  <body onload="init()">
    <script src="js/three/three.min.js"></script>
    <script src="js/three/OBJLoader.js"></script>
    <script src="js/three/dat.gui.js"></script>
    <script src="js/three/controls/TrackballControls.js"></script>
    <script src="js/city.js"></script>
  </body>
</html>
```

Flujo de la aplicación

Fases y partes de la aplicación mas interesantes:

1. **Loading scene.** Creamos, configuramos y generamos la escena con los componentes.
2. **Creación del Skybox.** Reunión de texturas para generar el cubo con las caras interiores con las texturas definidas.
3. **Generación de edificios:** programatica y de diferente naturaleza; con respecto al material y elección de textura.
4. **Animación de objetos,** no fijos: coches.



Leyendo la escena

Lo importante en esta parte de la ciudad es la generación del contexto de OpenGL, creación del canvas, añadirlo a la estructura Dom del cuerpo de la página y añadir componentes gracias a las librerías Three.js. En esta parte, lo más importante es la definición de las cámaras en perspectiva y ortogonal que serán vinculadas y configurables en todo momento con los controles UI que proporciona Three.js (`data.gui.js`). El componente UI permite llevar a cabo acciones a modo de panel de control y poder vincular atributos de la escena o elementos a dichos controles; inclusive la posición de la cámara.

En `loadScene()` añadiremos las **partes de la ciudad**, por lo general por cada componente se va preparando el material, la geometría, aplicar el material, definir la posición inicial (o fija si es inamovible) y finalmente se añade a la escena. Se tiene los siguientes componentes de gran relevancia:

- **Skybox:** las texturas de los planos de cada cuadrante se cargaron antes de iniciar la carga de escenas gracias a la función LoadTextureCube. La geometría esta basada en un cubo [2048]³.

```
const skyboxPlanes = [ "posx.jpg", "negx.jpg",
                      "posy.jpg", "negy.jpg",
                      "posz.jpg", "negz.jpg"];
skyboxTextures = THREE.ImageUtils.loadTextureCube(skyboxPlanes.map(
  x => texturesPath + "skybox/" + x));
```

Las **texturas** han sido creadas y editadas de fuentes de imágenes bajo fuente de inspiración en la temática de la comedia TV llamada Futurama. Han sido manipuladas para ajustarse a un tamaño 2^n ; y coinciden entre ellas para formar y texturizar finalmente las caras internas de un cubo (Three.Backside).

En la generación del **shader de fragmentos y de vértices**, se utiliza la dada por la librería como "cube", por ejemplo el de vértices, su contenido se encuentra en [cube_vert.gsl](#); junto con todos los códigos Shaders disponibles por Three.js ([shaders/ShaderLib](#)).

- Skybox floor: es un plano añadido para conseguir un efecto de suelo a modo de calles de índole futurista y con estilo cartoonizado con el fin de adaptarse a la temática general de la ciudad.
-
- **Puntos de luz, y de ambiente.**
- **Niebla (fog).** Se ha añadido un efecto de nieble localizado en (700,1950) de manera que coincide a la parte más cercana del plano Far.
- **Edificios.** En esta parte del código se van a generar edificios de forma programática y con naturaleza distinta dependiente del factor de aleatoriedad (1,4). Existen cuatro modelos de edificios con texturas y mapa de normales generadas; y los restantes son generados de forma programática, con ayuda de los pasos seguidos en el libro Three.js Essentials.
 - Las texturas han sido preparadas en la función `loadingTextures()` y de forma asíncrona; que se obtienen desde la carpeta de texturas/buildings como ficheros de texturas con nomenclatura `build<N>.jpg` y de normales como `map<N>.jpg`.
 - **Configuraciones: dimensiones y naturaleza.** Se han definido cuatro tipos de edificios con respecto a sus dimensiones. Se genera una estructura que define un layout que utilizaremos durante la generación procedural y disposición sobre el plano.

```
// de 1 a 4
let type = () => Math.floor(Math.random() * (4 - 1 + 1)) + 1;

// Configuraciones
let dimensions = [ { w:100, h: 250, d: 100, r: true},
                   { w: 80, h: 450, d: 80 },
                   { w:100, h: 200, d: 100 },
```

```
{ w: 50, h: 300, d: 50, r: true}];
```

- **Instancia de edificios: geometría y material.** Existen dos tipos de edificios con respecto al material: los que utilizan una de las texturas existentes con su mapa de normales; o las que usan texturas generadas automáticamente.



Para la generación de texturas de forma programática (ver *generateTexture()*):

1. Creamos un canvas pequeño para utilizar como lienzo de partes del edificio a colorear: para las filas de color naranja oscuro de tamaño pequeño o amarillas.
2. Creamos el canvas con tamaño $2n$ al que convertiremos en textura. Sobre él se pintará de forma programática añadiendo efecto de aleatoriedad en los colores.
3. Dibujamos el canvas pequeño sobre el grande y lo añadimos como textura al material, sin mapa de normales.

- Destacamos además que los techos se han blackpixelado para mejorar el aspecto visible desde arriba de los edificios. La instancia se realiza con la función :

```
let addBuilding = function(nth, dimension, position)
```

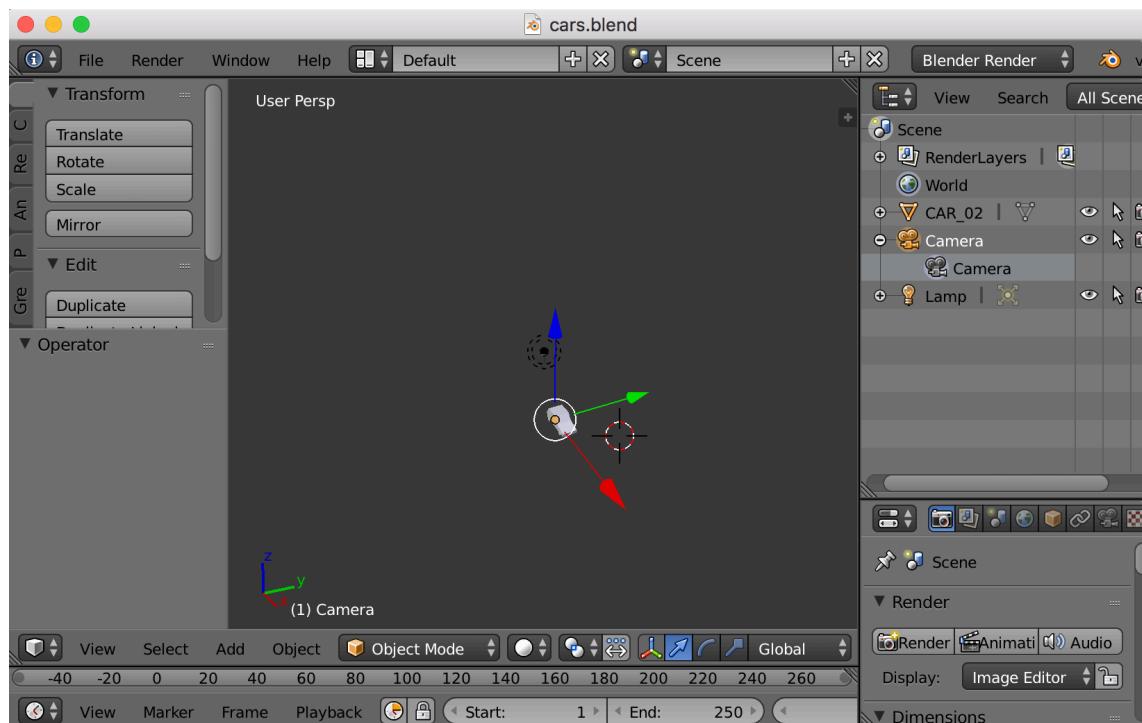
- Se generan los **edificios de forma arbitraria** dependiendo de la rejilla establecida. Dado que el máximo de ancho en dimensiones será de 120x120; se ha dividido el plano de colocación en celdas de 120x120, por lo que se podrán generar un máximo de 20 filas y 20 columnas (< 40 edificios).

```
for (let i = -8; i < 10; i++) {
    for (let j = -8; j < 10; j++) {
        if (j!=1 && j!=2 && i!=4 && i!=-5 && i!=3 && j!=6 && j!=9){
            let n = type() - 1;
            scene.add(addBuilding(n, dimensions[n], {x: j*120, y:0,z:i*120}));
        }
    }
}
```

- Dentro del bucle interno, se han añadido las restricciones que nos permiten obviar generar edificios si aparecen en las cuadrículas correspondiente a las calles, tal y como podemos comprobar en un dibujo básico de la cuadrícula del suelo:



- **Carga y animación de objetos.** Finalmente para obtener un efecto vivo sobre la temática elegida, hemos añadido la capacidad de insertar objetos ya creados con una geometría utilizando Blender: los automóviles de carácter futurista.



Para ellos hemos hecho uso de la librería Three.ObjectLoader y de Blender para generar el objeto adecuado con la forma de un coche.

En el bucle de renderizado hemos añadido la opción de animar objetos, es decir, modificar la posición con respecto a (x, y, z) y dada una velocidad (atributo de un objeto o clase creada - CarObject).

```
(car,nth) => {
    if (car.obj && car.obj!=null){
        if (car.direction == 'z'){
            car.position.z = (car.position.z < -600 ||car.position.z > 600) ?
                -car.startAt.z : car.position.z - car.speed;
        }else {
            car.position.x = (car.position.x < -700 ||car.position.x > 700) ?
                car.startAt.x : car.position.x + car.speed;
        }
        car.obj.position.z = car.position.z;
        car.obj.position.x = car.position.x;
        car.obj.position.y = car.position.y;
    }
}
```

Hemos dispuesto los coches en las calles sobre diferentes alturas en el eje x y con distintas velocidades. Todos ellos al sobrepasar unos límites se vuelven a posicionar en el mismo eje en el valor inicial.