Three JS

Guillermo Hernández González
Lucas Christian Bodson Lobato

What are we going to learn?

- What is Three js
- How to setup a project using Three js
- Model some simple figures in 3d
- Apply textures to an object
- Configure the camera
- Apply some lights to the scene
- Control the camera with your mouse

What is Three js

- Three.js is a cross-browser JavaScript library and application programming interface (API) used to create and display animated 3D computer graphics in a web browser.
- It was built to make the usage of WebGL in javascript easier.

Setting up a Three js Project

Setup a project using Three js

There’s two ways of doing it:

1. Using a CDN link

```
<script type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r83/three.js" /></script>
```

2. Installing it using npm

```
>> npm install three
```

Setup a project using Three js

Now that we have the library included, we can start writing our program.

Every Three js program will need to have at least the next variables:

1. Scene
2. Camera
3. Renderer

Creating a scene object

Scenes allow you to set up what and where is to be rendered by three.js. This is where you place objects, lights and cameras.

Code:

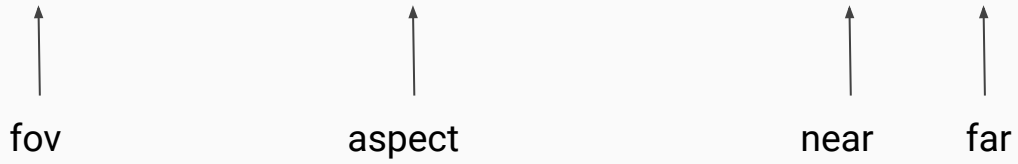
```
let scene = new THREE.Scene();
```

Creating a camera object

We need a camera “looking” at our scene for it to be displayed in our screens.

There are several types of cameras in Three js, but we only are going to use the “PerspectiveCamera”.

```
let camera = new THREE.PerspectiveCamera(75, window.innerWidth/window.innerHeight, 0.1, 1000);
```



Creating a renderer object

This object will be responsible of rendering our scene our scene.

For the examples we'll be using the WebGL renderer because it's the easiest to setup.

```
let renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);
```

The WebGL renderer has a lot of properties.

Rendering our scene

We need a render loop to have our scene show up on the screen.

We'll use a recursive function.

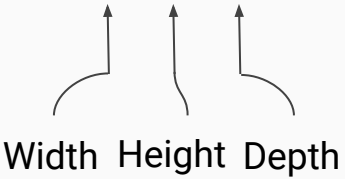
```
function render() {

    requestAnimationFrame(render);

    renderer.render(scene, camera);

  }
```

Geometric Figures

Create a geometric figure

We need to call the constructor of the figure.

```
let geometryFigure = new THREE.BoxGeometry(2, 2, 2);
```



Here's another examples.

Materials

We can create a material simply by calling its constructor.

```
let material = new THREE.MeshNormalMaterial();
```

There are many materials that we can use.

Examples of different materials.

Mesh

A Mesh is an object that combines the geometric figure and its material.

```
let cube = new THREE.Mesh(geometryFigure, material);
```

Adding the Mesh to the Scene

Now that we have our Mesh created, we can add it to the scene and make the camera look at it.

```
scene.add(cube);

camera.position.z = 10;

camera.lookAt(cube);
```

Adding textures

We can add an image as a texture to our figures using the method TextureLoader and changing the map property of the material.

```
let loader = new THREE.TextureLoader();

let texture = loader.load('../img/rubiks.jpg');

let material = new THREE.MeshBasicMaterial({map: texture});
```

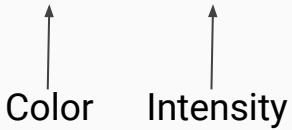
Adding lights

Lights

Adding lights can be useful or necessary in some cases.

You can add it by calling the constructor of its class.

```
let light = new THREE.DirectionalLight(0xffffff, 1.0);

light.position.set(0,1,1).normalize();
```



There are various types of Lights in Three JS

Adding lights to the scene

Adding lights to the scene has the same procedure as adding the camera or a mesh.

After creating the object, we use the method “add”.

```
scene.add(lights);
```

Camera

What are camera objects

-All camera objects are extensions to the Camera class.

- Cameras serve as our point of reference for rendering a scene using the renderer.

- They also affect the way the image renders, since it takes into account different variables like the FOV.

Types of cameras

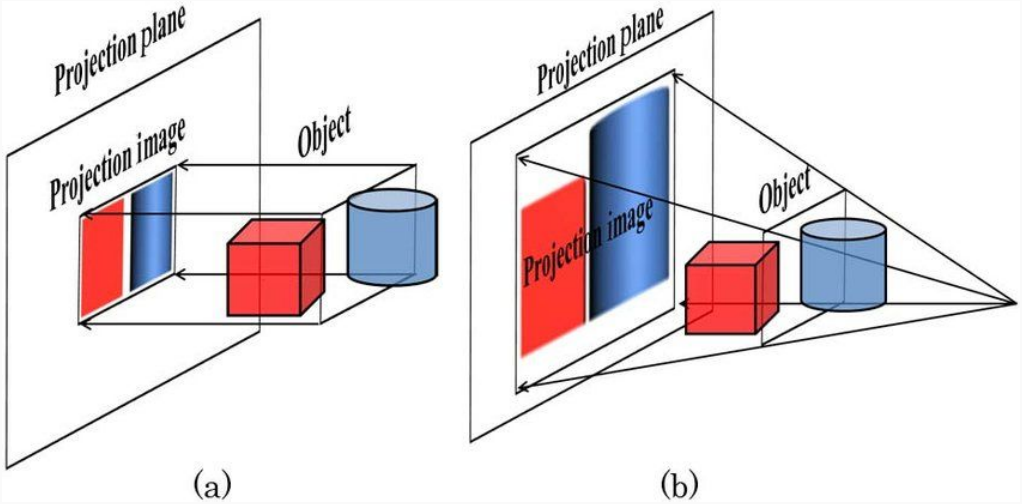
Orthographic Cameras

Perspective Cameras

Stereo cameras

Cube cameras

Array cameras



Perspective Camera properties:

- FOV (Field Of View): Represents the angle from the camera up to which it is able to see.
- Aspect ratio: It shapes the proportions of the camera.

    For a 16:8 screen the aspect ratio would be 16/8.

- Minimum render distance.
- Maximum render distance.

Camera controls

OrbitControls

We can make the user have control of the camera using the library/add-on “Orbit Controls” for Three Js.

Orbit controls allow the camera to orbit around a target.

Install and usage

We can install it using CDN:

```
<script src="https://unpkg.com/three@0.85.0/examples/js/controls/OrbitControls.js"></script>
```

And we instance it in our program:

```
let controls = new THREE.OrbitControls(camera);
```

Our camera

Configure the controls

As we instance the object, a set of default controls are installed for the camera.

We can change those editing the mouseButtons or keys properties.

```
controls.mouseButtons = {
    LEFT: THREE.MOUSE.ROTATE,
    MIDDLE: THREE.MOUSE.DOLLY,
    RIGHT: THREE.MOUSE.PAN
}
```

```
controls.keys = {
    LEFT: 37, //left arrow
    UP: 38, // up arrow
    RIGHT: 39, // right arrow
    BOTTOM: 40 // down arrow
}
```

Configure the controls

To keep our configuration, we have to use the method update everytime the user moves the camera or everytime we change the configuration.

If controls.enableDamping or controls.autoRotate are enabled we have to update in every frame.

```
function render() {
  requestAnimationFrame(render);
  controls.update();
  renderer.render(scene, camera);
}
```

Control other elements

We can use orbit control with another object in the scene besides the camera, for example, the lights. The code is exactly the same as the camera.

```
let lightControls = new THREE.OrbitControls(light);

lightControls.autoRotate = true;

lightControls.update();
```

Earth example

Making the Earth with Three Js

We'll use all the things mentioned during this presentation to simulate the planet Earth.

Our objective will be doing this.

1. Setting up

The first thing we need to do is to setup all the initial variables (scene, camera, renderer).

```
let scene = new THREE.Scene();

let camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight,
0.1, 1000);

let renderer = new THREE.WebGLRenderer();

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

document.body.appendChild(renderer.domElement);
```

2. Creating the Sphere

Now we need to create the sphere, its texture and combining them with a mesh.

```
let sphere = new THREE.SphereGeometry(2, 32, 32);

let earthTexture = new THREE.TextureLoader().load('../img/earth_texture.png');

let material = new THREE.MeshPhongMaterial({map: earthTexture});

let earthMesh = new THREE.Mesh(sphere, material);
```

3. Adding light

We need to add lights because the MeshPhongMaterial is a reflective material and we need lights to see it.

```
let light = new THREE.DirectionalLight(0xffffff, 1.0);

light.position.set(0, 1, 1).normalize();
```

4. Initial position

We have to set a initial position for our planet and our camera to ensure everything will show up.

```
earthMesh.position.set(0,0,0);

camera.position.z = 5;

camera.position.x = 0;

camera.position.y = 2.9;
```

We should have something like this

5. Adding land relief

We will need to use another texture to make the relief.

We'll use bumpMap to change how the light affects the original texture to simulate mountains.

```
material.bumpMap = new THREE.TextureLoader().load('../img/earth_height.png');

material.bumpScale = 0.05;
```

Now we have something like this

6. Add reflectivity

To add reflectivity to the water, we must use another texture and change the “specularMap” property.

That property determines the brightness of each pixel.

```
material.specularMap = new THREE.TextureLoader().load('../img/earth_reflectivity.png');

material.specular = new THREE.Color('grey');
```

7. Adding clouds

We’ll use another Mesh to add the clouds. So we have to create a new sphere, a new material with new textures and create the cloud mesh.

```
let cloudSphere = new THREE.SphereGeometry(2, 32 ,32);
```

7. Adding clouds

We will need to change new properties to the material to make it look fairly realistic.

```
let clouds = new THREE.MeshPhongMaterial({
    map: new THREE.TextureLoader().load('../img/cloud_texture.png'),
    side: THREE.DoubleSide,
    opacity: 0.3,
    transparent: true,
    depthWriter: false,
    specularMap: new THREE.TextureLoader().load('../img/cloud_transparency.png'),
    specular: new THREE.Color('grey')
});
```

7. Adding clouds

Now we just create the cloud mesh and we add this Mesh to the earth one.

```
let cloudMesh = new THREE.Mesh(cloudSphere, clouds);

earthMesh.add(cloudMesh);
```

8. Camera and light controls

Now we need to use orbit control to have the controls of the camera.

```
let controls = new THREE.OrbitControls(camera);

let lightControls = new THREE.OrbitControls(light);
```

9. Adding everything to the scene

Now we just need to add everything to the scene and render it.

```
scene.add(earthMesh);

scene.add(light);

camera.lookAt(scene);

camera.rotation.x = 5.8;

controls.update();

lightControls.update();

render();
```

Bibliography

https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene

https://riptutorial.com/es/three-js

https://github.com/josdirksen/threejs-cookbook

Three.js Cookbook

https://threejsfundamentals.org/

http://davidscottlyons.com/threejs-intro/#slide-0

Thank you!

Contacts:

Github:
https://github.com/ULL-ESIT-INF-PAI-2019-2020/2019-2020-pai-threejs-presentacion-three-js

Guillermo:
alu0101121529@ull.edu.es
Lucas:
alu0101111254@ull.edu.es