Extended Reality – Lab1

—

Creating a 3D scene and interacting with it

Jean-Yves Didier

Lab goals:

- Getting used to WebGL and Three.js;
- Being able to create and animate a scene graph.

Students's requirements:

- Being able to write a program, either in javascript or C language.

Environment's requirements:

- Webbrowser supporting WebGL (Mozilla Firefox / Google Chrome / Microsoft Edge)[1] ;
- Text editor with syntax highlighting[2].

Session's preparation

Digital versions of lectures and labs are available at the following address:
https://jydidier.page.link/vmr
Documentation of Three.js API is at the following address:
http://threejs.org/docs/
For this, we will use javascript as a programming language. For the ones who are not comfortable with it, you can use the following resources:
http://eloquentjavascript.net/
http://www.w3schools.com/js/

[1] Browser ordered by preference and compliance, from best to worst.
[2] It can be done online with our own CodeLab: https://deptgi.iup.univ-evry.fr/codelab/.

Contents

This lab uses web browser resources in order to show what are the techniques, beyond programming, to model 3D scenes, which is the basic contents for extended reality applications. Here, we aim at creating a reduced solar system and navigating inside of it.

1 Solar system setup

1.1 Sun setup

For this first question, you must;

- Reproduce the tutorial (given in appendix A), that is to say:
  - Keep the HTML structure as it is;
  - Insert the script at the right place;
- Modify it by:
  - Suppressing animation;
  - Replacing the green cube by a yellow sphere with a radius of 70;
  - Modifying (optional) canvas background color to black (prototype `WebGLRenderer`, method `setClearColor()`);
  - Positioning the camera where it can view the whole scene.

1.2 Earth setup

We will add a blue sphere that we will animate:

- The sphere has a radius of 6.5 and orbits around the sun at a distance of 150 in (O, x, z) plane;
- Orbit period of your simulated Earth will be of 1 minute. To achieve this, you can modify the signature of *render* function to `function render(time)`. `time` parameter will store elapsed time (in seconds) since the beginning of the simulation;
- The sphere wil use a Lambertian material.

Do not forget to add a white point light source at the origin of the scene.

1.3 Moon setup

A third sphere of gray color is added for the Moon:

- The sphere has a radius of 1.7 and orbits arount Earth at a distance of 19 in (O, x, z) plane;
- Orbit period of the Moon will be of 5 seconds;
- This sphere will also use a Lambertian material.

```
let texture = new THREE.TextureLoader().load( "textures/EarthMap_2500x1250.jpg")
    ;
texture.wrapS = THREE.RepeatWrapping;
texture.wrapT = THREE.RepeatWrapping;
```

Listing 1: Import a texture

1.4 Texture mapping

Now, we will enhance realism by applying a texture to Earth. To do this, you will use the code snippet from listing 1 and modify Earth's material properties:

- Its color will be white;
- It will have a `map` property which will reference the texture.

Apply the same process to Sun (texture `sunmap.jpg`) and Moon (texture `moonmap1k.jpg`).

Once the previous questions completed, you should have a solar system looking like the one exposed in figure 1.

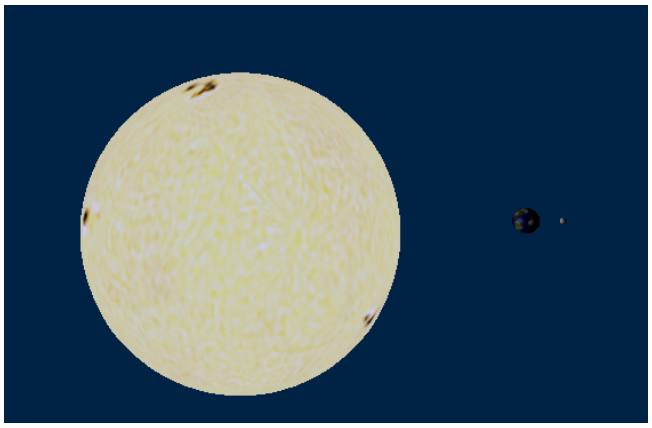

Figure 1: Sun - Earth - Moon system

2 Interacting with a scene

2.1 Let's add a slight touch of interaction

Since the browser is a window in which 3D scene is rendered, it is possible to intercept mouse and keyboard events in order to animate our scene. The web browser has a `window` objet containing a `onkeydown` property that can point to callback function handling these events.

Such a function has a unique parameter of type `KeyEvent`. The latter one owns a `keyCode` property which encode the key that has been stroked as we can see on listing 2 on the following page.

Todo:

- Modify the code sample given in listing 2 on the next page so that you can pilot how the camera moves:
  - The latter one must always look at the scene coordinate system origin;
  - key allows to go nearer the sun;
  - key allow to go further the sun;
  - Arrow keys move the camera so that it orbits at a constant distance around the Sun.

```
let myKeyboardHandler = function(keyEvent) {
  alert("Key " + keyEvent.keyCode + " has been pressed.");
}

window.onkeydown = myKeyboardHandler;
```

Listing 2: Code sample handling keyboard events

Note:

- appendix B.3 on page 8 lists codes associated to keys you must use.

2.2 Screen and camera management

By pressing the F key, we would like to go full screen.

A Web browser has tools to:

- Display a webpage element full screen: `element.requestFullscreen();`
- Escape full screen mode: `document.exitFullscreen();`

Here, the element we will use is `document.body`. It also means that the window size will change and that you will have to update the rendering area. Listing 3 shows how we can handle window resizing.

```
let function resize() {
    console.log(
        "New window size: ",
        window.innerWidth, window.innerHeight
    );
}
window.onresize = resize;
```

Listing 3: Handling window resizing

2.3 Let's go stereo

Stereo vision implies that software generates two image, one for each eye. By using the `StereoEffect.js` script you will setup a stereo effect that generates a pair of images instead of one.

Table 1 gives an abridged documentation of `StereoEffect` prototype.

Method	Effect
`StereoEffect(renderer)`	Constructor referencing the rendering area
`setEyeSeparation(sep)`	Set interpupilar distance between left and right eyes (by default, $0.064m$)
`setSize(width,height)`	Set the size of the rendering area for stereo camera
`render(scene,camera)`	Use this instead of standard rendering to achieve a stereo effect.

Table 1: Main method of `StereoEffect` prototype

2.4 Use device orientation (optional)

For the curious ones, it is possible to move the camera according to smartphone orientation. In order to achieve this, you will use `DeviceOrientationControls` prototype.

Table 2 describes the use of `DeviceOrientationControls` prototype.

Method	Effect
`DeviceOrientationControls(object)`	Constructor that takes a reference to the object we want to control by using orientation device
`connect()`	Enables orientation device tracking
`disconnect()`	Disables orientation device tracking
`update()`	Updates object orientation according to device orientation
`dispose()`	Same as `disconnect()`

Table 2: Main methods of `DeviceOrientationControls` prototype

Notes:

- At this point, your teacher will help you to setup a way to visualize your simulation on your smartphone;
- To enhance user experience, it is suitable to go full screen by touch. One way to achieve this is by replacing the opening `BODY` markup of your HTML document by the following line:

```
<body style="margin: 0; padding: 0" onclick="(document.body.requestFullscreen())()">
```

A Three.js tutorial code

A.1 Prepared HTML document

```
<html>
  <head>
    <title>First scene with Three.js</title>
    <style>
        body { margin: 0;}
        canvas { width: 100%; height: 100%;}
    </style>
  </head>
  <body>
    <script type="module">  // Javascript code is to put here. </script>
  </body>
</html>
```

A.2 Associated javascript code

```
// scene setup
import * as THREE from './js/three.module.js';
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 );

// preparing things to display
let geometry = new THREE.BoxBufferGeometry(1,1,1);
let material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
let cube = new THREE.Mesh( geometry, material );
scene.add( camera);
scene.add( cube );

camera.position.z = 3;

// rendering loop definition
function render() {
    cube.rotation.x += 0.01;
    cube.rotation.y += 0.01;
    renderer.render(scene, camera);
}
renderer.setAnimationLoop(render);
```

B Developping for web browsers

B.1 Browser configuration and security

These past years, security of web browsers have been strengthened. These features are in the interest of end-users but not necessarily of developpers. In order to run our simulation we might have to lift some browser restrictions.

B.1.1 Mozilla Firefox

1. In address bar, type `about:config`,then accept the risk;
2. Look for key `security.fileuri.strict_origin_policy` and change its value to`false`.

B.1.2 Google chrome

Launch browser with the switch `--allow-file-access-from-files`.

Under windows one way to achieve this is:

1. close all *Chrome* windows;
2. copy *Chrome* icon;
3. modify its properties (right click, properties menu);
4. in dialog box, select *Shortcut* tab;
5. modify the target and add the switch `--allow-file-access-from-files`;