Stage 2022

-

Gelles Julien

Compte-Rendu n°1

-

Semaine du 19/04 au 22/04

LUXONDES

See the electromagnetic waves

Travaux réalisés :

1. Modélisation cube en rotation – three.js

Le but était de s'initier à *three.js* avec un exemple de base, étant la modélisation d'un cube et de le faire tourner sur lui même.

Voici le code *javascript* pour cela :

```javascript
import * as THREE from 'three';

// init

const camera = new THREE.PerspectiveCamera( 70, window.innerWidth / window.innerHeight, 0.01, 10 );
camera.position.z = 1;

const scene = new THREE.Scene();

const geometry = new THREE.BoxGeometry( 0.2, 0.2, 0.2 );
const material = new THREE.MeshNormalMaterial();

const mesh = new THREE.Mesh( geometry, material );
scene.add( mesh );

const renderer = new THREE.WebGLRenderer( { antialias: true } );
renderer.setSize( window.innerWidth, window.innerHeight );
renderer.setAnimationLoop( animation );
document.body.appendChild( renderer.domElement );

// animation

function animation( time ) {

  mesh.rotation.x = time / 2000;
  mesh.rotation.y = time / 1000;

  renderer.render( scene, camera );

}
```

code javascript

Et voici le rendu du cube :

rendu du cube

La visualisation du travail plus concrètement se déroule à l'adresse suivante : https://jsfiddle.net/7u84j6kp/

2. Contrôles OrbitControl – three.js

Ici, le principe était de pouvoir contrôler la caméra et de se déplacer dans l'environnement 3D du cube. Pour cela j'ai utilisé *OrbitControl* de *three.js* qui permet de contrôler la caméra en orbite autour d'un objet, ce qui correspondait exactement a nos besoins.

Voici le code *javascript* :

```javascript
import * as THREE from 'three';
import { OrbitControls } from 'https://unpkg.com/three/examples/jsm/controls/OrbitControls.js';

const scene = new THREE.Scene();

const camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 1000);
camera.position.z = 4;

const renderer = new THREE.WebGLRenderer( { antialias: true } );
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);

const controls = new OrbitControls(camera, renderer.domElement);

const geometry = new THREE.BoxGeometry(2, 2, 2);
const material = new THREE.MeshNormalMaterial();

const cube = new THREE.Mesh(geometry, material);
scene.add(cube);

const loop = function() {
  requestAnimationFrame(loop);
  renderer.render(scene, camera);
}

loop();
```

code javascript

Le rendu du cube, ici, est exactement le même que précédemment, la seule différence est le contrôle de la caméra en utilisant la souris sur le cube.

La visualisation de ce second travail plus concret se déroule à l'adresse suivante : https://jsfiddle.net/JulienGelles/gv9ob6ep/13/

3. Modélisation plan avec image et gestion amélioré de la caméra – three.js

A ce stade, l'objectif était de pouvoir visualiser une image dans un environnement en 3 dimensions.

Il ne fallait donc plus modéliser un cube, mais un plan sur lequel devait s'intégrer une image. Pour le moment, l'image n'est pas modifiable, j'ai choisi la *Nuit étoilée* de *Van Gogh* pour exemple.

Ensuite la gestion de la caméra n'était pas optimal pour la visualisation d'un plan, j'ai donc :

-Modifié l'axe de rotation du plan lors d'un contrôle verticale de la souris (a l'aide du clic gauche). (→ ligne 8)

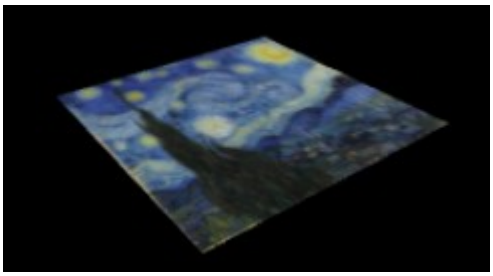
-Définis des limites d'angles lors d'un contrôle horizontal de la souris (toujours a l'aide du clic gauche), pour ne pas se retrouver avec un image retournée ou vu du dessous. (→ ligne 16)

Voici le code *javascript* :

```
1  import * as THREE from 'three';
2  import { OrbitControls } from 'https://unpkg.com/three/examples/jsm/controls/OrbitControls.js';
3
4  const scene = new THREE.Scene();
5
6  const camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 10000);
7  camera.position.z = 1;
8  camera.up.set( 0, 0, 1 );
9
10
11 const renderer = new THREE.WebGLRenderer( { antialias: true } );
12 renderer.setSize(window.innerWidth, window.innerHeight);
13 document.body.appendChild(renderer.domElement);
14
15 const controls = new OrbitControls(camera, renderer.domElement);
16 controls.maxPolarAngle = Math.PI/2-0.01;
17
18
19 const img = 'https://lh6.ggpht.com/HlgucZ0ylJAfZgusynnUwxNIgIp5htNhShF559x3dRXiuy_UdP3UQVLYW6c=s1200';
20
21 const geometry = new THREE.PlaneGeometry( 1, 1 );
22 const texture = new THREE.TextureLoader().load( img );
23 const material = new THREE.MeshBasicMaterial( { map: texture , side: THREE.DoubleSide} );
24 const plane = new THREE.Mesh( geometry, material );
25 scene.add( plane );
26
27 const loop = function() {
28   requestAnimationFrame(loop);
29   renderer.render(scene, camera);
30 }
31
32 loop();
```

code javascript

Et voici le rendu du plan imagé :



rendu du plan imagé

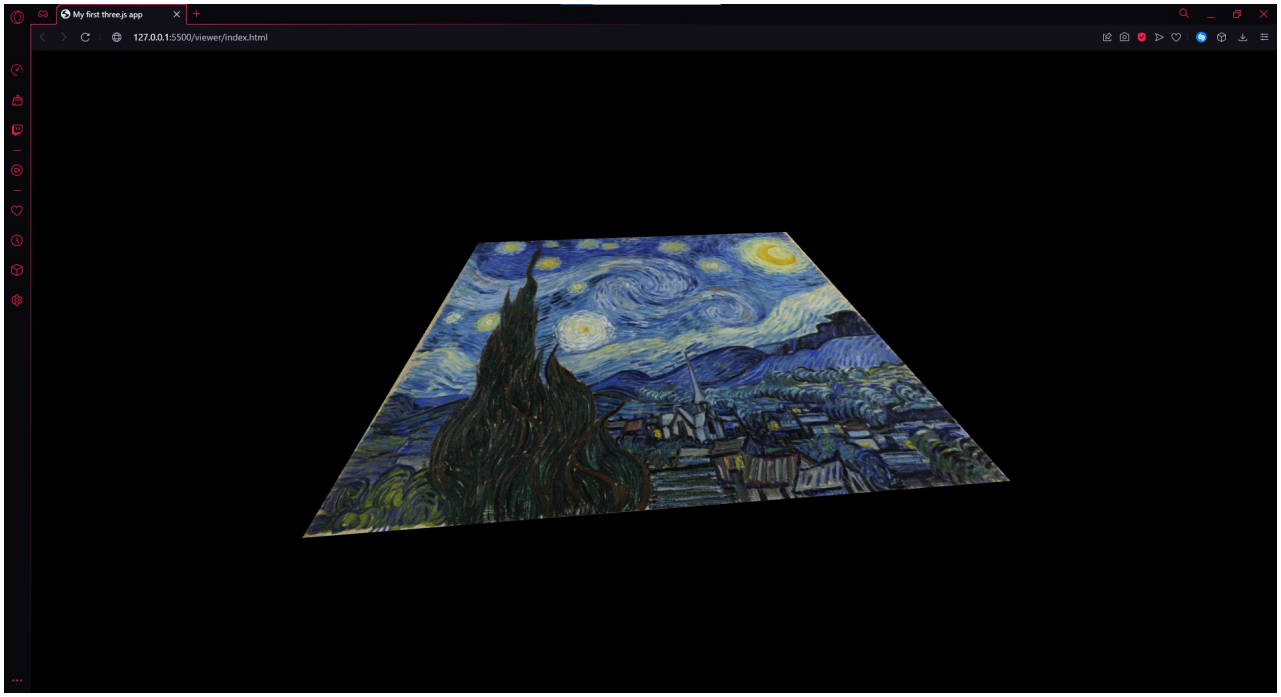
La visualisation de ce dernier travail de la semaine est plus concrètement observable à l'adresse suivante : https://jsfiddle.net/JulienGelles/gv9ob6ep/99/

Travaux suivants :

1. Gestion serveur

Pour le moment, toutes les représentations on été faite à l'aide de l'outil *fiddle* en ligne pour faciliter les tests. Néanmoins on voudrait qu'il soit possible de lancer les scripts sur navigateur, en dehors de l'outil en l'hébergeant sur un serveur, pour le moment local.

Voici ici un premier exemple du script lancé à l'aide de ce même concept de serveur local, mais ici grâce à la fonctionnalité de *Visual Studio Code* :



visualisation plan sur navigateur

2. Modification de l'image

On souhaiterais ensuite pouvoir modifier l'image du plan. Dans in premier temps avec un explorateur de fichiers.

Et a terme avec un système « drag & drop » afin de déposer l'image souhaité sur la fenêtre directement.