TP1 - Three.js

Three.js est une bibliothèque javascript pour le développement d'applications 3D avec WebGL. Vous trouverez toutes les informations (doc, exemples) à l'adresse officielle https://threejs.org. Pour ce TP, je conseille l'utilisation de Firefox mais vous pouvez vérifier notre navigateur préféré ici : https://threejs.org/docs/index.html#manual/en/introduction/How-to-run-things-locally

Pour pouvoir utiliser des textures en local :

- Firefox : entrez about :config dans la barre d'adresse. Cherchez l'option security.fileuri.strict_origin_policy et modifiez la valeur à false.
- Chrome : doit être lancé avec l'argument --allow-file-access-from-files as argument. Pour cela :
  - Dans Windows : créez un alias pour Chrome. Editez les propriétés du lien pour ajouter allow-file-access-from-files après Chrome.exe. Redémarrer Chrome avec l'alias.
  - Dans MacOS : Ouvrez le Terminal et exécutez open /Applications/Google\ Chrome.app --args --allow-file-access-from-files
- Safari (Mac) : Activez, si nécessaire, le menu Développement dans les préférences Safari. Puis, activez l'option Disable Local File Restrictions/Désactiver les restrictions de fichier local.

Décompresser les fichiers tp1.zip et tp1-models.zip de l'espace Moodle. Créez un document pour votre compte rendu, qui contiendra les captures d'écran demandées ainsi que vos explications, difficultés ou apports. Créez un nouveau fichier html, avec le code suivant, permettant de créer le canvas, élément qui sera utilisé pour afficher la scène :

```
<!doctype html>
<html lang="en">
 <head>
<meta charset="utf-8">
<title>Three.js Demo</title>
</head>

<body>
<canvas id="glcanvas" width="640" height="480"></canvas>
</body>

<script src="./libs/three.min.js"></script>
<script src="cubeThree.js"></script>
</html>
```

La première scène, très simple, va contenir un cube texturé animé. Créez un nouveau fichier pour le script cubeThree.js (code ci-dessous). L'objet Renderer représente le moteur de rendu, il est attaché au canvas et définit les dimensions de la scène (qui contient les objets). Vous avez également les objets camera et scène. Nous allons utiliser une PerspectiveCamera, pour laquelle il est possible de spécifier le champ de vision (field of view), le rapport hauteur/largeur (aspect ratio) et les plans de clipping (near and far). Une lumière directionnelle éclaire la scène. Elle permet de définir une source de lumière distante (comme le soleil), à partir d'une position, vers une cible. Par défaut, la cible est l'origine.

Pour chaque objet (THREE.Mesh), il est nécessaire de définir la géométrie et l'apparence. La géométrie dans ce cas est créée grâce au constructeur BoxBufferGeometry. Le matériau est créé en utilisant MeshBasicMaterial. Les couleurs s'expriment comme dans HTML en remplaçant le « # » par « 0x ». THREE.TextureLoader est utilisé pour définir une texture pour l'objet. Puis, la position et l'orientation de l'objet sont modifiées avant de l'ajouter à la scène. Il est nécessaire de modifier ces valeurs (ou la composant z de la caméra) pour « voir » l'objet. Très utile, la fonction lookAt (vector : Vector3) : null, or lookAt (x : Float, y : Float, z : Float) : null) modifie un objet pour faire face à un point précis e.g. camera.lookAt(scene.position).

```
var renderer = null,
scene = null,
camera = null,
cube = null;

main();

function main() {
```

```
var canvas = document.getElementById("glcanvas");

// Create the Three.js renderer and attach it to our canvas
renderer = new THREE.WebGLRenderer( { canvas: canvas, antialias: true } );

// Set the viewport size
renderer.setSize(canvas.width, canvas.height);
document.body.appendChild( renderer.domElement );

// Create a new Three.js scene
scene = new THREE.Scene();

// Add  a camera so we can view the scene
camera = new THREE.PerspectiveCamera( 45, canvas.width / canvas.height, 1, 400);
camera.position.z = 20;
camera.lookAt(scene.position);
scene.add(camera);

// Add a directional light to show off the object
var light = new THREE.DirectionalLight( 0xffffff, 1.5);

// Position the light out from the scene, pointing at the origin
light.position.set(0, 0, 10);
scene.add( light );

// Create a color cube and add it to the scene
var material = new THREE.MeshBasicMaterial( { color: 0x0000ff } );   //hex colors

   // Create the cube geometry
  var geometry = new THREE.BoxBufferGeometry(2, 2, 2);

  // And put the geometry and material together into a mesh
  cube = new THREE.Mesh(geometry, material);

 // Tilt the mesh toward the viewer
   cube.rotation.x = Math.PI / 5;
   cube.rotation.y = Math.PI / 5;

  // Finally, add the mesh to our scene
     scene.add( cube );

  // Run the run loop
 renderer.setAnimationLoop( () => {  // tells the browser we want to perform an animation
     update();
     render();
 } );};
```

Ajoutez les fonctions update() et render(). La fonction update concerne tout ce qui doit être mis à jour dans la scène.

```
function update () {
}

// render, or 'draw a still image', of the scene
function render() {
  renderer.render( scene, camera );
}
```

Testez votre script en ouvrant votre page html dans le navigateur.

Pour explorer facilement votre scène avec la souris, ajoutez la bibliothèque OrbitControls à vos fichiers :

```
Html :    <script src="./libs/OrbitControls.js"></script>

JS :      //Orbit Controls
          var orbit = new THREE.OrbitControls( camera);
```

L'utilitaire AxesHelper peut vous aider à comprendre le placement de vos objets. Il visualise les 3 axes, X en rouge, Y en vert et Z en bleu.

```
var axesHelper = new THREE.AxesHelper( 20 );
scene.add( axesHelper );
```

Testez votre script.

Ajoutez la fonction suivante à votre programme :

```
window.addEventListener( 'resize', function () {
        camera.aspect = window.innerWidth / window.innerHeight;
        camera.updateProjectionMatrix();
        renderer.setSize( window.innerWidth, window.innerHeight );
        }, false );
```

Testez la fonction en modifiant la taille de la fenêtre du navigateur. Faites une capture pour votre compte rendu.

Maintenant il est temps d'animer le cube en modifiant son orientation autour de l'axe Y. Modifiez donc la fonction update :

```
function update () {
   var angle = .005;
   cube.rotation.y += angle;
}
```

Cette fonction sera exécutée à chaque frame (normalement 60 fois par seconde).
Testez votre script.

Une lumière directionnelle a été définie, après la création de la caméra, mais le cube est encore « plat ». Pour que l'objet réagisse à la lumière vous devez modifier le matériau utilisé (de Basic à Phong) :

```
var material = new THREE.MeshPhongMaterial( { color: 0x0000ff } );
```

Testez votre script. Faites une capture d'écran et ajoutez-la à votre compte rendu.
Plus d'information sur les matériaux ici : https://threejsfundamentals.org/threejs/lessons/threejs-materials.html. Vous pouvez également tester les divers paramètres ici : http://stemkoski.github.io/Three.js/Color-Explorer.html

Modifiez maintenant le matériau pour utiliser une texture et avoir une couleur où la transparence est définie :

```
// First, create the texture map
var mapUrl = "./textures/kittenMask.png";
var texture = new THREE.TextureLoader().load(mapUrl);

// Material with texture
var material = new THREE.MeshPhongMaterial({ map: texture, emissive: 0x0000ff });
```

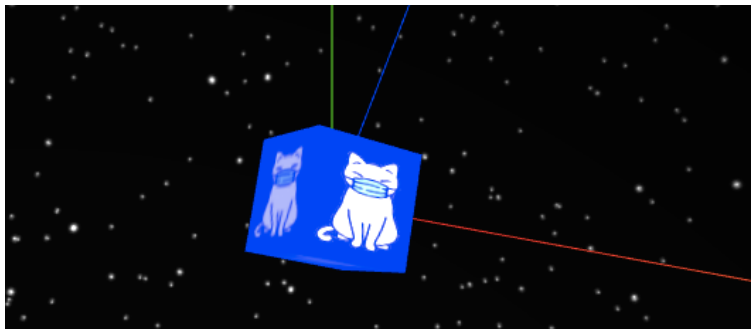
Pour améliorer l'environnement, ajoutez une skybox ainsi qu'une lumière ambiante, qui permet d'avoir un fond de lumière toujours présent :

```
// Stars' geometry and material
var starGeometry = new THREE.SphereGeometry(200, 50, 50);
var starMaterial = new THREE.MeshPhongMaterial({
map: new THREE. TextureLoader().load ("./textures/stars.png"),
      side: THREE.DoubleSide,
      shininess: 0
});

var starSkybox = new THREE.Mesh(starGeometry, starMaterial);
scene.add(starSkybox);

 var ambientLight = new THREE.AmbientLight( 0xcccccc, 0.4 );
 scene.add( ambientLight );
```

Testez votre script. Ajoutez une capture d'écran à votre compte-rendu.

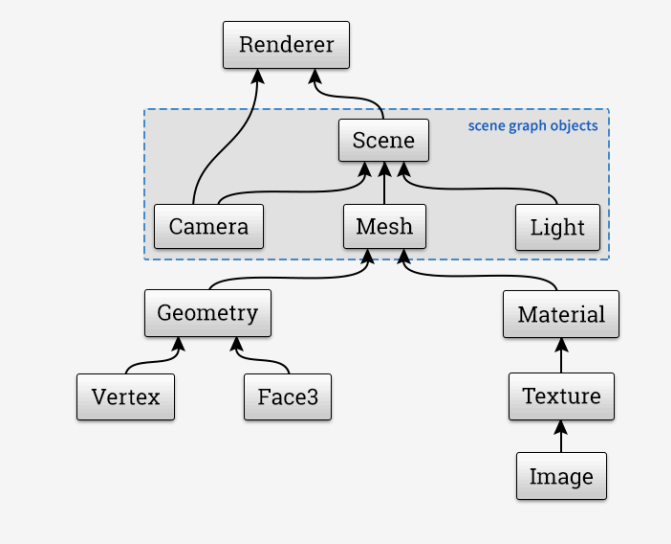


Faites une copie de votre script pour continuer.
Créez deux fonctions : createLights() et createContent() pour alléger votre fonction main(). Replacez la création de lumières et skybox dans createLigths, et la création des matériaux et géométries dans createContent.

Maintenant, dans la fonction createContent(), remplacez maintenant votre cube par un plateau placé sur le plan XZ :

```
var groundGeometry = new THREE.CylinderBufferGeometry(15, 15, 1, 32);
var groundMaterial = new THREE.MeshPhongMaterial({
map: new THREE. TextureLoader().load ("./textures/spiral-retro-background.jpg"),
        side: THREE.FrontSide,
        shininess: 2
 });
ground = new THREE.Mesh(groundGeometry, groundMaterial);
```

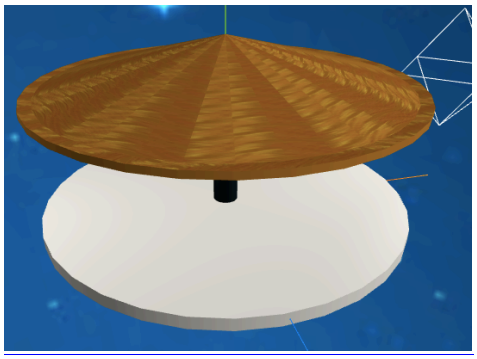
Voici le graphe de scène de notre environnement actuel :



Créez maintenant un cylindre et un cône pour obtenir l'objet de l'image ci-dessous. Vous pouvez bien évidemment modifier les textures et matériaux. Lorsque vous voulez qu'une texture se répète, vous devez modifier les paramètres de la manière suivante :

```
var texture = new THREE.TextureLoader().load( "./textures/gold.jpg" );
texture.wrapS = THREE.RepeatWrapping;
texture.wrapT = THREE.RepeatWrapping;
texture.repeat.set( 10, 10 );

   var hatMaterial = new THREE.MeshPhongMaterial({
     map: texture, side: THREE.FrontSide, shininess: 10
   });
```



Pour l'instant le plateau tourne seul, nous aimerions que tout l'objet tourne à la même vitesse et donc sans devoir s'adresser à chaque partie de manière indépendante. Pour cela et pour mieux contrôler l'application des transformations, il est possible de créer de groupes :

```
carousel = new  THREE.Group();
carousel.add(ground );
carousel.add(column);
carousel.add(hat1);
carousel.add(hat2);
scene.add(carousel);
```

Modifiez maintenant l'animation pour faire tourner la structure toute entière.

Testez votre script. Ajoutez à votre compte rendu les matériaux crées ainsi qu'une capture d'écran. Répondez également à la question : comment le graphe de scène est-il modifié ?

Déposez votre document et votre code sur Moodle ou envoyez le tout par mail à nancy.rodriguez@lirmm.fr