TP1

Première application "Hello Cube" avec Three.js

Vous allez ré-utiliser le cadre de travail mis en place au TP0 (starter-three).
L'objectif est de mettre en place les éléments classiques : scène, lumière et rendu avec quelques objets de décors simples : cube et sphère.
☺Généralement, lors de la découverte d'un langage, on code son classique "Bonjour la planète", ici, dans le monde de Three, vous allez transposer cela en "Bonjour le cube".

ÉLÉMENTS DE BASE

1. Copiez le dossier `/starter-three` et renommez le `/tp1`.
2. Dans le fichier `main.js`, copiez le code suivant qui va structurer votre travail. Les fonctions seront alimentées pendant l'avancement du TP.

```
import * as THREE from "three";

// variables globales
let scene;
let renderer;
let camera;

function main() {
  // 1. Ajouter la scène

  // 2. Mettre en place le rendu dans le canvas

  // 3. Ajouter les caméras

  // 4. Ajouter les lumières

  // 5. Ajouter les objets dans la scène

  // 6. Démarrer la boucle d'animation
}


/**
 * La boucle d'animation.
 */
function animate() {}


main();
```

Remarque : les 3 variables globales `scene`, `renderer` et `camera` seront mobilisées dans toutes les fonctions.

3. Ajoutez dans `main()` la cible du conteneur de notre futur canvas :

```
function main() {
  const container = document.querySelector("#scene-container");

  // …
}
```

4. Ajoutez et coloriser l'objet scène :

```
  // 1. Ajouter la scène
  scene = new THREE.Scene();
  scene.background = new THREE.Color("cornsilk");
```

Remarque : pour rappel, les éléments de la scène doivent être placés dans un système de coordonnées X, Y, Z
Par défaut, un nouvel élément est toujours placé au centre la scène (0,0,0).
A FAIRE : testez ce qui se passe quand aucune couleur n'est spécifiée (ni dans le CSS, ni dans le JS). Quelle est la couleur ?

Puis, il est nécessaire d'instancier **WebGLRenderer** pour mettre en place le **moteur de rendu**. C'est lui qui va venir injecter une balise **canvas** et construire les images animées dedans.
Dans cette application, le canvas doit avoir la taille de son conteneur `#scene-container`.

5. Complétez `main()` comme suit :

```
// 2. mettre en place le rendu dans le canvas
renderer = new THREE.WebGLRenderer();
renderer.setPixelRatio(window.devicePixelRatio);
renderer.setSize(container.offsetWidth, container.offsetHeight);
container.appendChild(renderer.domElement);
```

La méthode `setPixelRatio()` est utilisée pour définir un ratio entre la taille de l'écran et le nombre de pixels. Habituellement, cette valeur est égale au ratio natif de la page actuelle de notre navigateur web.
A FAIRE : avec la console du navigateur, vérifiez que la balise `<canvas>` figure bien dans le body.

Pour rappel, les 4 éléments de réglage d'une caméra sont :
Fov (**champ de vision**) qui représente la largeur du champ de vision de la caméra, en degrés.
aspect (**rapport hauteur/largeur**) qui représente le rapport entre la largeur de la scène et sa hauteur.
near (**près du plan de découpage**) détermine que tout ce qui est trop proche de la caméra sera invisible.
Far (**plan de découpage éloigné**) détermine que tout ce qui est trop éloigné de la caméra sera invisible.

6. Ajoutez un objet caméra avec ses 4 éléments de réglage :

```
// 3. Ajouter les caméras
camera = new THREE.PerspectiveCamera(
          75,
          container.offsetWidth / container.offsetHeight,
          0.1,
          100
);
scene.add(camera);
```

Remarque : pour repositionner la caméra on peut écrire :

```
camera.position.z = 5; // modifier sur l'axe Z
camera.position.set(0,0,5); // modifier sur les 3 axes
```

7. Positionnez la caméra en 2,2,15.

La scène doit être éclairée. Il faut utiliser un ou plusieurs éclairages.
L'éclairage **AmbiantLight** illumine de manière égale l'intégralité des objets de la scène.
Les deux paramètres sont :
- couleur de l'éclairage (0xFFFFFF par défaut)
- intensité de l'éclairage (1 par défaut)

L'éclairage AmbientLight n'est souvent pas suffisant car il ne simule pas une lumière naturelle.
Il est possible d'ajouter un éclairage DirectionalLight qui simule une source lumineuse émise dans une direction et ainsi améliore la restitution des effets de lumière.

8. Ajoutez le code suivant dans `render()` :

```
// 4. Ajouter les lumières
const ambientLight = new THREE.AmbientLight(0xcccccc, 1);
scene.add(ambientLight);
const directionalLight = new THREE.DirectionalLight(0xffffff, 1);
scene.add(directionalLight);
```

Pour actualiser l'affichage, le moteur de rendu doit lier la scène et la caméra.
Dans votre organisation de code, cela doit se faire dans la fonction `animate()` d'animation.

9. Ajoutez le code suivant dans `animate()` :

```
renderer.render(scene, camera);
```

10. Dans `main()` il faut appeler la fonction `animate()` :

```
function init() {
  …

  // 6. Démarrer la boucle d'animation
  animate();
}
```

Pour le moment, rien n'apparait à l'écran. Normal, il y rien dans la scène !

LES AIDES de CONTROLES

L'idée est maintenant d'ajouter des petits outils visuels qui permettent de mieux repérer (et manipuler) les éléments dans la scène.
La doc est ici threejs.org/docs/index.html#api/en/helpers/ArrowHelper
Pour cela vous allez ajouter :

- des axes
- et un quadrillage horizontal (une grille)

THREE propose quelques méthodes utiles :

- La méthode `AxesHelper(longueur)` permet tracer les 3 axes X,Y et Z.

Les axes seront ajoutés à la scène (x=axe rouge, Y=axe vert et Z=axe bleu).

- La méthode `GridHelper()` permet de poser une grille.

Three propose aussi un module complémentaire nommé **OrbitControls** qui permet de contrôler la caméra avec la souris et ainsi de tourner autour d'une cible.

11. Ajoutez une importation dans `main.js` :

```
import { OrbitControls } from 'three/addons/controls/OrbitControls.js';
```

12. Ajoutez une fonction `helpers()` comme suit :

```
/**
 * Les outils de visualisation.
 */
function helpers() {
  // les 3 axes
  const axesHelper = new THREE.AxesHelper(12);
  scene.add(axesHelper);
  // la grille (dimension 20 divisée en 20)
  const gridHelper = new THREE.GridHelper(20, 20);
  scene.add(gridHelper);
  // contrôle avec la souris
  const orbitControls = new OrbitControls(camera, renderer.domElement);
}
```

13. Dans `init()`, complétez :

```
function init() {
  …
  // 5. Ajouter les objets dans la scène
  helpers();

  // démarrer la boucle d'animation
  animate();
}
```

L'UX n'est pas encore aboutie. Normalement, quand vous déplacez la souris, tout le plateau de jeu doit bouger car c'est comme si vous déplaciez la caméra.
Pour corriger cela, il faut que la fonction d'animation soit appelée régulièrement au cours du temps pour ré-actualiser l'affichage.

14. Dans `main()` remplacez l'écriture `animate()` par `renderer.setAnimationLoop(animate)`.

Remarque : c'est l'équivalent d'une boucle infinie qui appellerait animate en permanence.

A FAIRE : consultez la documentation

https://threejs.org/docs/index.html#api/en/helpers/CameraHelper

Ajoutez une aide visuelle pour mieux comprendre vers quoi pointe la caméra.

LES ÉLÉMENTS de la SCÉNE

La création d'un objet dans Three.js se déroule en 3 phases :

- **PHASE 1** : créer une géométrie pour définir le squelette
- **PHASE 2** : poser une peau sur l'objet, on parle de matériau. Le choix du matériau va avoir une influence sur les ressources machines consommées.
- **PHASE 3** : dans cette phase, le matériau est appliqué à la géométrie, et le nouvel objet est introduit sur la scène. Cet objet est de type **Mesh**.

Le but est de créer un cube 3D en rotation. C'est le premier objet qui sera placé sur la scène … d'autres viendrons.

La classe `BoxGeometry` va être utilisée. Cette dernière permet d'instancier une Geometry rectangulaire avec trois paramètres qui déterminent les dimensions X,Y et Z. Une valeur identique permet de produire un carré.

Par défaut, le centre du cube est en (0,0,0).

Pour replacer le cube, on utilise sa méthode `position.set(X,Y,Z)`.

Pour le matériau du cube, on peut utiliser l'une des trois classes de base :

- MeshBasicMaterial

Pour les objets 3D ultra-basique sans prise en compte de l'éclairage ou shading. C'est léger et rapide.

- MeshLambertMaterial

Pour créer des surfaces non-brillantes et sans reflets lumineux

- MeshPhongMaterial

pour créer des surfaces brillantes, avec des reflets lumineux. C'est lourd et lent pour le processeur.

Les classes peuvent être configurées avec un objet JavaScript

```
{ color : …., wireframe : true|false }
```

Remarque : la propriété wireframe permet de produire un affichage en fil de fer.

15. Dans la fonction `init()`, ajoutez, juste avant `render()`, l'appel d'une nouvelle fonction `createCube1()`

```
function init() {
  …
  // 5. Ajouter les objets dans la scène
  createCube1();
  // démarrer la boucle d'animation
  renderer.setAnimationLoop(animate);
}
```

16. Ajoutez, à la liste des variables globales, la nouvelle variable cube :

```
let cube1;
```

17. Ajoutez le code de la fonction du cube :

```
/**
 * Ajouter un cube dans la scène.
 */
function createCube1() {
  const geometry = new THREE.BoxGeometry(6,6,6);
  const material = new THREE.MeshBasicMaterial({color: "orange", wireframe: false});
  cube1 = new THREE.Mesh(geometry, material);
  scene.add(cube1);
}
```

Votre cube est orange plein sans effet de lumière très avancé.

Remarque : une autre écriture plus compacte, sans la déclaration des variables geometry et material, est possible :

```
cube1 = new THREE.Mesh(
  new THREE.BoxGeometry(6,6,6),
  new THREE.MeshBasicMaterial({color: "orange", wireframe: false})
);
scene.add(cube1);
```

Vous trouverez en ANNEXE 1 les autres formes classiques des objets.

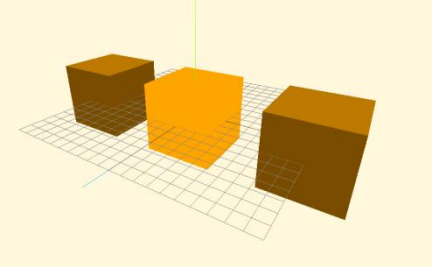
18. Modifiez la propriété `wireframe` à `true` et observez.

Pour modifier le nombre de segments du wireframe, il faut renseigner les 6 paramètres de la classe BoxGeometry (la doc est ici : https://threejs.org/docs/index.html#api/en/geometries/BoxGeometry)

```
new THREE.BoxGeometry(6,6,6,3,3,3)
```

A FAIRE : replacez le cube par une sphère de dimension 12 (nouvelle fonction `createSphere`).

A FAIRE : ajoutez deux autres cubes orange de part et d'autre de l'axe X. Mettez en place pour cela les 2 nouvelles fonctions `createCube2()` et `createCube3()`



Le cube central est habillé par MesBasicMaterial.
Le cube de droite est habillé par MeshLambertMaterial.
Le cube de gauche est habillé par MeshPhongMaterial.

ANIMATIONS

Le code de l'animation sera placé dans la fonction `animate()`.
Cette fonction est appelée avec l'instruction `renderer.setAnimationLoop(animate)`.
Indirectement, c'est bien l'instruction native `requestAnimationFrame` qui est exploitée.
Typiquement, le traitement `animate()` est relancé 60 fois par secondes.
On aurait pu utiliser la fonction de gestion des timer `setInterval`.
Un des avantages est que les animations sont figées quand l'internaute change d'onglet.

La première animation est de faire tourner le cube autour de son axe Y.

19. Complétez le code de la fonction `animate` :

```
function animate() {
  cube1.rotation.y += 0.01;
  renderer.render(scene, camera);
}
```

20. Modifiez le code pour doubler la vitesse de rotation.

Il faut maintenant créer un effet de rebond (sur l'axe Y). La fonction sinus en math est précieuse pour cela.

21. Complétez le code de la fonction `animate` comme suit :

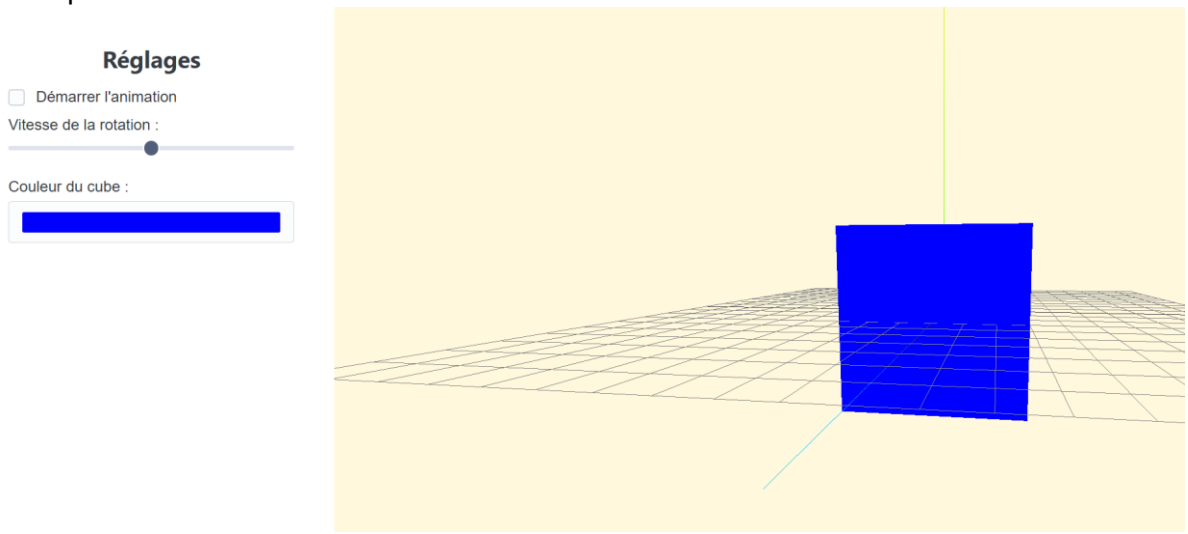
```
function animate(time) {
  cube1.position.y = Math.abs(50 * Math.sin(time * 0.01));
  //cube1.rotation.y += 0.01;
  renderer.render(scene, camera);
}
```

Les paramètres du sinus ne sont pas bien réglés. A vous de jouer ...

22. Les 3 cubes doivent être maintenant rebondissants mais pas avec le même rythme.

INTERACTION avec le reste de la page

Vous devez maintenant pouvoir faire interagir votre animation 3D avec des éléments de contrôles classiques que l'on trouve dans les formulaires (coche, liste, range, ...).
L'UX à produire est la suivante :



Récupérez, depuis le site, le fichier `interaction.html + interaction.css` qui contient le HTML et les petits réglages CSS pour produire l'UX. La librairie pico.css est utilisée.

23. Adaptez tout le code JS pour piloter l'animation en fonction de l'état de la coche, du réglage de la plage de vitesse et du choix de couleur. Pour mémoriser ces informations de réglages, vous ajouterez une variable d'état :

```
let state = {
  is_running: false,
  speed_on_Y_axe: 5,
  color: null,
};
```

Les 3 éléments de l'interface se verront affecter un écouteur d'évènement.
Quand un réglage change, la variable `state` doit être actualisée ainsi que le rendu 3D.

Vous remarquez rapidement que si la fenêtre est redimensionnée, la zone d'affichage du canvas ne s'adapte pas ... et c'est très disgracieux.
Il faut corriger cela en ajoutant un écouteur d'évènement sur le "resize" de l'écran.
Dans la fonction de rappel, il faut mettons à jour la propriété aspect de la caméra pour qu'elle corresponde au nouveau rapport hauteur/largeur du canvas redimensionné.
De plus, il est essentiel d'appeler la méthode updateProjectionMatrix() à chaque fois que nous modifions une propriété de caméra . Dans ce cas, nous l'appelons car nous ajustons le rapport hauteur/largeur lors du redimensionnement du canvas.

24. Ajoutez dans `main()` le code suivant et testez :

```
window.addEventListener("resize", function () {
  camera.aspect = container.offsetWidth / container.offsetHeight;
  camera.updateProjectionMatrix();
  renderer.setSize(container.offsetWidth, container.offsetHeight);
});
```

Vous avez précédemment mis en place une petite interface pour agir sur des paramètres de l'animation. Mais Three.js permet de faire cela assez nativement.
Une interface open source a été conçue spécifiquement à cet effet. Ce module d'interface se nomme **lil-gui** (documentation : https://lil-gui.georgealways.com).

25. Installez-le avec la commande **npm install lil-gui.**

26. Intégrez le module : `import { GUI } from 'lil-gui';`

Pour piloter l'interface à partir de ce panneau de contrôle, il faut mettre en place un objet qui contient le réglage.

27. Ajoutez cette nouvelle constante en global (avec les variables) :

`const options = {`
`  color: 0x0000ff,`
`  wireframe: false,`
`};`
	`color` spécifie la couleur du cube
	`wireframe` indique si le mode fil de fer doit être actif

La méthode `addColor()` va ajouter la palette de couleurs. Le premier paramètre indique l'objet avec lequel il faut travailler et le second paramètre la clé qui doit être modifiée. Puis, une fonction de rappel est déclenchée à chaque changement. La variable `e` de la callback contient ici la nouvelle valeur de couleur.

28. Dans `main()` complétez le code qui suit et testez :

```
const gui = new GUI();
gui.addColor(options, "color").onChange(function (e) {
  cube1.material.color.set(e);
});
```

Avec cette même logique, il est possible d'ajouter la coche pour activer/inactiver le mode wireframe :

```
gui.add(options, "wireframe").onChange(function (e) {
  cube1.material.setValues({ wireframe: e });
});
```

29. Complétez le traitement pour prendre en compte le réglage de la vitesse de rotation du cube.

La fonction de callback `onChange` n'est pas nécessaire. La valeur min de vitesse est 0 et la valeur max est 0.1 (0.01 par défaut).

Pour terminer ce premier TP, vous pouvez choisir d'appliquer sur votre canvas une image comme texture (et non plus une simple couleur).
Choisissez une image que vous allez placer dans le dossier /public.
L'image doit être transformée en texture.
Il faut appeler une instance de la classe `TextureLoader()`, charger cette image est l'appliquer à la propriété `background` de l'objet `scene`.

```
const textureLoader = new THREE.TextureLoader();
const backgroundImage = textureLoader.load("/background.jpg");
scene.background = backgroundImage;
```

ANNEXE 1 : principales formes possibles

Three.js propose quelques formes de base pour produire un carré, un cylindre, un cône, un plan et une sphère

- **BoxGeometry** : le carré

```
const geometry = new THREE.BoxGeometry( 150, 150, 150 );
const material = new THREE.MeshPhongMaterial( {color: 0x00ffff} );
cube = new THREE.Mesh( geometry, material );
scene.add( cube );
```

Le constructeur de la classe BoxGeometry accepte 3 paramètres :

X	dimension sur les X
Y	dimension sur les Y
Z	dimension sur les Z

- **ConeGeometry** : le cône

```
const geometry = new THREE.ConeGeometry( 150, 150, 15 );
const material = new THREE.MeshPhongMaterial( {color: 0x00ffff} );
cone = new THREE.Mesh( geometry, material );
scene.add( cone );
```

Le constructeur de la classe ConeGeometry accepte 3 paramètres :

Rayon	taille de rayon du cône
Hauteur	hauteur du cône
Complexité de la structure	nombre de segments du rayon

- **CylinderGeometry** : le cylindre

```
const geometry = new THREE.CylinderGeometry( 150, 150, 300, 8, 4 );
const material = new THREE.MeshPhongMaterial( {color: 0x00ffff} );
mesh = new THREE.Mesh( geometry, material );
scene.add( mesh );
```

Le constructeur de la classe CylinderGeometry accepte 5 paramètres :

Rayon supérieur	taille de rayon de la face supérieure de la structure
Rayon inférieur	taille de rayon de la face inférieure de la structure
Hauteur	hauteur du cylindre
Segments rayon	nombre de segments rayons de la structure
Segments hauteur	nombre de segments sur la hauteur de la structure

- **PlaneGeometry** : le plan

```
const geometry = new THREE.PlaneGeometry( 250, 150, 2, 2 );
const material = new THREE.MeshPhongMaterial( {color: 0x00ffff, side : THREE.DoubleSide} );
mesh = new THREE.Mesh( geometry, material );
scene.add( mesh );
```

Remarque : la propriété side permet d'appliquer le matériau sur les deux faces.

Le constructeur de la classe PlaneGeometry accepte 4 paramètres :

Largeur	largeur de la structure sur l'axe X
Hauteur	hauteur de la structure sur l'axe Y
Nombre de segments sur la largeur	
Nombre de segments sur la hauteur	

- **SphereGeometry** : le cercle

```
const geometry = new THREE.SphereGeometry( 250, 32, 32 );
const material = new THREE.MeshPhongMaterial( {color: 0x00ffff} );
mesh = new THREE.Mesh( geometry, material );
scene.add( mesh );
```

Le constructeur de la classe SphereGeometry accepte 3 paramètres :

Rayon	taille de rayon de la sphère
Complexité Largeur	nombre de segments sur la largeur
Complexité Hauteur	nombre de segments sur la hauteur