LYCÉE ARISTIDE BERGÈS
SIO

Exploration

251 - Fonctionnement Three.JS

Rédigé par

David ROUMANET
Professeur BTS SIO

Changement

Date	Révision
2022-02-20	Erreur syntaxe p.4 (/* ai lieu de //) et rajout avertissement localhost.

Sommaire

Nomenclature à supprimer :

- **Assimilation** : cours pur. Explication théorique et détaillée (globalement supérieur à 4 pages).
- **Décoder** : fiche de cours, généralement inférieure à 5 pages.
- **Découverte** : Travaux dirigés. Faisable sans matériel.
- **Explorer** : Travaux pratiques. Nécessite du matériel ou des logiciels.
- **Mission** : Projet encadré ou partie d'un projet.
- **Projet** : Projet en autonomie totale. Environnement ouvert.

A Introduction

La représentation de données ou d'information se fait très souvent de manière graphique. Nous avons déjà découvert le framework chart.JS qui permet de représenter des graphiques en 2D, comme dans les feuilles de calculs (Excel, Calc, etc.).

Cette exploration va ouvrir les portes du jeu et de la représentation en 3D d'objets : en effet, même si JavaScript est un langage interprété, il est possible d'utiliser les capacités graphiques de la machine en utilisant le standard WebGL[1].

A.1 Présentation

Pour découvrir comment créer une application utilisant de la 3D, nous allons découvrir Three.JS.

Three.JS est un framework capable de générer et afficher une scène en 3D dans un canevas HTML.

Il facilite la vie du développeur, en prenant en charge les calculs, les traitements et le rendu des éléments créés et disposés par le développeur. De plus, il peut utiliser un moteur de rendu (comme WebGL) pour afficher, mais aussi animer les objets et la scène : c'est un équivalent des moteurs de jeu RealEngine ou Unity3D, mais pour JavaScript.

A.2 Prérequis

Cette exploration nécessite de connaître :

- La programmation en JavaScript
- Quelques notions de 3D (notion d'espace, coordonnées, etc.)

Pour fonctionner correctement, les différents projets et codes doivent être placés dans un serveur web (http://localhost) : soit le répertoire `www` d'Apache si vous activer WAMP, soit vous utilisez la commande `python3 -m http.server 80` dans le répertoire de votre projet.

1 https://developer.mozilla.org/fr/docs/Web/API/WebGL_API

B Implémenter Three.JS

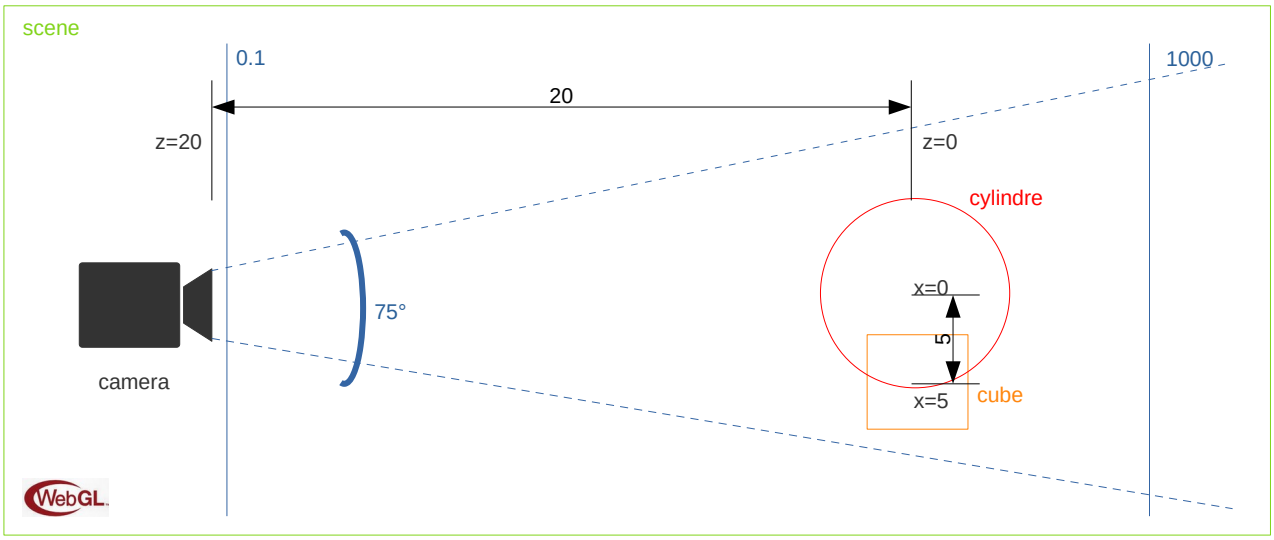
L'implémentation de Three.JS est très simple, il suffit de télécharger le fichier dans le répertoire `build` du site Github dédié :

https://github.com/mrdoob/three.js/tree/dev/build

B.1 Premier projet

Le plus simple des projets consiste en une scène contenant deux objets et une caméra. Le moteur de rendu sera WebGL.

B.1.1 Préparation de la scène



La caméra aura un champ de vue (FOV pour Field Of View) de 75°, une distance minimum pour représenter un objet de 0.1 (sans unité) et une distance maximum de 1000 (toujours sans unité).

Le premier objet sera un cylindre (donc, la base haute sera de 5 et la base basse aussi) avec une hauteur de 10 et 32 arcs (ou segments).

Le deuxième objet sera un cube de 4 (largeur, hauteur et profondeur identiques).

Le tout, sera dessiné après le dernier objet de la page HTML. Les fichiers seront :

- index.html
- exemple1.js
- style.css

B.1.2 Les fichiers de code

Voici les trois fichiers :

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Exemple Three.JS</title>
    <style>
        @import url("style.css");
    </style>
</head>
<body>
    <h1>Premier exemple</h1>
</body>
<script src="js/three.js"></script>
<script src="exemple1.js"></script>
</html>
```

style.css

```
canvas {
    cursor:grab;
}
canvas:active {
    cursor:grabbing;
}

body {
    background-color: black;
    color: white;
    margin: 0;
    overflow: hidden;
}
```

Le dernier fichier est celui qui intègre le code pour lancer l'affichage.

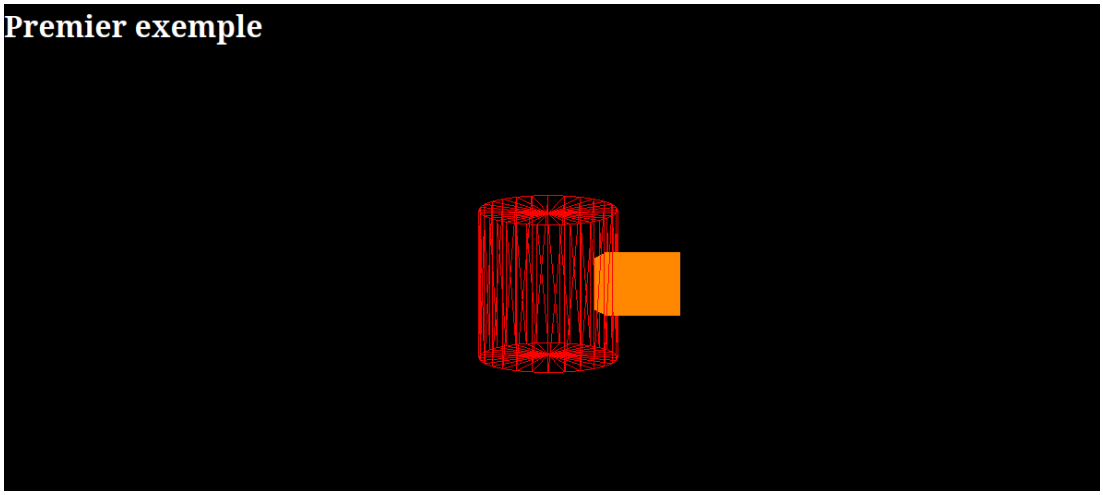
Pour chaque objet on doit définir une forme et un matériel (texture, couleur).

exemple1.js

```
// Exemple 1 : forme dans une scène avec une caméra

const scene = new THREE.Scene()
const renderer = new THREE.WebGLRenderer()

// Caméra en perspective (position) : FOV=75°, aspect ratio, distance min, distance max
const camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 1000)

// Création d'un cylindre (base haute 4, base basse 4, hauteur 10, 32 segments, rouge)
const formeGeometrique1 = new THREE.CylinderGeometry(4, 4, 10, 32)
const materiel1 = new THREE.MeshBasicMaterial({ color: 0xff0000, wireframe: true })
const cylindre = new THREE.Mesh(formeGeometrique1, materiel1)

// Création d'un cube (largeur 4, hauteur 4, profondeur 4 en orange plein)
const formeGeometrique2 = new THREE.BoxGeometry( 4, 4, 4 )
const materiel2 = new THREE.MeshBasicMaterial({ color: 0xff8800 })
const cube = new THREE.Mesh(formeGeometrique2, materiel2)

scene.add(cylindre)
scene.add(cube)

cube.position.x = 5         // déplacement du cube de 5 en largeur
camera.position.z = 20      // déplacement de la caméra de 20 en profondeur

// Création d'un rendu sur la presque totalité de la fenêtre du navigateur
renderer.setSize(window.innerWidth-10, window.innerHeight-100)
document.body.appendChild(renderer.domElement)
renderer.render(scene, camera)
```

B.1.3 Résultat d'affichage

Le résultat est (normalement) le suivant :



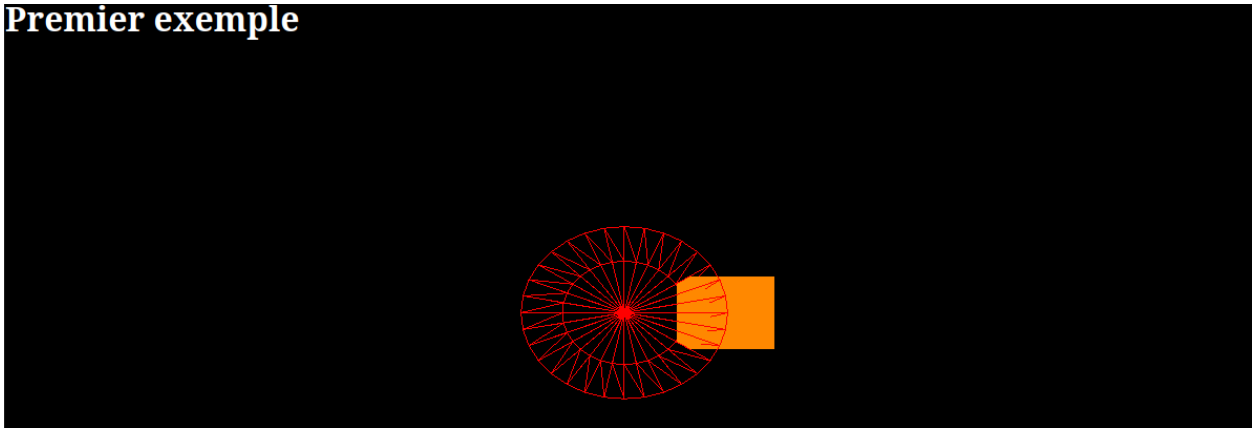
Scene, CylinderGeometry et WebGLRenderer() sont-elles des classes ? Le module `THREE` apporte de nombreuses méthodes pour utiliser des objets 3D. Le mot clé `new` permet de répondre à cette question.

B.1.4 Variation

Vous pouvez ajouter une rotation sur le cylindre après les changements de positions pour la caméra et le cube :

```
cube.position.x = 5          // déplacement du cube de 5 en largeur
camera.position.z = 20       // déplacement de la caméra de 20 en profondeur
cylindre.rotation.x = 1.57   // rotation sur l'axe Y de 45
```

La rotation s'exprime en radians : 2π = 360°, soit un tour complet. Ici, π/2 = un quart de tour. Le résultat est le suivant :



B.1.5 Animation

Remplacer la ligne `renderer.render(scene, camera)` par :

```
renderer.render(scene, camera)

// ajout de l'animation
function animate() {
    cylindre.rotation.x += 0.002
    cylindre.rotation.y += 0.01
    cube.rotation.z += 0.02
    renderer.render(scene, camera)
    requestAnimationFrame(animate)
}

animate()
```

Désormais, votre cylindre tourne sur lui-même (l'axe X est le plus lent) et le cube tourne autour de l'axe Z.

L'utilisation des `.` dans les accès aux membres, permet de constater que l'objet instancié appelé 'cylindre' a des propriétés accessibles directement. Par exemple, `cylindre.rotation.x` est une propriété qui contient l'angle de l'objet sur l'axe X.

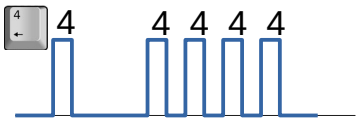
C Tester les interactions

Pour interagir avec Three.JS, il faut gérer les événements dans la page web.

La plupart des événements en provenance du clavier porte sur l'objet du DOM "document". Il est possible d'interagir de plusieurs façons, mais les deux méthodes traditionnelles sur un ordinateur, sont le clavier et la souris.

C.1 Le clavier

Il y a une particularité dans le fonctionnement du clavier : si on laisse une touche enfoncée, il existe un mécanisme qui permet à la touche de répéter le caractère ou symbole enfoncé, tant qu'on ne relâche pas la touche. Malheureusement, il existe un délai qui entraîne une latence au premier appui de la touche (en laissant enfoncé).



Pour contourner ce délai, il faut gérer le clavier différemment.

La solution la plus utilisée, est de gérer les deux événements `onKeyDown` et `onKeyUp`. Lorsque la touche s'enfonce, on change l'état d'une variable qui gère la position ou la rotation d'un objet. Lorsque la touche se relâche, la détection de l'événement doit rendre la variable inactive. Ce genre de variable est un peu comme un drapeau que l'on change d'état, et c'est pourquoi, il est possible d'entendre parler de variable-drapeau.

Dans le code du précédent exemple, il faut donc ajouter deux variables : rotationXCylindre et rotationZCube qui sont initialement à zéro et dont l'état change lorsqu'on enfonce une touche. Ajoutez ce code, sous la fonction `animate()` (dessous et pas dedans) :

```
let rotationXCylindre = 0, rotationZCube = 0
function onKeyDown(event) {
    switch (event.code) {
        case 'ArrowUp':
        case 'KeyW':
            rotationXCylindre = +0.008;
            console.log("UP")
            break;
        case 'ArrowLeft':
        case 'KeyA':
            rotationZCube = +0.02;
            console.log("LEFT")
            break;
        case 'ArrowDown':
        case 'KeyS':
            rotationXCylindre = -0.008;
            console.log("DOWN")
            break;
        case 'ArrowRight':
        case 'KeyD':
            rotationZCube = -0.02;
            console.log("RIGHT")
            break;
    }
}
```

Il faut évidemment créer la fonction inverse, qui remet à zéro les variables lorsque la touche est relâchée :

```
function onKeyUp(event) {
    switch (event.code) {
        case 'ArrowUp':
        case 'KeyW':
            rotationXCylindre = 0;
            console.log("UP END")
            break;
        case 'ArrowLeft':
        case 'KeyA':
            rotationZCube = 0;
            console.log("LEFT END")
            break;
        case 'ArrowDown':
        case 'KeyS':
            rotationXCylindre = 0;
            console.log("DOWN END")
            break;
        case 'ArrowRight':
        case 'KeyD':
            rotationZCube = 0;
            console.log("RIGHT END")
            break;
    }

}

document.addEventListener('keydown', onKeyDown);
document.addEventListener('keyup', onKeyUp
```

Les deux dernières lignes permettent à JavaScript de savoir quelles fonctions appeler, lorsqu'un événement survient.

Il faut enfin prendre en compte les changements de variables dans la fonction `animate()` :

```
function animate() {
    if (rotationXCylindre) {
        cylindre.rotation.x += rotationXCylindre
    }
    cylindre.rotation.y += 0.01
    if (rotationZCube) {
        cube.rotation.z += rotationZCube
        cube.position.z += rotationZCube*2  // Bonus, le changement de position pendant la rotation
    }
    renderer.render(scene, camera)
    requestAnimationFrame(animate)
}
```

Sauvegardez et testez le code.

Les flèches de gauche et de droite modifient l'angle et la position du cube. Les flèches verticales changent l'angle du cylindre, sur l'axe X.

Note : Il serait possible de remplacer le code de la fonction `onKeyUp()` par des conditions classiques `if`... `else` mais les instructions `switch()` et `case` sont plus lisibles. Le moteur d'exécution remplace ces instructions automatiquement par les conditions adéquates. On appelle cela du "*sucre syntaxique*".

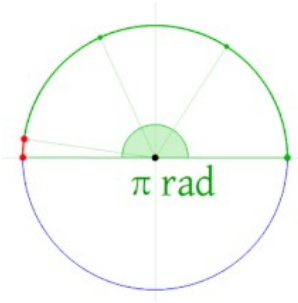
C.2 Gestion de la caméra

Afin de pouvoir créer un changement de vue, il faut pouvoir modifier les positions ou les rotations de la caméra active. La position se définit comme pour les autres objets. La rotation s'exprime en radian :

Rappel :

Bien que ce ne soit un cours de mathématiques, il est bon de comprendre cette unité internationale : le radian est la longueur du rayon d'un cercle, reporté sur ce cercle[2]. Il faut alors un peu plus de 3 radians pour avoir la longueur d'un demi-cercle. Il faut exactement π radian !

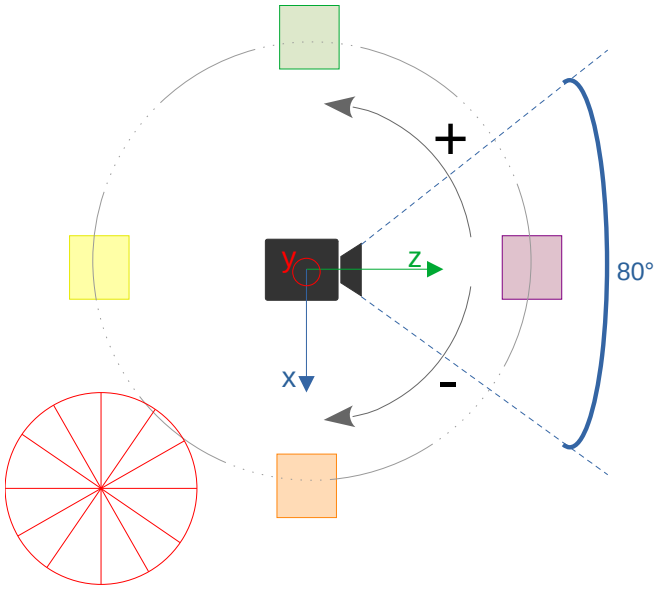
Il est donc logique d'avoir 2π pour faire un tour complet et donc π/2 pour faire un quart de tour.



Pour vérifier le fonctionnement de notre caméra et de sa rotation, nous devons créer une scène contenant plusieurs objets. Nous nous contenterons de rajouter 3 cubes et de replacer correctement le cylindre.

C.2.1 Préparation de la scène

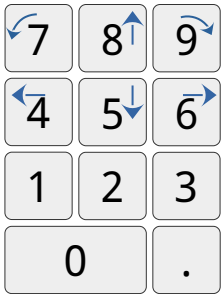
La modélisation en 3D implique de bien définir son environnement. Le schéma suivant permet de bien préparer la scène, et ainsi, ne pas se tromper : les tâtonnements peuvent devenir longs et finir par avoir raison de notre patience.



Nous agrandissons la focale de la caméra (80°), pour pouvoir voir plusieurs cubes lors des rotations (le pas sera en quart de radians, soit 45°). Une rotation vers la gauche ajoute π/4 tandis qu'une rotation à droite soustrait π/4.

2 L'animation de Wikipedia est très explicite, n'hésitez pas à consulter https://fr.wikipedia.org/wiki/Radian

Concernant le clavier, nous allons déplacer les touches vers le pavé numérique, afin d'avoir les quatre touches de direction (avancer, reculer, translation gauche et translation droite) et les rotations gauche et droite.



Il est possible de choisir d'autres touches si vous n'avez pas de pavé numérique sur votre clavier (ex: certains PC portables).

Les touches portent le nom "Numpad0", "Numpad1", etc.

C.2.2 Le fichier de code

Le nouveau code est le suivant :

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Gestion caméra Three.JS</title>
    <style>
        @import url("style.css");
    </style>
</head>
<body>
    <h1>Premier exemple</h1>
</body>

<script src="js/three.js"></script>
<script src="exemple4.js"></script>
</html>
```

La partie JavaScript change comme ceci :

exemple4.js

```
/* Exemple 4 : gestion de la rotation de caméra
    Le minimum nécessaire :
    - une scène,
    - un moteur de rendu,
    - une caméra,
    - plusieurs formes (4 cubes et un cylindre)
*/

const scene = new THREE.Scene()
const renderer = new THREE.WebGLRenderer()

// Caméra en perspective (position) : FOV=80°, aspect ratio, distance min, distance max
const camera = new THREE.PerspectiveCamera(80, window.innerWidth / window.innerHeight, 0.1, 1000)

// Création d'un cylindre (base haute 4, base basse 4, hauteur 10, 32 segments, rouge)
const formeGeometrique1 = new THREE.CylinderGeometry(4, 4, 10, 32)
const materiel1 = new THREE.MeshBasicMaterial({ color: 0xff0000, wireframe: true })
const cylindre = new THREE.Mesh(formeGeometrique1, materiel1)

// Création de 4 cubes (orange : geométrie 4x4x4, jaune, vert, violet : géométrie 2x2x2)
const formeGeometrique2 = new THREE.BoxGeometry(4, 4, 4)
const materiel2 = new THREE.MeshBasicMaterial({ color: 0xff8800 })
const materiel3 = new THREE.MeshBasicMaterial({ color: 0x008800 })
const materiel4 = new THREE.MeshBasicMaterial({ color: 0xffff00 })
const materiel5 = new THREE.MeshBasicMaterial({ color: 0xff88FF })
const formeGeometrique3 = new THREE.BoxGeometry(2, 2, 2)
const cube = new THREE.Mesh(formeGeometrique2, materiel2)
```

```
const cube2 = new THREE.Mesh(formeGeometrique3, materiel3)
const cube3 = new THREE.Mesh(formeGeometrique3, materiel4)
const cube4 = new THREE.Mesh(formeGeometrique3, materiel5)
const PI = 3.14159

// Création de la scène
scene.add(cylindre)
scene.add(cube)
scene.add(cube2)
scene.add(cube3)
scene.add(cube4)

// positionnement des objets
cube.position.x = 5         // déplacement du cube de 5 en largeur
cube.position.z = 0
camera.position.z = 0
cylindre.position.x = 5
cylindre.position.z = 5
cylindre.rotation.x = 0
cube2.position.x = -5
cube2.position.z = 0
cube3.position.x = 0
cube3.position.z = 5
cube4.position.x = 0
cube4.position.z = -5
let rotationCubes = 0.01      // pour l'animation constante des cubes
let rotationXCylindre = 0, rotationZCube = 0

// Création d'un rendu sur la presque totalité de la fenêtre du navigateur
renderer.setSize(window.innerWidth - 10, window.innerHeight - 100)
document.body.appendChild(renderer.domElement)
// renderer.render(scene, camera)

// ajout de l'animation
function animate() {
    cube2.rotation.x += rotationCubes
    cube2.rotation.z += rotationCubes
    cube3.rotation.x += rotationCubes
    cube3.rotation.z += rotationCubes
    cube4.rotation.x += rotationCubes
    cube4.rotation.z += rotationCubes

    if (rotationXCylindre) {
        cylindre.rotation.x += rotationXCylindre
    }
    cylindre.rotation.y += 0.01
    if (rotationZCube) {
        cube.rotation.z += rotationZCube
        cube.position.z += rotationZCube*2
    }
    renderer.render(scene, camera)
    requestAnimationFrame(animate)
}

function onKeyDown(event) {
    switch (event.code) {
        case 'ArrowUp':
        case 'Numpad8':
        case 'KeyW':
            rotationXCylindre = +0.008;
            console.log("UP")
            break;
        case 'ArrowLeft':
        case 'Numpad4':
        case 'KeyA':
            rotationZCube = +0.02;
            console.log("LEFT")
            break;
        case 'ArrowDown':
        case 'Numpad5':
```

```
        case 'KeyS':
            rotationXCylindre = -0.008;
            console.log("DOWN")
            break;
        case 'ArrowRight':
        case 'NumPad6':
        case 'KeyD':
            rotationZCube = -0.02;
            console.log("RIGHT")
            break;
        case 'Numpad7':
            console.log("ROT LEFT")
            camera.rotation.y = camera.rotation.y +PI/4
            break;
        case 'Numpad9':
            console.log("ROT RIGHT")
            camera.rotation.y = camera.rotation.y -PI/4
            break;
    }
}

function onKeyUp(event) {
    console.log(event.code)
    switch (event.code) {
        case 'ArrowUp':
        case 'Numpad8':
        case 'KeyW':
            rotationXCylindre = 0;
            console.log("UP END")
            break;
        case 'ArrowLeft':
        case 'Numpad4':
        case 'KeyA':
            rotationZCube = 0;
            console.log("LEFT END")
            break;
        case 'ArrowDown':
        case 'Numpad5':
        case 'KeyS':
            rotationXCylindre = 0;
            console.log("DOWN END")
            break;
        case 'ArrowRight':
        case 'Numpad6':
        case 'KeyD':
            rotationZCube = 0;
            console.log("RIGHT END")
            break;
    }

}

function onWindowResize ()
{
    camera.aspect = window.innerWidth / window.innerHeight;
    camera.updateProjectionMatrix();
    renderer.setSize (window.innerWidth, window.innerHeight);
}

document.addEventListener('keydown', onKeyDown);
document.addEventListener('keyup', onKeyUp);
window.addEventListener('resize', onWindowResize, false);

console.log("GO")
animate()
```

Votre animation doit fonctionner comme avant, mais vous pouvez tourner la caméra.

C.3 La lumière

L'ajout d'une source de lumière permet de donner plus de réalisme à la scène : ce peut être une source large, mais aussi un spot directionnel. On peut ajouter autant de sources que l'on souhaite.

Nous allons tester une lampe d'ambiance (AmbientLight) puis ajouter une lampe directionnelle (DirectionalLight). Cependant, nous allons devoir modifier la texture des formes : seul le premier cube conservera sa texture orange, afin de comparer les propriétés des textures.

Note : une lumière d'ambiance inonde toute la scène, il n'y a donc pas de projection d'ombre, de profondeur de lumière, etc.
La lumière d'ambiance n'a que deux paramètres : couleur et intensité.

C.3.1 Lumière d'ambiance

Modification du code précédent, remplacez :

```
const materiel3 = new THREE.MeshBasicMaterial({ color: 0x008800 })
const materiel4 = new THREE.MeshBasicMaterial({ color: 0xffff00 })
const materiel5 = new THREE.MeshBasicMaterial({ color: 0xff88FF })
```

Par :

```
const materiel3 = new THREE.MeshPhongMaterial({color: 0x008800})
const materiel4 = new THREE.MeshPhongMaterial({color: 0xffff00})
const materiel5 = new THREE.MeshPhongMaterial({color: 0xff88ff})
```

Puis ajoutez l'éclairage d'ambiance (juste après la déclaration de la constante Pi, par exemple) :

```
// Lampe d'ambiance
const color = 0xccffcc
const intensity = 0.6
const light = new THREE.AmbientLight(color, intensity)
scene.add(light)
```

Testez votre nouveau code.

Note : il est possible de transmettre les paramètres sous forme d'objets JSON ou bien directement aux classes du module THREE. L'intérêt de JSON, est de décrire chaque paramètre et sa valeur. C'est plus lisible pour le programmeur qui veut modifier le code.

C.3.2 Lumière directionnelle

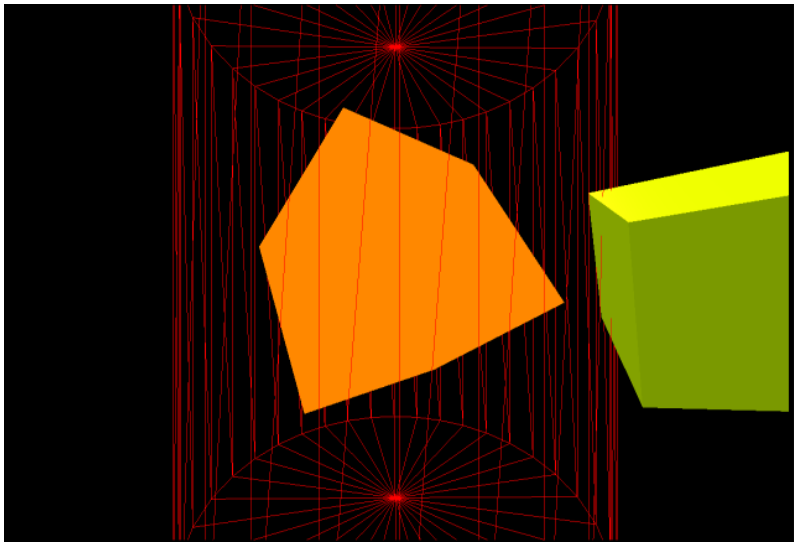
Contrairement à la lumière d'ambiance, la lumière directionnelle agit comme un spot que l'on peut diriger.

Ajoutez le code suivant (après l'éclairage d'ambiance) :

```
// Lampe directionnelle
const dirLightColor = 0xFFFFFF;
const dirLightIntensity = 1;
const dirLight = new THREE.DirectionalLight(dirLightColor, dirLightIntensity);
dirLight.position.set(0, 10, 8);
dirLight.target.position.set(-5, 0, 0)
scene.add(dirLight)
scene.add(dirLight.target)
```

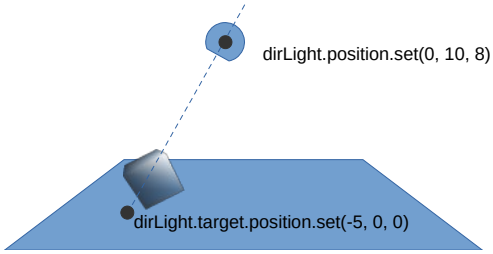
Testez le code.

On remarque que le cube orange et le cylindre ne sont pas modifiés par les lumières, tandis que les autres cubes ont des reflets différents selon l'exposition de leurs faces.



Les lampes directionnelles nécessitent deux points pour fonctionner :

- La position de la source (x, y, z)
- La position de la zone éclairée (x, y, z)



C.3.3 Point de lumière

Ce point à la particularité de prendre en compte la distance (certainement le plus utile pour un jeu de labyrinthe).

Remplacez toute la section de la lampe directionnelle par ceci :

```
// Lampe type point de lumière (ou torche en bois)
let flamme = 7
const pointLight = new THREE.PointLight(0xFFFF55, 0.8)
pointLight.position.set(0, 2, 0)        // au-dessus de la caméra
pointLight.distance = flamme
scene.add(pointLight)
```

et nous allons simuler un effet de torche en bois qui brûle en modifiant aléatoirement la distance d'éclairage, dans la fonction `animate()` comme suit :

```
pointLight.distance = flamme+(Math.random()*2-1)
```

Sauvegardez et testez votre code : vous devriez voir les cubes avoir une sorte de clignotement.

C.4 Plan et textures

Pour cette avant-dernière partie d'exploration, nous allons ajouter un sol et un plafond à notre code. Pour cela, il existe un objet pratique qui s'appelle `PlaneGeometry` : dans un premier temps, nous allons l'utiliser sans texture. Puis nous appliquerons une texture de petite taille, en la répétant autant de fois que nécessaire.

C.4.1 Plan simple

Nous ajoutons le plan à 3 unités sous la caméra (y-3).

Copiez le code sous les lumières directionnelles :

```
// Sol gris
const planeSize = 40
const planeGeo = new THREE.PlaneGeometry(planeSize, planeSize)
const planeMat = new THREE.MeshPhongMaterial({color: 0x777777, side: THREE.DoubleSide})
const sol = new THREE.Mesh(planeGeo, planeMat)
sol.position.y = -3
sol.rotation.x = Math.PI/2      // bascule le plan vertical (mur) à l'horizontal
scene.add(sol)
```

Sauvegardez et testez le code. Le plan paraît vert à cause de l'éclairage d'ambiance.

Pour créer un plafond, il suffit d'ajouter :

```
const plafond = new THREE.Mesh(planeGeo, planeMat)
plafond.position.y = +3
plafond.rotation.x = -Math.PI/2 // bascule le plan de vertical à horizontal
```

C.4.2 Texture

Cette partie est un peu plus complexe : elle consiste à appliquer une texture (en général, une image répétitive) sur la surface voulue. Pour éviter d'avoir une image énorme, les textures sont souvent limitées à quelques centaines de pixels, il faut donc la répéter sur toute la surface.

Dans un premier temps, nous allons charger la texture depuis ce lien (enregistrer sous…) :

https://opengameart.org/sites/default/files/oga-textures/15876/ground_asphalt_synth_07.png

Modifiez le nom du fichier par ground.png.

Puis nous allons utiliser l'exemple de Three.js pour les textures (à placer avant la définition de `planeMat`) :

```
const loader = new THREE.TextureLoader()
const texture = loader.load('assets/ground.png');
texture.wrapS = THREE.RepeatWrapping;
texture.wrapT = THREE.RepeatWrapping;
texture.magFilter = THREE.NearestFilter;
const repeats = planeSize / 2;
texture.repeat.set(repeats, repeats);
```

Nous utilisons le filtre `nearestFilter` qui affiche de manière plus précise que `LinearFilter`, la texture :

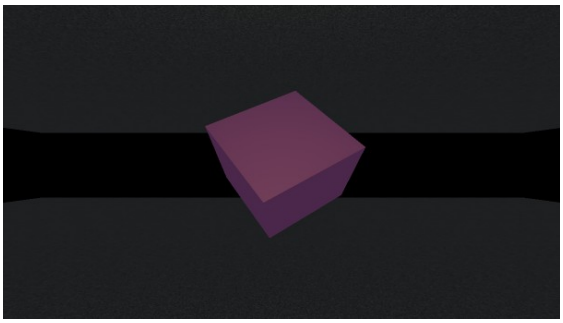
Enfin, il faut remplacer l'instruction

```
const planeMat = new THREE.MeshPhongMaterial({color: 0x777777, side: THREE.DoubleSide})
```

par

```
const planeMat = new THREE.MeshPhongMaterial({ map: texture, side: THREE.DoubleSide})
```

Enregistrez et testez.



C.5 La souris (et la caméra)

Dans l'idéal, il serait intéressant de pouvoir utiliser la bibliothèque OrbitControls qui permet de gérer les mouvements de caméra de manière fluide, grâce au déplacement de la caméra autour d'un objet (d'où le nom OrbitControls).

L'usage de modules dans une page HTML imposent une syntaxe assez particulière, pour éviter les erreurs de type CORS (Cross Origin Resource Sharing). Il s'agit d'une sécurisation normale.

Créez un nouveau projet, car les modifications sont importantes. Dans un nouveau répertoire (ThreejsExemple6) créez le fichier index.html suivant :

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>OrbitControls et Objets 3D</title>
    <style>
        @import url("style.css");
    </style>
</head>
<body>
    <h1>OrbitControls et Objets 3D</h1>
</body>

<!-- Utiliser es-module-shims pour importer les frameworks comme avec npm-->
<script async src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.35.5/es6-shim.min.js"></script>
<script type="importmap">
    {
        "imports": {
            "three": "https://unpkg.com/three@0.149.0/build/three.module.js"
        }
    }
</script>

<script type="module">
    import * as THREE from 'three'
    import { OrbitControls } from
'https://unpkg.com/three@0.149.0/examples/jsm/controls/OrbitControls.js'
    import { GLTFLoader } from 'https://unpkg.com/three@0.149.0/examples/jsm/loaders/GLTFLoader.js'
</script>
<script type="module" src="exemple6.js"></script>
</html>
```

Créez ensuite un fichier exemple6.js contenant le code suivant (pour tester dans le navigateur. Attention à bien utiliser un lien de type http://localhost)

exemple6.js

```
import * as THREE from 'three'
import { OrbitControls } from 'https://unpkg.com/three@0.149.0/examples/jsm/controls/OrbitControls.js'
import { GLTFLoader } from 'https://unpkg.com/three@0.149.0/examples/jsm/loaders/GLTFLoader.js'

const renderer = new THREE.WebGLRenderer()
renderer.setSize(window.innerWidth-20, window.innerHeight-20)
document.body.appendChild(renderer.domElement)

const scene = new THREE.Scene()
```

Si tout va bien, en ouvrant la console de débogage, il ne doit pas y avoir d'erreurs (à part le favicon non trouvé).

Voici maintenant le code complet :

exemple6.js

```
import * as THREE from 'three'
import { OrbitControls } from 'https://unpkg.com/three@0.149.0/examples/jsm/controls/OrbitControls.js'
import { GLTFLoader } from 'https://unpkg.com/three@0.149.0/examples/jsm/loaders/GLTFLoader.js'

const renderer = new THREE.WebGLRenderer()
renderer.setSize(window.innerWidth-20, window.innerHeight-20)
document.body.appendChild(renderer.domElement)

const scene = new THREE.Scene()

const camera = new THREE.PerspectiveCamera(70, window.innerWidth/window.innerHeight, 0.1, 1000)
camera.position.set(3,3,2.5)

const ambientLight = new THREE.AmbientLight({color: 0xFFFFFF, intensity: 0.9})
const directionalLight = new THREE.DirectionalLight(0xFFFFFF, 6)
directionalLight.position.set(3,3,2.5)
directionalLight.target.position.set(0,0,0)
const controls = new OrbitControls(camera, renderer.domElement)
const loader = new GLTFLoader()

controls.addEventListener('change', () => { renderer.render(scene, camera)})
controls.target.set(0,0,0)
controls.update()

loader.load('./assets/ship.gltf', (gltf) => {
    const model = gltf.scene
    model.position.set(0,0,-1)     // décalage sur z pour tourner autour du centre du vaisseau
    scene.add(model)
}, undefined, (error) => {console.error(error)})

scene.add(ambientLight)
scene.add(directionalLight)
```

Avant de l'exécuter, vous devez télécharger les ressources dans le répertoire `assets` de votre projet.

Lorsque vous n'avez pas d'erreur, vous devriez voir un petit vaisseau spatial bouger lorsque vous cliquez et déplacez la souris.

Avez-vous remarqué ? L'utilisation de la méthode `position.set(x, y, z)` est plus pratique que de modifier successivement les propriétés `position.x`, `position.y` et `position.z` ! il s'agit d'un actuateur ou en anglais, setter.

C.6 Les objets

https://sketchfab.com/

À EFFACER LORSQUE LE DOCUMENT EST TERMINÉ

Un encart de base

Attention, il est important de vérifier blabla que blabla. Cela peut arriver lorsque blabla ou bien que truc active le machin et dépasse la limite du bidule. Il est donc important de ne pas laisser les choses dériver sans un contrôle des valeurs sinon blabla.

Il reste cependant la possibilité d'écrire en dessous du nombre de lignes.

Un encart d'attention + puce attention

Attention : ce type d'encart informe d'une difficulté ou d'un danger potentiel. Le caractère de priorité signifie que l'utilisateur doit prendre en compte le commentaire pour réussir avec succès.

Un encart stop + puce stop

L'encart STOP indique une information capitale, à caractère obligatoire. Ne pas suivre les informations dans cet encart entraîne un mauvais fonctionnement plus tard ou immédiatement.

Un encart de réflexion

La note de réflexion propose au lecteur d'approfondir lui-même un sujet ou un élément du cours. L'objectif de l'encart de réflexion et de permettre un apprentissage plus poussé et des compétences meilleures.

Un encart de note

Note : il est utile de fournir des astuces ou bien des rappels. Une note reste facultative : elle permet d'approfondir une connaissance.