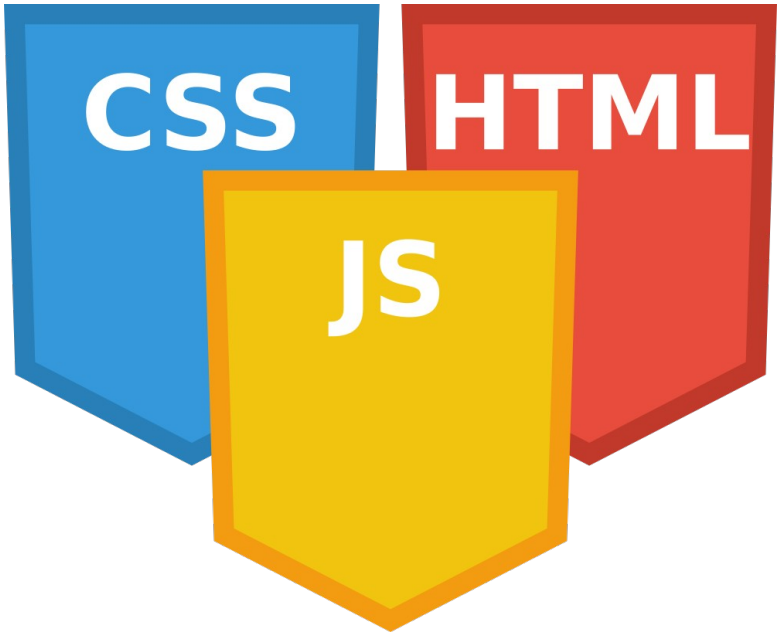
Three.JS
University

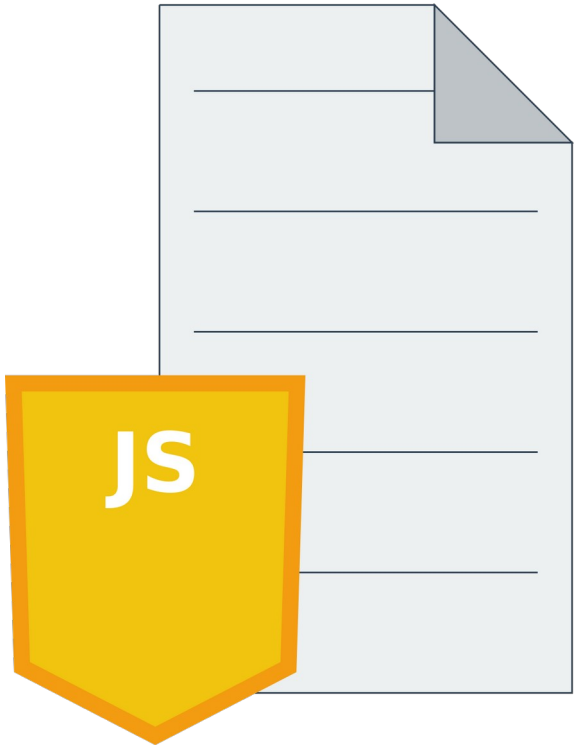
Hello World

Architecture de base du projet



- Architecture web **HTML** / **CSS** / **JS**
- Utilisation de Three.js avec les modules JavaScript **ES6**
- Il est toujours possible d'importer Three.js avec une balise **script**, sans les modules **ES6**

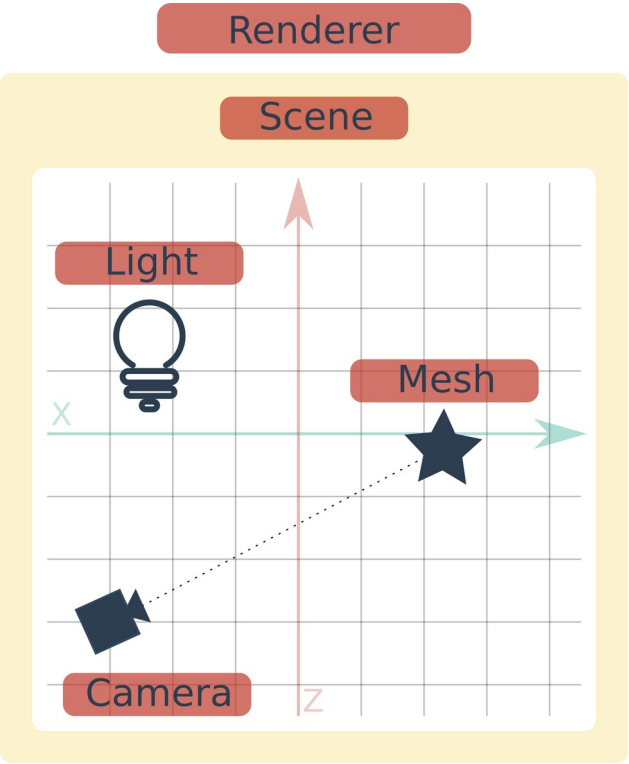
Architecture de base du code JavaScript



Notre code JavaScript de base se divise en quatre parties principales :

- Chargement des modules **ES6** (**import**)
- Variables globales
- Fonction d’initialisation (**init**)
- Boucle principale d’animation (**render**)

La classe Scene

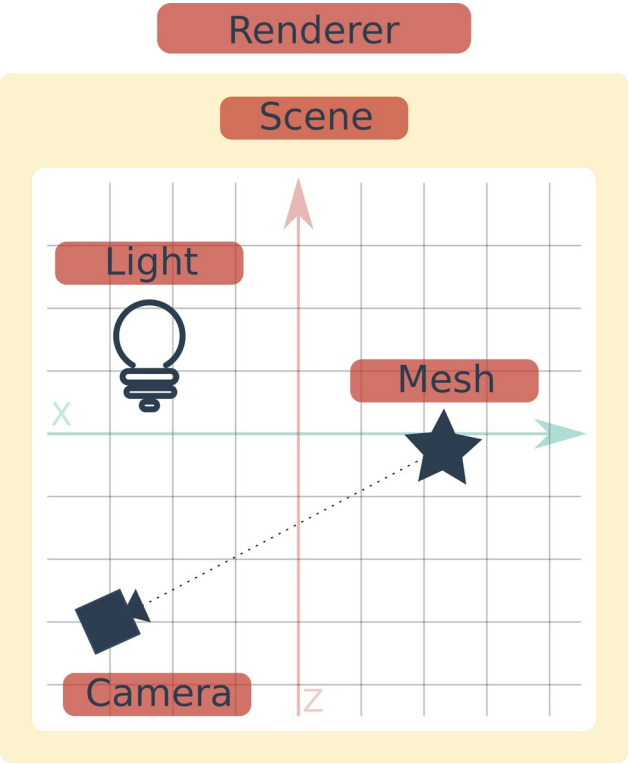


- La classe **Scene** permet de créer un univers 3D
- Aucun paramètre ou option dans le constructeur
- Utilisation de la méthode **add** pour ajouter des éléments dans la scène

```
// Global Variable
var scene;

//init
scene = new THREE.Scene();
```

Renderer – Le moteur de rendu 3D



- La classe **WebGLRenderer** affiche une scène Three.js grâce à la technologie **WebGL**
- Un objet JavaScript est utilisé en paramètre pour configurer **WebGLRenderer**
- La propriété **antialias** active ou désactive l'anticrénelage

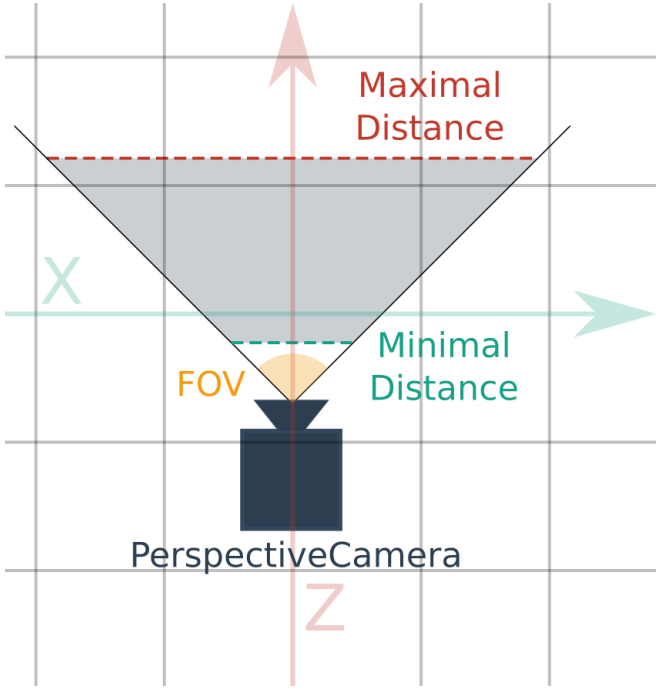
```
renderer = new THREE.WebGLRenderer( { antialias : true } );
renderer.setPixelRatio( window.devicePixelRatio );
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement );
```

WebGLRenderer – setPixelRatio, setSize et domElement

- La méthode **setPixelRatio** est utilisée pour définir un ratio entre la taille de l'écran et le nombre de pixels
- La méthode **setSize** redimensionne l'élément **HTML** du rendu 3D en fonction du **pixelRatio**
- L'objet **domElement** représente l'élément **canvas HTML** utilisé par le rendu 3D

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

PerspectiveCamera – La caméra de notre projet



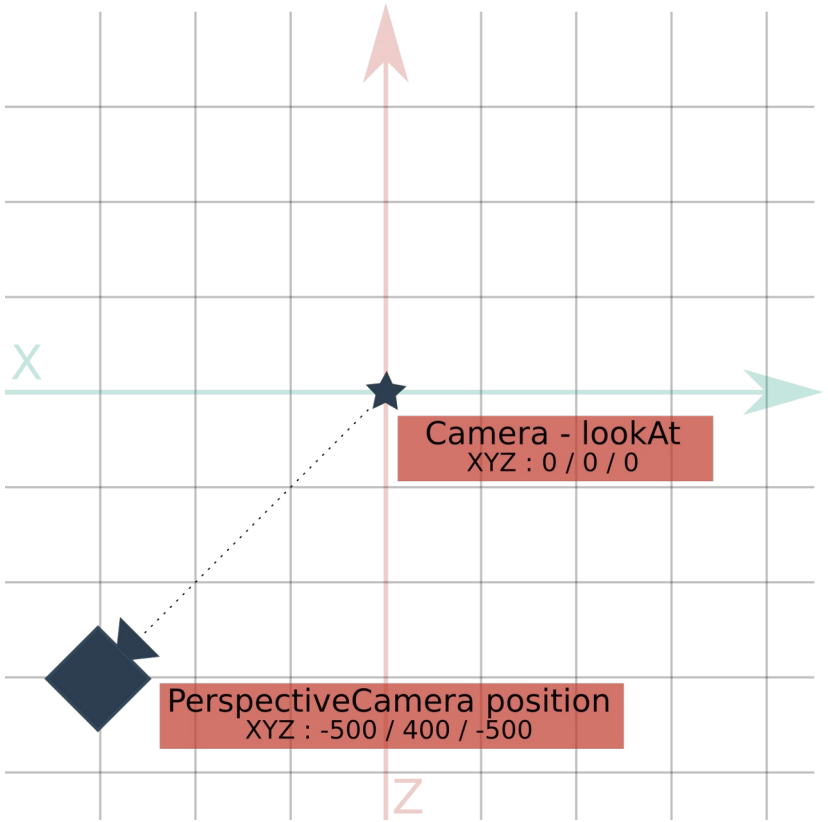
Projection basée sur la perspective, similaire à la vision humaine

Le constructeur de **PerspectiveCamera** accepte quatre paramètres :

- **FOV** – Field of View (Ouverture de l'objectif)
- **Aspect** – Le ratio de projection
- **Distance minimale** – Distance de vue minimale de la caméra
- **Distance maximale** – Distance de vue maximale de la caméra

```
camera = new THREE.PerspectiveCamera( 75, window.innerWidth / window.innerHeight, 1, 10000);
```

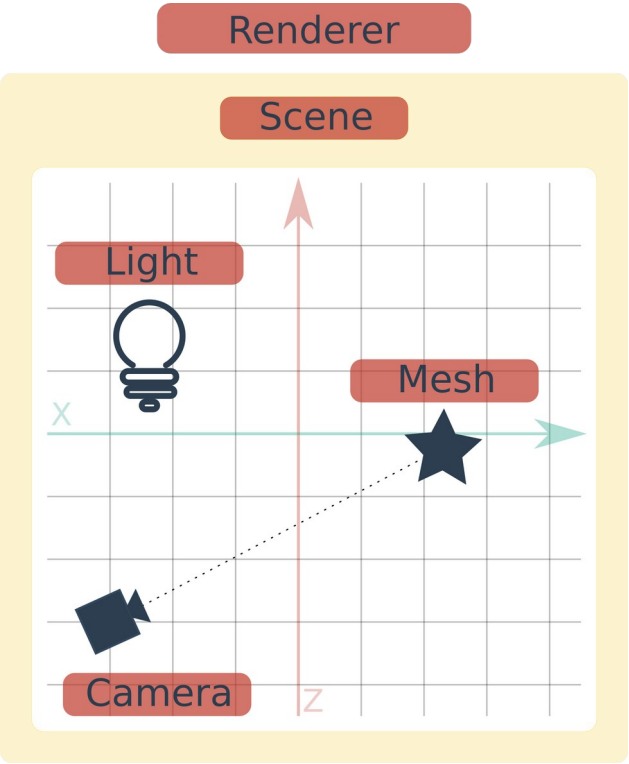
PerspectiveCamera – lookAt



- La méthode **lookAt** oriente un élément de la scène en direction d'une cible
- Cette méthode nécessite un seul argument : la position de la cible sur les trois axes **X**, **Y** et **Z**

```
camera = new THREE.PerspectiveCamera( 75, windo
camera.position.set(-500, 400, -500);
camera.lookAt( new THREE.Vector3( 0, 0, 0) );
scene.add(camera);
```

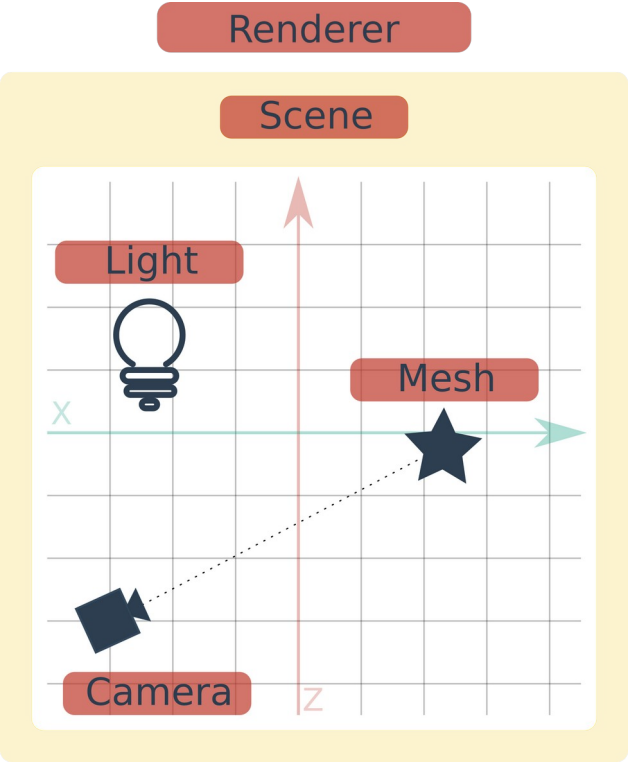
Ajouter la caméra à la scène - add



- La méthode **add** permet de créer une relation **parent** / **child** (enfant) entre deux éléments
- Cette méthode nécessite un paramètre : l'objet enfant
- Ici, nous utilisons **add** pour ajouter notre caméra à la scène

```
camera = new THREE.PerspectiveCamera( 75, windo
camera.position.set(-500, 400, -500);
camera.lookAt( new THREE.Vector3( 0, 0, 0) );
scene.add(camera);
```

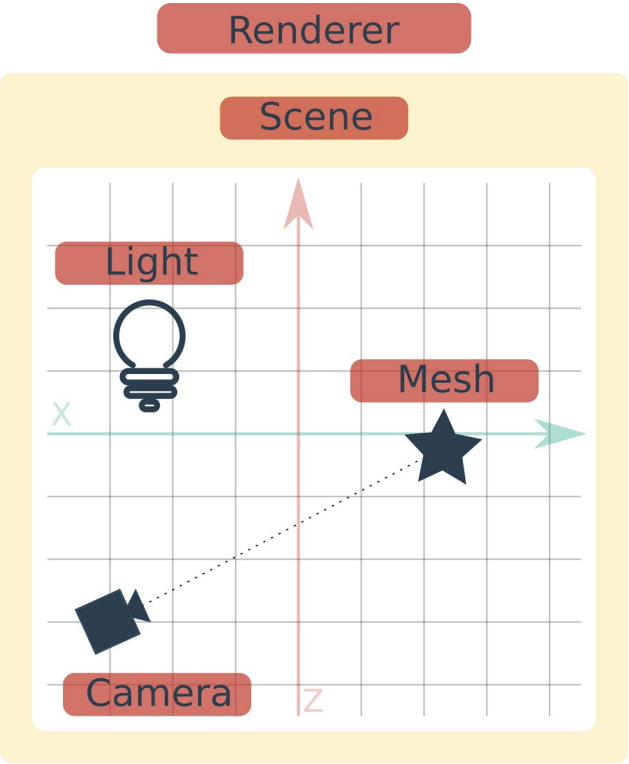
Éclairage global - AmbientLight



- Illumine de manière égale chaque face de tous les objets 3D de la scène
- Ne simule pas une source de lumière naturelle, mais un éclairage global des objets
- De par sa nature, impossible de créer des ombres et effets de lumière

```
var ambientLight = new THREE.AmbientLight(0xCCCCCC , 0.5);
scene.add(ambientLight);
```

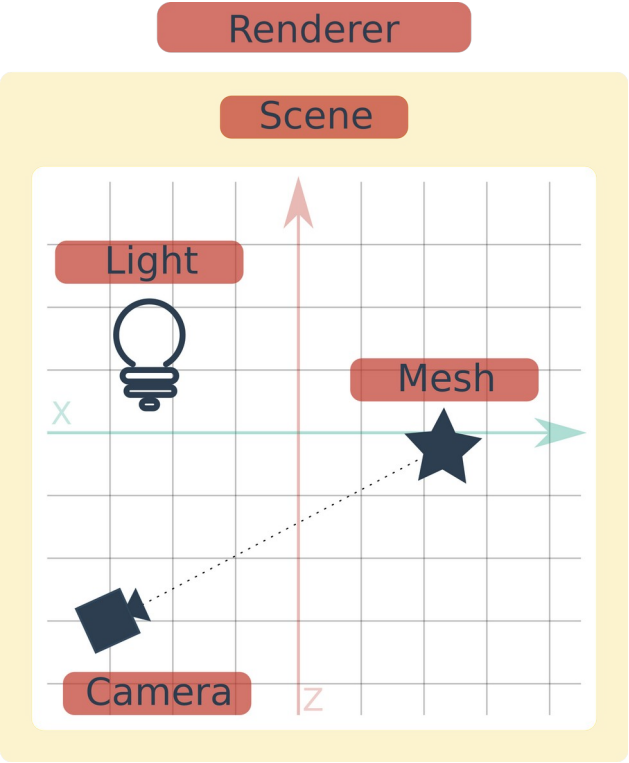
Éclairage directionnel - DirectionalLight



- La classe **AmbientLight** n’est pas suffisante pour distinguer correctement les objets 3D
- **DirectionalLight** simule une source lumineuse émise dans une direction
- Ce type d’éclairage est capable de créer des ombres et effets de lumière

```
var directionalLight = new THREE.DirectionalLight( 0xFFFFFF, 1 );
scene.add(directionalLight);
```

AmbientLight et DirectionalLight - Constructeur



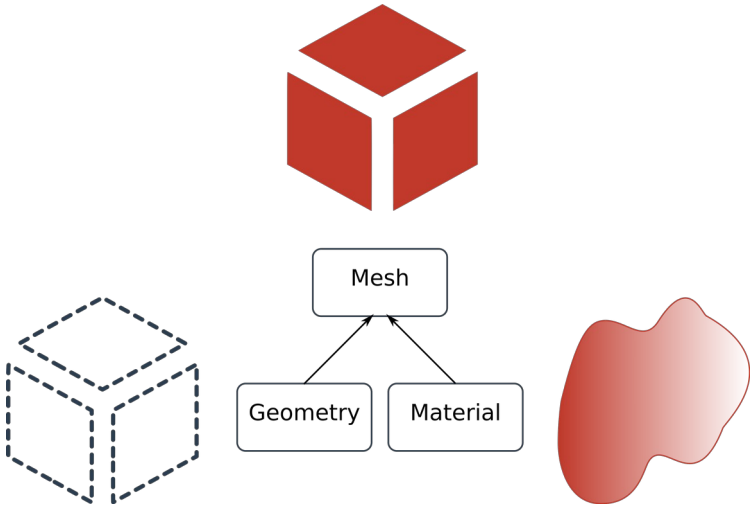
Le constructeur de ces deux classes accepte ici deux paramètres :

- Couleur de l'éclairage
- Intensité de l'éclairage

```
var ambientLight = new THREE.AmbientLight(0xCCCCCC , 0.5);
scene.add(ambientLight);

var directionalLight = new THREE.DirectionalLight( 0xFFFFFF, 1 );
scene.add(directionalLight);
```

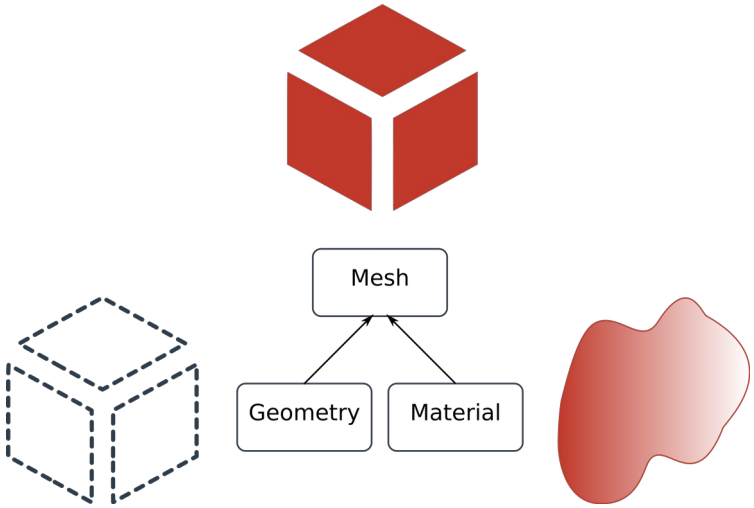
Notre premier Mesh – Un cube en 3D



- Notre **Mesh** sera créé à partir d'une **Geometry** de type **BoxGeometry** et d'un **Material** de type **MeshPhongMaterial**
- **BoxGeometry** représente une **Geometry** de forme parallélépipédique
- **MeshPhongMaterial** sera utilisé pour définir la couleur de notre objet 3D
- Le constructeur de **Mesh** accepte deux paramètres : Une **Geometry** et un **Material**

```
const geometry = new THREE.BoxGeometry( 800, 150, 300);
const material = new THREE.MeshPhongMaterial( {color : 0xc0392b } );
cube = new THREE.Mesh(geometry, material);
scene.add(cube);
```

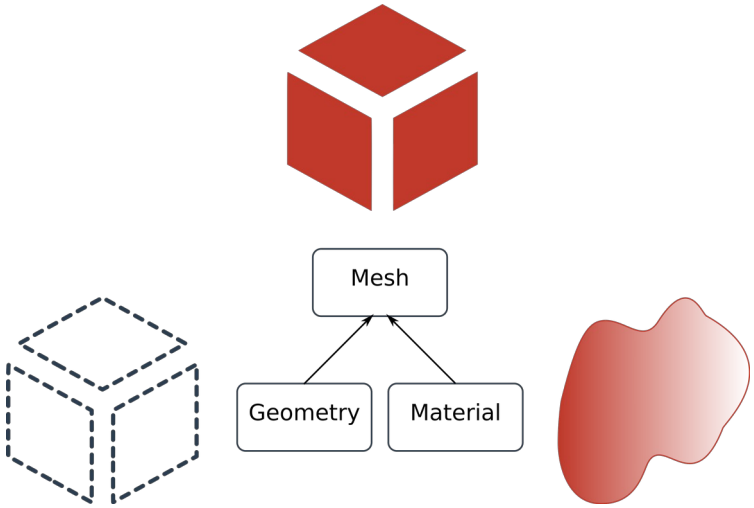
Notre premier Mesh – BoxGeometry



- **BoxGeometry** représente une **Geometry** de forme parallélépipédique
- Son constructeur nécessite trois paramètres : l'échelle de la structure sur chaque axe **X**, **Y** et **Z**
- En utilisant trois valeurs identiques, il est possible de créer une **Geometry** cubique
- Il existe plusieurs types de **Geometry**

```
const geometry = new THREE.BoxGeometry( 800, 150, 300);
const material = new THREE.MeshPhongMaterial( {color : 0xc0392b } );
cube = new THREE.Mesh(geometry, material);
scene.add(cube);
```

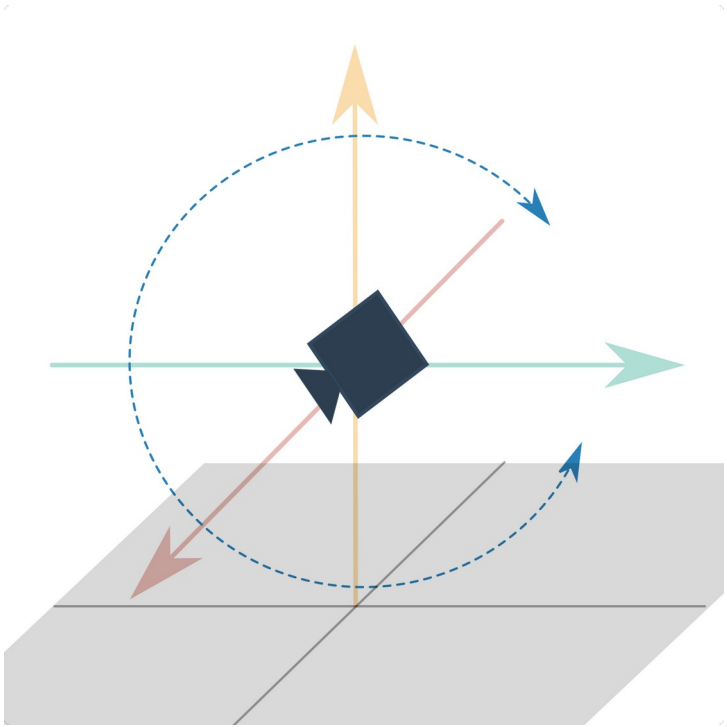
Notre premier Mesh – MeshPhongMaterial



- Nous utiliserons **MeshPhongMaterial** pour l'aspect visuel de notre cube
- Il existe plusieurs types de classes **Material**
- Un objet JavaScript est utilisé en paramètre pour configurer **MeshPhongMaterial**
- La propriété **color** permet de définir la couleur de notre **Material**

```
const geometry = new THREE.BoxGeometry( 800, 150, 300);
const material = new THREE.MeshPhongMaterial( {color : 0xc0392b } );
cube = new THREE.Mesh(geometry, material);
scene.add(cube);
```

Position, Rotation et Scale



- Les propriétés **position**, **rotation** et **scale** sont utilisées pour définir respectivement la **Position**, **Rotation** et l'**Échelle** d'un objet
- Il est possible de définir une valeur par axe **X**, **Y** et **Z**
- Pour cela, il est possible d'utiliser directement les sous-propriétés **x**, **y** ou **z**
- Il est également possible d'utiliser la méthode **set**

```
cube.rotation.y = Math.PI/2;
cube.rotation.x += 0.005;

camera.position.set(-500, 400, -500);

cube.scale.z = 2;
```

Hello World

Si vous êtes prêts, passons à la pratique !