Bibliothèque 3D Three.js

Enseignante: Amel TILOUCHE
Niveau: MDW5

Les bases de Three.js

- Présentation de Three.js
- Graphe de scène
- Structure d'un programme Three.js
- Prototypes de base sous Three.js
  - Object3D
  - Camera
  - Mesh – Geometry – Material
  - Light

Limites de WebGL

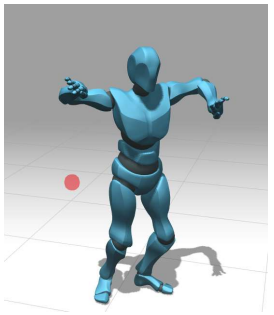
- Permet d'accéder à l'accélération 3D via JS mais reste de bas niveau :
  - Aucun outil de modélisation de la géométrie.
  - Gère uniquement le pipeline graphique 3D → 2D pour une image, l'animation nécessite du code supplémentaire.
  - Les mouvements et les interactions entre les objets et l'environnement nécessite du code supplémentaire.
  - La gestion des événements (entrées utilisateur), la sélection d'objet, etc. nécessite du code supplémentaire.

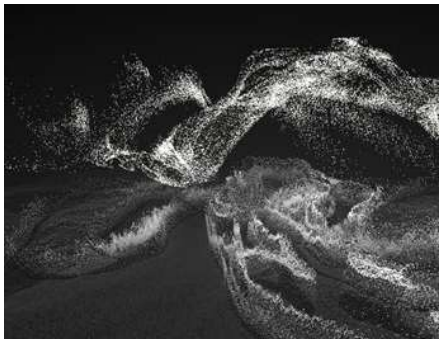
Three.js ?

- Bibliothèque 3D pour JavaScript créé en 2010.
- Bibliothèque à base de **graphe de scène**.
- Bibliothèque de haut niveau, la plus utilisée pour développer et maintenir plus aisément un projet WebGL.
- Permet des rendus en WebGL, CSS3 et SVG.
- Permet la création et l'animation des scènes 3D dans le navigateur.

Three.js ?

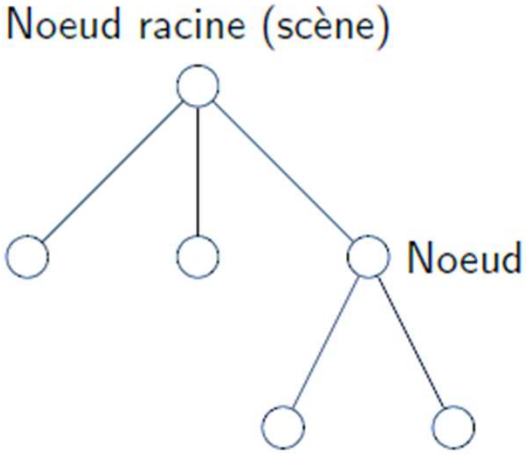
- Fonctionnalités avancées offertes:
  - Animation par squelette,
  - LOD (niveau de détails pour les objets),
  - Chargement de Fichiers au formats .OBJ, .JSON, .FBX,
  - Système de particules.





Graphe de scène

- Organisation hiérarchique des différents éléments composant une scène 3D.
- Une structure arborescente représentée par un arbre (graphe orienté acyclique) décrivant les objets constituant un scène 3D.
- Les propriétés appliquées aux ancêtres s'appliquent aussi aux descendants.

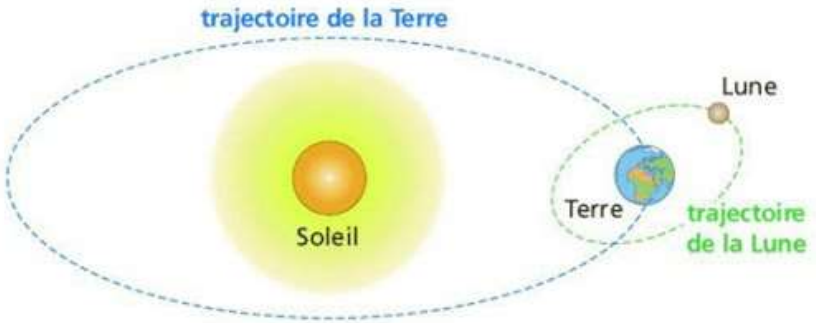


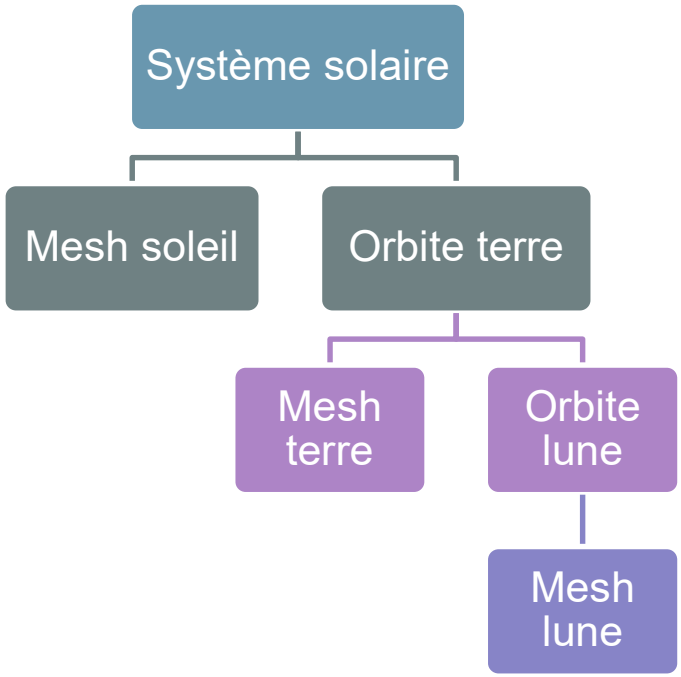
Graphe de scène

- Un nœud possède des propriétés ;
- Dans la pratique, elles sont modifiables ;
- Ces propriétés sont appelées des champs.
- *Exemples:*
  - Sphère : Rayon ;
  - Boite : Largeur, Longueur, Hauteur ;
  - Transformation : Matrice de transformation.

Graphe de scène

- *Exemple:*





Liens Three.js

- Site officiel de Three.js: https://threejs.org/
- Téléchargement de Three.js: https://discoverthreejs.com/book/introduction/get-threejs/

Installation Three.js

- Méthode 1
  - Extraire le dossier zippé de la bibliothèque.
  - Importer les fichiers nécessaires pour votre application à travers la balise *<script>*.
    - Tous les fichiers JS utiles se trouvent sous les dossiers *build* et *examples\jsm*

Installation Three.js

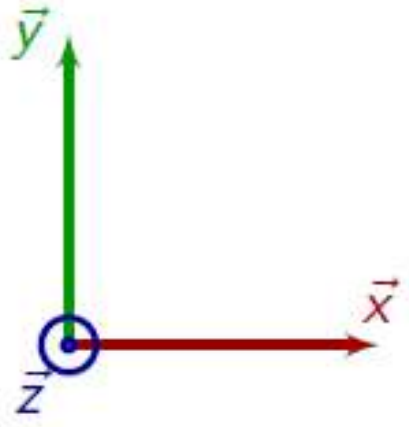
- Méthode 2
  - Installation du Framework node.js (https://nodejs.org/en/download/)
    - Pour vérifier si node.js est déjà installé, lancer l'éditeur des commandes et taper: *node --version*
  - Au niveau du dossier de la bibliothèque exécuter l'éditeur de commandes (cmd) et installer three.js en utilisant les commandes suivantes:

    npm init

    npm install --save three

Conventions Three.js

- Prototypes (classes) dans l'espace de nommage THREE ;
- Notations de type CamelCase ;
- Repère main droite ;
- Angles en radians.



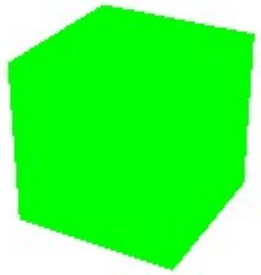
Etapes de création d'une scène

- Préparation du document html à afficher dans le navigateur ;
- Création de la scène ;
- Génération de rendu de la scène ;
- Mise en place des animations.

Premier programme HTML/Three.js

```
<html>
<head>
  <title>Mon premier programme three.js</title>
</head>
<body>
<script src="three.js"></script>
<script>
var scene = new THREE.Scene();
var camera = new
     THREE.PerspectiveCamera(75,window.innerWidth/window.inner
     Height, 0.1, 1000 );
var renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);
```

```
var geometry = new THREE.BoxGeometry();
var material = new THREE.MeshBasicMaterial({color: 0x00ff00});
var cube = new THREE.Mesh(geometry, material);
cube.position.y = 1;
Scene.add(camera);
scene.add(cube);
camera.position.z = 5;
var animate = function () {
    requestAnimationFrame(animate);
    cube.rotation.x += 0.01;
    cube.rotation.y += 0.01;
    renderer.render(scene, camera);};
animate();
</script>
</body>
</html>
```



Préparation de document HTML

```
<html>
    <head>
        <title>Mon premier programme three.js</title>
    </head>
    <body>
        <script src="three.js"></script>
        <script> //code JavaScript à mettre ici. </script>
    </body>
</html>
```

Remarque: Le script introduira un canvas pour effectuer le rendu.

Création de la scène

Instructions mettant en place la scène

```
var scene = new THREE.Scene();
var camera = new
     THREE.PerspectiveCamera(75,window.innerWidth/window.innerH
     eight, 0.1, 1000 );
var renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);
```

- *Commentaires:*
  - ✓ Eléments nécessaires : scène, caméra, zone de rendu ;
  - ✓ La taille de la zone de rendu est indépendante de la taille du canvas dans lequel est effectué le rendu ;
  - ✓ La zone de rendu est un canvas ajouté au document HTML.

Création de la scène

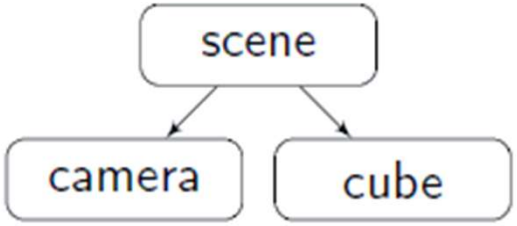
Ajout d'une géométrie à la scène

```
var geometry = new THREE.BoxGeometry();
var material = new THREE.MeshBasicMaterial({color: 0x00ff00});
var cube = new THREE.Mesh(geometry, material);
cube.position.y = 1;
Scene.add(camera);
scene.add(cube);
camera.position.z = 5;
```

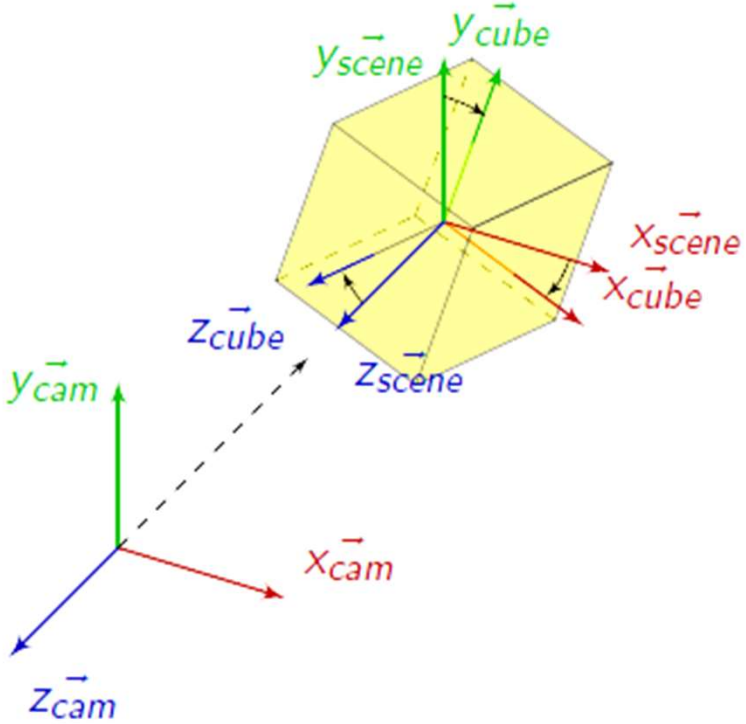
- *Commentaires:*
  - ✓ Création d'un cube avec une géométrie et un matériau ;
  - ✓ Ajout du cube à la scène ;
  - ✓ Positionnement de la caméra par rapport à la scène.

Création de la scène

Graphe de scène



Repères



Génération de rendu

Création de la boucle de rendu

```
var animate = function () {
   requestAnimationFrame(animate);
   renderer.render(scene, camera);};
animate();
```

- *Commentaires:*
  - ✓ Création de la fonction animate();
  - ✓ Appel de cette fonction pour chaque image;
  - ✓ Génération de rendu de la scène avec la caméra donnée dans la zone de rendu créé;
  - ✓ Premier appel de la fonction animate().

Mise en place des animations

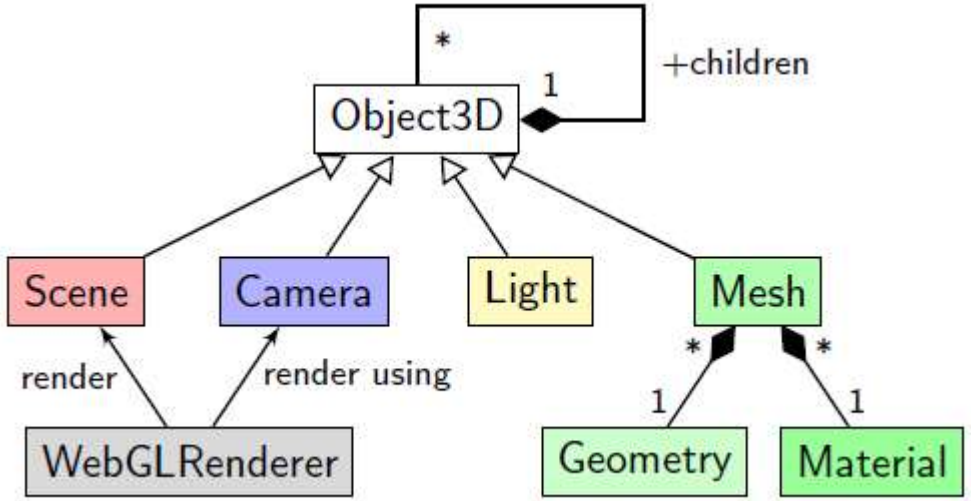
Animation du cube

```
cube.rotation.x += 0.01;
cube.rotation.y += 0.01;
```

- *Remarque:*
  - ✓ La fréquence de rafraichissement est de l'ordre de 60 Hz.
  - ✓ Mais elle est dépendante de la vitesse à laquelle le navigateur fournit une nouvelle image via requestAnimationFrame().

Hiérarchie des prototypes Three.js

- Les principaux prototypes de Three.js sont:



Prototype Object 3D

- Prototype de base pour tous les nœuds des graphes de scène Three.js capable de gérer :
  - les transformations ;
  - les relations père-enfants dans le graphe.
- Propriétés:
  - *parent, children :* relations au niveau du graphe de scène ;
  - *position, rotation, scale, matrix :* transformations locales ;
  - *id, name :* nom et identifiant de l'objet.

Prototype Object 3D

- Three.js évite la manipulation directe des matrices 4x4 en proposant une interface plus intuitive via les propriétés *position*, *rotation* et *scale* des objets.
- La position par défaut est (0, 0, 0).
- *Exemples:*
  - Pour placer le cube à la position (0, 1, 0): `cube.position.y = 1;`
  - Pour éloigner la caméra de l'origine de la scène: `camera.position.z = 5;`

Prototype Object 3D

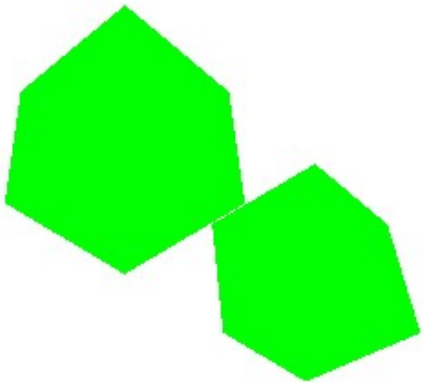
- Les transformations appliquées au parent s'appliquent aussi aux descendants.
- Ordre d’application des transformations:
  1. Translation
  2. Rotation
  3. Mise en échelle

Prototype Object 3D

- Méthodes principales:
  - *add(), remove() :* gestion des enfants ;
  - *applyMatrix4(), translate[X|Y|Z](), rotate[X|Y|Z](), lookAt(), translateOnAxis(), rotateOnAxis() :* diverses méthodes pour appliquer des transformations ;
  - *getObjectByName(), getObjectById() :* recherche d'objet dans le graphe par son nom ou son identifiant.

Exemple: créer un objet 3D animé composé de deux cubes dont le deuxième est identique au premier.

```
var geometry = new THREE.BoxGeometry();
var material = new THREE.MeshBasicMaterial({color: 0x00ff00});
var cube1 = new THREE.Mesh(geometry, material);
cube1.position.y = 1;
var cube2 = cube1.clone();
cube2.position.set(-2,0,0);
var parent= new THREE.Object3D();
parent.add(cube1);
parent.add(cube2);
scene.add(parent);
var animate = function () {
    requestAnimationFrame(animate);
    parent.rotation.x += 0.01;
    parent.rotation.y += 0.01;
    renderer.render(scene, camera);};
animate();
</script>
</body>
</html>
```

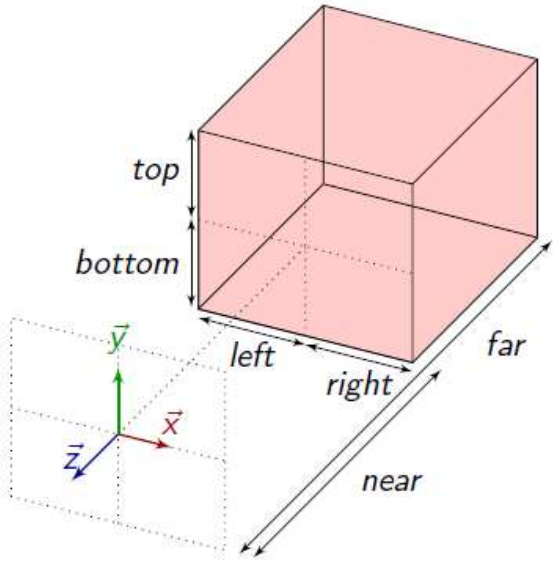


Prototype Camera

- Décrive une caméra qui peut être:
  - *OrthographicCamera:* caméra en perspective cavalière ;
  - *PerspectiveCamera:* caméra en perspective classique.
- Propriété de base: *projectionMatrix*

Prototype Camera

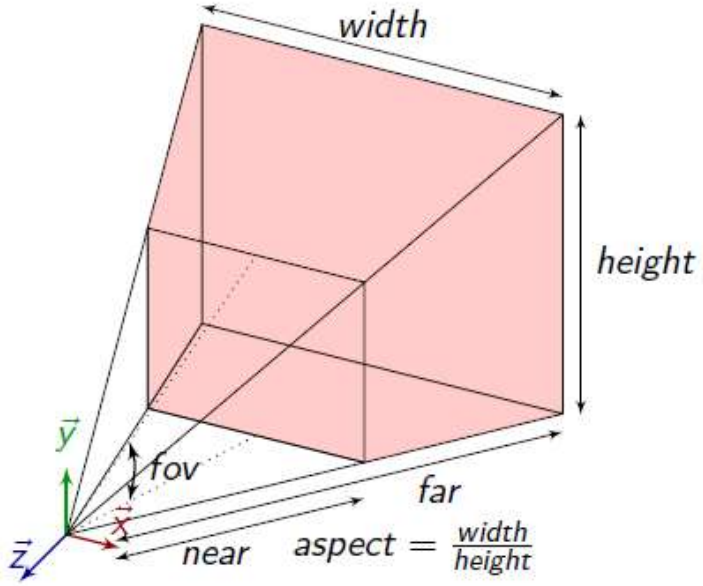
- Prototype OrthographicCamera:
  - Constructeur : *`OrthographicCamera(left, right, top, bottom, near, far)`*
  - **left, right, top, bottom, near, far:** distances des plans de clipping par rapport au repère local de la caméra.
  - **Propriétés :** paramètres du constructeur.



La zone visualisée est en boîte.

Prototype Camera

- Prototype PerspectiveCamera:
  - Constructeur : *`PerspectiveCamera(fov, aspect, near, far)`*
    - ✓ fov : angle de vision vertical ;
    - ✓ aspect : rapport largeur/hauteur ;
    - ✓ near, far : distances des plans de clipping proches et éloignés.
  - Propriétés : paramètres du constructeur.



La zone visualisée est en pyramide tronquée.

Prototype Mesh

- Prototype permettant de décrire une géométrie et son aspect ;
- Constructeur : *`Mesh(geometry, material)`*
  - geometry : instance du prototype *`BufferGeometry`*;
  - material : instance du prototype *`Material`*.
- Propriétés : geometry, material.

Prototype BufferGeometry

- Prototype de base contenant toutes les données nécessaires à décrire un modèle 3D.
- Géométries prédéfinies
  - Cube: *BoxGeometry*
  - Cône: *ConeGeometry*
  - Sphère: *SphereGeometry*
  - Cylindre: *CylinderGeometry*
  - Torus: *TorusGeometry*
  - Tube: *TubeGeometry*
  - Tétraèdre: *TetrahedronGeometry*
  - Icosaèdre: *IcosahedronGeometry*
  - Octaèdre: *OctahedronGeometry*
  - Plan: *PlaneGeometry*
  - Texte: *TextGeometry*

Prototype BufferGeometry

- Exemple: constructeur d'une boîte

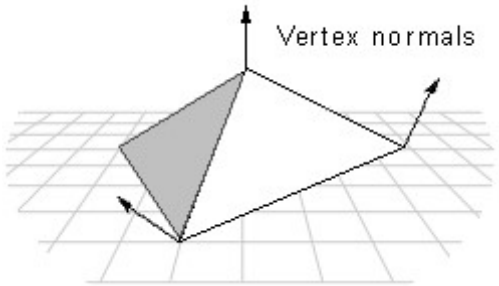
  `BoxGeometry(width, height, depth, widthSegments, heightSegments, depthSegments)`

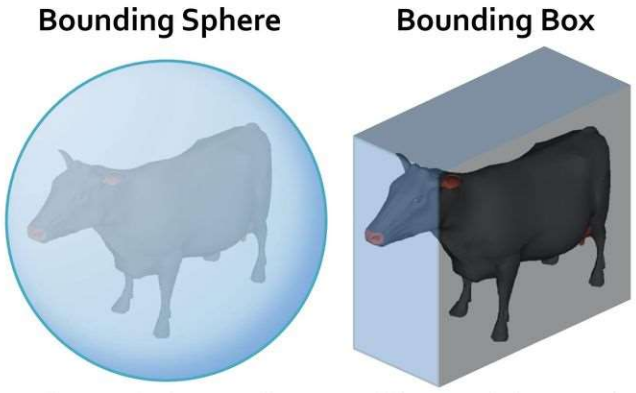
  - Width, height, depth: largeur, hauteur et longueur de la boîte.
  - widthSegments, heightSegments, depthSegments: nombre segments pour le maillage sur la largeur, l'hauteur et la longueur.

- Propriétés:
  - *parameters:* les paramètres du constructeur.
  - *boundingBox:* la boite englobante
  - *boundingSphere:* la sphère englobante.

Prototype BufferGeometry

- Méthodes importantes :
  - *computeVertexNormals() :* calcule normales par sommet ;
  - *computeBoundingBox() :* calcule la boite englobante ;
  - *computeBoundingSphere() :* calcule la sphère englobante.





Prototype BufferGeometry

- Définition d'une géométrie:

```
const geometry = new THREE.BufferGeometry();
const vertices = new Float32Array([-1.0, -1.0, 1.0,
1.0, -1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, -1.0,
1.0, 1.0, -1.0, -1.0, 1.0]);
geometry.setAttribute('position', new
THREE.BufferAttribute(vertices,3));
const material = new THREE.MeshBasicMaterial( {
color: 0xff0000 } );
const mesh = new THREE.Mesh(geometry, material );
```

Prototype BufferGeometry

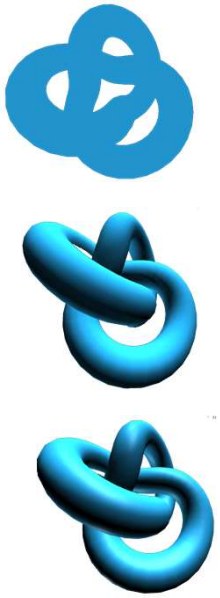
- Calcul de boîte englobante et normales sommets

```
geometry.computeBoundingBox();
console.log(JSON.stringify(geometry.boundingBox));

var normals = new Float32Array(vertices,length);
geometry.addAttribute('normal', new
THREE.BufferAttribute(normals,3));
geometry.computeVertexNormals();
console.log(normals);
```

Prototype Material

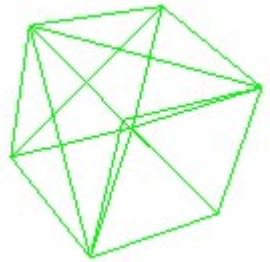
- Prototype permettant de décrire un matériau (l'apparence de l'objet) qui peut être de plusieurs types :
  - *MeshBasicMaterial:* pour un ombrage simple ne nécessitant pas de lumière pour le rendu (aspect fil de fer ou plat).
  - *MeshLambertMaterial:* illumination par sommet selon le modèle Lambertien (aspect mat).
  - *MeshPhongMaterial:* illumination par pixel selon le modèle de Phong (aspect brillant).
  - *ShaderMaterial:* utilisation de son propre shader.



Prototype Material

- Propriétés de base:
  - *opacity :* 0 transparent, 1 opaque ;
  - *transparent :* true si transparent (aspect contrôlé par opacity).
- Dessin en mode filaire:

```
var geometry = new THREE.BoxGeometry();
var material = new
THREE.MeshBasicMaterial({color: 0x00ff00,
wireframe: true});
var cube = new THREE.Mesh(geometry, material);
```



Prototype Material

- Remarques
  - les couleurs sont codées en hexadécimal (*Exemple:* `0x00ff00` pour le vert).
  - Three.js prend en charge l'allocation du tableau des couleurs et son chargement dans le buffer approprié.
  - De plus, contrairement à WebGL, le code GLSL correspondant, sa compilation et son édition de lien sont masqués.

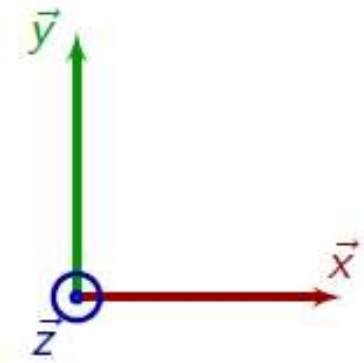
Prototype Light

- Prototype permettant de décrire une lumière qui peut être de plusieurs types :
  - *AmbientLight :* lumière ambiante ;
  - *DirectionalLight :* lumière directionnelle ;
  - *PointLight :* lumière ponctuelle ;
  - *SpotLight :* lumière de type spot.
- Constructeur de base: `Light(color, intensity)`
- Propriétés de base : *color, intensity.*

TP 1

Transformations 3D

Translation



- Définir une position 3D:

```
objet.position.set(x,y,z);
```

- Translation suivant l'un des axes 3D:

```
objet.position.x+=pasX; ou objet.translateX(pasX);
objet.position.y+=pasY; ou objet.translateY(pasY);
objet.position.z+=pasZ; ou objet.translateZ(pasZ);
```

Translation

- Translation suivant un axe normalisé dans l'espace d'objet:

```
objet.translateOnAxis(axis,distance);
```

  - *axis:* vecteur normalisé de l'espace objet.
  - *distance:* le pas de translation

Rotation

- Définir une orientation 3D:

```
objet.rotation.set(angleX,angleY,angleZ);
```

- Rotation autour de l'un des axes 3D:

```
objet.rotation.x+=angleX; ou objet.rotateX(angleX);
objet.rotation.y+=angleY; ou objet.rotateY(angleY);
objet.rotation.z+=angleZ; ou objet.rotateZ(angleZ);
```

Rotation

- Rotation suivant un axe normalisé dans l'espace d'objet:

```
objet.rotateOnAxis(axis:Vector3,angle:Float);
```

  - *axis:* vecteur normalisé de l'espace objet.
  - *angle:* angle de rotation.

Mise en échelle

- Définir simultanément les 3 facteurs d'échelle:

```
objet.scale.set(scaleX,scaleY,scaleZ);
```

- Définir un facteur d'échelle sur l'un des axes 3D:

```
objet.scale.x=scaleX;
objet.scale.y=scaleY;
objet.scale.z=scaleZ;
```

Définition d'une orientation

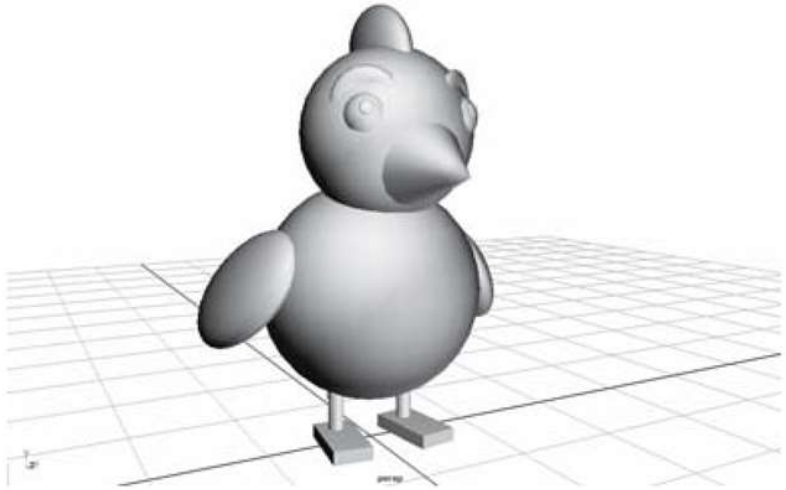
- Faire pivoter l'objet pour faire face à un point dans l'espace global:

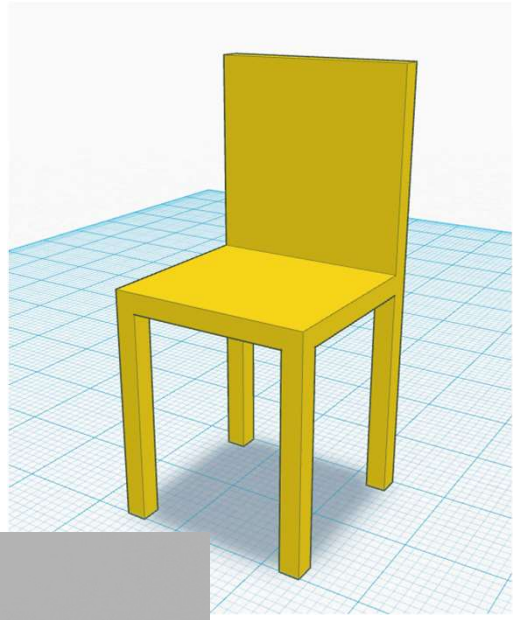
```
objet.lookAt((vector:Vector3));
objet.lookAt(x:Float,y:Float,z:Float);
```

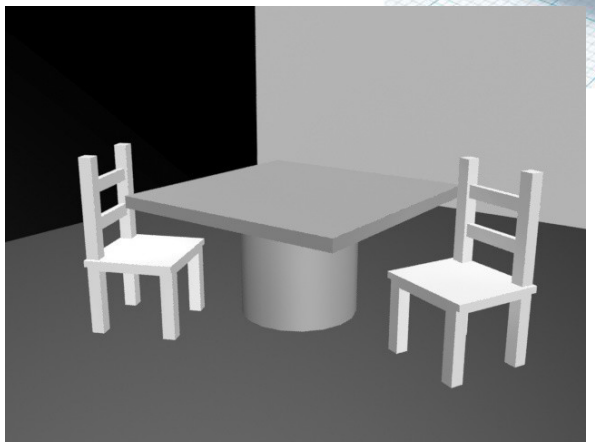
- *Exemples:*

```
camera.lookAt(new THREE.Vector3(1,2,1));
camera.lookAt(cube.position);
```

TP 2







3

Eclairage et matériaux

- Sources de lumières
  - Lumière ambiante
  - Lumière ponctuelle
  - Lumière en spot
  - Lumière directionnelle
- Matériaux
  - Plat
  - Lambertien
  - Phong

Introduction

- On rappelle que le THREE.MeshBasicMaterial ne nécessite aucune source lumineuse pour le rendu.
- Pour augmenter le réalisme, on doit ajouter des sources lumineuses :
  - Ambiante
  - Ponctuelle
  - Directionnelle
  - Spot

Lumière ambiante

- Eclaire la scène uniformément.
- Définie par une couleur et une intensité.

```
var material=new THREE.MeshLambertMaterial({map:texture});
…
var alight= new THREE.AmbientLight(0x909090,0.9);
scene.add(alight);
```

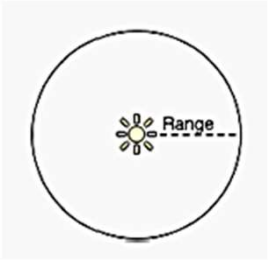
Lumière ambiante

- Propriétés:
  - *intensity:* décrit l'intensité de la source de lumière (par défaut 1).
  - *color:* décrit la couleur de la source de lumière.
  - *visible:* définit si la source de lumière est visible (true: valeur par défaut) ou invisible (false).

Lumière ambiante

- Color Object: la couleur en Three.js peut être définie suivant plusieurs présentations:
  - Constructeur vide: `new THREE.Color()` pour créer une couleur par défaut blanche
  - En valeur hexadécimale: `new THREE.Color(0xababab)`
  - En valeurs RVB: `new THREE.Color(1, 0, 0)`
  - En chaîne RVB: `new THREE.Color(« rgb(255, 0, 0) »)` ou `new THREE.Color(« rgb(100%, 0%, 0%) »)`
  - En chaîne HSL: `new THREE.Color(« hsl(0, 100%, 50%) »)`
  - Nom de la couleur: `new THREE.Color('skyblue')`

Lumière ponctuelle



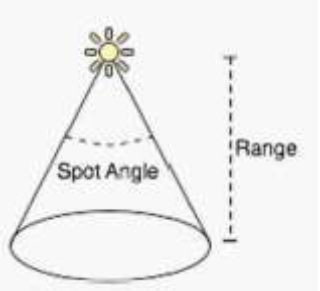
- Emet la lumière d'un seul point vers toutes les directions.
- Définie par une position, une couleur, une intensité et un rayon d'action.
- Intensité en un point varie selon la distance qui sépare celui-ci de la source lumineuse.

```
var material=new THREE.MeshLambertMaterial({map:texture});
…
var alight= new THREE.AmbientLight(0x909090,0.9);
scene.add(alight);
var plight = new THREE.PointLight(0xffffff,1,100);
plight.position.set(50,50,50);
scene.add(plight);
```

Lumière ponctuelle

- Propriétés de base:
  - *intensity:* décrit l'intensité de la source de lumière (par défaut 1).
  - *color:* décrit la couleur de la source de lumière.
  - *position:* décrit la position de la source de lumière.
  - *distance:* décrit le rayon d'action de la source de lumière.
  - *visible:* définit si la source de lumière est visible (true: valeur par défaut) ou invisible (false).

Lumière en spot



- Définie par une couleur, une intensité, un rayon d'action (distance à partir de laquelle l'intensité devient nulle), un angle de spot et une position.
- Eclaire la scène à travers un cône (intensité forte au centre du cône et faible à ses bords).

Lumière en spot

```
var material=new THREE.MeshPhongMaterial({map:texture});
…
var alight= new THREE.AmbientLight(0x909090,0.9);
scene.add(alight);

var slight = new THREE.SpotLight(0xffffff,1,100,Math.PI/2);
slight.position.set(30,40,50);
slight.castShadow = true;
slight.shadow.mapSize.width = 1024;
slight.shadow.mapSize.height = 1024;
slight.shadow.camera.near = 500;
slight.shadow.camera.far = 4000;
slight.shadow.camera.fov = 30;
scene.add(slight);
```

Lumière en spot

- Propriétés:
  - *intensity:* décrit l'intensité de la source de lumière (par défaut 1).
  - *color:* décrit la couleur de la source de lumière.
  - *position:* décrit la position de la source de lumière.
  - *distance:* rayon d'action de la source de lumière.
  - *angle:* angle du spot (par défaut Math.PI/3).
  - *target:* décrit l'objet ou la position cible à laquelle pointe la source de lumière.
  - *visible:* définit si la source de lumière est visible (true: valeur par défaut) ou invisible (false).

Lumière en spot

- *castShadow:* décrit si la source de lumière crée des ombres (true) ou non (false).
- *shadow.camera.far:* détermine à quelle distance maximale les ombres de la source de lumière doivent être crées (par défaut 5000).
- *shadow.camera.fov:* détermine la taille de FOV utilisé pour créer des ombres (par défaut 50).
- *shadow.camera.near:* détermine à quelle distance minimale les ombres de la source de lumière doivent être crées (par défaut 50).

Lumière en spot

- *shadow.mapSize.width* et *shadow.mapSize.height:* définissent combien de pixels sur la largeur et l'hauteur sont utilisés pour créer les ombres (par défaut 512).
- *shadow.raduis:* si cette valeur est supérieur à 1, les contours des ombres seront flous.

Lumière directionnelle

- Définie par une couleur, une intensité et une direction.
- Intensité de la lumière qui atteint une surface n'est pas dépendante de la distance à la lumière.

```
var material=new THREE.MeshPhongMaterial({map:texture});
…
var alight= new THREE.AmbientLight(0x909090,0.9);
scene.add(alight);
var dlight = new THREE.DirectionalLight(0xffffff,0.5,(1,0,0));
scene.add(dlight);
```

Lumière directionnelle

- Propriétés:
  - *intensity:* décrit l'intensité de la source de lumière (par défaut 1).
  - *color:* décrit la couleur de la source de lumière.
  - *position:* décrit la position de la source de lumière.
  - *target:* décrit l'objet ou la position cible à laquelle pointe la source de lumière.
  - *visible:* définit si la source de lumière est visible (true: valeur par défaut) ou invisible (false).

Lumière directionnelle

- *castShadow:* décrit si la source de lumière crée des ombres (true) ou non (false).
- *shadow.camera.far:* détermine à quelle distance maximale les ombres de la source de lumière doivent être crées (par défaut 5000).
- *shadow.camera.near:* détermine à quelle distance minimale les ombres de la source de lumière doivent être crées (par défaut 50).
- *shadow.camera.right:* détermine à quelle distance par rapport au droite les ombres de la source de lumière doivent être crées.

Lumière directionnelle

- *shadow.camera.left:* détermine à quelle distance par rapport au gauche les ombres de la source de lumière doivent être crées.
- *shadow.camera.top:* détermine à quelle distance haute les ombres de la source de lumière doivent être crées.
- *shadow.camera.bottom:* détermine à quelle distance basse les ombres de la source de lumière doivent être crées.
- *shadow.mapSize.width* et *shadow.mapSize.height:* définissent combien de pixels sur la largeur et l'hauteur sont utilisés pour créer les ombres (par défaut 512).

Sources de lumière spéciales

- HemisphereLight

https://threejs.org/docs/#api/en/lights/HemisphereLight

- RectAreaLight

https://threejs.org/docs/#api/en/lights/RectAreaLight

- LensFlare

https://threejs.org/docs/?q=lensFlare#examples/en/objects/Lensflare