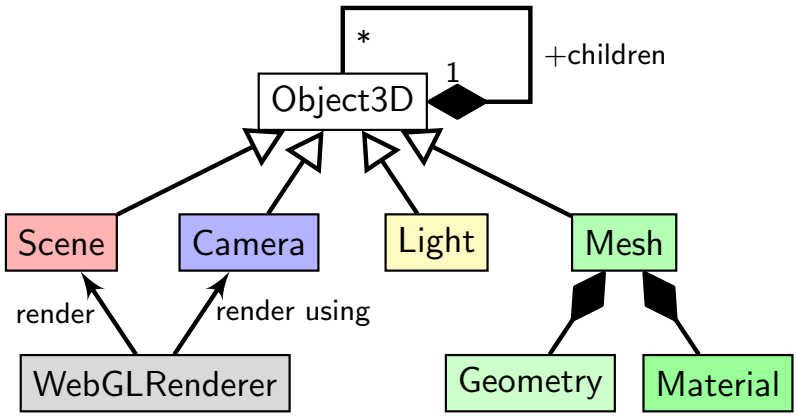
Présentation de three.js

ENSIIE2 : Option RVIG

Jean-Yves Didier

didier@ufrst.univ-evry.fr



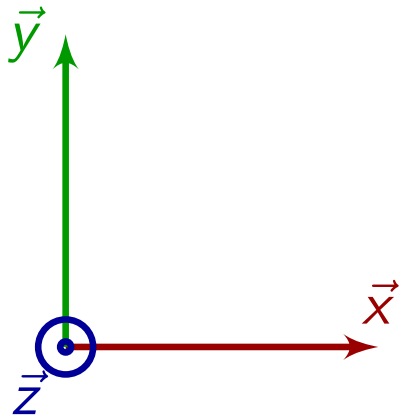
Caractéristiques de three.js

Caractéristiques

- Bibliothèque javascript de **graphes de scène** créé en 2010 ;
- S'appuie sur l'API **WebGL**
  (graphismes 3D – associée au standard **HTML5**) ;
- Création et animation de scènes 3D dans le **navigateur**.

Conventions

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



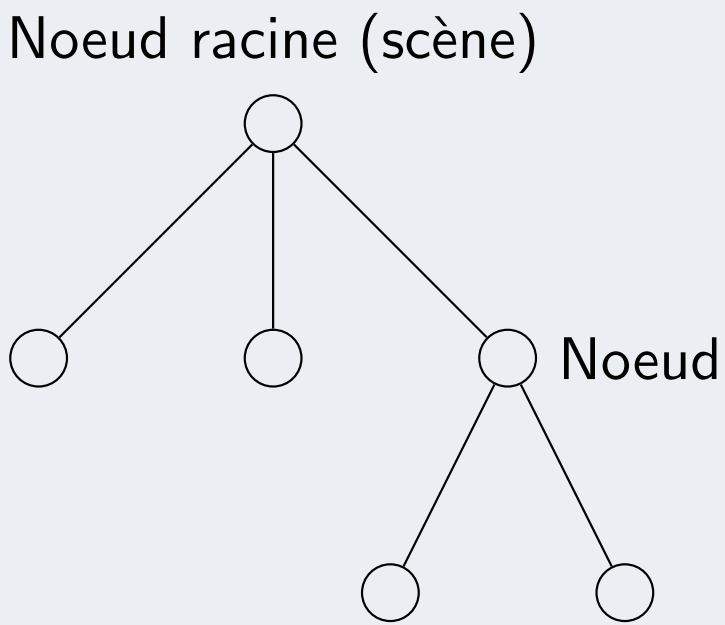
Graphe de scène (1/2)

Définition

Graphe de scène : Organisation hiérarchique des différents éléments composant une scène 3D.

Une structure arborescente

- Représenté par un arbre (graphe orienté acyclique) ;
- Les propriétés appliquées aux ancêtres s'appliquent aussi aux descendants.



Graphe de scène (2/2)

Propriétés des noeuds et champs

- Un noeud possède des propriétés ;
- Dans la pratique, elles sont modifiables ;
- Ces propriétés sont appelées des **champs**.

Examples

Sphere : Rayon ;

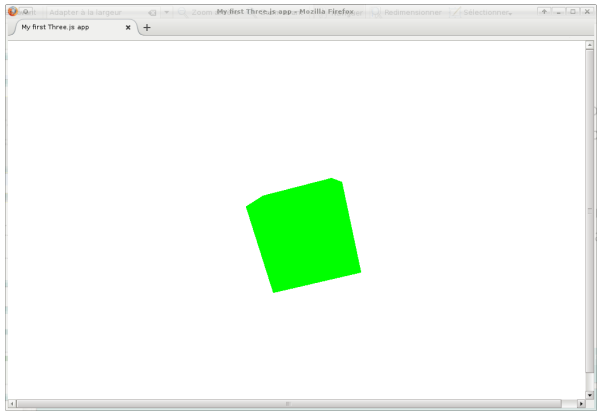
Boite : Largeur, Longeur, Hauteur ;

Transformation : Matrice de transformation.

three.js par l'exemple

Étapes de la création d'une scène

1. Préparer le document html à afficher dans le navigateur ;
2. Créer la scène ;
3. Effectuer le rendu de la scène ;
4. Mettre en place les animations.



Préparation du document HTML

Code du document HTML

```
<html>
  <head>
    <title>Premiere application Three.js</title>
    <style>canvas { width: 100%; height: 100% }</style>
  </head>
  <body>
    <script src="https://rawgit.com/mrdoob/three.js/
        master/build/three.js"></script>
    <script> // Code javascript à mettre ici. </script>
  </body>
</html>
```

Remarque

Le script introduira un *canvas* pour effectuer le rendu.

Créer la scène (1/3)

Instructions mettant en place la scène

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

Commentaires

- Éléments 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.

Créer la scène (2/3)

Ajout d'une géométrie dans la scène

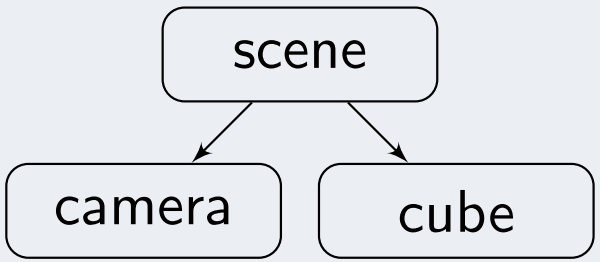
```
var geometry = new THREE.BoxGeometry(1,1,1);
var material = new THREE.MeshBasicMaterial( { color: 0
    x00ff00 } );
var cube = new THREE.Mesh( geometry, material );
scene.add( camera);
scene.add( cube );

camera.position.z = 3;
```

Commentaires

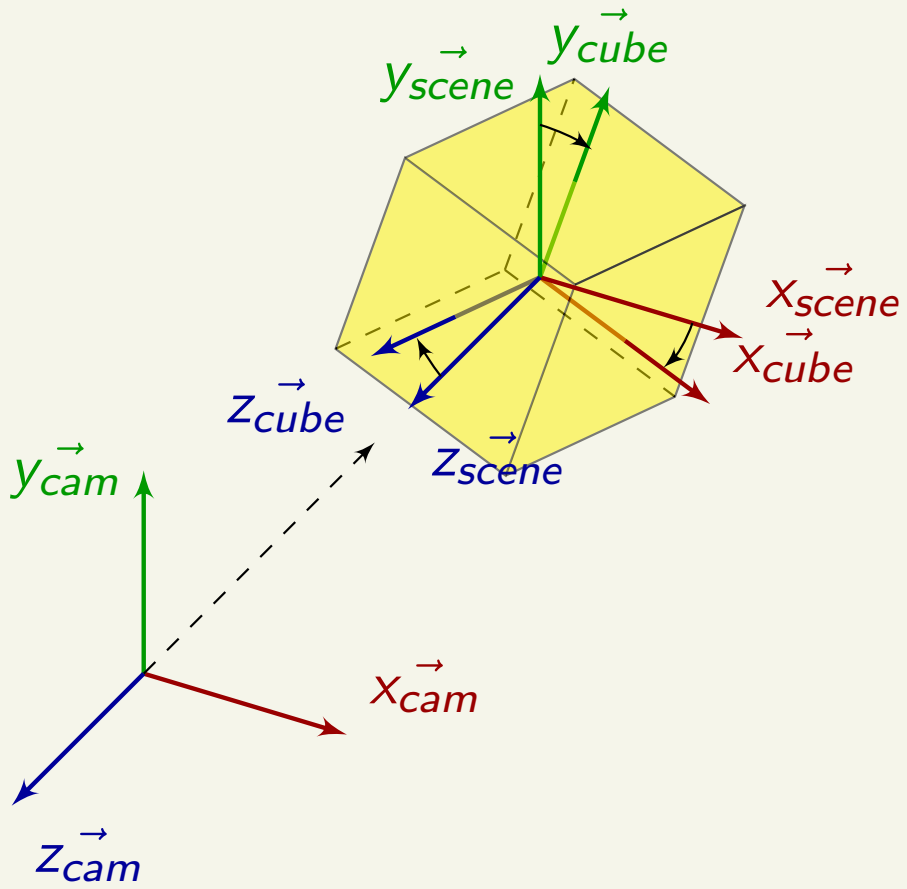
- Création d'un cube avec une géométrie et un matériau ;
- Ajout du cube à la scène ;
- Positionnement de la caméra par rapport à la scène.

Créer la scène (3/3)

Graphe de scène



Repères



Effectuer le rendu

Création de la boucle de rendu

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

Commentaires

- Création de la fonction *render()* :
  - Demande à ce que *render()* soit appelée pour la prochaine image ;
  - Effectue le rendu de la scène avec la caméra donnée dans la zone de rendu ;
- Appelle une première fois la fonction *render()*.

Animer la scène

Animer = redessiner

Pour animer, il faut refaire un rendu de la scène modifiée.

Faire tourner le cube

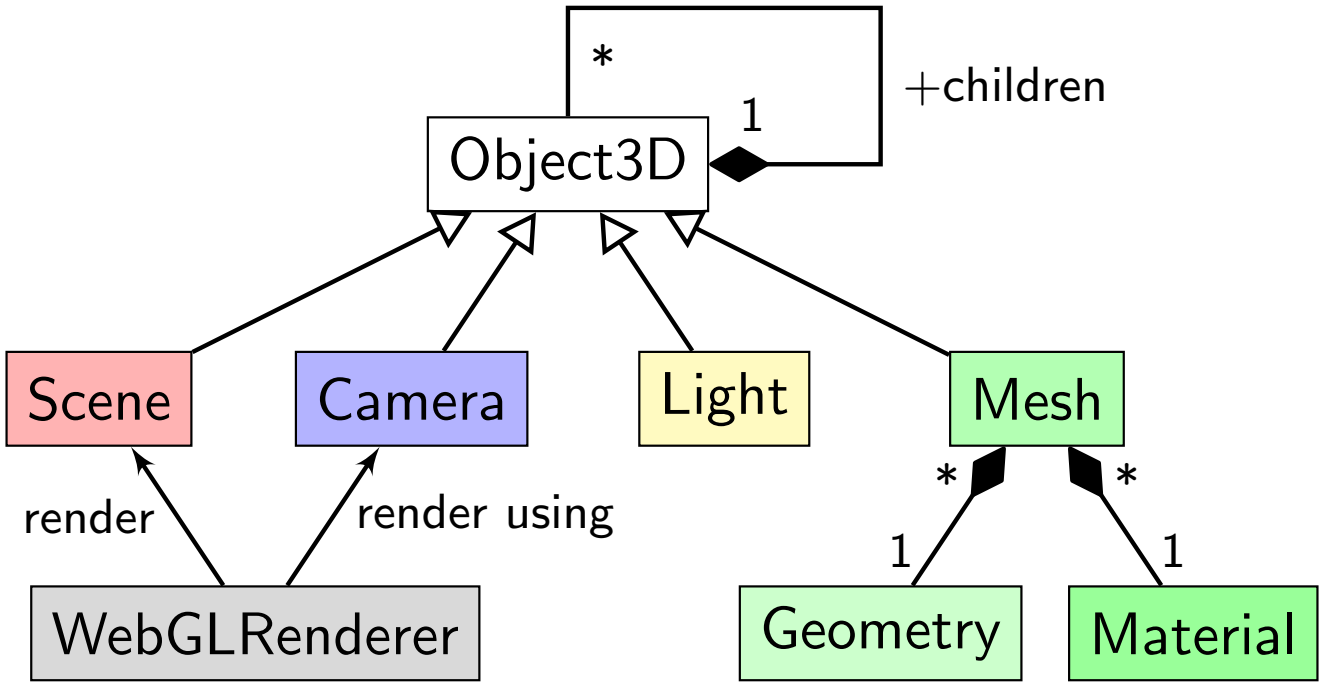
```
// à insérer avant renderer.render(...)
cube.rotation.x += 0.01;
cube.rotation.y += 0.01;
```

Fréquence de rafraîchissement

- Spécifiée à 60 Hz ;
- Mais dépendante de la vitesse à laquelle le navigateur fournit une nouvelle image via `requestAnimationFrame(...)` !

Hiérarchie des prototypes de three.js

Prototypes principaux



Le prototype *Object3D* (1/2)

Description

Prototype de base pour tous les noeuds des graphes de scène Three.js capable de gérer :

- les transformations ;
- les relations père-enfants dans le graphe.

Propriétés importantes

- `parent`, `children` : relations dans le graphe de scène ;
- `position`, `rotation`, `scale`, `matrix` : transformations locales ;
- `id`, `name` : nom et identifiant de l'objet.

Les transformations s'appliquent aussi aux descendants !

Le prototype *Object3D* (2/2)

Ordre d'application des transformations

1. Translation ;
2. Rotation ;
3. Mise à l'échelle.

Méthodes principales

- `add()`, `remove()` : gestion des enfants ;
- `applyMatrix()`, `translate[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.

Le prototype *Camera* (1/4)

Description

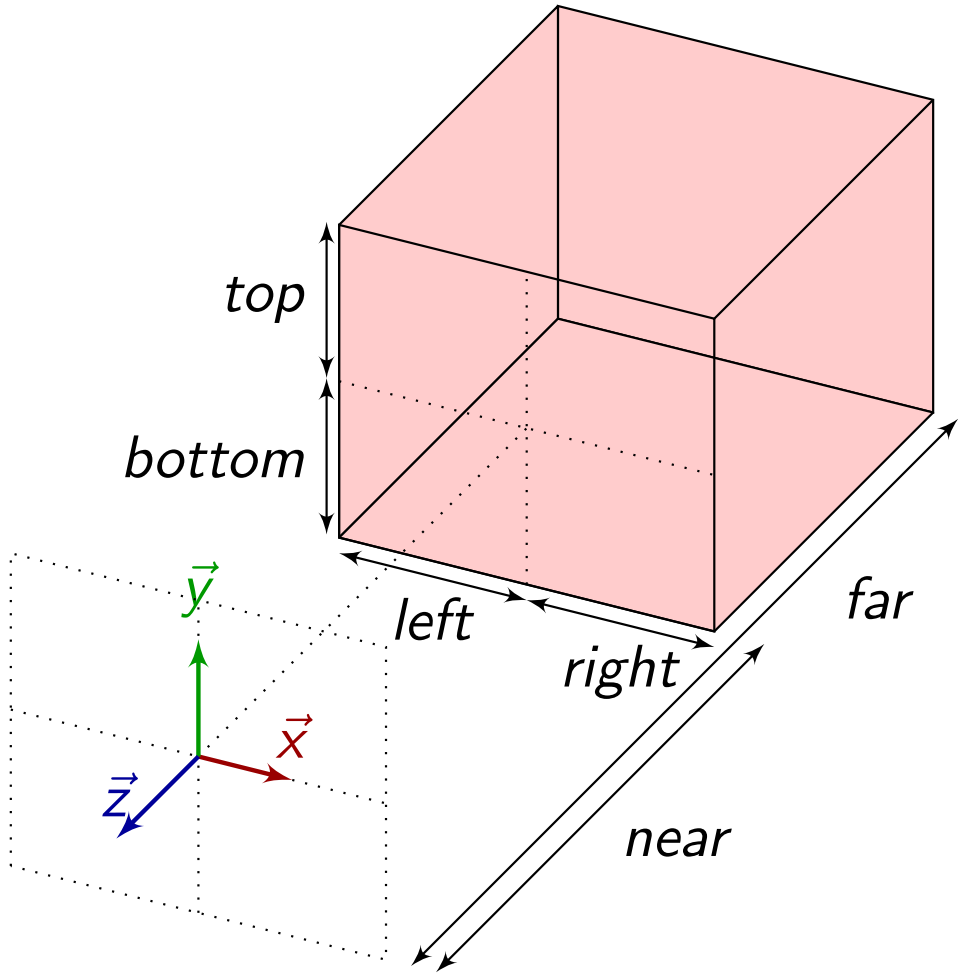
- Prototype décrivant une caméra pouvant être de deux types :
  - `OrthographicCamera` : caméra en perspective cavalière ;
  - `PerspectiveCamera` : caméra en perspective classique.
- Propriété notable : `projectionMatrix`.

Le 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.

Le prototype *Camera* (2/4)

Camera othographique : la zone visualisée est une boîte



Le prototype *Camera* (3/4)

Direction du regard de la caméra

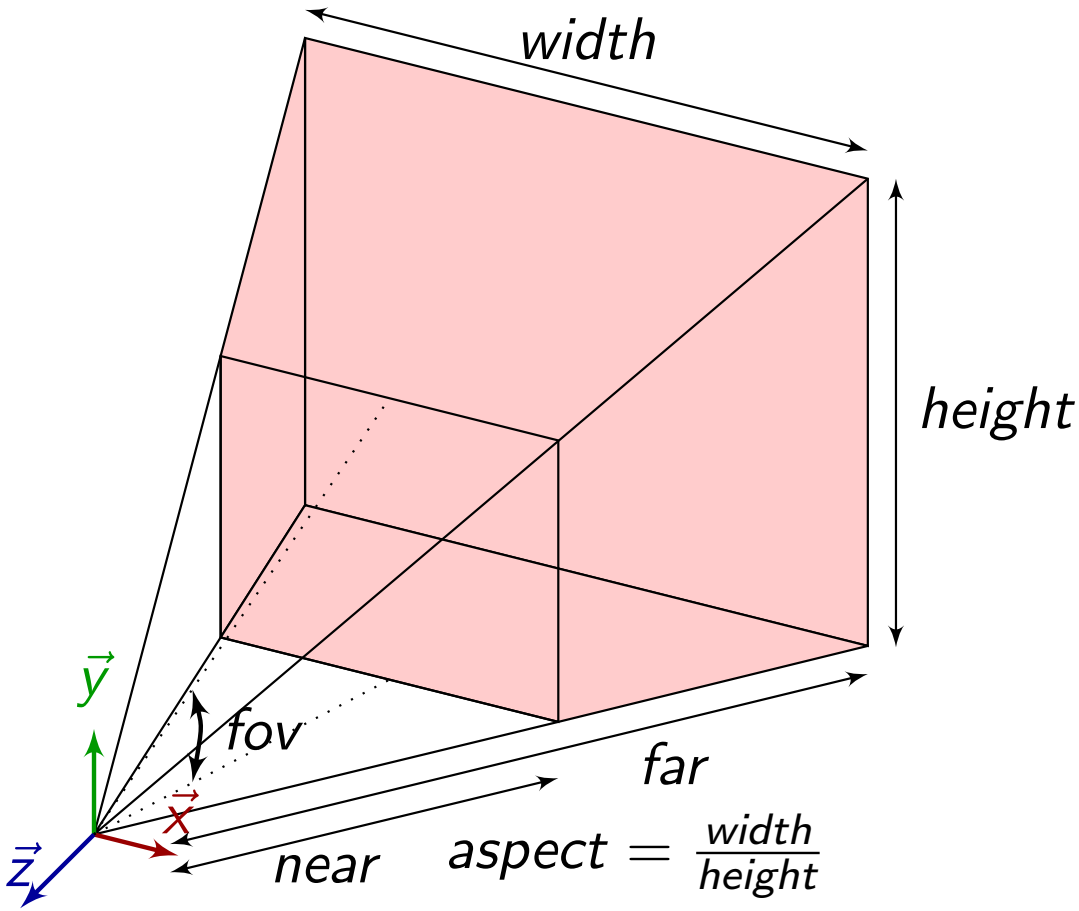
Toujours suivant les z négatifs par rapport à son repère local !

Le 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.

Le prototype *Camera* (4/4)

Camera perspective : la zone visualisée est une pyramide tronquée



Le prototype *Light*

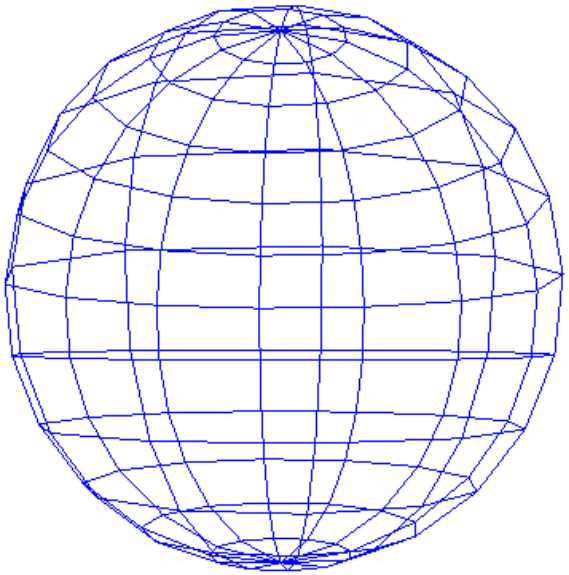
Description

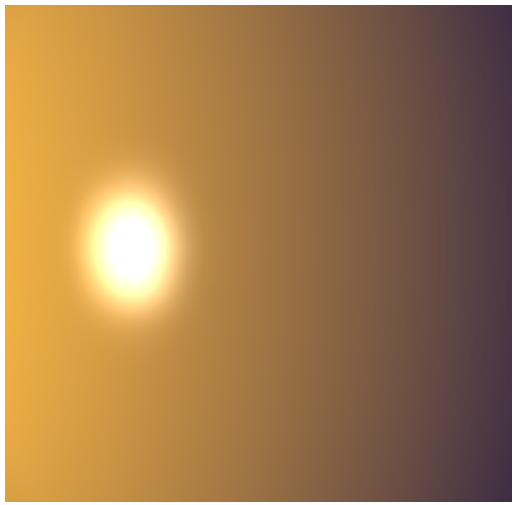
- Prototype décrivant une lumière pouvant être de plusieurs types :
  - `AmbientLight` : lumière ambiante ;
  - `DirectionalLight` : lumière directionnelle ;
  - `PointLight` : lumière ponctuelle ;
  - `SpotLight` : lumière de type spot.
- Propriété notable : `color`.

Le prototype *Mesh*

Description

- Prototype pour décrire une géométrie et son aspect ;
- Constructeur : `Mesh( geometry, material )` :
  - `geometry` : instance de `Geometry` ;
  - `material` : instance de `Material`.
- Propriétés : `geometry`, `material`.





Le prototype *Geometry* (1/3)

Description

Prototype de base contenant toutes les données nécessaires à décrire un modèle 3D.

Propriétés importantes

- `vertices` : tableau de sommets ;
- `colors` : tableau de couleurs par sommets ;
- `faces` : tableau de triangles ;
- `faceVertexUvs` : tableau de coordonnées texture ;
- `boundingBox` : boîte englobante ;
- `boundingSphere` : sphère englobante.

Le prototype *Geometry* (2/3)

Méthodes principales

- `computeFaceNormals()` : calcule les normales par face ;
- `computeVertexNormals()` : normales par sommet ;
- `computeBoundingBox()` : calcule la boite englobante ;
- `computeBoundingSphere()` : calcule la sphère englobante.

Géométries prédéfinies

`BoxGeometry`, `CylinderGeometry`, `IcosahedronGeometry`, `OctahedronGeometry`, `PlaneGeometry`, `SphereGeometry`, `TextGeometry`, etc.

Leur repère local est situé en leur centre !

Le prototype *Geometry* (3/3)

Définir soi-même une géométrie

```
// création d'un triangle
var geometry = new THREE.Geometry();
// liste des sommets
geometry.vertices.push(new THREE.Vector3(-10, 10, 0));
geometry.vertices.push(new THREE.Vector3(-10, -10, 0));
geometry.vertices.push(new THREE.Vector3(10, -10, 0));
// liste des faces, cad des triangles
geometry.faces.push(new THREE.Face3(0, 1, 2));
// opérations sur la géométrie
geometry.computeBoundingSphere();
```

Le prototype *Material*

Description

- Prototype décrivant un matériau (conditionne l'apparence de l'objet) pouvant être de plusieurs types :
  - `MeshBasicMaterial` : aspect « fil de fer »ou « plat » ;
  - `MeshLambertMaterial` : illumination par sommet selon le modèle lambertien ;
  - `MeshPhongMaterial` : illumination par pixel selon le modèle de Phong ;
  - `ShaderMaterial` : utilisation de son propre *shader*.
- Propriétés notables :
  - `opacity` : 0 transparent, 1 opaque ;
  - `transparent` : `true` si transparent (aspect contrôlé par *opacity*).

Ressources

- Site officiel de la bibliothèque de rendu three.js :
  - `http://threejs.org/`
- Visualisation Three.js basée sur WebGL :
  - MSDN (Microsoft)