

# H.T.M.L. et Three.js

L. GARNIER

L2 Info3B, Synthèse d'Images

**1** HTML et Three.js**2** Points et Vecteurs

- Définitions
- Méthodes communes à THREE.Vector $n$ ,  $n \in \{2;3\}$
- Cas du plan i.e.  $n = 2$
- Cas de l'espace i.e.  $n = 3$

**3** La caméra**4** Les lumières

- Lumière ambiante
- Définition et propriétés communes des autres lumières
- Lumière ponctuelle et spot
- Spot et lumière directionnelle : ombre
- Le spot

**5** Les matériaux**6** Les courbes

- Cas classique
- Cas spécifique de THREE.js
- Courbes de Bézier polynomiales

**7** Les surfaces primitives

- Le principe
- Cas particulier des surfaces planes
- Les surfaces non planes

**8** Transformations géométriques de  $\mathcal{E}_3$ 

- Application affine : translation
- Application affine ou vectorielle

**9** L'Animation**10** Création de menu G.U.I.

- Théorie
- Exemple avec un plan

**11** Le C.S.G.

## Librairies Three.js : installation pour les TPs

Récupérer le fichier **libs.tgz** sur

<http://ufrsciencestech.u-bourgogne.fr/licence2/Info3B/SyntheseImage/tp/init2site>

et le placer dans un répertoire, par exemple :

`~/Info3B/SyntheseImages/tp`

Décompresser le fichier **libs.tgz** via la commande

`tar -xzf libs.tgz`

Récupérer le fichier **tpInit.tgz**.

Toujours dans le répertoire **tp**, décompresser le fichier **tpInit.tgz** puis renommer le répertoire **tplnint** en les répertoires **tp1** , **tp2**, **tp3** et **tp4**.

Dans chacun de ces répertoires **tp1** , **tp2**, **tp3** et **tp4**, vous aurez les répertoires **html**, **css**, **js**.

## L'entête du fichier 00init.html

La syntaxe \*/ représente le chemin idoine.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="fr">

<head>
    <meta charset="utf-8">
    <title>Init </title>
    <script type="text/javascript" charset="UTF-8"
           src = "*/libs/three/controls/TrackballControls.js">
    </script>
    <script type="text/javascript" charset="UTF-8"
           src = "*/libs/util/dat.gui.js">
    </script>
    <script type="text/javascript" src="*/libs/util/dat.gui.js"></script>
    <script type="text/javascript" src="*/libs/util/Stats.js"></script>
    <script type="text/javascript" src="*/libs/util/util.js"></script>
    <script type="text/javascript" src="../js/00init.js"></script>
    <link rel="stylesheet" href="../css/00init.css">
</head>
```

Complément sur le fichier `00init.html`

```
<body onload="init()">
  <!-- creation de la zone pour le rendu de la scene 3D (Webgl) -->
  <div id="webgl"></div>
  <!-- pour afficher le debogage eventuel -->
  <div id="result"></div>
</body>
</html>
```

Définition (Fonction `requestAnimationFrame()`)

La fonction `requestAnimationFrame()` est proche de la fonction `setInterval()`, mais présente plusieurs avantages :

- le navigateur optimise les traitements pour garantir un rafraîchissement régulier ;
- afin de réduire la consommation CPU, le navigateur interrompt l'animation lorsque l'onglet n'est plus visible ;
- l'animation est plus fluide.

## Le fichier 00init.js

```
function init(){
    var stats = initStats();
    // creation de rendu et de la taille
    let rendu = new THREE.WebGLRenderer({antialias:true});
    // creation de la scene ou mettre les objets
    let scene = new THREE.Scene();
    // autoriser les ombres, defaut : false
    rendu.shadowMap.enabled = true;
    // pour avoir un fond blanc
    rendu.setClearColor(new THREE.Color(0xFFFFFF));
    // taille de webgl dans le fichier html
    rendu.setSize(window.innerWidth*.9, window.innerHeight*.9);
    // code a remplir en fonction de la scene dont la camera
    // appel de la fonction pour animer la scene
    renduAnim();
    // ajoute le rendu dans l'element HTML
    document.getElementById("webgl").appendChild(rendu.domElement);
    // affichage de la scene
    rendu.render(scene, camera);
    function renduAnim() {...}
} // fin fonction init()
```

## Le fichier 00init.js : La fonction renduAnim()

```
function renduAnim() {
    stats.update();
    // animation en utilisant requestAnimationFrame
    // appel de la fonction renduAnim sans les parentheses
    requestAnimationFrame(renduAnim);
    // ajoute le rendu dans l'element HTML
    // la camera sera definie plus tard
    rendu.render(scene, camera);
} // fin fonction renduAnim()
```

- 1 HTML et Three.js
- 2 Points et Vecteurs
  - Définitions
  - Méthodes communes à THREE.Vector $n$ ,  $n \in \{2;3\}$
  - Cas du plan i.e.  $n = 2$
  - Cas de l'espace i.e.  $n = 3$
- 3 La caméra
- 4 Les lumières
  - Lumière ambiante
  - Définition et propriétés communes des autres lumières
  - Lumière ponctuelle et spot
  - Spot et lumière directionnelle : ombre
  - Le spot
- 5 Les matériaux
- 6 Les courbes
  - Cas classique
  - Cas spécifique de THREE.js
  - Courbes de Bézier polynomiales
- 7 Les surfaces primitives
  - Le principe
  - Cas particulier des surfaces planes
  - Les surfaces non planes
- 8 Transformations géométriques de  $\mathcal{E}_3$ 
  - Application affine : translation
  - Application affine ou vectorielle
- 9 L'Animation
- 10 Création de menu G.U.I.
  - Théorie
  - Exemple avec un plan
- 11 Le C.S.G.

## 2 Points et Vecteurs

- Définitions

- Méthodes communes à THREE.Vector $n$ ,  $n \in \{2; 3\}$
- Cas du plan i.e.  $n = 2$
- Cas de l'espace i.e.  $n = 3$

## Définitions d'un point, d'un vecteur

## Définition (Définition d'un point)

Soit  $A(x_A; y_A)$  un point du plan affine  $\mathcal{P}$  et  $B(x_B; y_B; z_B)$  un point de l'espace affine  $\mathcal{E}_3$ .

## Définitions d'un point, d'un vecteur

## Définition (Définition d'un point)

Soit  $A(x_A; y_A)$  un point du plan affine  $\mathcal{P}$  et  $B(x_B; y_B; z_B)$  un point de l'espace affine  $\mathcal{E}_3$ .

- `let A = new THREE.Vector2(xA, yA);`

## Définitions d'un point, d'un vecteur

## Définition (Définition d'un point)

Soit  $A(x_A; y_A)$  un point du plan affine  $\mathcal{P}$  et  $B(x_B; y_B; z_B)$  un point de l'espace affine  $\mathcal{E}_3$ .

- `let A = new THREE.Vector2 (xA,yA);`
- `let B = new THREE.Vector3 (xB,yB,zB);`

## Définitions d'un point, d'un vecteur

## Définition (Définition d'un point)

Soit  $A(x_A; y_A)$  un point du plan affine  $\mathcal{P}$  et  $B(x_B; y_B; z_B)$  un point de l'espace affine  $\mathcal{E}_3$ .

- `let A = new THREE.Vector2(xA, yA);`
- `let B = new THREE.Vector3(xB, yB, zB);`

## Définition (Définition d'un vecteur)

Soit  $\vec{u}(x_0; y_0)$  un vecteur du plan vectoriel  $\vec{\mathcal{P}}$  et  $\vec{v}(x_1; y_1; z_1)$  un vecteur de l'espace vectoriel  $\vec{\mathcal{E}}_3$ .

## Définitions d'un point, d'un vecteur

## Définition (Définition d'un point)

Soit  $A(x_A; y_A)$  un point du plan affine  $\mathcal{P}$  et  $B(x_B; y_B; z_B)$  un point de l'espace affine  $\mathcal{E}_3$ .

- `let A = new THREE.Vector2(xA, yA);`
- `let B = new THREE.Vector3(xB, yB, zB);`

## Définition (Définition d'un vecteur)

Soit  $\vec{u}(x_0; y_0)$  un vecteur du plan vectoriel  $\vec{\mathcal{P}}$  et  $\vec{v}(x_1; y_1; z_1)$  un vecteur de l'espace vectoriel  $\vec{\mathcal{E}}_3$ .

Leurs définitions sont :

- `let u = new THREE.Vector2(x0, y0);`

## Définitions d'un point, d'un vecteur

### Définition (Définition d'un point)

Soit  $A(x_A; y_A)$  un point du plan affine  $\mathcal{P}$  et  $B(x_B; y_B; z_B)$  un point de l'espace affine  $\mathcal{E}_3$ .

- `let A = new THREE.Vector2(xA, yA);`
- `let B = new THREE.Vector3(xB, yB, zB);`

### Définition (Définition d'un vecteur)

Soit  $\vec{u}(x_0; y_0)$  un vecteur du plan vectoriel  $\vec{\mathcal{P}}$  et  $\vec{v}(x_1; y_1; z_1)$  un vecteur de l'espace vectoriel  $\vec{\mathcal{E}}_3$ .

Leurs définitions sont :

- `let u = new THREE.Vector2(x0, y0);`
- `let v = new THREE.Vector3(x1, y1, z1);`

## Définitions d'un point, d'un vecteur

### Définition (Définition d'un point)

Soit  $A(x_A; y_A)$  un point du plan affine  $\mathcal{P}$  et  $B(x_B; y_B; z_B)$  un point de l'espace affine  $\mathcal{E}_3$ .

- `let A = new THREE.Vector2(xA, yA);`
- `let B = new THREE.Vector3(xB, yB, zB);`

### Définition (Définition d'un vecteur)

Soit  $\vec{u}(x_0; y_0)$  un vecteur du plan vectoriel  $\vec{\mathcal{P}}$  et  $\vec{v}(x_1; y_1; z_1)$  un vecteur de l'espace vectoriel  $\vec{\mathcal{E}}_3$ .

Leurs définitions sont :

- `let u = new THREE.Vector2(x0, y0);`
- `let v = new THREE.Vector3(x1, y1, z1);`

### Danger

Ne pas confondre point et vecteur même si les définitions sont identiques.

Accès en lecture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2; 3\}$ 

Définition (Accès aux coordonnées ou composantes en lecture)

Soit *Toto* une instance de `Vector2` ou `Vector3`.

Accès en lecture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$ 

## Définition (Accès aux coordonnées ou composantes en lecture)

Soit *Toto* une instance de `Vector2` ou `Vector3`.

- L'abscisse de *Toto* s'obtient par `Toto.x`;

Accès en lecture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$ 

Définition (Accès aux coordonnées ou composantes en lecture)

Soit *Toto* une instance de `Vector2` ou `Vector3`.

- L'abscisse de *Toto* s'obtient par `Toto.x`;
- L'ordonnée de *Toto* s'obtient par `Toto.y`;

Accès en lecture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$ 

## Définition (Accès aux coordonnées ou composantes en lecture)

Soit *Toto* une instance de `Vector2` ou `Vector3`.

- L'abscisse de *Toto* s'obtient par `Toto.x`;
- L'ordonnée de *Toto* s'obtient par `Toto.y`;
- Si  $n = 3$ , la cote de *Toto* s'obtient par `Toto.z`;

Accès en lecture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$ 

## Définition (Accès aux coordonnées ou composantes en lecture)

Soit *Toto* une instance de `Vector2` ou `Vector3`.

- L'abscisse de *Toto* s'obtient par `Toto.x`;
- L'ordonnée de *Toto* s'obtient par `Toto.y`;
- Si  $n = 3$ , la cote de *Toto* s'obtient par `Toto.z`;
- Il est possible d'appeler la méthode `getComponent(i)` où *i* vaut :

Accès en lecture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$ 

## Définition (Accès aux coordonnées ou composantes en lecture)

Soit *Toto* une instance de `Vector2` ou `Vector3`.

- L'abscisse de *Toto* s'obtient par `Toto.x`;
- L'ordonnée de *Toto* s'obtient par `Toto.y`;
- Si  $n = 3$ , la cote de *Toto* s'obtient par `Toto.z`;
- Il est possible d'appeler la méthode `getComponent(i)` où *i* vaut :
  - ▶ 0 pour l'abscisse ;

Accès en lecture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$ 

## Définition (Accès aux coordonnées ou composantes en lecture)

Soit *Toto* une instance de `Vector2` ou `Vector3`.

- L'abscisse de *Toto* s'obtient par `Toto.x`;
- L'ordonnée de *Toto* s'obtient par `Toto.y`;
- Si  $n = 3$ , la cote de *Toto* s'obtient par `Toto.z`;
- Il est possible d'appeler la méthode `getComponent(i)` où *i* vaut :
  - ▶ 0 pour l'abscisse ;
  - ▶ 1 pour l'ordonnée ;

Accès en lecture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$ 

## Définition (Accès aux coordonnées ou composantes en lecture)

Soit *Toto* une instance de `Vector2` ou `Vector3`.

- L'abscisse de *Toto* s'obtient par `Toto.x`;
- L'ordonnée de *Toto* s'obtient par `Toto.y`;
- Si  $n = 3$ , la cote de *Toto* s'obtient par `Toto.z`;
- Il est possible d'appeler la méthode `getComponent(i)` où *i* vaut :
  - ▶ 0 pour l'abscisse ;
  - ▶ 1 pour l'ordonnée ;
  - ▶ 2 pour la cote.

Accès en lecture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$ 

## Définition (Accès aux coordonnées ou composantes en lecture)

Soit *Toto* une instance de `Vector2` ou `Vector3`.

- L'abscisse de *Toto* s'obtient par `Toto.x`;
- L'ordonnée de *Toto* s'obtient par `Toto.y`;
- Si  $n = 3$ , la cote de *Toto* s'obtient par `Toto.z`;
- Il est possible d'appeler la méthode `getComponent(i)` où *i* vaut :
  - 0 pour l'abscisse ;
  - 1 pour l'ordonnée ;
  - 2 pour la cote.

```
let w = new THREE.Vector3(1,2,-3);
```

Accès en lecture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$ 

## Définition (Accès aux coordonnées ou composantes en lecture)

Soit *Toto* une instance de `Vector2` ou `Vector3`.

- L'abscisse de *Toto* s'obtient par `Toto.x`;
- L'ordonnée de *Toto* s'obtient par `Toto.y`;
- Si  $n = 3$ , la cote de *Toto* s'obtient par `Toto.z`;
- Il est possible d'appeler la méthode `getComponent(i)` où *i* vaut :
  - 0 pour l'abscisse ;
  - 1 pour l'ordonnée ;
  - 2 pour la cote.

```
let w = new THREE.Vector3(1,2,-3);
let mes = "";
```

Accès en lecture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$ 

## Définition (Accès aux coordonnées ou composantes en lecture)

Soit *Toto* une instance de *Vector2* ou *Vector3*.

- L'abscisse de *Toto* s'obtient par *Toto.x*;
- L'ordonnée de *Toto* s'obtient par *Toto.y*;
- Si  $n = 3$ , la cote de *Toto* s'obtient par *Toto.z*;
- Il est possible d'appeler la méthode *getComponent(i)* où *i* vaut :
  - 0 pour l'abscisse ;
  - 1 pour l'ordonnée ;
  - 2 pour la cote.

```
let w = new THREE.Vector3(1,2,-3);
let mes = "";
for (let i = 0; i < 3;i++)
  mes += "<br /> composante i : "+w.getComponent(i);
```

Accès en lecture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$ 

## Définition (Accès aux coordonnées ou composantes en lecture)

Soit *Toto* une instance de *Vector2* ou *Vector3*.

- L'abscisse de *Toto* s'obtient par *Toto.x*;
- L'ordonnée de *Toto* s'obtient par *Toto.y*;
- Si  $n = 3$ , la cote de *Toto* s'obtient par *Toto.z*;
- Il est possible d'appeler la méthode *getComponent(i)* où *i* vaut :
  - 0 pour l'abscisse ;
  - 1 pour l'ordonnée ;
  - 2 pour la cote.

```
let w = new THREE.Vector3(1,2,-3);
let mes = "";
for (let i = 0; i < 3;i++)
    mes += "<br /> composante i : "+w.getComponent(i);
document.getElementById("result").innerHTML += mes;
```

Accès en écriture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$

## Définition (Accès aux coordonnées ou composantes en écriture)

*Soit Toto une instance de Vector2 ou de Vector3. Soit k<sub>0</sub> un réel .*

Accès en écriture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$

### Définition (Accès aux coordonnées ou composantes en écriture)

*Soit Toto une instance de Vector2 ou de Vector3. Soit  $k_0$  un réel .*

- *Attribuer la valeur  $k_0$  à l'abscisse de Toto s'effectue par Toto.setX ( $k_0$ );*

Accès en écriture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$ 

## Définition (Accès aux coordonnées ou composantes en écriture)

Soit *Toto* une instance de *Vector2* ou de *Vector3*. Soit  $k_0$  un réel .

- Attribuer la valeur  $k_0$  à l'abscisse de *Toto* s'effectue par *Toto.setX* ( $k_0$ ) ;
- Attribuer la valeur  $k_0$  à l'ordonnée de *Toto* s'effectue par *Toto.setY* ( $k_0$ ) ;

Accès en écriture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$ 

## Définition (Accès aux coordonnées ou composantes en écriture)

Soit *Toto* une instance de *Vector2* ou de *Vector3*. Soit  $k_0$  un réel .

- Attribuer la valeur  $k_0$  à l'abscisse de *Toto* s'effectue par *Toto.setX* ( $k_0$ );
- Attribuer la valeur  $k_0$  à l'ordonnée de *Toto* s'effectue par *Toto.setY* ( $k_0$ );
- Si  $n = 3$ , attribuer la valeur  $k_0$  à la cotée de *Toto* s'effectue par  
*Toto.setZ* ( $k_0$ );

Accès en écriture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$ 

## Définition (Accès aux coordonnées ou composantes en écriture)

Soit *Toto* une instance de *Vector2* ou de *Vector3*. Soit  $k_0$  un réel .

- Attribuer la valeur  $k_0$  à l'abscisse de *Toto* s'effectue par *Toto.setX* ( $k_0$ ) ;
- Attribuer la valeur  $k_0$  à l'ordonnée de *Toto* s'effectue par *Toto.setY* ( $k_0$ ) ;
- Si  $n = 3$ , attribuer la valeur  $k_0$  à la cotée de *Toto* s'effectue par *Toto.setZ* ( $k_0$ ) ;
- Il est possible d'appeler la méthode *Toto.setComponent*( $i, k_0$ ) ; où  $i$  vaut :

Accès en écriture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$ 

## Définition (Accès aux coordonnées ou composantes en écriture)

Soit *Toto* une instance de *Vector2* ou de *Vector3*. Soit  $k_0$  un réel .

- Attribuer la valeur  $k_0$  à l'abscisse de *Toto* s'effectue par *Toto.setX* ( $k_0$ ) ;
- Attribuer la valeur  $k_0$  à l'ordonnée de *Toto* s'effectue par *Toto.setY* ( $k_0$ ) ;
- Si  $n = 3$ , attribuer la valeur  $k_0$  à la cotée de *Toto* s'effectue par *Toto.setZ* ( $k_0$ ) ;
- Il est possible d'appeler la méthode *Toto.setComponent*( $i, k_0$ ) ; où  $i$  vaut :
  - ▶ 0 pour l'abscisse ;

Accès en écriture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$ 

## Définition (Accès aux coordonnées ou composantes en écriture)

Soit *Toto* une instance de *Vector2* ou de *Vector3*. Soit  $k_0$  un réel .

- Attribuer la valeur  $k_0$  à l'abscisse de *Toto* s'effectue par *Toto.setX* ( $k_0$ ) ;
- Attribuer la valeur  $k_0$  à l'ordonnée de *Toto* s'effectue par *Toto.setY* ( $k_0$ ) ;
- Si  $n = 3$ , attribuer la valeur  $k_0$  à la cotée de *Toto* s'effectue par *Toto.setZ* ( $k_0$ ) ;
- Il est possible d'appeler la méthode *Toto.setComponent*( $i, k_0$ ) ; où  $i$  vaut :
  - ▶ 0 pour l'abscisse ;
  - ▶ 1 pour l'ordonnée ;

Accès en écriture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$ 

## Définition (Accès aux coordonnées ou composantes en écriture)

Soit *Toto* une instance de *Vector2* ou de *Vector3*. Soit  $k_0$  un réel .

- Attribuer la valeur  $k_0$  à l'abscisse de *Toto* s'effectue par *Toto.setX* ( $k_0$ ) ;
- Attribuer la valeur  $k_0$  à l'ordonnée de *Toto* s'effectue par *Toto.setY* ( $k_0$ ) ;
- Si  $n = 3$ , attribuer la valeur  $k_0$  à la cotee de *Toto* s'effectue par *Toto.setZ* ( $k_0$ ) ;
- Il est possible d'appeler la méthode *Toto.setComponent*( $i, k_0$ ) ; où  $i$  vaut :
  - ▶ 0 pour l'abscisse ;
  - ▶ 1 pour l'ordonnée ;
  - ▶ 2 pour la cote.

Accès en écriture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$ 

## Définition (Accès aux coordonnées ou composantes en écriture)

Soit *Toto* une instance de *Vector2* ou de *Vector3*. Soit  $k_0$  un réel .

- Attribuer la valeur  $k_0$  à l'abscisse de *Toto* s'effectue par *Toto.setX* ( $k_0$ ) ;
- Attribuer la valeur  $k_0$  à l'ordonnée de *Toto* s'effectue par *Toto.setY* ( $k_0$ ) ;
- Si  $n = 3$ , attribuer la valeur  $k_0$  à la cotee de *Toto* s'effectue par *Toto.setZ* ( $k_0$ ) ;
- Il est possible d'appeler la méthode *Toto.setComponent*( $i, k_0$ ) ; où  $i$  vaut :
  - ▶ 0 pour l'abscisse ;
  - ▶ 1 pour l'ordonnée ;
  - ▶ 2 pour la cote.

```
let w = new THREE.Vector3(0,0,0);
```

Accès en écriture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$

## Définition (Accès aux coordonnées ou composantes en écriture)

Soit *Toto* une instance de *Vector2* ou de *Vector3*. Soit  $k_0$  un réel .

- Attribuer la valeur  $k_0$  à l'abscisse de *Toto* s'effectue par *Toto.setX* ( $k_0$ ) ;
- Attribuer la valeur  $k_0$  à l'ordonnée de *Toto* s'effectue par *Toto.setY* ( $k_0$ ) ;
- Si  $n = 3$ , attribuer la valeur  $k_0$  à la cotee de *Toto* s'effectue par *Toto.setZ* ( $k_0$ ) ;
- Il est possible d'appeler la méthode *Toto.setComponent*( $i, k_0$ ) ; où  $i$  vaut :
  - ▶ 0 pour l'abscisse ;
  - ▶ 1 pour l'ordonnée ;
  - ▶ 2 pour la cote.

```
let w = new THREE.Vector3(0,0,0);
for (let i = 0; i < 3;i++)
    w.setComponent(i,Math.sqrt(i));
```

Accès en écriture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$

### Définition (Accès aux coordonnées ou composantes en écriture)

Soit *Toto* une instance de Vector2 ou de Vector3. Soit  $k_0$  un réel .

- Attribuer la valeur  $k_0$  à l'abscisse de *Toto* s'effectue par *Toto.setX*( $k_0$ );
- Attribuer la valeur  $k_0$  à l'ordonnée de *Toto* s'effectue par *Toto.setY*( $k_0$ );
- Si  $n = 3$ , attribuer la valeur  $k_0$  à la cotee de *Toto* s'effectue par *Toto.setZ*( $k_0$ );
- Il est possible d'appeler la méthode *Toto.setComponent*( $i, k_0$ ); où  $i$  vaut :
  - ▶ 0 pour l'abscisse ;
  - ▶ 1 pour l'ordonnée ;
  - ▶ 2 pour la cote.

```
let w = new THREE.Vector3(0,0,0);
for (let i = 0; i < 3;i++)
  w.setComponent(i,Math.sqrt(i));
```

### Définition (Accès à toutes les coordonnées ou composantes en écriture)

Soit *Toto2* (resp. *Toto3*) une instance de Vector2 (resp. Vector3). Soit  $x_0$ ,  $y_0$  et  $z_0$  trois réels.

Accès en écriture aux champs d'une instance de THREE.Vector $n$ ,  $n \in \{2;3\}$

### Définition (Accès aux coordonnées ou composantes en écriture)

Soit *Toto* une instance de *Vector2* ou de *Vector3*. Soit  $k_0$  un réel .

- Attribuer la valeur  $k_0$  à l'abscisse de *Toto* s'effectue par *Toto.setX* ( $k_0$ ) ;
- Attribuer la valeur  $k_0$  à l'ordonnée de *Toto* s'effectue par *Toto.setY* ( $k_0$ ) ;
- Si  $n = 3$ , attribuer la valeur  $k_0$  à la cotee de *Toto* s'effectue par *Toto.setZ* ( $k_0$ ) ;
- Il est possible d'appeler la méthode *Toto.setComponent*( $i, k_0$ ) ; où  $i$  vaut :
  - ▶ 0 pour l'abscisse ;
  - ▶ 1 pour l'ordonnée ;
  - ▶ 2 pour la cote.

```
let w = new THREE.Vector3(0,0,0);
for (let i = 0; i < 3;i++)
  w.setComponent(i,Math.sqrt(i));
```

### Définition (Accès à toutes les coordonnées ou composantes en écriture)

Soit *Toto2* (resp. *Toto3*) une instance de *Vector2* (resp. *Vector3*). Soit  $x_0$ ,  $y_0$  et  $z_0$  trois réels.

Nous pouvons utiliser *Toto2.set* ( $x_0, y_0$ ) ;  
et *Toto3.set* ( $x_0, y_0, z_0$ ) ;

Normalisation d'un vecteur de  $\mathcal{P}$  ou de  $\mathcal{E}_3$ Définition (Méthode `normalize`)

Soit  $\underline{u}$  une instance de `Vector $n$` ,  $n \in \{2; 3\}$ , représentant le vecteur  $\vec{u}$  de  $\mathcal{P}$  ou de  $\mathcal{E}_3$ .

Normalisation d'un vecteur de  $\mathcal{P}$  ou de  $\mathcal{E}_3$ Définition (Méthode `normalize`)

Soit `u` une instance de `Vector $n$` ,  $n \in \{2; 3\}$ , représentant le vecteur  $\vec{u}$  de  $\mathcal{P}$  ou de  $\mathcal{E}_3$ . La syntaxe `u.normalize();` permet, en écrasant la valeur initiale de `u`, d'affecter à l'instance le vecteur, unitaire, de même direction et de même sens que  $\vec{u}$  c'est-à-dire que nous faisons l'opération :

$$\vec{u} \leftarrow \frac{1}{\|\vec{u}\|} \vec{u}$$

Normalisation d'un vecteur de  $\mathcal{P}$  ou de  $\mathcal{E}_3$ Définition (Méthode `normalize`)

Soit `u` une instance de `Vector $n$` ,  $n \in \{2; 3\}$ , représentant le vecteur  $\vec{u}$  de  $\mathcal{P}$  ou de  $\mathcal{E}_3$ . La syntaxe `u.normalize();` permet, en écrasant la valeur initiale de `u`, d'affecter à l'instance le vecteur, unitaire, de même direction et de même sens que  $\vec{u}$  c'est-à-dire que nous faisons l'opération :

$$\vec{u} \leftarrow \frac{1}{\|\vec{u}\|} \vec{u}$$

Définition (Méthode `distanceTo`)

Soit `A` et `B` deux instances de `Vector $n$` ,  $n \in \{2; 3\}$ , représentant respectivement les points  $A$  et  $B$  de  $\mathcal{P}$  ou de  $\mathcal{E}_3$ .

Normalisation d'un vecteur de  $\mathcal{P}$  ou de  $\mathcal{E}_3$ Définition (Méthode `normalize`)

Soit `u` une instance de `Vector $n$` ,  $n \in \{2;3\}$ , représentant le vecteur  $\vec{u}$  de  $\mathcal{P}$  ou de  $\mathcal{E}_3$ . La syntaxe `u.normalize();` permet, en écrasant la valeur initiale de `u`, d'affecter à l'instance le vecteur, unitaire, de même direction et de même sens que  $\vec{u}$  c'est-à-dire que nous faisons l'opération :

$$\vec{u} \leftarrow \frac{1}{\|\vec{u}\|} \vec{u}$$

Définition (Méthode `distanceTo`)

Soit `A` et `B` deux instances de `Vector $n$` ,  $n \in \{2;3\}$ , représentant respectivement les points  $A$  et  $B$  de  $\mathcal{P}$  ou de  $\mathcal{E}_3$ .

La syntaxe `A.distanceTo(B);` renvoie la longueur  $AB$  du segment  $[AB]$ .

Normalisation d'un vecteur de  $\mathcal{P}$  ou de  $\mathcal{E}_3$ Définition (Méthode `normalize`)

Soit `u` une instance de `Vector $n$` ,  $n \in \{2;3\}$ , représentant le vecteur  $\vec{u}$  de  $\mathcal{P}$  ou de  $\mathcal{E}_3$ . La syntaxe `u.normalize();` permet, en écrasant la valeur initiale de `u`, d'affecter à l'instance le vecteur, unitaire, de même direction et de même sens que  $\vec{u}$  c'est-à-dire que nous faisons l'opération :

$$\vec{u} \leftarrow \frac{1}{\|\vec{u}\|} \vec{u}$$

Définition (Méthode `distanceTo`)

Soit `A` et `B` deux instances de `Vector $n$` ,  $n \in \{2;3\}$ , représentant respectivement les points  $A$  et  $B$  de  $\mathcal{P}$  ou de  $\mathcal{E}_3$ .

La syntaxe `A.distanceTo(B);` renvoie la longueur  $AB$  du segment  $[AB]$ .

## Danger

Ne pas confondre la longueur  $AB$  du segment  $[AB]$  avec le vecteur  $\vec{AB}$  dans le code. Solution : coder le vecteur  $\vec{AB}$  par `vecAB` ou `vAB`.

Affichage d'un point de  $\mathcal{E}_3$  (explication ultérieure concernant l'affichage de sphères)

```
function tracePt(MaScene, P, CoulHexa, dimPt, bol){
```

Affichage d'un point de  $\mathcal{E}_3$  (explication ultérieure concernant l'affichage de sphères)

```
function tracePt(MaScene, P, CoulHexa, dimPt, bol){  
let sphereGeometry = new THREE.SphereGeometry(dimPt,12,24);
```

Affichage d'un point de  $\mathcal{E}_3$  (explication ultérieure concernant l'affichage de sphères)

```
function tracePt(MaScene, P, CoulHexa, dimPt, bol){  
    let sphereGeometry = new THREE.SphereGeometry(dimPt, 12, 24);  
    let sphereMaterial = new THREE.MeshBasicMaterial({  
        color: CoulHexa });
```

Affichage d'un point de  $\mathcal{E}_3$  (explication ultérieure concernant l'affichage de sphères)

```
function tracePt(MaScene, P, CoulHexa, dimPt, bol){  
    let sphereGeometry = new THREE.SphereGeometry(dimPt,12,24);  
    let sphereMaterial = new THREE.MeshBasicMaterial({  
        color: CoulHexa });  
    let sphere = new THREE.Mesh(sphereGeometry, sphereMaterial);  
}
```

Affichage d'un point de  $\mathcal{E}_3$  (explication ultérieure concernant l'affichage de sphères)

```
function tracePt(MaScene, P, CoulHexa, dimPt, bol){  
    let sphereGeometry = new THREE.SphereGeometry(dimPt,12,24);  
    let sphereMaterial = new THREE.MeshBasicMaterial({  
        color: CoulHexa });  
    let sphere = new THREE.Mesh(sphereGeometry, sphereMaterial);  
    sphere.position.set(P.x,P.y,P.z);  
}
```

Affichage d'un point de  $\mathcal{E}_3$  (explication ultérieure concernant l'affichage de sphères)

```
function tracePt(MaScene, P, CoulHexa, dimPt, bol){  
    let sphereGeometry = new THREE.SphereGeometry(dimPt, 12, 24);  
    let sphereMaterial = new THREE.MeshBasicMaterial({  
        color: CoulHexa });  
    let sphere = new THREE.Mesh(sphereGeometry, sphereMaterial);  
    sphere.position.set(P.x,P.y,P.z);  
    if (bol) MaScene.add(sphere); // Affichage du point dans la scene  
    return sphere;  
} // renvoi du point(la sphere) pour une utilisation ultérieure  
} // fin function tracePt
```

Affichage d'un point de  $\mathcal{E}_3$  (explication ultérieure concernant l'affichage de sphères)

```
function tracePt(MaScene, P, CoulHexa, dimPt, bol){  
    let sphereGeometry = new THREE.SphereGeometry(dimPt, 12, 24);  
    let sphereMaterial = new THREE.MeshBasicMaterial({  
        color: CoulHexa });  
    let sphere = new THREE.Mesh(sphereGeometry, sphereMaterial);  
    sphere.position.set(P.x,P.y,P.z);  
    if (bol) MaScene.add(sphere); // Affichage du point dans la scene  
    return sphere;  
// renvoi du point(la sphere) pour une utilisation ultérieure  
} // fin function tracePt
```

Exemple :

```
let dimPt = 0.05; // Grosseur du point
```

Affichage d'un point de  $\mathcal{E}_3$  (explication ultérieure concernant l'affichage de sphères)

```
function tracePt(MaScene, P, CoulHexa, dimPt, bol){  
    let sphereGeometry = new THREE.SphereGeometry(dimPt,12,24);  
    let sphereMaterial = new THREE.MeshBasicMaterial({  
        color: CoulHexa });  
    let sphere = new THREE.Mesh(sphereGeometry, sphereMaterial);  
    sphere.position.set(P.x,P.y,P.z);  
    if (bol) MaScene.add(sphere); // Affichage du point dans la scene  
    return sphere;  
// renvoi du point(la sphere) pour une utilisation ultérieure  
} // fin function tracePt
```

Exemple :

```
let dimPt = 0.05; // Grosseur du point  
let A = new THREE.Vector3(2,0,0);  
let B = new THREE.Vector3(0,2,0);
```

Affichage d'un point de  $\mathcal{E}_3$  (explication ultérieure concernant l'affichage de sphères)

```

function tracePt(MaScene, P, CoulHexa, dimPt, bol){
    let sphereGeometry = new THREE.SphereGeometry(dimPt, 12, 24);
    let sphereMaterial = new THREE.MeshBasicMaterial({
        color: CoulHexa });
    let sphere = new THREE.Mesh(sphereGeometry, sphereMaterial);
    sphere.position.set(P.x,P.y,P.z);
    if (bol) MaScene.add(sphere); // Affichage du point dans la scene
    return sphere;
// renvoi du point(la sphere) pour une utilisation ultérieure
} // fin function tracePt

```

Exemple :

```

let dimPt = 0.05; // Grosseur du point
let A = new THREE.Vector3(2,0,0);
let B = new THREE.Vector3(0,2,0);
tracePt(scene, A, "#AA0000", dimPt, true);
// cf. let scene=new THREE.Scene();
tracePt(scene, B, "#00AA00", dimPt, true);

```

Affichage de deux points

Affichage d'un segment de  $\mathcal{E}_3$  (explication ultérieure concernant l'affichage de courbes)

```
function segment(MaScene,A,B,CoulHexa,epai){
```

Affichage d'un segment de  $\mathcal{E}_3$  (explication ultérieure concernant l'affichage de courbes)

```
function segment(MaScene,A,B,CoulHexa,epai){  
var geometry = new THREE.Geometry();
```

Affichage d'un segment de  $\mathcal{E}_3$  (explication ultérieure concernant l'affichage de courbes)

```
function segment(MaScene,A,B,CoulHexa,epai){  
    var geometry = new THREE.Geometry();  
    geometry.vertices.push(A,B);
```

Affichage d'un segment de  $\mathcal{E}_3$  (explication ultérieure concernant l'affichage de courbes)

```
function segment(MaScene,A,B,CoulHexa,epai){  
    var geometry = new THREE.Geometry();  
    geometry.vertices.push(A,B);  
    let segAB = new THREE.Line(geometry, new THREE.LineDashedMaterial  
        ({ // pas besoin de retour chariot dans le fichier js  
            color: CoulHexa,  
            linewidth: epai,  
        })); // fin variable segAB
```

Affichage d'un segment de  $\mathcal{E}_3$  (explication ultérieure concernant l'affichage de courbes)

```
function segment(MaScene,A,B,CoulHexa,epai){  
    var geometry = new THREE.Geometry();  
    geometry.vertices.push(A,B);  
    let segAB = new THREE.Line(geometry, new THREE.LineDashedMaterial  
        ({ // pas besoin de retour chariot dans le fichier js  
            color: CoulHexa,  
            linewidth: epai,  
        })); // fin variable segAB  
    MaScene.add(segAB);  
} // fin fonction segment [AB]
```

Affichage d'un segment de  $\mathcal{E}_3$  (explication ultérieure concernant l'affichage de courbes)

```
function segment(MaScene,A,B,CoulHexa,epai){  
    var geometry = new THREE.Geometry();  
    geometry.vertices.push(A,B);  
    let segAB = new THREE.Line(geometry, new THREE.LineDashedMaterial  
        ({ // pas besoin de retour chariot dans le fichier js  
            color: CoulHexa,  
            linewidth: epai,  
        })); // fin variable segAB  
    MaScene.add(segAB);  
} // fin fonction segment [AB]
```

Exemple :

```
let epaiCbe=2;
```

Affichage d'un segment de  $\mathcal{E}_3$  (explication ultérieure concernant l'affichage de courbes)

```
function segment(MaScene,A,B,CoulHexa,epai){
    var geometry = new THREE.Geometry();
    geometry.vertices.push(A,B);
    let segAB = new THREE.Line(geometry, new THREE.LineDashedMaterial
        ({ // pas besoin de retour chariot dans le fichier js
            color: CoulHexa,
            linewidth: epai,
        }));
    MaScene.add(segAB);
}
```

// fin fonction segment [AB]

Exemple :

```
let epaiCbe=2;
if (document.forms["formu"].Oui.checked)
    segment(scene, A, B, 0x555555, epaiCbe);
```

Affichage d'un segment : cocher le bouton radio oui

Affichage d'un segment de  $\mathcal{E}_3$  (explication ultérieure concernant l'affichage de courbes)

```
function segment(MaScene,A,B,CoulHexa,epai){
    var geometry = new THREE.Geometry();
    geometry.vertices.push(A,B);
    let segAB = new THREE.Line(geometry, new THREE.LineDashedMaterial
        ({ // pas besoin de retour chariot dans le fichier js
            color: CoulHexa,
            linewidth: epai,
        }));
    MaScene.add(segAB);
}
```

// fin fonction segment [AB]

Exemple :

```
let epaiCbe=2;
if (document.forms["formu"].Oui.checked)
    segment(scene, A, B, 0x555555, epaiCbe);
```

Affichage d'un segment : cocher le bouton radio oui

Rappel : boutons radio et onchange() vus en L1 en Info1B

Affichage d'un segment de  $\mathcal{E}_3$  (explication ultérieure concernant l'affichage de courbes)

```
function segment(MaScene,A,B,CoulHexa,epai){
    var geometry = new THREE.Geometry();
    geometry.vertices.push(A,B);
    let segAB = new THREE.Line(geometry, new THREE.LineDashedMaterial
        ({ // pas besoin de retour chariot dans le fichier js
            color: CoulHexa,
            linewidth: epai,
        }));
    MaScene.add(segAB);
}
```

// fin fonction segment [AB]

Exemple :

```
let epaiCbe=2;
if (document.forms["formu"].Oui.checked)
    segment(scene, A, B, 0x555555, epaiCbe);
```

Affichage d'un segment : cocher le bouton radio oui

Rappel : boutons radio et onchange() vus en L1 en Info1B

**Problème : les scènes s'ajoutent dans "webgl".**

Affichage d'un vecteur de  $\overrightarrow{\mathcal{E}_3}$ 

```
function vecteur(MaScene,A,B,CoulHexa,longCone,RayonCone){
```

Affichage d'un vecteur de  $\vec{\mathcal{E}}_3$ 

```
function vecteur(MaScene,A,B,CoulHexa,longCone,RayonCone){  
var vecAB = new THREE.Vector3( B.x-A.x, B.y-A.y, B.z-A.z );
```

Affichage d'un vecteur de  $\vec{\mathcal{E}}_3$ 

```
function vecteur(MaScene,A,B,CoulHexa,longCone,RayonCone){  
    var vecAB = new THREE.Vector3( B.x-A.x, B.y-A.y, B.z-A.z );  
    vecAB.normalize(); //vecteur unitaire
```

Affichage d'un vecteur de  $\mathcal{E}_3$  

```
function vecteur(MaScene,A,B,CoulHexa,longCone,RayonCone){  
    var vecAB = new THREE.Vector3( B.x-A.x, B.y-A.y, B.z-A.z );  
    vecAB.normalize(); //vecteur unitaire  
    MaScene.add( new THREE.ArrowHelper( vecAB, A, B.distanceTo(A),  
        CoulHexa,longCone,RayonCone ));  
}
```

Affichage d'un vecteur de  $\vec{E}_3$ 

```
function vecteur(MaScene,A,B,CoulHexa,longCone,RayonCone){  
    var vecAB = new THREE.Vector3( B.x-A.x, B.y-A.y, B.z-A.z );  
    vecAB.normalize(); //vecteur unitaire  
    MaScene.add( new THREE.ArrowHelper( vecAB, A, B.distanceTo(A),  
                                         CoulHexa,longCone,RayonCone ));  
}
```

Tracer  $\vec{AB}$  avec `ArrowHelper` :

Affichage d'un vecteur de  $\vec{\mathcal{E}_3}$ 

```
function vecteur(MaScene,A,B,CoulHexa,longCone,RayonCone){  
    var vecAB = new THREE.Vector3( B.x-A.x, B.y-A.y, B.z-A.z );  
    vecAB.normalize(); //vecteur unitaire  
    MaScene.add( new THREE.ArrowHelper( vecAB, A, B.distanceTo(A),  
                                         CoulHexa,longCone,RayonCone ));  
}
```

Tracer  $\overrightarrow{AB}$  avec `ArrowHelper` :

1) 
$$\frac{1}{\|\overrightarrow{AB}\|} \overrightarrow{AB} (\text{vecAB});$$

Affichage d'un vecteur de  $\vec{\mathcal{E}_3}$ 

```
function vecteur(MaScene,A,B,CoulHexa,longCone,RayonCone){  
    var vecAB = new THREE.Vector3( B.x-A.x, B.y-A.y, B.z-A.z );  
    vecAB.normalize(); //vecteur unitaire  
    MaScene.add( new THREE.ArrowHelper( vecAB, A, B.distanceTo(A),  
                                         CoulHexa,longCone,RayonCone ) );  
}
```

Tracer  $\overrightarrow{AB}$  avec **ArrowHelper** :                    2)  $A (A)$ ;

1) 
$$\frac{1}{\|\overrightarrow{AB}\|} \overrightarrow{AB} (\text{vecAB});$$

Affichage d'un vecteur de  $\vec{\mathcal{E}_3}$ 

```
function vecteur(MaScene,A,B,CoulHexa,longCone,RayonCone){  
    var vecAB = new THREE.Vector3( B.x-A.x, B.y-A.y, B.z-A.z );  
    vecAB.normalize(); //vecteur unitaire  
    MaScene.add( new THREE.ArrowHelper( vecAB, A, B.distanceTo(A),  
                                         CoulHexa,longCone,RayonCone ) );  
}
```

Tracer  $\overrightarrow{AB}$  avec `ArrowHelper` :

$$1) \frac{1}{\|\overrightarrow{AB}\|} \overrightarrow{AB} (\text{vecAB});$$

2)  $A$  ( $A$ );

3)  $AB$  ( $B.\text{distanceTo}(A)$ );...

Affichage d'un vecteur de  $\vec{\mathcal{E}_3}$ 

```
function vecteur(MaScene,A,B,CoulHexa,longCone,RayonCone){
    var vecAB = new THREE.Vector3( B.x-A.x, B.y-A.y, B.z-A.z );
    vecAB.normalize(); //vecteur unitaire
    MaScene.add( new THREE.ArrowHelper( vecAB, A, B.distanceTo(A),
        CoulHexa,longCone,RayonCone ) );
}
```

Tracer  $\overrightarrow{AB}$  avec `ArrowHelper` :

$$1) \frac{1}{\|\overrightarrow{AB}\|} \overrightarrow{AB} (\text{vecAB});$$

2)  $A$  (`A`);

3)  $AB$  (`B.distanceTo(A)`);...

Exemple :

```
let dimPt = 0.05;
```

Affichage d'un vecteur de  $\vec{\mathcal{E}_3}$ 

```
function vecteur(MaScene,A,B,CoulHexa,longCone,RayonCone){
    var vecAB = new THREE.Vector3( B.x-A.x, B.y-A.y, B.z-A.z );
    vecAB.normalize(); //vecteur unitaire
    MaScene.add( new THREE.ArrowHelper( vecAB, A, B.distanceTo(A),
        CoulHexa,longCone,RayonCone ) );
}
```

Tracer  $\overrightarrow{AB}$  avec `ArrowHelper` :

$$1) \frac{1}{\|\overrightarrow{AB}\|} \overrightarrow{AB} (\text{vecAB});$$

2)  $A (A);$

3)  $AB (B.\text{distanceTo}(A));...$

Exemple :

```
let dimPt = 0.05;
let A = new THREE.Vector3(2,0,0);
```

Affichage d'un vecteur de  $\vec{\mathcal{E}_3}$ 

```
function vecteur(MaScene, A, B, CoulHexa, longCone, RayonCone) {
    var vecAB = new THREE.Vector3( B.x-A.x, B.y-A.y, B.z-A.z );
    vecAB.normalize(); //vecteur unitaire
    MaScene.add( new THREE.ArrowHelper( vecAB, A, B.distanceTo(A),
        CoulHexa, longCone, RayonCone ) );
}
```

Tracer  $\overrightarrow{AB}$  avec `ArrowHelper` :

- 1)  $\frac{1}{\|\overrightarrow{AB}\|} \overrightarrow{AB}$  (`vecAB`);
- 2)  $A$  (`A`);
- 3)  $AB$  (`B.distanceTo(A)`);...

Exemple :

```
let dimPt = 0.05;
let A = new THREE.Vector3(2,0,0);
let B = new THREE.Vector3(0,2,0);
```

Affichage d'un vecteur de  $\vec{\mathcal{E}_3}$ 

```
function vecteur(MaScene, A, B, CoulHexa, longCone, RayonCone){
    var vecAB = new THREE.Vector3( B.x-A.x, B.y-A.y, B.z-A.z );
    vecAB.normalize(); //vecteur unitaire
    MaScene.add( new THREE.ArrowHelper( vecAB, A, B.distanceTo(A),
        CoulHexa, longCone, RayonCone ) );
}
```

Tracer  $\overrightarrow{AB}$  avec `ArrowHelper` :

$$1) \frac{1}{\| \overrightarrow{AB} \|} \overrightarrow{AB} (\text{vecAB});$$

2)  $A (A);$

3)  $AB (B.\text{distanceTo}(A)); \dots$

Exemple :

```
let dimPt = 0.05;
let A = new THREE.Vector3(2,0,0);
let B = new THREE.Vector3(0,2,0);
tracePt(scene, A, "#AA0000", dimPt, true);
tracePt(scene, B, "#00AA00", dimPt, true);
```

Affichage d'un vecteur de  $\vec{\mathcal{E}_3}$ 

```
function vecteur(MaScene, A, B, CoulHexa, longCone, RayonCone){
    var vecAB = new THREE.Vector3( B.x-A.x, B.y-A.y, B.z-A.z );
    vecAB.normalize(); //vecteur unitaire
    MaScene.add( new THREE.ArrowHelper( vecAB, A, B.distanceTo(A),
        CoulHexa, longCone, RayonCone ) );
}
```

Tracer  $\overrightarrow{AB}$  avec `ArrowHelper` :

$$1) \frac{1}{\|\overrightarrow{AB}\|} \overrightarrow{AB} (\text{vecAB});$$

2)  $A$  ( $A$ );

3)  $AB$  ( $B.\text{distanceTo}(A)$ );...

Exemple :

```
let dimPt = 0.05;
let A = new THREE.Vector3(2,0,0);
let B = new THREE.Vector3(0,2,0);
tracePt(scene, A, "#AA0000", dimPt, true);
tracePt(scene, B, "#00AA00", dimPt, true);
let longCone=0.25, RayonCone = 0.125;
```

Affichage d'un vecteur de  $\vec{\mathcal{E}_3}$ 

```
function vecteur(MaScene, A, B, CoulHexa, longCone, RayonCone){
    var vecAB = new THREE.Vector3( B.x-A.x, B.y-A.y, B.z-A.z );
    vecAB.normalize(); //vecteur unitaire
    MaScene.add( new THREE.ArrowHelper( vecAB, A, B.distanceTo(A),
        CoulHexa, longCone, RayonCone ) );
}
```

Tracer  $\overrightarrow{AB}$  avec ArrowHelper :

$$1) \frac{1}{\|\overrightarrow{AB}\|} \overrightarrow{AB} (\text{vecAB});$$

2)  $A (A);$

3)  $AB (B.\text{distanceTo}(A));...$

Exemple :

```
let dimPt = 0.05;
let A = new THREE.Vector3(2,0,0);
let B = new THREE.Vector3(0,2,0);
tracePt(scene, A, "#AA0000", dimPt, true);
tracePt(scene, B, "#00AA00", dimPt, true);
let longCone=0.25, RayonCone = 0.125;
vecteur(scene, A, B, 0x0000FF, longCone, RayonCone );
```

Affichage d'un vecteur

## 2 Points et Vecteurs

### • Définitions

- Méthodes communes à THREE.Vector $n$ ,  $n \in \{2; 3\}$ 
  - Cas du plan i.e.  $n = 2$
  - Cas de l'espace i.e.  $n = 3$

## Additions de vecteurs

Définition (Méthode `add`)

Soit  $\underline{u}$  (resp.  $\underline{v}$ ) une instance de `Vector $n$`  représentant le vecteur  $\overrightarrow{u}$  (resp.  $\overrightarrow{v}$ ).

## Additions de vecteurs

Définition (Méthode `add`)

Soit  $\vec{u}$  (resp.  $\vec{v}$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

La syntaxe `v.add(u);` permet d'affecter à  $\vec{v}$  le vecteur  $\vec{u} + \vec{v}$  c'est-à-dire :

$$\vec{v} \leftarrow \vec{u} + \vec{v}$$

et la valeur initiale de  $\vec{v}$  est perdue.

## Additions de vecteurs

Définition (Méthode `add`)

Soit  $\underline{u}$  (resp.  $\underline{v}$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

La syntaxe `v.add(u);` permet d'affecter à  $\underline{v}$  le vecteur  $\vec{u} + \vec{v}$  c'est-à-dire :

$$\vec{v} \leftarrow \vec{u} + \vec{v}$$

et la valeur initiale de  $\underline{v}$  est perdue.

Définition (Méthode `addVectors`)

Soit  $\underline{u}$  (resp.  $\underline{v}$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

## Additions de vecteurs

### Définition (Méthode `add`)

Soit  $\vec{u}$  (resp.  $\vec{v}$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

La syntaxe `v.add(u);` permet d'affecter à  $\vec{v}$  le vecteur  $\vec{u} + \vec{v}$  c'est-à-dire :

$$\vec{v} \leftarrow \vec{u} + \vec{v}$$

et la valeur initiale de  $\vec{v}$  est perdue.

### Définition (Méthode `addVectors`)

Soit  $\vec{u}$  (resp.  $\vec{v}$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

Soit  $\vec{w}$  une instance de `Vector $n$`  représentant le vecteur  $\vec{w} = \vec{u} + \vec{v}$ .

La syntaxe est : `w.addVectors(u, v);`

Somme de deux vecteurs

## Additions de vecteurs : body du fichier html

```
<body onload="init()">
<div id="webgl"></div>
<form id="formu"    method="post" action="mailto:moi@wanadoo.fr">
```

## Additions de vecteurs : body du fichier html

```
<body onload="init()">
<div id="webgl"></div>
<form id="formu"    method="post" action="mailto:moi@wanadoo.fr">
  <fieldset>  <legend>Méthode add ou addVectors</legend>
```

## Additions de vecteurs : body du fichier html

```
<body onload="init()">
<div id="webgl"></div>
<form id="formu" method="post" action="mailto:moi@wanadoo.fr">
  <fieldset>  <legend>Méthode add ou addVectors</legend>
  <input type="radio" name="AdditionDeVecteurs" value="add"
         id="add" onchange="init(); " />
```

## Additions de vecteurs : body du fichier html

```
<body onload="init()">
<div id="webgl"></div>
<form id="formu" method="post" action="mailto:moi@wanadoo.fr">
  <fieldset>  <legend>Méthode add ou addVectors</legend>
  <input type="radio" name="AdditionDeVecteurs" value="add"
         id="add" onchange="init();"/>
  <label for="add"> add</label>
```

## Additions de vecteurs : body du fichier html

```
<body onload="init()">
<div id="webgl"></div>
<form id="formu" method="post" action="mailto:moi@wanadoo.fr">
  <fieldset>  <legend>Méthode add ou addVectors</legend>
    <input type="radio" name="AdditionDeVecteurs" value="add"
           id="add" onchange="init();;" />
    <label for="add"> add</label>
    <input type="radio" name="AdditionDeVecteurs"
           value="addVectors"   id="addVectors"
           checked="checked"  onchange="init();;" />
    <label for="addVectors"> addVectors</label>
```

## Additions de vecteurs : body du fichier html

```
<body onload="init()">
  <div id="webgl"></div>
  <form id="formu" method="post" action="mailto:moi@wanadoo.fr">
    <fieldset>  <legend>Méthode add ou addVectors</legend>
      <input type="radio" name="AdditionDeVecteurs" value="add"
             id="add" onchange="init();"/>
      <label for="add"> add</label>
      <input type="radio" name="AdditionDeVecteurs"
             value="addVectors" id="addVectors"
             checked="checked" onchange="init();"/>
      <label for="addVectors"> addVectors</label>
      <span id="spanEspace">Le vecteur <span id="orange"> orange
        </span> est la somme du vecteur <span id="cyan">
        cyan </span> et du vecteur <span id="magenta">
        magenta </span></span>
    </fieldset>
  </form>
</body>
```

Somme de deux vecteurs

## Additions de vecteurs : code JavaScript

```
let o3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(0.5,1,0);
let v = new THREE.Vector3(1,0.5,0);
```

## Additions de vecteurs : code JavaScript

```
let o3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(0.5,1,0);
let v = new THREE.Vector3(1,0.5,0);
vecteur(scene,o3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,o3,v, 0x00FFFF, 0.25, 0.125 );
```

## Additions de vecteurs : code JavaScript

```
let o3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(0.5,1,0);
let v = new THREE.Vector3(1,0.5,0);
vecteur(scene,o3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,o3,v, 0x00FFFF, 0.25, 0.125 );
if (document.forms["formu"].add.checked){
    v.add(u);
    vecteur(scene,u,v, 0x00FFFF, 0.25, 0.125 );
    vecteur(scene,o3,v, 0xFF9900, 0.25, 0.125 );
}
```

## Additions de vecteurs : code JavaScript

```
let o3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(0.5,1,0);
let v = new THREE.Vector3(1,0.5,0);
vecteur(scene,o3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,o3,v, 0x00FFFF, 0.25, 0.125 );
if (document.forms["formu"].add.checked){
    v.add(u);
    vecteur(scene,u,v, 0x00FFFF, 0.25, 0.125 );
    vecteur(scene,o3,v, 0xFF9900, 0.25, 0.125 );
}
else{
    let w = new THREE.Vector3(0,0,0);
    w.addVectors(u,v);
    vecteur(scene,u,w, 0x00FFFF, 0.25, 0.125 );
    vecteur(scene,v,w, 0xFF00FF, 0.25, 0.125 );
    vecteur(scene,o3,w, 0xFF9900, 0.25, 0.125 );
}
```

Somme de deux vecteurs

## Recopie de vecteurs ou de points

Définition (Recopie, méthodes `clone` et `copy`)

Soit `Toto` une instance de `Vector $n$`  représentant le point  $A$  ou le vecteur  $\vec{u}$ .

## Recopie de vecteurs ou de points

Définition (Recopie, méthodes `clone` et `copy`)

Soit `Toto` une instance de `Vector $n$`  représentant le point  $A$  ou le vecteur  $\vec{u}$ .

Soit `Gertrude` une instance de `Vector $n$`  dont on va affecter le contenu de `Toto`.

## Recopie de vecteurs ou de points

Définition (Recopie, méthodes `clone` et `copy`)

Soit `Toto` une instance de `Vector $n$`  représentant le point  $A$  ou le vecteur  $\vec{u}$ .

Soit `Gertrude` une instance de `Vector $n$`  dont on va affecter le contenu de `Toto`.

- La première solution est : `let Gertrude = Toto.clone();`

## Recopie de vecteurs ou de points

Définition (Recopie, méthodes `clone` et `copy`)

Soit `Toto` une instance de `Vector $n$`  représentant le point  $A$  ou le vecteur  $\vec{u}$ .

Soit `Gertrude` une instance de `Vector $n$`  dont on va affecter le contenu de `Toto`.

- La première solution est : `let Gertrude = Toto.clone();`
- La seconde solution est : `Gertrude.copy(Toto);`

## Recopie de vecteurs ou de points

Définition (Recopie, méthodes `clone` et `copy`)

Soit `Toto` une instance de `Vector $n$`  représentant le point  $A$  ou le vecteur  $\vec{u}$ .

Soit `Gertrude` une instance de `Vector $n$`  dont on va affecter le contenu de `Toto`.

- La première solution est : `let Gertrude = Toto.clone();`
- La seconde solution est : `Gertrude.copy(Toto);`

## Remarque

- Lors de l'utilisation de la syntaxe `Gertrude = Toto.clone();`, la définition « du type » de `Gertrude` n'est pas obligatoire;

## Recopie de vecteurs ou de points

Définition (Recopie, méthodes `clone` et `copy`)

Soit `Toto` une instance de `Vector $n$`  représentant le point  $A$  ou le vecteur  $\vec{u}$ .

Soit `Gertrude` une instance de `Vector $n$`  dont on va affecter le contenu de `Toto`.

- La première solution est : `let Gertrude = Toto.clone();`
- La seconde solution est : `Gertrude.copy(Toto);`

## Remarque

- Lors de l'utilisation de la syntaxe `Gertrude = Toto.clone();`, la définition « du type » de `Gertrude` n'est pas obligatoire;
- Lors de l'utilisation de la syntaxe `Gertrude.copy(Toto);`, nous devons définir « le type » de `Gertrude` et donc écrire avant :
  - ▶ `let Gertrude = new THREE.Vector2(0, 0); si n = 2;`

## Recopie de vecteurs ou de points

Définition (Recopie, méthodes `clone` et `copy`)

Soit `Toto` une instance de `Vector $n$`  représentant le point  $A$  ou le vecteur  $\vec{u}$ .

Soit `Gertrude` une instance de `Vector $n$`  dont on va affecter le contenu de `Toto`.

- La première solution est : `let Gertrude = Toto.clone();`
- La seconde solution est : `Gertrude.copy(Toto);`

## Remarque

- Lors de l'utilisation de la syntaxe `Gertrude = Toto.clone();`, la définition « du type » de `Gertrude` n'est pas obligatoire;
- Lors de l'utilisation de la syntaxe `Gertrude.copy(Toto);`, nous devons définir « le type » de `Gertrude` et donc écrire avant :
  - ▶ `let Gertrude = new THREE.Vector2(0, 0); si n = 2;`
  - ▶ `let Gertrude = new THREE.Vector3(0, 0, 0); si n = 3.`

## Vecteurs et mesures algébriques

### Définition (Mesure algébrique)

Soit  $\vec{u}$  un vecteur (unitaire) non nul. Soit  $\vec{v}$  un vecteur colinéaire à  $\vec{u}$ .

## Vecteurs et mesures algébriques

## Définition (Mesure algébrique)

Soit  $\vec{u}$  un vecteur (unitaire) non nul. Soit  $\vec{v}$  un vecteur colinéaire à  $\vec{u}$ . La mesure algébrique du vecteur  $\vec{v}$ , notée  $\overline{v}$ , est le scalaire vérifiant :

$$\vec{v} = \overline{v} \cdot \vec{u}$$

## Vecteurs et mesures algébriques

### Définition (Mesure algébrique)

Soit  $\vec{u}$  un vecteur (unitaire) non nul. Soit  $\vec{v}$  un vecteur colinéaire à  $\vec{u}$ . La mesure algébrique du vecteur  $\vec{v}$ , notée  $\overline{v}$ , est le scalaire vérifiant :

$$\vec{v} = \overline{v} \cdot \vec{u}$$

### Définition (Méthode `length`)

Soit  $u$  une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$ .

## Vecteurs et mesures algébriques

### Définition (Mesure algébrique)

Soit  $\vec{u}$  un vecteur (unitaire) non nul. Soit  $\vec{v}$  un vecteur colinéaire à  $\vec{u}$ . La mesure algébrique du vecteur  $\vec{v}$ , notée  $\overline{v}$ , est le scalaire vérifiant :

$$\vec{v} = \overline{v} \cdot \vec{u}$$

### Définition (Méthode `length`)

Soit  $u$  une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$ .

La syntaxe `u.length()`; retourne la norme  $\|\vec{u}\|$  du vecteur  $\vec{u}$ .

## Vecteurs et mesures algébriques

### Définition (Mesure algébrique)

Soit  $\vec{u}$  un vecteur (unitaire) non nul. Soit  $\vec{v}$  un vecteur colinéaire à  $\vec{u}$ . La mesure algébrique du vecteur  $\vec{v}$ , notée  $\overline{v}$ , est le scalaire vérifiant :

$$\vec{v} = \overline{v} \cdot \vec{u}$$

### Définition (Méthode `length`)

Soit  $u$  une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$ .

La syntaxe `u.length();` retourne la norme  $\|\vec{u}\|$  du vecteur  $\vec{u}$ .

### Définition (Méthode `setLength`)

Soit  $u$  une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  et  $k$  un flottant de valeur  $k$ .

## Vecteurs et mesures algébriques

### Définition (Mesure algébrique)

Soit  $\vec{u}$  un vecteur (unitaire) non nul. Soit  $\vec{v}$  un vecteur colinéaire à  $\vec{u}$ . La mesure algébrique du vecteur  $\vec{v}$ , notée  $\overline{v}$ , est le scalaire vérifiant :

$$\vec{v} = \overline{v} \cdot \vec{u}$$

### Définition (Méthode `length`)

Soit  $u$  une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$ .

La syntaxe `u.length();` retourne la norme  $\|\vec{u}\|$  du vecteur  $\vec{u}$ .

### Définition (Méthode `setLength`)

Soit  $u$  une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  et  $k$  un flottant de valeur  $k$ . La syntaxe `u.setLength( k );` construit le vecteur :

$$k \cdot \left( \frac{1}{\|\vec{u}\|} \vec{u} \right); \text{ Rappel : } \left\| \frac{1}{\|\vec{u}\|} \vec{u} \right\| = 1$$

Vecteurs et mesures algébriques

## Vecteurs et mesures algébriques : code JavaScript

```
var u = new THREE.Vector3(0.5,1,0); // variable globale
```

---

## Vecteurs et mesures algébriques : code JavaScript

```
var u = new THREE.Vector3(0.5,1,0); // variable globale  
  
//dans la fonction init()  
let O3 = new THREE.Vector3(0,0,0);  
vecteur(scene,O3,u, 0xFF00FF, 0.25, 0.125 ); //vecteur position  
let mes = "";
```

## Vecteurs et mesures algébriques : code JavaScript

```
var u = new THREE.Vector3(0.5,1,0); // variable globale

---

//dans la fonction init()  
let O3 = new THREE.Vector3(0,0,0);  
vecteur(scene,O3,u, 0xFF00FF, 0.25, 0.125 ); //vecteur position  
let mes = "";  
if (document.forms["formu"].norme.checked)  
    mes = "norme de u : "+u.length()+"<br /><hr />";
```

## Vecteurs et mesures algébriques : code JavaScript

```
var u = new THREE.Vector3(0.5,1,0); // variable globale

//dans la fonction init()
let O3 = new THREE.Vector3(0,0,0);
vecteur(scene,O3,u, 0xFF00FF, 0.25, 0.125 ); //vecteur position
let mes = "";
if (document.forms["formu"].norme.checked)
    mes = "norme de u : "+u.length()+"<br /><hr />";
else{
    let k = prompt("Entrer la nouvelle valeur de la norme");
    u.setLength(k);
    vecteur(scene,O3,u, 0xFF9900, 0.25, 0.125 );
```

## Vecteurs et mesures algébriques : code JavaScript

```

var u = new THREE.Vector3(0.5,1,0); // variable globale

//dans la fonction init()
let O3 = new THREE.Vector3(0,0,0);
vecteur(scene,O3,u, 0xFF00FF, 0.25, 0.125 ); //vecteur position
let mes = "";
if (document.forms["formu"].norme.checked)
    mes = "norme de u : "+u.length()+"<br /><hr />";
else{
    let k = prompt("Entrer la nouvelle valeur de la norme");
    u.setLength(k);
    vecteur(scene,O3,u, 0xFF9900, 0.25, 0.125 );
    if (k<0)
        mes = "vecteur de sens opposé au précédent <br />"
    mes += "nouvelle norme de u : "+u.length()+"<br /><hr />";
}
// fin else
document.getElementById("result").innerHTML +=mes;

```

Vecteurs et mesures algébriques

## Vecteurs et multiplication par un scalaire

Définition (Méthode `multiplyScalar`)

Soit  $u$  une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  et  $k$  un flottant de valeur  $k$ .

## Vecteurs et multiplication par un scalaire

Définition (Méthode `multiplyScalar`)

Soit  $u$  une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  et  $k$  un flottant de valeur  $k$ . La syntaxe `u.multiplyScalar( k );` construit le vecteur :

$$k \cdot \vec{u}$$

## Vecteurs et multiplication par un scalaire

Définition (Méthode `multiplyScalar`)

Soit  $u$  une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  et  $k$  un flottant de valeur  $k$ . La syntaxe `u.multiplyScalar(k);` construit le vecteur :

$$k \cdot \vec{u}$$

Différence entre `setLength(k);` et `multiplyScalar(k);` concernant les normes :

## Vecteurs et multiplication par un scalaire

Définition (Méthode `multiplyScalar`)

Soit  $u$  une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  et  $k$  un flottant de valeur  $k$ . La syntaxe `u.multiplyScalar(k);` construit le vecteur :

$$k \cdot \vec{u}$$

Différence entre `setLength(k);` et `multiplyScalar(k);` concernant les normes :

Vecteur défini par $u$	<code>u.setLength(k);</code>	<code>u.multiplyScalar(k);</code>
$\ \vec{u}\  \in \mathbb{R}^+$	$ k  \in \mathbb{R}^+$	$ k  \times \ \vec{u}\  \in \mathbb{R}^+$

## Vecteurs et multiplication par un scalaire

Définition (Méthode `multiplyScalar`)

Soit  $u$  une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  et  $k$  un flottant de valeur  $k$ . La syntaxe `u.multiplyScalar(k);` construit le vecteur :

$$k \cdot \vec{u}$$

Différence entre `setLength(k);` et `multiplyScalar(k);` concernant les normes :

Vecteur défini par <code>u</code>	<code>u.setLength(k);</code>	<code>u.multiplyScalar(k);</code>
$\ \vec{u}\  \in \mathbb{R}^+$	$ k  \in \mathbb{R}^+$	$ k  \times \ \vec{u}\  \in \mathbb{R}^+$

Remarque (Méthode `negate`)

La syntaxe `u.multiplyScalar(-1);` peut être remplacée par `u.negate();`

## Vecteurs et multiplication par un scalaire

```
let u = new THREE.Vector3(1,-1,0);
let v = new THREE.Vector3(1,-1,0);
let o3 = new THREE.Vector3(0,0,0);
vecteur(scene,o3,u, 0xFF00FF, 0.25, 0.125 ); //vecteur position
```

## Vecteurs et multiplication par un scalaire

```
let u = new THREE.Vector3(1,-1,0);
let v = new THREE.Vector3(1,-1,0);
let O3 = new THREE.Vector3(0,0,0);
vecteur(scene,O3,u, 0xFF00FF, 0.25, 0.125 ); //vecteur position
let normU = u.length();
let mes =" u("+v.x+";"+v.y+";"+v.z+) et v=u";
mes += "Norme de u et de v en 0xFF00FF : "+normU+ " <hr />    "
u.setLength(-2);
```

## Vecteurs et multiplication par un scalaire

```
let u = new THREE.Vector3(1,-1,0);
let v = new THREE.Vector3(1,-1,0);
let O3 = new THREE.Vector3(0,0,0);
vecteur(scene,O3,u, 0xFF00FF, 0.25, 0.125 ); //vecteur position
let normU = u.length();
let mes =" u("+v.x+";"+v.y+";"+v.z+) et v=u";
mes += "Norme de u et de v en 0xFF00FF : "+normU+ " <hr /> "
u.setLength(-2);
v.multiplyScalar(-2);
```

## Vecteurs et multiplication par un scalaire

```
let u = new THREE.Vector3(1,-1,0);
let v = new THREE.Vector3(1,-1,0);
let O3 = new THREE.Vector3(0,0,0);
vecteur(scene,O3,u, 0xFF00FF, 0.25, 0.125 ); //vecteur position
let normU = u.length();
let mes =" u("+v.x+";"+v.y+";"+v.z+) et v=u";
mes += "Norme de u et de v en 0xFF00FF : "+normU+" <hr /> "
u.setLength(-2);
v.multiplyScalar(-2);
mes += "Multiplication de u et v par -2<br />";
mes += "Nouvelle norme de u en 0x00FFFF : "+u.length()+"<br />";
mes += "Nouvelle norme de v en 0xFFAA00 : "+v.length()+" = "
+2+" &#xD7; "+normU+"<br />";
vecteur(scene,O3,u, 0x00FFFF, 0.25, 0.125 );
vecteur(scene,O3,v, 0xFFAA00, 0.25, 0.125 );
document.getElementById("result").innerHTML +=mes;
```

Vecteurs et multiplication par un scalaire

## Ajout d'un vecteur colinéaire à un vecteur donné

Définition (Méthode `addScaledVector`)

Soit  $\underline{u}$  (resp.  $\underline{v}$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

## Ajout d'un vecteur colinéaire à un vecteur donné

Définition (Méthode `addScaledVector`)

Soit  $\underline{u}$  (resp.  $\underline{v}$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

Soit  $k$  une variable réelle de valeur  $\lambda$ .

## Ajout d'un vecteur colinéaire à un vecteur donné

Définition (Méthode `addScaledVector`)

Soit `u` (resp. `v`) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

Soit `k` une variable réelle de valeur  $\lambda$ .

La syntaxe `u.addScaledVector(v, k);` permet d'affecter à `u` le vecteur  $\vec{u} + \lambda \vec{v}$  c'est-à-dire :

$$\vec{u} \leftarrow \vec{u} + \lambda \vec{v}$$

## Ajout d'un vecteur colinéaire à un vecteur donné

Définition (Méthode `addScaledVector`)

Soit  $\mathbf{u}$  (resp.  $\mathbf{v}$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

Soit  $k$  une variable réelle de valeur  $\lambda$ .

La syntaxe  $\mathbf{u}.addScaledVector(\mathbf{v}, k)$ ; permet d'affecter à  $\mathbf{u}$  le vecteur  $\vec{u} + \lambda \vec{v}$  c'est-à-dire :

$$\vec{u} \leftarrow \vec{u} + \lambda \vec{v}$$

et la valeur initiale de  $\mathbf{u}$  est perdue.

Ajout d'un vecteur colinéaire à un vecteur donné

## Ajout d'un vecteur colinéaire à un vecteur donné

```
let o3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(1,1,0);
let v = new THREE.Vector3(-1,1,0);
let k=2;
vecteur(scene,o3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,o3,v, 0x00FFFF, 0.25, 0.125 );
```

## Ajout d'un vecteur colinéaire à un vecteur donné

```
let o3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(1,1,0);
let v = new THREE.Vector3(-1,1,0);
let k=2;
vecteur(scene,o3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,o3,v, 0x00FFFF, 0.25, 0.125 );
let v0 = v.clone();
let u0 = u.clone();
```

## Ajout d'un vecteur colinéaire à un vecteur donné

```
let o3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(1,1,0);
let v = new THREE.Vector3(-1,1,0);
let k=2;
vecteur(scene,o3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,o3,v, 0x00FFFF, 0.25, 0.125 );
let v0 = v.clone();
let u0 = u.clone();
v0.multiplyScalar(k); // v0 <- k *v
vecteur(scene,o3,v0, 0xAAAAAA, 0.25, 0.125 );
```

## Ajout d'un vecteur colinéaire à un vecteur donné

```
let O3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(1,1,0);
let v = new THREE.Vector3(-1,1,0);
let k=2;
vecteur(scene,O3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,O3,v, 0x00FFFF, 0.25, 0.125 );
let v0 = v.clone();
let u0 = u.clone();
v0.multiplyScalar(k); // v0 <- k *v
vecteur(scene,O3,v0, 0xAAAAAA, 0.25, 0.125 );
u.addScaledVector(v,k); // u<- u + k *v
vecteur(scene,u0,u, 0x444444, 0.25, 0.125 );
vecteur(scene,O3,u, 0xFF9900, 0.25, 0.125 );
```

Ajout d'un vecteur colinéaire à un vecteur donné

## Vecteurs (points) et soustraction

Définition (Méthode `sub`)

Soit  $\underline{u}$  (resp.  $\underline{v}$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

## Vecteurs (points) et soustraction

Définition (Méthode `sub`)

Soit  $\vec{u}$  (resp.  $\vec{v}$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

La syntaxe `v = v.sub(u);` permet d'affecter à  $\vec{v}$  le vecteur  $\vec{v} - \vec{u}$ .

## Vecteurs (points) et soustraction

Définition (Méthode `sub`)

Soit  $\mathbf{u}$  (resp.  $\mathbf{v}$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

La syntaxe  $\mathbf{v}.sub(\mathbf{u})$ ; permet d'affecter à  $\mathbf{v}$  le vecteur  $\vec{v} - \vec{u}$ .

## Remarque (Point affine transformé en vecteur)

Soit  $\mathbf{u}$  (resp.  $\mathbf{v}$ ) une instance de `Vector $n$`  représentant le point A (resp. B).

La syntaxe  $\mathbf{v}.sub(\mathbf{u})$ ; transforme le contenu de  $\mathbf{v}$  puisque ce dernier contient  $\vec{AB}$  ( $\forall O, \vec{AB} = \vec{OB} - \vec{OA}$ ).

## Vecteurs (points) et soustraction

Définition (Méthode `sub`)

Soit  $\mathbf{u}$  (resp.  $\mathbf{v}$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

La syntaxe  $\mathbf{v}.sub(\mathbf{u})$ ; permet d'affecter à  $\mathbf{v}$  le vecteur  $\vec{v} - \vec{u}$ .

## Remarque (Point affine transformé en vecteur)

Soit  $\mathbf{u}$  (resp.  $\mathbf{v}$ ) une instance de `Vector $n$`  représentant le point  $A$  (resp.  $B$ ).

La syntaxe  $\mathbf{v}.sub(\mathbf{u})$ ; transforme le contenu de  $\mathbf{v}$  puisque ce dernier contient  $\vec{AB}$  ( $\forall O, \vec{AB} = \vec{OB} - \vec{OA}$ ).

Définition (Méthode `subScalar`)

Soit  $Toto$  une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  ou le point  $A$  et  $k$  une variable stockant le scalaire  $k$ .

## Vecteurs (points) et soustraction

Définition (Méthode `sub`)

Soit  $\mathbf{u}$  (resp.  $\mathbf{v}$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

La syntaxe `v . sub( u );` permet d'affecter à  $\mathbf{v}$  le vecteur  $\vec{v} - \vec{u}$ .

## Remarque (Point affine transformé en vecteur)

Soit  $\mathbf{u}$  (resp.  $\mathbf{v}$ ) une instance de `Vector $n$`  représentant le point  $A$  (resp.  $B$ ).

La syntaxe `v . sub( u );` transforme le contenu de  $\mathbf{v}$  puisque ce dernier contient  $\vec{AB}$  ( $\forall O, \vec{AB} = \vec{OB} - \vec{OA}$ ).

Définition (Méthode `subScalar`)

Soit  $Toto$  une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  ou le point  $A$  et  $k$  une variable stockant le scalaire  $k$ .

La syntaxe `Toto .subScalar( k );` permet de retrancher à chaque coordonnée de  $A$  ou composante de  $\vec{u}$  la valeur  $k$ .

## Vecteurs (points) et soustraction

```
let o3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(-1,1,0);
let v = new THREE.Vector3(0,0,-1);
vecteur(scene,o3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,o3,v, 0x00FFFF, 0.25, 0.125 );
```

## Vecteurs (points) et soustraction

```
let o3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(-1,1,0);
let v = new THREE.Vector3(0,0,-1);
vecteur(scene,o3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,o3,v, 0x00FFFF, 0.25, 0.125 );
let u0 = new THREE.Vector3( 0 , 0 , 0 );
u0.copy(u); //il faut definir u0 pour appeler la methode
```

## Vecteurs (points) et soustraction

```
let O3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(-1,1,0);
let v = new THREE.Vector3(0,0,-1);
vecteur(scene,O3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,O3,v, 0x00FFFF, 0.25, 0.125 );
let u0 = new THREE.Vector3( 0 , 0 , 0 );
u0.copy(u); //il faut definir u0 pour appeler la methode
u.sub(v);
vecteur(scene,u0,u, 0x444444, 0.35, 0.25 );
vecteur(scene,O3,u, 0xFF9900, 0.25, 0.125 );
```

## Vecteurs (points) et soustraction

```
let o3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(-1,1,0);
let v = new THREE.Vector3(0,0,-1);
vecteur(scene,o3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,o3,v, 0x00FFFF, 0.25, 0.125 );
let u0 = new THREE.Vector3( 0 , 0 , 0 );
u0.copy(u); //il faut definir u0 pour appeler la methode
u.sub(v);
vecteur(scene,u0,u, 0x444444, 0.35, 0.25 );
vecteur(scene,o3,u, 0xFF9900, 0.25, 0.125 );
u.subScalar(1);
vecteur(scene,o3,u, 0x005555, 0.25, 0.125 );
```

Soustraction : points et vecteurs

## Vecteurs (points) et soustraction

```
let dimPt = 0.05;
let O3 = new THREE.Vector3(0,0,0);
let A = new THREE.Vector3(2,0,0);
let B = new THREE.Vector3(0,2,0);
let B0 = B.clone();
tracePt(scene, A, "#FF00FF",dimPt,true);
tracePt(scene, B, "#00FFFF",dimPt,true);
```

## Vecteurs (points) et soustraction

```
let dimPt = 0.05;
let O3 = new THREE.Vector3(0,0,0);
let A = new THREE.Vector3(2,0,0);
let B = new THREE.Vector3(0,2,0);
let B0 = B.clone();
tracePt(scene, A, "#FF00FF",dimPt,true);
tracePt(scene, B, "#00FFFF",dimPt,true);
B.sub(B,A);
```

## Vecteurs (points) et soustraction

```
let dimPt = 0.05;
let O3 = new THREE.Vector3(0,0,0);
let A = new THREE.Vector3(2,0,0);
let B = new THREE.Vector3(0,2,0);
let B0 = B.clone();
tracePt(scene, A, "#FF00FF", dimPt, true);
tracePt(scene, B, "#00FFFF", dimPt, true);
B.sub(B,A);
vecteur(scene,O3,B, 0xFF9900, 0.25, 0.125 );
vecteur(scene,A,B0, 0xFF9900, 0.25, 0.125 );
vecteur(scene,O3,B0, 0xAAAAAA, 0.25, 0.125 );
vecteur(scene,B0,B, 0x555555, 0.25, 0.125 );
```

Vecteurs comme « barycentre »

## Vecteur comme soustraction de deux vecteurs

Définition (Méthode `subVectors`)

Soit  $\underline{u}$  (resp.  $\underline{v}$ ) une instance de `Vector $n$`  représentant le vecteur  $\overrightarrow{u}$  (resp.  $\overrightarrow{v}$ ).

## Vecteur comme soustraction de deux vecteurs

Définition (Méthode `subVectors`)

Soit  $\vec{u}$  (resp.  $\vec{v}$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

Soit  $\vec{w}$  une instance de `Vector $n$`  représentant le vecteur  $\vec{w} = \vec{u} - \vec{v}$ .

## Vecteur comme soustraction de deux vecteurs

Définition (Méthode `subVectors`)

Soit  $\vec{u}$  (resp.  $\vec{v}$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

Soit  $\vec{w}$  une instance de `Vector $n$`  représentant le vecteur  $\vec{w} = \vec{u} - \vec{v}$ .

La syntaxe est `w.subVectors( u, v );`

## Vecteur comme soustraction de deux vecteurs

Définition (Méthode `subVectors`)

Soit  $u$  (resp.  $v$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

Soit  $w$  une instance de `Vector $n$`  représentant le vecteur  $\vec{w} = \vec{u} - \vec{v}$ .

La syntaxe est `w.subVectors(u, v);`

```
let o3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(-1,1,0);
let v = new THREE.Vector3(0,0,-1);
vecteur(scene,o3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,o3,v, 0x00FFFF, 0.25, 0.125 );
let w = new THREE.Vector3(0,0,0);
```

## Vecteur comme soustraction de deux vecteurs

Définition (Méthode `subVectors`)

Soit  $u$  (resp.  $v$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

Soit  $w$  une instance de `Vector $n$`  représentant le vecteur  $\vec{w} = \vec{u} - \vec{v}$ .

La syntaxe est `w.subVectors(u, v);`

```
let o3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(-1,1,0);
let v = new THREE.Vector3(0,0,-1);
vecteur(scene,o3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,o3,v, 0x00FFFF, 0.25, 0.125 );
let w = new THREE.Vector3(0,0,0);
w.subVectors(u,v);
vecteur(scene,u,w, 0x444444, 0.35, 0.25 ); //vecteur -v
vecteur(scene,o3,w, 0xFF9900, 0.25, 0.125 );
```

Vecteur comme soustraction de deux vecteurs

## Egalité de deux vecteurs

### Définition (Test d'égalité)

Soit `TotoA` (resp. `TotoB`) une instance de `Vector $n$`  représentant le point A (resp. B).

## Egalité de deux vecteurs

### Définition (Test d'égalité)

Soit `TotoA` (resp. `TotoB`) une instance de `Vector $n$`  représentant le point  $A$  (resp.  $B$ ).

Soit `TotoU` (resp. `TotoV`) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

## Égalité de deux vecteurs

## Définition (Test d'égalité)

Soit `TotoA` (resp. `TotoB`) une instance de `Vector $n$`  représentant le point  $A$  (resp.  $B$ ).

Soit `TotoU` (resp. `TotoV`) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

- La syntaxe `A.equals(B)` retourne vrai si  $A = B$  et faux si  $A \neq B$ ;

## Egalité de deux vecteurs

## Définition (Test d'égalité)

Soit `TotoA` (resp. `TotoB`) une instance de `Vector $n$`  représentant le point  $A$  (resp.  $B$ ).

Soit `TotoU` (resp. `TotoV`) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

- La syntaxe `A.equals(B)`; retourne vrai si  $A = B$  et faux si  $A \neq B$ ;
- La syntaxe `A.equals(B)`; retourne vrai si  $\vec{u} = \vec{v}$  et faux si  $\vec{u} \neq \vec{v}$ .

## Egalité de deux vecteurs

### Définition (Test d'égalité)

Soit `TotoA` (resp. `TotoB`) une instance de `Vector $n$`  représentant le point  $A$  (resp.  $B$ ).

Soit `TotoU` (resp. `TotoV`) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

- La syntaxe `A.equals(B)`; retourne vrai si  $A = B$  et faux si  $A \neq B$ ;
- La syntaxe `A.equals(B)`; retourne vrai si  $\vec{u} = \vec{v}$  et faux si  $\vec{u} \neq \vec{v}$ .

### Danger

Il n'est pas possible, du point de vue sémantique, de réaliser un test d'égalité entre un point et un vecteur :

## Egalité de deux vecteurs

### Définition (Test d'égalité)

Soit `TotoA` (resp. `TotoB`) une instance de `Vector $n$`  représentant le point  $A$  (resp.  $B$ ).

Soit `TotoU` (resp. `TotoV`) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

- La syntaxe `A.equals(B)`; retourne vrai si  $A = B$  et faux si  $A \neq B$ ;
- La syntaxe `A.equals(B)`; retourne vrai si  $\vec{u} = \vec{v}$  et faux si  $\vec{u} \neq \vec{v}$ .

### Danger

**Il n'est pas possible, du point de vue sémantique, de réaliser un test d'égalité entre un point et un vecteur** : un point appartient à un espace affine alors qu'un vecteur appartient à un espace vectoriel.

Test d'égalité entre points ou entre vecteurs

## Égalité de deux vecteurs

```
//definition des trois vecteurs u, v et w
let u = new THREE.Vector3(1,-1,0);
let v = new THREE.Vector3(1,-1,0);
let w = new THREE.Vector3(1,1,0);
```

Test d'égalité entre points ou entre vecteurs

## Égalité de deux vecteurs

```
//definition des trois vecteurs u, v et w
let u = new THREE.Vector3(1,-1,0);
let v = new THREE.Vector3(1,-1,0);
let w = new THREE.Vector3(1,1,0);
//definition des trois points A, B et C
let A = new THREE.Vector3(1,-1,1);
let B = new THREE.Vector3(1,-1,1);
let C = new THREE.Vector3(1,1,1);
let O3 = new THREE.Vector3(0,0,0);
```

Test d'égalité entre points ou entre vecteurs

## Egalité de deux vecteurs

```
//definition des trois vecteurs u, v et w
let u = new THREE.Vector3(1,-1,0);
let v = new THREE.Vector3(1,-1,0);
let w = new THREE.Vector3(1,1,0);
//definition des trois points A, B et C
let A = new THREE.Vector3(1,-1,1);
let B = new THREE.Vector3(1,-1,1);
let C = new THREE.Vector3(1,1,1);
let O3 = new THREE.Vector3(0,0,0);
let dimPt = 0.05;
//Affichage des trois vecteurs u, v et w dans la scene
vecteur(scene,O3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,O3,v, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,O3,w, 0xFF00FF, 0.25, 0.125 );
//Affichage des trois points A, B et C dans la scene
tracePt(scene, A, "#AA0000",dimPt,true);
tracePt(scene, B, "#00AA00",dimPt,true);
tracePt(scene, C, "#0000AA",dimPt,true);
```

Test d'égalité entre points ou entre vecteurs

## Égalité de deux vecteurs

```
let dimPt = 0.05;  
//Affichage des trois vecteurs u, v et w dans la scene  
vecteur(scene, 03, u, 0xFF00FF, 0.25, 0.125 );  
vecteur(scene, 03, v, 0xFF00FF, 0.25, 0.125 );  
vecteur(scene, 03, w, 0xFF00FF, 0.25, 0.125 );  
//Affichage des trois points A, B et C dans la scene  
tracePt(scene, A, "#AA0000", dimPt, true);  
tracePt(scene, B, "#00AA00", dimPt, true);  
tracePt(scene, C, "#0000AA", dimPt, true);  
//Pour l'affichage des points et vecteurs  
let mes = "u("+u.x+";"+u.y+";"+u.z+"); "  
mes += "v("+v.x+";"+v.y+";"+v.z+"); "  
mes += "w("+w.x+";"+w.y+";"+w.z+"); "  
mes += "<br />A("+A.x+";"+A.y+";"+A.z+"); "  
mes += "B("+B.x+";"+B.y+";"+B.z+"); "  
mes += "C("+C.x+";"+C.y+";"+C.z+"); "
```

Test d'égalité entre points ou entre vecteurs

## Égalité de deux vecteurs

```
//Pour l'affichage des points et vecteurs
let mes = "u(\"+u.x+\";\"+u.y+\";\"+u.z+\"); "
mes += "v(\"+v.x+\";\"+v.y+\";\"+v.z+\"); ";
mes += "w(\"+w.x+\";\"+w.y+\";\"+w.z+\"); "
mes += "<br />A(\"+A.x+\";\"+A.y+\";\"+A.z+\"); "
mes += "B(\"+B.x+\";\"+B.y+\";\"+B.z+\"); ";
mes += "C(\"+C.x+\";\"+C.y+\";\"+C.z+\"); "
//Pour l'affichage des tests
mes += '<br /><span id="true">u.equals(v) : '+u.equals(v)+'
    '</span>;    u.equals(w) : <span id="false">' +u.equals(w)+'
    '</span>';
mes += '<br /><span id="true">A.equals(B) : '+A.equals(B)+'
    '</span>;    A.equals(C) : <span id="false">' +A.equals(C)+'
    '</span>';
document.getElementById("result").innerHTML +=mes;
```

Test d'égalité entre points ou entre vecteurs

## Produit scalaire dans une base orthonormée

## Définition (Produit scalaire de deux vecteurs)

Soit  $vU$  (resp.  $vV$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

## Produit scalaire dans une base orthonormée

### Définition (Produit scalaire de deux vecteurs)

Soit  $vU$  (resp.  $vV$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

La syntaxe `vU.dot(vV);` retourne le produit scalaire  $\vec{u} \cdot \vec{v}$ .

## Produit scalaire dans une base orthonormée

## Définition (Produit scalaire de deux vecteurs)

Soit  $vU$  (resp.  $vV$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

La syntaxe `vU.dot(vV);` retourne le produit scalaire  $\vec{u} \cdot \vec{v}$ .

## Remarque

Soit  $\vec{u}(x_0; y_0; z_0)$  dans la base orthonormée  $(\vec{i}; \vec{j}; \vec{k})$ .

## Produit scalaire dans une base orthonormée

## Définition (Produit scalaire de deux vecteurs)

Soit  $vU$  (resp.  $vV$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

La syntaxe `vU.dot(vV);` retourne le produit scalaire  $\vec{u} \cdot \vec{v}$ .

## Remarque

Soit  $\vec{u}(x_0; y_0; z_0)$  dans la base orthonormée  $(\vec{i}; \vec{j}; \vec{k})$ . Alors :

- $\vec{u} \cdot \vec{i} = x_0$  ;

## Produit scalaire dans une base orthonormée

## Définition (Produit scalaire de deux vecteurs)

Soit  $vU$  (resp.  $vV$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

La syntaxe `vU.dot(vV)`; retourne le produit scalaire  $\vec{u} \cdot \vec{v}$ .

## Remarque

Soit  $\vec{u}(x_0; y_0; z_0)$  dans la base orthonormée  $(\vec{i}; \vec{j}; \vec{k})$ . Alors :

- $\vec{u} \cdot \vec{i} = x_0$  ;
- $\vec{u} \cdot \vec{j} = y_0$  ;

## Produit scalaire dans une base orthonormée

## Définition (Produit scalaire de deux vecteurs)

Soit  $vU$  (resp.  $vV$ ) une instance de `Vector $n$`  représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

La syntaxe `vU.dot(vV)`; retourne le produit scalaire  $\vec{u} \cdot \vec{v}$ .

## Remarque

Soit  $\vec{u}(x_0; y_0; z_0)$  dans la base orthonormée  $(\vec{i}; \vec{j}; \vec{k})$ . Alors :

- $\vec{u} \cdot \vec{i} = x_0$  ;
- $\vec{u} \cdot \vec{j} = y_0$  ;
- $\vec{u} \cdot \vec{k} = z_0$ .

Produit scalaire de deux vecteurs

## Produit scalaire dans une base orthonormée

```
let O3 = new THREE.Vector3(0,0,0);
//definition et affichage des trois vecteurs u, v et w
let u = new THREE.Vector3(1,-1,0);
let v = new THREE.Vector3(1,1,0);
let w = new THREE.Vector3(-1.5,-0.75,0.75);
vecteur(scene,O3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,O3,v, 0x00FFFF, 0.25, 0.125 );
vecteur(scene,O3,w, 0xFF9900, 0.25, 0.125 );
```

## Produit scalaire dans une base orthonormée

```
let O3 = new THREE.Vector3(0,0,0);
//definition et affichage des trois vecteurs u, v et w
let u = new THREE.Vector3(1,-1,0);
let v = new THREE.Vector3(1,1,0);
let w = new THREE.Vector3(-1.5,-0.75,0.75);
vecteur(scene,O3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,O3,v, 0x00FFFF, 0.25, 0.125 );
vecteur(scene,O3,w, 0xFF9900, 0.25, 0.125 );
// Composantes des vecteurs
let mes = "u("+u.x+";"+u.y+";"+u.z+"); "
mes += "v("+v.x+";"+v.y+";"+v.z+"); ";
mes += "      w("+w.x+";"+w.y+";"+w.z+"); "
```

## Produit scalaire dans une base orthonormée

```
let O3 = new THREE.Vector3(0,0,0);
//definition et affichage des trois vecteurs u, v et w
let u = new THREE.Vector3(1,-1,0);
let v = new THREE.Vector3(1,1,0);
let w = new THREE.Vector3(-1.5,-0.75,0.75);
vecteur(scene,O3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,O3,v, 0x00FFFF, 0.25, 0.125 );
vecteur(scene,O3,w, 0xFF9900, 0.25, 0.125 );
// Composantes des vecteurs
let mes = "u("+u.x+";"+u.y+";"+u.z+"); "
mes += "v("+v.x+";"+v.y+";"+v.z+"); ";
mes += "      w("+w.x+";"+w.y+";"+w.z+"); ";
//calculs des produits scalaires
mes += '<br /> u.dot(v) : '+u.dot(v);
mes += '<br /> u.dot(w) : '+u.dot(w);
document.getElementById("result").innerHTML +=mes;
```

Produit scalaire de deux vecteurs

## 2 Points et Vecteurs

### Définitions

- Méthodes communes à THREE.Vector $n$ ,  $n \in \{2; 3\}$
- **Cas du plan i.e.  $n = 2$**
- Cas de l'espace i.e.  $n = 3$

Déterminant de deux vecteurs du plan muni d'une base orthonormée directe ( $\vec{i}; \vec{j}$ )Définition (Déterminant de deux vecteurs du plan vectoriel, méthode `cross`)

Soit  $\text{vU}$  (resp.  $\text{vV}$ ) une instance de `Vector2` représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

Déterminant de deux vecteurs du plan muni d'une base orthonormée directe ( $\vec{i}; \vec{j}$ )

Définition (Déterminant de deux vecteurs du plan vectoriel, méthode `cross`)

Soit  $vU$  (resp.  $vV$ ) une instance de `Vector2` représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

La syntaxe `vU.cross(vV);` retourne le déterminant  $\det(\vec{u}; \vec{v})$ .

Déterminant de deux vecteurs du plan muni d'une base orthonormée directe ( $\vec{i}; \vec{j}$ )

Définition (Déterminant de deux vecteurs du plan vectoriel, méthode `cross`)

Soit  $vU$  (resp.  $vV$ ) une instance de `Vector2` représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

La syntaxe `vU.cross(vV);` retourne le déterminant  $\det(\vec{u}; \vec{v})$ .

Rappel

Soit  $\vec{u}(2; -2)$  et  $\vec{v}(2; 2)$  dans la base orthonormée directe ( $\vec{i}; \vec{j}$ ).

Déterminant de deux vecteurs du plan muni d'une base orthonormée directe ( $\vec{i}; \vec{j}$ )Définition (Déterminant de deux vecteurs du plan vectoriel, méthode `cross`)

Soit  $vU$  (resp.  $vV$ ) une instance de `Vector2` représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

La syntaxe `vU.cross(vV);` retourne le déterminant  $\det(\vec{u}; \vec{v})$ .

## Rappel

Soit  $\vec{u}(2; -2)$  et  $\vec{v}(2; 2)$  dans la base orthonormée directe ( $\vec{i}; \vec{j}$ ). Alors :

- $\det(\vec{u}; \vec{v}) = \begin{vmatrix} 2 & 2 \\ -2 & 2 \end{vmatrix} = 4 - (-4) = 8 ;$

Déterminant de deux vecteurs du plan muni d'une base orthonormée directe ( $\vec{i}; \vec{j}$ )

Définition (Déterminant de deux vecteurs du plan vectoriel, méthode `cross`)

Soit  $vU$  (resp.  $vV$ ) une instance de `Vector2` représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

La syntaxe `vU .cross ( vV )`; retourne le déterminant  $\det(\vec{u}; \vec{v})$ .

## Rappel

Soit  $\vec{u}(2; -2)$  et  $\vec{v}(2; 2)$  dans la base orthonormée directe ( $\vec{i}; \vec{j}$ ). Alors :

- $\bullet \det(\vec{u}; \vec{v}) = \begin{vmatrix} 2 & 2 \\ -2 & 2 \end{vmatrix} = 4 - (-4) = 8 ;$
- $\bullet \det(\vec{v}; \vec{u}) = \begin{vmatrix} 2 & 2 \\ 2 & -2 \end{vmatrix} = -4 - 4 = -8 ;$

Déterminant de deux vecteurs du plan muni d'une base orthonormée directe ( $\vec{i}; \vec{j}$ )

Définition (Déterminant de deux vecteurs du plan vectoriel, méthode `cross`)

Soit  $vU$  (resp.  $vV$ ) une instance de `Vector2` représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

La syntaxe `vU .cross ( vV )`; retourne le déterminant  $\det(\vec{u}; \vec{v})$ .

## Rappel

Soit  $\vec{u}(2; -2)$  et  $\vec{v}(2; 2)$  dans la base orthonormée directe ( $\vec{i}; \vec{j}$ ). Alors :

- $\det(\vec{u}; \vec{v}) = \begin{vmatrix} 2 & 2 \\ -2 & 2 \end{vmatrix} = 4 - (-4) = 8 ;$
- $\det(\vec{v}; \vec{u}) = \begin{vmatrix} 2 & 2 \\ 2 & -2 \end{vmatrix} = -4 - 4 = -8 ;$
- $\det(\vec{u}; \vec{u}) = \begin{vmatrix} 2 & 2 \\ -2 & -2 \end{vmatrix} = -4 - (-4) = 0.$

Déterminant de deux vecteurs

## Angle d'un vecteur avec le premier vecteur de base

Soit  $(\vec{i}; \vec{j})$  une base orthonormée directe du plan vectoriel  $\vec{\mathcal{P}}$ .

## Angle d'un vecteur avec le premier vecteur de base

Soit  $(\vec{i}; \vec{j})$  une base orthonormée directe du plan vectoriel  $\vec{\mathcal{P}}$ .

Définition (Angle d'un vecteur du plan vectoriel, méthode `angle`)

Soit `vU` une instance de `Vector2` représentant le vecteur  $\vec{u}$ .

## Angle d'un vecteur avec le premier vecteur de base

Soit  $(\vec{i}; \vec{j})$  une base orthonormée directe du plan vectoriel  $\vec{\mathcal{P}}$ .

Définition (Angle d'un vecteur du plan vectoriel, méthode `angle`)

Soit `vU` une instance de `Vector2` représentant le vecteur  $\vec{u}$ .

La syntaxe `vU.angle()` retourne une mesure, en radian dans l'intervalle  $[0; 2\pi[$ , de l'angle orienté  $(\vec{i}; \vec{u})$ .

Angle entre un vecteur et le vecteur unitaire de l'axe des abscisses

## Rotation plane

Soit  $(\vec{i}; \vec{j})$  une base orthonormée directe du plan vectoriel  $\vec{\mathcal{P}}$ .

## Rotation plane

Soit  $(\vec{i}; \vec{j})$  une base orthonormée directe du plan vectoriel  $\vec{\mathcal{P}}$ .

Définition (Rotation plane avec `rotateAround`)

Soit `Toto` une instance de `Vector2` représentant le point A ou le vecteur  $\vec{u}$ .

## Rotation plane

Soit  $(\vec{i}; \vec{j})$  une base orthonormée directe du plan vectoriel  $\vec{\mathcal{P}}$ .

Définition (Rotation plane avec `rotateAround`)

Soit `Toto` une instance de `Vector2` représentant le point A ou le vecteur  $\vec{u}$ .

Soit `Or` une instance de `Vector2` représentant le centre de rotation  $\Omega$  lors de la rotation affine A ou le vecteur  $\vec{0}$  lors d'une rotation vectorielle.

## Rotation plane

Soit  $(\vec{i}; \vec{j})$  une base orthonormée directe du plan vectoriel  $\vec{\mathcal{P}}$ .

Définition (Rotation plane avec `rotateAround`)

Soit `Toto` une instance de `Vector2` représentant le point  $A$  ou le vecteur  $\vec{u}$ .

Soit `Or` une instance de `Vector2` représentant le centre de rotation  $\Omega$  lors de la rotation affine  $A$  ou le vecteur  $\vec{O}$  lors d'une rotation vectorielle.

Soit `Theta` une variable contenant l'angle de rotation  $\theta$  exprimé en radian.

## Rotation plane

Soit  $(\vec{i}; \vec{j})$  une base orthonormée directe du plan vectoriel  $\vec{\mathcal{P}}$ .

### Définition (Rotation plane avec `rotateAround`)

Soit `Toto` une instance de `Vector2` représentant le point  $A$  ou le vecteur  $\vec{u}$ .

Soit `Or` une instance de `Vector2` représentant le centre de rotation  $\Omega$  lors de la rotation affine  $A$  ou le vecteur  $\vec{0}$  lors d'une rotation vectorielle.

Soit `Theta` une variable contenant l'angle de rotation  $\theta$  exprimé en radian.

La syntaxe `Toto.rotateAround(Or, Theta);` permet d'obtenir l'image du point  $A$  par la rotation affine de centre  $\Omega$  et d'angle  $\theta$  ou du vecteur  $\vec{u}$  par la rotation vectorielle d'angle  $\theta$ .

## Rotation plane

Soit  $(\vec{i}; \vec{j})$  une base orthonormée directe du plan vectoriel  $\vec{\mathcal{P}}$ .

### Définition (Rotation plane avec `rotateAround`)

Soit `Toto` une instance de `Vector2` représentant le point  $A$  ou le vecteur  $\vec{u}$ .

Soit `Or` une instance de `Vector2` représentant le centre de rotation  $\Omega$  lors de la rotation affine  $A$  ou le vecteur  $\vec{0}$  lors d'une rotation vectorielle.

Soit `Theta` une variable contenant l'angle de rotation  $\theta$  exprimé en radian.

La syntaxe `Toto.rotateAround(Or, Theta);` permet d'obtenir l'image du point  $A$  par la rotation affine de centre  $\Omega$  et d'angle  $\theta$  ou du vecteur  $\vec{u}$  par la rotation vectorielle d'angle  $\theta$ .

### Danger

Lors de l'emploi de `Toto.rotateAround(Or, Theta);`, la valeur initiale de `Toto` est perdue.

Rotation plane

## Rotation plane

Rotation affine :

```
//centre de la rotation affine  
let Or = new THREE.Vector2(1,1);  
let A = new THREE.Vector2(2,1);
```

## Rotation plane

Rotation affine :

```
//centre de la rotation affine
let Or = new THREE.Vector2(1,1);
let A = new THREE.Vector2(2,1);
//copie du point initial
let A1 = A.clone();
```

## Rotation plane

Rotation affine :

```
//centre de la rotation affine
let Or = new THREE.Vector2(1,1);
let A = new THREE.Vector2(2,1);
//copie du point initial
let A1 = A.clone();
//rotation affine, de centre Or, du point A1
A1.rotateAround(Or,-Math.PI/3);
```

## Rotation plane

Rotation affine :

```
//centre de la rotation affine
let Or = new THREE.Vector2(1,1);
let A = new THREE.Vector2(2,1);
//copie du point initial
let A1 = A.clone();
//rotation affine, de centre Or, du point A1
A1.rotateAround(Or,-Math.PI/3);
```

Rotation vectorielle :

```
//Vecteur nul pour la rotation vectorielle
let O2 = new THREE.Vector2(0,0);
let u = new THREE.Vector2(2,-2);
```

## Rotation plane

Rotation affine :

```
//centre de la rotation affine
let Or = new THREE.Vector2(1,1);
let A = new THREE.Vector2(2,1);
//copie du point initial
let A1 = A.clone();
//rotation affine, de centre Or, du point A1
A1.rotateAround(Or,-Math.PI/3);
```

Rotation vectorielle :

```
//Vecteur nul pour la rotation vectorielle
let O2 = new THREE.Vector2(0,0);
let u = new THREE.Vector2(2,-2);
//copie du vecteur initial
let u1 = u.clone();
```

## Rotation plane

Rotation affine :

```
//centre de la rotation affine
let Or = new THREE.Vector2(1,1);
let A = new THREE.Vector2(2,1);
//copie du point initial
let A1 = A.clone();
//rotation affine, de centre Or, du point A1
A1.rotateAround(Or,-Math.PI/3);
```

Rotation vectorielle :

```
//Vecteur nul pour la rotation vectorielle
let O2 = new THREE.Vector2(0,0);
let u = new THREE.Vector2(2,-2);
//copie du vecteur initial
let u1 = u.clone();
//rotation vectorielle du vecteur position u1
// O2 represente le vecteur nul
u1.rotateAround(O2,-5*Math.PI/6);
```

Rotation plane

## 2 Points et Vecteurs

### Définitions

- Méthodes communes à THREE.Vector $n$ ,  $n \in \{2; 3\}$
- Cas du plan i.e.  $n = 2$
- Cas de l'espace i.e.  $n = 3$

## Produit vectoriel avec écrasement du premier vecteur

Soit  $(\vec{i}; \vec{j}; \vec{k})$  une base orthonormée directe de l'espace vectoriel  $\vec{\mathcal{E}}_3$ .

## Produit vectoriel avec écrasement du premier vecteur

Soit  $(\vec{i}; \vec{j}; \vec{k})$  une base orthonormée directe de l'espace vectoriel  $\vec{\mathcal{E}}_3$ .

Définition (Produit vectoriel, méthode `cross`)

Soit  $vU$  (resp.  $vV$ ) une instance de `Vector3` représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

## Produit vectoriel avec écrasement du premier vecteur

Soit  $(\vec{i}; \vec{j}; \vec{k})$  une base orthonormée directe de l'espace vectoriel  $\vec{\mathcal{E}}_3$ .

Définition (Produit vectoriel, méthode `cross`)

Soit  $vU$  (resp.  $vV$ ) une instance de `Vector3` représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

La syntaxe `vU.cross(vV);` permet de stocker dans  $vU$  le vecteur  $\vec{u} \wedge \vec{v}$  et la valeur de  $vU$  est écrasée.

## Produit vectoriel avec écrasement du premier vecteur

Soit  $(\vec{i}; \vec{j}; \vec{k})$  une base orthonormée directe de l'espace vectoriel  $\vec{\mathcal{E}}_3$ .

Définition (Produit vectoriel, méthode `cross`)

Soit  $vU$  (resp.  $vV$ ) une instance de `Vector3` représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

La syntaxe `vU.cross(vV);` permet de stocker dans  $vU$  le vecteur  $\vec{u} \wedge \vec{v}$  et la valeur de  $vU$  est écrasée.

```
let o3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(-1.5,-1,0);
let v = new THREE.Vector3(0,.5,-0.5);
vecteur(scene,o3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,o3,v, 0x00FFFF, 0.25, 0.125 );
```

## Produit vectoriel avec écrasement du premier vecteur

Soit  $(\vec{i}; \vec{j}; \vec{k})$  une base orthonormée directe de l'espace vectoriel  $\vec{\mathcal{E}}_3$ .

Définition (Produit vectoriel, méthode `cross`)

Soit  $vU$  (resp.  $vV$ ) une instance de `Vector3` représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

La syntaxe `vU.cross(vV);` permet de stocker dans  $vU$  le vecteur  $\vec{u} \wedge \vec{v}$  et la valeur de  $vU$  est écrasée.

```
let o3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(-1.5,-1,0);
let v = new THREE.Vector3(0,.5,-0.5);
vecteur(scene,o3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,o3,v, 0x00FFFF, 0.25, 0.125 );
u.cross(v);
vecteur(scene,o3,u, 0xFF9900, 0.25, 0.125 );
```

Produit vectoriel avec écrasement du premier vecteur

## Produit vectoriel en gardant le premier vecteur

Soit  $(\vec{i}; \vec{j}; \vec{k})$  une base orthonormée directe de l'espace vectoriel  $\vec{\mathcal{E}}_3$ .

## Produit vectoriel en gardant le premier vecteur

Soit  $(\vec{i}; \vec{j}; \vec{k})$  une base orthonormée directe de l'espace vectoriel  $\vec{\mathcal{E}}_3$ .

Définition (Produit vectoriel, méthode `crossVectors`)

Soit  $vU$  (resp.  $vV$ ) une instance de `Vector3` représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ).

## Produit vectoriel en gardant le premier vecteur

Soit  $(\vec{i}; \vec{j}; \vec{k})$  une base orthonormée directe de l'espace vectoriel  $\vec{\mathcal{E}}_3$ .

Définition (Produit vectoriel, méthode `crossVectors`)

Soit  $vU$  (resp.  $vV$ ) une instance de `Vector3` représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ). Soit  $vW$  une instance de `Vector3` représentant le vecteur  $\vec{w}$ .

## Produit vectoriel en gardant le premier vecteur

Soit  $(\vec{i}; \vec{j}; \vec{k})$  une base orthonormée directe de l'espace vectoriel  $\vec{\mathcal{E}}_3$ .

Définition (Produit vectoriel, méthode `crossVectors`)

Soit  $vU$  (resp.  $vV$ ) une instance de `Vector3` représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ). Soit  $vW$  une instance de `Vector3` représentant le vecteur  $\vec{w}$ .

La syntaxe `vW.crossVectors(vU, vV);` permet de stocker dans  $vW$  le vecteur  $\vec{w} = \vec{u} \wedge \vec{v}$ .

## Produit vectoriel en gardant le premier vecteur

Soit  $(\vec{i}; \vec{j}; \vec{k})$  une base orthonormée directe de l'espace vectoriel  $\vec{\mathcal{E}}_3$ .

Définition (Produit vectoriel, méthode `crossVectors`)

*Soit  $vU$  (resp.  $vV$ ) une instance de `Vector3` représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ). Soit  $vW$  une instance de `Vector3` représentant le vecteur  $\vec{w}$ .*

*La syntaxe `vW.crossVectors(vU, vV);` permet de stocker dans  $vW$  le vecteur  $\vec{w} = \vec{u} \wedge \vec{v}$ .*

```
let o3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(-1.5,-1,0);
let v = new THREE.Vector3(0,.5,-0.5);
let w = new THREE.Vector3(0,0,0);
vecteur(scene,o3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,o3,v, 0x00FFFF, 0.25, 0.125 );
```

## Produit vectoriel en gardant le premier vecteur

Soit  $(\vec{i}; \vec{j}; \vec{k})$  une base orthonormée directe de l'espace vectoriel  $\vec{\mathcal{E}}_3$ .

Définition (Produit vectoriel, méthode `crossVectors()`)

Soit  $vU$  (resp.  $vV$ ) une instance de `Vector3` représentant le vecteur  $\vec{u}$  (resp.  $\vec{v}$ ). Soit  $vW$  une instance de `Vector3` représentant le vecteur  $\vec{w}$ .

La syntaxe `vW.crossVectors(vU, vV);` permet de stocker dans `vW` le vecteur  $\vec{w} = \vec{u} \wedge \vec{v}$ .

```
let o3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(-1.5,-1,0);
let v = new THREE.Vector3(0,.5,-0.5);
let w = new THREE.Vector3(0,0,0);
vecteur(scene,o3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,o3,v, 0x00FFFF, 0.25, 0.125 );
w.crossVectors(u,v);
vecteur(scene,o3,w, 0xFF9900, 0.25, 0.125 );
```

Produit vectoriel en gardant le premier vecteur

## Rotation 3d

Définition (Rotation 3d avec `applyAxisAngle`)

Soit `Toto` une instance de `Vector3` représentant le point  $A$  ou le vecteur  $\vec{v}$ .

## Rotation 3d

Définition (Rotation 3d avec `applyAxisAngle`)

Soit `Toto` une instance de `Vector3` représentant le point  $A$  ou le vecteur  $\vec{v}$ .

Soit `vU` une instance de `Vector3` représentant le vecteur unitaire  $\vec{u}$ .

## Rotation 3d

Définition (Rotation 3d avec `applyAxisAngle`)

Soit `Toto` une instance de `Vector3` représentant le point  $A$  ou le vecteur  $\vec{v}$ .

Soit `vU` une instance de `Vector3` représentant le vecteur unitaire  $\vec{u}$ .

Soit `Theta` une variable contenant l'angle de rotation  $\theta$  exprimé en radian.

La syntaxe `Toto .applyAxisAngle ( vU , Theta );` permet d'obtenir l'image, après écrasement de la valeur de `Toto`,

## Rotation 3d

Définition (Rotation 3d avec `applyAxisAngle`)

Soit `Toto` une instance de `Vector3` représentant le point  $A$  ou le vecteur  $\vec{v}$ .

Soit `vU` une instance de `Vector3` représentant le vecteur unitaire  $\vec{u}$ .

Soit `Theta` une variable contenant l'angle de rotation  $\theta$  exprimé en radian.

La syntaxe `Toto .applyAxisAngle ( vU , Theta );` permet d'obtenir l'image, après écrasement de la valeur de `Toto`, de :

- soit du point  $A$  par la rotation affine d'axe  $(O_3; \vec{u})$  et d'angle  $\theta$  ;

## Rotation 3d

Définition (Rotation 3d avec `applyAxisAngle`)

Soit `Toto` une instance de `Vector3` représentant le point  $A$  ou le vecteur  $\vec{v}$ .

Soit `vU` une instance de `Vector3` représentant le vecteur unitaire  $\vec{u}$ .

Soit `Theta` une variable contenant l'angle de rotation  $\theta$  exprimé en radian.

La syntaxe `Toto .applyAxisAngle ( vU , Theta );` permet d'obtenir l'image, après écrasement de la valeur de `Toto`, de :

- soit du point  $A$  par la rotation affine d'axe ( $O_3; \vec{u}$ ) et d'angle  $\theta$  ;
- soit du vecteur  $\vec{v}$  par la rotation vectorielle de droite vectorielle orientée par le vecteur  $\vec{u}$  et d'angle  $\theta$ .

Rotation 3d vectorielle

Rotation 3d affine

## Rotation vectorielle 3d

```
let o3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(1,1,0);
```

## Rotation vectorielle 3d

```
let O3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(1,1,0);
u.normalize();
```

## Rotation vectorielle 3d

```
let o3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(1,1,0);
u.normalize();
let v = new THREE.Vector3(-0.75,0.75,1);
let v0 = v.clone();
```

## Rotation vectorielle 3d

```
let O3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(1,1,0);
u.normalize();
let v = new THREE.Vector3(-0.75,0.75,1);
let v0 = v.clone();
vecteur(scene,O3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,O3,v, 0x00FFFF, 0.25, 0.125 );
```

## Rotation vectorielle 3d

```
let O3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(1,1,0);
u.normalize();
let v = new THREE.Vector3(-0.75,0.75,1);
let v0 = v.clone();
vecteur(scene,O3,u, 0xFF00FF, 0.25, 0.125 );
vecteur(scene,O3,v, 0x00FFFF, 0.25, 0.125 );
// image de v par la rotation vectorielle
v.applyAxisAngle(u,Math.PI/2);
vecteur(scene,O3,v, 0xFF9900, 0.25, 0.125 );
```

Rotation 3d vectorielle

## Rotation affine 3d

```
let o3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(1,1,0);
```

## Rotation affine 3d

```
let O3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(1,1,0);
u.normalize();
```

## Rotation affine 3d

```
let O3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(1,1,0);
u.normalize();
let dimPt = 0.025;
let A = new THREE.Vector3(1.5,0,0);
tracePt(scene, A, "#00FFFF",dimPt,true);
vecteur(scene,O3,u, 0xFF00FF, 0.25, 0.125 );
```

## Rotation affine 3d

```
let O3 = new THREE.Vector3(0,0,0);
let u = new THREE.Vector3(1,1,0);
u.normalize();
let dimPt = 0.025;
let A = new THREE.Vector3(1.5,0,0);
tracePt(scene, A, "#00FFFF",dimPt,true);
vecteur(scene,O3,u, 0xFF00FF, 0.25, 0.125 );
// image de A par la rotation affine
A.applyAxisAngle(u,Math.PI/2);
tracePt(scene, A, "#FF9900",dimPt,true);
```

Rotation 3d affine

**1** HTML et Three.js**2** Points et Vecteurs

- Définitions
- Méthodes communes à THREE.Vector $n$ ,  $n \in \{2;3\}$
- Cas du plan i.e.  $n = 2$
- Cas de l'espace i.e.  $n = 3$

**3** La caméra**4** Les lumières

- Lumière ambiante
- Définition et propriétés communes des autres lumières
- Lumière ponctuelle et spot
- Spot et lumière directionnelle : ombre
- Le spot

**5** Les matériaux**6** Les courbes

- Cas classique
- Cas spécifique de THREE.js
- Courbes de Bézier polynomiales

**7** Les surfaces primitives

- Le principe
- Cas particulier des surfaces planes
- Les surfaces non planes

**8** Transformations géométriques de  $\mathcal{E}_3$ 

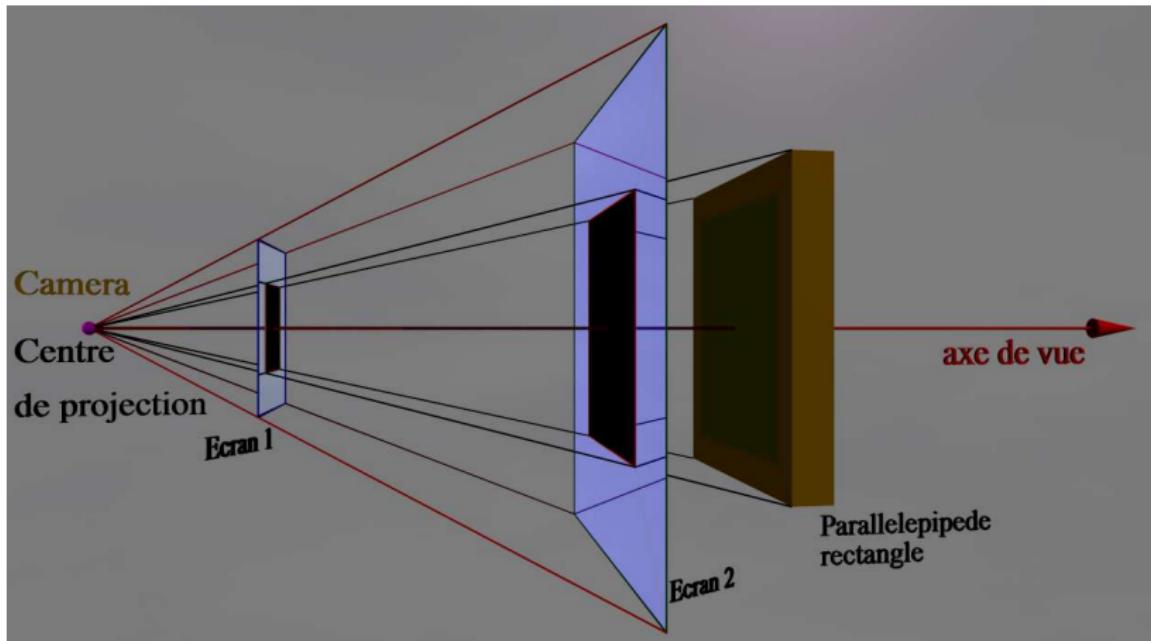
- Application affine : translation
- Application affine ou vectorielle

**9** L'Animation**10** Création de menu G.U.I.

- Théorie
- Exemple avec un plan

**11** Le C.S.G.

## Caméra perspective : projection centrale



**Deux droites parallèles sont sécantes (à l'infini)** : cf. les rails de chemins de fer.

## Définition d'une caméra perspective via une fonction

## Définition (Fonction définissant une caméra (perspective))

*Utilisation d'une fonction pour éviter les copier-coller :*

```
function cameraLumiere(scene,camera){ // creation de la camera}
```

## Définition d'une caméra perspective via une fonction

## Définition (Fonction définissant une caméra (perspective))

*Utilisation d'une fonction pour éviter les copier-coller :*

```
function cameraLumiere(scene,camera){    // creation de la camera  
    camera.up = new THREE.Vector3( 0, 0, 1 ); // axe des cotes vertical
```

## Définition d'une caméra perspective via une fonction

## Définition (Fonction définissant une caméra (perspective))

*Utilisation d'une fonction pour éviter les copier-coller :*

```
function cameraLumiere(scene,camera){    // creation de la camera
    camera.up = new THREE.Vector3( 0, 0, 1 ); // axe des cotes vertical
    let xPos= 6 , yPos= 6 , zPos= 5 , xDir= 0 , yDir= 0 , zDir= 0 ;
    camera.position.set(xPos, yPos, zPos); // position de la camera
```

## Définition d'une caméra perspective via une fonction

## Définition (Fonction définissant une caméra (perspective))

*Utilisation d'une fonction pour éviter les copier-coller :*

```
function cameraLumiere(scene,camera){      // creation de la camera
    camera.up = new THREE.Vector3( 0, 0, 1 ); // axe des cotes vertical
    let xPos= 6 , yPos= 6 , zPos= 5 , xDir= 0 , yDir= 0 , zDir= 0 ;
    camera.position.set(xPos, yPos, zPos); // position de la camera
    camera.lookAt(xDir, yDir, zDir); // point sur lequel la camera regarde
} // fin fonction cameraLumiere
```

## Définition d'une caméra perspective via une fonction

## Définition (Fonction définissant une caméra (perspective))

*Utilisation d'une fonction pour éviter les copier-coller :*

```
function cameraLumiere(scene,camera){      // creation de la camera
    camera.up = new THREE.Vector3( 0, 0, 1 ); // axe des cotes vertical
    let xPos= 6 , yPos= 6 , zPos= 5 , xDir= 0 , yDir= 0 , zDir= 0 ;
    camera.position.set(xPos, yPos, zPos); // position de la camera
    camera.lookAt(xDir, yDir, zDir); // point sur lequel la camera regarde
} // fin fonction cameraLumiere
```

Dans la fonction `init()`, ajouter :

```
let r = window.innerWidth / window.innerHeight ;
```

## Définition d'une caméra perspective via une fonction

## Définition (Fonction définissant une caméra (perspective))

*Utilisation d'une fonction pour éviter les copier-coller :*

```
function cameraLumiere(scene,camera){ // creation de la camera
    camera.up = new THREE.Vector3( 0, 0, 1 ); // axe des cotes vertical
    let xPos= 6 , yPos= 6 , zPos= 5 , xDir= 0 , yDir= 0 , zDir= 0 ;
    camera.position.set(xPos, yPos, zPos); // position de la camera
    camera.lookAt(xDir, yDir, zDir); // point sur lequel la camera regarde
} // fin fonction cameraLumiere
```

Dans la fonction `init()`, ajouter :

```
let r = window.innerWidth / window.innerHeight ;
let camera = new THREE.PerspectiveCamera(20, r, 0.1, 100) ;
```

où :

- 20 représente l'angle (en degré) de vision (Field of View) ;

## Définition d'une caméra perspective via une fonction

## Définition (Fonction définissant une caméra (perspective))

*Utilisation d'une fonction pour éviter les copier-coller :*

```
function cameraLumiere(scene,camera){ // creation de la camera
    camera.up = new THREE.Vector3( 0, 0, 1 ); // axe des cotes vertical
    let xPos= 6 , yPos= 6 , zPos= 5 , xDir= 0 , yDir= 0 , zDir= 0 ;
    camera.position.set(xPos, yPos, zPos); // position de la camera
    camera.lookAt(xDir, yDir, zDir); // point sur lequel la camera regarde
} // fin fonction cameraLumiere
```

Dans la fonction `init()`, ajouter :

```
let r = window.innerWidth / window.innerHeight ;
let camera = new THREE.PerspectiveCamera(20, r, 0.1, 100) ;
```

où :

- 20 représente l'angle (en degré) de vision (Field of View) ;
- $r$  représente le ratio horizontal-vertical ;

## Définition d'une caméra perspective via une fonction

## Définition (Fonction définissant une caméra (perspective))

*Utilisation d'une fonction pour éviter les copier-coller :*

```
function cameraLumiere(scene,camera){ // creation de la camera
    camera.up = new THREE.Vector3( 0, 0, 1 ); // axe des cotes vertical
    let xPos= 6 , yPos= 6 , zPos= 5 , xDir= 0 , yDir= 0 , zDir= 0 ;
    camera.position.set(xPos, yPos, zPos); // position de la camera
    camera.lookAt(xDir, yDir, zDir); // point sur lequel la camera regarde
} // fin fonction cameraLumiere
```

Dans la fonction `init()`, ajouter :

```
let r = window.innerWidth / window.innerHeight ;
let camera = new THREE.PerspectiveCamera(20, r, 0.1, 100) ;
```

où :

- 20 représente l'angle (en degré) de vision (Field of View) ;
- $r$  représente le ratio horizontal-vertical ;
- 0.1 la distance du premier plan à partir duquel la scène est contenue ;

## Définition d'une caméra perspective via une fonction

## Définition (Fonction définissant une caméra (perspective))

*Utilisation d'une fonction pour éviter les copier-coller :*

```
function cameraLumiere(scene,camera){ // creation de la camera
    camera.up = new THREE.Vector3( 0, 0, 1 ); // axe des cotes vertical
    let xPos= 6 , yPos= 6 , zPos= 5 , xDir= 0 , yDir= 0 , zDir= 0 ;
    camera.position.set(xPos, yPos, zPos); // position de la camera
    camera.lookAt(xDir, yDir, zDir); // point sur lequel la camera regarde
} // fin fonction cameraLumiere
```

Dans la fonction `init()`, ajouter :

```
let r = window.innerWidth / window.innerHeight ;
let camera = new THREE.PerspectiveCamera(20, r, 0.1, 100) ;
```

où :

- 20 représente l'angle (en degré) de vision (Field of View) ;
- $r$  représente le ratio horizontal-vertical ;
- 0.1 la distance du premier plan à partir duquel la scène est contenue ;
- 100 la distance du dernier plan à partir duquel la scène n'est plus contenue.

## Définition d'une caméra perspective via une fonction

## Définition (Fonction définissant une caméra (perspective))

*Utilisation d'une fonction pour éviter les copier-coller :*

```
function cameraLumiere(scene,camera){ // creation de la camera
    camera.up = new THREE.Vector3( 0, 0, 1 ); // axe des cotes vertical
    let xPos= 6 , yPos= 6 , zPos= 5 , xDir= 0 , yDir= 0 , zDir= 0 ;
    camera.position.set(xPos, yPos, zPos); // position de la camera
    camera.lookAt(xDir, yDir, zDir); // point sur lequel la camera regarde
} // fin fonction cameraLumiere
```

Dans la fonction `init()`, ajouter :

```
let r = window.innerWidth / window.innerHeight ;
let camera = new THREE.PerspectiveCamera(20, r, 0.1, 100);
```

où :

- 20 représente l'angle (en degré) de vision (Field of View) ;
- $r$  représente le ratio horizontal-vertical ;
- 0.1 la distance du premier plan à partir duquel la scène est contenue ;
- 100 la distance du dernier plan à partir duquel la scène n'est plus contenue.

puis ajouter (toujours dans la fonction `init()`) :

```
cameraLumiere(scene,camera); // cf. let scene=new THREE.Scene();
```

## Caméra perspective : propriétés

- `camera.position.set(xPos, yPos, zPos)` : position de la caméra ;

## Caméra perspective : propriétés

- `camera.position.set(xPos, yPos, zPos)` : position de la caméra ;
- `camera.lookAt(xDir, yDir, zDir)` : point de visée de la caméra ;

## Caméra perspective : propriétés

- `camera.position.set(xPos, yPos, zPos)` : position de la caméra ;
- `camera.lookAt(xDir, yDir, zDir)` : point de visée de la caméra ;
- `camera.position.x` : abscisse de la position de la caméra ;

## Caméra perspective : propriétés

- `camera.position.set(xPos, yPos, zPos)` : position de la caméra ;
- `camera.lookAt(xDir, yDir, zDir)` : point de visée de la caméra ;
- `camera.position.x` : abscisse de la position de la caméra ;
- `camera.position.y` : ordonnée de la position de la caméra ;

## Caméra perspective : propriétés

- `camera.position.set(xPos, yPos, zPos)` : position de la caméra ;
- `camera.lookAt(xDir, yDir, zDir)` : point de visée de la caméra ;
- `camera.position.x` : abscisse de la position de la caméra ;
- `camera.position.y` : ordonnée de la position de la caméra ;
- `camera.position.z` : cote de la position de la caméra ;

## Caméra perspective : propriétés

- `camera.position.set(xPos, yPos, zPos)` : position de la caméra ;
- `camera.lookAt(xDir, yDir, zDir)` : point de visée de la caméra ;
- `camera.position.x` : abscisse de la position de la caméra ;
- `camera.position.y` : ordonnée de la position de la caméra ;
- `camera.position.z` : cote de la position de la caméra ;
- `camera.fov` : angle de vue de la caméra ;

## Caméra perspective : propriétés

- `camera.position.set(xPos, yPos, zPos)` : position de la caméra ;
- `camera.lookAt(xDir, yDir, zDir)` : point de visée de la caméra ;
- `camera.position.x` : abscisse de la position de la caméra ;
- `camera.position.y` : ordonnée de la position de la caméra ;
- `camera.position.z` : cote de la position de la caméra ;
- `camera.fov` : angle de vue de la caméra ;
- `camera.near` : distance du plan le plus proche de la caméra ;

## Caméra perspective : propriétés

- `camera.position.set(xPos, yPos, zPos)` : position de la caméra ;
- `camera.lookAt(xDir, yDir, zDir)` : point de visée de la caméra ;
- `camera.position.x` : abscisse de la position de la caméra ;
- `camera.position.y` : ordonnée de la position de la caméra ;
- `camera.position.z` : cote de la position de la caméra ;
- `camera.fov` : angle de vue de la caméra ;
- `camera.near` : distance du plan le plus proche de la caméra ;
- `camera.far` : distance du plan le plus éloigné de la caméra ;

## Caméra perspective : propriétés

- `camera.position.set(xPos, yPos, zPos)` : position de la caméra ;
- `camera.lookAt(xDir, yDir, zDir)` : point de visée de la caméra ;
- `camera.position.x` : abscisse de la position de la caméra ;
- `camera.position.y` : ordonnée de la position de la caméra ;
- `camera.position.z` : cote de la position de la caméra ;
- `camera.fov` : angle de vue de la caméra ;
- `camera.near` : distance du plan le plus proche de la caméra ;
- `camera.far` : distance du plan le plus éloigné de la caméra ;
- `camera.aspect` : ratio de la vue ;

## Caméra perspective : propriétés

- `camera.position.set(xPos, yPos, zPos)` : position de la caméra ;
- `camera.lookAt(xDir, yDir, zDir)` : point de visée de la caméra ;
- `camera.position.x` : abscisse de la position de la caméra ;
- `camera.position.y` : ordonnée de la position de la caméra ;
- `camera.position.z` : cote de la position de la caméra ;
- `camera.fov` : angle de vue de la caméra ;
- `camera.near` : distance du plan le plus proche de la caméra ;
- `camera.far` : distance du plan le plus éloigné de la caméra ;
- `camera.aspect` : ratio de la vue ;
- `camera.zoom` : zoom de la caméra.

Caméra : projection perspective ou projection orthographique

## Caméra orthographique : projection parallèle

### Définition (Caméra orthographique)

```
//gauche (g), droite (d), haut (h), bas (b) : window.innerWidth / 16;  
//0 pour near, 500 pour far  
camera = new THREE.OrthographicCamera( g , d , h , b , 0 , 500 );
```

**Deux droites parallèles restent parallèles** : contraire de la vision humaine.

**1** HTML et Three.js**2** Points et Vecteurs

- Définitions
- Méthodes communes à THREE.Vector $n$ ,  $n \in \{2;3\}$
- Cas du plan i.e.  $n = 2$
- Cas de l'espace i.e.  $n = 3$

**3** La caméra**4** Les lumières

- Lumière ambiante
- Définition et propriétés communes des autres lumières
- Lumière ponctuelle et spot
- Spot et lumière directionnelle : ombre
- Le spot

**5** Les matériaux**6** Les courbes

- Cas classique
- Cas spécifique de THREE.js
- Courbes de Bézier polynomiales

**7** Les surfaces primitives

- Le principe
- Cas particulier des surfaces planes
- Les surfaces non planes

**8** Transformations géométriques de  $\mathcal{E}_3$ 

- Application affine : translation
- Application affine ou vectorielle

**9** L'Animation

- Création de menu G.U.I.
- Théorie
- Exemple avec un plan

**11** Le C.S.G.

## 4

## Les lumières

- Lumière ambiante
- Définition et propriétés communes des autres lumières

- Lumière ponctuelle et spot
- Spot et lumière directionnelle : ombre
- Le spot

## Définition de la lumière ambiante et ajout dans la scène

Soit  $\text{CoulH}$  une couleur en hexadécimal (par exemple `0x222222` ou `#222222`).

## Définition de la lumière ambiante et ajout dans la scène

Soit CoulH une couleur en hexadécimal (par exemple 0x222222 ou #222222).

Définition (Lumière ambiante dans la variable `LumAmbiante`)

```
var LumAmbiante = new THREE.AmbientLight( CoulH );
```

## Définition de la lumière ambiante et ajout dans la scène

Soit CoulH une couleur en hexadécimal (par exemple 0x222222 ou #222222).

Définition (Lumière ambiante dans la variable LumAmbiante)

```
var LumAmbiante = new THREE.AmbientLight( CoulH );
```

Remarque (Ajouter la lumière à la scène)

Soit une lumière définie dans la variable Lum . Après les propriétés, mettre :  
scene.add( Lum );

---

Bibliographie (Source du programme)

Learn Three.js - Third Edition de Jos Dirksen

ISBN : 9781788833288

Lumière ambiante

## 4

## Les lumières

- Lumière ambiante
- Définition et propriétés communes des autres lumières

- Lumière ponctuelle et spot
- Spot et lumière directionnelle : ombre
- Le spot

## Définition des lumières non ambiantes

Soit CoulH une couleur en hexadécimal (par exemple 0x222222 ou #222222).

## Définition des lumières non ambiantes

Soit CoulH une couleur en hexadécimal (par exemple 0x222222 ou #222222).

Définition (Lumière ponctuelle dans la variable lumPt)

```
var lumPt = new THREE.PointLight( CoulH );
```

## Définition des lumières non ambiantes

Soit CoulH une couleur en hexadécimal (par exemple 0x222222 ou #222222).

Définition (Lumière ponctuelle dans la variable lumPt)

```
var lumPt = new THREE.PointLight( CoulH );
```

Définition (Spot conique dans la variable Spot1)

```
var Spot1 = new THREE.SpotLight( CoulH );
```

## Définition des lumières non ambiantes

Soit CoulH une couleur en hexadécimal (par exemple 0x222222 ou #222222).

Définition (Lumière ponctuelle dans la variable lumPt)

```
var lumPt = new THREE.PointLight( CoulH );
```

Définition (Spot conique dans la variable Spot1)

```
var Spot1 = new THREE.SpotLight( CoulH );
```

Définition (Lumière directionnelle (rayons parallèles) dans la variable LumDir)

```
var LumDir = new THREE.DirectionalLight( CoulH );
```

## Définition des lumières non ambiantes

Soit CoulH une couleur en hexadécimal (par exemple 0x222222 ou #222222).

Définition (Lumière ponctuelle dans la variable lumPt)

```
var lumPt = new THREE.PointLight( CoulH );
```

Définition (Spot conique dans la variable Spot1)

```
var Spot1 = new THREE.SpotLight( CoulH );
```

Définition (Lumière directionnelle (rayons parallèles) dans la variable LumDir)

```
var LumDir = new THREE.DirectionalLight( CoulH );
```

Remarque ( : Ne pas oublier d'ajouter la lumière à la scène.)

Lumière ponctuelle

Lumière spot conique

Lumière directionnelle

## Bibliographie (Source du programme)

*Learn Three.js - Third Edition de Jos Dirksen. ISBN : 9781788833288*

## Comparaison des trois lumières

- Lumière ponctuelle :

La source est un point qui éclaire dans toutes les directions.

## Comparaison des trois lumières

- Lumière ponctuelle :

La source est un point qui éclaire dans toutes les directions.

- Lumière spot conique :

La source est un point qui éclaire dans un cône de révolution.

## Comparaison des trois lumières

- Lumière ponctuelle :

La source est un point qui éclaire dans toutes les directions.

- Lumière spot conique :

La source est un point qui éclaire dans un cône de révolution.

- Lumière directionnelle :

La source est un point situé à l'infini qui éclaire dans un parallélépipède et toutes les rayons lumineux sont parallèles (comme le soleil par exemple).

Lumière ponctuelle

Lumière spot conique

Lumière directionnelle

### Bibliographie (Source du programme)

*Learn Three.js - Third Edition de Jos Dirksen. ISBN : 9781788833288*

## Propriétés communes aux trois lumières (non ambiantes)

Soit `lum` une lumière ponctuelle, directionnelle ou un spot.

## Propriétés communes aux trois lumières (non ambiantes)

Soit `lum` une lumière ponctuelle, directionnelle ou un spot.

- En écriture : `lum.position.set(x0, y0, z0);` permet de positionner la lumière au point de coordonnées  $(x_0; y_0; z_0)$ .

## Propriétés communes aux trois lumières (non ambiantes)

Soit `lum` une lumière ponctuelle, directionnelle ou un spot.

- En écriture : `lum.position.set(x0, y0, z0);` permet de positionner la lumière au point de coordonnées  $(x_0; y_0; z_0)$ .
- En lecture : récupération de la couleur en RGB
  - ▶ `lum.color.getStyle();` dans  $[0; 255]^3$ .

## Propriétés communes aux trois lumières (non ambiantes)

Soit `lum` une lumière ponctuelle, directionnelle ou un spot.

- En écriture : `lum.position.set(x0, y0, z0);` permet de positionner la lumière au point de coordonnées  $(x_0; y_0; z_0)$ .
- En lecture : récupération de la couleur en RGB
  - `lum.color.getStyle();` dans  $[0; 255]^3$ .
  - `lum.color.getHexString();` en hexadécimal.

## Propriétés communes aux trois lumières (non ambiantes)

Soit `lum` une lumière ponctuelle, directionnelle ou un spot.

- En écriture : `lum.position.set(x0, y0, z0)`; permet de positionner la lumière au point de coordonnées  $(x_0; y_0; z_0)$ .
- En lecture : récupération de la couleur en RGB
  - `lum.color.getStyle()`; dans  $[0; 255]$ <sup>3</sup>.
  - `lum.color.getHexString()`; en hexadécimal.
- En écriture et en lecture :

## Propriétés communes aux trois lumières (non ambiantes)

Soit `lum` une lumière ponctuelle, directionnelle ou un spot.

- En écriture : `lum.position.set(x0, y0, z0)`; permet de positionner la lumière au point de coordonnées  $(x_0; y_0; z_0)$ .
- En lecture : récupération de la couleur en RGB
  - ▶ `lum.color.getStyle()`; dans  $[0; 255]$ <sup>3</sup>.
  - ▶ `lum.color.getHexString()`; en hexadécimal.
- En écriture et en lecture :
  - ▶ Abscisse de la position de la lumière : `lum.position.x`;

## Propriétés communes aux trois lumières (non ambiantes)

Soit `lum` une lumière ponctuelle, directionnelle ou un spot.

- En écriture : `lum.position.set(x0, y0, z0)`; permet de positionner la lumière au point de coordonnées  $(x_0; y_0; z_0)$ .
- En lecture : récupération de la couleur en RGB
  - ▶ `lum.color.getStyle()`; dans  $[0; 255]^3$ .
  - ▶ `lum.color.getHexString()`; en hexadécimal.
- En écriture et en lecture :
  - ▶ Abscisse de la position de la lumière : `lum.position.x`;
  - ▶ Ordonnée de la position de la lumière : `lum.position.y`;

## Propriétés communes aux trois lumières (non ambiantes)

Soit `lum` une lumière ponctuelle, directionnelle ou un spot.

- En écriture : `lum.position.set(x0, y0, z0)`; permet de positionner la lumière au point de coordonnées  $(x_0; y_0; z_0)$ .
- En lecture : récupération de la couleur en RGB
  - ▶ `lum.color.getStyle()`; dans  $[0; 255]^3$ .
  - ▶ `lum.color.getHexString()`; en hexadécimal.
- En écriture et en lecture :
  - ▶ Abscisse de la position de la lumière : `lum.position.x`;
  - ▶ Ordonnée de la position de la lumière : `lum.position.y`;
  - ▶ Cote de la position de la lumière : `lum.position.z`;

## Propriétés communes aux trois lumières (non ambiantes)

Soit `lum` une lumière ponctuelle, directionnelle ou un spot.

- En écriture : `lum.position.set(x0, y0, z0);` permet de positionner la lumière au point de coordonnées  $(x_0; y_0; z_0)$ .
- En lecture : récupération de la couleur en RGB
  - ▶ `lum.color.getStyle();` dans  $[0; 255]^3$ .
  - ▶ `lum.color.getHexString();` en hexadécimal.
- En écriture et en lecture :
  - ▶ Abscisse de la position de la lumière : `lum.position.x`;
  - ▶ Ordonnée de la position de la lumière : `lum.position.y`;
  - ▶ Cote de la position de la lumière : `lum.position.z`;
  - ▶ Intensité de la lumière : `lum.intensity` et la valeur par défaut est 1 ;

## Propriétés communes aux trois lumières (non ambiantes)

Soit `lum` une lumière ponctuelle, directionnelle ou un spot.

- En écriture : `lum.position.set(x0, y0, z0);` permet de positionner la lumière au point de coordonnées  $(x_0; y_0; z_0)$ .
- En lecture : récupération de la couleur en RGB
  - `lum.color.getStyle();` dans  $[0; 255]^3$ .
  - `lum.color.getHexString();` en hexadécimal.
- En écriture et en lecture :
  - Abscisse de la position de la lumière : `lum.position.x`;
  - Ordonnée de la position de la lumière : `lum.position.y`;
  - Cote de la position de la lumière : `lum.position.z`;
  - Intensité de la lumière : `lum.intensity` et la valeur par défaut est 1;
  - Visibilité de la lumière : `lum.visible` qui est un booléen (`true`, la lumière est active, `false`, la lumière n'est pas active).

## 4

## Les lumières

- Lumière ambiante
- Définition et propriétés communes des autres lumières

- Lumière ponctuelle et spot
- Spot et lumière directionnelle : ombre
- Le spot

## Propriété commune à la lumière ponctuelle et au spot

Soit `lum` une lumière ponctuelle ou un spot.

## Propriété commune à la lumière ponctuelle et au spot

Soit `lum` une lumière ponctuelle ou un spot.

La syntaxe `lum.distance` permet d'obtenir la valeur de la distance.

## Propriété commune à la lumière ponctuelle et au spot

Soit `lum` une lumière ponctuelle ou un spot.

La syntaxe `lum.distance` permet d'obtenir la valeur de la distance.

- Si la valeur `lum.distance` vaut 0, il n'y a pas d'atténuation de l'intensité lumineuse ;

## Propriété commune à la lumière ponctuelle et au spot

Soit `lum` une lumière ponctuelle ou un spot.

La syntaxe `lum.distance` permet d'obtenir la valeur de la distance.

- Si la valeur `lum.distance` vaut 0, il n'y a pas d'atténuation de l'intensité lumineuse ;
  - Si la valeur `lum.distance` est strictement positive, l'intensité de la lumière décroît linéairement de la position de la lumière jusqu'à une distance correspondant à cette valeur.
- 

Lumière ponctuelle

Lumière spot conique

Bibliographie (Source du programme)

*Learn Three.js - Third Edition de Jos Dirksen. ISBN : 9781788833288*

## 4

## Les lumières

- Lumière ambiante
- Définition et propriétés communes des autres lumières

- Lumière ponctuelle et spot
- Spot et lumière directionnelle : ombre
- Le spot

## Ombre (shadow)

### Définition (`shadowMap.type`)

*Pour affiner le rendu et les ombres, il faut définir `rendu.shadowMap.type` après*

```
let rendu = new THREE.WebGLRenderer({antialias:true});
```

## Ombre (shadow)

### Définition (`shadowMap.type`)

Pour affiner le rendu et les ombres, il faut définir `rendu.shadowMap.type` après  
`let rendu = new THREE.WebGLRenderer({antialias:true});`  
et les valeurs possibles sont :

- `THREE.BasicShadowMap` qui ne gère pas les ombres ;

## Ombre (shadow)

### Définition (`shadowMap.type`)

Pour affiner le rendu et les ombres, il faut définir `rendu.shadowMap.type` après  
`let rendu = new THREE.WebGLRenderer({antialias:true});`  
et les valeurs possibles sont :

- `THREE.BasicShadowMap` qui ne gère pas les ombres ;
- `THREE.PCFSoftShadowMap` utilise l'algorithme Percentage-Closer Soft Shadows (PCSS) afin de filtrer les ombres ;

## Ombre (shadow)

### Définition (`shadowMap.type`)

Pour affiner le rendu et les ombres, il faut définir `rendu.shadowMap.type` après  
`let rendu = new THREE.WebGLRenderer({antialias:true});` ;  
et les valeurs possibles sont :

- `THREE.BasicShadowMap` qui ne gère pas les ombres ;
- `THREE.PCFSoftShadowMap` utilise l'algorithme Percentage-Closer Soft Shadows (PCSS) afin de filtrer les ombres ;
- par défaut, `THREE.PCFShadowMap` utilise l'algorithme Percentage-Closer Filtering (PCF) ;

## Ombre (shadow)

### Définition (`shadowMap.type`)

Pour affiner le rendu et les ombres, il faut définir `rendu.shadowMap.type` après  
`let rendu = new THREE.WebGLRenderer({antialias:true});`  
et les valeurs possibles sont :

- `THREE.BasicShadowMap` qui ne gère pas les ombres ;
- `THREE.PCFSoftShadowMap` utilise l'algorithme Percentage-Closer Soft Shadows (PCSS) afin de filtrer les ombres ;
- par défaut, `THREE.PCFShadowMap` utilise l'algorithme Percentage-Closer Filtering (PCF) ;
- `THREE.VSMShadowMap` utilise l'algorithme Variance Shadow Map (VSM) et est le plus performant.

## Ombre (shadow)

### Définition (`shadowMap.type`)

Pour affiner le rendu et les ombres, il faut définir `rendu.shadowMap.type` après  
`let rendu = new THREE.WebGLRenderer({antialias:true});`  
et les valeurs possibles sont :

- `THREE.BasicShadowMap` qui ne gère pas les ombres ;
- `THREE.PCFSoftShadowMap` utilise l'algorithme Percentage-Closer Soft Shadows (PCSS) afin de filtrer les ombres ;
- par défaut, `THREE.PCFShadowMap` utilise l'algorithme Percentage-Closer Filtering (PCF) ;
- `THREE.VSMShadowMap` utilise l'algorithme Variance Shadow Map (VSM) et est le plus performant.

### Exemple

```
let rendu = new THREE.WebGLRenderer({antialias:true});  
rendu.shadowMap.type = THREE.VSMShadowMap;
```

## Réglage de l'ombre et de la visée

Soit `lum` un spot ou une lumière directionnelle. Alors :

## Réglage de l'ombre et de la visée

Soit `lum` un spot ou une lumière directionnelle. Alors :

- `lum.target.position.set(x0, y0, z0)`; permet à la lumière d'éclairer dans la direction du point de coordonnées  $(x_0; y_0; z_0)$ ;

## Réglage de l'ombre et de la visée

Soit `lum` un spot ou une lumière directionnelle. Alors :

- `lum.target.position.set(x0, y0, z0);` permet à la lumière d'éclaire dans la direction du point de coordonnées  $(x_0; y_0; z_0)$ ;
- `lum.castShadow = true; //false` permet de spécifier que la lumière `lum` génère de l'ombre (mettre `false` pour le contraire) ;

## Réglage de l'ombre et de la visée

Soit `lum` un spot ou une lumière directionnelle. Alors :

- `lum.target.position.set(x0, y0, z0);` permet à la lumière d'éclaire dans la direction du point de coordonnées  $(x_0; y_0; z_0)$ ;
- `lum.castShadow = true; //false` permet de spécifier que la lumière `lum` génère de l'ombre (mettre `false` pour le contraire) ;
- `lum.shadow.mapSize.width` (resp. `lum.shadow.mapSize.height`) permet de spécifier le nombre de pixels (en puissance de 2) de la largeur (resp. hauteur) de l'ombre et les valeurs par défaut sont 512 ;

## Réglage de l'ombre et de la visée

Soit `lum` un spot ou une lumière directionnelle. Alors :

- `lum.target.position.set(x0, y0, z0);` permet à la lumière d'éclaire dans la direction du point de coordonnées  $(x_0; y_0; z_0)$ ;
- `lum.castShadow = true; //false` permet de spécifier que la lumière `lum` génère de l'ombre (mettre `false` pour le contraire) ;
- `lum.shadow.mapSize.width` (resp. `lum.shadow.mapSize.height`) permet de spécifier le nombre de pixels (en puissance de 2) de la largeur (resp. hauteur) de l'ombre et les valeurs par défaut sont 512 ;
- Il est possible de spécifier les dimensions de l'ombre en une seule fois comme en utilisant : `lum.shadow.mapSize` en lui affectant par exemple  
`//= new THREE.Vector2(Math.pow(2,10), Math.pow(2,10));`

## Réglage de l'ombre et de la visée

Soit `lum` un spot ou une lumière directionnelle. Alors :

- `lum.target.position.set(x0, y0, z0)`; permet à la lumière d'éclaire dans la direction du point de coordonnées ( $x_0; y_0; z_0$ );
- `lum.castShadow = true; //false` permet de spécifier que la lumière `lum` génère de l'ombre (mettre `false` pour le contraire);
- `lum.shadow.mapSize.width` (resp. `lum.shadow.mapSize.height`) permet de spécifier le nombre de pixels (en puissance de 2) de la largeur (resp. hauteur) de l'ombre et les valeurs par défaut sont 512;
- Il est possible de spécifier les dimensions de l'ombre en une seule fois comme en utilisant : `lum.shadow.mapSize` en lui affectant par exemple  
`//= new THREE.Vector2(Math.pow(2,10), Math.pow(2,10));`
- `lum.shadow.camera.far` détermine la distance de la portée de l'ombre, la valeur par défaut est 5000;

## Réglage de l'ombre et de la visée

Soit `lum` un spot ou une lumière directionnelle. Alors :

- `lum.target.position.set(x0, y0, z0)`; permet à la lumière d'éclaire dans la direction du point de coordonnées ( $x_0; y_0; z_0$ );
- `lum.castShadow = true; //false` permet de spécifier que la lumière `lum` génère de l'ombre (mettre `false` pour le contraire);
- `lum.shadow.mapSize.width` (resp. `lum.shadow.mapSize.height`) permet de spécifier le nombre de pixels (en puissance de 2) de la largeur (resp. hauteur) de l'ombre et les valeurs par défaut sont 512;
- Il est possible de spécifier les dimensions de l'ombre en une seule fois comme en utilisant : `lum.shadow.mapSize` en lui affectant par exemple  
`//= new THREE.Vector2(Math.pow(2,10), Math.pow(2,10));`
- `lum.shadow.camera.far` détermine la distance de la portée de l'ombre, la valeur par défaut est 5000 ;
- `lum.shadow.camera.near` détermine la distance à partir de laquelle l'ombre est créée, la valeur par défaut est 50.

## 4

## Les lumières

- Lumière ambiante
- Définition et propriétés communes des autres lumières

- Lumière ponctuelle et spot
- Spot et lumière directionnelle : ombre
- **Le spot**

## Propriété spécifique au spot

Soit Spot1 un spot (cône de révolution). Alors :

## Propriété spécifique au spot

Soit **Spot1** un spot (cône de révolution). Alors :

- la syntaxe `Spot1.angle` permet d'obtenir l'angle, en radian, de deux génératrices coplanaires ;

## Propriété spécifique au spot

Soit **Spot1** un spot (cône de révolution). Alors :

- la syntaxe `Spot1.angle` permet d'obtenir l'angle, en radian, de deux génératrices coplanaires ;
- la syntaxe `Spot1.penumbra`, entre 0 et 1, permet de créer une zone de pénombre (la valeur par défaut est 0 et il n'y a pas de pénombre) ;

## Propriété spécifique au spot

Soit `Spot1` un spot (cône de révolution). Alors :

- la syntaxe `Spot1.angle` permet d'obtenir l'angle, en radian, de deux génératrices coplanaires ;
- la syntaxe `Spot1.penumbra`, entre 0 et 1, permet de créer une zone de pénombre (la valeur par défaut est 0 et il n'y a pas de pénombre) ;
- si la valeur de `Spot1.shadow.radius` est plus supérieure à 1, les bords de l'ombre sont floues à condition de ne pas avoir donné la valeur `THREE.BasicShadowMap` à `shadowMap.type` de `THREE.WebGLRenderer`.

Lumière spot conique

- 1 HTML et Three.js
- 2 Points et Vecteurs
  - Définitions
  - Méthodes communes à THREE.Vector $n$ ,  $n \in \{2;3\}$
  - Cas du plan i.e.  $n = 2$
  - Cas de l'espace i.e.  $n = 3$
- 3 La caméra
- 4 Les lumières
  - Lumière ambiante
  - Définition et propriétés communes des autres lumières
  - Lumière ponctuelle et spot
  - Spot et lumière directionnelle : ombre
  - Le spot
- 5 Les matériaux
- 6 Les courbes
  - Cas classique
  - Cas spécifique de THREE.js
  - Courbes de Bézier polynomiales
- 7 Les surfaces primitives
  - Le principe
  - Cas particulier des surfaces planes
  - Les surfaces non planes
- 8 Transformations géométriques de  $\mathcal{E}_3$ 
  - Application affine : translation
  - Application affine ou vectorielle
- 9 L'Animation
- 10 Création de menu G.U.I.
  - Théorie
  - Exemple avec un plan
- 11 Le C.S.G.

## Matériaux de base

### Définition (`MeshBasicMaterial`)

Ce matériau est insensible à la lumière et ne se définit que par sa couleur. Par exemple via la variable `Matiereau`, la syntaxe suivante :

```
let Matiereau = new THREE.MeshBasicMaterial({color: 0xFF0000});
```

définit un matériau de base de couleur rouge.

Les matériaux

## Modèle de Gouraud

### Définition ( `MeshLambertMaterial` )

Ce matériau a deux paramètres :

- la couleur `color`;

## Modèle de Gouraud

### Définition ( `MeshLambertMaterial` )

Ce matériau a deux paramètres :

- la couleur `color`;
- l'émissivité `emissive`.

## Modèle de Gouraud

### Définition ( `MeshLambertMaterial` )

Ce matériau a deux paramètres :

- la couleur `color`;
- l'émissivité `emissive`.

## Modèle de Gouraud

### Définition ( `MeshLambertMaterial` )

Ce matériau a deux paramètres :

- la couleur `color`;
- l'émissivité `emissive`.

Par exemple via la variable `Materiau`, la syntaxe suivante :

```
let Materiau = new THREE.MeshLambertMaterial({  
    color: 0xFF0000,  
    emissive: 0x000000  
}); // fin Materiau
```

définit un matériau, modèle de Gouraud, de couleur rouge avec une lumière émissive noire.

## Modèle de Phong

Définition ( `MeshPhongMaterial` )

*Ce matériaux a huit paramètres :*

## Modèle de Phong

Définition ( `MeshPhongMaterial` )

*Ce matériaux a huit paramètres :*

- la couleur `color`;

## Modèle de Phong

### Définition ( `MeshPhongMaterial` )

*Ce matériaux a huit paramètres :*

- la couleur `color`;
- la couleur émissive `emissive`

## Modèle de Phong

### Définition ( `MeshPhongMaterial` )

*Ce matériaux a huit paramètres :*

- la couleur `color`;
- la couleur émissive `emissive`
- la couleur spéculaire `specular`

## Modèle de Phong

### Définition (`MeshPhongMaterial`)

Ce matériaux a huit paramètres :

- la couleur `color`;
- la couleur émissive `emissive`
- la couleur spéculaire `specular`
- l'opacité `opacity` dans  $[0;1]$

## Modèle de Phong

### Définition ( `MeshPhongMaterial` )

Ce matériaux a huit paramètres :

- la couleur `color`;
- la couleur émissive `emissive`
- la couleur spéculaire `specular`
- l'opacité `opacity` dans  $[0;1]$
- la transparence `transparent`, valeur booléenne fausse par défaut, qui permet de rendre un objet transparent ou non ;

## Modèle de Phong

### Définition ( `MeshPhongMaterial` )

Ce matériaux a huit paramètres :

- la couleur `color`;
- la couleur émissive `emissive`
- la couleur spéculaire `specular`
- l'opacité `opacity` dans `[0;1]`
- la transparence `transparent`, valeur booléenne fausse par défaut, qui permet de rendre un objet transparent ou non ;
- l'ombrage plat `flatShading`, valeur booléenne valant faux par défaut, qui permet d'avoir une couleur uniforme sur chaque face ;

## Modèle de Phong

### Définition ( `MeshPhongMaterial` )

Ce matériaux a huit paramètres :

- la couleur `color`;
- la couleur émissive `emissive`
- la couleur spéculaire `specular`
- l'opacité `opacity` dans  $[0;1]$
- la transparence `transparent`, valeur booléenne fausse par défaut, qui permet de rendre un objet transparent ou non ;
- l'ombrage plat `flatShading`, valeur booléenne valant faux par défaut, qui permet d'avoir une couleur uniforme sur chaque face ;
- la brillance de la lumière spéculaire `shininess` et la valeur par défaut est 30 ;

## Modèle de Phong

### Définition ( `MeshPhongMaterial` )

Ce matériaux a huit paramètres :

- la couleur `color`;
- la couleur émissive `emissive`
- la couleur spéculaire `specular`
- l'opacité `opacity` dans  $[0;1]$
- la transparence `transparent`, valeur booléenne fausse par défaut, qui permet de rendre un objet transparent ou non ;
- l'ombrage plat `flatShading`, valeur booléenne valant faux par défaut, qui permet d'avoir une couleur uniforme sur chaque face ;
- la brillance de la lumière spéculaire `shininess` et la valeur par défaut est 30 ;
- le rendu des faces `side`

## Modèle de Phong

### Définition ( `MeshPhongMaterial` )

Ce matériaux a huit paramètres :

- la couleur `color`;
- la couleur émissive `emissive`
- la couleur spéculaire `specular`
- l'opacité `opacity` dans  $[0;1]$
- la transparence `transparent`, valeur booléenne fausse par défaut, qui permet de rendre un objet transparent ou non ;
- l'ombrage plat `flatShading`, valeur booléenne valant faux par défaut, qui permet d'avoir une couleur uniforme sur chaque face ;
- la brillance de la lumière spéculaire `shininess` et la valeur par défaut est 30 ;
- le rendu des faces `side`
  - ▶ `THREE.DoubleSide` pour les deux faces qui renvoie la valeur 2 ;

## Modèle de Phong

### Définition (`MeshPhongMaterial`)

Ce matériaux a huit paramètres :

- la couleur `color`;
- la couleur émissive `emissive`
- la couleur spéculaire `specular`
- l'opacité `opacity` dans  $[0;1]$
- la transparence `transparent`, valeur booléenne fausse par défaut, qui permet de rendre un objet transparent ou non ;
- l'ombrage plat `flatShading`, valeur booléenne valant faux par défaut, qui permet d'avoir une couleur uniforme sur chaque face ;
- la brillance de la lumière spéculaire `shininess` et la valeur par défaut est 30 ;
- le rendu des faces `side`
  - ▶ `THREE.DoubleSide` pour les deux faces qui renvoie la valeur 2 ;
  - ▶ `THREE.FrontSide` pour la face avant (valeur par défaut) qui renvoie la valeur 0 ;

## Modèle de Phong

### Définition (`MeshPhongMaterial`)

Ce matériaux a huit paramètres :

- la couleur `color`;
- la couleur émissive `emissive`
- la couleur spéculaire `specular`
- l'opacité `opacity` dans  $[0;1]$
- la transparence `transparent`, valeur booléenne fausse par défaut, qui permet de rendre un objet transparent ou non ;
- l'ombrage plat `flatShading`, valeur booléenne valant faux par défaut, qui permet d'avoir une couleur uniforme sur chaque face ;
- la brillance de la lumière spéculaire `shininess` et la valeur par défaut est 30 ;
- le rendu des faces `side`
  - ▶ `THREE.DoubleSide` pour les deux faces qui renvoie la valeur 2 ;
  - ▶ `THREE.FrontSide` pour la face avant (valeur par défaut) qui renvoie la valeur 0 ;
  - ▶ `THREE.BackSide` pour la face arrière qui renvoie la valeur 1.

## Modèle de Phong

Par exemple via la variable `Materiau`, la syntaxe suivante :

## Modèle de Phong

Par exemple via la variable `Materiau`, la syntaxe suivante :

```
let MaterialPhong = new THREE.MeshPhongMaterial({
```

## Modèle de Phong

Par exemple via la variable `Materiau`, la syntaxe suivante :

```
let MaterialPhong = new THREE.MeshPhongMaterial({  
    color: "#999900", // couleur de l'objet
```

## Modèle de Phong

Par exemple via la variable `Materiau`, la syntaxe suivante :

```
let MaterialPhong = new THREE.MeshPhongMaterial({  
    color: "#999900", // couleur de l'objet  
    opacity: 1,
```

## Modèle de Phong

Par exemple via la variable `Materiau`, la syntaxe suivante :

```
let MaterialPhong = new THREE.MeshPhongMaterial({  
    color: "#999900", // couleur de l'objet  
    opacity: 1,  
    transparent: true,
```

## Modèle de Phong

Par exemple via la variable `Materiau`, la syntaxe suivante :

```
let MaterialPhong = new THREE.MeshPhongMaterial({  
    color: "#999900", // couleur de l'objet  
    opacity: 1,  
    transparent: true,  
    emissive: 0x000000, //couleur emissive
```

## Modèle de Phong

Par exemple via la variable `Materiau`, la syntaxe suivante :

```
let MaterialPhong = new THREE.MeshPhongMaterial({  
    color: "#999900", // couleur de l'objet  
    opacity: 1,  
    transparent: true,  
    emissive: 0x000000, //couleur emissive  
    specular:"#00FFFF", //couleur speculaire
```

## Modèle de Phong

Par exemple via la variable `Materiau`, la syntaxe suivante :

```
let MaterialPhong = new THREE.MeshPhongMaterial({  
    color: "#999900", // couleur de l'objet  
    opacity: 1,  
    transparent: true,  
    emissive: 0x000000, //couleur emissive  
    specular:"#00FFFF", //couleur speculaire  
    flatShading: true,
```

## Modèle de Phong

Par exemple via la variable `Materiau`, la syntaxe suivante :

```
let MaterialPhong = new THREE.MeshPhongMaterial({  
    color: "#999900", // couleur de l'objet  
    opacity: 1,  
    transparent: true,  
    emissive: 0x000000, //couleur emissive  
    specular:"#00FFFF", //couleur speculaire  
    flatShading: true,  
    shininess:30, //brillance
```

## Modèle de Phong

Par exemple via la variable `Materiau`, la syntaxe suivante :

```
let MaterialPhong = new THREE.MeshPhongMaterial({  
    color: "#999900", // couleur de l'objet  
    opacity: 1,  
    transparent: true,  
    emissive: 0x000000, //couleur emissive  
    specular:"#00FFFF", //couleur speculaire  
    flatShading: true,  
    shininess:30,//brillance  
    side: THREE.DoubleSide,//2  
//    side: THREE.FrontSide,//0  
//    side: THREE.BackSide,//1  
});
```

## Modèle de Phong

Par exemple via la variable `Materiau`, la syntaxe suivante :

```
let MaterialPhong = new THREE.MeshPhongMaterial({  
    color: "#999900", // couleur de l'objet  
    opacity: 1,  
    transparent: true,  
    emissive: 0x000000, //couleur emissive  
    specular:"#00FFFF", //couleur spéculaire  
    flatShading: true,  
    shininess:30,//brillance  
    side: THREE.DoubleSide,//2  
//    side: THREE.FrontSide,//0  
//side: THREE.BackSide,//1  
});
```

définit un matériau, modèle de Phong, de couleur jaune, de lumière spéculaire cyan avec une lumière émissive noire.

- 1 HTML et Three.js
- 2 Points et Vecteurs
  - Définitions
  - Méthodes communes à THREE.Vector $n$ ,  $n \in \{2;3\}$
  - Cas du plan i.e.  $n = 2$
  - Cas de l'espace i.e.  $n = 3$
- 3 La caméra
- 4 Les lumières
  - Lumière ambiante
  - Définition et propriétés communes des autres lumières
  - Lumière ponctuelle et spot
  - Spot et lumière directionnelle : ombre
  - Le spot
- 5 Les matériaux
- 6 Les courbes
  - Cas classique
  - Cas spécifique de THREE.js
  - Courbes de Bézier polynomiales
- 7 Les surfaces primitives
  - Le principe
  - Cas particulier des surfaces planes
  - Les surfaces non planes
- 8 Transformations géométriques de  $\mathcal{E}_3$ 
  - Application affine : translation
  - Application affine ou vectorielle
- 9 L'Animation
- 10 Création de menu G.U.I.
  - Théorie
  - Exemple avec un plan
- 11 Le C.S.G.

## 6

## Les courbes

- Cas classique
- Cas spécifique de THREE.js
- Courbes de Bézier polynomiales

## Méthode classique

## Définition (Courbe simple, fermée)

*Soit  $\gamma$  une courbe définie sur l'intervalle fermé  $[a; b]$  à valeur dans  $\mathcal{P}$  ou  $\mathcal{E}_3$ .*

## Méthode classique

## Définition (Courbe simple, fermée)

Soit  $\gamma$  une courbe définie sur l'intervalle fermé  $[a; b]$  à valeur dans  $\mathcal{P}$  ou  $\mathcal{E}_3$ .

- La courbe  $\gamma$  est simple si  $\gamma([a; b])$  ne passe pas deux fois par le même point ;

## Méthode classique

## Définition (Courbe simple, fermée)

Soit  $\gamma$  une courbe définie sur l'intervalle fermé  $[a; b]$  à valeur dans  $\mathcal{P}$  ou  $\mathcal{E}_3$ .

- La courbe  $\gamma$  est simple si  $\gamma([a; b])$  ne passe pas deux fois par le même point ;
- La courbe simple  $\gamma$  est fermée si  $\gamma(a) = \gamma(b)$ .

## Méthode classique

## Définition (Courbe simple, fermée)

Soit  $\gamma$  une courbe définie sur l'intervalle fermé  $[a; b]$  à valeur dans  $\mathcal{P}$  ou  $\mathcal{E}_3$ .

- La courbe  $\gamma$  est simple si  $\gamma([a; b])$  ne passe pas deux fois par le même point ;
- La courbe simple  $\gamma$  est fermée si  $\gamma(a) = \gamma(b)$ .

## Exemple (Type de courbes)

- Courbe non simple : huit ;

## Méthode classique

## Définition (Courbe simple, fermée)

Soit  $\gamma$  une courbe définie sur l'intervalle fermé  $[a; b]$  à valeur dans  $\mathcal{P}$  ou  $\mathcal{E}_3$ .

- La courbe  $\gamma$  est simple si  $\gamma([a; b])$  ne passe pas deux fois par le même point ;
- La courbe simple  $\gamma$  est fermée si  $\gamma(a) = \gamma(b)$ .

## Exemple (Type de courbes)

- Courbe non simple : huit ;
- Courbe simple non fermée : segment, demi-cercle, arc de parabole ;

## Méthode classique

## Définition (Courbe simple, fermée)

Soit  $\gamma$  une courbe définie sur l'intervalle fermé  $[a; b]$  à valeur dans  $\mathcal{P}$  ou  $\mathcal{E}_3$ .

- La courbe  $\gamma$  est simple si  $\gamma([a; b])$  ne passe pas deux fois par le même point ;
- La courbe simple  $\gamma$  est fermée si  $\gamma(a) = \gamma(b)$ .

## Exemple (Type de courbes)

- Courbe non simple : huit ;
- Courbe simple non fermée : segment, demi-cercle, arc de parabole ;
- Courbe simple fermée : polygone, cercle, ellipse.

## Méthode classique

### Définition (Courbe simple, fermée)

Soit  $\gamma$  une courbe définie sur l'intervalle fermé  $[a; b]$  à valeur dans  $\mathcal{P}$  ou  $\mathcal{E}_3$ .

- La courbe  $\gamma$  est simple si  $\gamma([a; b])$  ne passe pas deux fois par le même point ;
- La courbe simple  $\gamma$  est fermée si  $\gamma(a) = \gamma(b)$ .

### Exemple (Type de courbes)

- Courbe non simple : huit ;
- Courbe simple non fermée : segment, demi-cercle, arc de parabole ;
- Courbe simple fermée : polygone, cercle, ellipse.

### Principe (Tracé d'une courbe simple $\gamma$ )

1/ Si la courbe est fermée, subdiviser l'intervalle  $[a; b[$ , sinon, subdiviser l'intervalle  $[a; b]$  ;

## Méthode classique

### Définition (Courbe simple, fermée)

Soit  $\gamma$  une courbe définie sur l'intervalle fermé  $[a; b]$  à valeur dans  $\mathcal{P}$  ou  $\mathcal{E}_3$ .

- La courbe  $\gamma$  est simple si  $\gamma([a; b])$  ne passe pas deux fois par le même point ;
- La courbe simple  $\gamma$  est fermée si  $\gamma(a) = \gamma(b)$ .

### Exemple (Type de courbes)

- Courbe non simple : huit ;
- Courbe simple non fermée : segment, demi-cercle, arc de parabole ;
- Courbe simple fermée : polygone, cercle, ellipse.

### Principe (Tracé d'une courbe simple $\gamma$ )

- 1/ Si la courbe est fermée, subdiviser l'intervalle  $[a; b[$ , sinon, subdiviser l'intervalle  $[a; b]$  ;
- 2/ Pour chaque valeur de la subdivision, placer les points (images des valeurs de la subdivision par  $\gamma$ ) dans un tableau ;

## Méthode classique

### Exemple (Type de courbes)

- *Courbe non simple : huit ;*
- *Courbe simple non fermée : segment, demi-cercle, arc de parabole ;*
- *Courbe simple fermée : polygone, cercle, ellipse.*

### Principe (Tracé d'une courbe simple $\gamma$ )

- 1/ *Si la courbe est fermée, subdiviser l'intervalle  $[a; b[$ , sinon, subdiviser l'intervalle  $[a; b]$  ;*
- 2/ *Pour chaque valeur de la subdivision, placer les points (images des valeurs de la subdivision par  $\gamma$ ) dans un tableau ;*
- 3/ *Relier les éléments consécutifs du tableau par un segment ;*

## Méthode classique

### Exemple (Type de courbes)

- *Courbe non simple : huit ;*
- *Courbe simple non fermée : segment, demi-cercle, arc de parabole ;*
- *Courbe simple fermée : polygone, cercle, ellipse.*

### Principe (Tracé d'une courbe simple $\gamma$ )

- 1/ *Si la courbe est fermée, subdiviser l'intervalle  $[a; b[$ , sinon, subdiviser l'intervalle  $[a; b]$  ;*
- 2/ *Pour chaque valeur de la subdivision, placer les points (images des valeurs de la subdivision par  $\gamma$ ) dans un tableau ;*
- 3/ *Relier les éléments consécutifs du tableau par un segment ;*
- 4/ *Si la courbe est fermée, relier le dernier point du tableau avec le premier point du tableau par un segment...*

## 6 Les courbes

- Cas classique
- Cas spécifique de THREE.js
- Courbes de Bézier polynomiales

Utilisation de THREE.Line pour tracer une courbe simple  $\gamma$ 

```
1) // nb segments => tableau de (nb+1) points  
let points = new Array(nb+1);
```

Utilisation de THREE.Line pour tracer une courbe simple  $\gamma$ 

```
1) // nb segments => tableau de (nb+1) points  
let points = new Array(nb+1);  
2) // discréétisation de l'intervalle [0;1]  
for(let k=0;k<=nb;k++){  
    let t2=k/nb // à modifier si [a;b];  
    let x0 = ... ;// en fonction de t2  
    let y0 = ... ;// en fonction de t2  
    let z0 = ... ;// en fonction de t2  
    points[k] = new THREE.Vector3(x0,y0,z0);  
}  
// fin for
```

Utilisation de THREE.Line pour tracer une courbe simple  $\gamma$ 

```
1) // nb segments => tableau de (nb+1) points
let points = new Array(nb+1);
2) // discréétisation de l'intervalle [0;1]
for(let k=0;k<=nb;k++){
    let t2=k/nb // à modifier si [a;b];
    let x0 = ... ;// en fonction de t2
    let y0 = ... ;// en fonction de t2
    let z0 = ... ;// en fonction de t2
    points[k] = new THREE.Vector3(x0,y0,z0);
}
3) // Création des points pour la courbe paramétrée
let PtsTab = new THREE.BufferGeometry().setFromPoints(points);
```

Utilisation de THREE.Line pour tracer une courbe simple  $\gamma$ 

```
1) // nb segments => tableau de (nb+1) points
let points = new Array(nb+1);
2) // discréétisation de l'intervalle [0;1]
for(let k=0;k<=nb;k++){
    let t2=k/nb // à modifier si [a;b];
    let x0 = ... ;// en fonction de t2
    let y0 = ... ;// en fonction de t2
    let z0 = ... ;// en fonction de t2
    points[k] = new THREE.Vector3(x0,y0,z0);
}
3) // Création des points pour la courbe paramétrée
let PtsTab = new THREE.BufferGeometry().setFromPoints(points);
4) // Propriétés de la courbe
let ProprieteCbe = new THREE.LineBasicMaterial( {
    color:"#FF0000", // couleur
    linewidth: 3;//épaisseur en pixels
} );
```

Utilisation de THREE.Line pour tracer une courbe simple  $\gamma$ 

- 2) // discretisation de l'intervalle [0;1]
 

```
for(let k=0;k<=nb;k++){
    let t2=k/nb // a modifier si [a;b];
    let x0 = ... ;// en fonction de t2
    let y0 = ... ;// en fonction de t2
    let z0 = ... ;// en fonction de t2
    points[k] = new THREE.Vector3(x0,y0,z0);
}
```
- 3) // Creation des points pour la courbe parametree
 

```
let PtsTab = new THREE.BufferGeometry().setFromPoints(points);
```
- 4) // Proprietes de la courbe
 

```
let ProprieteCbe = new THREE.LineBasicMaterial( {
  color:"#FF0000", // couleur
  linewidth: 3;//epaisseur en pixels
} );
```
- 5) //creation de la courbe parametree avec ses proprietes
 

```
let courbePara = new THREE.Line( PtsTab, ProprieteCbe );
```

Utilisation de THREE.Line pour tracer une courbe simple  $\gamma$ 

```
3) // Creation des points pour la courbe parametree  
let PtsTab = new THREE.BufferGeometry().setFromPoints(points);  
4) // Proprietes de la courbe  
let ProprieteCbe = new THREE.LineBasicMaterial( {  
    color:"#FF0000", // couleur  
    linewidth: 3; //epaisseur en pixels  
} );  
5) //creation de la courbe parametree avec ses proprietes  
let courbePara = new THREE.Line( PtsTab, ProprieteCbe );  
6) //ajout de la courbe dans la scene  
scene.add(courbePara);
```

Traçage d'une courbe

## 6

## Les courbes

- Cas classique
- Cas spécifique de THREE.js
- Courbes de Bézier polynomiales

## Courbe de Bézier de degré 2 ou 3

## Définition (Courbe de Bézier de degré 2)

Pour  $i \in \llbracket 0; 2 \rrbracket$ , soit  $P_i$  une instance stockant les coordonnées du point  $P_i$  de  $\mathcal{E}_3$ .

## Courbe de Bézier de degré 2 ou 3

### Définition (Courbe de Bézier de degré 2)

Pour  $i \in \llbracket 0; 2 \rrbracket$ , soit  $P_i$  une instance stockant les coordonnées du point  $P_i$  de  $\mathcal{E}_3$ .  
La syntaxe :

```
new THREE.QuadraticBezierCurve3(P0, P1, P2);
```

permet de définir la courbe de Bézier polynomiale quadratique de points de contrôle  $P_0$ ,  $P_1$  et  $P_2$ .

## Courbe de Bézier de degré 2 ou 3

### Définition (Courbe de Bézier de degré 2)

Pour  $i \in \llbracket 0; 2 \rrbracket$ , soit  $P_i$  une instance stockant les coordonnées du point  $P_i$  de  $\mathcal{E}_3$ .  
La syntaxe :

```
new THREE.QuadraticBezierCurve3(P0, P1, P2);
```

permet de définir la courbe de Bézier polynomiale quadratique de points de contrôle  $P_0$ ,  $P_1$  et  $P_2$ .

### Définition (Courbe de Bézier de degré 3)

Pour  $i \in \llbracket 0; 3 \rrbracket$ , soit  $P_i$  une instance stockant les coordonnées du point  $P_i$  de  $\mathcal{E}_3$ .

## Courbe de Bézier de degré 2 ou 3

### Définition (Courbe de Bézier de degré 2)

Pour  $i \in \llbracket 0; 2 \rrbracket$ , soit  $P_i$  une instance stockant les coordonnées du point  $P_i$  de  $\mathcal{E}_3$ .  
La syntaxe :

```
new THREE.QuadraticBezierCurve3 (P0, P1, P2);
```

permet de définir la courbe de Bézier polynomiale quadratique de points de contrôle  $P_0$ ,  $P_1$  et  $P_2$ .

### Définition (Courbe de Bézier de degré 3)

Pour  $i \in \llbracket 0; 3 \rrbracket$ , soit  $P_i$  une instance stockant les coordonnées du point  $P_i$  de  $\mathcal{E}_3$ .  
La syntaxe :

```
new THREE.CubicBezierCurve3 (P0, P1, P2, P3);
```

permet de définir la courbe de Bézier polynomiale cubique de points de contrôle  $P_0$ ,  $P_1$ ,  $P_2$  et  $P_3$ .

## Courbe de Bézier polynomiale de degré 2 : exemple

```
function TraceBezierQuadratique(P0, P1, P2, nbPts, coul, epaiCbe){
```

## Courbe de Bézier polynomiale de degré 2 : exemple

```
function TraceBezierQuadratique(P0, P1, P2, nbPts, coul, epaiCbe){  
let cbeBez = new THREE.QuadraticBezierCurve3 (P0, P1, P2 );
```

## Courbe de Bézier polynomiale de degré 2 : exemple

```
function TraceBezierQuadratique(P0, P1, P2, nbPts, coul, epaiCbe){  
    let cbeBez = new THREE.QuadraticBezierCurve3 (P0, P1, P2 );  
    //Propriete geometrique de la courbe  
    let cbeGeometry = new THREE.Geometry();
```

## Courbe de Bézier polynomiale de degré 2 : exemple

```
function TraceBezierQuadratique(P0, P1, P2, nbPts, coul, epaiCbe){  
    let cbeBez = new THREE.QuadraticBezierCurve3 (P0, P1, P2 );  
    //Propriete geometrique de la courbe  
    let cbeGeometry = new THREE.Geometry();  
    // Points de la courbe de Bezier  
    cbeGeometry.vertices = cbeBez.getPoints(nbPts);
```

## Courbe de Bézier polynomiale de degré 2 : exemple

```
function TraceBezierQuadratique(P0, P1, P2, nbPts, coul, epaiCbe){  
    let cbeBez = new THREE.QuadraticBezierCurve3 (P0, P1, P2 );  
    //Propriete geometrique de la courbe  
    let cbeGeometry = new THREE.Geometry();  
    // Points de la courbe de Bezier  
    cbeGeometry.vertices = cbeBez.getPoints(nbPts);  
    //Aspect de la courbe  
    let material = new THREE.LineBasicMaterial(  
        { color : coul ,  
          linewidth: epaiCbe  
    } );
```

## Courbe de Bézier polynomiale de degré 2 : exemple

```
function TraceBezierQuadratique(P0, P1, P2, nbPts, coul, epaiCbe){  
    let cbeBez = new THREE.QuadraticBezierCurve3(P0, P1, P2);  
    //Propriete geometrique de la courbe  
    let cbeGeometry = new THREE.Geometry();  
    // Points de la courbe de Bezier  
    cbeGeometry.vertices = cbeBez.getPoints(nbPts);  
    //Aspect de la courbe  
    let material = new THREE.LineBasicMaterial(  
        { color : coul ,  
          linewidth: epaiCbe  
    } );  
    // Courbe de Bezier avec les proprietes geometriques et l'aspect  
    let BezierQuadratique = new THREE.Line( cbeGeometry, material );
```

## Courbe de Bézier polynomiale de degré 2 : exemple

```
function TraceBezierQuadratique(P0, P1, P2, nbPts, coul, epaiCbe){  
    let cbeBez = new THREE.QuadraticBezierCurve3(P0, P1, P2);  
    //Propriete geometrique de la courbe  
    let cbeGeometry = new THREE.Geometry();  
    // Points de la courbe de Bezier  
    cbeGeometry.vertices = cbeBez.getPoints(nbPts);  
    //Aspect de la courbe  
    let material = new THREE.LineBasicMaterial(  
        { color : coul ,  
          linewidth: epaiCbe  
    } );  
    // Courbe de Bezier avec les proprietes geometriques et l'aspect  
    let BezierQuadratique = new THREE.Line( cbeGeometry, material );  
    //Renvoi de la courbe pour une utilisation ultérieure  
    return (BezierQuadratique);  
}
```

## Courbe de Bézier polynomiale de degré 2 : exemple

```
// definition des points
let P0 = new THREE.Vector3(1,0,0);
let P1 = new THREE.Vector3(0,0,1);
let P2 = new THREE.Vector3(-1,0,0);
```

## Courbe de Bézier polynomiale de degré 2 : exemple

```
// definition des points
let P0 = new THREE.Vector3(1,0,0);
let P1 = new THREE.Vector3(0,0,1);
let P2 = new THREE.Vector3(-1,0,0);
let nb=100;//nombre de pts par courbe
```

## Courbe de Bézier polynomiale de degré 2 : exemple

```
// definition des points
let P0 = new THREE.Vector3(1,0,0);
let P1 = new THREE.Vector3(0,0,1);
let P2 = new THREE.Vector3(-1,0,0);
let nb=100;//nombre de pts par courbe
let epai=2;//epaisseur de la courbe
```

## Courbe de Bézier polynomiale de degré 2 : exemple

```
// definition des points
let P0 = new THREE.Vector3(1,0,0);
let P1 = new THREE.Vector3(0,0,1);
let P2 = new THREE.Vector3(-1,0,0);
let nb=100;//nombre de pts par courbe
let epai=2;//epaisseur de la courbe
let dimPt=0.025;// dimension d'un point
```

## Courbe de Bézier polynomiale de degré 2 : exemple

```
// definition des points
let P0 = new THREE.Vector3(1,0,0);
let P1 = new THREE.Vector3(0,0,1);
let P2 = new THREE.Vector3(-1,0,0);
let nb=100;//nombre de pts par courbe
let epai=2;//epaisseur de la courbe
let dimPt=0.025;// dimension d'un point
//tracer des points
tracePt(scene, P0, "#000000",dimPt,true);
tracePt(scene, P1, "#000000",dimPt,true);
tracePt(scene, P2, "#000000",dimPt,true);
```

## Courbe de Bézier polynomiale de degré 2 : exemple

```
// definition des points
let P0 = new THREE.Vector3(1,0,0);
let P1 = new THREE.Vector3(0,0,1);
let P2 = new THREE.Vector3(-1,0,0);
let nb=100;//nombre de pts par courbe
let epai=2;//epaisseur de la courbe
let dimPt=0.025;// dimension d'un point
//tracer des points
tracePt(scene, P0, "#000000",dimPt,true);
tracePt(scene, P1, "#000000",dimPt,true);
tracePt(scene, P2, "#000000",dimPt,true);
// tracer du polygone de controle
segment(scene,P0,P1,"#00FF00",epai);
segment(scene,P1,P2,"#0000FF",epai);
```

## Courbe de Bézier polynomiale de degré 2 : exemple

```
// definition des points
let P0 = new THREE.Vector3(1,0,0);
let P1 = new THREE.Vector3(0,0,1);
let P2 = new THREE.Vector3(-1,0,0);
let nb=100;//nombre de pts par courbe
let epai=2;//epaisseur de la courbe
let dimPt=0.025;// dimension d'un point
//tracer des points
tracePt(scene, P0, "#000000",dimPt,true);
tracePt(scene, P1, "#000000",dimPt,true);
tracePt(scene, P2, "#000000",dimPt,true);
// tracer du polygone de controle
segment(scene,P0,P1,"#00FF00",epai);
segment(scene,P1,P2,"#0000FF",epai);
// definition de la courbe de Bezier de degré 2
let cbeBez2 = TraceBezierQuadratique(P0, P1, P2, nb, "#FF0000",
epai);
```

## Courbe de Bézier polynomiale de degré 2 : exemple

```
// definition des points
let P0 = new THREE.Vector3(1,0,0);
let P1 = new THREE.Vector3(0,0,1);
let P2 = new THREE.Vector3(-1,0,0);
let nb=100;//nombre de pts par courbe
let epai=2;//epaisseur de la courbe
let dimPt=0.025;// dimension d'un point
//tracer des points
tracePt(scene, P0, "#000000",dimPt,true);
tracePt(scene, P1, "#000000",dimPt,true);
tracePt(scene, P2, "#000000",dimPt,true);
// tracer du polygone de controle
segment(scene,P0,P1,"#00FF00",epai);
segment(scene,P1,P2,"#0000FF",epai);
// definition de la courbe de Bezier de degré 2
let cbeBez2 = TraceBezierQuadratique(P0, P1, P2, nb, "#FF0000",
                                         epai);
// ajout de la courbe dans la scene
scene.add(cbeBez2);
```

Courbe de Bézier quadratique

## Courbe de Bézier polynomiale de degré 3 : exemple

```
function TraceBezierCubique(P0, P1, P2, P3, nbPts, coul, epaiCbe){
```

## Courbe de Bézier polynomiale de degré 3 : exemple

```
function TraceBezierCubique(P0, P1, P2, P3, nbPts, coul, epaiCbe){  
let cbeBez = new THREE.CubicBezierCurve3(P0, P1, P2, P3);
```

## Courbe de Bézier polynomiale de degré 3 : exemple

```
function TraceBezierCubique(P0, P1, P2, P3, nbPts, coul, epaiCbe){  
let cbeBez = new THREE.CubicBezierCurve3(P0, P1, P2, P3);  
//Propriete geometrique de la courbe  
let cbeGeometry = new THREE.Geometry();
```

## Courbe de Bézier polynomiale de degré 3 : exemple

```
function TraceBezierCubique(P0, P1, P2, P3, nbPts, coul, epaiCbe){  
    let cbeBez = new THREE.CubicBezierCurve3(P0, P1, P2, P3);  
    //Propriete geometrique de la courbe  
    let cbeGeometry = new THREE.Geometry();  
    // Points de la courbe de Bezier  
    cbeGeometry.vertices = cbeBez.getPoints(nbPts);  
}
```

## Courbe de Bézier polynomiale de degré 3 : exemple

```
function TraceBezierCubique(P0, P1, P2, P3, nbPts, coul, epaiCbe){  
    let cbeBez = new THREE.CubicBezierCurve3(P0, P1, P2, P3);  
    //Propriete geometrique de la courbe  
    let cbeGeometry = new THREE.Geometry();  
    // Points de la courbe de Bezier  
    cbeGeometry.vertices = cbeBez.getPoints(nbPts);  
    //Aspect de la courbe  
    let material = new THREE.LineBasicMaterial(  
        { color : coul ,  
          linewidth: epaiCbe  
    } );
```

## Courbe de Bézier polynomiale de degré 3 : exemple

```
function TraceBezierCubique(P0, P1, P2, P3, nbPts, coul, epaiCbe){  
    let cbeBez = new THREE.CubicBezierCurve3(P0, P1, P2, P3);  
    //Propriete geometrique de la courbe  
    let cbeGeometry = new THREE.Geometry();  
    // Points de la courbe de Bezier  
    cbeGeometry.vertices = cbeBez.getPoints(nbPts);  
    //Aspect de la courbe  
    let material = new THREE.LineBasicMaterial(  
        { color : coul ,  
          linewidth: epaiCbe  
    } );  
    // courbe de Bezier avec les proprietes geometriques et l'aspect  
    let BezierCubique = new THREE.Line( cbeGeometry, material );
```

## Courbe de Bézier polynomiale de degré 3 : exemple

```
function TraceBezierCubique(P0, P1, P2, P3, nbPts, coul, epaiCbe){  
    let cbeBez = new THREE.CubicBezierCurve3(P0, P1, P2, P3);  
    //Propriete geometrique de la courbe  
    let cbeGeometry = new THREE.Geometry();  
    // Points de la courbe de Bezier  
    cbeGeometry.vertices = cbeBez.getPoints(nbPts);  
    //Aspect de la courbe  
    let material = new THREE.LineBasicMaterial(  
        { color : coul ,  
          linewidth: epaiCbe  
    } );  
    // courbe de Bezier avec les proprietes geometriques et l'aspect  
    let BezierCubique = new THREE.Line( cbeGeometry, material );  
    //Renvoi de la courbe pour une utilisation ultérieure  
    return (BezierCubique);  
} // fin fonction TraceBezierCubique
```

## Courbe de Bézier polynomiale de degré 3 : exemple

```
let P0 = new THREE.Vector3(1,0,0);
let P1 = new THREE.Vector3(1,0,1);
let P2 = new THREE.Vector3(-1,0,1);
let P3 = new THREE.Vector3(-1,0,0);
```

## Courbe de Bézier polynomiale de degré 3 : exemple

```
let P0 = new THREE.Vector3(1,0,0);
let P1 = new THREE.Vector3(1,0,1);
let P2 = new THREE.Vector3(-1,0,1);
let P3 = new THREE.Vector3(-1,0,0);
let nb=100;//nombre de pts de la courbe
let epai=2;//epaisseur de la courbe
```

## Courbe de Bézier polynomiale de degré 3 : exemple

```
let P0 = new THREE.Vector3(1,0,0);
let P1 = new THREE.Vector3(1,0,1);
let P2 = new THREE.Vector3(-1,0,1);
let P3 = new THREE.Vector3(-1,0,0);
let nb=100;//nombre de pts de la courbe
let epai=2;//epaisseur de la courbe
let dimPt=0.025;//dimension des points puis tracer
tracePt(scene, P1, "#000000",dimPt,true);
tracePt(scene, P0, "#000000",dimPt,true);
tracePt(scene, P2, "#000000",dimPt,true);
tracePt(scene, P3, "#000000",dimPt,true);
```

## Courbe de Bézier polynomiale de degré 3 : exemple

```
let P0 = new THREE.Vector3(1,0,0);
let P1 = new THREE.Vector3(1,0,1);
let P2 = new THREE.Vector3(-1,0,1);
let P3 = new THREE.Vector3(-1,0,0);
let nb=100;//nombre de pts de la courbe
let epai=2;//epaisseur de la courbe
let dimPt=0.025;//dimension des points puis tracer
tracePt(scene, P1, "#000000",dimPt,true);
tracePt(scene, P0, "#000000",dimPt,true);
tracePt(scene, P2, "#000000",dimPt,true);
tracePt(scene, P3, "#000000",dimPt,true);
//polygone de controle
segment(scene, P0, P1, "#00FF00", epai);
segment(scene, P1, P2, "#0000FF", epai);
segment(scene, P2, P3, "#FF00FF", epai);
```

## Courbe de Bézier polynomiale de degré 3 : exemple

```
let P0 = new THREE.Vector3(1,0,0);
let P1 = new THREE.Vector3(1,0,1);
let P2 = new THREE.Vector3(-1,0,1);
let P3 = new THREE.Vector3(-1,0,0);
let nb=100;//nombre de pts de la courbe
let epai=2;//epaisseur de la courbe
let dimPt=0.025;//dimension des points puis tracer
tracePt(scene, P1, "#000000",dimPt,true);
tracePt(scene, P0, "#000000",dimPt,true);
tracePt(scene, P2, "#000000",dimPt,true);
tracePt(scene, P3, "#000000",dimPt,true);
//polygone de controle
segment(scene, P0, P1, "#00FF00", epai);
segment(scene, P1, P2, "#0000FF", epai);
segment(scene, P2, P3, "#FF00FF", epai);
// courbe de Bezier cubique
let cbeBez3 = TraceBezierCubique(P0, P1, P2, P3,
nb, "#FF0000", epai);
```

## Courbe de Bézier polynomiale de degré 3 : exemple

```
let P0 = new THREE.Vector3(1,0,0);
let P1 = new THREE.Vector3(1,0,1);
let P2 = new THREE.Vector3(-1,0,1);
let P3 = new THREE.Vector3(-1,0,0);
let nb=100;//nombre de pts de la courbe
let epai=2;//epaisseur de la courbe
let dimPt=0.025;//dimension des points puis tracer
tracePt(scene, P1, "#000000",dimPt,true);
tracePt(scene, P0, "#000000",dimPt,true);
tracePt(scene, P2, "#000000",dimPt,true);
tracePt(scene, P3, "#000000",dimPt,true);
//polygone de controle
segment(scene, P0, P1, "#00FF00", epai);
segment(scene, P1, P2, "#0000FF", epai);
segment(scene, P2, P3, "#FF00FF", epai);
// courbe de Bezier cubique
let cbeBez3 = TraceBezierCubique(P0, P1, P2, P3,
nb, "#FF0000", epai);
// ajout de la courbe dans la scene
scene.add(cbeBez3);
```

Courbe de Bézier cubique

**1** HTML et Three.js**2** Points et Vecteurs

- Définitions
- Méthodes communes à THREE.Vector $n$ ,  $n \in \{2;3\}$
- Cas du plan i.e.  $n = 2$
- Cas de l'espace i.e.  $n = 3$

**3** La caméra**4** Les lumières

- Lumière ambiante
- Définition et propriétés communes des autres lumières
- Lumière ponctuelle et spot
- Spot et lumière directionnelle : ombre
- Le spot

**5** Les matériaux**6** Les courbes

- Cas classique
- Cas spécifique de THREE.js
- Courbes de Bézier polynomiales

**7** Les surfaces primitives

- Le principe
- Cas particulier des surfaces planes
- Les surfaces non planes

**8** Transformations géométriques de  $\mathcal{E}_3$ 

- Application affine : translation
- Application affine ou vectorielle

**9** L'Animation**10** Création de menu G.U.I.

- Théorie
- Exemple avec un plan

**11** Le C.S.G.

## 7

## Les surfaces primitives

- Le principe
- Cas particulier des surfaces planes
- Les surfaces non planes

## Affichage d'une surface

Analogue à l'affichage des courbes :

Principe (Etape pour la définition d'une surface)

1) *Définir les propriétés géométriques de la surface ;*

## Affichage d'une surface

Analogue à l'affichage des courbes :

### Principe (Etape pour la définition d'une surface)

- 1) *Définir les propriétés géométriques de la surface ;*
- 2) *Définir les propriétés du matériaux (Basique, Gouraud ou Phong) de la surface ;*

## Affichage d'une surface

Analogue à l'affichage des courbes :

### Principe (Etape pour la définition d'une surface)

- 1) Définir les propriétés géométriques de la surface ;
- 2) Définir les propriétés du matériaux (Basique, Gouraud ou Phong) de la surface ;
- 3) Définir la surface  $S$  à afficher en utilisant les deux points précédents ;

## Affichage d'une surface

Analogue à l'affichage des courbes :

### Principe (Etape pour la définition d'une surface)

- 1) Définir les propriétés géométriques de la surface ;
- 2) Définir les propriétés du matériaux (Basique, Gouraud ou Phong) de la surface ;
- 3) Définir la surface  $S$  à afficher en utilisant les deux points précédents ;
- 4) Faire les transformations géométriques éventuelles concernant la surface  $S$  ;

## Affichage d'une surface

Analogue à l'affichage des courbes :

### Principe (Etape pour la définition d'une surface)

- 1) *Définir les propriétés géométriques de la surface ;*
- 2) *Définir les propriétés du matériaux (Basique, Gouraud ou Phong) de la surface ;*
- 3) *Définir la surface  $S$  à afficher en utilisant les deux points précédents ;*
- 4) *Faire les transformations géométriques éventuelles concernant la surface  $S$  ;*
- 5) *Ajouter la surface  $S$  à la scène.*

## 7

## Les surfaces primitives

- Le principe
- Cas particulier des surfaces planes
- Les surfaces non planes

Plan d'équation  $z = 0$ 

## Définition (Définition du plan)

1) *Définition des paramètres pour le plan :*

Plan d'équation  $z = 0$ 

## Définition (Définition du plan)

- 1) *Définition des paramètres pour le plan :*

```
let largPlan = 25 ;
```

```
let hautPlan = 25 ;
```

Plan d'équation  $z = 0$ 

## Définition (Définition du plan)

- 1) *Définition des paramètres pour le plan :*

```
let largPlan = 25 ;
```

```
let nbSegmentLarg = 30 ;
```

```
let hautPlan = 25 ;
```

```
let nbSegmentHaut = 30 ;
```

Plan d'équation  $z = 0$ 

## Définition (Définition du plan)

- 1) *Définition des paramètres pour le plan :*

```
let largPlan = 25 ;           let nbSegmentLarg = 30 ;  
let hautPlan = 25 ;           let nbSegmentHaut = 30 ;
```

```
let planGeometry = new THREE.PlaneGeometry(largPlan,hautPlan,  
                                         nbSegmentLarg,nbSegmentHaut);
```

Plan d'équation  $z = 0$ 

## Définition (Définition du plan)

- 1) *Définition des paramètres pour le plan :*

```
let largPlan = 25 ;           let nbSegmentLarg = 30 ;  
let hautPlan = 25 ;           let nbSegmentHaut = 30 ;
```

```
let planGeometry = new THREE.PlaneGeometry(largPlan, hautPlan,  
                                         nbSegmentLarg, nbSegmentHaut);
```

- 2) *Définition des propriétés du matériaux, (cf. section sur les matériaux) :*

```
let MaterialPhong = new THREE.MeshPhongMaterial({...});
```

Plan d'équation  $z = 0$ 

## Définition (Définition du plan)

- 1) *Définition des paramètres pour le plan :*

```
let largPlan = 25 ;           let nbSegmentLarg = 30 ;  
let hautPlan = 25 ;           let nbSegmentHaut = 30 ;
```

```
let planGeometry = new THREE.PlaneGeometry(largPlan,hautPlan,  
                                            nbSegmentLarg,nbSegmentHaut);
```

- 2) *Définition des propriétés du matériaux, (cf. section sur les matériaux) :*

```
let MaterialPhong = new THREE.MeshPhongMaterial({...});
```

- 3) *Définir la surface S à afficher;*

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);
```

Plan d'équation  $z = 0$ 

## Définition (Définition du plan)

- 1) *Définition des paramètres pour le plan :*

```
let largPlan = 25 ;  
let hautPlan = 25 ;  
let nbSegmentLarg = 30 ;  
let nbSegmentHaut = 30 ;
```

```
let planGeometry = new THREE.PlaneGeometry(largPlan,hautPlan,  
nbSegmentLarg,nbSegmentHaut);
```

- 2) *Définition des propriétés du matériaux, (cf. section sur les matériaux) :*

```
let MaterialPhong = new THREE.MeshPhongMaterial({...});
```

- 3) *Définir la surface S à afficher;*

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);
```

- 4) *Permettre les ombres :*

```
planPhong.castShadow = true ;  
planPhong.receiveShadow = true ;
```

Plan d'équation  $z = 0$ 

## Définition (Définition du plan)

- 1) *Définition des paramètres pour le plan :*

```
let largPlan = 25 ;  
let hautPlan = 25 ;  
let nbSegmentLarg = 30 ;  
let nbSegmentHaut = 30 ;
```

```
let planGeometry = new THREE.PlaneGeometry(largPlan,hautPlan,  
nbSegmentLarg,nbSegmentHaut);
```

- 2) *Définition des propriétés du matériaux, (cf. section sur les matériaux) :*

```
let MaterialPhong = new THREE.MeshPhongMaterial({...});
```

- 3) *Définir la surface S à afficher;*

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);
```

- 4) *Permettre les ombres :*

```
planPhong.castShadow = true ;  
planPhong.receiveShadow = true ;
```

- 5) *Ajouter la surface S à la scène :*

```
scene.add(planPhong);
```

Traçage d'un plan

## Facette triangulaire

## Définition (Définition d'une facette triangulaire)

- 1) *Définition des sommets par leurs coordonnées :*

```
let PtA = new THREE.Vector3 (1,0,0);
let PtB = new THREE.Vector3 (0,1,0);
let PtC = new THREE.Vector3 (0,0,0);
```

## Facette triangulaire

## Définition (Définition d'une facette triangulaire)

- 1) *Définition des sommets par leurs coordonnées :*

```
let PtA = new THREE.Vector3 (1,0,0);  
let PtB = new THREE.Vector3 (0,1,0);  
let PtC = new THREE.Vector3 (0,0,0);
```

- 2) *Définition de la face géométrique : let faceT = new THREE.Geometry();*

## Facette triangulaire

## Définition (Définition d'une facette triangulaire)

- 1) *Définition des sommets par leurs coordonnées :*

```
let PtA = new THREE.Vector3 (1,0,0);  
let PtB = new THREE.Vector3 (0,1,0);  
let PtC = new THREE.Vector3 (0,0,0);
```

- 2) *Définition de la face géométrique : let faceT = new THREE.Geometry();*
- 3) *Définition des sommets : faceT.vertices = [PtA, PtB, PtC];*

## Facette triangulaire

## Définition (Définition d'une facette triangulaire)

- 1) *Définition des sommets par leurs coordonnées :*

```
let PtA = new THREE.Vector3 (1,0,0);  
let PtB = new THREE.Vector3 (0,1,0);  
let PtC = new THREE.Vector3 (0,0,0);
```

- 2) *Définition de la face géométrique : let faceT = new THREE.Geometry();*

- 3) *Définition des sommets : faceT.vertices = [PtA, PtB, PtC];*

- 4) *Parcours de la face : faceT.faces = [  
new THREE.Face3 ( 0, 1, 2 ) //ordre A, B, C  
];*

## Facette triangulaire

## Définition (Définition d'une facette triangulaire)

- 1) *Définition des sommets par leurs coordonnées :*

```
let PtA = new THREE.Vector3(1,0,0);
let PtB = new THREE.Vector3(0,1,0);
let PtC = new THREE.Vector3(0,0,0);
```

- 2) *Définition de la face géométrique : let faceT = new THREE.Geometry();*

- 3) *Définition des sommets : faceT.vertices = [PtA, PtB, PtC];*

- 4) *Parcours de la face : faceT.faces = [  
new THREE.Face3(0, 1, 2) //ordre A, B, C  
];*

- 5) *Définition de la facette avec la matériau :*

```
let faceTriang = new THREE.Mesh(faceT, MaterialPhong);
```

## Facette triangulaire

## Définition (Définition d'une facette triangulaire)

- 1) *Définition des sommets par leurs coordonnées :*

```
let PtA = new THREE.Vector3(1,0,0);
let PtB = new THREE.Vector3(0,1,0);
let PtC = new THREE.Vector3(0,0,0);
```

- 2) *Définition de la face géométrique :* let faceT = new THREE.Geometry();

- 3) *Définition des sommets :* faceT.vertices = [PtA, PtB, PtC];

- 4) *Parcours de la face :* faceT.faces = [  
 new THREE.Face3(0, 1, 2) //ordre A, B, C  
];

- 5) *Définition de la facette avec la matériau :*

```
let faceTriang = new THREE.Mesh(faceT, MaterialPhong);
```

- 6) *Ajout de la facette dans la scène :* scene.add(faceTriang);

Traçage d'un triangle

## Facette quadrangulaire

## Définition (Définition d'une facette quadrangulaire)

- 1) *Définition des sommets par leurs coordonnées :*

```
let PtA = new THREE.Vector3 (1,0,0);
let PtB = new THREE.Vector3 (0,1,0);
let PtC = new THREE.Vector3 (-1,0,0);
let PtD = new THREE.Vector3 (0,-1,0);
```

## Facette quadrangulaire

## Définition (Définition d'une facette quadrangulaire)

- 1) *Définition des sommets par leurs coordonnées :*

```
let PtA = new THREE.Vector3 (1,0,0);  
let PtB = new THREE.Vector3 (0,1,0);  
let PtC = new THREE.Vector3 (-1,0,0);  
let PtD = new THREE.Vector3 (0,-1,0);
```

- 2) *Définition de la face géométrique :*

```
let faceGeom = new THREE.Geometry();
```

## Facette quadrangulaire

## Définition (Définition d'une facette quadrangulaire)

- 1) *Définition des sommets par leurs coordonnées :*

```
let PtA = new THREE.Vector3 (1,0,0);  
let PtB = new THREE.Vector3 (0,1,0);  
let PtC = new THREE.Vector3 (-1,0,0);  
let PtD = new THREE.Vector3 (0,-1,0);
```

- 2) *Définition de la face géométrique :*

```
let faceGeom = new THREE.Geometry();
```

- 3) *Définition des sommets :*

```
faceGeom .vertices = [PtA, PtB, PtC, PtD];
```

## Facette quadrangulaire

## Définition (Définition d'une facette quadrangulaire)

- 1) *Définition des sommets par leurs coordonnées :*

```
let PtA = new THREE.Vector3 (1,0,0);
let PtB = new THREE.Vector3 (0,1,0);
let PtC = new THREE.Vector3 (-1,0,0);
let PtD = new THREE.Vector3 (0,-1,0);
```

- 2) *Définition de la face géométrique :*

```
let faceGeom = new THREE.Geometry();
```

- 3) *Définition des sommets :*

```
faceGeom.vertices = [PtA, PtB, PtC, PtD];
```

- 4) *Parcours de la face par deux facettes triangulaires :*  `faceGeom.faces = [`

```
new THREE.Face3 ( 0, 1, 2 ), //A, B, C
new THREE.Face3 ( 0, 2, 3 ), //A, C, D
```

```
];
```

## Facette quadrangulaire

## Définition (Définition d'une facette quadrangulaire)

- 1) *Définition des sommets par leurs coordonnées :*

```
let PtA = new THREE.Vector3 (1,0,0);
let PtB = new THREE.Vector3 (0,1,0);
let PtC = new THREE.Vector3 (-1,0,0);
let PtD = new THREE.Vector3 (0,-1,0);
```

- 2) *Définition de la face géométrique :*

```
let faceGeom = new THREE.Geometry();
```

- 3) *Définition des sommets :*

```
faceGeom.vertices = [PtA, PtB, PtC, PtD];
```

- 4) *Parcours de la face par deux facettes triangulaires :* `faceGeom.faces = [`

```
new THREE.Face3 ( 0, 1, 2 ), //A, B, C
new THREE.Face3 ( 0, 2, 3 ), //A, C, D
```

```
];
```

- 5) *Définition de la facette avec la matériau :*

```
let faceQuad = new THREE.Mesh( faceGeom, MaterialPhong );
```

## Facette quadrangulaire

## Définition (Définition d'une facette quadrangulaire)

1) *Définition des sommets par leurs coordonnées :*

```
let PtA = new THREE.Vector3 (1,0,0);
let PtB = new THREE.Vector3 (0,1,0);
let PtC = new THREE.Vector3 (-1,0,0);
let PtD = new THREE.Vector3 (0,-1,0);
```

2) *Définition de la face géométrique :*

```
let faceGeom = new THREE.Geometry();
```

3) *Définition des sommets :*

```
faceGeom.vertices = [PtA, PtB, PtC, PtD];
```

4) *Parcours de la face par deux facettes triangulaires :* `faceGeom.faces = [  
 new THREE.Face3 ( 0, 1, 2 ), //A, B, C  
 new THREE.Face3 ( 0, 2, 3 ), //A, C, D  
];`5) *Définition de la facette avec la matériau :*

```
let faceQuad = new THREE.Mesh( faceGeom, MaterialPhong );
```

6) *Ajout de la facette dans la scène :*`scene.add( faceQuad );`

Traçage d'un carré

## 7

## Les surfaces primitives

- Le principe
- Cas particulier des surfaces planes
- Les surfaces non planes

## Définition (Définition d'une surface)

- 1) *Définition des paramètres géométriques de la surface;*  
*Partie à modifier*

## Définition (Définition d'une surface)

- 1) *Définition des paramètres géométriques de la surface;*  
**Partie à modifier**
- 2) *Définition des propriétés du matériaux, (cf. le cas du plan);*

## Définition (Définition d'une surface)

- 1) *Définition des paramètres géométriques de la surface;*  
**Partie à modifier**
- 2) *Définition des propriétés du matériaux, (cf. le cas du plan);*
- 3) *Définir la surface  $S$  à afficher, (cf. le cas du plan);*

## Définition (Définition d'une surface)

- 1) *Définition des paramètres géométriques de la surface;*  
**Partie à modifier**
- 2) *Définition des propriétés du matériaux, (cf. le cas du plan);*
- 3) *Définir la surface  $S$  à afficher, (cf. le cas du plan);*
- 4) *Permettre les ombres, (cf. le cas du plan);*

## Définition (Définition d'une surface)

- 1) *Définition des paramètres géométriques de la surface;*  
**Partie à modifier**
- 2) *Définition des propriétés du matériaux, (cf. le cas du plan);*
- 3) *Définir la surface  $S$  à afficher, (cf. le cas du plan);*
- 4) *Permettre les ombres, (cf. le cas du plan);*
- 5) *Ajouter la surface  $S$  à la scène, (cf. le cas du plan).*

## Parallélépipède rectangle : définition

Centre du parallélépipède : origine du repère ; arêtes parallèles aux axes du repère.

Définition (Définition géométrique d'un parallélépipède rectangle)

1) *Définir la largeur (abscisse) : let largeur = 0.5;*

## Parallélépipède rectangle : définition

Centre du parallélépipède : origine du repère ; arêtes parallèles aux axes du repère.

Définition (Définition géométrique d'un parallélépipède rectangle)

- 1) *Définir la largeur (abscisse) : let largeur = 0.5;*
- 2) *Définir la hauteur (ordonnée) : let hauteur = 1;*

## Parallélépipède rectangle : définition

Centre du parallélépipède : origine du repère ; arêtes parallèles aux axes du repère.

## Définition (Définition géométrique d'un parallélépipède rectangle)

- 1) *Définir la largeur (abscisse) : let largeur = 0.5;*
- 2) *Définir la hauteur (ordonnée) : let hauteur = 1;*
- 3) *Définir la profondeur (côte) : let profondeur = 1;*

## Parallélépipède rectangle : définition

Centre du parallélépipède : origine du repère ; arêtes parallèles aux axes du repère.

## Définition (Définition géométrique d'un parallélépipède rectangle)

- 1) *Définir la largeur (abscisse) : let largeur = 0.5;*
- 2) *Définir la hauteur (ordonnée) : let hauteur = 1;*
- 3) *Définir la profondeur (côte) : let profondeur = 1;*
- 4) *Définir le nombre de segment en largeur : let largSegments = 10;*

## Parallélépipède rectangle : définition

Centre du parallélépipède : origine du repère ; arêtes parallèles aux axes du repère.

Définition (Définition géométrique d'un parallélépipède rectangle)

- 1) *Définir la largeur (abscisse) : let largeur = 0.5;*
- 2) *Définir la hauteur (ordonnée) : let hauteur = 1;*
- 3) *Définir la profondeur (côte) : let profondeur = 1;*
- 4) *Définir le nombre de segment en largeur : let largSegments = 10;*
- 5) *Définir le nombre de segment en hauteur : let hautSegments = 10;*

## Parallélépipède rectangle : définition

Centre du parallélépipède : origine du repère ; arêtes parallèles aux axes du repère.

## Définition (Définition géométrique d'un parallélépipède rectangle)

- 1) *Définir la largeur (abscisse) : let largeur = 0.5;*
- 2) *Définir la hauteur (ordonnée) : let hauteur = 1;*
- 3) *Définir la profondeur (côte) : let profondeur = 1;*
- 4) *Définir le nombre de segment en largeur : let largSegments = 10;*
- 5) *Définir le nombre de segment en hauteur : let hautSegments = 10;*
- 6) *Définir le nombre de segment en profondeur : let profSegments = 10;*

## Parallélépipède rectangle : définition

Centre du parallélépipède : origine du repère ; arêtes parallèles aux axes du repère.

## Définition (Définition géométrique d'un parallélépipède rectangle)

- 1) *Définir la largeur (abscisse) : let largeur = 0.5;*
- 2) *Définir la hauteur (ordonnée) : let hauteur = 1;*
- 3) *Définir la profondeur (côte) : let profondeur = 1;*
- 4) *Définir le nombre de segment en largeur : let largSegments = 10;*
- 5) *Définir le nombre de segment en hauteur : let hautSegments = 10;*
- 6) *Définir le nombre de segment en profondeur : let profSegments = 10;*
- 7) *Définir le parallélépipède rectangle :*

```
let BoiteGeom = new THREE.BoxGeometry(largeur, hauteur,  
profondeur, largSegments, hautSegments, profSegments);
```

Traçage d'un parallélépipède rectangle

## Parallélépipède rectangle : propriété

Rappel (Définition géométrique d'un parallélépipède rectangle)

```
let BoiteGeom = new THREE.BoxGeometry (largeur, hauteur,  
profondeur, largSegments, hautSegments, profSegments );
```

## Parallélépipède rectangle : propriété

Rappel (Définition géométrique d'un parallélépipède rectangle)

```
let BoiteGeom = new THREE.BoxGeometry (largeur, hauteur,  
profondeur, largSegments, hautSegments, profSegments );
```

Définition (Propriétés géométriques d'un parallélépipède rectangle BoiteGeom )

- 1) Largeur : BoiteGeom .parameters.width

## Parallélépipède rectangle : propriété

Rappel (Définition géométrique d'un parallélépipède rectangle)

```
let BoiteGeom = new THREE.BoxGeometry (largeur, hauteur,  
profondeur, largSegments, hautSegments, profSegments );
```

Définition (Propriétés géométriques d'un parallélépipède rectangle `BoiteGeom`)

- 1) *Largeur* : `BoiteGeom.parameters.width`
- 2) *Hauteur* : `BoiteGeom.parameters.height`

## Parallélépipède rectangle : propriété

Rappel (Définition géométrique d'un parallélépipède rectangle)

```
let BoiteGeom = new THREE.BoxGeometry (largeur, hauteur,  
profondeur, largSegments, hautSegments, profSegments );
```

Définition (Propriétés géométriques d'un parallélépipède rectangle BoiteGeom )

- 1) Largeur : BoiteGeom .parameters.width
- 2) Hauteur : BoiteGeom .parameters.height
- 3) Profondeur : BoiteGeom .parameters.depth

## Parallélépipède rectangle : propriété

Rappel (Définition géométrique d'un parallélépipède rectangle)

```
let BoiteGeom = new THREE.BoxGeometry (largeur, hauteur,  
profondeur, largSegments, hautSegments, profSegments );
```

Définition (Propriétés géométriques d'un parallélépipède rectangle BoiteGeom )

- 1) Largeur : BoiteGeom .parameters.width
- 2) Hauteur : BoiteGeom .parameters.height
- 3) Profondeur : BoiteGeom .parameters.depth
- 4) Nombre de points en largeur : BoiteGeom .parameters.widthSegments

## Parallélépipède rectangle : propriété

Rappel (Définition géométrique d'un parallélépipède rectangle)

```
let BoiteGeom = new THREE.BoxGeometry (largeur, hauteur,  
profondeur, largSegments, hautSegments, profSegments );
```

Définition (Propriétés géométriques d'un parallélépipède rectangle BoiteGeom )

- 1) Largeur : BoiteGeom .parameters.width
- 2) Hauteur : BoiteGeom .parameters.height
- 3) Profondeur : BoiteGeom .parameters.depth
- 4) Nombre de points en largeur : BoiteGeom .parameters.widthSegments
- 5) Nombre de points en hauteur : BoiteGeom .parameters.heightSegments

## Parallélépipède rectangle : propriété

Rappel (Définition géométrique d'un parallélépipède rectangle)

```
let BoiteGeom = new THREE.BoxGeometry (largeur, hauteur,  
profondeur, largSegments, hautSegments, profSegments );
```

Définition (Propriétés géométriques d'un parallélépipède rectangle BoiteGeom )

- 1) Largeur : BoiteGeom .parameters.width
- 2) Hauteur : BoiteGeom .parameters.height
- 3) Profondeur : BoiteGeom .parameters.depth
- 4) Nombre de points en largeur : BoiteGeom .parameters.widthSegments
- 5) Nombre de points en hauteur : BoiteGeom .parameters.heightSegments
- 6) Nombre de points en profondeur : BoiteGeom .parameters.depthSegments

Traçage d'un parallélépipède rectangle

## Cube d'arête de longueur 1 appuyé sur les axes du repère

```
let arete = 1;  
let largSegments = 10;  
let hautSegments = 10;  
let profSegments = 10;
```

## Cube d'arête de longueur 1 appuyé sur les axes du repère

```
let arete = 1;  
let largSegments = 10;  
let hautSegments = 10;  
let profSegments = 10;  
//cube géométrique  
let cubeGeom = new THREE.BoxGeometry(arete, arete, arete,  
largSegments, hautSegments, profSegments );
```

## Cube d'arête de longueur 1 appuyé sur les axes du repère

```
let arete = 1;
let largSegments = 10;
let hautSegments = 10;
let profSegments = 10;
//cube geometrique
let cubeGeom = new THREE.BoxGeometry(arete, arete, arete,
largSegments, hautSegments, profSegments );
//cube avec materiau
let cubePhong = new THREE.Mesh(cubeGeom,MaterialPhong);
```

## Cube d'arête de longueur 1 appuyé sur les axes du repère

```
let arete = 1;
let largSegments = 10;
let hautSegments = 10;
let profSegments = 10;
//cube geometrique
let cubeGeom = new THREE.BoxGeometry(arete, arete, arete,
largSegments, hautSegments, profSegments );
//cube avec materiau
let cubePhong = new THREE.Mesh(cubeGeom,MaterialPhong);
//positionnement
cubePhong.position.set(cubeGeom.parameters.width/2,
    cubeGeom.parameters.height/2,
    cubeGeom.parameters.depth/2 );
```

## Cube d'arête de longueur 1 appuyé sur les axes du repère

```
let arete = 1;
let largSegments = 10;
let hautSegments = 10;
let profSegments = 10;
//cube geometrique
let cubeGeom = new THREE.BoxGeometry(arete, arete, arete,
largSegments, hautSegments, profSegments );
//cube avec materiau
let cubePhong = new THREE.Mesh(cubeGeom,MaterialPhong);
//positionnement
cubePhong.position.set(cubeGeom.parameters.width/2,
    cubeGeom.parameters.height/2,
    cubeGeom.parameters.depth/2 );
//ombrage
cubePhong.castShadow = true;
cubePhong.receiveShadow = true;
```

## Cube d'arête de longueur 1 appuyé sur les axes du repère

```
let arete = 1;
let largSegments = 10;
let hautSegments = 10;
let profSegments = 10;
//cube geometrique
let cubeGeom = new THREE.BoxGeometry(arete, arete, arete,
largSegments, hautSegments, profSegments );
//cube avec materiau
let cubePhong = new THREE.Mesh(cubeGeom,MaterialPhong);
//positionnement
cubePhong.position.set(cubeGeom.parameters.width/2,
    cubeGeom.parameters.height/2,
    cubeGeom.parameters.depth/2 );
//ombrage
cubePhong.castShadow = true;
cubePhong.receiveShadow = true;
//ajout dans la scene
scene.add(cubePhong);
```

Cube

## Sphères

Définition (Définition géométrique d'une sphère)

1) *Définir le rayon de la sphère : let rayon = 0.5 ;*

## Sphères

Définition (Définition géométrique d'une sphère)

- 1) *Définir le rayon de la sphère : let rayon = 0.5 ;*
- 2) *Définir le nombre de parallèles : let nbeParallel = 100 ;*

## Sphères

## Définition (Définition géométrique d'une sphère)

- 1) *Définir le rayon de la sphère : let rayon = 0.5 ;*
- 2) *Définir le nombre de parallèles : let nbeParallel = 100 ;*
- 3) *Définir le nombre de méridiens : let nbeMeridien = 60 ;*

## Sphères

## Définition (Définition géométrique d'une sphère)

- 1) Définir le rayon de la sphère : `let rayon = 0.5 ;`
- 2) Définir le nombre de parallèles : `let nbeParallel = 100 ;`
- 3) Définir le nombre de méridiens : `let nbeMeridien = 60 ;`
- 4) Définir la sphère géométrique :

```
let sphereGeom = new THREE.SphereGeometry (rayon ,  
nbeParallel, nbeMeridien );
```

## Sphères

## Définition (Définition géométrique d'une sphère)

- 1) Définir le rayon de la sphère : `let rayon = 0.5 ;`
- 2) Définir le nombre de parallèles : `let nbeParallel = 100 ;`
- 3) Définir le nombre de méridiens : `let nbeMeridien = 60 ;`
- 4) Définir la sphère géométrique :

```
let sphereGeom = new THREE.SphereGeometry (rayon ,  
nbeParallel, nbeMeridien );
```

Définition (Propriétés géométriques la sphère `sphereGeom`)

- 1) Rayon de la sphère : `sphereGeom.parameters.radius ;`

## Sphères

## Définition (Définition géométrique d'une sphère)

- 1) Définir le rayon de la sphère : `let rayon = 0.5 ;`
- 2) Définir le nombre de parallèles : `let nbeParallel = 100 ;`
- 3) Définir le nombre de méridiens : `let nbeMeridien = 60 ;`
- 4) Définir la sphère géométrique :

```
let sphereGeom = new THREE.SphereGeometry (rayon ,  
nbeParallel, nbeMeridien );
```

Définition (Propriétés géométriques la sphère `sphereGeom`)

- 1) Rayon de la sphère : `sphereGeom .parameters.radius ;`
- 2) Nombre de parallèles : `sphereGeom .parameters.widthSegments ;`

## Sphères

## Définition (Définition géométrique d'une sphère)

- 1) Définir le rayon de la sphère : `let rayon = 0.5 ;`
- 2) Définir le nombre de parallèles : `let nbeParallel = 100 ;`
- 3) Définir le nombre de méridiens : `let nbeMeridien = 60 ;`
- 4) Définir la sphère géométrique :

```
let sphereGeom = new THREE.SphereGeometry (rayon ,  
nbeParallel, nbeMeridien );
```

Définition (Propriétés géométriques la sphère `sphereGeom`)

- 1) Rayon de la sphère : `sphereGeom .parameters.radius` ;
- 2) Nombre de parallèles : `sphereGeom .parameters.widthSegments` ;
- 3) Nombre de méridiens : `sphereGeom .parameters.heightSegments` ;

Traçage d'une sphère

## Sphères

Définition (Définition des « bornes » d'une sphère, valeur par défaut)

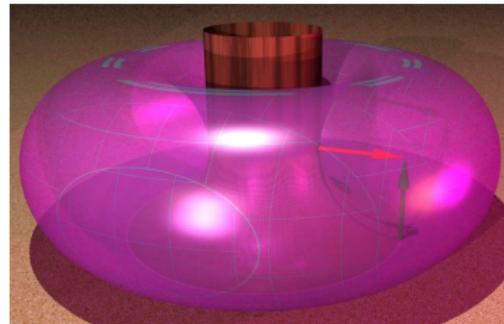
- 1) Définir le début du premier intervalle : `let phi0 = 0;`
- 2) Définir la longueur du premier intervalle : `let phiLg = 2 *Math.PI;`
- 3) Définir le début du second intervalle : `let theta0 = 0;`
- 4) Définir la longueur du second intervalle : `let thetaLg = Math.PI;`
- 5) `let sphereGeom = new THREE.SphereGeometry (rayon, nbeParallel, nbeMeridien, phi0, phiLg, theta0, thetaLg);`

Définition (Propriétés géométriques de la sphère `sphereGeom` (suite))

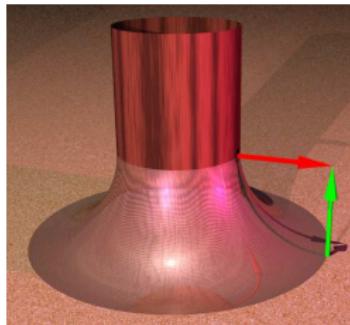
- 1) Début du premier intervalle : `sphereGeom.parameters.phiStart`
- 2) Longueur du premier intervalle : `sphereGeom.parameters.phiLength`
- 3) Début du second intervalle : `sphereGeom.parameters.thetaStart`
- 4) Longueur du second intervalle : `sphereGeom.parameters.thetaLength`

Traçage d'une sphère

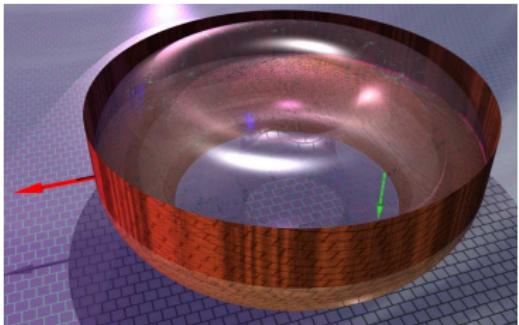
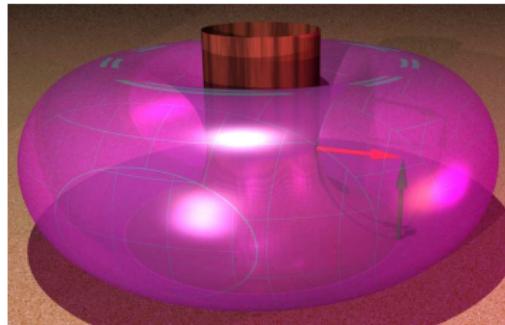
## Sphères orientées : pourquoi ?



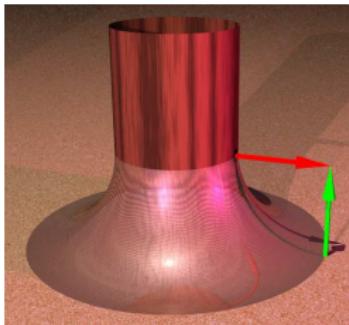
Extérieur du tore : partie bornée.



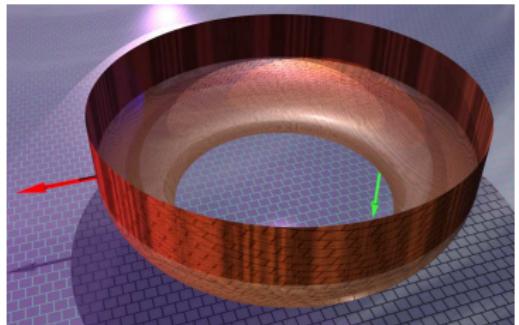
## Sphères orientées : pourquoi ?



Extérieur du tore : partie bornée.



Extérieur du tore : partie non bornée.

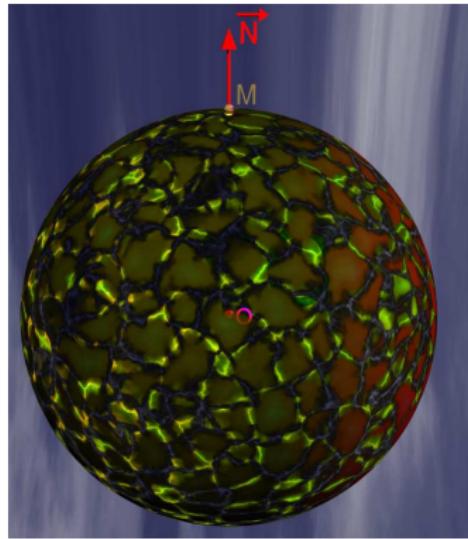


## Sphères orientées : définition

Une sphère  $S$  de centre  $O$  et de rayon  $r \implies$  deux sphères orientées  $S^+$  et  $S^-$  : En tout point  $M$  de  $S$ ,

## Sphères orientées : définition

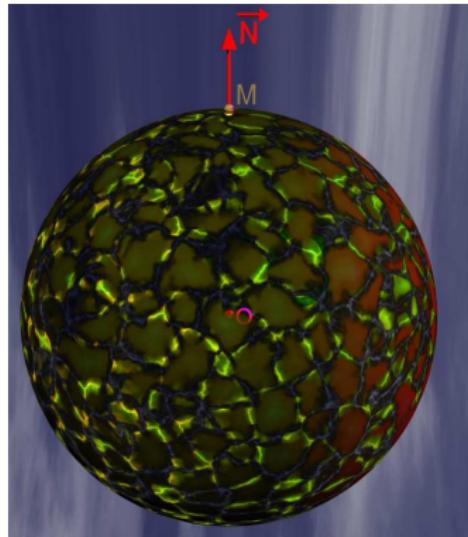
Une sphère  $S$  de centre  $O$  et de rayon  $r \implies$  deux sphères orientées  $S^+$  et  $S^-$  : En tout point  $M$  de  $S$ ,



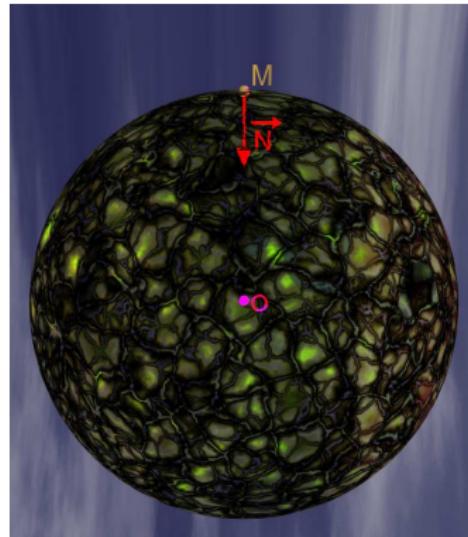
$S^+$ ,  $\rho = r$  et  $\vec{N}$  est sortant.

## Sphères orientées : définition

Une sphère  $S$  de centre  $O$  et de rayon  $r \implies$  deux sphères orientées  $S^+$  et  $S^-$  : En tout point  $M$  de  $S$ ,



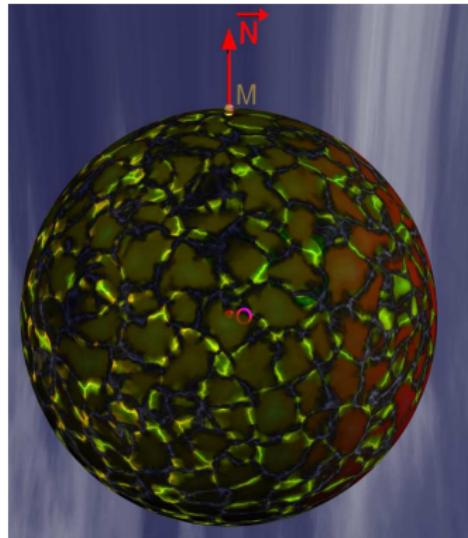
$S^+$ ,  $\rho = r$  et  $\vec{N}$  est sortant.



$S^-$ ,  $\rho = -r$  et  $\vec{N}$  est rentrant.

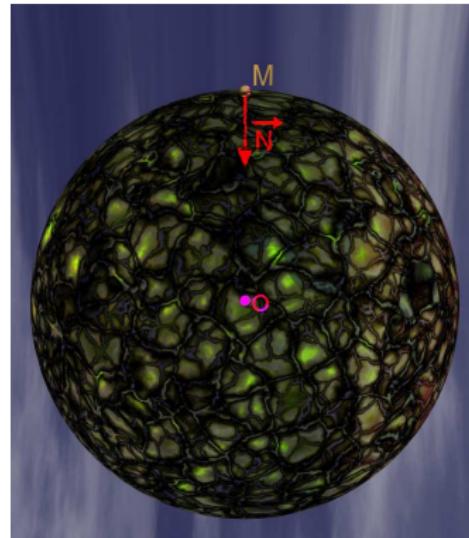
## Sphères orientées : définition

Une sphère  $S$  de centre  $O$  et de rayon  $r \implies$  deux sphères orientées  $S^+$  et  $S^-$  : En tout point  $M$  de  $S$ ,



$S^+$ ,  $\rho = r$  et  $\vec{N}$  est sortant.

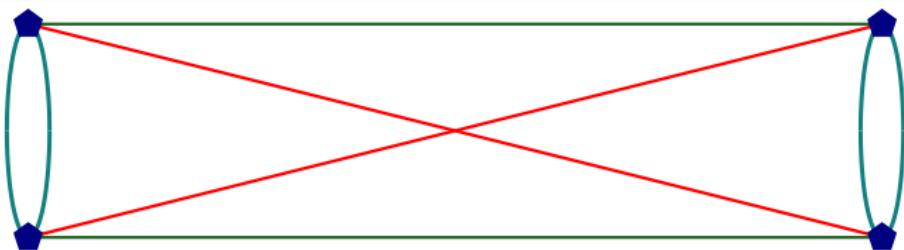
$$\overrightarrow{OM} = \rho \vec{N} \quad (1)$$



$S^-$ ,  $\rho = -r$  et  $\vec{N}$  est rentrant.

## Génération d'un cylindre ou d'un cône de révolution

Deux cercles dans deux plans parallèles, les segments sont dans un plan perpendiculaire aux plans précités :



Même rayon algébrique  $\Rightarrow$  cylindre de révolution

Rayons opposés  $\Rightarrow$  cône de révolution

## Cylindre de révolution - Cône de révolution

Axe de la surface : axe des ordonnées; surface « centrée » à l'origine.

## Cylindre de révolution - Cône de révolution

Axe de la surface : axe des ordonnées; surface « centrée » à l'origine.

Définition (Définition géométrique de l'objet `CylConeGeom`)

1) *Premier rayon* : `let rayon1 = 0.5;`

## Cylindre de révolution - Cône de révolution

Axe de la surface : axe des ordonnées; surface « centrée » à l'origine.

Définition (Définition géométrique de l'objet `CylConeGeom` )

- 1) *Premier rayon* : `let rayon1 = 0.5;`
- 2) *Second rayon* : `let rayon2 = -0.25; // cone`

## Cylindre de révolution - Cône de révolution

Axe de la surface : axe des ordonnées; surface « centrée » à l'origine.

Définition (Définition géométrique de l'objet `CylConeGeom` )

- 1) *Premier rayon* : `let rayon1 = 0.5;`
- 2) *Second rayon* : `let rayon2 = -0.25; // cone`
- 3) *Hauteur* : `let hauteur = 1.5;`

## Cylindre de révolution - Cône de révolution

Axe de la surface : axe des ordonnées; surface « centrée » à l'origine.

Définition (Définition géométrique de l'objet `CylConeGeom`)

- 1) *Premier rayon* : `let rayon1 = 0.5;`
- 2) *Second rayon* : `let rayon2 = -0.25; // cone`
- 3) *Hauteur* : `let hauteur = 1.5;`
- 4) *Nombre de points par cercles* : `let nbePtsCercle = 60;`

## Cylindre de révolution - Cône de révolution

Axe de la surface : axe des ordonnées; surface « centrée » à l'origine.

Définition (Définition géométrique de l'objet `CylConeGeom` )

- 1) *Premier rayon* : `let rayon1 = 0.5;`
- 2) *Second rayon* : `let rayon2 = -0.25; // cone`
- 3) *Hauteur* : `let hauteur = 1.5;`
- 4) *Nombre de points par cercles* : `let nbePtsCercle = 60;`
- 5) *Nombre de points par génératrices* : `let nbePtsGenera = 2;`

## Cylindre de révolution - Cône de révolution

Axe de la surface : axe des ordonnées; surface « centrée » à l'origine.

Définition (Définition géométrique de l'objet `CylConeGeom` )

- 1) *Premier rayon* : `let rayon1 = 0.5 ;`
- 2) *Second rayon* : `let rayon2 = -0.25 ; // cone`
- 3) *Hauteur* : `let hauteur = 1.5 ;`
- 4) *Nombre de points par cercles* : `let nbePtsCercle = 60 ;`
- 5) *Nombre de points par génératrices* : `let nbePtsGenera = 2 ;`
- 6) *Ouvert ou fermé* : `let bolOuvert = true ; // ou false`

## Cylindre de révolution - Cône de révolution

Axe de la surface : axe des ordonnées; surface « centrée » à l'origine.

Définition (Définition géométrique de l'objet `CylConeGeom`)

- 1) *Premier rayon* : `let rayon1 = 0.5 ;`
- 2) *Second rayon* : `let rayon2 = -0.25 ; // cone`
- 3) *Hauteur* : `let hauteur = 1.5 ;`
- 4) *Nombre de points par cercles* : `let nbePtsCercle = 60 ;`
- 5) *Nombre de points par génératrices* : `let nbePtsGenera = 2 ;`
- 6) *Ouvert ou fermé* : `let bolOuvert = true ; // ou false`
- 7) *Début de l'intervalle (cercle)* : `let theta0 = 0 ;`

## Cylindre de révolution - Cône de révolution

Axe de la surface : axe des ordonnées; surface « centrée » à l'origine.

Définition (Définition géométrique de l'objet `CylConeGeom`)

- 1) *Premier rayon* : `let rayon1 = 0.5 ;`
- 2) *Second rayon* : `let rayon2 = -0.25 ; // cone`
- 3) *Hauteur* : `let hauteur = 1.5 ;`
- 4) *Nombre de points par cercles* : `let nbePtsCercle = 60 ;`
- 5) *Nombre de points par génératrices* : `let nbePtsGenera = 2 ;`
- 6) *Ouvert ou fermé* : `let bolOuvert = true ; // ou false`
- 7) *Début de l'intervalle (cercle)* : `let theta0 = 0 ;`
- 8) *Longueur de l'intervalle* : `let thetaLg = 2 *Math.PI ;`

## Cylindre de révolution - Cône de révolution

Axe de la surface : axe des ordonnées; surface « centrée » à l'origine.

Définition (Définition géométrique de l'objet `CylConeGeom`)

- 1) *Premier rayon* : `let rayon1 = 0.5;`
- 2) *Second rayon* : `let rayon2 = -0.25; // cone`
- 3) *Hauteur* : `let hauteur = 1.5;`
- 4) *Nombre de points par cercles* : `let nbePtsCercle = 60;`
- 5) *Nombre de points par génératrices* : `let nbePtsGenera = 2;`
- 6) *Ouvert ou fermé* : `let bolOuvert = true; // ou false`
- 7) *Début de l'intervalle ( cercle)* : `let theta0 = 0;`
- 8) *Longueur de l'intervalle* : `let thetaLg = 2 *Math.PI;`
- 9) *Définir l'objet géométrique :*  
`let CylConeGeom = new THREE.CylinderGeometry(rayon1, rayon2,  
 hauteur, nbePtsCercle, nbePtsGenera, bolOuvert,  
 theta0, thetaLg);`

## Cylindre de révolution - Cône de révolution

## Définition (Propriétés géométriques)

- 1) *Premier rayon : CylConeGeom.parameters.radiusTop*

## Cylindre de révolution - Cône de révolution

## Définition (Propriétés géométriques)

- 1) *Premier rayon : CylConeGeom .parameters.radiusTop*
- 2) *Second rayon : CylConeGeom .parameters.radiusBottom*

## Cylindre de révolution - Cône de révolution

## Définition (Propriétés géométriques)

- 1) *Premier rayon* : `CylConeGeom .parameters.radiusTop`
- 2) *Second rayon* : `CylConeGeom .parameters.radiusBottom`
- 3) *Hauteur* : `CylConeGeom .parameters.height`

## Cylindre de révolution - Cône de révolution

## Définition (Propriétés géométriques)

- 1) *Premier rayon* : `CylConeGeom .parameters.radiusTop`
- 2) *Second rayon* : `CylConeGeom .parameters.radiusBottom`
- 3) *Hauteur* : `CylConeGeom .parameters.height`
- 4) *Nombre de points par cercles* :  
`CylConeGeom .parameters.radialSegments`

## Cylindre de révolution - Cône de révolution

## Définition (Propriétés géométriques)

- 1) *Premier rayon* : `CylConeGeom .parameters.radiusTop`
- 2) *Second rayon* : `CylConeGeom .parameters.radiusBottom`
- 3) *Hauteur* : `CylConeGeom .parameters.height`
- 4) *Nombre de points par cercles* :  
`CylConeGeom .parameters.radialSegments`
- 5) *Nombre de points par génératrices* :  
`CylConeGeom .parameters.heightSegments`

## Cylindre de révolution - Cône de révolution

## Définition (Propriétés géométriques)

- 1) *Premier rayon : CylConeGeom .parameters.radiusTop*
- 2) *Second rayon : CylConeGeom .parameters.radiusBottom*
- 3) *Hauteur : CylConeGeom .parameters.height*
- 4) *Nombre de points par cercles :*  
*CylConeGeom .parameters.radialSegments*
- 5) *Nombre de points par génératrices :*  
*CylConeGeom .parameters.heightSegments*
- 6) *Ouvert ou fermé : CylConeGeom .parameters.openEnded*

## Cylindre de révolution - Cône de révolution

## Définition (Propriétés géométriques)

- 1) *Premier rayon* : `CylConeGeom .parameters.radiusTop`
- 2) *Second rayon* : `CylConeGeom .parameters.radiusBottom`
- 3) *Hauteur* : `CylConeGeom .parameters.height`
- 4) *Nombre de points par cercles* :  
`CylConeGeom .parameters.radialSegments`
- 5) *Nombre de points par génératrices* :  
`CylConeGeom .parameters.heightSegments`
- 6) *Ouvert ou fermé* : `CylConeGeom .parameters.openEnded`
- 7) *Début de l'intervalle ( cercle)* : `CylConeGeom .parameters.thetaStart`

## Cylindre de révolution - Cône de révolution

## Définition (Propriétés géométriques)

- 1) *Premier rayon* : `CylConeGeom .parameters.radiusTop`
- 2) *Second rayon* : `CylConeGeom .parameters.radiusBottom`
- 3) *Hauteur* : `CylConeGeom .parameters.height`
- 4) *Nombre de points par cercles* :  
`CylConeGeom .parameters.radialSegments`
- 5) *Nombre de points par génératrices* :  
`CylConeGeom .parameters.heightSegments`
- 6) *Ouvert ou fermé* : `CylConeGeom .parameters.openEnded`
- 7) *Début de l'intervalle ( cercle )* : `CylConeGeom .parameters.thetaStart`
- 8) *Longueur de l'intervalle* : `CylConeGeom .parameters.thetaLength`

Traçage d'un cylindre ou d'un cône

## Cône de révolution ( cercle + sommet)

Axe du cône : axe des ordonnées ; cône « centré » à l'origine.

## Cône de révolution ( cercle + sommet )

Axe du cône : axe des ordonnées ; cône « centré » à l'origine.

Définition (Définition géométrique de l'objet `ConeGeom`)

1) *Rayon* : `let rayon = 0.25;`

## Cône de révolution ( cercle + sommet )

Axe du cône : axe des ordonnées ; cône « centré » à l'origine.

Définition (Définition géométrique de l'objet `ConeGeom`)

- 1) *Rayon* : `let rayon = 0.25;`
- 2) *Hauteur* : `let hauteur = 1.5;`

## Cône de révolution ( cercle + sommet )

Axe du cône : axe des ordonnées ; cône « centré » à l'origine.

Définition (Définition géométrique de l'objet `ConeGeom`)

- 1) *Rayon* : `let rayon = 0.25;`
- 2) *Hauteur* : `let hauteur = 1.5;`
- 3) *Nombre de points par cercles* : `let nbePtsCercle = 60;`

## Cône de révolution ( cercle + sommet )

Axe du cône : axe des ordonnées ; cône « centré » à l'origine.

Définition (Définition géométrique de l'objet `ConeGeom` )

- 1) *Rayon* : `let rayon = 0.25 ;`
- 2) *Hauteur* : `let hauteur = 1.5 ;`
- 3) *Nombre de points par cercles* : `let nbePtsCercle = 60 ;`
- 4) *Nombre de points par génératrices* : `let nbePtsGenera = 2 ;`

## Cône de révolution ( cercle + sommet )

Axe du cône : axe des ordonnées ; cône « centré » à l'origine.

Définition (Définition géométrique de l'objet `ConeGeom` )

- 1) *Rayon* : `let rayon = 0.25 ;`
- 2) *Hauteur* : `let hauteur = 1.5 ;`
- 3) *Nombre de points par cercles* : `let nbePtsCercle = 60 ;`
- 4) *Nombre de points par génératrices* : `let nbePtsGenera = 2 ;`
- 5) *Ouvert ou fermé* : `let bolOuvert = true ; // ou false`

## Cône de révolution ( cercle + sommet )

Axe du cône : axe des ordonnées ; cône « centré » à l'origine.

Définition (Définition géométrique de l'objet `ConeGeom` )

- 1) *Rayon* : `let rayon = 0.25 ;`
- 2) *Hauteur* : `let hauteur = 1.5 ;`
- 3) *Nombre de points par cercles* : `let nbePtsCercle = 60 ;`
- 4) *Nombre de points par génératrices* : `let nbePtsGenera = 2 ;`
- 5) *Ouvert ou fermé* : `let bolOuvert = true ; // ou false`
- 6) *Début de l'intervalle ( cercle )* : `let theta0 = 0 ;`

## Cône de révolution ( cercle + sommet)

Axe du cône : axe des ordonnées ; cône « centré » à l'origine.

Définition (Définition géométrique de l'objet `ConeGeom`)

- 1) *Rayon* : `let rayon = 0.25 ;`
- 2) *Hauteur* : `let hauteur = 1.5 ;`
- 3) *Nombre de points par cercles* : `let nbePtsCercle = 60 ;`
- 4) *Nombre de points par génératrices* : `let nbePtsGenera = 2 ;`
- 5) *Ouvert ou fermé* : `let bolOuvert = true ; // ou false`
- 6) *Début de l'intervalle ( cercle )* : `let theta0 = 0 ;`
- 7) *Longueur de l'intervalle* : `let thetaLg = 2 *Math.PI ;`

## Cône de révolution ( cercle + sommet )

Axe du cône : axe des ordonnées ; cône « centré » à l'origine.

Définition (Définition géométrique de l'objet `ConeGeom` )

- 1) *Rayon* : `let rayon = 0.25 ;`
- 2) *Hauteur* : `let hauteur = 1.5 ;`
- 3) *Nombre de points par cercles* : `let nbePtsCercle = 60 ;`
- 4) *Nombre de points par génératrices* : `let nbePtsGenera = 2 ;`
- 5) *Ouvert ou fermé* : `let bolOuvert = true ; // ou false`
- 6) *Début de l'intervalle ( cercle )* : `let theta0 = 0 ;`
- 7) *Longueur de l'intervalle* : `let thetaLg = 2 *Math.PI ;`
- 8) *Définir l'objet géométrique* :  
`let ConeGeom = new THREE.ConeGeometry(rayon, hauteur,  
nbePtsCercle, nbePtsGenera, bolOuvert, theta0, thetaLg );`

Traçage d'un cône

## Cône de révolution ( cercle + sommet)

### Définition (Propriétés géométriques)

1) *Rayon : ConeGeom.parameters.radius*

## Cône de révolution ( cercle + sommet)

## Définition (Propriétés géométriques)

- 1) *Rayon* : `ConeGeom.parameters.radius`
- 2) *Hauteur* : `ConeGeom.parameters.height`

## Cône de révolution ( cercle + sommet)

## Définition (Propriétés géométriques)

- 1) *Rayon* : `ConeGeom.parameters.radius`
- 2) *Hauteur* : `ConeGeom.parameters.height`
- 3) *Nombre de points par cercles* : `ConeGeom.parameters.radialSegments`

## Cône de révolution ( cercle + sommet)

## Définition (Propriétés géométriques)

- 1) *Rayon* : `ConeGeom .parameters.radius`
- 2) *Hauteur* : `ConeGeom .parameters.height`
- 3) *Nombre de points par cercles* : `ConeGeom .parameters.radialSegments`
- 4) *Nombre de points par génératrices* :  
`ConeGeom .parameters.heightSegments`

## Cône de révolution ( cercle + sommet)

## Définition (Propriétés géométriques)

- 1) *Rayon* : `ConeGeom .parameters.radius`
- 2) *Hauteur* : `ConeGeom .parameters.height`
- 3) *Nombre de points par cercles* : `ConeGeom .parameters.radialSegments`
- 4) *Nombre de points par génératrices* :  
`ConeGeom .parameters.heightSegments`
- 5) *Ouvert ou fermé* : `ConeGeom .parameters.openEnded`

## Cône de révolution ( cercle + sommet)

## Définition (Propriétés géométriques)

- 1) *Rayon* : `ConeGeom .parameters.radius`
- 2) *Hauteur* : `ConeGeom .parameters.height`
- 3) *Nombre de points par cercles* : `ConeGeom .parameters.radialSegments`
- 4) *Nombre de points par génératrices* :  
`ConeGeom .parameters.heightSegments`
- 5) *Ouvert ou fermé* : `ConeGeom .parameters.openEnded`
- 6) *Début de l'intervalle ( cercle)* : `ConeGeom .parameters.thetaStart`

## Cône de révolution ( cercle + sommet)

## Définition (Propriétés géométriques)

- 1) *Rayon* : `ConeGeom .parameters.radius`
- 2) *Hauteur* : `ConeGeom .parameters.height`
- 3) *Nombre de points par cercles* : `ConeGeom .parameters.radialSegments`
- 4) *Nombre de points par génératrices* :  
`ConeGeom .parameters.heightSegments`
- 5) *Ouvert ou fermé* : `ConeGeom .parameters.openEnded`
- 6) *Début de l'intervalle ( cercle )* : `ConeGeom .parameters.thetaStart`
- 7) *Longueur de l'intervalle* : `ConeGeom .parameters.thetaLength`

Traçage d'un cône

## Tore

Axe du tore : axe des cotes; Centre du tore : origine du repère.

Axe du tore : axe des cotes; Centre du tore : origine du repère.

Définition (Définition géométrique de l'objet `ToreGeom`)

1) *Rayon majeur : let rayonMajeur = 2;*

Axe du tore : axe des cotes; Centre du tore : origine du repère.

Définition (Définition géométrique de l'objet `ToreGeom`)

- 1) *Rayon majeur* : `let rayonMajeur = 2;`
- 2) *Rayon mineur* : `let rayonMajeur = 1;`

Axe du tore : axe des cotes; Centre du tore : origine du repère.

Définition (Définition géométrique de l'objet `ToreGeom`)

- 1) *Rayon majeur* : `let rayonMajeur = 2;`
- 2) *Rayon mineur* : `let rayonMajeur = 1;`
- 3) *Nombre de points par méridien* : `let nbePtsMeridien = 30;`

Axe du tore : axe des cotes; Centre du tore : origine du repère.

Définition (Définition géométrique de l'objet `ToreGeom`)

- 1) *Rayon majeur* : `let rayonMajeur = 2;`
- 2) *Rayon mineur* : `let rayonMineur = 1;`
- 3) *Nombre de points par méridien* : `let nbePtsMeridien = 30;`
- 4) *Nombre de points par parallèle* : `let nbePtsParallele = 60;`

Axe du tore : axe des cotes; Centre du tore : origine du repère.

Définition (Définition géométrique de l'objet `ToreGeom`)

- 1) *Rayon majeur* : `let rayonMajeur = 2;`
- 2) *Rayon mineur* : `let rayonMineur = 1;`
- 3) *Nombre de points par méridien* : `let nbePtsMeridien = 30;`
- 4) *Nombre de points par parallèle* : `let nbePtsParallele = 60;`
- 5) *Longueur du tore* : `let LgArc = 2 *Math.PI;`

Axe du tore : axe des cotes; Centre du tore : origine du repère.

Définition (Définition géométrique de l'objet `ToreGeom`)

- 1) *Rayon majeur* : `let rayonMajeur = 2;`
- 2) *Rayon mineur* : `let rayonMajeur = 1;`
- 3) *Nombre de points par méridien* : `let nbePtsMeridien = 30;`
- 4) *Nombre de points par parallèle* : `let nbePtsParallele = 60;`
- 5) *Longueur du tore* : `let LgArc = 2 *Math.PI;`
- 6) *Définir l'objet géométrique* :

```
let ToreGeom = new THREE.TorusGeometry (rayonMajeur,  
rayonMineur, nbePtsMeridien , nbePtsParallele, LgArc );
```

Traçage d'un tore

Définition (propriétés géométriques de l'objet `ToreGeom`)

- 1) *Rayon majeur : `ToreGeom.parameters.radius`*

## Tore

Définition (propriétés géométriques de l'objet `ToreGeom`)

- 1) *Rayon majeur : `ToreGeom.parameters.radius`*
- 2) *Rayon mineur : `ToreGeom.parameters.tube`*

## Tore

Définition (propriétés géométriques de l'objet `ToreGeom`)

- 1) *Rayon majeur : `ToreGeom.parameters.radius`*
- 2) *Rayon mineur : `ToreGeom.parameters.tube`*
- 3) *Nombre de points par méridien : `ToreGeom.parameters.radialSegments`*

## Définition (propriétés géométriques de l'objet `ToreGeom`)

- 1) *Rayon majeur* : `ToreGeom.parameters.radius`
- 2) *Rayon mineur* : `ToreGeom.parameters.tube`
- 3) *Nombre de points par méridien* : `ToreGeom.parameters.radialSegments`
- 4) *Nombre de points par parallèle* : `ToreGeom.parameters.tubularSegments`

### Définition (propriétés géométriques de l'objet `ToreGeom`)

- 1) *Rayon majeur : `ToreGeom.parameters.radius`*
- 2) *Rayon mineur : `ToreGeom.parameters.tube`*
- 3) *Nombre de points par méridien : `ToreGeom.parameters.radialSegments`*
- 4) *Nombre de points par parallèle : `ToreGeom.parameters.tubularSegments`*
- 5) *Longueur du tore : `ToreGeom.parameters.arc`*

Traçage d'un tore

## Définition (Surface obtenue par rotation d'une courbe)

*Soit  $\gamma$  une courbe située dans le demi-plan des abscisses positives dans le plan d'équation  $z = 0$ .*

## Définition (Surface obtenue par rotation d'une courbe)

*Soit  $\gamma$  une courbe située dans le demi-plan des abscisses positives dans le plan d'équation  $z = 0$ .*

*La surface lathe est la surface obtenue par rotation de la courbe  $\gamma$  autour de l'axe des ordonnées.*

## Lathe

Définition (Surface obtenue par rotation d'une courbe)

*Soit  $\gamma$  une courbe située dans le demi-plan des abscisses positives dans le plan d'équation  $z = 0$ .*

*La surface lathe est la surface obtenue par rotation de la courbe  $\gamma$  autour de l'axe des ordonnées.*

Définition (Construction géométrique d'une Lathe)

1) *Tableau des points de la courbe  $\gamma$  : `tabPoints`*

## Lathe

## Définition (Surface obtenue par rotation d'une courbe)

*Soit  $\gamma$  une courbe située dans le demi-plan des abscisses positives dans le plan d'équation  $z = 0$ .*

*La surface lathe est la surface obtenue par rotation de la courbe  $\gamma$  autour de l'axe des ordonnées.*

## Définition (Construction géométrique d'une Lathe)

- 1) *Tableau des points de la courbe  $\gamma$  : `tabPoints`*
- 2) *Nombre de points par cercles : `let nbePtRot = ...`*

## Lathe

## Définition (Surface obtenue par rotation d'une courbe)

Soit  $\gamma$  une courbe située dans le demi-plan des abscisses positives dans le plan d'équation  $z = 0$ .

La surface lathe est la surface obtenue par rotation de la courbe  $\gamma$  autour de l'axe des ordonnées.

## Définition (Construction géométrique d'une Lathe)

- 1) Tableau des points de la courbe  $\gamma$  : `tabPoints`
- 2) Nombre de points par cercles : `let nbePtRot = ...`
- 3) Angle de départ : `let theta0 = ...`

## Définition (Surface obtenue par rotation d'une courbe)

*Soit  $\gamma$  une courbe située dans le demi-plan des abscisses positives dans le plan d'équation  $z = 0$ .*

*La surface lathe est la surface obtenue par rotation de la courbe  $\gamma$  autour de l'axe des ordonnées.*

## Définition (Construction géométrique d'une Lathe)

- 1) *Tableau des points de la courbe  $\gamma$  : `tabPoints`*
- 2) *Nombre de points par cercles : `let nbePtRot = ...`*
- 3) *Angle de départ : `let theta0 = ...`*
- 4) *Longueur des arcs de cercles : `let thetaLg = ...`*

## Définition (Surface obtenue par rotation d'une courbe)

Soit  $\gamma$  une courbe située dans le demi-plan des abscisses positives dans le plan d'équation  $z = 0$ .

La surface lathe est la surface obtenue par rotation de la courbe  $\gamma$  autour de l'axe des ordonnées.

## Définition (Construction géométrique d'une Lathe)

- 1) Tableau des points de la courbe  $\gamma$  : `tabPoints`
- 2) Nombre de points par cercles : `let nbePtRot = ...`
- 3) Angle de départ : `let theta0 = ...`
- 4) Longueur des arcs de cercles : `let thetaLg = ...`
- 5) Définition géométrique de la Lathe :

```
let latheGeometry = new THREE.LatheGeometry(tabPoints,  
    nbePtRot, theta0, thetaLg);
```

Lathe : surface qui a fait 1789

## Surfaces paramétriques avec THREE.ParametricGeometry

Tracer une surface paramétrique définie sur le pavé  $[u_0; u_1] \times [v_0; v_1]$ .

## Surfaces paramétriques avec THREE.ParametricGeometry

Tracer une surface paramétrique définie sur le pavé  $[u_0; u_1] \times [v_0; v_1]$ .

### Principe

- 1) Donner les valeurs aux bornes du côté du pavé :

```
let u0 = .... , u1 = .... ;  
let v0 = .... , v1 = .... ;
```

## Surfaces paramétriques avec THREE.ParametricGeometry

Tracer une surface paramétrique définie sur le pavé  $[u_0; u_1] \times [v_0; v_1]$ .

### Principe

- 1) *Donner les valeurs aux bornes du côté du pavé :*

```
let u0 = .... , u1 = .... ;
```

```
let v0 = .... , v1 = .... ;
```

- 2) *Calculer les coordonnées des points via la fonction SurfPara qui calcule les coordonnées des points en fonction des variables  $u$  et  $v$ ;*

## Surfaces paramétriques avec THREE.ParametricGeometry

Tracer une surface paramétrique définie sur le pavé  $[u_0; u_1] \times [v_0; v_1]$ .

### Principe

- 1) Donner les valeurs aux bornes du côté du pavé :

```
let u0 = ..., u1 = ... ;  
let v0 = ..., v1 = ... ;
```

- 2) Calculer les coordonnées des points via la fonction *SurfPara* qui calcule les coordonnées des points en fonction des variables *u* et *v*;
- 3) Dans la fonction *init()*, définir le nombre de points *NbeU* dans  $[u_0; u_1]$  et *NbeV* dans  $[v_0; v_1]$  :

```
let NbeU = ..., NbeV = ... ;
```

## Surfaces paramétriques avec THREE.ParametricGeometry

Tracer une surface paramétrique définie sur le pavé  $[u_0; u_1] \times [v_0; v_1]$ .

## Principe

- 1) Donner les valeurs aux bornes du côté du pavé :

```
let u0 = ..., u1 = ... ;  
let v0 = ..., v1 = ... ;
```

- 2) Calculer les coordonnées des points via la fonction *SurfPara* qui calcule les coordonnées des points en fonction des variables *u* et *v*;

- 3) Dans la fonction *init()*, définir le nombre de points *NbeU* dans  $[u_0; u_1]$  et *NbeV* dans  $[v_0; v_1]$  :

```
let NbeU = ..., NbeV = ... ;
```

- 4) Définir la surface paramétrique géométrique :

```
let SurfParaGeom = new THREE.ParametricGeometry (SurfPara,  
NbeU, NbeV);
```

## Surfaces paramétriques avec THREE.ParametricGeometry

Tracer une surface paramétrique définie sur le pavé  $[u_0; u_1] \times [v_0; v_1]$ .

## Principe

- 1) Donner les valeurs aux bornes du côté du pavé :

```
let u0 = ..., u1 = ... ;  
let v0 = ..., v1 = ... ;
```

- 2) Calculer les coordonnées des points via la fonction *SurfPara* qui calcule les coordonnées des points en fonction des variables *u* et *v*;

- 3) Dans la fonction *init()*, définir le nombre de points *NbeU* dans  $[u_0; u_1]$  et *NbeV* dans  $[v_0; v_1]$  :

```
let NbeU = ..., NbeV = ... ;
```

- 4) Définir la surface paramétrique géométrique :

```
let SurfParaGeom = new THREE.ParametricGeometry(SurfPara,  
NbeU, NbeV);
```

- 5) Définir la surface paramétrique avec le matériau :

```
let SurfParaPhong = new THREE.Mesh(SurfPara, MaterialPhong);
```

## Surfaces paramétriques avec THREE.ParametricGeometry

Définition (La variable `resultCalcul`)

*La variable `resultCalcul` représente les résultats du calcul des coordonnées des points de la surface.*

## Surfaces paramétriques avec THREE.ParametricGeometry

Définition (La variable `resultCalcul`)

La variable `resultCalcul` représente les résultats du calcul des coordonnées des points de la surface.

Définition (La fonction `SurfPara`)

```
SurfPara = function(u, v, resultCalcul) {  
    u = u0 + u * (u1 - u0 )//intervalle en U
```

## Surfaces paramétriques avec THREE.ParametricGeometry

Définition (La variable `resultCalcul`)

La variable `resultCalcul` représente les résultats du calcul des coordonnées des points de la surface.

Définition (La fonction `SurfPara`)

```
SurfPara = function(u, v, resultCalcul) {  
    u = u0 + u * (u1 - u0) //intervalle en U  
    v = v0 + v * (v1 - v0) //intervalle en V
```

## Surfaces paramétriques avec THREE.ParametricGeometry

Définition (La variable `resultCalcul`)

La variable `resultCalcul` représente les résultats du calcul des coordonnées des points de la surface.

Définition (La fonction `SurfPara`)

```
SurfPara = function(u, v, resultCalcul) {  
    u = u0 + u * (u1 - u0) //intervalle en U  
    v = v0 + v * (v1 - v0) //intervalle en V  
    let result = resultCalcul || new THREE.Vector3();
```

## Surfaces paramétriques avec THREE.ParametricGeometry

Définition (La variable **resultCalcul**)

*La variable **resultCalcul** représente les résultats du calcul des coordonnées des points de la surface.*

Définition (La fonction **SurfPara**)

```
SurfPara = function(u, v, resultCalcul) {  
    u = u0 + u * (u1 - u0) //intervalle en U  
    v = v0 + v * (v1 - v0) //intervalle en V  
    let result = resultCalcul || new THREE.Vector3();  
    let x = ...; // abscisse de la surface
```

## Surfaces paramétriques avec THREE.ParametricGeometry

Définition (La variable **resultCalcul**)

La variable **resultCalcul** représente les résultats du calcul des coordonnées des points de la surface.

Définition (La fonction **SurfPara**)

```
SurfPara = function(u, v, resultCalcul) {  
    u = u0 + u * (u1 - u0) //intervalle en U  
    v = v0 + v * (v1 - v0) //intervalle en V  
    let result = resultCalcul || new THREE.Vector3();  
    let x = ...; // abscisse de la surface  
    let y = ...; // ordonnée de la surface
```

## Surfaces paramétriques avec THREE.ParametricGeometry

Définition (La variable **resultCalcul**)

La variable **resultCalcul** représente les résultats du calcul des coordonnées des points de la surface.

Définition (La fonction **SurfPara**)

```
SurfPara = function(u, v, resultCalcul) {  
    u = u0 + u * (u1 - u0) //intervalle en U  
    v = v0 + v * (v1 - v0) //intervalle en V  
    let result = resultCalcul || new THREE.Vector3();  
    let x = ...; // abscisse de la surface  
    let y = ...; // ordonnée de la surface  
    let z = ...; // cote de la surface  
    return result.set(x, y, z);  
}
```

## Surfaces paramétriques avec THREE.ParametricGeometry

Définition (La variable **resultCalcul**)

La variable **resultCalcul** représente les résultats du calcul des coordonnées des points de la surface.

Définition (La fonction **SurfPara**)

```
SurfPara = function(u, v, resultCalcul) {  
    u = u0 + u * (u1 - u0) //intervalle en U  
    v = v0 + v * (v1 - v0) //intervalle en V  
    let result = resultCalcul || new THREE.Vector3();  
    let x = ...; // abscisse de la surface  
    let y = ...; // ordonnée de la surface  
    let z = ...; // cote de la surface  
    return result.set(x, y, z);  
}
```

Surfaces paramétriques : un plan  $\mathcal{P}_1$  d'équation  $ax + by + cz + d = 0$

Soit  $A(x_A; y_A; z_A)$  un point du plan affine  $\mathcal{P}_1$ .

### Rappel (Equation paramétrique d'un plan)

1) Vecteur normal :  $\vec{N}(a; b; c)$  ;

Surfaces paramétriques : un plan  $\mathcal{P}_1$  d'équation  $ax + by + cz + d = 0$

Soit  $A(x_A; y_A; z_A)$  un point du plan affine  $\mathcal{P}_1$ .

### Rappel (Equation paramétrique d'un plan)

- 1) Vecteur normal :  $\vec{N}(a; b; c)$  ;
- 2) Choix d'un vecteur de  $\overrightarrow{\mathcal{P}_1}$  :  $\vec{e}_1(-c; -c; a+b)$  ;

Surfaces paramétriques : un plan  $\mathcal{P}_1$  d'équation  $ax + by + cz + d = 0$

Soit  $A(x_A; y_A; z_A)$  un point du plan affine  $\mathcal{P}_1$ .

### Rappel (Equation paramétrique d'un plan)

- 1) Vecteur normal :  $\vec{N}(a; b; c)$  ;
- 2) Choix d'un vecteur de  $\vec{\mathcal{P}}_1$  :  $\vec{e}_1(-c; -c; a+b)$  ;
- 3) Choix d'une base orthogonale directe  $(\vec{e}_1; \vec{e}_2)$  de  $\vec{\mathcal{P}}_1$  :  $\vec{e}_2 = \vec{N} \wedge \vec{e}_1$  ;

Surfaces paramétriques : un plan  $\mathcal{P}_1$  d'équation  $ax + by + cz + d = 0$

Soit  $A(x_A; y_A; z_A)$  un point du plan affine  $\mathcal{P}_1$ .

### Rappel (Equation paramétrique d'un plan)

- 1) Vecteur normal :  $\vec{N}(a; b; c)$  ;
- 2) Choix d'un vecteur de  $\mathcal{P}_1$  :  $\vec{e}_1(-c; -c; a+b)$  ;
- 3) Choix d'une base orthogonale directe  $(\vec{e}_1; \vec{e}_2)$  de  $\mathcal{P}_1$  :  $\vec{e}_2 = \vec{N} \wedge \vec{e}_1$  ;
- 4) Equation paramétrique du plan  $\overrightarrow{OM(u; v)} = \overrightarrow{OA} + u\vec{e}_1 + v\vec{e}_2$ .

Surfaces paramétriques : un plan  $\mathcal{P}_1$  d'équation  $ax + by + cz + d = 0$

Soit  $A(x_A; y_A; z_A)$  un point du plan affine  $\mathcal{P}_1$ .

### Rappel (Equation paramétrique d'un plan)

- 1) Vecteur normal :  $\vec{N}(a; b; c)$  ;
- 2) Choix d'un vecteur de  $\mathcal{P}_1$  :  $\vec{e_1}(-c; -c; a+b)$  ;
- 3) Choix d'une base orthogonale directe  $(\vec{e_1}; \vec{e_2})$  de  $\mathcal{P}_1$  :  $\vec{e_2} = \vec{N} \wedge \vec{e_1}$  ;
- 4) Equation paramétrique du plan  $\overrightarrow{OM(u; v)} = \overrightarrow{OA} + u\vec{e_1} + v\vec{e_2}$ .

### Exemple ( Variables globales et fonction PlanPara)

#### 1) Variables globales

- a) Vecteur normal unitaire au plan (orienté) `vecN` :

```
let a = ..., b = ..., c = ...;
let vecN = new THREE.Vector3(a, b, c);
vecN.normalize();
```

Surfaces paramétriques : un plan  $\mathcal{P}_1$  d'équation  $ax + by + cz + d = 0$ 

## Rappel (Equation paramétrique d'un plan)

- 1) Vecteur normal :  $\vec{N}(a; b; c)$  ;
- 2) Choix d'un vecteur de  $\vec{\mathcal{P}}_1$  :  $\vec{e}_1(-c; -c; a+b)$  ;
- 3) Choix d'une base orthogonale directe  $(\vec{e}_1; \vec{e}_2)$  de  $\vec{\mathcal{P}}_1$  :  $\vec{e}_2 = \vec{N} \wedge \vec{e}_1$  ;
- 4) Equation paramétrique du plan  $\overrightarrow{OM(u; v)} = \overrightarrow{OA} + u\vec{e}_1 + v\vec{e}_2$ .

## Exemple ( Variables globales et fonction PlanPara)

## 1) Variables globales

- a) Vecteur normal unitaire au plan (orienté) `vecN` :

```
let a = ..., b = ..., c = ...;
let vecN = new THREE.Vector3(a, b, c);
vecN.normalize();
```

- b) Base directe du plan engendré par le vecteur `vecN` :

```
// premier vecteur normalisé de la base du plan
let vec_e1 = new THREE.Vector3(-c, -c, a+b);
vec_e1.normalize();
```

Surfaces paramétriques : un plan  $\mathcal{P}_1$  d'équation  $ax + by + cz + d = 0$ 

## Exemple ( Variables globales et fonction PlanPara)

## 1) Variables globales

- a) Vecteur normal unitaire au plan (orienté) `vecN` :

```
let a = ..., b = ..., c = ...;  
let vecN = new THREE.Vector3(a, b, c);  
vecN.normalize();
```

- b) Base directe du plan engendré par le vecteur `vecN` :

```
// premier vecteur normalise de la base du plan  
let vec_e1 = new THREE.Vector3(-c, -c, a+b);  
vec_e1.normalize();  
  
// second vecteur normalise de la base du plan  
let vec_e2 = new THREE.Vector3(0, 0, 0);  
vec_e2.crossVectors(vecN, vec_e1);
```

Surfaces paramétriques : un plan  $\mathcal{P}_1$  d'équation  $ax + by + cz + d = 0$ 

Exemple ( Variables globales et fonction PlanPara)

## 1) Variables globales

- a) Vecteur normal unitaire au plan (orienté) `vecN` :

```
let a = ..., b = ..., c = ...;
let vecN = new THREE.Vector3(a, b, c);
vecN.normalize();
```

- b) Base directe du plan engendré par le vecteur `vecN` :

```
// premier vecteur normalise de la base du plan
let vec_e1 = new THREE.Vector3(-c, -c, a+b);
vec_e1.normalize();
// second vecteur normalise de la base du plan
let vec_e2 = new THREE.Vector3(0, 0, 0);
vec_e2.crossVectors(vecN, vec_e1);
```

- c) Point du plan :

```
let xA = ..., yA = ..., zA = ...;
let ptA = new THREE.Vector3(xA, yA, zA);
```

Surfaces paramétriques : un plan  $\mathcal{P}_1$  d'équation  $ax + by + cz + d = 0$ 

## Exemple ( Variables globales et fonction PlanPara)

## 1) Variables globales

- a) Vecteur normal unitaire au plan (orienté) `vecN` :

```
let a = ..., b = ..., c = ...;
let vecN = new THREE.Vector3(a, b, c);
vecN.normalize();
```

- b) Base directe du plan engendré par le vecteur `vecN` :

```
// premier vecteur normalisé de la base du plan
let vec_e1 = new THREE.Vector3(-c, -c, a+b);
vec_e1.normalize();
// second vecteur normalisé de la base du plan
let vec_e2 = new THREE.Vector3(0, 0, 0);
vec_e2.crossVectors(vecN, vec_e1);
```

- c) Point du plan :

```
let xA = ..., yA = ..., zA = ...;
let pta = new THREE.Vector3(xA, yA, zA);
```

- d) Bornes (paramétriques) du plan :

```
let u0 = ..., u1 = ...; // en u
let v0 = ..., v1 = ...; // en v
```

Surfaces paramétriques : un plan  $\mathcal{P}_1$  d'équation  $ax + by + cz + d = 0$ 

## Exemple ( Variables globales et fonction PlanPara)

## 1) Variables globales

- b) Base directe du plan engendré par le vecteur `vecN` :

```
// premier vecteur normalisé de la base du plan
let vec_e1 = new THREE.Vector3(-c, -c, a+b);
vec_e1.normalize();

// second vecteur normalisé de la base du plan
let vec_e2 = new THREE.Vector3(0, 0, 0);
vec_e2.crossVectors(vecN, vec_e1);
```

- c) Point du plan :

```
let xA = ..., yA = ..., zA = ...;
let ptA = new THREE.Vector3(xA, yA, zA);
```

- d) Bornes (paramétriques) du plan :

```
let u0 = ..., u1 = ...; // en u
let v0 = ... v1 = ...; // en v
```

## 2) Fonction PlanPara

Surfaces paramétriques : un plan  $\mathcal{P}_1$  d'équation  $ax + by + cz + d = 0$ 

Exemple ( Variables globales et fonction PlanPara)

## 1) Variables globales

- b) Base directe du plan engendré par le vecteur `vecN` :

```
// premier vecteur normalisé de la base du plan
let vec_e1 = new THREE.Vector3(-c, -c, a+b);
vec_e1.normalize();

// second vecteur normalisé de la base du plan
let vec_e2 = new THREE.Vector3(0, 0, 0);
vec_e2.crossVectors(vecN, vec_e1);
```

- c) Point du plan :

```
let xA = ..., yA = ..., zA = ...;
let ptA = new THREE.Vector3(xA, yA, zA);
```

- d) Bornes (paramétriques) du plan :

```
let u0 = ..., u1 = ...; // en u
let v0 = ..., v1 = ...; // en v
```

## 2) Fonction PlanPara

```
PlanPara = function (u, v, resultCalcul) {
```

Surfaces paramétriques : un plan  $\mathcal{P}_1$  d'équation  $ax + by + cz + d = 0$ 

## Exemple ( Variables globales et fonction PlanPara)

## 1) Variables globales

## c) Point du plan :

```
let xA = ..., yA = ..., zA = ...;
let ptA = new THREE.Vector3(xA, yA, zA);
```

## d) Bornes (paramétriques) du plan :

```
let u0 = ..., u1 = ...; // en u
let v0 = ..., v1 = ...; // en v
```

## 2) Fonction PlanPara

```
PlanPara = function (u, v, resultCalcul) {
    u = u0 + u * (u1 - u0); // intervalle en U
    v = v0 + v * (v1 - v0); // intervalle en V
    let result = resultCalcul // new THREE.Vector3();
```

Surfaces paramétriques : un plan  $\mathcal{P}_1$  d'équation  $ax + by + cz + d = 0$ 

Exemple ( Variables globales et fonction PlanPara)

## 1) Variables globales

## d) Bornes (paramétriques) du plan :

```
let u0 = ...; u1 = ...; // en u
let v0 = ...; v1 = ...; // en v
```

## 2) Fonction PlanPara

```
PlanPara = function (u, v, resultCalcul) {
    u = u0 + u * (u1 - u0); // intervalle en U
    v = v0 + v * (v1 - v0); // intervalle en V
    let result = resultCalcul || new THREE.Vector3();
    let x = xA + u * vec_e1.x + v * vec_e2.x;
    let y = yA + u * vec_e1.y + v * vec_e2.y;
    let z = zA + u * vec_e1.z + v * vec_e2.z;
```

Surfaces paramétriques : un plan  $\mathcal{P}_1$  d'équation  $ax + by + cz + d = 0$

## Exemple ( Variables globales et fonction PlanPara)

### 1) Variables globales

d) Bornes (paramétriques) du plan :

```
let u0 = ...; u1 = ...; // en u
let v0 = ...; v1 = ...; // en v
```

### 2) Fonction PlanPara

```
PlanPara = function (u, v, resultCalcul) {
    u = u0 + u * (u1 - u0); // intervalle en U
    v = v0 + v * (v1 - v0); // intervalle en V
    let result = resultCalcul || new THREE.Vector3();
    let x = xA + u * vec_e1.x + v * vec_e2.x;
    let y = yA + u * vec_e1.y + v * vec_e2.y;
    let z = zA + u * vec_e1.z + v * vec_e2.z;
    return result.set(x, y, z);
}; //fin fonction PlanPara
```

Plan paramétrique

## Surfaces paramétriques : une cyclide de Dupin

Equation d'une cyclide de Dupin,  $b = \sqrt{a^2 - c^2}$  :

$$\left\{ \begin{array}{lcl} x(u,v) & = & \frac{\mu(c - a \cos(u) \cos(v)) + b^2 \cos(u)}{a - c \cos(u) \cos(v)} \\ y(u,v) & = & \frac{b \sin(u) \times (a - \mu \cos(v))}{a - c \cos(u) \cos(v)} \\ z(u,v) & = & \frac{b \sin(v) \times (c \cos(u) - \mu)}{a - c \cos(u) \cos(v)} \end{array} \right.$$

## Surfaces paramétriques : une cyclide de Dupin

Equation d'une cyclide de Dupin,  $b = \sqrt{a^2 - c^2}$  :

$$\left\{ \begin{array}{lcl} x(u,v) & = & \frac{\mu(c - a \cos(u) \cos(v)) + b^2 \cos(u)}{a - c \cos(u) \cos(v)} \\ y(u,v) & = & \frac{b \sin(u) \times (a - \mu \cos(v))}{a - c \cos(u) \cos(v)} \\ z(u,v) & = & \frac{b \sin(v) \times (c \cos(u) - \mu)}{a - c \cos(u) \cos(v)} \end{array} \right.$$

## Exemple

- 1) Définition des constantes idoines :

```
let a = 8, c = 1, mu = 3.1;
let b = Math.sqrt(Math.pou(a,2)-Math.pou(c,2));
let u0 = 0, v0 = 0 //depart de l'intervalle
let u1 = 2 *Math.PI, v1 = 2 *Math.PI; //fin de l'intervalle
```

## Surfaces paramétriques : une cyclide de Dupin

## Exemple

- 1) Définition des constantes idoines :

```
let a = 8, c = 1, mu = 3.1;
let b = Math.sqrt(Math.pow(a,2)-Math.pow(c,2));
let u0 = 0, v0 = 0; //depart de l'intervalle
let u1 = 2 *Math.PI, v1 = 2 *Math.PI; //fin de l'intervalle
```

- 2) La fonction SurfPara appelée CyclideDupin4 :

```
CyclideDupin4 = function (u, v, resultCalcul) {
  u = u0 + u * (u1 - u0) //intervalle en U
  v = v0 + v * (v1 - v0) //intervalle en V
  let result = resultCalcul || new THREE.Vector3();
  let den = (a-c*Math.cos(u)*Math.cos(v));
  let x = (mu*(c-a*Math.cos(u)*Math.cos(v))+b*b*Math.cos(u));
  let y = b*Math.sin(u)*(a-mu*Math.cos(v));
  let z = b*Math.sin(v)*(c*Math.cos(u)-mu);
  return result.set( x/den, y/den, z/den );
}; //fin fonction CyclideDupin4
```

Cyclide de Dupin

## Surfaces paramétriques : un ruban de Möbius

Une équation paramétrique du ruban de Möbius est :

$$\begin{cases} x(u, v) = \left(a + u \cos\left(\frac{v}{2}\right)\right) \cos(v) \\ y(u, v) = \left(a + u \cos\left(\frac{v}{2}\right)\right) \sin(v) \\ z(u, v) = u \sin\left(\frac{v}{2}\right) \end{cases}$$

## Surfaces paramétriques : un ruban de Möbius

Une équation paramétrique du ruban de Möbius est :

$$\begin{cases} x(u,v) = \left(a + u \cos\left(\frac{v}{2}\right)\right) \cos(v) \\ y(u,v) = \left(a + u \cos\left(\frac{v}{2}\right)\right) \sin(v) \\ z(u,v) = u \sin\left(\frac{v}{2}\right) \end{cases}$$

### Exemple (Fonction RubanMobius)

```
RubanMobius = function (u, v, resultCalcul) {
  u = u0 + u * (u1 - u0) //intervalle en U
  v = v0 + v * (v1 - v0) //intervalle en V
  let result = resultCalcul || new THREE.Vector3();
  let x = (a+u*Math.cos(v/2))*Math.cos(v);
  let y = (a+u*Math.cos(v/2))*Math.sin(v);
  let z = u*Math.sin(v/2);
  return result.set(x, y, z);
}//fin fonction RubanMobius
```

Ruban de Möbius

## Surfaces paramétriques : une bouteille de Klein, $a$ , $b$ et $c$ constantes

Une équation paramétrique d'une bouteille de Klein est donnée par les équations :

$$\begin{cases} x(u, v) &= (a + a \sin(u) + r(u) \cos(v)) \cos(u) \\ y(u, v) &= (b + r(u) \cos(v)) \sin(u) \\ z(u, v) &= r(u) \sin(v) \end{cases}$$

pour  $(u, v) \in [0, \pi] \times [0, 2\pi]$  et :

$$\begin{cases} x(u, v) &= (a + a \sin(u)) \cos(u) - r(u) \cos(v) \\ y(u, v) &= b \sin(u) \\ z(u, v) &= r(u) \sin(v) \end{cases}$$

pour  $(u, v) \in [\pi, 2\pi] \times [0, 2\pi]$  où  $r(u) = c \left(1 - \frac{\cos(u)}{2}\right)$ .

## Surfaces paramétriques : une bouteille de Klein, $a$ , $b$ et $c$ constantes

Une équation paramétrique d'une bouteille de Klein est donnée par les équations :

$$\begin{cases} x(u, v) = (a + a \sin(u) + r(u) \cos(v)) \cos(u) \\ y(u, v) = (b + r(u) \cos(v)) \sin(u) \\ z(u, v) = r(u) \sin(v) \end{cases}$$

pour  $(u, v) \in [0, \pi] \times [0, 2\pi]$  et :

$$\begin{cases} x(u, v) = (a + a \sin(u)) \cos(u) - r(u) \cos(v) \\ y(u, v) = b \sin(u) \\ z(u, v) = r(u) \sin(v) \end{cases}$$

pour  $(u, v) \in [\pi, 2\pi] \times [0, 2\pi]$  où  $r(u) = c \left(1 - \frac{\cos(u)}{2}\right)$ .

### Exercice (Fonction Klein)

*A faire pendant vos longues soirées d'hiver*

Bouteille de Klein

- 1 HTML et Three.js
- 2 Points et Vecteurs
  - Définitions
  - Méthodes communes à THREE.Vector $n$ ,  $n \in \{2;3\}$
  - Cas du plan i.e.  $n = 2$
  - Cas de l'espace i.e.  $n = 3$
- 3 La caméra
- 4 Les lumières
  - Lumière ambiante
  - Définition et propriétés communes des autres lumières
  - Lumière ponctuelle et spot
  - Spot et lumière directionnelle : ombre
  - Le spot
- 5 Les matériaux
- 6 Les courbes
  - Cas classique
  - Cas spécifique de THREE.js
  - Courbes de Bézier polynomiales
- 7 Les surfaces primitives
  - Le principe
  - Cas particulier des surfaces planes
  - Les surfaces non planes
- 8 Transformations géométriques de  $\mathcal{E}_3$ 
  - Application affine : translation
  - Application affine ou vectorielle
- 9 L'Animation
- 10 Création de menu G.U.I.
  - Théorie
  - Exemple avec un plan
- 11 Le C.S.G.

## 8 Transformations géométriques de $\mathcal{E}_3$

- Application affine : translation
- Application affine ou vectorielle

## Translation de la surface SurfPhong

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

## Translation de la surface SurfPhong

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

Définition (Translation selon les axes)

- *De  $x_0$  sur l'axe des abscisses :* `SurfPhong.translateX( x0 );`

## Translation de la surface SurfPhong

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

## Définition (Translation selon les axes)

- *De  $x_0$  sur l'axe des abscisses* : `SurfPhong.translateX( x0 );`
- *De  $y_0$  sur l'axe des ordonnées* : `SurfPhong.translateY( y0 );`

## Translation de la surface SurfPhong

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

## Définition (Translation selon les axes)

- *De  $x_0$  sur l'axe des abscisses* : `SurfPhong.translateX ( x0 );`
- *De  $y_0$  sur l'axe des ordonnées* : `SurfPhong.translateY ( y0 );`
- *De  $z_0$  sur l'axe des cotes* : `SurfPhong.translateZ ( z0 );`

## Translation de la surface SurfPhong

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

## Définition (Translation selon les axes)

- *De  $x_0$  sur l'axe des abscisses* : `SurfPhong.translateX ( x0 );`
- *De  $y_0$  sur l'axe des ordonnées* : `SurfPhong.translateY ( y0 );`
- *De  $z_0$  sur l'axe des cotes* : `SurfPhong.translateZ ( z0 );`

Définition (Translation selon le vecteur  $\vec{u} (x_0; y_0; z_0)$ )1) *Définir le vecteur de l'axe :*

```
let vecU = new THREE.Vector3 ( x0 , y0 , z0 );
```

## Translation de la surface SurfPhong

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

## Définition (Translation selon les axes)

- De  $x_0$  sur l'axe des abscisses : `SurfPhong.translateX ( x0 );`
- De  $y_0$  sur l'axe des ordonnées : `SurfPhong.translateY ( y0 );`
- De  $z_0$  sur l'axe des cotes : `SurfPhong.translateZ ( z0 );`

Définition (Translation selon le vecteur  $\vec{u} (x_0; y_0; z_0)$ )

- 1) Définir le vecteur de l'axe :

```
let vecU = new THREE.Vector3 ( x0 , y0 , z0 );
```

- 2) Normaliser ce vecteur : `vecU.normalize ();`

## Translation de la surface SurfPhong

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

## Définition (Translation selon les axes)

- De  $x_0$  sur l'axe des abscisses : `SurfPhong.translateX ( x0 );`
- De  $y_0$  sur l'axe des ordonnées : `SurfPhong.translateY ( y0 );`
- De  $z_0$  sur l'axe des cotes : `SurfPhong.translateZ ( z0 );`

Définition (Translation selon le vecteur  $\vec{u}(x_0; y_0; z_0)$ )

- 1) Définir le vecteur de l'axe :

```
let vecU = new THREE.Vector3 ( x0 , y0 , z0 );
```

- 2) Normaliser ce vecteur : `vecU.normalize ();`

- 3) Calculer le carré de la norme du vecteur de translation :

```
let d2 = Math.pow ( x0 , 2 )+Math.pow ( y0 , 2 )+Math.pow ( z0 , 2 );
```

## Translation de la surface SurfPhong

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

## Définition (Translation selon les axes)

- De  $x_0$  sur l'axe des abscisses : `SurfPhong.translateX ( x0 );`
- De  $y_0$  sur l'axe des ordonnées : `SurfPhong.translateY ( y0 );`
- De  $z_0$  sur l'axe des cotés : `SurfPhong.translateZ ( z0 );`

Définition (Translation selon le vecteur  $\vec{u}(x_0; y_0; z_0)$ )

- 1) Définir le vecteur de l'axe :

```
let vecU = new THREE.Vector3 ( x0 , y0 , z0 );
```

- 2) Normaliser ce vecteur : `vecU.normalize ();`

- 3) Calculer le carré de la norme du vecteur de translation :

```
let d2 = Math.pow ( x0 , 2 )+Math.pow ( y0 , 2 )+Math.pow ( z0 , 2 );
```

- 4) Appliquer la translation :

```
SurfPhong.translateOnAxis(vecU,Math.sqrt(d2));
```

Translation d'un tore

## 8 Transformations géométriques de $\mathcal{E}_3$

- Application affine : translation
- Application affine ou vectorielle

## Rotation de la surface SurfPhong

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

## Rotation de la surface SurfPhong

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

Définition (Rotation de  $\theta$  selon l'axe)

- *Des abscisses : SurfPhong.rotateX (  $\theta$  );*

## Rotation de la surface SurfPhong

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

Définition (Rotation de  $\theta$  selon l'axe)

- Des abscisses : `SurfPhong.rotateX ( θ );`
- Des ordonnées : `SurfPhong.rotateY ( θ );`

## Rotation de la surface SurfPhong

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

Définition (Rotation de  $\theta$  selon l'axe)

- *Des abscisses* : `SurfPhong.rotateX ( θ );`
- *Des ordonnées* : `SurfPhong.rotateY ( θ );`
- *Des cotes* : `SurfPhong.rotateZ ( θ );`

## Rotation de la surface SurfPhong

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

Définition (Rotation de  $\theta$  selon l'axe)

- Des abscisses : `SurfPhong.rotateX ( θ );`
- Des ordonnées : `SurfPhong.rotateY ( θ );`
- Des cotes : `SurfPhong.rotateZ ( θ );`

Définition (Rotation d'angle  $\theta$  d'axe défini par l'origine et le vecteur  $\vec{u}(x_0; y_0; z_0)$ )

- 1) Définir l'angle theta en radian ;

## Rotation de la surface SurfPhong

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

Définition (Rotation de  $\theta$  selon l'axe)

- Des abscisses : `SurfPhong.rotateX ( θ );`
- Des ordonnées : `SurfPhong.rotateY ( θ );`
- Des cotes : `SurfPhong.rotateZ ( θ );`

Définition (Rotation d'angle  $\theta$  d'axe défini par l'origine et le vecteur  $\vec{u}(x_0; y_0; z_0)$ )

1) Définir l'angle theta en radian ;

2) Définir le vecteur de l'axe :

```
let vecU = new THREE.Vector3 ( x0 , y0 , z0 );
```

3) Normaliser ce vecteur : `vecU.normalize ();`

## Rotation de la surface SurfPhong

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

Définition (Rotation de  $\theta$  selon l'axe)

- Des abscisses : `SurfPhong.rotateX ( theta );`
- Des ordonnées : `SurfPhong.rotateY ( theta );`
- Des cotes : `SurfPhong.rotateZ ( theta );`

Définition (Rotation d'angle  $\theta$  d'axe défini par l'origine et le vecteur  $\vec{u}(x_0; y_0; z_0)$ )

1) Définir l'angle `theta` en radian ;

2) Définir le vecteur de l'axe :

```
let vecU = new THREE.Vector3 ( x0 , y0 , z0 );
```

3) Normaliser ce vecteur : `vecU.normalize ();`

4) Appliquer la rotation : `SurfPhong.rotateOnAxis(vecU, theta);`

Rotation d'un tore (axe du repère)

$$\theta_{\text{rad}} = \theta_{\text{degre}} \times \frac{\pi}{180}$$

Rotation d'un tore (axe quelconque)

Mise à l'échelle de coefficient ( $k_x; k_y; k_z$ )

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

Mise à l'échelle de coefficient ( $k_x; k_y; k_z$ )

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

## Définition (Mise à l'échelle)

- *Coefficient pour les abscisses :  $k_x$*

Mise à l'échelle de coefficient ( $k_x; k_y; k_z$ )

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

## Définition (Mise à l'échelle)

- *Coefficient pour les abscisses :*  $k_x$
- *Coefficient pour les ordonnées :*  $k_y$

Mise à l'échelle de coefficient ( $k_x; k_y; k_z$ )

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

## Définition (Mise à l'échelle)

- *Coefficient pour les abscisses :*  $k_x$
- *Coefficient pour les ordonnées :*  $k_y$
- *Coefficient pour les cotes :*  $k_z$

Mise à l'échelle de coefficient ( $k_x; k_y; k_z$ )

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

## Définition (Mise à l'échelle)

- *Coefficient pour les abscisses :*  $k_x$
- *Coefficient pour les ordonnées :*  $k_y$
- *Coefficient pour les cotes :*  $k_z$
- *Application de la transformation :*

```
SurfPhong.scale.set( kx , ky , kz );
```

Mise à l'échelle de coefficient ( $k_x; k_y; k_z$ )

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

## Définition (Mise à l'échelle)

- *Coefficient pour les abscisses :*  $k_x$
- *Coefficient pour les ordonnées :*  $k_y$
- *Coefficient pour les cotes :*  $k_z$
- *Application de la transformation :*  
`SurfPhong.scale.set(  $k_x$  ,  $k_y$  ,  $k_z$  );`

## Remarque (Ordre des transformations avec THREE.js)

- 1) *Mise à l'échelle ;*

Mise à l'échelle de coefficient ( $k_x; k_y; k_z$ )

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

## Définition (Mise à l'échelle)

- *Coefficient pour les abscisses :*  $k_x$
- *Coefficient pour les ordonnées :*  $k_y$
- *Coefficient pour les cotes :*  $k_z$
- *Application de la transformation :*

```
SurfPhong.scale.set( k_x , k_y , k_z );
```

## Remarque (Ordre des transformations avec THREE.js)

- 1) *Mise à l'échelle ;*
- 2) *Rotation(s) d'axes  $(O_3; \vec{i})$  puis  $(O_3; \vec{j})$  puis  $(O_3; \vec{k})$  ;*

Mise à l'échelle de coefficient ( $k_x; k_y; k_z$ )

```
let SurfPhong = new THREE.Mesh( SurfGeom , MaterialPhong );
```

## Définition (Mise à l'échelle)

- *Coefficient pour les abscisses :  $k_x$*
- *Coefficient pour les ordonnées :  $k_y$*
- *Coefficient pour les cotes :  $k_z$*
- *Application de la transformation :*  
`SurfPhong.scale.set(  $k_x$  ,  $k_y$  ,  $k_z$  );`

## Remarque (Ordre des transformations avec THREE.js)

- 1) *Mise à l'échelle ;*
- 2) *Rotation(s) d'axes ( $O_3; \vec{i}$ ) puis ( $O_3; \vec{j}$ ) puis ( $O_3; \vec{k}$ ) ;*
- 3) *Translation.*

- 1 HTML et Three.js
- 2 Points et Vecteurs
  - Définitions
  - Méthodes communes à THREE.Vector $n$ ,  $n \in \{2;3\}$
  - Cas du plan i.e.  $n = 2$
  - Cas de l'espace i.e.  $n = 3$
- 3 La caméra
- 4 Les lumières
  - Lumière ambiante
  - Définition et propriétés communes des autres lumières
  - Lumière ponctuelle et spot
  - Spot et lumière directionnelle : ombre
  - Le spot
- 5 Les matériaux
- 6 Les courbes
  - Cas classique
  - Cas spécifique de THREE.js
  - Courbes de Bézier polynomiales
- 7 Les surfaces primitives
  - Le principe
  - Cas particulier des surfaces planes
  - Les surfaces non planes
- 8 Transformations géométriques de  $\mathcal{E}_3$ 
  - Application affine : translation
  - Application affine ou vectorielle
- 9 L'Animation
- 10 Création de menu G.U.I.
  - Théorie
  - Exemple avec un plan
- 11 Le C.S.G.

## Principe d'une animation

### Principe (Modification des propriétés dans la fonction `reAffichage` () )

- ➊ *Définir des variables globales si nécessaire ;*

## Principe d'une animation

### Principe (Modification des propriétés dans la fonction `reAffichage` () )

- ① *Définir des variables globales si nécessaire ;*
- ② *Supprimer les objets idoines ;*

## Principe d'une animation

### Principe (Modification des propriétés dans la fonction `reAffichage` () )

- ① Définir des variables globales si nécessaire ;
- ② Supprimer les objets inutiles ;
- ③ Modifier les propriétés souhaitées ;

## Principe d'une animation

### Principe (Modification des propriétés dans la fonction `reAffichage` () )

- ① Définir des variables globales si nécessaire ;
- ② Supprimer les objets inutiles ;
- ③ Modifier les propriétés souhaitées ;
- ④ Ajouter les objets précédemment supprimés ;

## Principe d'une animation

### Principe (Modification des propriétés dans la fonction `reAffichage` () )

- ① Définir des variables globales si nécessaire ;
- ② Supprimer les objets inutiles ;
- ③ Modifier les propriétés souhaitées ;
- ④ Ajouter les objets précédemment supprimés ;
- ⑤ Rappeler la fonction `reAffichage` () .

Animation d'une sphère

Animation d'une sphère dans la fonction `init()`

- 1) Définition des variables globales :

```
var theta = 0; // debut du rebond  
var pasTheta = 0.25; //pas d'incrementation du rebond  
var h = 0.5; //hauteur du rebond
```

Animation d'une sphère dans la fonction `init()`

## 1) Définition des variables globales :

```
var theta = 0; // debut du rebond  
var pasTheta = 0.25; //pas d'incrementation du rebond  
var h = 0.5; //hauteur du rebond  
var r = 1.5; // rayon du cercle de la trajectoire dans z=0  
var phi = 0; //rotation sur le cercle  
var pasPhiRot = 0.05; // pas d'incrementation sur le cercle
```

Animation d'une sphère dans la fonction `init()`

- 1) Définition des variables globales :

```
var theta = 0; // debut du rebond  
var pasTheta = 0.25; //pas d'incrementation du rebond  
var h = 0.5; //hauteur du rebond  
var r = 1.5; // rayon du cercle de la trajectoire dans z=0  
var phi = 0; //rotation sur le cercle  
var pasPhiRot = 0.05; // pas d'incrementation sur le cercle
```

- 2) Définition de la sphère dans la fonction `init()` :

```
let spherePhong = new THREE.Mesh(sphereGeom,MaterialPhong);
```

Animation d'une sphère dans la fonction `init()`

- 1) Définition des variables globales :

```
var theta = 0; // debut du rebond  
var pasTheta = 0.25; //pas d'incrementation du rebond  
var h = 0.5; //hauteur du rebond  
var r = 1.5; // rayon du cercle de la trajectoire dans z=0  
var phi = 0; //rotation sur le cercle  
var pasPhiRot = 0.05; // pas d'incrementation sur le cercle
```

- 2) Définition de la sphère dans la fonction `init()` :

```
let spherePhong = new THREE.Mesh(sphereGeom,MaterialPhong);
```

- 3) Modification de la fonction `reAffichage()` dans la fonction `init()`.

Animation d'une sphère dans la fonction `init()`

- 1) Définition des variables globales :

```
var theta = 0; // debut du rebond
var pasTheta = 0.25; //pas d'incrementation du rebond
var h = 0.5; //hauteur du rebond
var r = 1.5; // rayon du cercle de la trajectoire dans z=0
var phi = 0; //rotation sur le cercle
var pasPhiRot = 0.05; // pas d'incrementation sur le cercle
```

- 2) Définition de la sphère dans la fonction `init()` :

```
let spherePhong = new THREE.Mesh(sphereGeom,MaterialPhong);
```

- 3) Modification de la fonction `reAffichage()` dans la fonction `init()`.

**Trajectoire, de hauteur  $2h$ , de la sphère de rayon  $R$  :**

$$\begin{cases} x(\phi; \theta) &= r \cos(\phi) \\ y(\phi; \theta) &= r \sin(\phi) \\ z(\phi; \theta) &= R + h(1 - \cos(\theta)) \end{cases}$$

Animation d'une sphère

Animation d'une sphère, fonction `reAffichage()`

- 1) Fonction `setTimeout()` dans la fonction `reAffichage()` :

```
reAffichage(){  
    setTimeout(function () {
```

Animation d'une sphère

Animation d'une sphère, fonction `reAffichage()`

- 1) Fonction `setTimeout()` dans la fonction `reAffichage()` :

```
reAffichage(){  
    setTimeout(function () {
```

- 2) Suppression de la surface `spherePhong` de la scène :

```
if (spherePhong) scene.remove(spherePhong);
```

Animation d'une sphère, fonction `reAffichage()`

- 1) Fonction `setTimeout()` dans la fonction `reAffichage()` :

```
reAffichage(){  
    setTimeout(function () {
```

- 2) Suppression de la surface `spherePhong` de la scène :

```
    if (spherePhong) scene.remove(spherePhong);
```

- 3) Actualisation de la valeur de  $\theta$  : `theta += pasTheta;`

Animation d'une sphère, fonction `reAffichage()`

- 1) Fonction `setTimeout()` dans la fonction `reAffichage()` :

```
reAffichage(){  
    setTimeout(function () {
```

- 2) Suppression de la surface `spherePhong` de la scène :

```
    if (spherePhong) scene.remove(spherePhong);
```

- 3) Actualisation de la valeur de  $\theta$  : `theta += pasTheta;`

- 4) Actualisation de la valeur de  $\phi$  : `phi += pasPhiRot;`

Animation d'une sphère, fonction `reAffichage()`

- 1) Fonction `setTimeout()` dans la fonction `reAffichage()` :

```
reAffichage(){  
    setTimeout(function () {
```

- 2) Suppression de la surface `spherePhong` de la scène :

```
    if (spherePhong) scene.remove(spherePhong);
```

- 3) Actualisation de la valeur de  $\theta$  : `theta += pasTheta;`

- 4) Actualisation de la valeur de  $\phi$  : `phi += pasPhiRot;`

- 5) Actualisation de la position de la sphère :

```
spherePhong.position.set(r*Math.cos(phi),r*Math.sin(phi),  
    sphereGeom.parameters.radius +h *(1-Math.cos(theta)));
```

Animation d'une sphère

Animation d'une sphère, fonction `reAffichage()`

- 1) Fonction `setTimeout()` dans la fonction `reAffichage()` :

```
reAffichage(){  
    setTimeout(function () {
```

- 2) Suppression de la surface `spherePhong` de la scène :

```
        if (spherePhong) scene.remove(spherePhong);
```

- 3) Actualisation de la valeur de  $\theta$  : `theta += pasTheta;`

- 4) Actualisation de la valeur de  $\phi$  : `phi += pasPhiRot;`

- 5) Actualisation de la position de la sphère :

```
        spherePhong.position.set(r*Math.cos(phi),r*Math.sin(phi),  
        sphereGeom.parameters.radius +h *(1-Math.cos(theta)));
```

- 6) Ajout de la sphère dans la scène : `scene.add(spherePhong);`

Animation d'une sphère, fonction `reAffichage()`

- 1) Fonction `setTimeout()` dans la fonction `reAffichage()` :

```
reAffichage(){  
    setTimeout(function () {
```

- 2) Suppression de la surface `spherePhong` de la scène :

```
        if (spherePhong) scene.remove(spherePhong);
```

- 3) Actualisation de la valeur de  $\theta$  : `theta += pasTheta;`

- 4) Actualisation de la valeur de  $\phi$  : `phi += pasPhiRot;`

- 5) Actualisation de la position de la sphère :

```
        spherePhong.position.set(r*Math.cos(phi),r*Math.sin(phi),  
        sphereGeom.parameters.radius +h *(1-Math.cos(theta)));
```

- 6) Ajout de la sphère dans la scène : `scene.add(spherePhong);`

- 7) Rappel de la fonction pour l'animation : `reAffichage();`

Animation d'une sphère, fonction `reAffichage()`

- 1) Fonction `setTimeout()` dans la fonction `reAffichage()` :

```
reAffichage(){
    setTimeout(function () {
```

- 2) Suppression de la surface `spherePhong` de la scène :

```
if (spherePhong) scene.remove(spherePhong);
```

- 3) Actualisation de la valeur de  $\theta$  : `theta += pasTheta;`

- 4) Actualisation de la valeur de  $\phi$  : `phi += pasPhiRot;`

- 5) Actualisation de la position de la sphère :

```
spherePhong.position.set(r*Math.cos(phi),r*Math.sin(phi),
    sphereGeom.parameters.radius +h *(1-Math.cos(theta)));
```

- 6) Ajout de la sphère dans la scène : `scene.add(spherePhong);`

- 7) Rappel de la fonction pour l'animation : `reAffichage();`

- 8) Fermeture de la fonction `setTimeout()` :

```
}); // fin setTimeout(function ()
```

Animation d'une sphère, fonction `reAffichage()`

- 1) Fonction `setTimeout()` dans la fonction `reAffichage()` :

```
reAffichage(){
    setTimeout(function () {
```

- 2) Suppression de la surface `spherePhong` de la scène :

```
if (spherePhong) scene.remove(spherePhong);
```

- 3) Actualisation de la valeur de  $\theta$  : `theta += pasTheta;`

- 4) Actualisation de la valeur de  $\phi$  : `phi += pasPhiRot;`

- 5) Actualisation de la position de la sphère :

```
spherePhong.position.set(r*Math.cos(phi),r*Math.sin(phi),
    sphereGeom.parameters.radius +h *(1-Math.cos(theta)));
```

- 6) Ajout de la sphère dans la scène : `scene.add(spherePhong);`

- 7) Rappel de la fonction pour l'animation : `reAffichage();`

- 8) Fermeture de la fonction `setTimeout()` :

```
}, 200); // fin setTimeout(function ()
```

- 9) Actualisation du rendu :

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

Animation d'une sphère, fonction `reAffichage()`

- 1) Fonction `setTimeout()` dans la fonction `reAffichage()` :

```
reAffichage(){
    setTimeout(function () {
```

- 2) Suppression de la surface `spherePhong` de la scène :

```
if (spherePhong) scene.remove(spherePhong);
```

- 3) Actualisation de la valeur de  $\theta$  : `theta += pasTheta;`

- 4) Actualisation de la valeur de  $\phi$  : `phi += pasPhiRot;`

- 5) Actualisation de la position de la sphère :

```
spherePhong.position.set(r*Math.cos(phi),r*Math.sin(phi),
    sphereGeom.parameters.radius +h *(1-Math.cos(theta)));
```

- 6) Ajout de la sphère dans la scène : `scene.add(spherePhong);`

- 7) Rappel de la fonction pour l'animation : `reAffichage();`

- 8) Fermeture de la fonction `setTimeout()` :

```
}, 200);// fin setTimeout(function ()
```

- 9) Actualisation du rendu :

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

- 10) // fin fonction `reAffichage()`

- 11) // fin fonction `init()`

Animation d'une sphère

Animation d'une sphère, fonction `init()`

```
function init(){
```

```
    ...
```

Animation d'une sphère, fonction `init()`

```
function init(){  
    ...  
    let spherePhong = new THREE.Mesh(sphereGeom,MaterialPhong);  
    scene.add(spherePhong);  
    ...  
    function reAffichage() {  
        setTimeout(function () {  
            if (spherePhong) scene.remove(spherePhong);  
            theta += pasTheta;  
            phi += pasPhiRot;  
        }, 100);  
    }  
    reAffichage();  
    requestAnimationFrame(reAffichage);  
}
```

Animation d'une sphère, fonction `init()`

```
function init(){  
    ...  
    let spherePhong = new THREE.Mesh(sphereGeom,MaterialPhong);  
    scene.add(spherePhong);  
    ...  
    function reAffichage() {  
        setTimeout(function () {  
            if (spherePhong) scene.remove(spherePhong);  
            theta += pasTheta;  
            phi += pasPhiRot;  
            spherePhong.position.set(r*Math.cos(phi),r*Math.sin(phi),  
                sphereGeom.parameters.radius+h*(1-Math.cos(theta)));  
        }, 100);  
    }  
    reAffichage();  
    requestAnimationFrame(reAffichage);  
}
```

Animation d'une sphère, fonction `init()`

```
function init(){  
    ...  
    let spherePhong = new THREE.Mesh(sphereGeom,MaterialPhong);  
    scene.add(spherePhong);  
    ...  
    function reAffichage() {  
        setTimeout(function () {  
            if (spherePhong) scene.remove(spherePhong);  
            theta += pasTheta;  
            phi += pasPhiRot;  
            spherePhong.position.set(r*Math.cos(phi),r*Math.sin(phi),  
                sphereGeom.parameters.radius+h*(1-Math.cos(theta)));  
            scene.add(spherePhong);  
            reAffichage();  
        }, 200);// fin setTimeout(function ()
```

Animation d'une sphère, fonction `init()`

```
function init(){  
    ...  
    let spherePhong = new THREE.Mesh(sphereGeom,MaterialPhong);  
    scene.add(spherePhong);  
    ...  
    function reAffichage() {  
        setTimeout(function () {  
            if (spherePhong) scene.remove(spherePhong);  
            theta += pasTheta;  
            phi += pasPhiRot;  
            spherePhong.position.set(r*Math.cos(phi),r*Math.sin(phi),  
                sphereGeom.parameters.radius+h*(1-Math.cos(theta)));  
            scene.add(spherePhong);  
            reAffichage();  
        }, 200); // fin setTimeout(function ()  
        rendu.render(scene, camera);  
    } // fin fonction reAffichage()  
} // fin fonction init()
```

- 1 HTML et Three.js
- 2 Points et Vecteurs
  - Définitions
  - Méthodes communes à THREE.Vector $n$ ,  $n \in \{2;3\}$
  - Cas du plan i.e.  $n = 2$
  - Cas de l'espace i.e.  $n = 3$
- 3 La caméra
- 4 Les lumières
  - Lumière ambiante
  - Définition et propriétés communes des autres lumières
  - Lumière ponctuelle et spot
  - Spot et lumière directionnelle : ombre
  - Le spot
- 5 Les matériaux
- 6 Les courbes
  - Cas classique
  - Cas spécifique de THREE.js
  - Courbes de Bézier polynomiales
- 7 Les surfaces primitives
  - Le principe
  - Cas particulier des surfaces planes
  - Les surfaces non planes
- 8 Transformations géométriques de  $\mathcal{E}_3$ 
  - Application affine : translation
  - Application affine ou vectorielle
- 9 L'Animation
- 10 Création de menu G.U.I.
  - Théorie
  - Exemple avec un plan
- 11 Le C.S.G.

## 10 Création de menu G.U.I.

- Théorie
- Exemple avec un plan

Création d'un menu GUI (dans la fonction `init()`)

## Principe (GUI : Interface Graphique Utilisateur)

1) *Définition d'une variable pour le menu : let gui = new dat.GUI();*

Création d'un menu GUI (dans la fonction `init()`)

## Principe (GUI : Interface Graphique Utilisateur)

- 1) *Définition d'une variable pour le menu : let gui = new dat.GUI();*
- 2) *Définition de la variable menuGUI gérant le menu (cf. diapo suivante);*

Création d'un menu GUI (dans la fonction `init()`)

## Principe (GUI : Interface Graphique Utilisateur)

- 1) *Définition d'une variable pour le menu : let gui = new dat.GUI();*
- 2) *Définition de la variable menuGUI gérant le menu (cf. diapo suivante) ;*
- 3) *Créer un répertoire dans la fonction init() :*

```
let guiReperatoire = gui.addFolder("Nom de l'objet");
```

Création d'un menu GUI (dans la fonction `init()`)

## Principe (GUI : Interface Graphique Utilisateur)

- 1) *Définition d'une variable pour le menu : let gui = new dat.GUI();*
- 2) *Définition de la variable menuGUI gérant le menu (cf. diapo suivante) ;*
- 3) *Créer un répertoire dans la fonction init() :*  
`let guiRepertoire = gui.addFolder("Nom de l'objet");`
- 4) *Créer des propriétés dans guiRepertoire :*

Création d'un menu GUI (dans la fonction `init()`)

## Principe (GUI : Interface Graphique Utilisateur)

1) Définition d'une variable pour le menu : `let gui = new dat.GUI();`

2) Définition de la variable `menuGUI` gérant le menu (cf. diapo suivante);

3) Créer un répertoire dans la fonction `init()` :

```
let guiRepertoire = gui.addFolder("Nom de l'objet");
```

4) Créer des propriétés dans `guiRepertoire` :

a) Valeurs dans un intervalle compris entre `b0` et `b1` :

```
let guiRepertoire.add(menuGUI, 'toto', b0, b1).onChange(...);
```

Création d'un menu GUI (dans la fonction `init()`)

## Principe (GUI : Interface Graphique Utilisateur)

- 1) Définition d'une variable pour le menu : `let gui = new dat.GUI();`
- 2) Définition de la variable `menuGUI` gérant le menu (cf. diapo suivante);
- 3) Créer un répertoire dans la fonction `init()` :  
`let guiRepertoire = gui.addFolder("Nom de l'objet");`
- 4) Créer des propriétés dans `guiRepertoire` :

- a) Valeurs dans un intervalle compris entre `b0` et `b1` :

```
let guiRepertoire.add(menuGUI, 'toto', b0, b1).onChange(...);
```

- b) Valeurs dans un menu déroulant :

```
let guiRepertoire.add(menuGUI, 'totoD', ['Oui', 'Non']).onChange(...);
```

Création d'un menu GUI (dans la fonction `init()`)

## Principe (GUI : Interface Graphique Utilisateur)

- 1) Définition d'une variable pour le menu : `let gui = new dat.GUI();`
- 2) Définition de la variable `menuGUI` gérant le menu (cf. diapo suivante);
- 3) Créer un répertoire dans la fonction `init()` :  
`let guiRepertoire = gui.addFolder("Nom de l'objet");`
- 4) Créer des propriétés dans `guiRepertoire` :

- a) Valeurs dans un intervalle compris entre `b0` et `b1` :

```
let guiRepertoire.add(menuGUI, 'toto', b0, b1).onChange(...);
```

- b) Valeurs dans un menu déroulant :

```
let guiRepertoire.add(menuGUI, 'totoD', ['Oui', 'Non']).onChange(...);
```

- c) Modification de couleurs :

```
let guiRepertoire.addColor(menuGUI, 'toto').onChange(...);
```

Création d'un menu GUI (dans la fonction `init()`)

## Principe (GUI : Interface Graphique Utilisateur)

- 1) Définition d'une variable pour le menu : `let gui = new dat.GUI();`
- 2) Définition de la variable `menuGUI` gérant le menu (cf. diapo suivante);
- 3) Créer un répertoire dans la fonction `init()` :  
`let guiRepertoire = gui.addFolder("Nom de l'objet");`
- 4) Créer des propriétés dans `guiRepertoire` :

- a) Valeurs dans un intervalle compris entre `b0` et `b1` :

```
let guiRepertoire.add(menuGUI, 'toto', b0, b1).onChange(...);
```

- b) Valeurs dans un menu déroulant :

```
let guiRepertoire.add(menuGUI, 'totoD', ['Oui', 'Non']).onChange(...);
```

- c) Modification de couleurs :

```
let guiRepertoire.addColor(menuGUI, 'toto').onChange(...);
```

- d) Case à cocher :

```
gui.add(menuGUI, 'toto').onChange(function (e) {
  if (!e) ...
  else ...
}); //fin cochage
```

Création d'un menu GUI (dans la fonction `init()`)

## Principe (GUI : Interface Graphique Utilisateur)

- 1) Définition d'une variable pour le menu : `let gui = new dat.GUI();`
- 2) Définition de la variable `menuGUI` gérant le menu (cf. diapo suivante);
- 3) Créer un répertoire dans la fonction `init()` :  
`let guiRepertoire = gui.addFolder("Nom de l'objet");`
- 4) Créer des propriétés dans `guiRepertoire` :

- a) Valeurs dans un intervalle compris entre `b0` et `b1` :

```
let guiRepertoire.add(menuGUI, 'toto', b0, b1).onChange(...);
```

- b) Valeurs dans un menu déroulant :

```
let guiRepertoire.add(menuGUI, 'totoD', ['Oui', 'Non']).onChange(...);
```

- c) Modification de couleurs :

```
let guiRepertoire.addColor(menuGUI, 'toto').onChange(...);
```

- d) Case à cocher :

```
gui.add(menuGUI, 'toto').onChange(function (e) {
  if (!e) ...
  else ...
}); //fin cochage
```

- e) Mise à jour éventuelle de `guiRepertoire` dans un « `onChange(...)` »

```
guiRepertoire.updateDisplay();
```

Exemple de menu GUI : cas d'un plan

La variable `menuGUI` gérant le menu dans la fonction `init()`

Définition (Propriété dans `menuGUI`)

```
let menuGUI = new function () {  
    // propriete de l'objet  
    this.toto = proprietObjet; // ou une valeur
```

La variable `menuGUI` gérant le menu dans la fonction `init()`

Définition (Propriété dans `menuGUI`)

```
let menuGUI = new function () {  
    // propriete de l'objet  
    this.toto = proprietObjet; // ou une valeur  
  
    // propriete de l'objet deroulant  
    if (proprieteObjetDeroulant)  
        this.totoD = 'Oui';  
    else this.totoD = 'Non';
```

La variable `menuGUI` gérant le menu dans la fonction `init()`Définition (Propriété dans `menuGUI`)

```

let menuGUI = new function () {
    // propriete de l'objet
    this.toto = proprietObjet; // ou une valeur

    // propriete de l'objet deroulant
    if (proprieteObjetDeroulant)
        this.totoD = 'Oui';
    else this.totoD = 'Non';

    //pour actualiser dans la scene
    this.actualisation = function () {
        ...
        reAffichage();
    }; // fin this.actualisation
}; //fin de la fonction menuGUI

```

Exemple de menu GUI : cas d'un plan

Utilisation de `guiRepertoire.add(menuGUI, 'toto', b0, b1)`

Rappel (Dans `menuGUI = new function ()`)

```
this.toto = proprieteObjet; // ou une valeur
```

Utilisation de `guiRepertoire.add(menuGUI, 'toto', b0, b1)`

Rappel (Dans `menuGUI = new function ()`)

```
this.toto = proprieteeObjet; // ou une valeur
```

Définition (Valeur dans un intervalle de réels ou d'entiers)

```
let guiRepertoire.add(menuGUI, 'toto', b0, b1).onChange (
    function () { // retour chariot pour le cours
        // si on veut un entier
        menuGUI.toto = Math.floor(menuGUI.toto);
        proprieteeObjet = menuGUI.toto;
        ...
    });

```

Exemple de menu GUI : cas d'un plan

Utilisation de `guiReperatoire.add(menuGUI, 'totoDerouulant', ['Oui', 'Non'])`

Rappel (Dans `menuGUI = new function ()`)

```
if (proprieteObjetDerouulant)
    this.totoD = 'Oui';
else this.totoD = 'Non';
```

```
Utilisation de guiReperatoire.add(menuGUI, 'totoDerouulant', ['Oui', 'Non'])
```

Rappel (Dans menuGUI = new function () )

```
if (proprieteObjetDerouant)
    this.totoD = 'Oui';
else this.totoD = 'Non';
```

Définition (Cas de chaînes de caractères)

```
guiReperatoire.add(menuGUI, 'totoD', ['Oui', 'Non']).onChange(
    function (e) { // retour chariot pour le cours
        if (e=='Oui')
            proprieteObjet = true;
        proprieteObjet = false;
    });
});
```

Exemple de menu GUI : cas d'un plan

## Utilisation d'un menu déroulant à valeurs numériques

Rappel (Dans `menuGUI = new function ()`)  
`this.toto = proprieteObjet;`

## Utilisation d'un menu déroulant à valeurs numériques

Rappel (Dans `menuGUI = new function ()`)  
`this.toto = proprieteObjet;`

Définition (Cas de valeurs numériques (ici `[1 ; 2]`))

```
guiRepertoire.add(menuGUI, 'toto', [1,2]).onChange(function (e) {  
    proprieteObjet = menuGUI.toto;  
    ...  
});
```

Exemple de menu GUI : cas d'un plan

Utilisation de `guiRepertoire.addColor(menuGUI, 'toto')`

Rappel (Dans `menuGUI = new function ()`)

```
this.toto = proprieteObjet;
```

Utilisation de `guiRepertoire.addColor(menuGUI, 'toto')`

Rappel (Dans `menuGUI = new function ()`)

```
this.toto = proprieteObjet;
```

## Définition

```
guiRepertoire.addColor(menuGUI, 'toto').onChange(function (e) {  
    proprieteObjet.setStyle(e);  
    // ou  
    proprieteObjet = new THREE.Color(e);  
    ...  
});
```

Exemple de menu GUI : cas d'un plan

## 10 Création de menu G.U.I.

- Théorie
- Exemple avec un plan

La variable `menuGUI` gérant le menu dans la fonction `init()`

## Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong );
```

La variable `menuGUI` gérant le menu dans la fonction `init()`

## Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong );
```

```
let gui = new dat.GUI(); //interface graphique utilisateur
```

La variable `menuGUI` gérant le menu dans la fonction `init()`

## Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong );
```

```
let gui = new dat.GUI(); //interface graphique utilisateur
```

```
let menuGUI = new function () {
```

La variable `menuGUI` gérant le menu dans la fonction `init()`

## Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong );
```

```
let gui = new dat.GUI(); //interface graphique utilisateur
```

```
let menuGUI = new function () {
    // propriete de la camera
    this.cameraxPos = camera.position.x;
```

La variable `menuGUI` gérant le menu dans la fonction `init()`

## Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);  
  
let gui = new dat.GUI(); //interface graphique utilisateur  
  
let menuGUI = new function () {  
    // propriete de la camera  
    this.cameraXPos = camera.position.x;  
  
    // propriete geometrique du plan  
    this.planLargeur = planGeometry.parameters.width;  
    this.planNbeLargeur = planGeometry.parameters.widthSegments;
```

La variable `menuGUI` gérant le menu dans la fonction `init()`

```
let menuGUI = new function () {
    // propriete de la camera
    this.cameraxPos = camera.position.x;

    // propriete geometrique du plan
    this.planLargeur = planGeometry.parameters.width;
    this.planNbeLargeur = planGeometry.parameters.widthSegments;

    //propriete materielle du plan
    this.AffichagePhong = true; // visible
    this.FilDeFer = true;
    if (MaterialPhong.wireframe)
        this.FilDeFer = 'Oui';
    else this.FilDeFer = 'Non';
    this.CouleurPhong = MaterialPhong.color.getStyle();
    this.opacitePhong = MaterialPhong.opacity;
    this.emissivePhong = MaterialPhong.emissive.getHex();
```

La variable `menuGUI` gérant le menu dans la fonction `init()`

```
let menuGUI = new function () {  
    ...  
    //propriete materielle du plan  
    this.AffichagePhong = true; // visible  
    this.FilDeFer = true;  
    if (MaterialPhong.wireframe)  
        this.FilDeFer = 'Oui';  
        else this.FilDeFer = 'Non';  
    this.CouleurPhong = MaterialPhong.color.getStyle();  
    this.opacitePhong = MaterialPhong.opacity;  
    this.emissivePhong = MaterialPhong.emissive.getHex();  
  
    //pour actualiser dans la scene  
    this.actualisation = function () {  
        // camera a ajouter dans la scene  
        cameraLumiere(scene,camera);  
        reAffichage();  
    }; // fin this.actualisation  
}; //fin de la fonction menuGUI
```

## Ajout des propriétés de la camera dans la fonction init()

```
// abscisse de la position de la camera dans le menu
let guiCamera = gui.addFolder("Camera");
guiCamera.add(menuGUI, "cameraxPos", -10, 10).onChange(function () {
    camera.position.set(menuGUI.cameraxPos, camera.position.y,
        camera.position.z );
});
```

Exemple de menu GUI : cas d'un plan

## Propriétés géométriques du plan

### Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);
```

## Propriétés géométriques du plan

### Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);
```

```
let guiPlanGeometrie = gui.addFolder("Plan : Prop. géom.");
```

## Propriétés géométriques du plan

### Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);
```

```
let guiPlanGeometrie = gui.addFolder("Plan : Prop. géom.");
```

## Propriétés géométriques du plan

### Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);  
  
let guiPlanGeometrie = gui.addFolder("Plan : Prop. géom.");  
  
guiPlanGeometrie.add(menuGUI,'planLargeur',1,40).onChange(  
    function () { // retour chariot pour le cours  
        console.log("Largeur : " + planLargeur);  
        plan几何学.setWidth(planLargeur);  
    }  
)
```

## Propriétés géométriques du plan

### Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);  
  
let guiPlanGeometrie = gui.addFolder("Plan : Prop. géom.");  
  
guiPlanGeometrie.add(menuGUI,'planLargeur',1,40).onChange(  
    function () { // retour chariot pour le cours  
        // on veut un entier  
        menuGUI.planLargeur = Math.floor(menuGUI.planLargeur);  
    }  
)
```

## Propriétés géométriques du plan

### Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);  
  
let guiPlanGeometrie = gui.addFolder("Plan : Prop. géom.");  
  
guiPlanGeometrie.add(menuGUI,'planLargeur',1,40).onChange(  
    function () { // retour chariot pour le cours  
        // on veut un entier  
        menuGUI.planLargeur = Math.floor(menuGUI.planLargeur);  
        planGeometry.parameters.width = menuGUI.planLargeur;
```

## Propriétés géométriques du plan

### Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);  
  
let guiPlanGeometrie = gui.addFolder("Plan : Prop. géom.");  
  
guiPlanGeometrie.add(menuGUI,'planLargeur',1,40).onChange(  
    function () { // retour chariot pour le cours  
        // on veut un entier  
        menuGUI.planLargeur = Math.floor(menuGUI.planLargeur);  
        planGeometry.parameters.width = menuGUI.planLargeur;  
        if (planPhong) scene.remove(planPhong);  
    }  
)
```

## Propriétés géométriques du plan

## Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);  
  
let guiPlanGeometrie = gui.addFolder("Plan : Prop. géom.");  
  
guiPlanGeometrie.add(menuGUI,'planLargeur',1,40).onChange(  
    function () { // retour chariot pour le cours  
        // on veut un entier  
        menuGUI.planLargeur = Math.floor(menuGUI.planLargeur);  
        planGeometry.parameters.width = menuGUI.planLargeur;  
        if (planPhong) scene.remove(planPhong);  
        planGeometry = new THREE.PlanGeometry(  
            planGeometry.parameters.width,  
            planGeometry.parameters.height,  
            planGeometry.parameters.widthSegments,  
            planGeometry.parameters.heightSegments);  
    } );
```

## Propriétés géométriques du plan

## Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);  
  
let guiPlanGeometrie = gui.addFolder("Plan : Prop. géom.");  
  
guiPlanGeometrie.add(menuGUI,'planLargeur',1,40).onChange(  
    function () { // retour chariot pour le cours  
        // on veut un entier  
        menuGUI.planLargeur = Math.floor(menuGUI.planLargeur);  
        planGeometry.parameters.width = menuGUI.planLargeur;  
        if (planPhong) scene.remove(planPhong);  
        planGeometry = new THREE.PlanGeometry(  
            planGeometry.parameters.width,  
            planGeometry.parameters.height,  
            planGeometry.parameters.widthSegments,  
            planGeometry.parameters.heightSegments);  
        planPhong = new THREE.Mesh(planGeometry,MaterialPhong);  
    } );
```

## Propriétés géométriques du plan

## Rappel

```

let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);

let guiPlanGeometrie = gui.addFolder("Plan : Prop. géom.");

guiPlanGeometrie.add(menuGUI,'planLargeur',1,40).onChange(
    function () { // retour chariot pour le cours
        // on veut un entier
        menuGUI.planLargeur = Math.floor(menuGUI.planLargeur);
        planGeometry.parameters.width = menuGUI.planLargeur;
        if (planPhong) scene.remove(planPhong);
        planGeometry = new THREE.PlanGeometry(
            planGeometry.parameters.width,
            planGeometry.parameters.height,
            planGeometry.parameters.widthSegments,
            planGeometry.parameters.heightSegments);
        planPhong = new THREE.Mesh(planGeometry,MaterialPhong);
        scene.add(planPhong);
    });
}); // fin planLargeur

```

Exemple de menu GUI : cas d'un plan

## Propriétés géométriques du plan

### Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);
```

```
let guiPlanGeometrie = gui.addFolder("Plan : Prop. géom.");
```

-----

## Propriétés géométriques du plan

### Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);  
  
let guiPlanGeometrie = gui.addFolder("Plan : Prop. géom.");  
  
...  
// la valeur initiale est une de valeurs du menu  
guiPlanGeometrie.add(menuGUI,'planNbeLargeur',[1,5,10,20,30]).  
    onChange(function () { // retour chariot pour le cours
```

## Propriétés géométriques du plan

### Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);  
  
let guiPlanGeometrie = gui.addFolder("Plan : Prop. géom.");  
  
...  
  
// la valeur initiale est une de valeurs du menu  
guiPlanGeometrie.add(menuGUI,'planNbeLargeur',[1,5,10,20,30]).  
    onChange(function () { // retour chariot pour le cours  
        planGeometry.parameters.widthSegments = menuGUI.planNbeLargeur;
```

## Propriétés géométriques du plan

### Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);  
  
let guiPlanGeometrie = gui.addFolder("Plan : Prop. géom.");  
...  
// la valeur initiale est une de valeurs du menu  
guiPlanGeometrie.add(menuGUI,'planNbeLargeur',[1,5,10,20,30]).  
    onChange(function () { // retour chariot pour le cours  
        planGeometry.parameters.widthSegments = menuGUI.planNbeLargeur;  
        if (planPhong) scene.remove(planPhong);  
    })
```

## Propriétés géométriques du plan

### Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);  
  
let guiPlanGeometrie = gui.addFolder("Plan : Prop. géom.");  
  
...  
// la valeur initiale est une de valeurs du menu  
guiPlanGeometrie.add(menuGUI,'planNbeLargeur',[1,5,10,20,30]).  
  onChange(function () { // retour chariot pour le cours  
    planGeometry.parameters.widthSegments = menuGUI.planNbeLargeur;  
    if (planPhong) scene.remove(planPhong);  
    planGeometry = new THREE.PlanGeometry(  
      planGeometry.parameters.width,  
      planGeometry.parameters.height,  
      planGeometry.parameters.widthSegments,  
      planGeometry.parameters.heightSegments);
```

## Propriétés géométriques du plan

### Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);  
  
let guiPlanGeometrie = gui.addFolder("Plan : Prop. géom.");  
  
...  
// la valeur initiale est une de valeurs du menu  
guiPlanGeometrie.add(menuGUI,'planNbeLargeur',[1,5,10,20,30]).  
  onChange(function () { // retour chariot pour le cours  
    planGeometry.parameters.widthSegments = menuGUI.planNbeLargeur;  
    if (planPhong) scene.remove(planPhong);  
    planGeometry = new THREE.PlanGeometry(  
      planGeometry.parameters.width,  
      planGeometry.parameters.height,  
      planGeometry.parameters.widthSegments,  
      planGeometry.parameters.heightSegments);  
    planPhong = new THREE.Mesh(planGeometry,MaterialPhong);
```

## Propriétés géométriques du plan

## Rappel

```
let planPhong = new THREE.Mesh(planGeometry,MaterialPhong);  
  
let guiPlanGeometrie = gui.addFolder("Plan : Prop. géom.");  
  
...  
// la valeur initiale est une de valeurs du menu  
guiPlanGeometrie.add(menuGUI,'planNbeLargeur',[1,5,10,20,30]).  
  onChange(function () { // retour chariot pour le cours  
    planGeometry.parameters.widthSegments = menuGUI.planNbeLargeur;  
    if (planPhong) scene.remove(planPhong);  
    planGeometry = new THREE.PlanGeometry(  
      planGeometry.parameters.width,  
      planGeometry.parameters.height,  
      planGeometry.parameters.widthSegments,  
      planGeometry.parameters.heightSegments);  
    planPhong = new THREE.Mesh(planGeometry,MaterialPhong);  
    scene.add(planPhong);  
}); // fin planNbeLargeur
```

Exemple de menu GUI : cas d'un plan

## Propriétés lumineuses (Phong) du plan

```
//case a cocher pour afficher ou non la surface  
gui.add(menuGUI, 'AffichagePhong').onChange(function (e) {  
    if (!e) scene.remove(planPhong);  
    else scene.add(planPhong);  
}); //fin cochage Phong
```

## Propriétés lumineuses (Phong) du plan

```
//case a cocher pour afficher ou non la surface  
gui.add(menuGUI, 'AffichagePhong').onChange(function (e) {  
    if (!e) scene.remove(planPhong);  
    else scene.add(planPhong);  
}); //fin cochage Phong  
  
// ajout de planPhong dans le menu du GUI  
let guiPlanPhong = gui.addFolder("Plan : Phong");
```

## Propriétés lumineuses (Phong) du plan

```
//case a cocher pour afficher ou non la surface
gui.add(menuGUI, 'AffichagePhong').onChange(function (e) {
    if (!e) scene.remove(planPhong);
    else scene.add(planPhong);
}); //fin cochage Phong

// ajout de planPhong dans le menu du GUI
let guiPlanPhong = gui.addFolder("Plan : Phong");

// couleur de l'objet
guiPlanPhong.addColor(menuGUI, 'CouleurPhong').onChange(
    function (e) { // retour chariot pour le cours
        MaterialPhong.color.setStyle(e);
    });
}); // fin couleur de l'objet
```

## Propriétés lumineuses (Phong) du plan

```
//case a cocher pour afficher ou non la surface
gui.add(menuGUI, 'AffichagePhong').onChange(function (e) {
    if (!e) scene.remove(planPhong);
    else scene.add(planPhong);
}); //fin cochage Phong

// ajout de planPhong dans le menu du GUI
let guiPlanPhong = gui.addFolder("Plan : Phong");

// couleur de l'objet
guiPlanPhong.addColor(menuGUI, 'CouleurPhong').onChange(
    function (e) { // retour chariot pour le cours
        MaterialPhong.color.setStyle(e);
    });
}); // fin couleur de l'objet

//emissivite : couleur
guiPlanPhong.addColor(menuGUI, 'emissivePhong').onChange(
    function (e) { // retour chariot pour le cours
        MaterialPhong.emissive = new THREE.Color(e);
    });
}); // fin emissivite : couleur
```

Exemple de menu GUI : cas d'un plan

## Propriétés lumineuses (Phong) du plan

```
//emissivite : couleur
guiPlanPhong.addColor(menuGUI, 'emissivePhong').onChange(
    function (e) { // retour chariot pour le cours
        MaterialPhong.emissive= new THREE.Color(e);
}); // fin emissivite : couleur
```

## Propriétés lumineuses (Phong) du plan

```
//emissivite : couleur
guiPlanPhong.addColor(menuGUI, 'emissivePhong').onChange(
    function (e) { // retour chariot pour le cours
        MaterialPhong.emissive= new THREE.Color(e);
}); // fin emissivite : couleur

//opacite : reel entre 0 et 1
guiPlanPhong.add(menuGUI, 'opacitePhong',0,1).onChange(
    function (e) { // retour chariot pour le cours
        MaterialPhong.opacity=e;
}); // fin opacitePhong
```

## Propriétés lumineuses (Phong) du plan

```
//emissivite : couleur
guiPlanPhong.addColor(menuGUI, 'emissivePhong').onChange(
    function (e) { // retour chariot pour le cours
        MaterialPhong.emissive= new THREE.Color(e);
}); // fin emissivite : couleur

//opacite : reel entre 0 et 1
guiPlanPhong.add(menuGUI, 'opacitePhong',0,1).onChange(
    function (e) { // retour chariot pour le cours
        MaterialPhong.opacity=e;
}); // fin opacitePhong

//FilDeFer
guiPlanPhong.add(menuGUI, 'FilDeFer', ['Oui', 'Non']).onChange(
    function (e) { // retour chariot pour le cours
        if (e=='Oui')
            MaterialPhong.wireframe = true;
        else MaterialPhong.wireframe = false;
});
```

Exemple de menu GUI : cas d'un plan

## Propriétés lumineuses (Phong) du plan

```
//opacite : reel entre 0 et 1
guiPlanPhong.add(menuGUI, 'opacitePhong', 0, 1).onChange(
    function (e) {// retour chariot pour le cours
        MaterialPhong.opacity=e;
   });// fin opacitePhong
```

## Propriétés lumineuses (Phong) du plan

```
//opacite : reel entre 0 et 1
guiPlanPhong.add(menuGUI, 'opacitePhong', 0, 1).onChange(
    function (e) {// retour chariot pour le cours
        MaterialPhong.opacity=e;
   }); // fin opacitePhong

//FilDeFer
guiPlanPhong.add(menuGUI, 'FilDeFer', ['Oui', 'Non']).onChange(
    function (e) {// retour chariot pour le cours
        if (e=='Oui')
            MaterialPhong.wireframe = true;
        else MaterialPhong.wireframe = false;
   });
});
```

## Propriétés lumineuses (Phong) du plan

```
//opacite : reel entre 0 et 1
guiPlanPhong.add(menuGUI, 'opacitePhong', 0, 1).onChange(
    function (e) {// retour chariot pour le cours
        MaterialPhong.opacity=e;
   }); // fin opacitePhong

//FilDeFer
guiPlanPhong.add(menuGUI, 'FilDeFer', ['Oui', 'Non']).onChange(
    function (e) {// retour chariot pour le cours
        if (e=='Oui')
            MaterialPhong.wireframe = true;
        else MaterialPhong.wireframe = false;
    });

gui.add(menuGUI, "actualisation");
menuGUI.actualisation();
```

Exemple de menu GUI : cas d'un plan

- 1 HTML et Three.js
- 2 Points et Vecteurs
  - Définitions
  - Méthodes communes à THREE.Vector $n$ ,  $n \in \{2;3\}$
  - Cas du plan i.e.  $n = 2$
  - Cas de l'espace i.e.  $n = 3$
- 3 La caméra
- 4 Les lumières
  - Lumière ambiante
  - Définition et propriétés communes des autres lumières
  - Lumière ponctuelle et spot
  - Spot et lumière directionnelle : ombre
  - Le spot
- 5 Les matériaux
- 6 Les courbes
  - Cas classique
  - Cas spécifique de THREE.js
  - Courbes de Bézier polynomiales
- 7 Les surfaces primitives
  - Le principe
  - Cas particulier des surfaces planes
  - Les surfaces non planes
- 8 Transformations géométriques de  $\mathcal{E}_3$ 
  - Application affine : translation
  - Application affine ou vectorielle
- 9 L'Animation
- 10 Création de menu G.U.I.
  - Théorie
  - Exemple avec un plan
- 11 Le C.S.G.

## Fichier H.T.M.L. et principe

Dans le fichier H.T.M.L., ajouter :

```
<script type="text/javascript" charset="UTF-8"  
       src="*/libs/other/ThreeBSP.js"></script>
```

où la syntaxe **\*/** représente le chemin idoine.

## Fichier H.T.M.L. et principe

Dans le fichier H.T.M.L., ajouter :

```
<script type="text/javascript" charset="UTF-8"  
       src="*/libs/other/ThreeBSP.js"></script>
```

où la syntaxe \*/ représente le chemin idoine.

---

```
let MaillageMat = new THREE.MeshBasicMaterial({ ... });
```

## Fichier H.T.M.L. et principe

Dans le fichier H.T.M.L., ajouter :

```
<script type="text/javascript" charset="UTF-8"  
       src="*/libs/other/ThreeBSP.js"></script>
```

où la syntaxe **\*/** représente le chemin idoine.

---

```
let MaillageMat = new THREE.MeshBasicMaterial({ ... });
```

### Principe (Création d'une surface pour le C.S.G.)

1/ *Pour chacune des surfaces :*

## Fichier H.T.M.L. et principe

Dans le fichier H.T.M.L., ajouter :

```
<script type="text/javascript" charset="UTF-8"  
       src="*/libs/other/ThreeBSP.js"></script>
```

où la syntaxe **\*/** représente le chemin idoine.

---

```
let MaillageMat = new THREE.MeshBasicMaterial({ ... });
```

### Principe (Création d'une surface pour le C.S.G.)

1/ Pour chacune des surfaces :

a/ Créer la surface « géométrique » :

```
let surfGeom = new THREE....
```

## Fichier H.T.M.L. et principe

Dans le fichier H.T.M.L., ajouter :

```
<script type="text/javascript" charset="UTF-8"  
       src="*/libs/other/ThreeBSP.js"></script>
```

où la syntaxe **\*/** représente le chemin idoine.

---

```
let MaillageMat = new THREE.MeshBasicMaterial({ ... });
```

### Principe (Création d'une surface pour le C.S.G.)

1/ Pour chacune des surfaces :

a/ Créer la surface « géométrique » :

```
let surfGeom = new THREE....
```

b/ Créer le maillage de la surface :

```
let surfMat = new THREE.Mesh(surfGeom, MaillageMat);
```

## Fichier H.T.M.L. et principe

Dans le fichier H.T.M.L., ajouter :

```
<script type="text/javascript" charset="UTF-8"  
       src="*/libs/other/ThreeBSP.js"></script>
```

où la syntaxe **\*/** représente le chemin idoine.

---

```
let MaillageMat = new THREE.MeshBasicMaterial({ ... });
```

### Principe (Création d'une surface pour le C.S.G.)

1/ Pour chacune des surfaces :

a/ Créer la surface « géométrique » :

```
let surfGeom = new THREE....
```

b/ Créer le maillage de la surface :

```
let surfMat = new THREE.Mesh(surfGeom, MaillageMat);
```

c/ Créer la surface pour le C.S.G. :

```
let surfCSG = new ThreeBSP (surfMat);
```

## Fichier H.T.M.L. et principe

Dans le fichier H.T.M.L., ajouter :

```
<script type="text/javascript" charset="UTF-8"  
        src="*/libs/other/ThreeBSP.js"></script>
```

où la syntaxe **\*/** représente le chemin idoine.

---

```
let MaillageMat = new THREE.MeshBasicMaterial({ ... });
```

### Principe (Création d'une surface pour le C.S.G.)

1/ Pour chacune des surfaces :

a/ Créer la surface « géométrique » :

```
let surfGeom = new THREE....
```

b/ Créer le maillage de la surface :

```
let surfMat = new THREE.Mesh(surfGeom, MaillageMat);
```

c/ Créer la surface pour le C.S.G. :

```
let surfCSG = new ThreeBSP (surfMat);
```

2/ Réaliser l'opération booléenne souhaitée ;

## Fichier H.T.M.L. et principe

Dans le fichier H.T.M.L., ajouter :

```
<script type="text/javascript" charset="UTF-8"
       src="*/libs/other/ThreeBSP.js"></script>
```

où la syntaxe **\*/** représente le chemin idoine.

```
let MaillageMat = new THREE.MeshBasicMaterial({ ... });
```

### Principe (Création d'une surface pour le C.S.G.)

1/ Pour chacune des surfaces :

a/ Créer la surface « géométrique » :

```
let surfGeom = new THREE....
```

b/ Créer le maillage de la surface :

```
let surfMat = new THREE.Mesh(surfGeom, MaillageMat);
```

c/ Créer la surface pour le C.S.G. :

```
let surfCSG = new ThreeBSP (surfMat);
```

2/ Réaliser l'opération booléenne souhaitée ;

3/ Appliquer le matériau.

## C.S.G. entre une sphère et un cylindre

Définition de la sphère :

```
//sphere géométrique  
let sphereGeom = new THREE.SphereGeometry( ... );
```

## C.S.G. entre une sphère et un cylindre

Définition de la sphère :

```
//sphere geometrique  
let sphereGeom = new THREE.SphereGeometry( ... );  
//maillage de la sphere  
let sphereMaille = new THREE.Mesh(sphereGeom, wireFrameMat);
```

## C.S.G. entre une sphère et un cylindre

Définition de la sphère :

```
//sphere geometrique  
let sphereGeom = new THREE.SphereGeometry( ... );  
//maillage de la sphere  
let sphereMaille = new THREE.Mesh(sphereGeom, wireFrameMat);  
// sphere pour le C.S.G.  
let sphereCSG = new ThreeBSP(sphereMaille);
```

## C.S.G. entre une sphère et un cylindre

Définition de la sphère :

```
//sphere geometrique
let sphereGeom = new THREE.SphereGeometry( ... );
//maillage de la sphere
let sphereMaille = new THREE.Mesh(sphereGeom, wireFrameMat);
// sphere pour le C.S.G.
let sphereCSG = new ThreeBSP(sphereMaille);
```

Définition du cylindre :

```
// cylindre geometrique
let CylConeGeom = new THREE.CylinderGeometry( ... );
```

## C.S.G. entre une sphère et un cylindre

Définition de la sphère :

```
//sphère géométrique
let sphereGeom = new THREE.SphereGeometry( ... );
//maillage de la sphère
let sphereMaille = new THREE.Mesh(sphereGeom, wireFrameMat);
// sphère pour le C.S.G.
let sphereCSG = new ThreeBSP(sphereMaille);
```

Définition du cylindre :

```
// cylindre géométrique
let CylConeGeom = new THREE.CylinderGeometry( ... );
//maillage du cylindre
let CylConeAmora = new THREE.Mesh(CylConeGeom, wireFrameMat);
```

## C.S.G. entre une sphère et un cylindre

Définition de la sphère :

```
//sphère géométrique
let sphereGeom = new THREE.SphereGeometry( ... );
//maillage de la sphère
let sphereMaille = new THREE.Mesh(sphereGeom, wireFrameMat);
// sphère pour le C.S.G.
let sphereCSG = new ThreeBSP(sphereMaille);
```

Définition du cylindre :

```
// cylindre géométrique
let CylConeGeom = new THREE.CylinderGeometry( ... );
//maillage du cylindre
let CylConeAmora = new THREE.Mesh(CylConeGeom, wireFrameMat);
// cylindre pour le C.S.G.
let CylConeCSG = new ThreeBSP(CylConeAmora);
```

## C.S.G. entre une sphère et un cylindre

Définition de la sphère :

```
//sphère géométrique
let sphereGeom = new THREE.SphereGeometry( ... );
//maillage de la sphère
let sphereMaille = new THREE.Mesh(sphereGeom, wireFrameMat);
// sphère pour le C.S.G.
let sphereCSG = new ThreeBSP(sphereMaille);
```

Définition du cylindre :

```
// cylindre géométrique
let CylConeGeom = new THREE.CylinderGeometry( ... );
//maillage du cylindre
let CylConeAmora = new THREE.Mesh(CylConeGeom, wireFrameMat);
// cylindre pour le C.S.G.
let CylConeCSG = new ThreeBSP(CylConeAmora);
```

Est-ce que la moutarde (de Dijon) monte au nez ?

## Union - intersection entre une sphère et un cylindre

```
// resultat geometrique du C.S.G.  
//union  
let resultCSGGeom = sphereCSG.union(CylConeCSG);
```

## Union - intersection entre une sphère et un cylindre

```
// resultat geometrique du C.S.G.  
//union  
let resultCSGGeom = sphereCSG.union(CylConeCSG);  
  
//maillage du resultat du C.S.G.  
let resultCSG = resultCSGGeom.toMesh();  
//definition du materiau, de l'ombrage et ajout dans la scene  
resultCSG.material = MaterialPhong;  
resultCSG.castShadow = true;  
resultCSG.receiveShadow = true;  
scene.add(resultCSG);
```

---

## Union - intersection entre une sphère et un cylindre

```
// resultat geometrique du C.S.G.
//union
let resultCSGGeom = sphereCSG.union(CylConeCSG);

//maillage du resultat du C.S.G.
let resultCSG = resultCSGGeom.toMesh();
//definition du materiau, de l'ombrage et ajout dans la scene
resultCSG.material = MaterialPhong;
resultCSG.castShadow = true;
resultCSG.receiveShadow = true;
scene.add(resultCSG);
```

Pour l'intersection, remplacer :

```
let resultCSGGeom = sphereCSG.union(CylConeCSG);
par :
//intersection
let resultCSGGeom = CylConeCSG.intersect(sphereCSG);
```

## Différence entre une sphère et un cylindre

Remplacer :

```
let resultCSGGeom = sphereCSG.union(CylConeCSG);
```

par :

## Différence entre une sphère et un cylindre

Remplacer :

```
let resultCSGGeom = sphereCSG.union(CylConeCSG);
```

par :

- pour un cylindre creusant une sphère :

```
//sphere moins cylindre
```

```
let resultCSGGeom = sphereCSG.subtract(CylConeCSG);
```

## Différence entre une sphère et un cylindre

Remplacer :

```
let resultCSGGeom = sphereCSG.union(CylConeCSG);
```

par :

- pour un cylindre creusant une sphère :

```
//sphere moins cylindre
```

```
let resultCSGGeom = sphereCSG.subtract(CylConeCSG);
```

- pour une sphère creusant un cylindre :

```
//cylindre moins sphere
```

```
let resultCSGGeom = CylConeCSG.subtract(sphereCSG);
```

Différence d'une sphère et d'un cylindre

Différence d'un cylindre et d'une sphère

# Tore et plan : cercles d'Yvon-Villarceau (Musée de l'Œuvre Notre-Dame de Strasbourg)



## Tore et plan : cercles d'Yvon-Villarceau (Musée de l'Œuvre Notre-Dame de Strasbourg)



## Définition (Cercles d'Yvon-Villarceau)

*Soit un tore, centré à l'origine, de rayon majeur  $R$  et de rayon mineur  $r$  avec  $R > r > 0$ .*



## Tore et plan : cercles d'Yvon-Villarceau (Musée de l'Œuvre Notre-Dame de Strasbourg)



## Définition (Cercles d'Yvon-Villarceau)

*Soit un tore, centré à l'origine, de rayon majeur  $R$  et de rayon mineur  $r$  avec  $R > r > 0$ .*

*Soit le plan d'équation :  $r x + \sqrt{R^2 - r^2} z = 0$ .*





### Définition (Cercles d'Yvon-Villarceau)

*Soit un tore, centré à l'origine, de rayon majeur  $R$  et de rayon mineur  $r$  avec  $R > r > 0$ .*

*Soit le plan d'équation :  $r x + \sqrt{R^2 - r^2} z = 0$ .*

*L'intersection entre le plan et le tore donne deux cercles appelés cercles d'Yvon-Villarceau.*



Cercles d'Yvon-Villarceau