Aide-Mémoire Three.js : Physique & Vecteurs

Three.js est optimisé pour la performance (60 FPS). Pour éviter de surcharger la mémoire (Garbage Collection), la librairie modifie souvent les objets **sur place** au lieu d'en créer de nouveaux. C'est la source n°1 de bugs en physique.

1. Le Vector3 : La brique fondamentale

`THREE.Vector3` représente un point (x, y, z) ou une direction/vitesse.

```
const v = new THREE.Vector3(1, 0, 0); // x=1, y=0, z=0
```

ATTENTION : La Mutabilité !

En mathématiques pures, $a + b$ crée un nouveau nombre. En Three.js, `a.add(b)` modifie `a`.

```
let a = new THREE.Vector3(1, 0, 0);
let b = new THREE.Vector3(0, 1, 0);

let c = a.add(b);

// Résultat :
// c vaut (1, 1, 0)
// a vaut (1, 1, 0) AUSSI ! L'original est perdu.
// b vaut (0, 1, 0) Inchangé.
```

2. Les Opérations Indispensables (Cinématique)

Voici les méthodes que vous utiliserez 90% du temps pour calculer des trajectoires.

Opération	Code Three.js	Mathématiques
Addition	`v1.add(v2)`	$\vec{v}_1 = \vec{v}_1 + \vec{v}_2$
Soustraction	`v1.sub(v2)`	$\vec{v}_1 = \vec{v}_1 - \vec{v}_2$
Multiplication (Scalaire)	`v1.multiplyScalar(k)`	$\vec{v}_1 = k \cdot \vec{v}_1$
Produit Scalaire	`let d = v1.dot(v2)`	$d = \vec{v}_1 \cdot \vec{v}_2$ *(Retourne un nombre)*
Produit Vectoriel	`v1.cross(v2)`	$\vec{v}_1 = \vec{v}_1 \times \vec{v}_2$ *(Perpendiculaire)*
Normaliser	`v1.normalize()`	$\vec{v}_1 = \frac{\vec{v}_1}{\|\vec{v}_1\|}$ *(Longueur devient 1)*
Longueur (Norme)	`let L = v1.length()`	$L = \|\vec{v}_1\|$

3. La « Golden Method » pour la Physique

Pour l'intégration d'Euler (pos = pos + vel $\cdot$ dt), n'utilisez pas `add` et `multiply` séparément (cela nécessiterait de cloner). Utilisez `addScaledVector` :

```
// La méthode PRO (Rapide et propre)
position.addScaledVector(velocity, dt);

// Équivalent à : position += velocity * dt
```

4. Gestion de la mémoire : Clone vs Copy

Dans la boucle `animate()`, évitez de créer des objets avec `new THREE.Vector3`. Cela ralentit le navigateur.

Règle d'or : Créez vos vecteurs de travail UNE FOIS (variables globales ou temporaires réutilisables).

MÉTHODE LENTE (À éviter) *Crée un nouvel objet à chaque tour.*

```
// Dans animate()
let temp = vel.clone();
temp.multiplyScalar(dt);
pos.add(temp);
```

MÉTHODE RAPIDE (Recommandée) *Réutilise les boîtes existantes.*

```
// Variable globale
const temp = new THREE.Vector3();

// Dans animate()
temp.copy(vel); // Copie les valeurs x,y,z
temp.multiplyScalar(dt);
pos.add(temp);
```

5. Calculs fréquents en Jeu / Physique

Distance entre deux objets A et B :

```
const dist = objectA.position.distanceTo(objectB.position);
```

Astuce Pro : Si c'est juste pour comparer (ex: « est-ce que je touche ? »), utilisez `distanceToSquared()` et comparez au carré du rayon. Ça évite une racine carrée coûteuse.

Direction de A vers B (Vecteur unitaire) :

```
// 1. On prend la destination (B)
const direction = new THREE.Vector3().copy(B.position);
// 2. On soustrait l'origine (A)
direction.sub(A.position);
// 3. On normalise (longueur 1)
direction.normalize();
```

6. Débogage Visuel

Ne devinez pas où sont vos vecteurs, affichez-les !

```
// ArrowHelper(direction, origine, longueur, couleur)
const arrow = new THREE.ArrowHelper(
    velocity.clone().normalize(), // Direction (Doit être unitaire !)
    sphere.position,              // Origine
    velocity.length(),            // Longueur
    0xffff00                      // Couleur
);
```