Session 12 : RigidBodies & Colliders

> **Introduction**
>
> Nous passons maintenant de la théorie (écrire notre propre moteur) à la pratique professionnelle (utiliser un middleware).
>
> Pour les projets Web/Three.js, nous utiliserons **Rapier** (écrit en Rust, compilé en WASM). Pour les projets C++/Raylib, nous utiliserons **Box2D** ou **PhysX**.
>
> Les concepts sont identiques à 99% entre tous ces moteurs.

1. Le Monde (Physics World)

Tout commence par la création d'un monde. C'est le conteneur qui gère la gravité et la liste des corps.

Initialisation (Rapier).

```
import RAPIER from '@dimforge/rapier3d-compat';

await RAPIER.init(); // Charger le WASM
let gravity = { x: 0.0, y: -9.81, z: 0.0 };
let world = new RAPIER.World(gravity);
```

2. RigidBody : L'Existence Physique

Un objet physique est séparé en deux concepts : le **Corps** (sa physique) et la **Forme** (sa collision).

> **Les 3 Types de Corps**
>
> 1. Dynamic : Soumis aux forces et aux collisions (Caisse, Balle, Joueur). C'est ce que nous avons fait jusqu'ici ($$F = m.a$$).
> 2. Static : Masse infinie, vitesse nulle. Ne bouge JAMAIS (Sol, Mur, Maison).
> 3. Kinematic : « Téléguidé ». Il a une vitesse infinie (il écrase tout) mais n'est pas affecté par les forces. On contrôle sa position manuellement (Plateforme mobile, Ascenseur).

Création d'un corps.

```
// 1. Définir le corps
let rigidBodyDesc = RAPIER.RigidBodyDesc.dynamic()
    .setTranslation(0.0, 5.0, 0.0);

// 2. Créer le corps dans le monde
let rigidBody = world.createRigidBody(rigidBodyDesc);
```

3. Collider : La Forme de Collision

Le RigidBody n'a pas de forme. C'est juste un point avec une masse. Pour qu'il rebondisse, il faut lui attacher un **Collider**.

> **Collider vs Mesh**
>
> - Mesh (Three.js) : Ce qu'on **voit** (milliers de triangles, textures).
> - Collider (Rapier) : Ce qui **touche** (formes primitives simples).
>
> **Règle d'or :** Toujours utiliser la forme la plus simple possible pour le collider (Sphère, Cube, Capsule) pour les performances.

Attacher une forme.

```
// Une boite de 1x1x1 (demi-extents = 0.5)
let colliderDesc = RAPIER.ColliderDesc.cuboid(0.5, 0.5, 0.5)
    .setRestitution(0.7)  // Rebond
    .setFriction(0.5);    // Frottement

world.createCollider(colliderDesc, rigidBody);
```

4. La Boucle de Simulation

Comme pour notre moteur, il faut avancer le temps à chaque frame.

> **Synchronisation Visuelle**
>
> Rapier calcule la physique, mais ne dessine rien. C'est à vous de copier la position du corps physique vers le mesh visuel à chaque frame.

```
function animate() {
    requestAnimationFrame(animate);

    // 1. Avancer la physique
    world.step();

    // 2. Copier les positions (Physique -> Graphique)
    mesh.position.copy(rigidBody.translation());
    mesh.quaternion.copy(rigidBody.rotation());

    renderer.render(scene, camera);
}
```

5. Travaux Pratiques

Voir le sujet « Siege Engine - Partie A ». Vous devez construire un mur stable qui s'effondre de manière réaliste.