# Session 11 : Moteurs Physiques & Middleware

### Introduction

Félicitations ! Vous avez écrit votre propre moteur physique basé sur Euleur puis Verlet. Vous comprenez maintenant la complexité cachée derrière de simples balles qui rebondissent : intégration, détection (Broad/Narrow phase) et résolution.

Dans l'industrie (Jeux Vidéo, Cinéma, Simulation), nous utilisons rarement des moteurs faits maison (« Homebrew ») pour la physique complexe. Nous utilisons des **Middlewares**.

Cette session explore les géants du marché, leurs spécificités, et comment choisir le bon outil pour Raylib ou Three.js.

## 1. Concepts avancés des moteurs commerciaux

Avant de comparer les moteurs, il faut comprendre les problèmes qu'ils résolvent et que notre petit moteur Verlet ne gère pas encore.

### Rigid Body vs Soft Body

- **Rigid Body (Corps Rigide) :** L'objet ne se déforme jamais. La distance entre deux atomes de l'objet reste constante. C'est 95% de la physique dans les jeux (caisses, personnages, voitures).
- **Soft Body (Corps Mou) :** L'objet se déforme (tissus, gelée, pneus). C'est ce que nous avons simulé avec les ressorts et Verlet ! C'est beaucoup plus coûteux en calcul (CPU).

### Le problème du Tunneling (CCD)

Dans notre moteur, si une balle va très vite, elle peut traverser un mur entre deux frames car $P_{\text{old}}$ est d'un côté et $P_{\text{new}}$ de l'autre.

Les moteurs commerciaux utilisent le CCD (Continuous Collision Detection). Au lieu de regarder des positions discrètes, ils extrudent le volume de l'objet le long de son vecteur vitesse pour vérifier s'il a « balayé » un obstacle.

### Sleeping (L'endormissement)

Pour optimiser, un moteur physique détecte quand un objet arrête de bouger (vitesse < seuil pendant $x$ frames). Il le met en état de Sleep. L'objet est retiré des calculs d'intégration jusqu'à ce qu'une force extérieure le réveille.

**Sans cela, un empilement de 1000 caisses consommerait 100% du CPU même immobile.**

## 2. Les Géants du C++ (Industrie AAA)

Ces moteurs sont les standards de l'industrie pour les jeux PC et Consoles.

| Moteur | Spécificités |
|---|---|
| PhysX (NVIDIA) | • Le standard actuel (utilisé par Unity et Unreal Engine jusqu'à récemment).<br>• Très rapide, très stable.<br>• Spécificité : Peut utiliser la carte graphique (GPU) pour les particules et fluides complexes. |
| Havok | • Le vétéran (Half-Life 2, Zelda Breath of the Wild).<br>• Incroyablement robuste pour les **Character Controllers** (mouvements de personnages).<br>• Très cher (licence propriétaire historique), mais performant. |
| Jolt Physics | • La nouvelle star (Open Source).<br>• Utilisé dans **Horizon Forbidden West**.<br>• Spécificité : Conçu pour le Multithreading massif. Il écrase les autres sur les processeurs modernes à plusieurs cœurs. |

## 3. Le Roi de la 2D : Box2D

> **Box2D**
>
> Si vous avez joué à **Angry Birds**, **Limbo** ou **Shovel Knight**, vous avez utilisé Box2D.
>
> Créé par Erin Catto (Blizzard), c'est un moteur « Impulse Based » (comme notre résolution de collision, mais plus mathématique avec des matrices de masse).
>
> C'est la référence absolue pour la 2D. Il est déterministe, stable et gère parfaitement les « stacks » (piles d'objets).

## 4. Physique pour le Web (Three.js / Javascript)

Puisque nous travaillons parfois avec le Web, le choix du moteur est critique car le JavaScript est moins performant que le C++.

| Moteur | Techno | Verdict |
|---|---|---|
| Cannon.js | Pur JS | • Facile à utiliser avec Three.js.<br>• Lent si > 100 objets.<br>• Abandonné, mais fork « Cannon-ES » actif. |
| Ammo.js | WASM (C++) | • C'est le moteur « Bullet » compilé en WebAssembly.<br>• Très complet mais lourd à charger.<br>• API complexe et documentation difficile. |
| Rapier | Rust -> WASM | • Le choix moderne. |

| Moteur | Techno | Verdict |
|---|---|---|
| | | • Écrit en Rust, compilé pour le web.<br>• Extrêmement rapide et stable.<br>• API moderne et bien documentée. |

### Pourquoi Rapier ?

Si vous devez choisir un moteur aujourd'hui pour un projet Web/Three.js, choisissez Rapier. Sa structure de données (SIMD) permet de gérer des milliers d'objets sans ralentir le navigateur, là où Cannon.js s'effondrerait.

## 5. Déterminisme vs « Gameplay »

### Le Déterminisme

Une simulation est dite déterministe si : **« Pour un état initial donné et un delta time fixe, le résultat est strictement identique bit-à-bit sur n'importe quel ordinateur. »**

C'est vital pour :

1. Le Multijoueur (Pour que tous les joueurs voient la même caisse tomber au même endroit).
2. Les Replays (Pour rejouer une partie enregistrée).

Les flottants (`float`) ne sont pas gérés pareil par tous les processeurs (Intel vs AMD vs ARM). Les moteurs comme Box2D ou des versions « Fixed Point » s'efforcent d'être déterministes.

### Conclusion : Faut-il faire son propre moteur ?

- OUI pour apprendre (ce cours).
- OUI pour des cas très simples (Pong, particules visuelles sans gameplay).
- OUI pour des cas très spécifiques (Simulation de cordes pure, simulation de sable).
- NON pour un jeu commercial complexe. La gestion des collisions 3D (Mesh vs Mesh), l'optimisation spatiale (Octrees/BVH) et la stabilité numérique sont des années de travail que PhysX ou Rapier vous offrent gratuitement.