TP Noté : La Boîte à Particules (Broad & Narrow Phases)

Objectif du TP

L'objectif est de créer une simulation de « gaz » haute performance. Vous devez gérer 500+ sphères rebondissant dans un cube 3D. Le défi est double : assurer une physique réaliste (Narrow Phase) et maintenir 60 FPS via un partitionnement spatial (Broad Phase).

1. Barème de Notation (Total /30)

Critère d'évaluation	Points
Narrow Phase : Calcul d'impulsion 3D, restitution et correction d'overlap.	10 pts
Broad Phase : Implémentation d'une Grille Uniforme (Spatial Hashing).	10 pts
Environnement : Gestion des rebonds sur les 6 faces du cube conteneur.	5 pts
Optimisation & Code : Gestion de la mémoire (pas de `new` en boucle) et GUI.	5 pts

2. La Narrow Phase (Physique)

Calcul de l'impulsion

Détecter une collision entre 2 sphères est très simple :

$$\|P_A - P_B\| < 2R$$

Lorsqu'une collision est détectée entre deux sphères A et B, vous pouvez calculer l'impulsion j (en utilisant la formule de restitution) et appliquer l'impulsion à chaque sphère.

3. La Broad Phase (Optimisation)

La Grille Uniforme

Le monde est divisé en cases de taille $2 \times$ Diamètre.

- Enregistrement : À chaque frame, chaque boule est indexée dans une cellule : $X_{cell} = \left\lfloor \frac{P_x}{S} \right\rfloor$.
- Voisinage : Pour une boule donnée, vous ne devez tester que les boules présentes dans sa cellule et les 26 cellules adjacentes (en 3D).
- Performance : Vous devez inclure un bouton dans la GUI pour activer/désactiver la Broad Phase et constater la différence de FPS.

4. Contraintes Techniques

> ⚠ **Optimisation de la mémoire**
>
> Dans la fonction `updatePhysics`, il est strictement interdit de créer de nouveaux objets Three.js (`new THREE.Vector3()`).
> - Utilisez des variables globales ou des variables `static` pour vos calculs temporaires.
> - La création d'objets en boucle déclenche le **Garbage Collector** et provoque des saccades (stuttering).

Structure attendue du Spatial Hashing.

```
class SpatialGrid {
  constructor(cellSize) {
    this.cellSize = cellSize;
    this.cells = new Map();
  }

  // Génère une clé unique pour une position (ex: "5|2|-1")
  getKey(v) {
    const x = Math.floor(v.x / this.cellSize);
    const y = Math.floor(v.y / this.cellSize);
    const z = Math.floor(v.z / this.cellSize);
    return `${x}|${y}|${z}`;
  }

  // À remplir par l'étudiant...
}
```

5. Bonus : Friction et Gravité

Ajoutez un curseur de gravité et un coefficient de friction lors des collisions avec les murs pour obtenir un comportement de « tas de sable » au fond de la boîte (+2 pts bonus).

Date de rendu : Fin de semaine (2026-01-12)
Livrable : Archive .zip (Code source + Démo fonctionnelle)