Session 16 : Les Systèmes de Particules et le GPU

> **Objectif et Contexte**
>
> Cette session explore comment simuler des **Systèmes de Particules** réalistes en appliquant les lois de la physique.
>
> Jusqu'à présent, nous avons géré un petit nombre de Rigid Bodies. Les systèmes de particules (fumée, feu, explosions, pluie) exigent de gérer des **millions** d'entités avec une optimisation drastique.
>
> **L'objectif est d'apprendre :**
> - Le cycle de vie d'une particule (Émetteur, Forces, Collision).
> - L'application de forces continues (Gravité, Frottement de l'air).
> - L'optimisation massive par calcul parallèle (GPU).

1. Les Fondations : Architecture et Cycle de Vie

Le système de particules est structuré autour d'un **Émetteur** qui gère un **pool** de particules, chacune ayant son propre cycle de vie.

1.1 La Particule (Particle)

Chaque particule est une entité qui possède un état minimal pour sa simulation.

Structure de l'État d'une Particule.

Une particule type stocke au minimum :
- ***Position*** (x, y, z)
- ***Vitesse*** (v_x, v_y, v_z)
- ***Accélération*** (a_x, a_y, a_z) *(où l'on applique la somme des forces / masse)*
- ***Durée de Vie*** *(temps écoulé, temps total)*
- ***Propriétés Visuelles*** *(Taille, Couleur, Transparence/Alpha)*

1.2 L'Émetteur et le Pooling

- **Rôle de l'Émetteur :** Créer un flux continu de particules en initialisant leur état (position de départ, vitesse aléatoire dans un cône, durée de vie, etc.).
- **Optimisation (Pooling) :** Au lieu de créer et détruire des objets (coûteux en mémoire) lorsque leur durée de vie est épuisée, on les marque comme "inactifs" et on les réinitialise pour une nouvelle émission.

2. Interactions Physiques

L'étape cruciale est l'application des forces et l'intégration du mouvement à chaque frame.

2.1 Application des Forces, intégration, cinématique

Pour chaque particule active, on va intégrer l'équation de la physique que nous avons vue dans les sessions précédentes. En partant des forces, qui sont liées a l'accélération (Newton), puis on va mettre à jour la vitesse et la position de la particule.

- **Gravité :** Force constante et verticale (ex: pluie, gravats). $F_g = m \cdot \vec{g}$.
- **Vent / Force Externe :** Force constante ou variant selon le temps, ajoutée à l'accélération.
- **Frottement (Drag) :** Force **visqueuse** opposée à la vitesse (rappel Session - Frottements).
  - $F_{\text{drag}} = -k \cdot \vec{v}$ (Pour les particules lentes)

- $F_{\{\text{drag}\}} = -C \cdot |\vec{v}|^2$ (Pour les particules rapides, ex: explosions)

2.2 Collision Simples et Rebonds

Les particules sont généralement simulées avec des collisions simplifiées contre l'environnement (plans, boîtes).

Gestion des Collisions de Particules. *Lors d'une collision entre une particule et un plan (le sol, un mur) :*

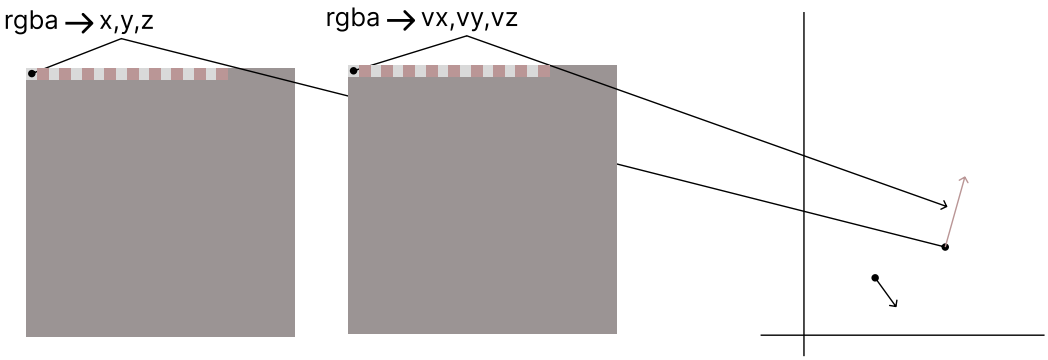
1. ***Détection :*** *On vérifie si la position P a traversé la surface (ex: $P.y < 0$).*
2. ***Résolution (Impulsion Simplifiée) :***
   - *On **repositionne** la particule sur la surface pour éviter la pénétration.*
   - *On **modifie la vitesse** (rebond) : on inverse sa composante normale et on la multiplie par un coefficient de **Restitution** (e, entre 0 et 1).*
   - *On applique un **Frottement** à la composante tangentielle de la vitesse (pour simuler la friction).*

3. Optimisation : Le Calcul Parallèle sur GPU

Simuler des milliers ou millions de particules sur le CPU est trop lent.

3.1 Le Goulot d'Étranglement CPU

- **Problème :** Même si le calcul pour une seule particule est simple, l'exécution séquentielle de l'itération `update(dt)` pour **chaque** particule devient un goulot d'étranglement de performance.
- **Solution Théorique :** L'architecture des particules est hautement parallèle : le calcul d'une particule est indépendant des autres. C'est parfait pour le **GPU**.



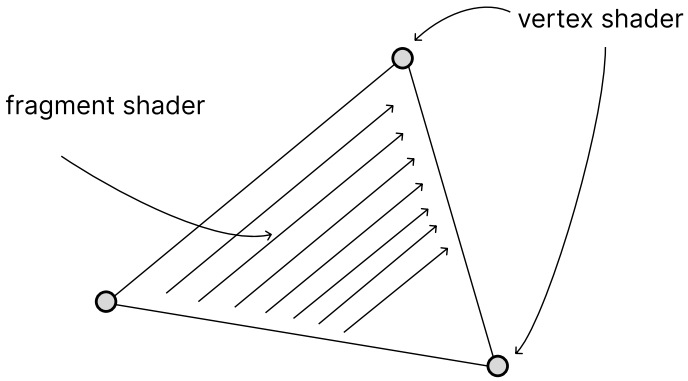


Figure 1: Intégration des particules sur le GPU

> **Le Calcul des Particules sur GPU (Compute Shaders)**
>
> L'approche moderne pour des millions de particules utilise le GPU :
> - **Principe :** On stocke les données de toutes les particules (Positions, Vitesses, Vie) dans une mémoire tampon accessible au GPU (`BufferAttribute` ou `Storage Buffer`).
> - **Exécution :** La fonction `update(dt)` (l'intégration, l'application des forces) est implémentée dans un programme (un **Compute Shader** ou un **Vertex Shader** astucieux).
> - **Parallélisme Massif :** Le GPU lance des milliers de threads simultanément, chaque thread calculant l'état d'une particule en une seule fois.
>
> **Résultat :** Un gain de performance permettant de passer de quelques milliers à plusieurs millions de particules en temps réel.

4. Travaux Pratiques et Cas d'Usage

4.1 Simulation de la Pluie / Neige

- **Particules :** Masse faible, durée de vie limitée.
- **Forces :** Gravité forte, Frottement de l'air (Drag) modéré.
- **Collision :** Avec un plan (le sol). Coefficient de restitution très faible ($e \approx 0$) pour simuler l'absorption.

4.2 Simulation d'une Explosion

- **Particules :** Grande quantité, durée de vie courte.
- **Forces :** Vitesse initiale **radiale** (s'éloignant du centre). Frottement de l'air quadratique pour ralentir rapidement les particules.
- **Cas d'Usage :** Explosion, débris, feu d'artifice.

4.3 Mise en Œuvre du Projet

L'objectif du TP est de créer un émetteur de base et d'y intégrer un moteur de physique simple (Euler ou Verlet) pour simuler :

1. De la **Pluie/Neige** tombant au sol.
2. Un **Jet d'Eau** parabolique avec frottement.

Rappel : Intégration Numérique (Euler/Verlet)

Pour chaque particule :

1. Calculer $\texttt{vec}\{a\} = \texttt{frac}\{\sum \texttt{vec}\{F\}\}\{\texttt{text}\{\text{masse}\}\}$
2. Mettre à jour la vitesse : $\texttt{vec}\{v\}_{\{n+1\}} = \texttt{vec}\{v\}_n + \texttt{vec}\{a\}c \cdot \Delta t$
3. Mettre à jour la position : $\texttt{vec}\{P\}_{\{n+1\}} = \texttt{vec}\{P\}_n + \texttt{vec}\{v\}_{\{n+1\}} \cdot \Delta t$
4. Gérer les **Collisions** (repositionnement + modification de v).
5. Décrémenter la durée de vie.