Grenoble INP : Ensimag

Compte Rendu TP

Simulation de systemes de particules

Authors :

Sana Ailla
Morad Laglil

Academic Year 2022/2023

Contents

Lab2 : Particules

1.1 Mise en place des structures de données

Nous avons implémenté la classe Particule qui possède les attributs suivants : position, vitesse, masse, identifiant, catégorie, force appliquée par les autres particules dans l'espace, ainsi qu'un attribut contenant la force à l'instant t-1 pour des raisons algorithmiques.

1.1.1 Choix de collection de particules

Pendant la simulation, les particules doivent être stockées dans une collection. Il est donc important de choisir une collection qui soit adaptée à un grand nombre de particules (dizaines, centaines, milliers, etc.). Nous avons testé l'insertion de 2^k particules pour k allant de 6 à 20 dans différentes collections : vector, deque, list et forward list. Le graphique ci-dessus montre les performances en temps pour chaque collection.

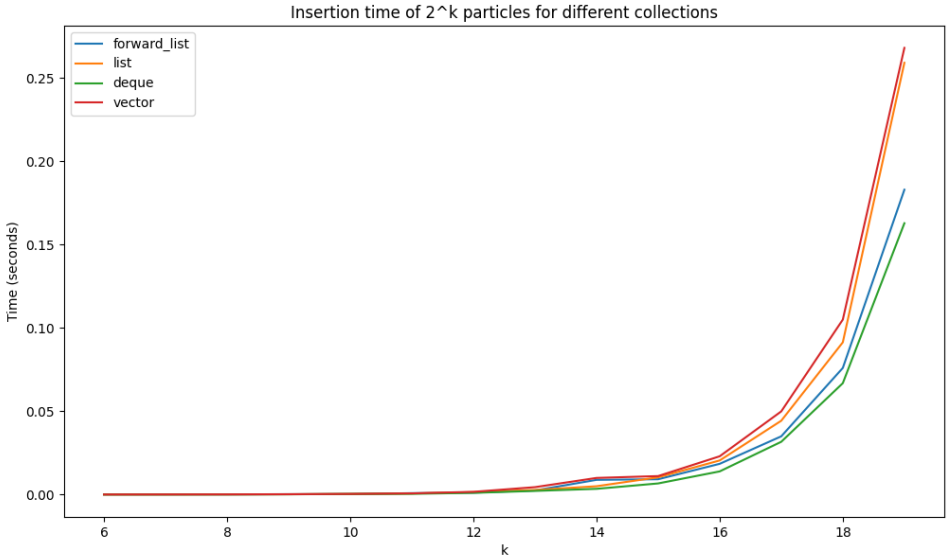

Figure 1.1: **Performances (temps en seconde) des collections pour l'insertion**

En testant 'insertion de particules pour différentes tailles avec chaque collection, on peut observer que la deque est la plus performante pour toutes les tailles testées. En effet, elle offre un temps d'insertion plus rapide que les autres collections, même pour des tailles très grandes.

En revanche, on peut remarquer que la performance relative des différentes collections varie selon la taille. Pour les plus petites tailles, la différence entre les performances est moins significative, mais pour les plus grandes tailles **(changement a partir de k = 13)**, la deque est nettement plus rapide que les autres collections.

Cela s'explique par la structure interne de la deque, qui permet une insertion et une suppression rapide en début et fin de file, ce qui est utile dans la simulation de particules.

En conclusion, pour notre simulation de particules, nous avons choisi d'utiliser une deque comme collection pour stocker les particules, en raison de ses performances supérieures aux autres collections pour des tailles de particules importantes.

1.2 Mise en place des structures de données

1.2.1 Implémentation de Strömer-Verlet

Dans la classe Particule, nous avons implémenté une fonction `stromer_verlet` qui prend en paramètre la collection de particules, le temps de fin de la simulation et le pas de temps. Cette fonction permet l'évolution de l'ensemble de particules en permettant l'interaction et la modification de leurs positions. À la fin de l'exécution, un fichier `solar_system.positions` est généré qui contient la position de chaque astre à chaque itération sur la même ligne.

Le résultat final de la simulation en affichant les différentes positions au cours de la simulation (à chaque instant) des 4 astres donnés (Terre, Jupyter, Halley et le Soleil) est le suivant :

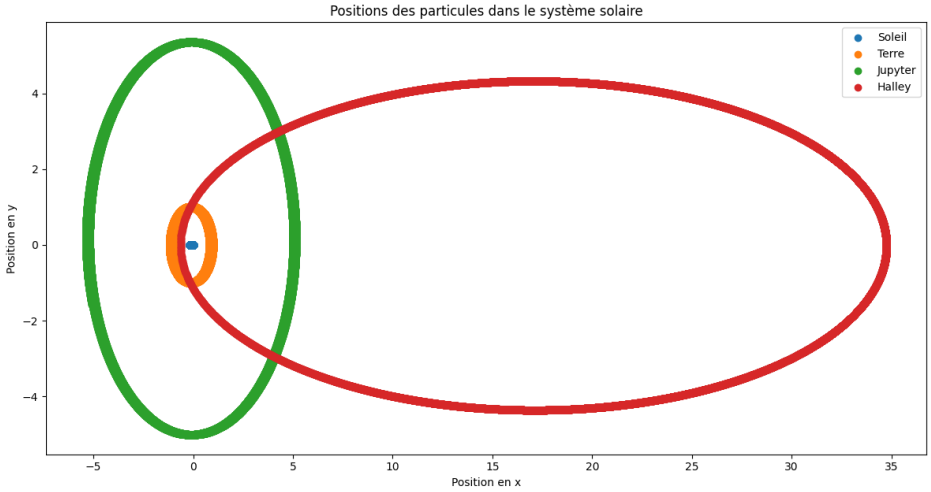

Figure 1.2: **Les trajectoires des astres dans un système solaire**

La figure ci dessous montre les trajectoires de quatre astres: Terre, Jupiter, Halley et le Soleil. Les trajectoires elliptiques des astres sont clairement visibles, avec le Soleil en position fixe au centre, ce qui est proche de la réalité. On peut également observer que les orbites de Jupiter et de Halley sont plus excentriques que celle de la Terre, ce qui signifie que ces planètes ont une trajectoire plus allongée par rapport au Soleil.

Lab3 : Vecteur et operateurs

2.1 Enrichissement des structures de données

2.1.1 Classe vecteur

Nous avons créé une classe Vecteur qui contient un attribut `val` de type `std::array<double,dim>` qui représente les différentes valeurs des attributs en fonction de la dimension de l'espace de travail (1D, 2D ou 3D).

Cette classe nous a permis de réaliser des opérations mathématiques sur les vecteurs telles que l'addition, la soustraction, la multiplication par un scalaire, le produit scalaire, le produit vectoriel, la division, l'accès par [] et le calcul de la norme ainsi que l'affichage grâce à l'opérateur «.

Par la suite, nous avons modifié la classe "Particule" pour prendre en compte cette classe "Vecteur" dans les attributs tels que la vitesse, les forces et la position, ainsi que pour inclure l'ensemble des calculs réalisés dans l'algorithme de stromer-verlet.

2.1.2 Univers des particules

On a implémenté une classe univers qui permet de créer et de manipuler un univers de particules en 1D, 2D ou 3D. Pour cela, nous avons utilisé une structure de données composée d'une collection de particules. Cette classe nous permet de faire avancer les particules, de calculer les forces d'interaction, de modifier les vitesses et d'afficher l'état de l'univers, c'est-à-dire les positions des particules à chaque instant.

Nous avons ajouté différentes méthodes pour mettre à jour l'univers, notamment une méthode d'évolution (stromer-verlet dans le lab2) pour faire avancer les particules dans le temps.

La figure ci-dessous montre un exemple d'univers 3D de particules uniformément distribuées sur le cube $[0,1]^3$.

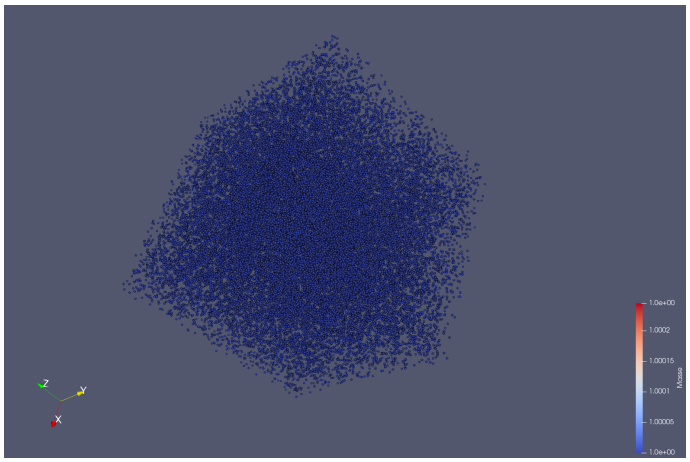

Figure 2.1: **Univers 3D de particules uniformément distribuées sur** $[0,1]^3$

2.1.3 Calcul d'interactions

Le calcul des interactions entre particules consiste à itérer sur toutes les paires possibles de particules et calculer la force de gravitation entre chaque paire. On a commencé par implementer cette approche, appelée méthode brute-force, est simple à mettre en œuvre mais a une complexité algorithmique en $O(n^2)$, où n est le nombre de particules dans le système. Ainsi, pour un système avec un grand nombre de particules, cette méthode peut rapidement devenir très coûteuse en termes de temps de calcul.

Ce calcul des interactions entre particules peut être réduit de moitié en exploitant la symétrie de la force gravitationnelle, qui s'applique dans les deux sens entre deux particules. Ainsi, au lieu de parcourir toutes les paires possibles de particules, on peut parcourir uniquement les paires distinctes, c'est-à-dire les paires (i, j) où $i < j$.

Avec cette approche, la complexité algorithmique devient $O(n(n-1)/2)$, ce qui est équivalent à $O(n^2)$. Cependant, le temps de calcul est divisé par deux car on ne calcule chaque paire qu'une seule fois.

Lab4 : Decoupage de l'espace

Lab5 : Tests et visualisation

4.1 Tests

Nous avons utilisé Google Test pour implémenter un ensemble de tests unitaires pour différentes classes (Vecteur, Particule, Univers et Cellule) afin de valider notre implémentation.

Pour les tests fonctionnels, nous avons mis en place les deux simulations des labos précédents (lab2 : système solaire et lab4 : collision entre deux objets). La validation des résultats a été effectuée en visualisant l'évolution du système au cours du temps à l'aide de Python ou Paraview.

4.2 Visualisation

Pour visualiser les résultats, nous avons implémenté une fonction `createVtkFile` dans la classe Univers pour créer un fichier contenant l'état de l'univers à l'instant où elle est appelée. Cette fonction crée un fichier au format VTK, qui peut être ouvert avec différents logiciels de visualisation comme Paraview. Nous avons également ajouté des options pour spécifier le nom du fichier et le nombre d'itérations à sauvegarder.