TP 11 : Problème à N-corps

Cours de modélisation numérique

19 mai 2023

Introduction

L'algorithme de Barnes-Hut, présenté au cours, permet d'améliorer les performances d'une simulation de N corps interagissant entre eux, par exemple les étoiles d'une galaxie. Pour ce faire, on approxime les groupes de corps lointains comme ne formant qu'un seul corps. En pratique, cette approximation s'appuie sur l'utilisation des arbres quaternaires.

Pour ce TP, le code `barneshut.py` vous est fourni sur Moodle. Ce code implémente entièrement le problème des n-corps en 2 dimensions approximé par l'algorithme de Barnes-Hut, ainsi qu'une instance d'un problème et un moyen de le visualiser. Prenez le temps de comprendre ce code et de vérifier qu'il fonctionne.

Travail à faire

1. Généralisation en 3D

Faites les modifications nécéssaires au code qui vous est donné afin de le généraliser au cas tridimensionnel.

Notez bien que le code original contient déjà des fonctions pour visualiser le résultat en 2D et en 3D, nommées respectivement `plot_bodies_2D` et `plot_bodies_3D`. Il en va de même pour l'initialisation des impulsions, avec les fonctions `init_momentum_2D` et `init_momentum_3D`. Vous devez utiliser cette dernière pour initialiser les impulsions des étoiles dans le cas 3D.

Pour indication, si l'implémentation est correcte, la coordonnée Z du corps numéro 0 devrait être, à l'itération 500, $Z = 0.55682037$, avec $N = 200$ corps et tous les autres paramètres laissés par défaut.

2. Étude de performance et complexité en temps dans le cas 3D

Étudiez les performances de l'algorithme en fonction du nombre d'étoiles (évidemment, il est indiqué de commenter les lignes relatives à la visualisation, pour cette partie, ainsi que de moyenner les temps pris par plusieurs mesures différentes).

Implémentez une version naïve de l'algorithme, qui traite l'interaction de chaque étoile avec toutes les autres. Pour ce faire, dans la boucle principale, à la place de la fonction `verlet`, utilisez la fonction `verlet_bruteforce`, que vous aurez implémenté vous-même et qui correspond à la vision naïve décrite ci-dessus. N'oubliez pas, pour les mesures de performances, de commenter également les lignes relatives à la reconstruction du quadtree.

Présentez, sur un même graphe, les performances des deux algorithmes (temps d'exécutions en fonction du nombre d'étoiles). Analysez vos résultats. Que pouvez-vous dire de la complexité en temps de chaque algorithme ?

La fonction `clock()` du module `time` permet d'obtenir le temps écoulé depuis l'importation du module (Unix) ou depuis le premier appel à la fonction (Windows). Ainsi, le code suivant stocke dans la variable `temps_total` le temps d'exécution de la fonction `ma_fonction` :

```
import time
begin = time.clock()
ma_fonction()
end = time.clock()
temps_total = end - begin
```