ENSTA Paris - IP Paris

IN104

Problème à N corps

Simon Pala
Maxim Leclere

mai 2021

Table des matières

Introduction

Lien du Git : `SimonPala/IN104_Pala-Simon_Leclere-Maxim`

Le problème à N corps consiste à résoudre les équations du mouvement de N corps quelconques interagissant dans un champs gravitationnel (ici en dimension $n = 2$). L'intérêt d'une telle étude et de réussir à simuler le mouvement de N corps dans un espace à 2 dimensions qui interagissent entre eux par le biais de leur propre champ gravitationnel. Cette étude est fondamentale en science et particulièrement en astronomie classique, en effet, dans le cas où les effets de la relativité générale peuvent être négligés, elle permet de décrire la dynamique de notre système solaire.

La complexité de ce problème est très variable, on peut aller du simple problème à deux corps sans collision, à des problèmes bien plus complexes mettant en jeu plusieurs corps en prenant en compte, en plus, les collisions et des méthodes de résolution d'équations plus efficaces pour des modèles physiques plus pratiques à la résolution.

Nous avons donc fait le choix de parcourir cette plage de complexité en proposant, initialement un modèle plutôt simple, basé sur l'interaction de deux corps sans collisions. Ensuite, afin de se rapprocher de situations plus réalistes telles que des étoiles ou des corps solides, nous nous sommes appliqués à programmer des phénomènes de collision.

L'ensemble de cet univers est affiché à l'utilisateur par le biais d'un interface graphique et l'ensemble du projet a été codé en langage Python. L'objectif du cours était de se former à la programmation orienté objets et de se familiariser avec l'utilisation des tests unitaires.

1 Chronologie et Organisation

1.1 Chronologie

Dans un premier temps nous avons exposé les grandes thématiques à étudier lors de la réalisation de ce projet.

Ce problème ce décompose en 3 sous-parties plus ou moins indépendantes. La première est la mise en place d'un moteur physique pertinent pour la bonne modélisation du problème. Ensuite il faut déterminer une méthode de résolution d'équation efficace pour traiter ce problème pouvant engendrer une certaine complexité. Enfin, la dernière partie est l'implémentation d'une bonne interface graphique, efficace et pertinente.

Nous avons dans un premier temps implémenté une simulation du problème dite "naïve". Ensuite nous avons décidé de changer de moteur physique pour essayer d'être plus efficace en réduisant le nombre de calculs. Dans la continuité, nous avons essayé de prendre en compte le phénomène de collision entre les corps.

1.2 Organisation

Nous nous sommes répartis les tâches de façon à ce que Maxim s'occupe de la partie moteur physique et de la modélisation des collisions tandis que Simon s'est occupé de l'interface graphique et du rapport.

2 Architecture

2.1 Principe

Pour faire fonctionner notre programme, nous avons eu besoin de définir notre univers ainsi que les corps qu'il contient. Ensuite, nous avons besoin d'une fonction pour calculer les interactions à distances entre les corps et donc leur accélération à tout instant grâce à la loi de Newton :

$$\sum \vec{F} = m\vec{a}$$

On doit finalement résoudre informatiquement les équations de la physique en intégrant l'accélération pour décrire l'évolution du mouvement.

2.2 Structure

Notre fichier `main.py` est simplement un réceptacle : il prend en argument les classes définit précédement (`solveur`, `engine` et `camera`). Ce sont donc les fonctions présentes dans ces sous-programmes qui permettent de résoudre le problème.

`solveur :` La classe `solveur` permet l'approximation de l'intégration des fonctions et donc de résoudre des équations différentielles. On a donc programmer l'algorithme d'Euler à pas fixe qui permet d'obtenir une première approximation des résultats.

`engine :` La classe `engine` donne les informations sur le fonctionnement de notre univers, comment les corps interagissent entre eux. C'est donc dans `engine.py` que nous avons codé le calcul des forces gravitationnelles auxquelles chaque corps était soumis. On peut ensuite avec la loi de Newton déterminer l'accélération de chaque corps. C'est ici que la classe `solveur` intervient,

on intègre l'accélération pour obtenir le mouvement de chaque corps. Cette fonction peut être amélioré en terme de complexité en utilisant par exemple la simulation de Barnes-Hut.

`camera :` La classe `camera` correspond au lien entre l'univers virtuel et la fenêtre d'affichage de l'utilisateur. L'objectif est que les corps soient visibles dans leur mouvement. La caméra doit alors s'adapter pour que les corps restent dans le cadre.

3 Première approche : Méthode naïve

Comme énoncé dans l'introduction, nous avons commencé par implémenter une méthode naïve du problème. Cette partie est très importante puisqu'elle est la base des modifications que nous avons apportées par la suite. Nous allons donc expliciter un maximum les étapes de cette modélisation.

3.1 Moteur Physique

Le moteur physique adopté ici est le plus simple, c'est celle où l'accélération d'un corps est obtenue en calculant les $N-1$ forces que le corps subit.

Pour ce faire, il a fallut coder la force qu'un corps 2 exerce sur un corps 1.

On a l'implémentation suivante :

```
def gravitational_force(pos1, mass1, pos2, mass2):
    """ Return the force applied to a body in pos1 with mass1
        by a body in pos2 with mass2
    """
    r=Vector.norm(pos1-pos2)
    Force2sur1=(-G*mass1*mass2/(r*r*r))*(pos1-pos2)
    return Force2sur1
```

Ensuite, une méthode de dérivation vectorielle a permis, à partir d'un vecteur contenant les positions et vitesses à un instant t de N corps, de trouver les vitesses (évident) et les accélérations à ce même instant de ces N corps grâce à la dynamique newtonienne et des forces mises en jeu.

3.2 Méthode de résolution

L'application de la dynamique newtonienne vue au paragraphe précèdent nous impose l'implémentation d'une méthode de solveur d'équations. Il reçoit du moteur physique les positions initiales des variables et leurs dérivées puis calcule la nouvelle valeur des variables à un horizon donné.

La méthode que l'on a d'abord codée est Euler explicite, peu coûteuse mais également peu précise.

3.3 Interface graphique

L'interface graphique utilise le module `Pygame` qui permet à la fois l'affichage d'une fenêtre et également une interaction avec notre univers. En effet, il est agréable pour l'utilisateur de pouvoir zoomer, dézoomer ou se déplacer dans l'univers pour suivre l'évolution du mouvement des corps dans ce dernier. La fenêtre est alors une caméra que l'utilisateur peut déplacer à sa guise.

Dans la méthode naïve, on affiche simplement une fenêtre de taille finie.
On a l'implémentation suivante :

```
    def to_screen_coords(self, position):
        screen_coord= position*self.scale +
        self.screen_size/2-self.position*self.scale
        return screen_coord

    def from_screen_coords(self, position):
        coord_screen = 1/self.scale * (position +
        self.position*self.scale - self.screen_size/2)
        return coord_screen
```

Ainsi, les premiers éléments sont réunis pour voir les premiers résultats !

3.4 Premiers résultats

Avec pour conditions initiales (C-I) :

```
b1 = Body(Vector2(0, 0),
          velocity=Vector2(0, 0),
          mass=10,
          color=(255,240,10),
          draw_radius=10)
b2 = Body(Vector2(1, 1),
          velocity=Vector2(0, 0.2),
          mass=1,
          color=(255,148,23),
          draw_radius=5)
```

On a :

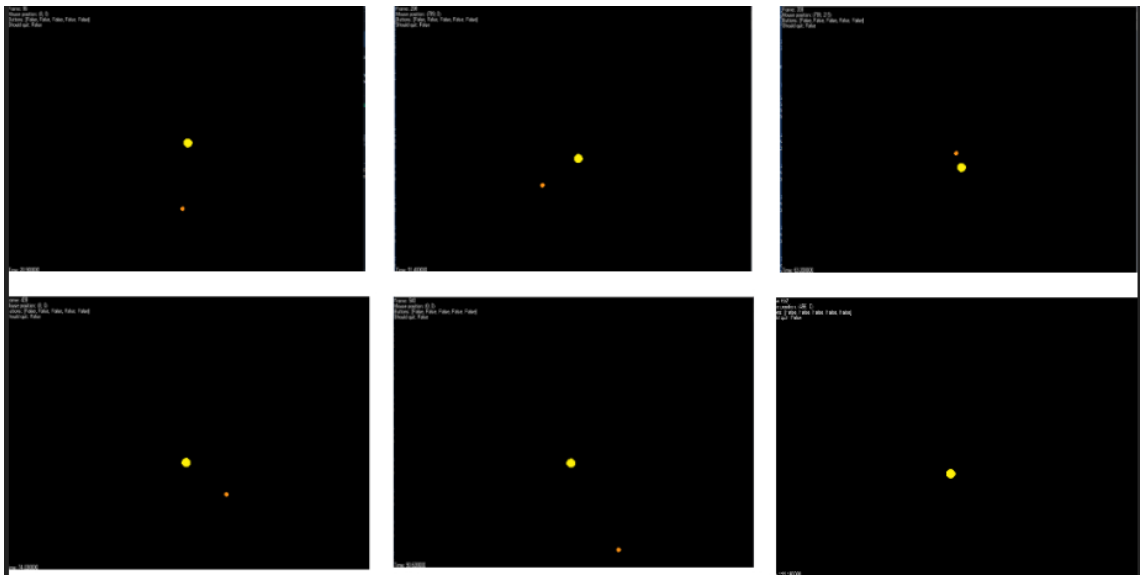

Figure 1 – Évolution du mouvement de deux corps en interaction.

En ajoutant des corps (5 par exemple) et en modifiant les conditions initiales pour avoir quelque chose de visuel, on a :

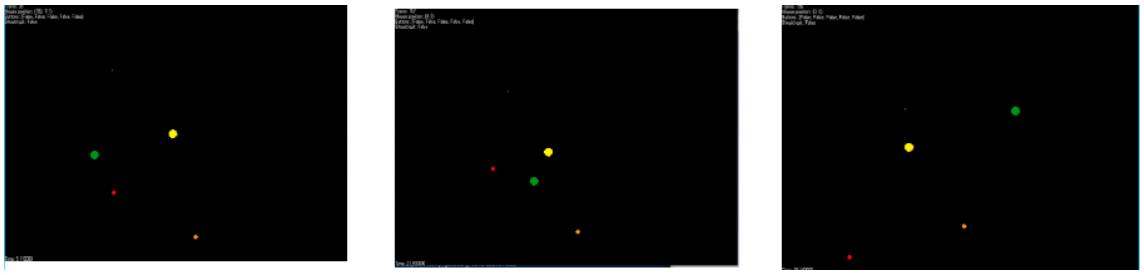

FIGURE 2 – Évolution du mouvement de 5 corps en interaction.

4 Amélioration de la méthode : Collisions

4.1 Moteur physique

Ce que nous avons modifié pour obtenir un modèle plus réaliste est l'ajout de collision. Ces collisions s'apparentent plus au rebond de particules ponctuelles les unes contre les autres dans un gaz que de la collisions de planètes ou d'étoiles. En effet, dans un modèle gravitationnel, on aura plutôt tendance à fusionner deux corps qui s'entrechoquent tel que

$$\begin{cases} m = m_1 + m_2 \\ m\vec{v} = m_1\vec{v}_1 + m_2\vec{v}_2 \end{cases}$$

Nous n'avons pas eu le temps d'implémenter ce modèle et avons choisit de nous concentrer sur le rebond.

Le modèle physique sur lequel nous nous sommes appuyés pour modéliser ce phénomène est plutôt simple, en effet, nous avons fait les considérations physiques suivantes, vraies dans un modèle de mécanique classique non relativiste :

- Pendant un choc entre deux corps, nous avons conservation des quantités de mouvement

$$m_1\vec{v}_1(t) + m_2\vec{v}_2(t) = m_1\vec{v}_1(t+dt) + m_2\vec{v}_2(t+dt)$$

- A l'instant après le choc, le vecteur vitesse $\vec{v}(t+dt)$ est symétrique au vecteur vitesse $\vec{v}(t)$ avant le choc par rapport au plan tangent aux corps passant par le point de contact (voir figure ci dessous).

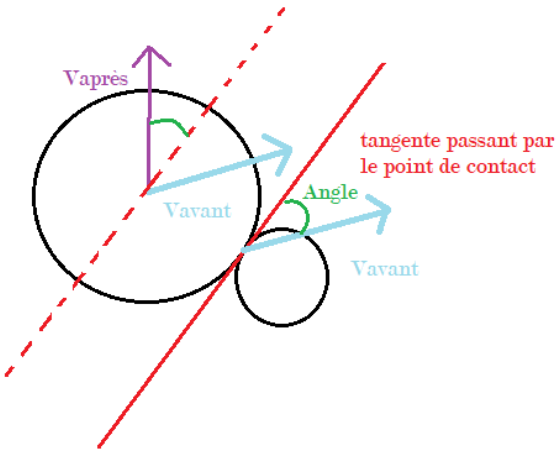

FIGURE 3 – Schéma du modèle physique du choc

Avec ces considérations, quelques calculs et quelques analyses géométriques nous donnent les valeurs du vecteurs vitesses du corps i juste après le choc.

Ensuite il a fallu trouver une condition de contact. Nous avons pris tout simplement pris la suivante :

$$u_{ij} \leq R = r_1 + r_2$$

avec u_{ij}, norme du vecteur ayant pour direction le centre du corps i vers le centre du corps j et de sens i vers j, r_1, rayon du corps i (resp. j).

4.2 Interface graphique

4.2.1 Zoom avec la molette

L'évolution des corps dans l'univers soumis à leur champ gravitationnel a pour effet de les faire se rapprocher ou s'éloigner, si la fenêtre n'est pas assez grande, les corps sortent du cadre. On fixe donc un rapport d'échelle entre le monde et la caméra ainsi par simple changement d'échelle on peut zoomer. On utilise donc la molette pour faire varier cette échelle.

```
if screen.get_wheel_up():
    screen.camera.scale *= 1.1
elif screen.get_wheel_down():
    screen.camera.scale *= 0.9
```

4.2.2 Autres fioritures

Le module `Pygame` permet de programmer de nombreuses fonctionnalités pour l'utilisateur. Nous avons pensé à ajouter un contrôle de la fenêtre avec les flèche du clavier ou l'ajout de corps en cliquant avec la souris mais nous n'avons pas eu le temps de nous atteler à ce travail.

4.3 Résultats

Après avoir implémenté tout cela, nous avons fait quelques observations, la première est que, pour certaines simulations, lorsque deux corps ce choquent, ce qu'il se passe ne respecte pas le sentiment physique (les deux corps partent dans la même direction ou encore restent collés...). De plus, lors de la simulation avec $N > 2$ corps, ces phénomènes ont plus de chances de se produire, et, dans certains cas, ils se produisent...

En voulant corriger ce problème, nous avons remarqué que cela venait de la condition de contact, en effet, pour respecter au maximum cette condition, il faut, et connaître la norme du vecteur les séparant (ceci est possible connaissant les coordonnées des centres d'inertie des corps) et connaître le rayon de ces corps. Le problème est que l'échelle des rayons et celle des coordonnées sont différentes. Nous n'avons pas eu le temps de corriger à la perfection ce problème. Pour compenser, nous avons vu empiriquement que rajouter un facteur 50 à la norme du vecteur résolvait beaucoup de nos problèmes, néanmoins, ce n'est pas suffisant pour une version finale.

Pour améliorer notre projet, il faut donc, comme nous l'avons vu ci dessus, trouver cette bonne échelle commune pour avoir la condition respectée comme il se doit.