N-Body Problem

Hugo MANGNAN
Antoine JOURDAN
Kilian CAILLOT
Paul GOSSE

Mai 2020

UNIVERSITÉ
CAEN
NORMANDIE

Table des matières

1 Introduction

Pour les étudiants en informatique, la programmation orientée objet apparaît comme un passage obligatoire. Cette année l'unité d'enseignement "Conception Logicielle" nous a proposé différents projets dans le cadre des travaux personnels approfondis. Par groupe de 4 nous devions réaliser une application complète à partir d'un sujet de départ. Parmi la liste variée nous avons choisi le sujet intitulé : "Simulateur pour N corps" car il apparaissait comme un mélange intéressant entre Informatique et Physique/Mécanique. Son énoncé est le suivant :

"Le problème à N corps est un problème d'astronomie classique où plusieurs corps se déplacent dans l'espace en étant soumis à leur propre inertie et l'attraction des autres corps. L'équation différentielle qui modélise ce problème est en pratique inutilisable pour $N > 2$. Le but de ce projet est dans un premier de simuler un espace newtonien où N corps interagissent et visualiser cette simulation. Il s'agira ensuite d'améliorer ce simulateur avecdiverses propositions parmi les suivantes : instancier des chorégraphies à N corps, accélérer l'optimisation avec un découpage spatial récursif, intégrer un jeu de pilotage d'un corps au clavier, développer une IA pour optimiser les déplacements avec une trajectoire faible en énergie comme les orbites de transfert."

Ainsi le projet peut se diviser en plusieurs étapes : d'abord la compréhension théorique du phénomène mécanique de l'attraction entre les corps, puis son implémentation dans le langage informatique pour enfin aboutir à une visualisation/simulation claire et précise des différentes interactions entre les corps.

1.1 Théorie

Le problème à N corps traduit le phénomène physique d'attraction s'exerçant sur différents corps selon des conditions initiales données : masse, vitesse et position dans l'espace. Il implique généralement la résolution d'une équation différentielle dérivant de la formule de base de gravitation entre deux corps telle que :

$$\overrightarrow{F} = G \times \frac{m1 \times m2}{R^3} \times \overrightarrow{R} \tag{1}$$

- $\overrightarrow{F}$: vecteur Force
- G : constante gravitationnelle
- m1 : masse du corps 1
- m2 : masse du corps 2
- R : distance entre les deux corps
- $\overrightarrow{R}$: vecteur entre le corps attiré vers le corps attracteur

Pour faire évoluer les corps dans l'espace chacune des conditions initiales vont être modifiées en fonction des autres corps présents (on ne considère comme seule force exercée celle de la gravitation pour des raisons évidentes de simplification).

1.2 Implémentation algorithmique

Il existe de très nombreuses manières d'implémenter le problème à N-Corps en suivant différents algorithmes existants :

- La méthode Leapfrog : elle permet la résolution d'équations différentielles du second ordre par une méthode de récurrence.
- Les méthodes Runge-Kutta : elles vont notamment composer la méthode d'Euler.
- La méthode d'Euler : elle permet la résolution d'équations différentielles du premier ordre.

Nous avons choisi la méthode d'Euler qui peut s'expliciter de cette manière : une équation différentielle permet de suivre une évolution temporelle, le principe est alors d'évaluer une fonction à différents moments. Nous avons alors implémenté la méthode d'une manière similaire : nous avons défini un laps de temps (ΔT) ; tous les ΔT, la force gravitationnelle est calculée, la vitesse quant à elle est définie selon une moyenne entre la vitesse initiale et la vitesse après application de la force gravitationnelle.

2 Organisation du projet

2.1 Répartition des tâches

Il y a d'abord eu une période de réflexion afin de comprendre la théorie et les principes mathématiques impliqués dans la dynamique. Le problème étant très connu, nous avons pu étudier les différentes méthodes de résolution. Suite à cela nous nous sommes focalisés sur la méthode d'Euler et nous avons pu répartir le travail. Antoine s'est occupé de la gravité et l'implémentation de cette force, Kilian s'est concentré sur le menu et les scripts de compilation, Paul a conçu l'interface graphique et Hugo a réalisé les chorégraphies proposée. Le rapport et le diaporama ont étaient réalisés par l'entièreté du groupe en associant les explications de nos parties respectives.

2.2 Architecture du programme

On peut distinguer deux majeures parties dans notre projet :
- Modélisation
- Interface Graphique

Pour la modélisation on peut expliciter n classes primordiales :
- La classe Corps qui va instancier les différents corps et qui contient également les méthodes permettant la mise à jour des différentes conditions initiales, notamment le calcul de la force gravitationnelle.
- La classe Systeme qui va définir les interactions entre les différents corps.
- Les classes Point et Vecteur qui vont tout simplement permettre une manipulation des différents corps selon des coordonnées cartésiennes et des vecteurs multiples.

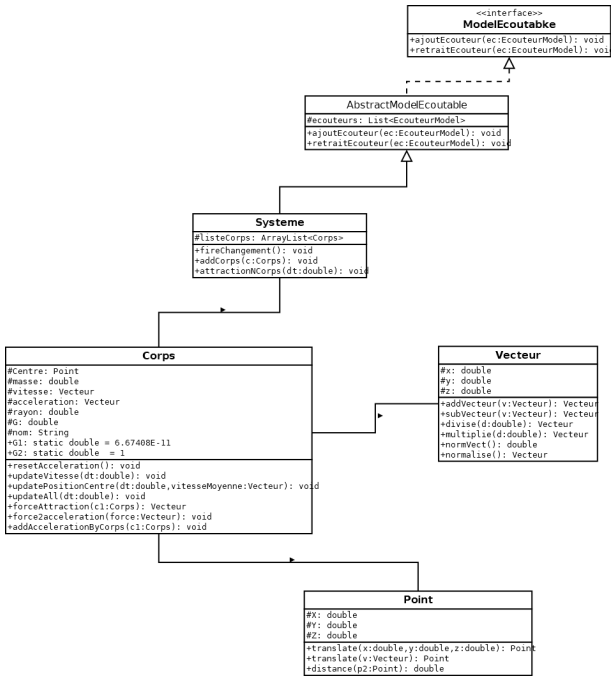


FIGURE 1 – Diagramme du modèle

3 Élements techniques

3.1 Implémentation mathématique

La méthode forceAttraction

Cette méthode permet de retourner un vecteur force qui va être appliqué à un corps. On obtient la direction de ce vecteur à l'aide des coordonnées cartésiennes des deux centres des corps. La fonction normalise() va alors rendre ce vecteur unitaire afin qu'on puisse l'utiliser dans la formule de l'attraction gravitationnelle de Newton.

3.2 Implémentation des données

Afin d'obtenir un rendu cohérent, les conditions initiales sont définis telles que :

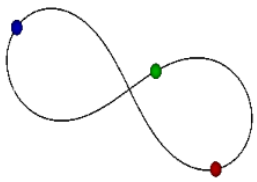
Corps	Coordonnées initiales (x,y,z)	Masse (kg)	Rayon (km)	Vecteur vitesse initiale (x,y,z)
Soleil	0 0 0	1.98855E+30	639934	0 0 0
Mercure	1.40191E+10 -6.64159E+10 -6.71319E+9	0.33011E+24	2440	3.78936E+ 1.25295E+01 -2.45039E+01
Venus	-9.90252E+10 4.14922E+10 6.28376E+9	4.8675E+24	6052	-1.36922E+01 -3.24633E+01 3.44688E-01
Terre	-1.44286E+11 -3.97378E+10 1.86460E+6	5.9723E+24	6371	7.42866E+00 -2.88200E+01 7.64891E-04
Mars	-2.68704E+10 -2.18308E+11 -3.91509E+9	0.64171E+24	3390	2.49637E+01 -8.76675E-01 -6.30821E-01
Jupiter	1.83152E+11 -7.54925E+11 -9.62242E+8	1898.19E+24	69991	1.25518E+01 3.70149E+00 -2.96220E-01
Saturne	6.36389E+11 -1.35785E+12 -1.72278E+9	568.34E+24	58232	8.22566E+00 4.07725E+00 -3.98732E-01
Uranus	4.38375E+12 -9.08578E+11 -8.23305E+10	102.413E+24	24622	-4.04915E+00 5.18937E+00 7.16753E-02
Neptune	4.38375E+12 -9.08578E+11 -8.23305E+10	86.813E+24	25362	1.07933E+00 5.35931E+00 -1.34968E-01

FIGURE 2 – Tableau des données

Application des chorégraphies

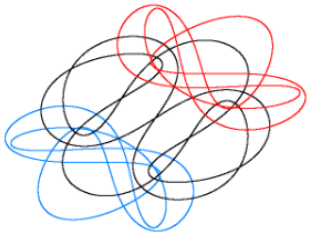
Les classes **Choregraphie1** et **Choregraphie2** vont permettre, à l'aide de paramètres initiaux de visualiser différentes chorégraphies entre plusieurs corps.

Chorégraphie en 8 :



Données :
positions :
(x1,y1) = (-0.97000436, 0.24308753),
(x2,y2) = (-x1, -y1), (x3,y3) = (0,0)
vélocités :
(vx1,vy1) = (vx2, vy2) = -(vx3, vy3)/2 ;
dont (vx3,vy3) = (0.93240737, 0.86473146)

Chorégraphie en moth :



Données :
positions :
(x1,y1) = (-1, 0),
(x2,y2) = (-x1, -y1), (x3,y3) = (0,0)
vélocités :
(vx1,vy1) = (vx2, vy2) = -(vx3, vy3)/2 ;
dont (vx3,vy3) = (0.464445, 0.396060)

3.3 Rendu Graphique

Menu

Le menu est composé d'un menu principal et d'un sous-menu. Le menu principal permet d'accéder à la simulation, la chorégraphie et à un troisième bouton permettant de se rendre vers le sous-menu.

Une Vbox est une boîte qui va contenir des éléments et les afficher à la verticale avec un espacement entre chaque élément si cela est souhaité.

Il est impossible de mettre plusieurs Vbox qui eux-même contiennent des boutons dans une même scène.. Donc pour éviter de créer une nouvelle scène qui ouvrirait une nouvelle fenêtre, nous avons dû effectuer une transition qui fait apparaître le sous-menu et fait disparaître le précédent menu. Ceci est effectué à l'aide d'un **getChildren().add(Le menu souhaité)** pour le faire apparaître dans la scène et un **getChildren().remove(Le précédent menu)** pour l'enlever de la scène et ainsi n'avoir qu'un seul VBox dans la scène

Il est possible de régler la vitesse de la transition et de voir le précédent menu disparaître dans une durée souhaitée et le menu souhaité apparaître aussi avec la durée souhaitée. Nous avons aussi ajouté un offset qui permet d'ajouter une vitesse à laquelle le menu va apparaître une fois que son temps de transition est terminé.

La classe Menu3 est composée de plusieurs fonctions :

— La fonction **start** qui crée et lance une scène afin d'obtenir une fenêtre.
— La fonction **GameMenu** qui va créer concrètement les menus, avec les Vbox,les transitions entre chaque menu, les boutons et les textes qu'ils contiennent. Elle va également instancier les évènements permettant de lancer la simulation et les chorégraphies.

— La fonction **MenuButton** qui génère des boutons avec les mêmes caractéristiques (taille, couleur, forme, évènements) ; les évènements sont par exemple un changement de couleur du bouton au passage de la souris sur le bouton par l'utilisateur.
— La fonction **main** qui va lancer tout ce que la classe contient.

Interface Graphique

L'interface graphique est principalement composée de trois grandes classes :
— La classe **PanelSystemView** est composée de fonctions qui permettent le contrôle de l'application pour zoomer ou déplacer notre point de vue (**MouseEvent et MouseWheelEvent**) la fonction **paintComponent** permet d'afficher les éléments de cette interface comme le background, les corps, leurs couleurs, la légende. Enfin, la fonction **initRatioDiminution** calcule l'espacement sur les corps les plus éloignés par rapport à la fenêtre pour y appliquer un zoom automatique à l'ouverture de la simulation.
— La classe **PanelInteractions** correpond aux champs à remplir pour ajouter un corps à la simulation et aux boutons d'interaction pour accélérer ou ralentir la simulation.
— La classe **FrameSystemGui** permet d'appeler les fonctions pour créer la vue et les interactions.

4 Manuel

L'application possède différentes fonctionnalités : elle permet de simuler le système solaire, d'y ajouter des corps, de visualiser une chorégraphie (et d'y ajouter des corps également), les boutons Accélérer 20% et Ralentir 20% vont assez intuitivement agir sur la vitesse de tous les corps ; dans le menu il sera également possible d'accéder au menu des planètes comportant des informations diverses.
Afin de pouvoir lancer l'application il suffit d'exécuter le .jar fourni. Lors de la fermeture de la simulation, il est nécessaire de relancer l'application si on souhaite en démarrer une nouvelle.
Si vous souhaitez ajouter un corps, il faut remplir les différents champs proposés :
— **Nom** qui va donner un nom à votre corps.
— **Centre** qui va définir les coordonnées cartésiennes du centre gravitationnel de votre corps.
— **Masse** qui va définir la masse de votre corps.
— **Vitesse** qui va définir le vecteur vitesse initiale de votre corps.
— **Chore** (y/n) permet de préciser si la simulation lancée est une chorégraphie ou non.

Exemple de corps convenables pour un système solaire (pas une chorégraphie) :
— **Nom** : Corps1
— **Centre** : entre 1E10 et 1E13
— **Masse** : proche de 0 (si on veut limiter l'influence du corps sur les autres) ou entre 1E23 et 1E29 (pour simuler un vrai corps)
— **Vitesse** : entre 1E3 et 1E4
— **Chore** : n

5 Conclusion

5.1 Objectifs atteints ?

Au cours de la réalisation de ce projet nous avons rencontré quelques difficultés notamment au niveau de l'implémentation du mouvement dynamique des corps par exemple. Bien que la méthode choisi pour modéliser le problème ne soit pas la plus rapide, elle reste néanmoins plus accessible et fonctionnelle. L'un de nos regrets reste l'absence de trajectoire qui aurait pu rendre plus clair le projet mais nous n'avons malheureusement pas réussi à ajouter cette fonctionnalité.

5.2 Pistes d'améliorations

Ce type de résolution informatique du problème à N-Corps est très commune et on peut trouver un nombre incalculable de projets similaires. Ainsi il est évident que de nombreuses personnes bien plus compétentes s'y sont affairés et ont réalisé une application plus précise, plus rapide, plus malléable. L'implémentation et la résolution d'équations différentielles peut se faire avec des méthodes plus rapides et même réversibles (cf. Leapfrog). La fermeture de la simulation entraîne la fermeture de l'application ce qui est problématique, tout comme l'absence de trajectoire d'ailleurs. Il pourrait être également possible d'ajouter les améliorations proposées dans la liste des sujets : accélérer l'optimisation avec un découpage spatial récursif, intégrer un jeu de pilotage d'un corps au clavier, développer une IA pour optimiser les déplacements avec une trajectoire faible en énergie comme les orbites de transfert...

Références

[1] Jet Propulsion Laboratory HORIZONS Web-Interface

[2] A collection of animations, sites, pictures, a few papers ... concerning the classical N-body problem

[3] MOTH I Choregraphy

[4] FIGURE 8 Choregraphy

[5] Exploring N-Body Algorithms

[6] Problème à N corps

[7] Méthodes de Runge-Kutta