Rapport Projet Programmation

Samuel Toplis / Alex Peter

May 2025

1 État du logiciel

Tout d'abord la fonction main du projet fonctionne, le fichier créé un GIF qui représente la simulation du problème à N corps, dans sa configuration initiale avec un trou noir et des planètes en orbite. On peut choisir la simulation Euler ou Verlet, bien que Verlet soit bien plus précis. Enfin s'il y a un problème pour make l'exécutable c' est UNIQUEMENT à cause du bêta test des fonctions dans simulation.c, n'hésitez pas à le commenter pour passer outre les erreurs.

Les fichiers de simulation simple tel que Euler et Verlet peuvent être observés dans simulation.c et fonctionnent pour tout nombre d'astres. On peut changer les paramètres de la fonction bigbang pour créer des configurations différentes.

Dans le fichier gif.c, nous créons un GIF qui apparait dans la racine du projet. Il à une taille que nous avons testé expérimentalement à 512 mo. Nous sommes donc limité à 600 images de taille 256 ko. Cela fait 600 frames de simulations.

Concernant les fonctions Quadtree et Runge-Kutta, Runge-Kutta a été écrit à l'ordre 2 dans simulation.c. On observe donc une simulation très proche de la simulation d'euler ce qui est très peux précis pour une simulation de type Runge-Kutta. Concernant Quadtree, nous sommes en présence d'une erreur d'allocation de l'espace. Les éléments récursifs créent des problème que l'on détecte avec Valgrind.

L'organisation des fichier a été refaite afin d'avoir 4 fichiers de code, le main, les simulations, les différents calculs de forces ainsi que le code de génération de gif. Les includes ont été rédigés parallèlement aux fichiers src.

Finalement un dossier bin et un dossier obj ont été ajoutés afin de pouvoir créer des GIF directement depuis la console.

2 Justifications des structures de données

Nous avons utilisé plusieurs structures de données afin de mieux représenter les éléments utilisés dans le code. Tout d'abord nous avons utilisé une structure de vecteur afin de mieux représenter les différentes distances dans un espace de dimension 2. Pour cela nous avons créé une structure vect composé d'un réel long x et d'un réel long, ainsi que plusieurs fonctions usuelles de l'algèbre lineaire qui sont referencées dans vect.h. Nous avons ensuite voulu représenter les différents corps du problème à N corps afin de pouvoir les manipuler à travers différentes fonctions.

Pour cela nous avons créé la structure corps composé d'un vecteur représentant la position, d'un vecteur représentant la vitesse et d'un réel long représentant la masse du corps.

Nous avons également créer un typedef de pixel car le but est de simuler à chaque frame, la prochaine position de chaque astre, qui seront alors représentées par des carrés blancs dans l'image d'une frame. Bien sûr, la coordonnée de position (double) est arrondi afin de mettre à jour la bonne adresse du pixel à ses coordonnées entières (integer). Le paramètre vitesse est alors utilisé uniquement pour les calculs des simulations. Que ce soit Euler, Verlet ou autres... On recalcule les forces, positions et les vitesses à chaque frame et on fait créer l'image à chaque fin de boucle dans simulation.c.

Dans ce contexte, la structure de pixel est utilisée dans toute la section du code image afin d'allouer la mémoire correctement pour créer les images dans gif.c. Nous avons donc également créé une structure d'image comprenant le nombre de lignes, le nombre de colonnes, le nombre de canaux de couleurs et un tableau des lignes de pixels afin de créer des images.

Finalement nous avons créé une structure quadtree qui n'est pas utilisé mais nous l'avons gardé car nous étions en train d'enlever les bugs du code quadtree et jusqu'à que cela soit terminé nous avons souhaité garder la structure afin de pouvoir faire machine arrière. Cette structure est composée d'un niveau, d'une masse et d'une longueur (en long réels), d'un vecteur pour le barycentre et d'une liste de corps qui inclut tout les corps du quadtree.

3 Méthodologie des tests

À la manière classique : on connaît l'ordre chronologique d'exécution et on incorpore des printf dans le code afin qu'on puisse savoir où on en est pendant l'exécution, en effet dès qu'il y a un problème, on sait où il se trouve en fonction des print affichés sur la console et ainsi de suite.

Afin d'accélérer le processus, on s'est permis de ne faire qu'un seul fichier test qui demande alors le test spécifique qu'on veut réaliser (option 0 = Euler, Option 1 = Verlet...) ainsi que d'autres paramètres comme le nombre d'astres et le temps de simulation. Ce fut très pratique car les commandes de création de fichier test sont toujours les mêmes, et il n'y a pas eu de confusions.

Finalement pour tester le fonctions de simulations en elle même, nous avons fait un problème à 1 corps autour d'un trou noir, en effet, cela devrait donner si il n'y a pas de vitesse initiale, un trajectoire circulaire ce qui n'était pas immédiatement le cas mais à été réparé.

4 Analyse des performances et mémoire

1 Seconde de simulation = 100 frames (images) d'après nos tests on est limité à environ 600 frames par simulation. On a en moyenne 60 lignes de code par programme. Les images générées font environ 256 ko car la taille choisie est de 512x512

pixels avec un seul channel de couleur.

Pas de problème avec le valgrind à priori lors de l'exécution de testsimulation.exe. Ce qui prend le plus de temps, c'est la création du GIF une fois que toutes les images ont été générées. Lors de la création, on peut observer l'ordre de grandeur des forces et distances mises en jeu à chaque frame par les print.

5 Outils de développement utilisés

Tout d'abord, nous avons rédiger l'entièreté du code sur VS code car il s'agit de l'éditeur de code en C que nous connaissons le mieux. Par ailleurs afin de pouvoir travailler à plusieurs, nous avons eu une utilisation importante de git, cela nous permettait de transmettre les avancées du projet en faisant attention à ne pas créer des nouvelles branches conflictuelles.

Concernant l'utilisation de Valgrind, lorsque nous avons compris comment l'utiliser, nous avions déjà réglé la majorité de nos problèmes de mémoires donc cela n'a été que très peu utile. Nous avons néanmoins eu quelques problèmes de "core segmentation" que nous avons pu régler à l'aide de Valgrind.

La compilation à été réalisé à l'aide d'un makefile fourni par le sujet puis légèrement modifié pour ne pas qu'il y ai de conflit avec notre code, notamment au niveau de la création du GIF. Nous avons make de façons à ce qu'il n'y ai plus d'erreur mais des avertissements subsistes du à certains codes qui ne sont pas encore complets donc qui envoie vers une fonction non utilisé.

6 Organisation

Alex Peter c'est occupé des algorithmes de simulation initiaux notamment l'algorithme d'euler et de verlet. Il s'est également occupé de la majeur partie de la réparation de bug en particulier sur les différentes simulations et sur la mise en commun des différentes parties du code ce qui a été la plus grande partie de la réparation.

Samuel Toplis c'est occupé des algorithmes concernant les différentes structures notamment le main, les conditions initiales ainsi que la majorité des fonctions de création des images. Il s'est également occupé de rechercher les fonctions Quadtree et Runge Kutta qui n'ont malheureusement pas pu être inclue dans la code principale dans le temps impartit.

7 Notes proposées

Nous jugeons que le travail à été répartit équitablement entre les deux membres du binôme. Il faut noter que nous avons souvent travailler à deux sur un même ordinateur pour la réparation de bug ou bien la théorisation des fonctions plus avancées comme Runge Kutta ou Quadtree. Nous pensons donc qu'une répartition juste de notre traval est de 10 - 10.

8 Conclusion

Nous avons compris l'intérêt du C au cours du projet, qui nous permet d'avoir une approche un peu plus native au fonctionnement de l'ordinateur (seulement en surface bien sûr, le C reste un language mid level contrairement à l'assembly). Toutefois, nous nous sommes heurtés à des difficultés sur runge kutta et quadtree car il était difficile de les incorporer dans notre version minimale.