Simulateur de Galaxie

Algorithme de Barnes-Hut

Ingénieur Informatique I - Algorithmique II

MARTI Emilie & SOUSTRE Ludovic

Introduction

Le but de ce projet est de réaliser en binôme un simulateur de Galaxie en C, en utilisant la librairie graphique MLV.

Deux versions seront présentées ; la première, où les forces appliquées sur chaque corps de la galaxie seront calculées en brute force, et la deuxième, qui utilisera un arbre quaternaire (ou quadtree), pour calculer de façon efficace et optimisée les forces sur chaque corps.

Galaxy_A/ est le dossier contenant la première version et Galaxy/ est le dossier contenant la version avec l'arbre quaternaire.

Le but de ce projet est la prise en main des arbres quaternaires et de se sensibiliser sur l'importance des algorithmes optimisés lors de la gestion de grands univers avec beaucoup d'éléments qui interagissent (ou pas) entre eux, ainsi que l'utilisation de la récursivité au service de ce genre d'algorithmes.

Manuel Utilisateur

Afin d'organiser l'espace de travail, plusieurs dossiers ont été créés pour bien séparer tous les types de fichiers créés. Pour les deux versions, le plan est le suivant :

- ➔ root (nommé Galaxy_A et Galaxy respectivement)
  - ◆ bin/ - Exécutable
  - ◆ data/ - Fichiers récupérés par le programme
  - ◆ doc/ - Documentation
  - ◆ headers/ - Fichiers header .h
  - ◆ log/ - Logs de debug avec gdb
  - ◆ obj/ - Fichiers objet .obj
  - ◆ src/ - Fichiers source .c

Compilation

Un makefile prévu à cet effet a été créé pour les deux versions, A et B respectivement. Pour compiler, il faut se situer au dossier racine de la version choisie et faire la commande suivante :

```
>> make
```

Le makefile va récupérer les fichiers sources présents dans src/ et va créer les fichiers .o nécessaires au linkage, qui vont être placés dans obj/ puis va créer l'exécutable qui va se trouver dans le dossier bin/ .

Exécution

Pour exécuter le programme, il faut se mettre dans le dossier source et faire la commande suivante :

```
>> ./bin/galaxy data/path_to_file
```

path_to_file peut être l'une des trois options suivantes :

- ➔ ex1.txt
- ➔ ex2.txt
- ➔ ex3.txt

Un quatrième fichier, ex0.txt, est présent et contient quelques corps. Il ne s'agit pas de fichier final quelconque, mais d'une version de débug pour tester les fonctionnalités du projet à partir du début avec un très petit univers (surtout lors de l'insertion).

Pour l'exécutable de la version B du projet, un menu permet de décider à l'utilisateur s'il veut voir la galaxie simulée en temps réel ou s'il veut voir le quadtree et les statistiques du projet en temps réel aussi. Lorsque l'une des options est cliquée, pour pouvoir voir l'autre il faut exécuter à nouveau le binaire.

État du projet

Les deux versions sont terminées et les fuites mémoires sont normalement gérées (sans compter les fuites liées à MLV). Il n'y a pas de bugs significatifs à relever, les trois fichiers donnés dans le projet marchent correctement.

Une structure supplémentaire, Galaxy, a été utilisée. Elle contient des informations de la galaxie et les structures qui sauvegardent les corps (un tableau pour la version A, un quadtree pour la version B). Ceci permet d'avoir facilement accès à toutes les informations de la galaxie à partir d'une simple structure.

L'amélioration principale proposée est la visualisation en temps réel de l'arbre quaternaire formé à partir de l'univers, dans la version B du projet, ainsi que les statistiques à jour de cet arbre en même temps que l'affichage de l'arbre.

Déroulement du projet

Le projet a été réalisé pour la quasi-totalité pendant la semaine de révisions d'avril, avec les deux camarades du binôme présents à tout moment pendant l'avancement du projet.

Le projet est géré sous git depuis le début et les deux composants du binôme y sont en tant que collaborateurs. Le lien du git ci-dessous :

https://github.com/hyliancloud/INFO_Algo_Galaxy

Les modifications principales ont été faites sur la branche Master à tout moment, et les tests ont été faits dans d'autres branches.

La toute première phase consistait à faire toute l'organisation du projet, pour avoir dès le début une base ordonnée et prête à l'emploi. Nous avons commencé par créer des dossiers pour organiser le dossier, puis le makefile, présent dans root, a été le premier fichier codé.

Dans le dossier src/, les fichiers suivants ont été créés :

- ➔ galaxy.c
  - ◆ Fichier du main.
- ➔ galaxy_manager.c
  - ◆ Fonctions concernant les structures de la galaxie.
- ➔ physics.c
  - ◆ Fonctions qui gèrent la physique.
- ➔ graphic.c
  - ◆ Fonctions qui gèrent l'affichage graphique.

Le binôme s'est mit d'accord sur des conventions de nommage pour le projet entier, ce qui a pour but d'avoir un code cohérent. Les conventions de nommage sont les suivantes :

- ➔ Variables
  - ◆ nomVariable
- ➔ Variables globales
  - ◆ _nomVariableGlobale
- ➔ Macros dans #define
  - ◆ NOM_MACRO
- ➔ Fonctions
  - ◆ nom_fonction()

La démo de M. Thapper avec l'affichage et l'interaction des deux corps a été décomposée entre ces divers fichiers pour bien établir l'organisation du projet et travailler à partir d'ici.

Il a fallu ensuite prendre en main le projet et bien le comprendre. La démo avec deux corps donnée par M. Thapper nous a bien aidé à voir comment marche la physique des forces et, pour la partie A, il fallait faire une simple généralisation, en brute force, de cette démo.

Cette partie a été faite entièrement en binôme, sans trop de soucis rencontrés. La décision de ne pas encore se séparer en différentes tâches était consciente et motivée par le fait qu'on voulait être sur la même longueur d'onde sur l'interprétation du sujet.

La consigne était claire et simple ; la partie la plus compliquée était l'écriture de la fonction galaxy_reader(), dans galaxy_manager.c, qui est une fonction qui va parser un fichier qui contient toutes les informations nécessaires à la création de tous les corps qui composeront la galaxie.

Aussi, lors de l'affichage, il a fallu faire des tests et réfléchir sur le choix des valeurs d'affichage pour que la galaxie soit bien affichée entièrement dans la fenêtre MLV.

Néanmoins cette partie à été terminée en l'espace de une ou deux demi-journées sans trop de problèmes. La gestion de la mémoire a été prise en compte sérieusement depuis le début, donc les fuites ont été minimisées et le code a été amélioré à posteriori si valgrind témoignait de fuites venant de nos fonctions.

Pour la phase II, il y a eu un temps de recherche sur le fonctionnement et l'utilisation des arbres quaternaires, qu'on n'avait pas vu en cours, on devait donc se renseigner en amont. On a décidé de commencer à répartir les tâches à ce moment là pour optimiser le temps de travail sur le projet, qui devenait corsé.

Au début, on a créé le fichier supplémentaire suivant, toujours dans l'optique d'organiser le code pour faciliter la compréhension du code et le débug :

- ➔ quadtree.c
  - ◆ Fonctions de gestion de l'arbre quaternaire

La plus grosse difficulté rencontrée a été à ce stade là : comprendre le fonctionnement du quadtree dans ce contexte et faire la fonction insert_body qui va créer le quadtree en lui-même.

Nous avons écrit les structures et ses fonctions d'initialisation et de libération de mémoire.

En voulant commencer par la fonction qui construit le quadtree, on a bloqué. On a décidé de séparer nos tâches comme suit :

- ➔ Ludovic décomposait la consigne sur le quadtree et faisait une deuxième phase de recherche pour bien comprendre et mettre sur la table les éléments nécessaires pour la construction du quadtree.

- ➔ Emilie se chargeait de coder la première partie du calcul de la force gravitationnelle.

On a ensuite commencé à coder la fonction insert_body(), qui se charge de l'insertion des corps dans le quadtree (ainsi que la construction de ce dernier au fur et à mesure). Cette phase a été, de loins, la plus dure et longue du projet.

En suivant la consigne, on a réussi à coder ensemble la base de insert_body, mais on a rencontré des bugs bloquants qui nous ont pris bien deux ou trois demi-journées à debugger. Cette phase a été répartie également :

- ➔ Emilie se chargeait de poursuivre le code des fonctionnalités principales :
  - ◆ Mise en place de la physique (calcul de forces, position, etc ...)
  - ◆ Débug de insert_body() avec gdb
  - ◆ Gestion de la mémoire

- ➔ Ludovic a proposé de faire de nouvelles fonctionnalités pour le débug :
  - ◆ Tests unitaires avec des univers plus petits
  - ◆ Affichage de l'arbre en .dot
  - ◆ Création de l'affichage du quadtree

Les fonctions de Ludovic ont été très utiles au débug et à la réalisation de la fonction insert_body(). Ses fonctions ont permis de visualiser très clairement l'état de l'arbre et de mieux comprendre ses bug de comportements quand les segfault avaient été réglés avec l'aide de gdb.

L'affichage du quadtree en MLV est tellement propre qu'on a décidé de l'ajouter à la version finale en tant que amélioration.

Lorsqu'on avait fini, il restait un bug non-bloquant, mais qui provoquait une mauvaise gestion de la mise à jour des forces sur un corps. En effet, une condition ne se vérifiait jamais et on se retrouvait dans une situation où le calcul des force se faisait en brute force, malgré avoir géré correctement le quadtree.

Après consultation avec M. Thapper, on a découvert que le bug était à cause d'un simple manque de parenthèses dans un calcul sur une ligne. Le projet était à présent fini, il ne restait que de l'intégration des fonctions de débug et la rédaction du compte rendu.

Lors de l'intégration, on a rajouté un menu pour laisser à l'utilisateur le choix de voir la galaxie ou de voir le quadtree.

Comparaison des versions

Pour mesurer la performance des programmes dans les même conditions pour les deux versions, nous avons fait le calcul du nombre d'opérations par seconde puis nous les avons affiché dans le terminal.

Pour la comparaison, afin de s'affranchir du biais que représente le calcul graphique pour afficher les statistiques sur la version B (non présent sur la version A), on a enlevé l'affichage des statistiques. Chaque valeur d'opérations par seconde pour un nombre X de corps est une moyenne du nombre d'opérations par seconde (OpS).

Pour chaque entrée du tableau ci-dessous, le nombre de corps est augmenté d'un facteur d'environ 1.5 par rapport à l'entrée précédente. Lorsqu'on a dépassé les 4000, on a rajouté l'entrée 10001 qui correspond à l'exemple de données 3.

Table des résultats

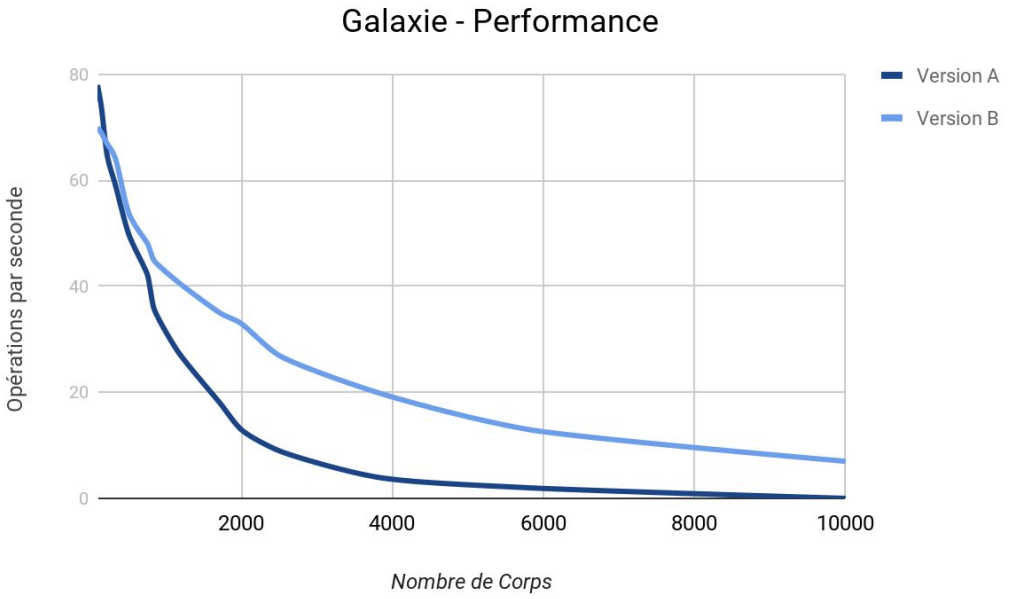
Nombre de Corps	Version A (OpS)	Version B (OpS)	Données
100	78	70	
150	74	69	
225	65	67	
340	59	64	
510	50	54	
765	42	48	
843	36	45	ex1.txt
1150	28	41	
1720	18	35	
2001	13	33	ex2.txt
2500	9	27	
3800	4	20	
5800	2	13	
10001	0	7	ex3.txt

Interprétation des résultats

Après avoir effectué les mesures sur les deux version, on peut conclure que la version A est légèrement meilleure en termes de performance pour un petit nombre de corps, mais dès qu'il y a plus de 200 étoiles la structure en quadtree pour les calculs dépasse largement la version A en performance.

La raison pour laquelle la version B est moins bonne pour un petit nombre d'étoile est la génération du quadtree à chaque opération qui est coûteuse et donc contre productive pour une petite quantité de calcus.

Représentation graphique



Constantes de complexité :

- ➔ Version A
  - ◆ Les opérations se font en $O(n^2)$ à chaque tour de boucle, car le calcul des forces se fait avec une double boucle for qui, pour chaque corps, parcourt le tableau une deuxième fois.

- ➔ Version B
  - ◆ Les opérations se font en O(n log (n)), car les forces sont calculées qu'avec les corps proches, les autres ne sont pas tenus en compte.

Conclusion

Ce projet s'est globalement très bien passé. Il s'agissait d'un projet dur, surtout lors de la gestion du quadtree, néanmoins on y est arrivés, et le résultat est réellement satisfaisant.

Le fait de devoir faire de vraies recherches et en avoir eu le temps a été vraiment enrichissant pour ce projet. Apparemment les moteurs de jeux utilisent ce genre de structures pour gérer la physique de façon optimisée, et on a découvert par la même occasion que cet algorithme a en fait un nom, l'algorithme de Barnes-Hut.

Nous avons vraiment boosté nos connaissances sur les logiciels de debug, en utilisant gdb et valgrind au plus pour débugger efficacement.

La communication du binôme a été excellente, surtout grâce au respect mutuel et à la motivation des deux parties pour réussir et avoir un résultat propre, ainsi que pour l'envie d'apprendre. La disponibilité quasi totale des deux camarades et le fait de pouvoir y consacrer plusieurs heures d'affilé lors de la semaine de révisions a été un grand atout pour la bonne réalisation de ce projet.

En conclusion, le projet était intéressant et très enrichissant.