Simulation du problème à N particules

Clément Blaudeau, Elliott Benisty

1 Introduction

La simulation de l'évolution d'un système de N particules soumis à l'interaction gravitationnelle est un problème classique de simulation numérique. En effet, le seul moyen de traduire l'évolution d'un système de plus de trois particules est de réaliser une simulation numérique fondée sur une discrétisation du temps. Nous utilisons ici le fait qu'à chaque instant les particules doivent effectuer un calcul indépendant des autres pour caractériser leur évolution subséquente.

2 Principe de fonctionnement

Nous discrétisons le temps en unités que l'on appelle *frame*, correspondant selon un paramètre éditable directement au début du code à un certain nombre d'unités de temps. Nous utilisons un *thread* par particule, chargé de calculer la somme des forces qui lui sont appliquées afin d'en déduire sa position et sa vitesse à la prochaine frame. La nature du problème fait que tous les threads doivent travailler en même temps sur une même frame, et ne peuvent se distancier.

C'est pour cela que l'on utilise la classe *lock* ayant la particularité de libérer par vague les *threads* appelant la méthode *attempt()*. Chaque *thread* incrémente un compteur puis attend. Le *thread* faisant passer le compteur à N (nombre de particules) n'attend pas et libère tous les autres *threads*. Ce mécanisme permet d'assurer la synchronisation sur une frame de tous les *threads*.

Ainsi, on remplit *frame* par *frame* le tampon et la variable *buffer_frame* signifie sur quel *frame* les *threads* sont actuellement en train de travailler. En concurrence avec ce curseur, on trouve *view_frame* qui traduit quelle *frame* est en train d'être affichée. L'enjeu est alors de contrôler les valeurs relatives de ces deux variables afin d'éviter les collisions. Autrement dit, on ne veut pas afficher une *frame* avant qu'elle ne soit calculée, ce qui aurait pour effet à cause de la structure cyclique de *buffer* d'afficher une *frame* depuis longtemps passée. On se prémunit de cela en assurant une avance d'au moins une moitié de *buffer* de *buffer_frame* par rapport à *view_frame*.

Un autre cas de collision survient lorsque les *frames* sont calculées trop vite par rapport à l'affichage. Pour cela, on arrête le calcul lorsque les *threads* prennent une avance d'environ une taille de *buffer*. Lorsque N est petit, on constate alors que les *threads* passent le plus clair de leur temps à attendre l'affichage. Pour remédier à cela, on modifie dynamiquement, lorsque la fonctionnalité *optimize_speed*

est activée, la vitesse de lecture du buffer par l'affichage. En effet, celui-ci est alors capable de sauter l'affichage de plus ou moins de *frames*. On obtient ainsi un affichage fluide d'une simulation rapide mais très précise. Bien entendu, il est possible de désactiver cette fonctionnalité et de paramétrer soi même la vitesse de simulation et le pas de discrétisation du temps.

La dernière difficulté qui apparaît est de pouvoir quitter le programme sans que quelques *threads* continuent à tourner. Pour cela, une méthode *flush* du verrou permet de libérer tous les *threads* ce qui leur permet de comprendre immédiatement qu'il est temps de terminer.

3 Conclusion

En définitive, ce programme exploite la puissance du parallélisme afin d'accélérer significativement le calcul des interactions d'un grand nombre de particules. Il serait désormais intéressant d'ajouter l'interaction électromagnétique afin de limiter l'impact catastrophique des collisions directes enter particules (l'interaction diverge à très courte distance). C'est une façon d'ajouter du "volume" aux particules. Par ailleurs, on constate qu'il serait intéressant de faire varier dynamiquement le pas de temps en fonction des particules : une particule isolée et loin de toute autre particule est relativement stable dans sa trajectoire, tandis que deux particules très rapprochées exhibent une discontinuité de leur paramètres lorsque le pas de discrétisation du temps est trop important.