8INF935 - Mathématiques et physique pour le jeu vidéo

Phase 1 - Journal de bord

Jules RAMOS - RAMJ27020100 - Mikhael ChOURA - CHOM23060100 - Gaëlle THIBAUDAT - THIG24539900 - Béatrice GARCIA CEGARRA - GARB19510105

Table des matières

Introduction

L'objectif de cette première phase de développement était d'obtenir un moteur élémentaire de gestion de particules. Pour cela, plusieurs étapes ont été nécessaires et plusieurs choix de développement ont été faits afin de préparer les prochaines étapes du projet.

I. Le stockage des particules

Afin de stocker et d'itérer sur toutes les particules pour les calculs de position ainsi que l'affichage, il fallait une structure pour les contenir. Plusieurs options se présentaient alors, mais la plupart des structures classiques telles que des arrays avaient pour désavantage une gestion complexe de l'allocation de mémoire. C'est pourquoi la solution adoptée a été celle des vecteurs de la Standard Template Library (ou STL) du C++. En effet, ces structures semblables à des listes chaînées sont implémentées de base en C++ et viennent avec des méthodes gérant automatiquement l'allocation dynamique de mémoire. Ainsi, l'ajout et la suppression de particules sont faits avec les fonctions *push_back()* et *erase()* de la STL.

Cela permet également l'utilisation d'une boucle foreach pour l'affichage des particules ce qui facilite cette étape.

II. Implémentation des intégrateurs (position des particules)

Il est rendu possible à l'utilisateur, dans le projet, de changer de méthode de calcul de la position des particules entre l'intégrateur d'Euler et celui de Verlet. Si les équations de calcul de la vélocité (dans le cas d'Euler) et de la position sont données explicitement dans le cours. Plusieurs problématiques et choix de conception ont émergé lors de l'implémentation :

Gestion des variables : La position, la vitesse et la masse d'une particule sont données en argument de la classe puisque totalement inhérents à cette dernière. Mais la gravité, la durée d'une frame et le % de vélocité sont considérés comme des facteurs globaux du monde où évoluent les particules et donc définis à l'extérieur de la classe.

Position p-1 (Verlet) : La position antérieure à la position initiale est approximée à l'aide de la méthode d'intégration d'Euler. L'expression de la position à juste été modifiée pour donner la position antérieure en fonction de la position actuelle.

Accélération fonction du temps (Verlet) : Pour l'intégrateur de Verlet, la position est calculée en fonction de l'accélération, soit des forces appliquées à la particule à chaque instant t, et non de la vitesse. A cette phase de développement du projet, l'accélération est, d'après la troisième loi de Newton, égale à la gravité. Pour cette phase, l'accélération est considérée constante et tout égale à g dans le calcul de la position par la méthode de Verlet.

Restauration des variables au changement de mode : La modification du mode de calcul de la position d'une particule en plein mouvement, pour des méthodes utilisant les mêmes variables mais des éléments de calcul différents a demandé une initialisation de valeurs à chaque changement, symbolisé par un booléen propre à la particule. L'inconvénient de ce choix est l'ajout d'une instruction, soit une structure conditionnelle, à chaque frame. En fonction de la position courante, sont ensuite calculés la vitesse antérieure, pour Euler, ou la position antérieure, pour Verlet.

III. Implémentation d'une trace des particules

Afin d'afficher la trace des positions des particules, un vector de la STL a également été utilisé. Afin de ne pas inutilement augmenter la consommation de mémoire du programme, deux décisions ont été prises :

- Les positions ne seraient enregistrées que toutes les 0,2s
- Seuls les vecteurs position des particules seraient enregistrés.

Pour obtenir le délai de 0,2s, un attribut float a été utilisé et est incrémenté en utilisant la fonction *ofGetLastFrameTime()* chaque itération de la fonction *update()*.

Le résultat obtenu avec cette méthode est visuellement satisfaisant et permet d'avoir un grand nombre de particules et leur trace sans causer de ralentissement du programme.

IV. Implémentation de la classe Input (récupération des inputs de l'utilisateur)

La classe Input est une classe singleton, son constructeur par défaut est privé et une méthode statique permet son instanciation. C'est un type de classe auquel nous n'étions pas habitués en C++, cependant des exemples de ce type de classe trouvés en ligne ont pu être adaptés à notre cas assez facilement.

Ainsi l'utilisateur peut modifier un vecteur grâce à l'input qu'il effectue. Dans notre cas, nous travaillons avec une longueur (norme) et deux angles (theta et phi). Les deux angles ont été définis de manière arbitraire, pour que, lorsque ces deux angles valent 0, le vecteur soit aligné avec l'axe x.

V. Développement des tests de la classe Vector et de la classe Input

Nous avons choisi de créer un projet exclusivement consacré à la compilation des tests des classes que nous avons implémenté, car ces classes utilisent des bibliothèques de OpenFramework, donc nous avions besoin que l'édition de lien soit correctement réalisé.

VI. Développement de la physique et des classes héritant de particule

Le plus dur dans cette partie fût de se rappeler de comment appliquer le polymorphisme sur la fonction de collision. Au départ, la liste de particule était stockée sans utiliser de pointeur, ce qui avait pour effet d'appeler toujours la méthode mère pour la détection de collision, peu importe le type de variable contenu dans la liste. Il a fallu transformer cette liste de particule en liste de pointeur de particule pour régler le problème.

Un bug de débordement de liste se produisait également après ce changement. En effet, comme la fonction de collision peut changer le nombre d'éléments dans la liste (par exemple avec FireBall Particule, qui fait spawn des particules), la liste était modifiée lors de son parcours, ce qui entraînait des erreurs. Pour régler ce bug, nous avons décidé de faire le parcours de la liste sur une copie de la liste d'origine. Ainsi, nous pouvons modifier la liste originale sans problème puisque le parcours se fait sur la copie.