Université Toulouse III
Paul Sabatier

Chef d'œuvre - M2IM

Simulation Physique Basée Sur Les Positions

Spécifications

Auteurs :
Fabien Boco,
Agathe Duyck,
Anselme François,
Dimitri Raguet

Superviseurs :
Charly Mourglia
Valentin Rousselet

4 décembre 2015

Résumé

Ce chef d'œuvre a pour but d'implémenter un moteur physique au sein du Radium Engine en suivant l'approche Position Based Dynamics développée par des chercheurs de chez NVidia.

Table des matières

Chapitre 1

Analyse fonctionnelle

Le projet est actuellement découpé en quatre "milestones". Chaque milestone représente un état d'avancement livrable du projet et doit être réalisée dans son ensemble (toutes les taches doivent être implémentées) pour être considérée. La première milestone est évidemment obligatoire et représente le minimum livrable. On considérera que les modules de chaque milestone dépendent de tous les modules de la milestone précédente.

1.1 Milestone 1 : Les Particules

1.1.1 Scène avec particules

La scène doit pouvoir contenir des particules. Une particule sera représentée par une position et un rayon. Pour créer la scène, il faut également qu'il soit possible d'ajouter manuellement des particules ou bien d'en charger à partir un modèle.

Dépendance : Radium Engine

1.1.2 Rendu

On doit pouvoir afficher une scène contenant des particules, représentées sous forme de petites sphères. Cette étape va permettre la prise en main du moteur de rendu par tous les membres de l'équipe.

Dépendance : Radium engine, Scène avec particules

NB : Une dépendance vis à vis du Radium Engine signifie que le moteur de rendu doit fonctionner correctement pour que nous puissions mettre en œuvre, tester et visualiser nos résultats. Cette dépendance ne dépend pas de nous mais elle existe bel et bien.

1.1.3 Solveur

Implémentation de la méthode de résolution des contraintes. Le solveur prend en entrée une collection de particules et les contraintes associées, et renvoie les nouvelles positions de ces particules, corrigées en fonction des contraintes.

Dépendance : Intégrateur

1.1.4 Intégrateur

L'Intégrateur prend la Scène en entrée et met à jour les positions et les vitesses de toutes les particules. On doit pouvoir appliquer des impulsions aux particules pour les faire se mouvoir, et également pouvoir ajouter des contraintes au système. L'intégrateur doit aussi comporter une étape d'amortissement qui réduit d'une proportion arbitraire toutes les vitesses à chaque pas de temps, pour que le système tende vers un système stable.

Dépendance : Scène avec particules

1.1.5 Contraintes basiques

— Contrainte de non-intersection entre 2 sphères
— Contrainte de non intersection avec un plan aligné avec les axes

Ces deux contraintes sont les plus simples à réaliser, la première permettra de tester et étalonner le solveur, et la deuxième permettra de garder les particules dans un volume limité.

Dépendance : Scène avec particules, Solveur

1.1.6 Test

On admettra une scène dans laquelle 1000 particules seront contenues dans une "boite" invisible. Il devra paraître visuellement évident que les contraintes sont respectées, à savoir que les particules ne se rentrent pas dedans et ne sortent pas de la boite.

1.2 Milestone 2 : Les objets rigides

1.2.1 Contraintes plus complexes

— Contrainte de non-intersection entre particule et triangle.
— Contrainte sur les degrés de liberté. Cette contrainte altère les 6 degrés de liberté d'une particule par rapport à une autre. Ces 6 degrés sont la translation sur les axes X, Y et Z, et les rotations autour des axes X, Y et Z.

Dépendances : Solveur

1.2.2 Scène avec triangles

La scène doit pouvoir contenir des triangles. Un triangle sera représenté par 3 références à des particules. Les triangles seront stockés dans une table de hachage spatiale. Il faut être capable de charger des maillages manifold dans la scène en tant qu'objet rigide. Un objet rigide est représenté par un maillage dont chaque arête implique une contrainte sur les degrés de libertés qui bloque les 6 degrés entre les deux particules qui représentent les 2 sommets de l'arête.

Dépendance : Radium Engine, Contrainte sur les degrés de liberté

1.2.3 Calcul d'intersection

Un algorithme de calcul d'intersection entre un segment et un triangle doit être implémenté.

Dépendances : Aucune

1.2.4 Collisions

Une étape de détection de collisions sera ajoutée à l'intégrateur, Cette étape vérifiera s'il y a collision entre chaque particule et triangle. Pour chaque collision détectée, une contrainte de non-intersection sera générée.

Dépendances : Scène avec triangles, Calcul d'intersection, Contrainte de non-intersection

1.2.5 réponses aux collision

Après l'exécution du solveur, on ajoutera à l'intégrateur une étape de mise à jour des vitesses qui pour chaque contrainte de non-intersection résolue, appliquera une impulsion à la particule concernée, en réponse à la collision.

Dépendances : Scène avec triangles, Collisions

1.2.6 Test

Une scène avec au moins deux maillages et 100 particules sera créée. sur le même modèle que le premier test on devra observer que les collisions ont bien lieu et que des déplacements dus à la réponse impulsionnelle de ces collisions se produisent.

1.3 Milestone 3 - Les Objets déformables

la scène peut contenir des objets déformables : tissus, objets gonflables, objets altérables (par destruction ou altération de la géométrie et topologie). C'est l'ajout de nouvelles contraintes permettra d'arriver à un tel résultat. La liste suivante n'est pas exhaustive et on peut imaginer n'implémenter qu'une seule de ces contraintes ou bien plus pour valider cette milestone.

— Contraintes d'objets mous
— Contrainte d'élasticité
— Contrainte de volume/pression
— Contrainte "pâte à modeler"

1.3.1 Test

Ici il faudra mettre en scène la contrainte que l'on veut observer. Par exemple pour les contraintes d'objet mou, on pourra fixer un maillage (en forme de long pavé) sur un mur invisible et observer sa déformation sous l'effet de la gravité.

1.4 Milestone 4 - Pour aller plus loin / Optimisation

1.4.1 Solveur sur GPU

Il est possible de paralléliser le solveur sur GPU. IMais il faudra Ré-implémenter les contraintes sur le GPU ainsi qu'utiliser un algorithme de coloration de graphes pour identifier les contraintes indépendantes afin de pouvoir les exécuter en parallèle.

Dépendances : CUDA ou OpenCL

1.4.2 Optimisation additionnelle

Il s'agit d'améliorer les performances globales de l'algorithme, cela consistera à identifier les goulots d'étranglement grâce à des outils de profilage, et proposer des solutions adaptées. On précisera tout de même que cette étape d'optimisation sera réalisée à chaque milestone si l'application a une faible réactivité.

1.4.3 Test

Que ce soit pour l'optimisation finale ou à chaque milestone on pourra directement observer les performances de notre moteur grâce au nombre d'images par seconde rendus. On peut se considérer interactif à partir de 14 images par seconde mais l'objectif reste de maximiser ce chiffre.

En plus de nos propres test, on peut imaginer comparer nous résultat avec une démo de l'implémentation de l'approche PBD à notre disposition

1.5 Spécifications globales

Voici les spécification qui sont valables quelque soit la milestone.

— Les algorithmes sont implémentés en tant que plugin du Radium engine.
— Avoir un temps de calcul interactif.
— Un stabilité correcte pour des interactions raisonnables.
— Satisfaction du client.

Chapitre 2

Planning prévisionnel

Voici comment nous avons décider d'organiser notre développement. Les ressources allouées à chaque tache, leur durée est approximative. Aussi les tache s'enchaînent directement sans période de battement mais il va de soit que la réalité sera autre.