Université de Limoges
Faculté des Sciences et Techniques

Année Universitaire 2017 - 2018
version du 15 octobre 2018

Modélisation et Animation
GPU-based Cloth Simulation

Auteurs

Baron Marin
Nicot Fabien

Master Informatique ISICG, semestre 3

Responsables : CRESPIN Benoit

Table des matières

Introduction

Nous avons choisi de présenter le papier « Contact-Aware Matrix Assembly with Unified Collision Handling for GPU-based Cloth Simulation » rédigé par Min Tang, Huamin Wang, Le Tang, Ruofeng Tong et Dinesh Manocha. Ils présentent une nouvelle méthode permettant de réaliser la simulation des vêtements. C'est une méthode fonctionnant de manière parallèle sur GPU. Il permet de résoudre les intégrations sur le temps ainsi que la gestion des collisions sur le GPU.

Les collisions et les contacts entre les tissus ont toujours été un problème récurrent dans le domaine de la simulation de vêtements. Beaucoup de recherches ont été effectuées pour améliorer la résolution de ces problèmes comme : la détection de collisions continues étudiée par [BEB12, Wan14, TTWM14], l'impulsion des collisions par [BFA02], la méthode de résolution des contraintes par [OTSG09], la méthode des zones d'impacts [Pro95, HVTG08] pour la réponse des collisions.

Même si la plupart des collisions sont détectées il ne suffit que d'une seule manquée pour que cela entraine des artéfacts visuels visibles. La méthode la plus utilisée pour résoudre ce problème est la méthode de détection de collision continue (CCD). Cependant la méthode est très couteuse en calcul et limite les méthodes d'utilisation des simulateurs de tissus.

Ce papier s'est donc intéressé à une méthode hautement parallélisé sur GPU permettant de réaliser la simulation de vêtements à l'aide de maillages flexibles, l'intégration temporelle et la gestion des collisions.

1 Présentation générale de la méthode

Cette méthode présente un nouveau pipeline basé GPU pour la simulation de vêtements. Elle utilise en entrée un maillage de triangles et produit une matrice creuse afin d'optimiser la résolution linéaire sur GPU [NVI15, BG13, WBS* 13]. Un système de simulation classique de vêtements se sépare en 3 parties : l'intégration temporelle, la détection des collisions et la réponse à la collision.

Afin de réaliser l'intégration implicite temporelle il faut collecter toutes les forces de contact dont notamment les forces de ressort, de frottement et d'adhérence en utilisant le contrôle de proximité. Ensuite il faut combiner ces forces avec les autres forces externes (gravité, vent, ...) pour former la force résultante (principe fondamental de la dynamique). On isole le vecteur vitesse qu'on va déterminer en intégrant.

Tout d'abord, il faut chercher les objets à proximité à l'aide de la méthode de détection de collisions discrètes (DCD) qui est beaucoup plus rapide à calculer que la CCD. Pour optimiser on sauvegarde le devant du BVH des objets dans un BVTT [LC98] mais la mise à jour coûte cher. On ne modifie donc qu'une vingtaine de noeuds le plus proche du point

d'impact : c'est la détection de collision localisée. Pour vérifier les pénétrations on teste la détection de collisions continues (CCD) avec chaque paire de triangles dans le BVTT et on produit une paire Arête/Arête ou Vertex/Triangle quand c'est vraie.

Avec les zones d'impactes on forme un système linéaire pour calculer les vecteurs de vitesse de la réponse à la collision, à chaque vertexe.

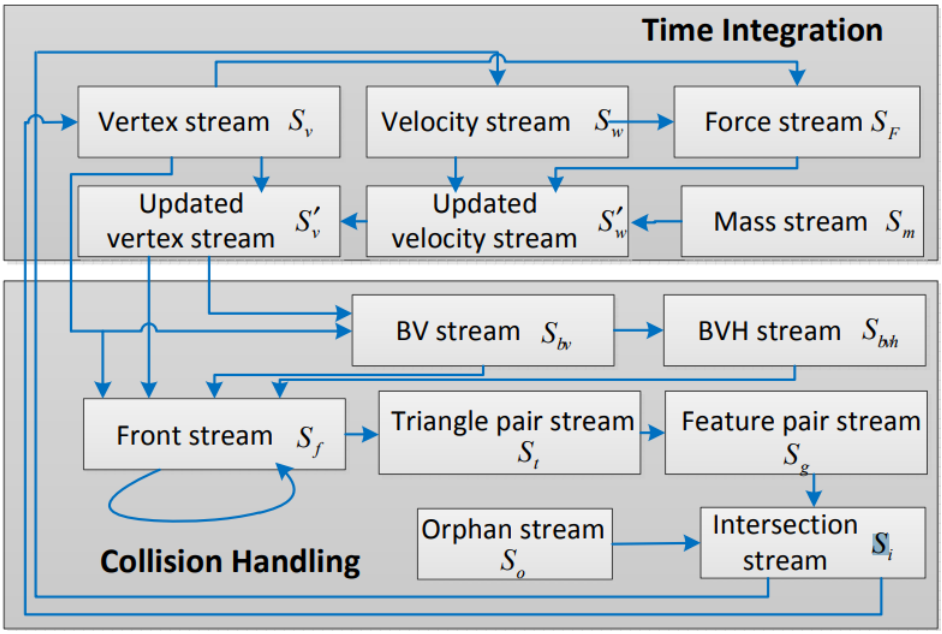

FIGURE 1 – Pipeline d'exécution

Cette méthode utilise un nouveau pipeline (voir figure 1) qui diffère des précédents pipelines utilisés [BFA02, TTN*13]. La matrice utilisée dans le système linéaire utilise le format BCSR [NVI15]. Un point-clé dans la réalisation de cette méthode est d'identifier les différents flux de données :

- Les flux de données de sommets et de vélocités contiennent l'état courant des sommets du tissu.
- Les flux de mise à jour des précédents flux de données
- Les flux de données des différentes boîtes englobantes (BV/BVH/BVTT)
- Les flux de données des intersections

2 Intégration implicite du temps

Une fonction principale de cet algorithme est la réalisation implicite de l'intégration temporelle de manière parallèle sur le GPU. Pour cela il faut réaliser un système linéaire creux. Cette méthode résout les systèmes linéaires en utilisant le préconditionneur de Jacobi. $M\ddot{u} + D\dot{u} = f(u)$ où M et D sont les matrices de masses et d'amortissements de ressorts. Avec u la position et $f(u)$ la force résultante. u est intégré par rapport au temps.

On approxime f^{t+1} au temps $t+1$ à l'aide de cette équation : $f(u^{t+1}) \approx f(u^t) + K(u^{t+1} - u^t)$ où K est la matrice Jacobienne évaluant f à l'instant t, et on obtient $Av^{t+1} = (M + \Delta tD - \Delta t^2 K)v^{t+1} = Mv^t - \Delta t f(u^t)$. Où v^{t+1} est le vecteur inconnu correspondant à la vélocité au temps $t+1$. (Si $f(u)$ contient les forces d'élasticité et les forces externes, les valeurs différentes de zéro de la matrice A correspondent aux voisins de chaque sommet.)

L'utilisation de la fonction $f(u)$ afin de gérer les forces de contacts et les contraintes sur la structure de la matrice varie en fonction du temps et doit donc être reconstruite à chaque étape de temps. Il est plutôt simple de réaliser le traitement des matrices creuses sur le CPU, cependant il n'existait pas d'algorithmes rapides pouvant réaliser le traitement sur le GPU.

L'article présente aussi une nouvelle méthode afin de réaliser l'assemblage de la matrice creuse en CSR et étendu ou BCSR. La première étape est de remplir les index :

- Déterminer l'espace mémoire à allouer en comptant les éléments de chaque ligne.
- On remplit les indices dans l'espace allouer de la partie précédente.
- Ensuite il faut trier et supprimer les doublons de ces indices.
- On réduit le nombre de lignes en réalisant une somme pour déterminer les indices afin de supprimer une ligne sur deux.

Ensuite il faut remplir les données correspondant à chaque élément trouvé en fonction des données d'index précalculées. On étend la méthode CSR à la méthode BCSR au lieu de stocker les données de manière indépendante, elles sont stockées dans des petits blocs de données représentées par une petite matrice 3 par 3.

Toutes les étapes de la construction de la matrice creuse sont effectuées de manière parallèle et le plus souvent sur chaque ligne de la matrice.

La compression de cette matrice permet d'exploiter les performances de l'architecture du GPU. Toutes les forces sont prise en compte dans la méthode, incluant les forces internes, les forces externes, les forces de contacts et de friction.

La méthode résout le système d'équations obtenues à l'aide des forces, grâce à la méthode du gradient conjugué et accélérée par le préconditionneur de Jacobi.

3 Gestion des collisions

La gestion des collisions se fait en parallèle par détection de collisions continues. Des travaux précédant ont réussit à implémenter cette méthode sur GPU [TMLT11]. Pour améliorer ces méthodes, cet article propose de décomposer le problème en deux parties, le calcul de la proximité à l'aide d'une détection de collision discrète, pour ensuite réaliser le calcul de pénétration avec la méthode de détection de collisions continues qui est réalisée en

une propagation localisée. Ces deux méthodes permettent de réaliser le calcul de collision de manière parallélisér et bien plus rapide que les méthodes précédentes.

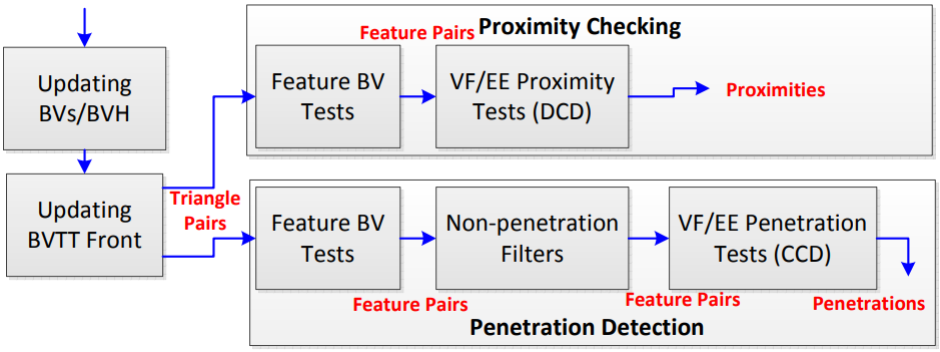

FIGURE 2 – Détection des collisions

L'ensemble des étapes sont résumées dans la figure 2. La première étape est de créer les boites englobantes (BV) pour ensuite créer l'arbre transversal des boîtes englobantes (BVTT) qui est utilisé afin de réaliser la propagation localisée.

La détection localisée des collisions peut être réalisée car seuls les éléments proches des éléments qui viennent d'être mis à jour peuvent subir une collision. C'est pour cela que l'utilisation d'un BVTT pour réaliser la propagation locale qui teste les collisions permet d'augmenter les performances de la méthode.

4 Présentation du premier article

Le premier papier cité dans cet article que je vais présenter est « Robust Treatment of Collisions, Contact and Friction for Cloth Animation » par Robert Bridson, Ronald Fedkiw et John Anderson. Cet article propose une méthode afin de gérer les collisions qui peut être implémentée avec toutes les méthodes simulant les mouvements internes. Cet article propose la combinaison entre deux améliorations, une meilleure géométrie diminuant le nombre d'intersections, et une méthode introduisant des forces de répulsions.

Pour accélérer la détection des proximités et des collisions, cet article utilise des boîtes englobantes AABB. Après avoir déterminé les triangles qui peuvent être intersectés, il faut calculer la proximité du point que l'on teste avec les triangles obtenus, pour cela il faut calculer la distance du point avec le plan du triangle puis calculé le barycentre du point projeté sur le plan. Pour calculer la proximité de deux arêtes il suffit de calculer la distance entre les deux points les plus proches sur les arêtes. Pour des objets qui se déplacent (points et arêtes), il suffit de calculer pour quel delta de temps les objets seront coplanaires.

Pour appliquer les calculs de la vélocité à des triangles il faut interpoler les vélocités des sommets des triangles avec les coordonnés barycentriques. Après avoir détecté les collisions comme expliquées au paragraphe précédent il faut appliquer une force de répulsion basée sur des ressorts. Le modèle utilise la loi de Coulomb afin de calculer l'énergie statique et l'énergie cinétique en un seul paramètre afin de représenter les forces de frictions.

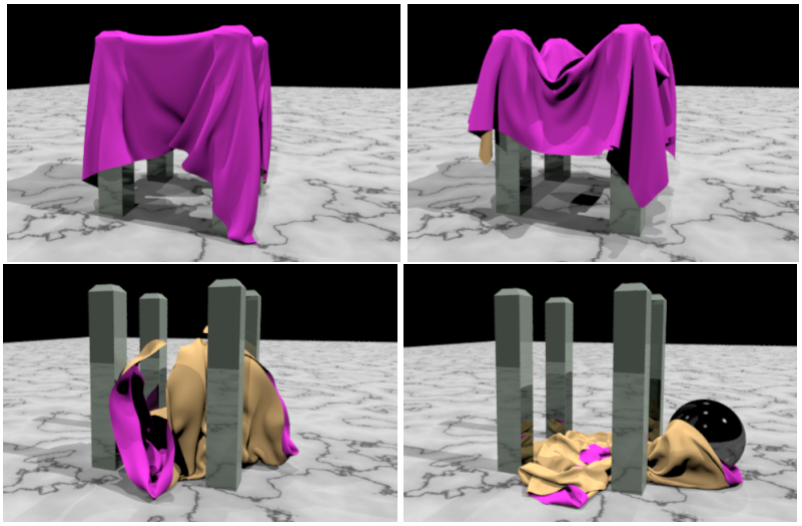

FIGURE 3 – Rendu obtenu par cet article de recherche

Les forces de répulsions permettent de traiter la plupart des cas de collisions, cependant il peut rester quelques collisions qu'il faut traiter de manière géométrique. Le meilleur rendu serait de traiter ces collisions de manière géométrique de manière chronologique cependant cette méthode utilise une résolution itérative simultanée de toutes les collisions. Le résultat obtenu n'est pas aussi convaincant que la méthode chronologique, cependant le gain en temps d'exécution est plus important. Cet article propose aussi de traiter les nœuds qui subissent plusieurs collisions en utilisant une zone impact.

5 Présentation du deuxième article

Le deuxième que l'on va vous présenter est « A GPU-based Streaming Algorithm for High-Resolution Cloth Simulation » par Min Tang, Ruofeng Tong, Rahul Narain, Chang Meng et Dinesh Manocha. Cet article qui date de 2013 est la version précédente de la méthode que l'on présente, on va donc présenter les améliorations qui ont pu être ajoutées à cet algorithme.

Dans le papier de 2013 utilise la détection des collisions discrètes pour calculer les forces de répulsions, cela ne prend pas en charge les couches multiples de vêtements. L'algorithme permet de gérer uniquement des tissus rectangulaires. Ils gèrent aussi les pénétrations en utilisant des pénalités et une méthode d'impulsion, cependant la nouvelle méthode implémente les contraintes de proximités dans l'intégration sur le temps. De plus

cette méthode n'utilise pas un arbre de boîte englobante afin de réaliser la propagation localisée pour la détection de collisions.

6 Présentation du travail a réaliser

Il faudra se mettre d'accord à quel point on respecte fidèlement l'algorithme, et sur les détails d'implémentation :

L'exécution se fera sur CPU (séquentiel/parallèle ?) ou GPU.

Le choix du langage est contraint par le format du programme : si on fait un programme autonome on est libre du langage mais si on fait une extension on est dicté par le logiciel de rendu. Pour python (Blender), C++ (Unreal Engine 4), java (lancer de rayon de Synthèse d'Images de l'an dernier), OpenCL, Cuda, ... Julia ?

Les possibilités de format de matrices sont : denses, COO, CSR, BSR, DIA, ELL.

Lancement du programme :
- indépendant (en ligne de commande qui génére des "obj", ...)
- plugin (Blender, ... affiche le rendu)

A la fin de l'implémentation on espère avoir des propositions de solution aux problèmes soulevés par les auteurs.

Au minimum on pense développer un système masse-ressort si possible en parallèle. Le langage serait C++. Le fonctionnement est le suivant chaque sommet du maillage possède un système masse-ressort-amortisseur.

$$\sum F_x = F_{(t)} - b\dot{x} - kx = m\ddot{x}$$

avec F_x une force extérieur selon x, $F_{(t)}$ la force de réponse du ressort, $-b\dot{x}$ la force d'amortissement, $-kx$ la force de rigidité.

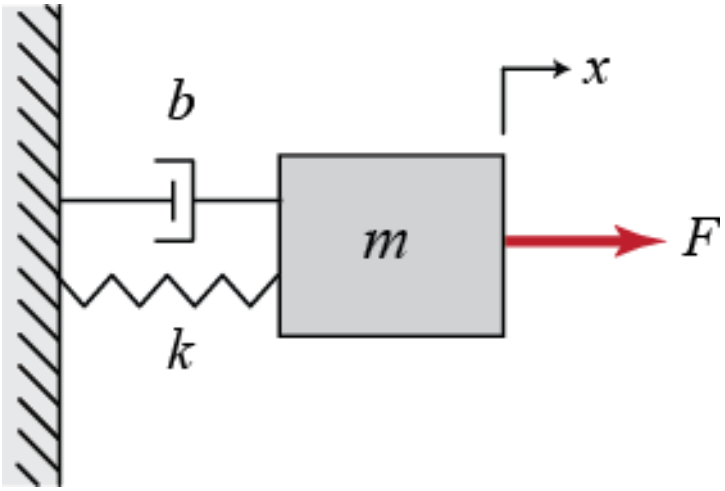

FIGURE 4 – Masse-Ressort-Amortisseur

On essayera d'intégrer le modèle physique de ressort 4x4 ou 3x3 avec les ressorts de structure, déchirure et courbure.

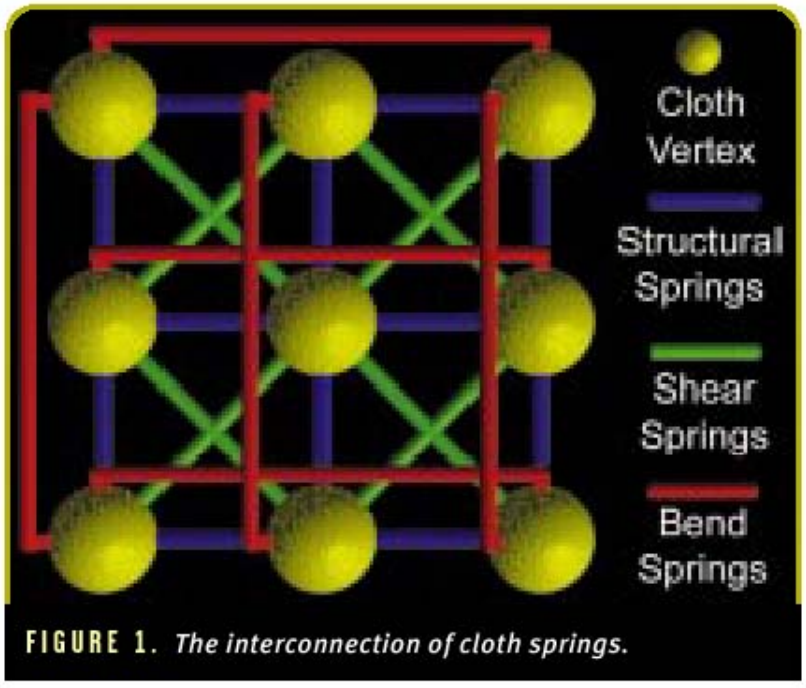

FIGURE 5 – Modèle de ressort 3x3

A Illustrations et Images

Table des figures

Références

[1] Jérémie Allard, Hadrien Courtecuisse, François Faure, et al. Implicit fem solver on gpu for interactive deformation simulation. *GPU computing gems Jade Edition*, pages 281–294, 2011.

Formats de matrices : `https://medium.com/@jmaxg3/101-ways-to-store-a-sparse-matrix-c7f2bf15a229` Masse-Ressort-Amortisseur : `http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=SystemModeling#5`