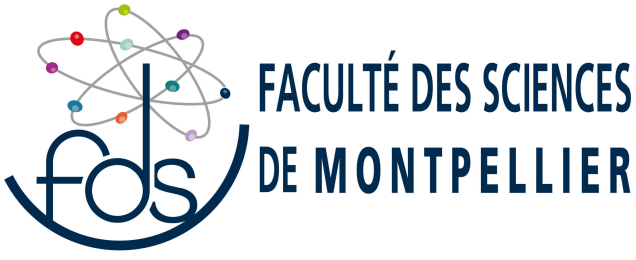
Université de Montpellier

M2 - Imagine
Projet 3D
Simulation d'objet non-rigide
Système masse-ressort

Etudiant :
Nicolas LUCIANI

Encadrant :
Noura Faraj

Année 2023-2024



Sommaire

1. Introduction

La création d'un système masse ressort pour simuler des objets non-rigide est un travail très intéressant qui touche à plusieurs aspects de l'informatique graphique. L'intérêt premier auquel est liée la simulation d'objets non-rigide par les systèmes masses-ressorts est pour la simulation ou l'animation de vêtements, ou de tissus plus généralement. Lors du choix de ce sujet, il été clair qu'une difficulté allait être rencontrée très vite : la stabilité du système. Nous verrons dans le travail présenté ici comment nous avons implémenté différentes méthodes pour atteindre une simulation efficace, et stable, en plus d'être réaliste. Nous présenterons d'abord le sujet du travail, les outils utilisés, les objectifs du projets ainsi que les inspirations. Ensuite, nous aborderons quelques définitions et lois nécessaire à l'implémentation du projet. Nous discuterons des différentes méthodes mises en places et quelle méthode a été retenue. Nous présenterons des captures d'écran de la simulation et de l'application.

2. Présentation du sujet

On implémentera une simulation d'objets non rigide à l'aide d'un système masse-ressort. On détaillera ensuite quels sont les objectifs précis. On trouvera dans nos références des articles et des sujets de travaux pratiques jugés pertinents. Vous trouverez ici le lien de la présentation orale.
Le lien suivant dirige vers une vidéo de démonstration.
Enfin, vous trouverez le code sur un dépot *GitHub* personnel ici

2.1 Outils et Matériel

On développera une application Qt sur la base de code fournie. Le langage utilisé sera donc C++ et la librairie OpenGL. On utilisera la librairie Eigen pour les représentations mathématiques et les calculs sur les matrices, la résolution de système linéaires et autres calculs matriciels.

2.2 Objectifs

— Implémenter le système masse ressort pour simuler un plan non rigide
— Possibilité d'interagir avec ce plan en contrôlant un objet
— Détection et résolution des collisions

La simulation finale proposera une interface dans laquelle l'utilisateur pour fixer au choix des points du plan non rigide, puis contrôler ou lancer des sphères sur ce plan pour observer le comportement de l'objet non rigide.
Pour aller plus loin, la simulation pourrait proposer de modifier le matériau de l'objet non-rigide pour adapter le comportement de l'objet.

2.3 Inspirations

La simulation pourrait par exemple représenter cette visualisation de la gravité selon Einstein proposée par ce professeur dans cette vidéo.

3. Système masse ressort

3.1 Définitions

Admettons une particule, un point, avec une masse donnée. On parlera plus simplement de *masse* pour définir la particule. On lui donne également une position, une vélocité, une accélération. Un ressort est défini comme un moyen de connecter deux masses. Un ressort possède les propriété suivantes :

— Une longueur au repos
— Une constante de raideur (stiffness)
— Une constante de frottement fluide ou amortissement (damping)

Un système masse ressort se décrit comme un ensemble composé d'au moins deux masses, fixées ou non, connectée par un ressort. La figure 3.1 est un schéma présentant un système masse ressort, appelé aussi oscillateur, car composé d'une masse fixe à laquelle est attachée une masse qui oscillera grâce au ressort. Cette base nous permet d'étudier les forces impliquées dans la simulation de ce

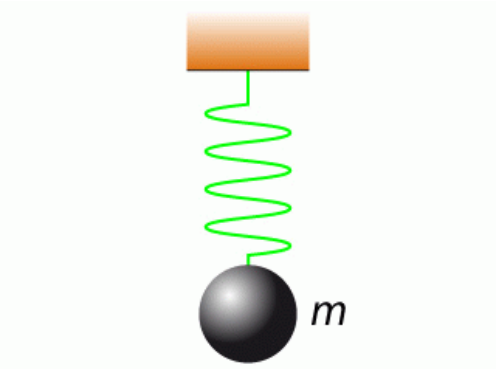


FIGURE 3.1 – Système masse-ressort avec une masse fixe
Source : http://res-nlp.univ-lemans.fr

genre de système. Pour cela, nous avons besoin de rappeler la loi de Hooke [3.1] :

$$F = k\Delta l \tag{3.1}$$

avec F la force du ressort appliquée sur une masse, k la raideur (ou contrainte de rappel) du ressort, et Δl la variation de longueur (la différence entre la longueur au repos et la longueur du ressort à un instant donné). Une constante d'amortissement peut être donnée au ressort, et va servir à amortir la vitesse de la masse au cours du temps, de façon à s'assurer qu'elle s'arrête et n'oscille pas à l'infini.

3.2 Structures de données

Pour notre première implémentation, qui utilisera les méthodes d'Euler et de Verlet pour la simulation (sections **4.1** et **4.2**), nous avons défini plusieurs classes C++. D'abord une classe pour définir une masse comme vu précédemment. C'est dans cette classe que seront implémentées les

méthodes d'Euler et Verlet pour directement mettre à jour la position des masses. Une méthode est implémentée pour calculer la somme des forces appliquées à la masse telles que décrites plus tôt également. L'ensemble des forces appliquées à une masse est la somme des forces appliquées par les ressort auxquels elle est attachée, et son poids.
Un ressort est défini par les deux masses qu'il relie, ainsi que les constantes décrites plus tôt. Un ressort implémente les méthodes permettant de calculer la force qu'il exerce sur une des masses auquel il est attaché.
Une classe définit un système masse ressort, permettant de stocker toutes les masses et ressorts, appliquer les mise à jour temporelles.
Enfin, un gestionnaire de simulation permet de générer un système, et communique avec l'interface.
Cette implémentation a évolué au fur et à mesure du projet, mais il était intéressant de l'évoquer avant de discuter des méthodes d'integration numérique.

4. Méthodes d'intégration numérique

4.1 Méthode d'Euler

Comme proposé dans le cours de Damien Rohmer [1], on peut assez aisément disposer d'un système masse ressort fonctionnel en implémentant la méthode d'Euler et la seconde loi du mouvement de Newton [4.1].

$$\begin{aligned} a(t) &= \frac{F(t)}{m} \\ v(t+\Delta t) &= v(t) + a(t)\Delta t \\ p(t+\Delta t) &= p(t) + v(t)\Delta t \end{aligned} \tag{4.1}$$

avec a(t), v(t), p(t) respectivement l'accélération, la vélocité et la position d'une masse m à l'instant t, et F(t) la somme des forces appliquées sur la masse m à l'instant t.
Les forces appliquées sont donc calculées par la loi de Hooke et la force de gravité. Nous avons pu mettre en place cette méthodes pour deux systèmes :

- Système à deux masses, dont une fixe
- Système à trois masses, dont une fixe

La figure 4.1 montre des captures d'écran des simulations pour ces deux systèmes. Déjà un problème

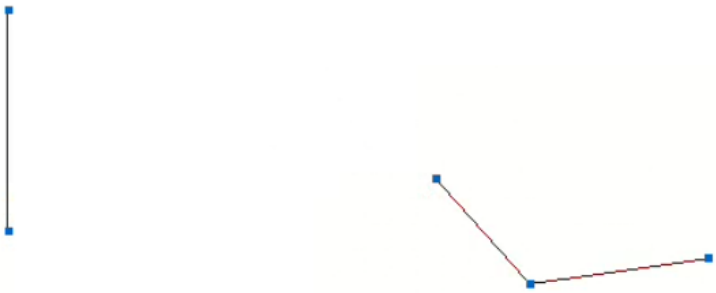


FIGURE 4.1 – Systèmes masse ressort simulés avec la méthode d'Euler

se posait pour le système à 3 masses. Plusieurs tests de valeurs de constantes ont été réalisés pour obtenir ces résultats, la stabilité du système en est très dépendante. Très vite, le système à 3 masses arrive vers l'état ou la seconde masse tourne de plus en plus vite autour de la masse fixe, entraînant avec elle la troisième.
Nous avons testé cette méthode sur un système de dimensions 16x16, sans succès. En effet, on pouvait se douter que les problèmes décelés avec 3 masses allaient sérieusement impacter un système

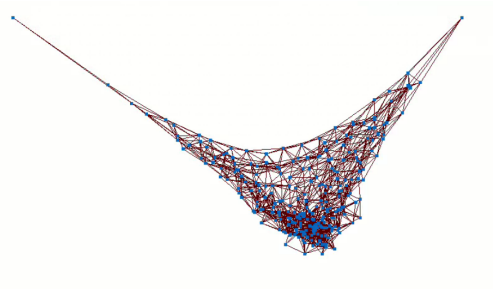


FIGURE 4.2 – Simulation d'un système masse ressort de dimension 16x16

plus important. La capture d'écran en figure 4.2 le montre. Il est clair que cette méthode n'est pas satisfaisante pour atteindre nos objectifs. Nous avons jugé intéressant d'utiliser une méthode d'intégration que l'on sait plus précise sur le calcul des positions : l'Intégration de Verlet.

4.2 Intégration de Verlet

La méthode d'intégration de Verlet est une méthode intéressante pour sa stabilité et sa rapidité. Les formules d'intégration sont données dans l'équation 4.2 .

$$\begin{aligned} a(t) &= \frac{F(t)}{m} \\ p(t+\Delta t) &= 2p(t) - p(t-\Delta t) + \Delta t^2 a(t) \end{aligned} \tag{4.2}$$

Cette méthode d'intégration numérique permet en effet d'avoir des résultats assez stable pour simuler un système avec 3 masses comme en Figure 4.1 et même un système de dimensions 32x32 avec deux masses fixes telle que le montre la figure 4.3. Cependant, lorsqu'on fait varier les constantes de raideur des ressorts du systèmes (toute équivalentes), on atteint très vite des instabilités qui cassent complètement le système comme on peut le voir dans la figure 4.4. De plus, la méthode de Verlet est connue pour ne pas être adéquate pour le calcul des collisions, notamment la réponse aux collisions. Il est donc paru évident que ce n'était pas la méthode idéale non plus.

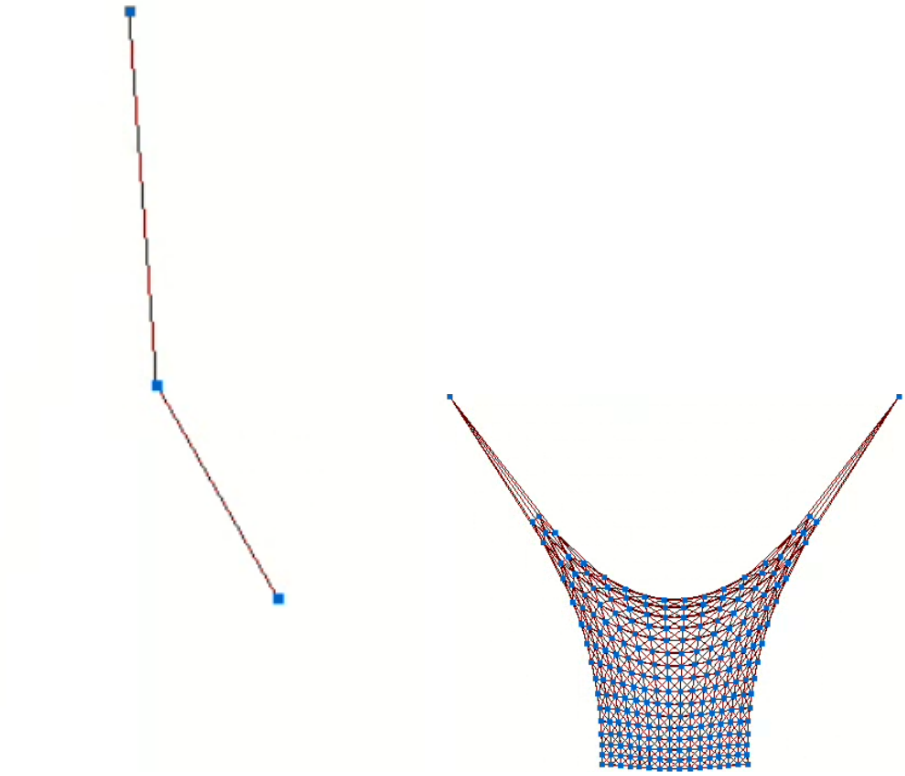


FIGURE 4.3 – Systèmes masse ressort simulés avec la méthode de Verlet

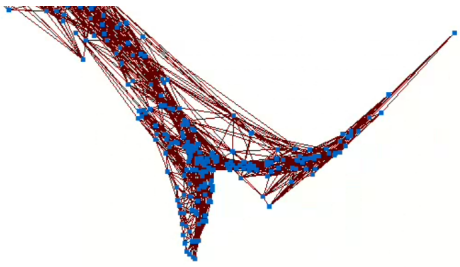


FIGURE 4.4 – Simulation d'un système masse ressort de dimension 32x32 instable avec une constante de raideur $k > 80$

5. Méthode d'Euler implicite

5.1 Introduction

En parcourant le cours de Florence Zara [2] sur les systèmes masse ressort, l'intuition de la solution à nos problèmes s'est tournée vers la méthode d'Euler implicite. La méthode présentée ensuite est une méthode décrite dans l'article [3] basée sur la méthode d'Euler implicite, qui propose un *solver* rapide pour des systèmes masses ressorts régis par la loi de Hooke. La méthode étant proposée pour des applications en temps réel, les résultats sont décrits comme visuellement acceptable, avec une méthode rapide et stable pour un coût par intervalle de temps très intéressant. Ce que nous cherchons ici ce ne sont pas des résultats physiquement exacts, mais plutôt esthétiques et réalistes. Nous nous intéresserons d'abord à comprendre le principe de cette méthode, en reprenant les éléments avancés par les auteurs. Il est important de noter que certaines notions mathématiques de la méthode (également survolés dans l'article) ne seront pas abordées ici. D'une part, elles sont au delà de nos connaissances à ce jour, et d'autre part elle dépassent le sujet du projet.

5.2 Définitions

La méthode propose plusieurs notations, que nous reprendrons ici pour éviter les confusions.
On définit h l'intervalle de temps constant. L'état du système à un instant t_n est noté q_n, tel que q_n est l'ensemble des positions des particules à l'instant t_n.
On définit M la matrice des masses de taille $3m \times 3m$ avec m nombre de masses dans le système. M est une matrice de masse diagonale condensée. Autrement dit, la matrice est construite telle que les masses des particules du système se trouvent sur sa diagonale, le reste de la matrice est constitué de zéros.
On définit une matrice L de taille $3m \times 3m$, contenant les constantes de raideurs des ressorts par rapport aux masses auxquels ils sont liés. Pour mieux cerner ce que décrit cette matrice, on définit un ressort de raideur k entre les masses d'indices i et j dans q_n. Alors on a :

$$\begin{array}{rrrr} L_{ii} = k & L_{ij} = -k & L_{ji} = -k & L_{jj} = k \\ L_{i+1i+1} = k & L_{i+1j+1} = -k & L_{j+1i+1} = -k & L_{j+1j+1} = k \\ L_{i+2i+2} = k & L_{i+2j+2} = -k & L_{j+2i+2} = -k & L_{j+2j+2} = k \end{array}$$

La matrice J est similaire, mais associe une masse et un ressort par leurs indices. En effet, on dispose d'un ensemble de ressorts (ou plus exactement de contraintes) indexés.
On définit enfin d comme un vecteur contenant les direction des longueur au repos des ressorts. Détaillons à présent comment nous nous servons de tout cela.

5.3 Principe

L'intégration se fait en plusieurs temps. À l'initialisation du système, on calcule les matrices M, L, et J. Cela nous permet de calculer la factorisation *sparse* de Cholesky de la matrice A telle que :

$$A = M + h^2 L \tag{5.1}$$

Ensuite, à chaque intervalle de temps, lors d'une étape dite "locale", on calcule le vecteur d. L'étape dite "globale" permet de calculer q_{n+1} comme la solution du système $AX = b$ avec b tel que :

$$b = M(q_n(c+1) - q_{n-1}c) + h^2Jd + h^2F \tag{5.2}$$

avec c la constante d'amortissement, et F les forces extérieures (ici seulement la gravité).

5.4 Implémentation

Pour implémenter cela en *C++*, nous nous sommes inspirés de l'implémentation des auteurs, ce qui a grandement aidé à la compréhension des calculs décrit plus tôt.
Les classes représentant les masses, et les ressort comme vu pour nos premières implémentations n'ont plus raison d'être car ces éléments sont directement portés par les matrices et l'état du système q_n.
Nous avons d'une part une classe centrée sur ce système, avec des vecteurs comprenant les valeurs des masses, les positions, les constantes de raideurs, les forces extérieures. Des variables accessibles décrivent le nombre de masses ou le nombre de ressorts dans le système (utiles pour déterminer la taille de ces vecteurs).
Une autre classe, appelée *solver* est dédiée aux calculs matriciels et à l'instanciation de contraintes que nous détaillerons ensuite. C'est dans les méthodes de cette classe que nous implémentons les calculs vus dans les sections précédentes.
Nous avons conservé le gestionnaire de simulation, qui définit les paramètres du système, et sert d'intermédiaire avec les classes s'occupant de l'interface (affichage, interactions).
Deux classes ont été ajoutée pour construire le système (une grille uniforme) et l'afficher (avec les méthodes OpenGL).

5.4.1 Contraintes

Les contraintes sont une représentation des liens entre les masses par les ressorts. C'est par ces contraintes que sont calculées les matrices L et J ainsi que le vecteur d. Nous allons voir deux types de contraintes que nous avons défini dans notre simulation.

5.4.1.1 Contraintes de ressort

Les contraintes de ressorts sont les contraintes "basiques". Ce sont celles que l'on définit le plus facilement car décrivent l'effet des ressorts sur les masses libres. Celles-ci définissent les triplets insérés dans les matrices L et J et le vecteur d tels que décrits plus haut.

5.4.1.2 Contraintes de point fixe

Une contrainte de point fixe est une contrainte qui représente une masse fixe dans les systèmes des implémentations antérieures. Ces contraintes redéfinissent les triplets insérés dans les matrices L, J et le vecteur d. En effet, ces triplets ne sont plus définis que par rapport à une seule masse et non deux comme précédemment. Le vecteur d à l'indice i de la masse fixe ne contient plus la direction de la longueur au repos, mais simplement la position de la masse fixée. Maintenant que nous disposons d'une simulation d'un système masse ressort stable et rapide, il s'agit de s'intéresser aux collisions et aux interactions.

6. Collisions

Comme expliqué dans l'article [3], la réponse aux collisions peut être aisément effectuée en déplaçant les masses en dehors de l'objet avec lequel elles sont entrées en contact.
L'objet considéré ici est une sphère. Cela permet de bien visualiser la réponse aux collisions et le comportement du système autour de celle-ci.
On définit alors une sphère par son centre et son rayon. La détection est implémentée naïvement en vérifiant la distance de chaque masse au centre de la sphère, et en s'assurant que celle-ci soit supérieure ou égale au rayon de cette dernière. On compare les distances au carré pour éviter de calculer des racines carrés.

7. Interface

L'interface a été développée en QT avec les *widgets* OpenGL. Cela permet d'afficher une fenêtre OpenGL et d'interragir avec différents boutons, sliders, et autres élément d'interface utilisateurs. La figure 7.1 montre l'interface de l'application. Elle est composée d'un bouton permettant de réinitialiser la simulation, et deux sliders pour agir sur les contraintes du système. Aussi, il est possible de cliquer sur les contraintes de point fixe pour les déplacer sur le plan de la caméra. Pour les déplacer sur le plan formés par les axes x et z, il est possible d'appuyer sur la touche "Z" du clavier. Enfin, pour faire apparaître une sphère et étudier les interactions avec le système, la touche "S" peut être appuyée.

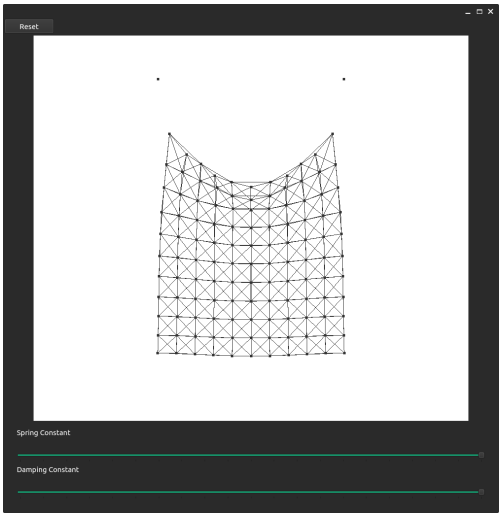


FIGURE 7.1 – Interface de l'application

8. Résultats

Vous trouverez ci-dessous des captures d'écran des résultats obtenus.

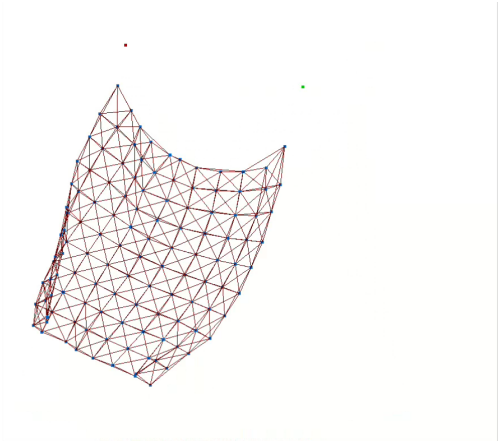


FIGURE 8.1 – Systèmes masse ressort simulés avec la méthode décrite plus tôt

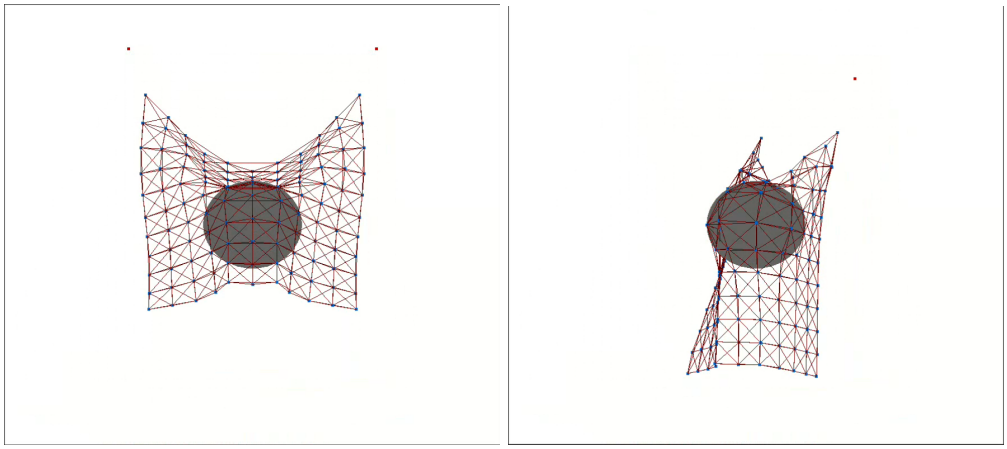


FIGURE 8.2 – Systèmes masse ressort en collision avec une sphère

9. Conclusion et perspectives

Nous avons donc implémenté un système masse ressort pour de la simulation d'objets non-rigides. Ce projet s'est trouvé très enrichissant. Implémenter une méthode développée dans un article est un exercice qui permet de cerner des aspects de l'informatique graphique simplement effleurés jusqu'alors. La phase de recherche et de documentation apporte aussi beaucoup au projet, l'inspiration venant de chaque lecture. Pour la suite, il serait très intéressant de pouvoir importer des maillages et les rendre en objet non-rigide, car la méthode présentée dans l'article le permet. Aussi, il serait pertinent de peaufiner la détection des collisions, en implémentant une structure d'accélération, pour faire de la détection par faces. De meilleures interactions utilisateurs et des rendus plus jolis (shaders, etc...) seraient aussi très appréciés.

Bibliographie

[1] *Damien Rohmer, Cours sur la simulation.* (consulté en oct. 2023).

[2] *Master 2 Image, Développement et Technologie 3D (ID3D) UE Animation, Cours Articulés et Moteurs Physiques Partie - Simulation par modèles physiques Cours - Système masses-ressorts Florence Zara - LIRIS - Université Lyon 1.* (consulté en oct. 2023).

[3] Tiantian Liu et al. « Fast Simulation of Mass-Spring Systems ». In : *ACM Trans. Graph.* 32.6 (nov. 2013). ISSN : 0730-0301. DOI : 10.1145/2508363.2508406. URL : https://doi.org/10.1145/2508363.2508406.