Projet de Master: Implémentation de l'algorithme Pivoting Ball dans Cgogn

Paul Demanze Encadré par Lionel Untereiner

1 Introduction

À partir d'un nuage de points déconnectés, dans un espace en 3 dimensions, on cherche à obtenir une surface par triangulation.

L'algorithme Pivoting Ball permet de construire cette surface itérativement, en maitrîsant la complexité de calcul par rapport au nombre de points en entrée.

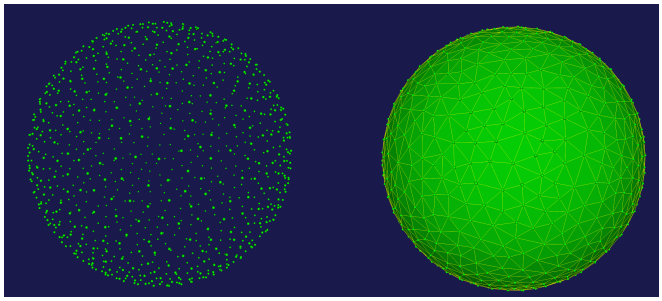

Figure 1: Exemple réalisé avec notre implémentation

Cgogn est une librairie C++ qui implémente les cartes combinatoires pour la modélisation géométrique. Le but de notre projet est d'implémenter l'algorithme du Pivoting Ball à partir des fonctionnalités existantes de Cgogn. Les entrées/sorties de l'algorithme seront donc des cartes combinatoires.

2 Principe du Pivoting Ball

Un triangle est considéré comme acceptable pour la surface, si et seulement s'il existe une balle qui touche les trois points du triangle, sans contenir aucun autre point.

On commence par chercher un triangle initial qui respecte cette propriété, et on l'appelle seed. Pour éviter de choisir n'importe quel triangle, on choisit un rayon de balle qui s'ajoute comme contrainte supplémentaire et limite le nombre de candidats possibles.

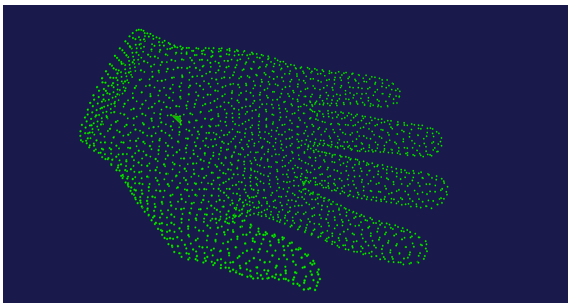

Figure 2: Le seed est le premier triangle du front

Une fois le seed trouvé, on ajoute ses trois arêtes dans le front avançant. Le front est un ensemble d'arêtes qui servent de point de départ pour trouver le prochain triangle, en faisant "pivoter" la balle qui a causé le triangle précédent.

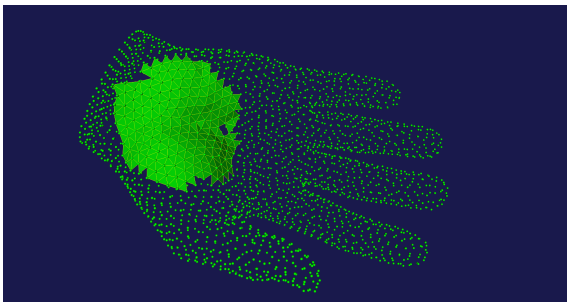

Figure 3: Le front avance

À chaque fois qu'un nouveau triangle est ajouté à la surface, ses trois arêtes sont chacune soit ajoutée au front si elles font partie du bord, soit supprimée si la même arête se trouve déjà dans le front.

Si on prend une arête du front, et la balle ne peut plus pivoter à partir de cette arête (aucun triangle valide), alors on considère qu'il s'agit du "bord" de la surface et l'arête est simplement retirée du front.

Selon le rayon de balle choisi, le résultat obtenu sera plus ou moins conforme à nos attentes.

- Un rayon trop petit laissera des trous. Des triangles supplémentaire seraient possibles mais la balle associée serait trop grande. Les arêtes sont retirées du front sans générer les triangles désirés.

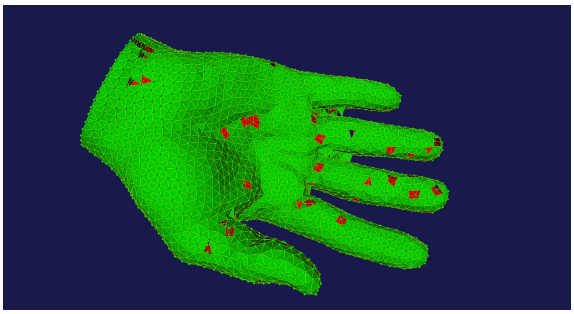

Figure 4: Le front n'a plus d'arêtes, et il reste des trous

- Un rayon trop grand autorisera des triangles trop larges et des détails de la surface importants peuvent être perdus.

Une solution à ces problèmes consiste à commencer avec un rayon de balle le plus petit possible pour obtenir les meilleurs détails, puis à continuer le maillage avec un rayon de plus en plus grand afin de "boucher" les trous.

3 Implémentation

La surface construite est une carte combinatoire de Cgogn (CMap2).

La fonction add_face(3) nous permet d'ajouter un nouveau triangle.
La fonction sew_faces nous permet de coller deux arêtes ensembles afin d'assembler les triangles.

Pour tout triangle considéré, les trois points sont systématiquement triés pour que tous les triangles soit toujours orientés de la même façon (le sens trigonométrique).

Les arêtes sont classées en trois catégories:
- Les arêtes du front non testées (Untried Edges): on n'a pas encore fait pivoter la balle à partir de ces arêtes.
- Les arêtes du front déjà testées (Tried Edges): on a déjà essayé de faire pivoter la balle autour de ces arêtes mais aucun triangle valide n'a pu être formé. Ces arêtes constituent le "bord" ou les contours des trous.
- Les autres arêtes de la surface (Sewed Edges): ces arêtes sont le produit de la fusion de deux triangles. Logiquement, on ne veut plus jamais créer de nouveaux triangles qui entrent en contact avec une telle arête.

Pour trouver un seed:
- Pour chaque point, on cherche ses voisins proches selon le rayon de balle choisi
- Parmis toutes les paires possibles parmis ses voisins, on teste si le triangle formé avec le point central est valide
- Dès qu'un triangle valide est trouvé, on s'arrête: c'est le seed et on l'ajoute à la surface

Pour faire pivoter la balle autour d'une arête du front:
- On cherche les voisins proches de l'arête selon le rayon de balle choisi
- Parmis tous les points possible parmis ses voisins, on cherche si le triangle formé avec l'arête actuelle est valide
- Parmis tous les triangles valides possible, on choisit celui avec l'angle minimal (pour que la balle pivote le moins possible)
- Si un triangle valide est trouvé, on l'ajoute à la surface
- Si aucun triangle n'est valide, la Untried Edge devient une Tried Edge.

Lorsqu'on ajoute un triangle à la surface, on doit pour chacune des trois arêtes:
- S'il existe une Tried Edge ou Untried Edge qui est orientée dans le sens opposé, alors on la supprime (les deux arêtes s'annulent)
- Sinon, on ajoute la nouvelle arête au front en tant que Untried Edge

Pour éviter que la balle fasse demi-tour à cause d'erreurs numériques, un triangle est considéré comme invalide si parmi ses trois arêtes:
- Il existe une Tried Edge ou Untried Edge sur les mêmes points qui est orientée dans le même sens (au lieu du sens opposé)
- Il existe une Sewed Edge sur les mêmes points

4 Efficacité et optimisation

Notre but est d'obtenir une surface visualisable rapidement, la complexité globale par rapport au nombre de points doit donc idéalement rester quasi-linéaire.

4.1 Recherche de points

Nous avons décidé d'utiliser Nanoflann, une librairie header-only qui nous permet de stocker les points sous forme d'arbre KD, une structure de données qui permet d'accélerer la recherche de points.

L'arbre KD nous permet d'extraire tous les points dans un rayon donné de manière efficace. C'est exactement ce que nous faisons dans l'algorithme du Pivoting Ball. La complexité naïve est encombrante: on doit parcourir tous les points et calculer leur distance pour prendre la décision de l'inclure ou non, donc la complexité locale est linéaire. Grâce à l'arbre KD, on passe à une complexité logarithmique.

L'arbre KD est construit une fois au début avec l'ensemble de points initial: ensuite, il n'y a plus qu'à faire des recherches avec la méthode radiusSearch.

4.2 Elimination de points

La recherche de seed a une lourde complexité (jusqu'à cubique).

- Ce n'est pas un problème lorsque le rayon de la balle est très petit car la recherche de voisins ne renvoie rien ou peu de points, et donc on passe une fois sur chaque point (complexité linéaire).
- Ce n'est pas non plus un problème lorsque on cherche le premier seed, car on va en trouver un très vite et donc l'algorithme se terminera dès les premier essais.

Cependant lorsqu'on cherche un nouveau seed après avoir commencé la surface, le temps de recherche peut exploser: beaucoup d'essais vont échouer car les triangles seront invalides, car les points sont déjà utilisées par la surface.

Pour palier à ce problème, il faut retirer les points dont on aura plus besoin. Il faut donc un critère cohérent qui permet d'éliminer un point. En utilisant les termes définis plus haut, un point ne sera plus utilisé lorsque: au moins 1 arête l'utilise ET toutes les arêtes qui l'utilisent sont des Sewed Edge.

Si jamais il y a des Tried Edges qui utilisent un point, alors il peut s'agir d'un bord ou du contour d'un trou qui sera bouché plus tard. Le point est dans ce

cas, encore candidat pour de nouveaux triangles.

5 Avancée du travail

Le programme de test de l'algorithme est une interface QT, un viewer basique de Cgogn.

La touche S permet trouver un seed. Le programme teste 64 rayons de balles possibles, partant d'un nombre très petit et doublant la taille à chaque fois, jusqu'à ce qu'un seed valide soit trouvé. Cela permet de trouver un seed rapidement peu importe l'échelle de l'ensemble de points en entrée.

La touche A permettent de faire avancer le front d'un pas seulement. Cela permet de visualiser comment la surface évolue pendant l'algorithme. Les touches 0 et 1 permettent d'avancer plus vite (beaucoup de pas à la fois).

La touche F permet de finir le front entièrement.

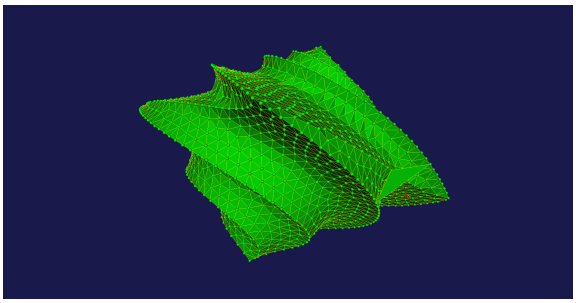

Figure 5: Exemple de surface entièrement reconstituée

Tous les principes décrit dans ce rapport sont implémentés sauf l'optimisation d'élimination des points, qui est cruciale pour pouvoir chercher plusieurs seed, afin de boucher les trous. En pratique, cette version n'est donc utilisable que pour calculer un seul seed puis finir le front. Il faudra étendre le code avec cette optimisation pour avoir une version générique intéressante pour Cgogn.

References

[1] The Ball-Pivoting Algorithm for Surface Reconstruction
IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, VOL. 5, NO. 4, OCTOBER-DECEMBER 1999

[2] Implementation of the Ball Pivoting Algorithm based on PCL
https://github.com/rodschulz/BPA

[3] Nanoflann
https://github.com/jlblancoc/nanoflann