Implantation fonctionnelle d'algorithmes géométriques en 3D

Titouan Laurent

Résumé

#TODO

1 Introduction

#TODO

1.1 État de l'art

2 Implémentation

2.1 Données d'entrées

Un *vec*3 permet de modéliser aussi bien un point qu'un vecteur dans l'espace 3D. Les composantes sont données dans l'ordre usuel x, y et z ($x, y, z \in \mathbf{R}$).
Dans notre implantation, on considèrera une simple liste ordonnée des n points (représentée par la variable globale $GLOBAL\_V\_LIST$) comme étant les données d'entrée du problème à résoudre.

2.2 Stockage et manipulation de l'enveloppe convexe en mémoire

La structure en liste de demi-arêtes orientées (doubly connected edge list, abrégé DCEL) est parmi les plus communes qu'il soit dès lors que l'on souhaite modéliser un maillage topologique (source nécessaire ?).
Ici on implante une version restreinte de cette dernière, puisqu'elle permet de créer maillages triangulaires uniquement — ce qui n'empêche pas au programme d'être fonctionnel comme nous le verrons ensuite.

2.2.1 Demi-arête — *he*

Une *he* (abrévation de halfedge, ou demi-arête) est une structure de données disposant d'informations sur elle-même et son voisinage topologique. Celles-ci sont les suivantes :

— $index \in \mathbf{Z}$, un identifiant unique pour la demi-arête, avec :

$$\begin{cases} index = -1 & \text{est une demi-arête nulle} \\ index \geq 0 & \text{sinon} \end{cases}$$

— $opposite \in \mathbf{Z}$, l'identifiant de la demi-arête opposée, avec :

$$\begin{cases} opposite = -1 & \text{est une demi-arête du bord} \\ opposite \geq 0 & \text{sinon} \end{cases}$$

— $vertex \in \mathbf{N}$, l'indice du sommet incident dont elle est issue.

2.2.2 Maillage triangulaire — *dcel*

Une *dcel* permet la modélisation d'un maillage topologique triangulaire. Elle dispose des attributs suivants :

— *he_list*, une liste de *he*.
— $available\_he\_index \in \mathbf{N}$, le prochain indice non attribué à une *he* de la liste.

2.2.3 Exemple simple en 2D

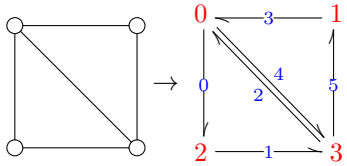

FIGURE 1 – Un maillage simple et un indiçage correspondant.

index	opposite	vertex
0	-1	0
1	-1	2
2	4	3
3	-1	1
4	2	0
5	-1	3

Indice	Coordonnées
0	(0,0)
1	(1,0)
2	(1,1)
3	(0,1)

available_he_index **:** 6

FIGURE 2 – Structure *dcel* à gauche et la table des sommets à droite.
Note : *avaiable_he_index* est un indice disponible pour la prochaine demi-arête qui pourrait être ajoutée à la *dcel*.

Considérons le maillage de la Figure 1. Une structure *dcel* associée valide serait alors telle que celle présentée Figure 2.

Note : si les numéros d'indices choisis pour les demi-arêtes sont ici plus ou moins arbitraires, la structure *dcel* impose tout de même des contraintes comme nous allons le voir.

2.2.4 Ajout et retrait d'une face dans une *dcel*

Les opérations de modification d'une *dcel* sont au nombre de deux (si l'on ne prend pas en compte le constructeur) : l'ajout et le retrait de face.

#TODO : ajouter un pseudo-code

L'ajout d'une nouvelle face entraine implicitement la création de trois demi-arêtes (*he*) dans la structure. Leur indice est lui calculé aux lignes 3, 4 et 5. De même, l'indice de leur demi-arête opposée respective (si elle existe) est recherché parmi toutes les demi-arêtes de *dcel* aux lignes 7, 8 et 9. Notez que l'attribut *dcel.available_he_index* est mis à jour dans la structure retournée ligne 29.

2.3 Stockage et manipulation des sous-ensembles de points associés aux faces

3 Évaluation des performances

Lors d'une itération dans l'algorithme de calcul d'enveloppe convexe 3D, deux instructions successives amènent à devoir choisir un élément parmi un ensemble — sachant que tous conduisent à la terminaison du programme et à la construction d'une enveloppe convexe valide. Dès lors l'utilisation d'heuristiques est envisageable, et cette partie porte sur la comparaison des performances en temps de calcul en fonction des méthodes employées.

À la section 3.1 on s'intéresse à la sélection du prochain sous-ensemble de points non-vide (et donc implicitement d'une face du polygone convexe en cours de construction). Tandis qu'à la section 3.2 on s'intéresse à la sélection du point parmi ceux du sous-ensemble choisit précédemment.

Les tests ont été conduits en moyennant le temps de calcul pour 50 échantillons de points générés aléatoirement (répartition spatiale aléatoire uniforme).

3.1 Heuristique sur le choix du point

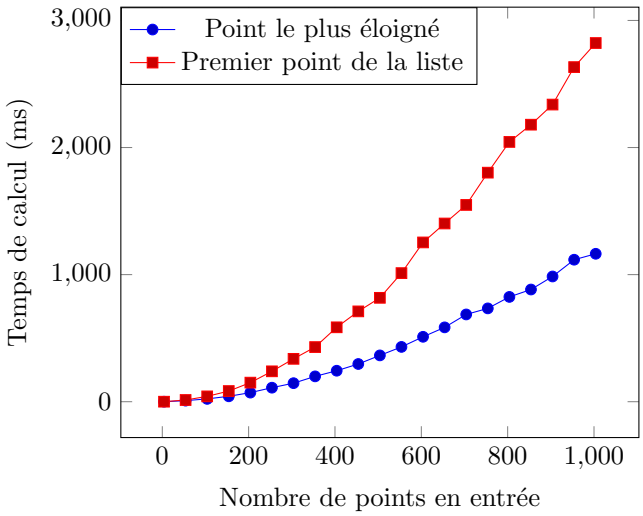

FIGURE 3 – La courbe bleue montre ici que la stratégie consistant à prendre le point le plus éloigné du plan courant offre de meilleures performances en temps que celle où l'on prend naïvement le premier point de la liste (courbe rouge).

3.2 Heuristique sur le choix du sous-ensemble

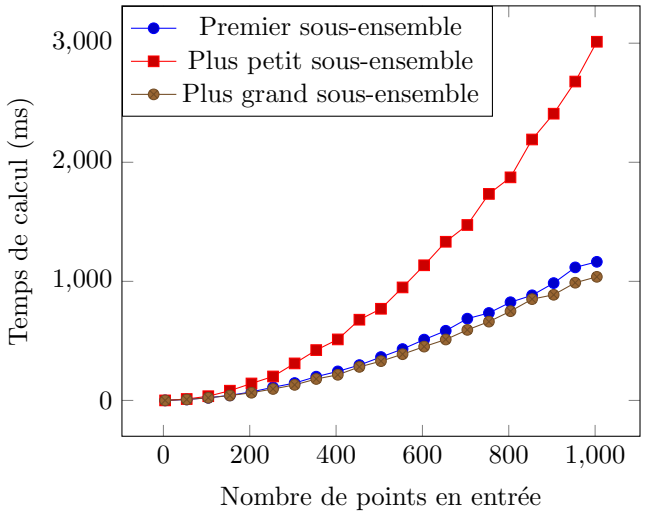

FIGURE 4 – De manière prévisible choisir le plus petit sous-ensemble à chaque itération nuit aux performances de l'algorithme (courbe rouge). En effet en ne prenant à chaque fois que un plus petit sous-ensemble de n points (avec $n > 0$), on se restreint alors à retirer de l'ensemble total n points au mieux à cette itération.
Il est intéressant de noter ici que prendre le plus grand sous-ensemble (courbe brune) est plus performant que de choisir le premier de la liste (courbe bleue), mais le gain n'est pas aussi significatif que ce qu'on a montré précédemment.

2