Implantation fonctionnelle d'algorithmes géométriques en 3D

Titouan Laurent

1 Structures de données implémentées

1.1 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.

```
const vec3 = (x, y, z) =>
  ({
    x:x,
    y:y,
    z:z
  })
;
```

Dans le cas du problème du calcul de l'enveloppe convexe d'un ensemble de n points, on travaille sur un ensemble de prédéfinis. Afin d'éviter toute ambiguïté dans l'identification de points individuels (deux points individuels peuvent partager les mêmes coordonnées spatiales), lesdits points sont stockés dans une liste de taille n, accessible en tant que variable globale du programme. Dès lors, un point est identifiable et accessible via son propre indice dans ladite liste.

1.2 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.

Ici on implante une version plus limitée de cette dernière, puisqu'elle permet de créer maillages triangulaires uniquement — ce qui est sans effet sur les capacités du programme final.

1.2.1 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 :

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

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

$$\begin{cases} opposite = -1 & \text{si la demi-arête n'a pas d'opposé (boundary)} \\ opposite \geq 0 & \text{sinon} \end{cases}$$

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

1.2.2 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.

1.2.3 Exemple simple en 2D

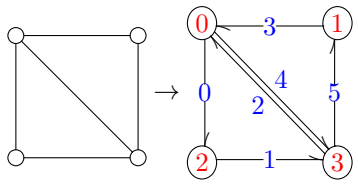

FIGURE 1 – Un maillage simple et un indiçage correspondant possible.

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 table des sommets à droite.

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 plus ou moins arbitraires, la structure *dcel* impose tout de même des contraintes comme nous allons le voir.

1.2.4 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.

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.

```
1  const add_face = (dcel, vert_A, vert_B, vert_C) =>
2  {
3    const he_AB_index = dcel.available_he_index;
4    const he_BC_index = dcel.available_he_index + 1;
5    const he_CA_index = dcel.available_he_index + 2;
6
7    const he_AB_opposite_index = look_up_for_opposite_he_index(dcel, vert_A, vert_B);
8    const he_BC_opposite_index = look_up_for_opposite_he_index(dcel, vert_B, vert_C);
9    const he_CA_opposite_index = look_up_for_opposite_he_index(dcel, vert_C, vert_A);
10
11   return new_dcel(
12     dcel.he_list.map((current_he) =>
13     {
14       switch(he_index(current_he))
15       {
16         case he_AB_opposite_index:
17           return set_he_opposite(current_he, he_AB_index);
```

```
          case he_BC_opposite_index:
            return set_he_opposite(current_he, he_BC_index);
          case he_CA_opposite_index:
            return set_he_opposite(current_he, he_CA_index);
        }
        return current_he;
      })
      .concat(new_he(he_AB_index, he_AB_opposite_index, vert_A))
      .concat(new_he(he_BC_index, he_BC_opposite_index, vert_B))
      .concat(new_he(he_CA_index, he_CA_opposite_index, vert_C)),
      dcel.available_he_index + 3
    );
  };
```

2 Évaluation des performances