Informatique géométrique et graphique : problématique et algorithmes fondamentaux

OIM - TD4

2014

1 Dimensionnement des maillages - Caractéristique d'Euler

Nous rappelons que la caractéristique d'Euler est définie par : $S - A + F = 2(1 - g) = C^{te}$ avec S le nombre de sommet d'un maillage, F son nombre de faces, A son nombre d'arêtes et g son genre (nombre de "trous").

1.1 Construction d'une sphère géodésique

On souhaite construire un maillage pour approcher une sphère par des hexagones. En utilisant la caractéristique d'Euler, justifier pourquoi cette opération est impossible.

1.2 construction d'un dé à 10 faces

On souhaite concevoir un dé à jouer, composé de 10 faces régulières (toutes identiques). en utilisant la caractéristique d'Euler, indiquer combien de sommets seront nécessaire pour la construction de ce dé dans le cas où l'on utilise des faces triangulaires et des faces quadrilatères. Quel est, d'après vous, le dé qui roulera le mieux ?

2 Implantation d'opérateurs sur les maillages

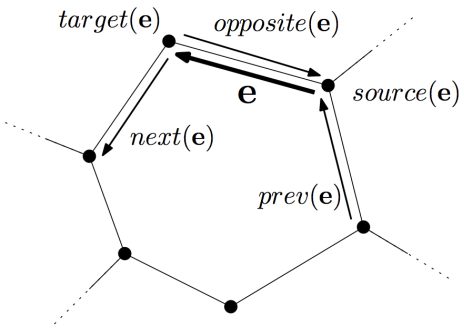

FIGURE 1 – Structure de données par demie-arêtes

On considère la structure de données de maillage, en se limitant aux maillages triangulaires, illustrée sur la figure 1 par demie-arêtes définies par les structures C suivantes :

```
struct Point {
  float x,y,z;
}
```

```
struct HalfEdge {
  struct HalfEdge *prev, *next, *opposite;
  struct Vertex *target;
  struct Face *f:
}

struct Vertex {
  struct HalfEdge *outHalfEdge;
  struct Point p;
}

struct Face {
  HalfEdge *e;
}
```

Chaque structure est munie d'un constructeur de même nom qui prends une liste de paramètres correspondant aux champs de la structure. La structure `HalfEdge` est dotée des opérateurs notés sur la figure 1.

On considère que l'on dispose d'une structure de donnée maillage (`Mesh`), contenant une collection `cPoints` de `Point`, `cHalfEdges` de `HalfEdge`, `cVertices` de `Vertex` et `cFaces` de `Face`. Ces collections sont représentées par un type abstrait de données possédant tous les opérateurs nécessaires à leur gestion : taille de la collection, parcours séquentiel des éléments d'une collection, ajout d'un élément à une collection, ... Dans votre proposition d'opérateurs de traitement, vous pouvez utiliser les opérateurs de gestion de collection qui vous semblent pertinents mais devrez en définir la sémantique lors de la première utilisation.

Exemple : `meshAddVertex(mesh,`Vertex(e, x_0,y_0,z_0)) qui ajoute le sommet de position (x_0,y_0,z_0), rattaché à la demie-arête e à la collection de `Vertex` du maillage et `meshRemoveFace(mesh,`f) qui enlève la face f de la collection de `Face` du maillage).

2.1 Implantation de l'opérateur de bascule d'arête

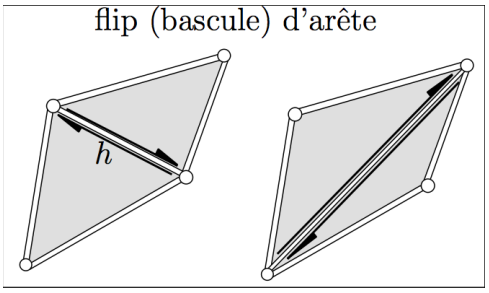

FIGURE 2 – Opérateur de bascule d'arete

Donner le pseudocode C de l'opérateur de bascule d'arête illustré sur la figure 2. Il est très fortement conseillé de dessiner clairement les modifications qui devront être apportées sur les arêtes et les faces avant d'écrire le pseudocode.

2.2 Construction de la liste de sommet du premier anneau de voisinage

Outre les opérations élémentaires vues en cours de traitement d'une géométrie, l'opération de construction du premier anneau de voisinage d'un sommet est une des opérations les plus fréquentes sur un maillage. Le premier anneau de voisinage d'un sommet c est défini par la liste ordonnée des sommets s appartenant à toutes les faces dont un des sommets est c. Dans cet exercice, nous considérons que nous implantons cet opérateur pour des maillages triangulaires représentés par une structure en demies-arêtes.

1. Illustrer par un schéma ce que doit être le résultat de l'opérateur pour un maillage triangulaire quelconque. Faites apparaitre sur ce shéma les différentes composantes du maillage : $Vertex$ (notés c et s_i)), $HalfEdge$ (notées h_i) et $Face$ (notées f_i)

2. En utilisant un Type abstrait de données *Queue* définissant une liste, proposer le pseudo-code C de l'opérateur dont le profil est :

```
Queue firstRingNeighborhood(Vertex *c);
```

3. Une représentation possible des maillages, utilisées dans les API de visualisation OpenGL et DirectX repose sur un tableau *vertices* de *Point* et un tableau *triangles* de *Triangle*. Dans cette représentation, un *Triangle* est défini par les 3 indices de ses sommets dans le tableau *vertices*. Si l'on connaît un sommet par son indice c dans le tableau *vertices*, donner l'algorithme général de l'opérateur

```
Queue firstRingNeighborhood(int c);
```

à partir de ce tableau.

4. Si on applique ces opérateurs sur les N sommets d'un maillage, estimer la complexité en temps des deux approches et, pour les maillages triangulaires réguliers ne possédant que des sommets de valence 6, indiquer la taille du maillage entrainant une différence d'un facteur 1000 entre les deux approches (passage de $1s$ à $20mn$ de traitement).

2.3 Triangulation d'un maillage polygonal

On considère un maillage définit par des polygones convexes ayant au plus 5 cotés. Le type des polygones d'un maillage est défini par le type énuméré suivant :

```
enum PolygonType {TRIANGLE=0, QUAD=1, PENTAGON=2};
```

1. Ecrire le code C de l'opérateur

```
PolygonType getType(Face *f);
```

qui détermine le type du polygone décrit par la face `f`.

2. Lors de la transformation d'une `Face` non triangulaire en un ensemble de triangles, afin d'optimiser la gestion mémoire, nous souhaitons minimiser le nombre de `Face` créées. Pour cela, les opérateurs de triangulation d'un polygone devront modifier la structure du maillage en ajoutant aux collections `cHalfEdges` et `cFaces` un nombre minimum de données en réutilisant les `Face` et `HalfEdge` existantes dans le maillage. Les `Face` et `HalfEdge` initiales ne seront pas supprimées des collections mais modifiées de façon à ce que le maillage reste cohérent d'un point de vue topologique et géométrique. A partir d'un schéma décrivant leur fonctionnement, écrire le code C des opérateurs suivants :

```
void transformQuad(Face *q, Mesh *m);

void transformPentagon(Face *p, Mesh *m);
```

3. En utilisant les opérateurs précédents, écrire le code C de l'opérateur de transformation d'un maillage hétérogène en un maillage homogène triangulaire :

```
void transformMesh(Mesh *m);
```

4. On souhaite maintenant pouvoir généraliser les opérateurs de triangulation pour des polygones plans et convexes à n côtés. Donner le code C de l'opérateur général de triangulation

```
void transformFace(Face *f, Mesh *m);
```

qui triangule tout polygone convexe sans en connaitre le type a-priori.