Géométrie algorithmique - T.P.

12 novembre 2012

Il existe de nombreux algorithmes qui calculent la triangulation de Delaunay d'un ensemble $\mathcal{P}$ de n points du plan. Le but du T.P. est d'implanter en C++ et de tester un de ces algorithmes. On vous fournit le squelette de l'algorithme ainsi que les classes nécessaires : `Point`, `PointSet`, `Triangle`, `Triangulation`.

1 Algorithme de Guibas[1], Knuth[2] et Sharir[3]

Cet algorithme [1] a été proposé en 1992 et est de complexité moyenne $O(n \log n)$, pour une place mémoire en $O(n)$. L'algorithme est détaillé page 3 (page extraite de l'article d'origine). Il commence par construire un grand triangle à partir de trois points Ω_1, Ω_2 et Ω_3 à l'infini, puis insert successivement les points dans ce triangle en modifiant la triangulation courante. Pour cela il remplace à chaque étape le triangle qui contient le nouveau point P par trois triangles de sommet P, puis modifie ces triangles et leurs voisins par *flip* d'arête dans le cas où ces triangles ne sont pas de Delaunay (voir Figure 4.4 du poly). Afin de trouver efficacement dans quel triangle se trouve le point P, les auteurs proposent d'utiliser un graphe orienté sans cycle (DAG) de triangles : la racine de ce graphe est le triangle $\Omega_1\Omega_2\Omega_3$, et à chaque insertion ou *flip* on crée un ou des nouveau(x) nœud(s) correspondant au(x) nouveau(x) triangle(s) créé(s). Un nouveau nœud aura pour parent(s) le(s) triangle(s) à partir duquel ou desquels il a été créé (voir Figure 4.5 du poly).

Cet algorithme contient deux difficultés d'implantation :
- créer une structure de données de DAG et l'utiliser proprement et efficacement (étapes 2.b et 2.c) ;
- parcourir les triangles et faire les *flips* d'arêtes éventuels (étape 2.d).

2 Travail demandé

Complétez le code fourni afin de terminer l'implantation de l'algorithme. Vous pouvez travailler en binôme afin que chacun se concentre sur une des deux difficultés. Vous pouvez modifier tout ou partie du code fourni.

3 Tests et visualisation

Pour tester votre algorithme, vous utiliserez des fichiers au format `.node` et vous générerez des fichiers au format `.ele`. Ces fichiers peuvent ensuite être lus par l'outil de visualisation `showme` [2] de Jonathan Shewchuk [3]. Trois fichiers au format `.node` vous sont fournis, mais vous pouvez créer les votres.

Description du format `.node` La première ligne ne comporte qu'un entier, qui est le nombre n de sommets de $\mathcal{P}$. Elle est suivie par n lignes, chacune comportant un entier et deux flottants, qui sont respectivement le numéro (l'identifiant) du sommet et ses coordonnées.

Description du format `.ele` La première ligne ne comporte qu'un entier, qui est le nombre m de triangles de la triangulation. Elle est suivie par m lignes, chacune comportant quatre entiers, qui sont respectivement le numéro (l'identifiant) du triangle et les numéros de ses sommets.

Un petit exemple

Fichier `.node`			Fichier `.ele`			
5			4			
1	0.14	0.913	1	1	2	3
2	0.323	0.724	2	2	3	5
3	0.213	0.23	3	3	4	5
4	0.684	0.323	4	2	4	5
5	0.354	0.34				

Bibliographie

[1] Leonidas Guibas, Donald Knuth, Micha Sharir. *Randomized Incremental Construction of Delaunay and Voronoi Diagrams.* Algorithmica, vol. 7, p. 381-413, 1992.
[2] `http://www.cs.cmu.edu/~quake/showme.html`
[3] `http://www.cs.berkeley.edu/~jrs/`

[1] `http://en.wikipedia.org/wiki/Leonidas_J._Guibas`
[2] `http://en.wikipedia.org/wiki/Donald_Knuth`
[3] `http://en.wikipedia.org/wiki/Micha_Sharir`

to the three vertices of Δ, and test each edge e of Δ as to whether it is still a valid Delaunay edge. For this it suffices to test whether P lies in the circumcircle of the triangle Δ' lying on the other side of e. If not, no update of the triangulation beyond e is necessary. Otherwise we delete e, connect P to the third vertex of the triangle Δ', and examine the two other edges of Δ' for validity. We continue this way until all edges that we encounter are valid, and then we stop. We remark that if P lies outside the convex hull of the first j points, the above triangle-flipping procedure requires some modifications. We can handle this issue by adding to $\mathscr{P}$ three dummy points "at infinity," Ω_1, Ω_2, Ω_3, whose spanning triangle contains all sites in $\mathscr{P}$, and by starting the incremental construction with the triangle $\Omega_1\Omega_2\Omega_3$. It is easily verified that the bounds derived in the preceding section do not change asymptotically when the construction is modified in this manner.

Thus we can write a high-level description of the algorithm as follows:

```
1. ⟨ Initialize the triangulation to the single triangle Ω1Ω2Ω3 ⟩;
2. for k ← 1 to n do
   a. begin ⟨ Select a random point P that has not previously been selected ⟩;
   b. ⟨ Find the triangle ABC containing P ⟩;
   c. ⟨ Replace ABC by the three triangles PAB, PBC, PCA ⟩;
      X ← A;
   d. repeat ⟨ Let Y be the third vertex of the triangle to the right of PX, in
                the current triangulation ⟩;
         progress ← true;
         if X and Y not both infinite then
            begin ⟨ Find Z ≠ P such that ZYX is in the current triangula-
                    tion ⟩;
            if in(P, X, Y, Z) then
               begin ⟨ Flip triangles PXY and ZYX to obtain PXZ and
                       PZY ⟩;
               progress ← false;
               end;
            end;
         if progress then X ← Y;
      until X = A and progress;
   end.
```

Here the test in(A, B, C, D) is true if D is on the left of the oriented circle that goes from A to B to C to A. In terms of coordinates,

$$\text{in}(A, B, C, D) \quad \Leftrightarrow \quad \det\begin{pmatrix} x_A & y_A & x_A^2 + y_A^2 & 1 \\ x_B & y_B & x_B^2 + y_B^2 & 1 \\ x_C & y_C & x_C^2 + y_C^2 & 1 \\ x_D & y_D & x_D^2 + y_D^2 & 1 \end{pmatrix} > 0.$$

If one or two of A, B, C, D is infinite, an appropriate limiting determinant should be calculated. A discussion of this and similar tests, as well as a proof of the