Sujets des projets (version 0)
Informatique de Base MM009
Université Pierre et Marie Curie

F. Hecht, X. Claeys, G. vergez

Master I / session 1/S1, 2015/2016

Table des matières

Introduction

Pour tous le projet , il faut faire un rapport sous une forme de pages web au format `html` d'une quinzaine de pages ou d'un document `Latex`.

Les Rapports et Programmes et un fichier explication sont a envoyer par mèl à `mailto:frederic.hecht@umpc.fr`, et à `mailto:claeys@ann.jussieu.fr`. Tous ces fichiers sont à envoyer dans une archive `tar` compressé de nom : `votrenom-mm009.tar.gz`

Ces projets sont a réaliser par binôme pour les étudiants de **M1**, le choix des projets est faite par les enseignants, une fois le binôme déclarer.

1 Sujet : Recherche rapide d'un triangle contenant un point dans un maillage

Un maillage conforme formé d'un ensemble de n_T triangles $\mathscr{T}_h$ telle que l'intersection de 2 triangles distincts soit une arête commune, un sommet commun ou rien, où les l'ensemble des n_S sommets de $\mathscr{T}_h$ est noté $\mathscr{S}_h$.

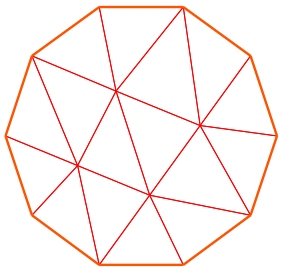


FIGURE 1 – exemple de maillage

Le but est d'écrire un algorithme de recherche d'un triangle K dans un maillage convexe $\mathscr{T}_h$ contenant un point (x,y) en $O(log_2(n_T))$.

1.1 Projet 1 (promenade)

Algorithme 1 *Partant du triangle K,*
Pour les 3 arêtes $(a_i,b_i), i = 0,1,2$ du triangle K, tournant dans le sens trigonométrique, calculer l'aire des 3 triangles (a_i,b_i,p). Si les trois aires sont positives alors $p \in K$ (stop), sinon nous choisirons comme nouveau triangle K l'un des triangles adjacents à l'une des arête associée à une aire négative (les ambiguïtés sont levées aléatoirement).

1. Construire une classe maillage formée d'un tableau de sommets et d'un tableau de triangles, qui lit le fichier maillage.
2. Construire une méthode dans la classe maillage donnant le triangle K' adjacent d'un triangle K opposé à son sommet $i \in \{0,1,2\}$. La complexité de cette fonction doit être constante après bien sur une unique initialisation en $O(n_T)$ ou $O(n_T log_2(n_T))$.
   Pour cela vous pourrez utiliser le chapitre chaîne et chaînage.
3. Programmez l'algorithme 1 en partant d'un triangle quelconque (ajoutez une nouvelle méthode de la classe maillage).
4. Puis trouver pour chaque sommet d'un autre maillage donné du même domaine, le triangle le contenant. Afin de briser la complexité algorithmique, il faut utiliser l'assertion suivante : les sommets d'un triangle sont proche car les triangles sont petits.

1.2 Projet 2

Programmation d'une méthode pour trouver, rapidement un triangle proche d'un point donné en $O(log(n))$
Pour cela nous utiliserons, un arbre quaternaire pour stocker les sommets dans maillages comme suit :

Algorithme 2

Soit B_0 une boite carre contenant tous les points du maillage. On applique la procédure récursive suivante à B_0.
Soit B_α la boite indexée par α qui est une chaîne de 0,1,2 ou 3 (exemple $\alpha = 011231$) la longueur de la chaîne est note $l(\alpha)$.
procedure appliqué à la boite B_α :

- *la boite B_α contient moins de 5 points, on les stockes dans la boite et on a fini,*
- *sinon on découpe la boite B_α en 4 sous boites égale noté $B_{\alpha 0}, B_{\alpha 1}, B_{\alpha 2}, B_{\alpha 3}$ (on stocke les 4 pointeurs vers ces boites dans la boite), et on applique la procédure au 4 nouvelles boites.*

Remarque : pour simplifier la programmation de l'arbre quaternaire, on peut construire une application qui transforme le point de $\mathbb{R}^2$ en points à coordonnées entières en construisant une application affine A qui transforme par exemple B_0 en $[0, 2^{31}[^2$. Dans ce cas, les coordonnées entières permettent de limiter la profondeur de l'arbre à 32 niveaux, et d'utiliser les opérateurs entiers en nombre binaire du C comme `&` , `|` , `^` .

Quelque remarque :

- décomposition binaire, soit i un nombre entier il peut s'écrire comme $\sum b_k^i 2^k$, avec $b_k^i \in \{0,1\}$ et b_k^i est appelé le k ieme bit de i.
- `( 1 << n)` est égal à 2^n pour $n = 0, ..., 31$
- `( i & ( 1 << n))` est vrai si le n ieme bit de i est égal à 1 pour $n = 0, ..., 31$
- `(( i >> n) & 1)` est égal à b_n^i le n ieme bit de i.
- La numérotation du découpage en 4 boites d'un niveau k sera numéroté par N_d :

$$(b_i, b_j) \in \{0,1\}^2 \mapsto \{0,1,2,3\} : \qquad N_d(b_i, b_j) = b_i + 2b_j.$$

- La boite b_α (si elle existe) de niveau n contenant un point de coordonnées (x,y) aura pour $\alpha = c_1 ... c_n$ avec $c_k = N_d(b^i_{32-k}, b^j_{32-k})$ pour $k = 1, .., n$ où les coordonnées entières associées sont $(i,j) = A(x,y)$ et où b^i_{32-k} (resp. b^j_{32-k}) est le (32-k) ieme bit de i (resp j).

L'algorithme est :

1. Construire une classe maillage formé d'un tableau de sommets et d'un tableau de triangle, qui lit le fichier maillage.
2. Construire l'arbre quaternaire contenant les barycentres des triangles du maillage.
3. Trouver la boite non vide la plus petit $B_{\alpha(p)}$ de l'arbre contenant un point p.
4. Nous choisirons comme triangle de départ K , l'un des triangles associées l'un des barycentres de la boite $B_{\alpha(p)}$.
5. Construire une méthode dans la classe maillage donnant le triangle K' adjacent d'un triangle K opposé à son sommet $i \in \{0,1,2\}$. La complexité de cette fonction doit être constante après bien sur une unique initialisation en $O(n_T)$ ou $O(n_T log_2(n_T))$.
   Pour cela vous pourrez utiliser le chapitre chaîne et chaînage.
6. Programmez l'algorithme 1 en partant du triangle K_s.

2 Sujet : Algorithme de Maillage de Delaunay

But : générer la triangulation de Delaunay, d'un ensemble de ,N_p points de $[0,1]^2$ crées aléatoirement.
Voilà deux algorithmes

Algorithme 3

1. *Trier les x^i avec l'ordre σ par rapport à la norme de x^i. Cette ordre est telle que le point courant $x^{\sigma(i)}$ soit à l'extérieur du convexifié des points précédents $\{x^{\sigma(j)}/j<i\}$.*
2. *Ajouter les points $x^{\sigma(i)}$ un à un suivant l'ordre σ pré-défini par l'étape 1. Les points ajoutés sont toujours à l'extérieur du maillage courant. Donc on peut créer un nouveau maillage du convexifié. Pour cela, il suffit de construire pour chaque arête (x_a,x_b) frontières orienté (le convexe est à gauche de l'arête), le triangle $(x_a,x^{\sigma(i)},x_b)$ si $(det(x_b-x^{\sigma(i)},x_a-x^{\sigma(i)}>0$ (cad. la arête (x_a,x_b) est visible du point $x_\sigma(i)$.*
3. *Pour rendre le maillage de Delaunay, il suffit d'appliquer la méthode* **optimisation** *du maillage autour du points $x^\sigma(i)$ en rendant toutes les motifs formés de 2 triangles contenant le point $x^\sigma(i)$ de Delaunay, et cela pour tous les points $x^k(i)$ du maillage juste après son insertion.*

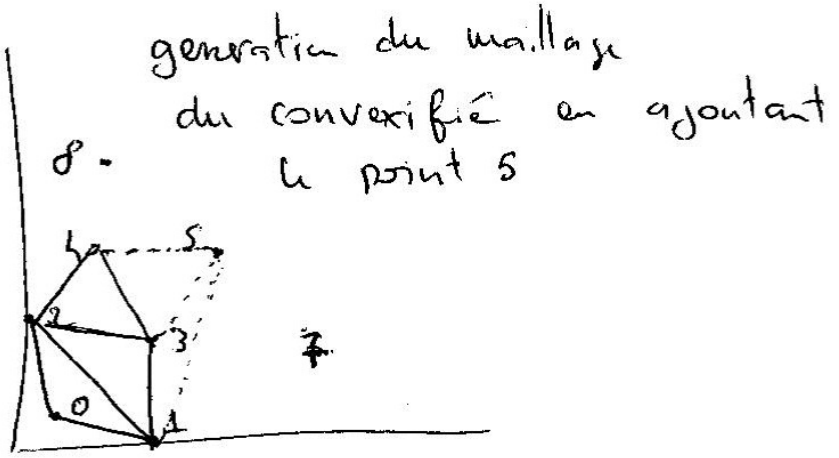


FIGURE 2 – Répresentation graphique de l'algorithme 3

Algorithme 4

1. *Generer un maillage formé d'un triangle, qui contiendra tous les points du maillage future.*
2. *Ajouter les points N_p un à un suivant un ordre aléatoire.*
   Les points ajoutés sont toujours dans un ou deux triangles. Trouvez ces triangles en utilisant l'algorithme 1 et puis découpez le triangle en 3 triangle ou le quadrangle en 4 triangles.
3. *Pour rendre le maillage de Delaunay, il suffit d'appliquer la méthode* **optimisation** *du maillage autour du points x^k en rendant toutes les motifs formés de 2 triangles contenant le point x^k de Delaunay, et cela pour tous les points x^k du maillage juste après son insertion.*

2.1 Méthode optimisation

Nous ferons un d'échange de diagonale $[s_a,s_b]$ dans un quadrilatère convexe de coordonnées s_1,s_a,s_2,s_b (tournant dans le sens trigonométrique) si le critère de la boule vide n'est pas vérifié comme dans la figure 4.

Le Critère de la boule vide dans un quadrilatère convexe s_1,s_a,s_2,s_b en $[s_1,s_2]$ est équivalent à l'inégalité angulaire (propriété des angles inscrits dans un cercle) :

$$\widehat{s_1s_as_b} < \widehat{s_1s_2s_b}$$

ce qui équivalent au le critère d'échange de diagonale optimisé

$$det(\overrightarrow{s_2s_1},\overrightarrow{s_2s_b})\quad(\overrightarrow{s_as_1},\overrightarrow{s_as_b}) \quad > \quad det(\overrightarrow{s_as_1},\overrightarrow{s_as_b})\quad(\overrightarrow{s_2s_1},\overrightarrow{s_2s_b}), \tag{1}$$

ou $(.,.)$ est produit scalaire de $\mathbb{R}^2$.

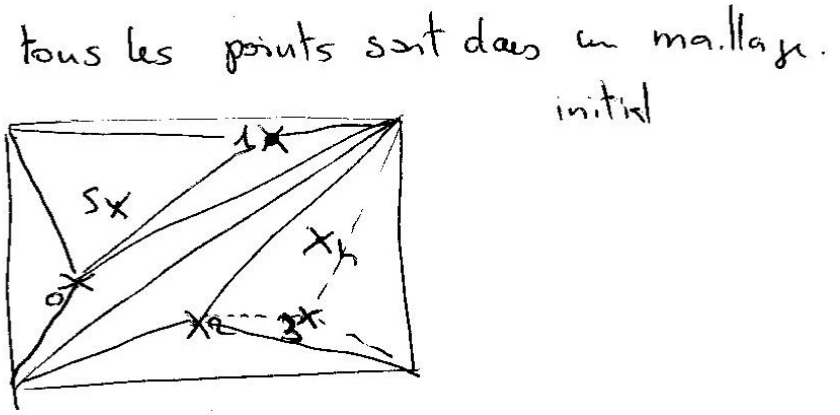


FIGURE 3 – Répresentation graphique de l'algorithme 4, où tous les points sont mis dans un rectangle.

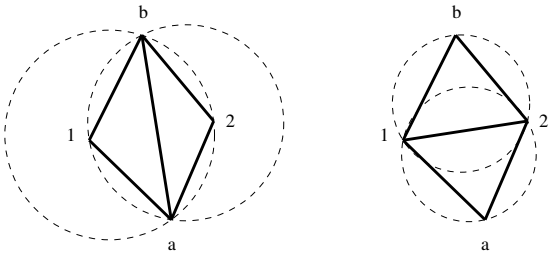
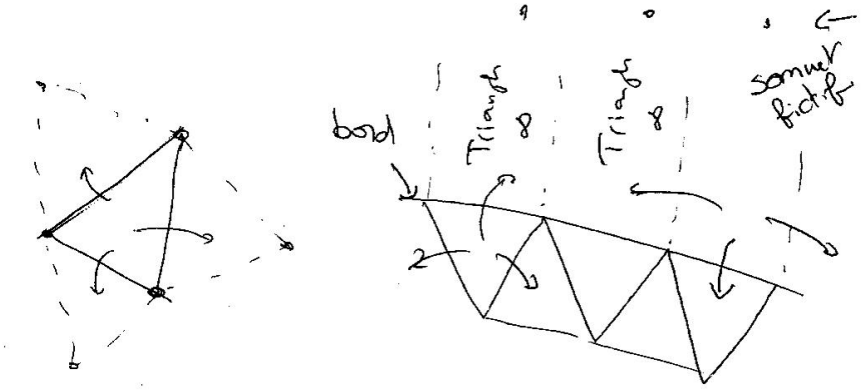


FIGURE 4 – Échange de diagonale d'un quadrilatère convexe selon le critère de la boule vide

2.2 Structure de données

Pour programmer ces algorithmes, il faut modéliser soit :

M-1) un triangle et connaitre ses voisins, de plus, il est nécessaire d'avoir la liste des arêtes frontières dans l'algorithme 3, pour cela nous introduiront un triangle fictif qui a un sommet à l'infini pour chaque arête frontière, et la arete frontiere seront juste l'ensemble de d'arête des triangles tournant autour du sommet fictif.

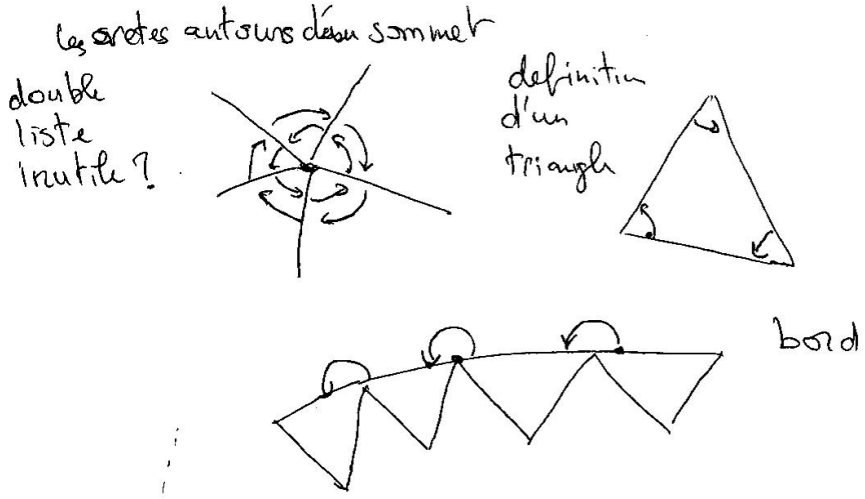


La classe Triangle pourra être du type (cette classe n'est pas testé, donc un il a des erreurs) et ce ne sont que des indications.

```
class Triangle;
class Sommet : public R2 {
 Triqngle * t;                              //    pointeur sur un triangle contenant se sommet
 ...
};
```

```
class Triangle  {
  Sommet *sommets[3];                          //    tableau de trois pointeurs de type Sommet
  Triangle *tadj[3];                          //    tableau de trois pointeurs de type Triangle
  char   nuatadj[3];                              //    numero des arete dans les triangle adj
public:
  Triangle(){};                                                  //    constructeur par défaut vide

...
 bool EstFictif ()                                    //    si l'un de sommet est un pointeur null
   { return! (sommets[0] && sommets[2] &1 sommets[3]);}

 .....

private:
  Triangle(const Triangle &);                          //    interdit la construction par copie
  void operator=(const Triangle &);                      //    interdit l'affectation par copie
};                                                                 //    fin de la class Triangle


class Triangulation { public:
  int nt,nv;
  int ntmaximal,nvmaximal;

  Sommet *sommets;                                                    //    alloue a nvmaximal
  Triangle *triangles;                                                //    alloue a ntmaximal

  Triangle & operator[](int i) const {return triangles[CheckT(i)];}
  Sommet & operator()(int i) const {return sommets[CheckS(i)];}

  inline Triangulation(int nvx) : ntmaximal(nvx*2), nvmaximal(nvx) , nt(0),nv(0) {}

  .....

                                                                 //    voila quelque fonction utile
  void swap(Triangle &t,int arete);
  void DecoupeEn3Triangle(Triangle &t,Sommet * v);
  void DecoupeEn4Triangle(Triangle &t,Sommet * v);
  void Optim(Triangle &t,int som);
....

private:
  Triangulation(const Triangulation &);                //    interdit la construction par copie
  void operator=(const Triangulation &);                 //    interdit l'affectation par copie
};
```

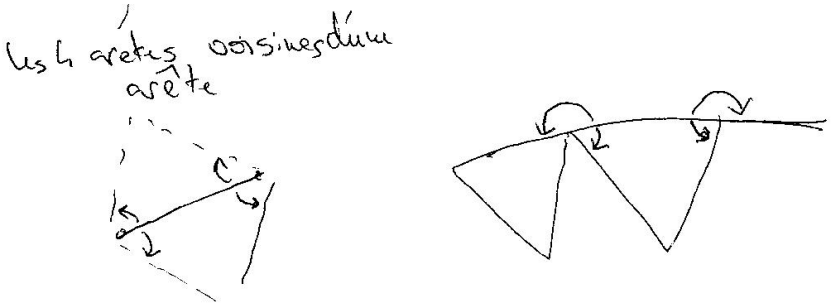
M-2) Les sommets et la liste de sommets voisins tournant dans le sens trigonométrique ;




```
inline R Angle(Sommet *b,Sommet *b);
```

```
class SommetetAngle{ public:
Sommet *v;
R angle a;
SommetetAngle(Sommet *vv,Sommet *from) : v(vv),a(Angle(*from,*vv)){}
bool operator<(const SommetetAngle & A) const { return a< A.a;)}
};

typedef set< SommetetAngle> LSommet;

typedef LSommet::iterator LSiterator;



class Sommet: public R2 { public:
  int num;                                                    //    le numero du sommet pour debug
   LSommet  l;                                                //    la liste des sommets voisin
                                                  //    tournant autour du sommet dans le sns trig
                                        //    remarque, cela défini bien sur implicitement les arêtes

  Sommet() : R2(),num(-1), l() {}
  void Set(R xx,R yy,int n) {x= xx;y=yy;num=n;}

  LSiterator  Trouver(Sommet * pe);
  void Erase(Sommet  * e);

  void Insert(Sommet * e);

  int Delaunay();

  void gnuplot(ostream & f,int debug=0);
  friend ostream & operator<<(ostream & f, const Sommet & A);
  void show(ostream &f);

private:                                                                  //    pas de copie
  Sommet(const Sommet &);
  void operator=(const Sommet &);
};

inline R Angle(Sommet *a,Sommet *b) {
  R2 (AB(*a,*b);
  return atan2(AB.y,AB.x);
}

class Triangulation { public:
  int nvmaximal;                                          //    nombre maximal de sommet
  int nv;
  Sommet *sommets;
  Arete  *aretes;
  list<Sommet *> thebord;

  Triangulation(int nvmaxi);                                                        //

  int operator()(const Sommet & v) const {return CheckS(&v - sommets);}
  int operator()(const Sommet * v) const {return CheckS(v - sommets);}
  Sommet & operator()(int i){ return sommets[CheckS(i)];}
                                                                  //    to check the bound
  int CheckS(int i) const { ASSERTION(i>=0 && i < nv); return i;}

  void gnuplot(const string &filename,int debug=0);
  void gnuplotFrontiere(const string &filebame);
  void write(const string & );
private:
  Triangulation(const Triangulation &);                    //    pas de construction par copie
  void operator=(const Triangulation &);                   //    pas affectation par copy

  void InitBord();
  void InsertVertex(int i);
  void BuildMesh();
  void BuildVertex();
};
```

M-3) une arête du maillage et ces 4 arêtes voisines comme sur la figure suivante




```
class Arete  {
  Sommet *sommets[2];                                  //    tableau de trois pointeurs de type Sommet
  Arete * adj[2][2]                                      //    tableau des 2x2 aretes adjactente
  //    adj[i][j] est pour i: sommet 0 ou 1 et j: 0) avant : 1) apres en tournant
public:
  Arete(){};                                                                  //    constructeur par défaut vide

  Sommet & operator[](int i) const {
    ASSERTION(i>=0 && i <2);
    return *sommets[i];}                          //    évaluation du pointeur -> retourne un sommet

  void Set( ..... ) {
 ......
  }


private:
  Arete(const Arete &);                                 //    interdit la construction par copie
  void operator=(const Arete &);                          //    interdit l'affectation par copie
};                                                                          //    fin de la class Arete


class Triangulation { public:
  int ne,nv;
  int nemaximal,nvmaximal;

  Sommet *sommets;                                                          //    alloue a nvmaximal
  Arete *aretes;                                                            //    alloue a ntmaximal

  Arete & operator[](int i) const {return Arete[CheckA(i)];}
  Sommet & operator()(int i) const {return sommets[CheckS(i)];}

  inline Triangulation(int nvx) : nemaximal(nvx*3), nvmaximal(nvx) , nt(0),nv(0)
   sommets(new Sommet[nvmaximal]), aretes(new Arete[nemaximal])  {}

  .....


private:
  Triangulation(const Triangulation &);                 //    interdit la construction par copie
  void operator=(const Triangulation &);                  //    interdit l'affectation par copie
};
```

Avec l'une de c'est donné, il est possible de contruire une fichier maillage.

Dans ces 6 projets suivant, une étude de la complexité des algorithmes est demandée et une vérification numérique est souhaité. Une option de visualisation pas à pas pour comprendre de déroulement du programme est impérative. Pour finir, il faut écrire un fichier maillage du type decrit dans l'annexe 8.

2.3 Projet 3 : Algorithme 3 / Modéle M-1

Définir des classe `triangle`, `Sommet`, `Triangulation` qui permettent de retrouver les 3 triangles adjacent a un triangles en $O(1)$ opération.

2.4 Projet 4 : Algorithme 3 / Modéle M-2

Définir des classe `Sommet`, `Triangulation` tel que pour chaque sommet, il soit possible de trouver l'arête suivante ou précédant en tournant autour de sommet en $O(1)$ opération.

Remarque , les triangles ne seront connus que «virtuellement» (construit au vol) à partir d'un angle et d'un sommet. Pour éviter les doublons de triangles, il suffit de remarquer qu'un triangle à toujours un sommet de plus petit numéro et donc de construit le triangle que si le numéro du sommet est plus petit que les autres.

Puis a la fin reconstruire les triangles pour un fichier maillage `.msh`.

2.5 Projet 5 : Algorithme 3/ Modéle M-3

Définir des classes `Arete`, `Sommet`, `Triangulation` tel que pour chaque arête, l'on connaîtra les 4 arêtes voisines.

Remarque , les triangles ne seront connus que «virtuellement» (construit au vol) à partir d'une arête et d'une orientation. Pour éviter les doublons de triangles, il suffit de remarquer est formé de 3 arêtes et comme précédemment, il suffit de construit le triangle que si le numéro d'arête est plus petit que les autres pour éviter les doublons.

Puis a la fin reconstruire les triangles pour un fichier maillage `.msh`.

2.6 Projet 6 : Algorithme 4 / Modéle M-1

Définir des classe `triangle`, `Sommet`, `Triangulation` qui permettent de retrouver les 3 triangles adjacent a un triangles en $O(1)$ opération.

2.7 Projet 7 : Algorithme 4 / Modéle M-2

Définir des classe `Sommet`, `Triangulation` tel que pour chaque sommet, il soit possible de trouver l'arête suivante ou précédant en tournant autour de sommet en $O(1)$ opération.

Remarque , les triangles ne seront connus que «virtuellement» (construit au vol) à partir d'un angle et d'un sommet. Pour éviter les doublons de triangles, il suffit de remarquer qu'un triangle à toujours un sommet de plus petit numéro et donc de construit le triangle que si le numéro du sommet est plus petit que les autres.

remarque : il faut quelque peut changer l'algorithme de promenade 1 car nous avons pas de triangles mais des sommets

Algorithme 5

Partant d'un sommet s,
trouver les 2 arêtes concécutive (s,a) et (s,b) du sommet s (où a et b sont les 2 autres sommets des arêtes), tournant dans le sens trigonométrique, tel que les aires signées des 2 triangles $A = aire(a,p,s)$ et $B = aire(b,s,p)$ soit positive.
Soit l'aire du triangle $C = aire(a,b,s)$ si C est négatif ou nul alors le point est à l'exterieur, si $C-A-B$ est positif alors on a trouver le triangle sinon choisir aléatoirement le nouveau point s entre les points a et b et continuer.

Puis à la fin reconstruire les triangles pour un fichier maillage `.msh`.

2.8 Projet 8 : Algorithme 4/ Modéle M-3

Définir des classes `triangle`, `Arete`, `Triangulation` tel que pour chaque arête, l'on connaîtra les 4 arêtes voisine.

Remarque , les triangles ne seront connus que «virtuellement» (construit au vol) à partir d'une arête et d'une orientation. Pour éviter les doublons de triangles, il suffit de remarquer est formé de 3 arêtes et comme précédemment, il suffit de construit le triangle que si le numéro d'arête est plus petit que les autres pour éviter les doublons.

remarque : il faut quelque peut changer l'algorithme de promenade 1 car non avons pas de triangles mais seulement des arêtes :

Algorithme 6

Partant d'une arête e de sommet (a,b). On peut supposer que p est a gauche de (a,b) pour cela échangé a et b si l'aire signée du triangle (a,b,p) est négative.
Soit $(a,c1)$ et $(b,c1)$ les deux arêtes du coté de p. Si $c1 \neq c1$ alors le point p est exterieur, sinon $c1 = c2$ que l'on notera c alors l'arête est interne, il faut choisir l'arête suivante dans le triangle (a,b,c) si nécessaire avec le même méthode que dans l'algorithme 1.

Puis a la fin reconstruire les triangles pour un fichier maillage `.msh`.

3 Visualisation et optimisation de maillage

Le but de ce projet de visualiser une fonction f du carré $]-1,1[^2$ à valeur dans $\mathbb{R}$ avec une précision de ε donnée. Il faut donc construire un maillage telle que l'erreur L^2 sur chaque triangle du maillage soit inférieure à ε.

L'algorithme de raffinement de maillage est très simple : il consiste à couper les triangles d'erreur trop grand en deux parties égales par rapport à une des 3 arêtes en choisissant l'arête qui générera l'erreur minimal.

Donc l'algorithme est :

1. insérer les triangle avec une erreur trop grande la queue
2. tant que la queue n'est pas vide faire :
   (a) Prendre le triangle de la queue et découper le triangle en deux en choissisant la bonne arête, et ajouter les nouveaux triangles dans la queue si nécessaire.

L'erreur sur un triangle K sera définie par :

$$E_K = \int_K |f - \Pi_K(f)|^2$$

Où $\Pi_K(f)$ est la projection $L_2(K)$ de f sur l'espaces $P_1(K)$ des fonctions polynomes de degre 1 de K a valeur dans $\mathbb{R}$, c'est-à-dire :

$$\int_K \Pi_K(f) - f = 0; \qquad \int_K (\Pi_K(f) - f)x = 0 \qquad \int_K (\Pi_K(f) - f)y = 0$$

Et vous utiliserez des formules d'integration pour évaluer les intégrales qui sont définies dans `http://arxiv.org/abs/math/0501496` Several new quadrature formulas for polynomial integration in the triangle de Mark A. Taylor, Beth A. Wingate, Len P. Bos. les sources sont dans `http://arxiv.org/e-print/math/0501496v2`

Remarque sur les formules intégration sur un triangle $K = (A,B,C)$ soit F_K la transformation affine de $\hat{K} = ((0,0),(1,0),(0,1))$ dans K qui est défini par :

$$F_K : (\hat{x},\hat{y}) \mapsto (1-\hat{x}-\hat{y})A + \hat{x}B + \hat{y}C$$

Les N points $\{\hat{z}_1, \hat{z}_2, \ldots, \hat{z}_N\}$ défini sur le triangle $\hat{K}$ ry les poids associés $\{w_1, w_2, \ldots, w_N\}$ sont définis dans le fichier `coords.txt` pour une formule integration à l'ordre d. La formule de quadrature sur K est défini par

$$\int_K g \simeq \frac{|K|}{2} \sum_{j=1}^{N} w_j \, g(F_K(\hat{z}_i)),$$

Cette formule est exact pour les fonctions g qui sont des polynômes de degré d.

3.1 Projet 9

Programmer l'algorithme precédent pour visualiser les fonctions

$$\begin{aligned} f_1(x,y) &= x^2 + y^2 \\ f_2(x,y) &= x^2 + y^3 + tanh(5sin(2(y+x)) \end{aligned}$$

avec la bibliothèque `GLUT` où le paramètre ε peut être changé interactivement, et bien sur il doit être possible d'afficher ou non le maillage, et la fonction sera défini via une chaine de caractères et sera évalué avec interpréteur de formule du cours.

4 Projet 10 : Optimisation d'un trajet sur une topographie quelconque

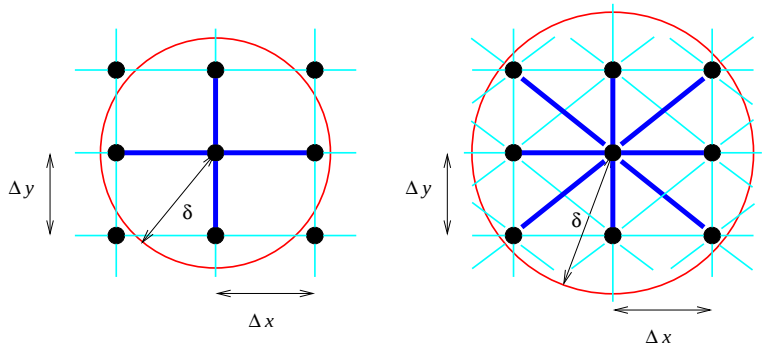
Le principe est d'optimiser le trajet d'un véhicule se déplaçant d'un point à un autre sur un terrain ayant une topographie quelconque. Cette optimisation devra être réalisée en minimisant deux quantités : la distance entre le point de départ et le point d'arrivée et la pente (positive et négative) du trajet. Il faudra donc trouver le chemin optimal pour que le véhicule ait le moins de distance à parcourir et qu'il ait le moins à monter et descendre possible.

4.1 Discrétisation et graphe

Soit $\Omega = [a,b] \times [c,d] \subset \mathbb{R}^2$ le domaine dans lequel le véhicule pourra évoluer. Soit la fonction $f : \Omega \mapsto \mathbb{R}$ associant à un point $X = (x,y)$ son altitude $f(X)$.

La première étape est la discrétisation de Ω. On se donne deux entiers N_x et N_y et on définit les points $X_{ij} = (x_i, y_j)$ avec $i = 1,...,N_x$ et $j = 1,...,N_y$. Si on note $\Delta x = (b-a)/(N_x-1)$ et $\Delta y = (d-c)/(N_y-1)$, on choisira N_x et N_y tels que $\Delta x \simeq \Delta y$. Les points $(X_{ij})_{i,j}$ sont les sommets du graphe associé à Ω (on notera $\mathscr{X}$ l'ensemble des sommets $(X_{ij})_{i,j}$ du graphe).

On définit ensuite l'ensemble $\mathscr{A}$ des arêtes du graphe. Pour cela, on se donne un réel $\delta > \max(\Delta x, \Delta y)$. Les arêtes du graphe sont définies par les segments $\sigma_{X_n X_m} = (X_n, X_m)_{1 \leq n,m \leq N_x N_y}$ vérifiant $0 < |X_n - X_m| \leq \delta$. Voici deux exemples de graphes obtenus suivant différentes valeurs de δ.



$\max(\Delta x, \Delta y) < \delta < \sqrt{\Delta x^2 + \Delta y^2}$ $\qquad$ $\sqrt{\Delta x^2 + \Delta y^2} \leq \delta < 2\min(\Delta x, \Delta y)$

La dernière étape pour la construction du graphe est d'affecter une valeur à chacune des arêtes. Soit l'arête $\sigma_{X_n X_m}$ reliant les points X_n et X_m. La valeur $c_{X_n X_m}$ (ou le *coût*) associée à cette arête sera une fonction qui dépendra à la fois de la distance $|X_n - X_m|$ ainsi que la valeur absolue de la pente $|f(X_m) - f(X_n)|/|X_n - X_m|$. Le coût $c_{X_n X_m}$ pour aller de X_m à X_n (ou bien de manière équivalente de X_n à X_m) par l'arête $\sigma_{X_n X_m}$ sera d'autant plus grand que la distance entre X_m et X_n sera importante et que la valeur absolue de la pente sera grande. Par exemple, on pourrait définir le coût ainsi :

$$c_{X_n X_m} = \alpha \frac{|X_n - X_m|}{\max(\Delta x, \Delta y)} + (1-\alpha)\frac{|f(X_m) - f(X_n)|}{|X_n - X_m|}, \quad 0 \leq \alpha \leq 1.$$

4.2 Détermination du plus court chemin

Soit $A \in \mathscr{X}$ le point de départ du véhicule et $B \in \mathscr{X}$ son point d'arrivée. On définit un *chemin* $\mathscr{C}$ allant de A à B comme une suite de sommets de $\mathscr{X}$ $\mathscr{C} = \{A = X_0, X_1, X_2, ..., X_{p-1}, X_p = B\}$ tels que pour tout $n = 0, ..., p-1$ on ait $\sigma_{X_n X_{n+1}} \in \mathscr{A}$, c'est-à-dire que tout segment (X_n, X_{n+1}) corresponde à une arête du graphe. On associe au chemin $\mathscr{C}$ son coût : $c_{\mathscr{C}} = \sum_{n=0}^{p} c_{X_n X_n+1}$. Le problème d'optimisation du trajet du véhicule revient donc à trouver le chemin allant de A à B dont le coût est le plus faible. Ce chemin est appelé *le plus court chemin*. Pour le déterminer, on utilisera l'algorithme de Dijkstra.

Soit $\mathscr{R}$ l'ensemble des sommets restant à visiter et $\mathscr{P}$ l'ensemble des sommets déjà parcourus. On définit aussi $d(X)$ comme le coût du plus court chemin reliant X à A et $p(X)$ le prédécesseur de X dans le plus court chemin le reliant à A. L'algorithme de Dijkstra :

— Initialisation :
  $\mathscr{R} = \mathscr{X} \setminus \{A\}$,
  $\mathscr{P} = \{A\}$,
  $d(A) = 0$, $d(X) = c_{AX}$ si $\sigma_{AX} \in \mathscr{A}$ et $d(X) = +\infty$ sinon.
— Tant que $B \notin \mathscr{P}$, Faire :

- — Trouver X le sommet réalisant le minimum de $d(.)$ sur $\mathscr{R}$.
- — Ajouter X à l'ensemble $\mathscr{P}$.
- — Enlever X à l'ensemble $\mathscr{R}$.
- — Pour tout $Y \in \mathscr{R}$ tel que $\sigma_{XY} \in \mathscr{A}$, Faire :
  - — Si $d(X)+c_{XY} < d(Y)$ Alors $d(Y) = d(X)+c_{XY}$, $p(Y) = X$; Fin Si.
- — Pour obtenir le plus court chemin reliant A et B, il suffit alors de cheminer à l'envers : on regarde B, puis on sauvegarde son prédécesseur $p(B)$, puis le prédécesseur de son prédécesseur $p(p(B))$, ... jusqu'à arriver à A.

4.3 But du projet

Le but de ce projet est donc de réaliser un programme utilisant GLUT permettant d'afficher dans une fenêtre graphique où les sommets du graphe correspondront aux pixels de la fenêtre, dont la couleur sera donnée par la valeur de la topographie en chacuns de ces points. On devra pouvoir visualiser en outre le plus court chemin entre deux points de la fenêtre obtenu par l'algorithme de Dijkstra. Enfin, les points de départ et d'arrivée et différents paramètres de la construction du graphe devront pouvoir être modifiés. Par exemple, différentes topographies pourront être utilisées (collines, vallées, labyrinthes, ...), la valeur de δ pourra changer et le calcul du coût par arête pourra être choisi de manière à privilégier la distance, la pente, ou les deux, en prenant par exemple une combinaison convexe de ces quantités dont la pondération pourra être modifiée par l'utilisateur. Attention, la gestion de la mémoire devra être particulèrement soignée, pour ne pas avoir à stocker des informations inutiles (matrices creuses ...). La fonction f sera défini via une chaine de caractères et sera évalué avec interpréteur de formule du cours.

5 Projet 11 : Problème du voyageur de commerce

Données : n villes numérotées $1,2,...,n$

Soit d_{ij} le coût (ici ce sera la distance) entre les villes i et j. On cherche le tour de coût minimum qui passe une fois et une seule par toutes les villes (cycle hamiltonien).

Pour les tests , on tirera aléatoirement suivant une loi uniforme les positions des villes dans un carré 100×100.

Si (x_i, y_j) sont les coordonnées de i et j

$d_i j = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ (distance euclidienne)

Algorithme à tester

5.1 Solution initiale

A1 Prendre les villes dans l'ordre $1,2,...,n$

A2 Partir d'une ville et choisir la plus proche puis la plus proche etc ... Si (1) et (k) sont les premières et dernières villes, refermer le chemin en ajoutant l'arc $(k,1)$.

A3 Partir de 3 villes ({1,2,3} par exemple). Supposons à une itération qu'on ait les villes $(i_1, i_2, ..., i_k)$ formant un cycle. Choisir une ville j non encore étudiée et la placer entre 2 villes $i_r, i_r + 1$ successives du circuit actuel $(1 \leq r < k)$ ou (i_k, i_1) de la manière la moins coûteuse.

A4 Construire l'arbre de poids minimum et parcourir 2 fois cet arbre puis enlever les villes parcourues 2 fois.

$1 \to 2 \to 3 \to 4 \to 3 \to 5 \to 3 \to 2 \to 6 \to 7 \to 6 \to 8 \to 6 \to 2 \to 1$

ce qui donne en fait [1 2 3 4 5 6 7 8 1]

5.2 Méthodes d'amélioration

B1 Echange de 2 arcs sur la solution actuelle (voir schma)

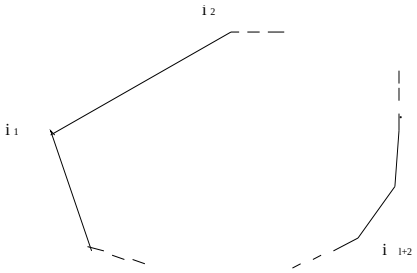
Soient 2 arcs $e = (i, i+1), f = (j, j+1)$ non consécutifs utilisés par la solution courante H (c.a.d.) les 4 indices $i, j, i+1, j+1$ sont distincts).

La nouvelle solution H_1 consiste à prendre j comme sommet suivant de i, de parcourir les sommets $(j-1, j-2, ...i+1)$ puis d'aller en $j+1$ et de compléter la tournée comme dans la solution H. Le coût $C(H_1)$ de H_1 se déduit de celui de $C(H)$ de H en ajoutant le coût des arcs $(i, j)(i+1, j+1)$ et en retirant le coût des arcs $(i, i+1)(j, j+1)$.

On dira que H_1 améliore H si $C(H_1) < C(H)$.

Si H_1 améliore H, remplacer H par H_1 et rechercher une nouvelle améloration Si aucune amélioration n'est possible à partir de H_1 pour tous les choix possibles de 2 arcs e et f de H, fournir H comme solution de l'algorithme.

B2 Enlever une ville i d'une solution actuelle et la replacer par un endroit plus intéressant entre $(j, j+1)$, c'est-à-dire les arcs $(i-1,i)$, $(i,i+1)$ et $j, j+1$ sont supprimés et les arcs $(i-1,i+1)$, (j,i) et $(i,j+1)$ sont créés. Pour faire cette transformation, il faut que deux arcs $(i,i+1)$, $(j,j+1)$ n'est aucun sommet en commun (c'est à dire que les 4 indices $i,i+1,j,j+1$ sont distincts) utilisés par la solution actuelle H.



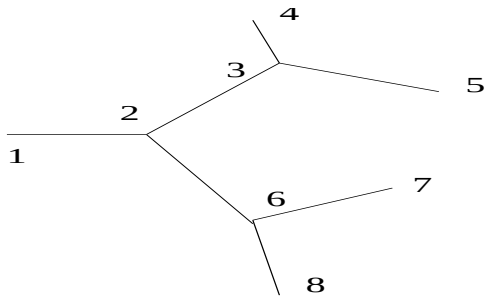


FIGURE 5 – figure A3 figure A4

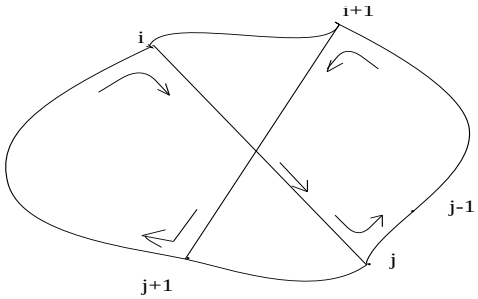


FIGURE 6 – Amélioration figure B1

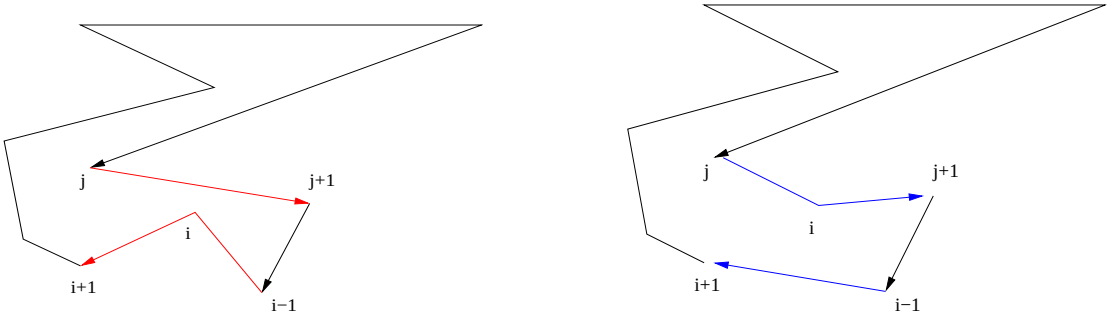


FIGURE 7 – Amélioration B2

5.3 Le projet

programmer les quatre initialisations et le deux méthodes d'amélioration, faite une analyse simple de la complexité des algorithmes.

6 Sujet : Résolution d'une EDP par la méthode des différences finis et visualisation

Le but est de résoudre numériquement l' équation suivante :
Trouver u une fonction régulière de $\Omega =]0,1[^2$ dans $\mathbb{R}$ tel que

$$-\Delta u = f, \qquad \text{dans} \quad \Omega \tag{2}$$

et telle que u soit nulle sur le bord du carré Ω et où l'opérateur Δ est défini par $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$.

Pour cela nous utiliserons une méthode aux différences finis. Nous allons calculé une approximation de la fonction u aux points $x_{n,m}$, noté $u_{n,m}$, où les points $x_{n,m}$ sont $(h_1 n, h_2 m)$ avec $h_1 = 1/N$ et $h_2 = 1/M$, pour $n = 0,...,N$, et $m = 0,...,M$.

Le schéma aux différences finis pour approcher Δ par Δ_d au point interne $x_{n,m}$ (c'est-à-dire n différent de 0 ou N et m différent de 0 ou M) , c'est à dire :

$$\forall (n,m) \in \{1,...,N-1\} \times m = \{1,...,M-1\} \quad \Delta_d u_{n,m} = \frac{u_{n-1,m} + u_{n+1,m} - 2u_{n,m}}{h_1^2} + \frac{u_{n,m-1} + u_{n,m-1} - 2u_{n,m}}{h_2^2} = f(x_{n,m}).$$

Remarque $u_{n,m}$ est connue et nulle sur les point du bord, c'est-à-dire $n \in \{0,N\}$ ou $m \in \{0,M\}$.

6.1 Projet 12/ UMFPACK

Le projet est de numéroté les inconnues du systèmes linéaire, pour construire la matrice creuse du système linéaire (où l'on ne stocke que les éléments non nuls)

Il existe plusieurs bibliothèque sur le toile (WEB) pour résoudre de grand système linéaire creux. Vous utiliserez le logiciel UMFPACK, qu'il faut trouver, compiler, tester, valider.

Afin de valider programme problème, il faut trouver des solutions du problème (2), le plus simple de de construire une solution manufacture, c'est à dire choisir un fonction analytique ϕ nulle sur le bord de Ω, et de calculer $f = -\Delta\phi$.

Pour, finir, vous utiliserez la bibliothèque GLUT, pour visualiser la représentation 3D de votre solution.

Il faut tester votre programme, pour une famille de N, M allant 5 à 100, faire des courbes de temps calcul, etc ...

6.2 Projet 13/ SuperLU

Le projet est de numéroté les inconnues du systèmes linéaire, pour construire la matrice creuse du système linéaire (où l'on ne stocke que les éléments non nuls)

Il existe plusieurs bibliothèque sur le toile (WEB) pour résoudre de grand système linéaire creux. Vous utiliserez le logiciel SuperLU, qu'il faut trouver, compiler, tester, valider.

Afin de valider programme problème, il faut trouver des solutions du problème (2), le plus simple de de construire une solution manufacture, c'est à dire choisir un fonction analytique ϕ nulle sur le bord de Ω, et de calculer $f = -\Delta\phi$.

Pour, finir, vous utiliserez la bibliothèque GLUT, pour visualiser la représentation 3D de votre solution.

Il faut tester votre programme, pour une famille de N, M allant 5 à 100, faire des courbes de temps calcul, etc ...

7 Sujet 15 : Découpe d'un cube en tétraèdre et visualisation

Le but est très simple, trouver les 74 découpes du cube en tétraèdres, et visualiser ces découpes avec OpenGL/GLUT.

Les découpes sont des partitions du cube telle que :

1. les sommets des tétraèdres sont des sommets du cube.
2. le maillage est conforme, c'est à dire que l'intersection de deux tétraèdres différents (supposés fermés) est soit
   (a) une face commune aux 2 tétraèdres
   (b) une arête commune aux 2 tétraèdres
   (c) un sommet commun aux 2 tétraèdres
   (d) l'ensemble vide.

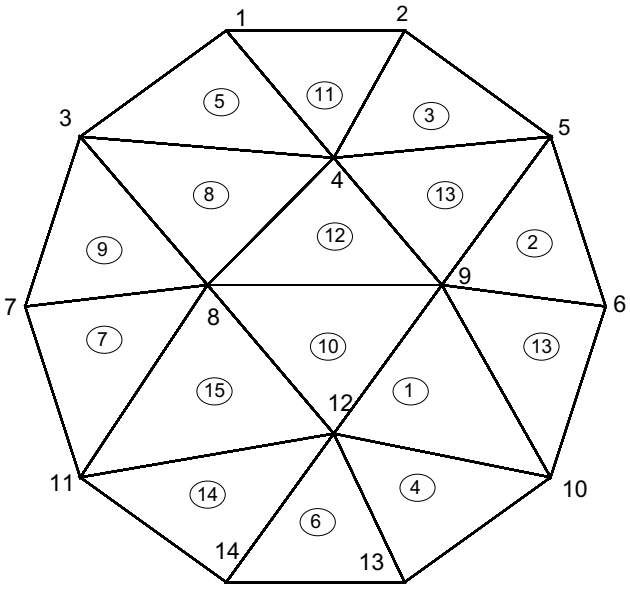
Pour ce faire, il est conseiller de construire l'ensemble de tétraèdre possible, et de décrire les relations de compatibilité entre les faces.

Question bonus, faire la même chose en dimension 4 (dur).

8 Annexe : Structure des fichiers maillage

We can read from Fig. 8 and "mesh_sample.msh" as in Table 1 where n_v is the number of vertices, n_t number of triangles and n_s the number of edges on boundary. For each vertex $q^i, i = 1, \cdots, n_v$, we denote by (q^i_x, q^i_y) the x-coordinate and y-coordinate.

Each triangle $T_k, k = 1, \cdots, 10$ have three vertices $q^{k_1}, q^{k_2}, q^{k_3}$ that are oriented in counterclockwise. The boundary consists of 10 lines $L_i, i = 1, \cdots, 10$ whose tips are q^{i_1}, q^{i_2}.



In the left figure, we have the following.

$n_v = 14, n_t = 16, n_s = 10$

$q^1 = (-0.309016994375, 0.951056516295)$
$\vdots \quad \vdots \quad \vdots$
$q^{14} = (-0.309016994375, -0.951056516295)$

The vertices of T_1 are q^9, q^{12}, q^{10}.
$\vdots \quad \vdots \quad \vdots$
The vertices of T_{16} are q^9, q^{10}, q^6.

The edge of 1st side L_1 are q^6, q^5.
$\vdots \quad \vdots \quad \vdots$
The edge of 10th side L_{10} are q^{10}, q^6.

FIGURE 8 – mesh by `buildmesh(C(10))`

Contents of file	Explanation
14 16 10	n_v n_t n_e
-0.309016994375 0.951056516295 1	q^1_x q^1_y boundary label=1
0.309016994375 0.951056516295 1	q^2_x q^2_y boundary label=1
$\ldots\ldots\vdots$	
-0.309016994375 -0.951056516295 1	q^{14}_x q^{14}_y boundary label=1
9 12 10 0	1_1 1_2 1_3 region label=0
5 9 6 0	2_1 2_2 2_3 region label=0
$\cdots$	
9 10 6 0	16_1 16_2 16_3 region label=0
6 5 1	1_1 1_2 boundary label=1
5 2 1	2_1 2_2 boundary label=1
$\cdots$	
10 6 1	10_1 10_2 boundary label=1

TABLE 1 – The structure of "mesh_sample.msh"