Génération de maillage pour le calcul scientifique

Frédéric Alauzet, frederic.alauzet@inria.fr

Mots-clés : génération de maillages, adaptation de maillage, estimateur d'erreur

Introduction

Ce document regroupe quelques applications numériques illustrant les notions vues dans le cours : triangulation, maillage, interpolation, adaptation, ... Les exemples et exercices sont de difficultés variées et doivent permettre aux élèves motivés d'aborder les problématiques liées à la génération et adaptation de maillage notamment en termes de complexité algorithmique. On se limitera à la dimension deux.

Pour réaliser ces projets, il est nécessaire d'avoir des connaissances minimales en langage C. Des algorithmes en pseudo-code sont donnés afin d'appréhender rapidement la structure des algorithmes.

Il n'est pas attendu de finir tous les exercices. Les exercices seront faits pendant les séances de cours. Un compte-rendu pour chacune des 3 parties devra être envoyé par email à : frederic.alauzet@inria.fr
L'évaluation sera faite sur les comptes-rendus.

Installation

L'archive du projet peut être récupérée sur ce lien : projet-AMS314.zip

L'archive est composée des répertoires suivants :

- `src` : Ce dossier contient des squelettes de fonction C qui vous pourrez librement adapter et modifier.
- `bin` : Ce dossier contient des exécutables externes qui permettent de générer des maillages adaptés. Il est nécessaire d'utiliser l'exécutable qui correspond à votre architecture.
- `data` : Ce dossier contient les fichiers maillages ou les fichiers images qui sont utilisés pour valider les différents algorithmes mis en place. Lire l'Annexe pour plus détails.
- `matlab` : Ce dossier contient les algorithmes en pseudo-code Matlab.

Pour valider l'installation, on pourra réaliser une adaptation de maillage uniforme en lançant la commande suivante dans un terminal :

```
./bin/mac/feflo.a_2d -in data/carre_h.mesh  -hmax 0.2 -out tmp.mesh
```

Remplacer `mac` par `linux` ou `win` suivant votre système d'exploitation.

Le logiciel de visualisation peut être téléchargé sur la page suivante : `ViZiR 4`.
Ce logiciel permet de visualiser les maillages au format `.mesh` et les champs de solutions au format `.sol` (voir Annexe). Pour valider l'installation, on pourra visualiser le maillage créé précédemment (après avoir créé un lien vers l'exécutable dans le `.bash_profile`) :

```
vizir4 -in tmp.mesh
```

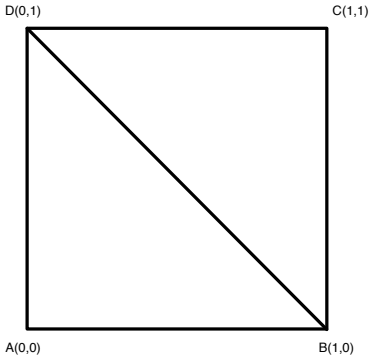
PARTIE I :

1 Lecture de maillage et manipulation (informatique) des maillages

Informatiquement, un maillage, dans son plus simple appareil, est stocké sous forme de 2 matrices. Une première matrice des coordonnées qui contient les x, y en 2D de chaque sommet du maillage et une matrice de connectivité entière afin de définir les éléments : triangles en 2D. Tous les fichiers de maillages fournis sont dans le format `.mesh` qui est le standard INRIA (cf. Annexe) pour la gestion de maillages non structurés. Par exemple, pour le maillage trivial d'un carré unité composé de deux triangles, la représentation matricielle est :

```
Crd = [ 0  1  1  0 ]
      [ 0  0  1  1 ];

Tri  = [ 1   2 ]
       [ 2   3 ]
       [ 4   4 ];
```



Pour tester les algorithmes développés dans ce chapitre, on sera amené à considérer des exemples plus complexes en deux dimensions. On fournit un exemple de fonction *main* dans `main_mesh.c`. Pour compiler cet exemple, et lire, par exemple, le maillage `carre_4h.mesh`, on fait :

```
cd src/
make -f makemesh.make
cd ..
./src/mesh data/carre_4h.mesh
```

Utiliser l'executable `src/mesh` pour lire d'autres maillage fournit dans `data`. Il est conseillé de parcourir les fichiers `mesh.h` et `mesh.c` pour comprendre les structures de données.

Attention : On a vu en cours qu'il est conseillé que les structures globales commence à l'indice 1, même en C. Dans les exemples de matrices `Crd` et `Tri` ci-dessus, on a omis volontairement l'indice 0. Par conséquent, lors de l'allocation mémoire en C, on alloue toujours les tableaux globaux avec le nombre d'entités plus un : `NbrEnt+1`.

Exercice : Implémenter les fonctions qualité standards pour les triangles :

$$Q_1(K) = \alpha_1 \frac{\sum_i \ell_i^2}{|K|} \qquad \text{et} \qquad Q_2(K) = \alpha_2 \frac{h_{max}}{\rho(K)}\,,$$

où ℓ_i représente la longueur des arêtes de l'élément K, h_{max} la longueur de la plus grande arête et $\rho(K)$ le rayon du cercle inscrit à l'élément K. On définira les constantes α_1 et α_2 telles que les fonctions de qualité soient égales à 1 pour le triangle équilatéral. Ces fonctions de qualité varient en 1 et $+\infty$. La déclaration de ces fonctions se feront dans `mesh.h` et l'implémentation dans `mesh.c`. On donnera un histogramme de qualité pour l'ensemble des maillages du répertoire `data/`.

Pour visualiser la qualité d'un maillage, on écrira la qualité dans le fichier solution `quality.solb` à l'aide de la fonction `msh_write2dfield_Triangles` et on utilisera `vizir` en faisant :

```
vizir -in output.meshb -sol quality.solb
```

Attention : Si vous avez ré-ordonné le maillage initial dans votre `main` par la fonction `msh_reorder` alors la solution écrite correspond au maillage ré-ordonné et n'est donc plus compatible avec votre maillage initial. Le maillage ré-ordonné est écrit dans votre `main` dans le fichier `output.meshb` par la fonction `msh_write`. C'est celui là qu'il faut utiliser pour la visualisation.

2 Algorithme quadratique et table de hachage

L'utilisation des deux seuls tableaux des coordonnées `Crd` et des éléments `Tri` n'est généralement pas suffisante pour écrire des algorithmes efficaces de maillages. Il très souvent utile de construire des tableaux annexes comme la structure des voisins (par arête en 2D ou en surface et par face en 3D) ou encore, le tableau des arêtes (de surface, de volume). Dans le cas du tableau des voisins, il est nécessaire de trouver rapidement pour chaque arête du maillage le triangle ou les deux triangles la partageant. Pour comprendre l'intérêt d'obtenir un algorithme efficace, il est instructif de construire ce tableau par une méthode naïve (comme celui présenté dans le cours). Un tel algorithme a une complexité quadratique, et on pourra vérifier qu'un tel algorithme n'est plus du tout envisageable pour des maillages ayant plus de 100 voire 1000 triangles si on est patient. Complétez la fonction `msh_neighborsQ2` pour les triangles.

Une table de hachage est une structure de données classique en algorithmique qui permet d'accéder rapidement à des objets par une clé qui les caractérisent. La clé n'est généralement pas unique. Il est donc nécessaire de pouvoir stocker plusieurs objets de même clé, mais qui sont différents. De tels objets sont dits en collision. Lorsqu'on recherche un objet donné, on calcule sa clé, puis on ne parcourt que les objets qui ont cette clé. Ce parcours est donc quadratique, mais seulement sur un petit nombre d'objets. C'est donc la qualité de la clé de hachage qui garantira l'efficacité du processus.

Une table de hachage est donc constituée d'un tableau de tête et d'un tableau de lien qui stocke les objets en collision (objets ayant la même clé). La tête donne l'index dans le tableau de lien du premier objet ayant cette clé. Les paramètres d'une table de hachage sont donc la taille maximum de la tête et le nombre maximal d'objets à stocker.

La structure de la table de hachage est donnée dans le fichier `mesh.h` pour des arêtes en 2D. On note `SizHead` la taille du tableau de tête `Head` et `NbrMaxObj` la taille du tableau de lien `LstObj` (*i.e.,* le nombre maximal d'objets que l'on peut mettre dans la table de hachage). Les valeurs de `SizHead` et `NbrMaxObj` dépendent du problème considéré. Dans les cas où l'on s'intéresse aux voisins par arête des triangles, on peut aisément déduire des valeurs de `SizHead` et `NbrMaxObj`. Pour la table de hachage, on utilisera cette convention :

```
NbrObj           = nombre d'objets stocké dans le tableau de lien
Head[iPos]       = lien sur la première face ayant iPos comme clé
LstObj[iObj][0]  = premier point de l'arête pour l'objet iObj
LstObj[iObj][1]  = deuxième point de l'arête pour l'objet iObj
LstObj[iObj][2]  = premier triangle ayant cette arête pour l'objet iObj
LstObj[iObj][3]  = deuxième triangle ayant cette arête pour l'objet iObj
LstObj[iObj][4]  = arête suivante dans le chaînage pour l'objet iObj
```

Les objets sont donc des arêtes. Ils sont donc caractérisés par deux points. On stocke aussi les triangles associés pour retrouver facilement les voisins.

Préciser les valeurs de `SizHead` et `NbrMaxObj` et allouer les tableaux `Head` et `LstObj`.

Il reste donc à implémenter les fonctions fondamentales des tables de hachage :

- l'initialisation de la table de hachage `hash_init`,
- la recherche d'un objet dans la table de hachage `hash_find`,
- l'ajout d'un objet dans la table de hachage `hash_add`,
- la suppression d'un objet dans la table de hachage `hash_suppr`.

Dans un premier temps, compléter les trois fonctions du fichier `mesh.c` dont les prototypes sont données dans le fichier `mesh.h` dans ce cas particulier :

```
HashTable * hash_init(int SizHead, int NbrMaxObj);
int hash_find(HashTable *hsh, int ip1, int ip2);
int hash_add(HashTable, *hsh, int ip1, int ip2, int iTri)
```

Exercice : Comparer les temps CPU pour construire les voisins entre l'approche naïve et celle basée sur la table de hachage. On pourra considérer les différents maillages `carre_4h.mesh`, ..., `carre_05h.mesh`.

Exercice : Utiliser la structure de table de hachage pour rechercher dans un maillage, le nombre d'arêtes frontières. Donner pour les maillages `naca0012.mesh`, `ls89.mesh` et `hlcrm.mesh` le nombre d'arêtes frontières.

Exercice : Adapter la table de hachage pour calculer rapidement le nombre d'arêtes d'un maillage. Comparer les nombres de collisions moyen, et maximum pour d'autres clés. Quel est le nombre d'arêtes des maillages `naca0012.mesh`, `ls89.mesh` et `hlcrm.mesh` ?

Dans la section suivante, la structure de voisin sera utilisée pour retrouver les composantes connexes d'un maillage. Puis, le concept de table de hachage sera utilisé pour la triangulation de Delaunay dans un problème de compression d'image dans la Partie II du projet.

3 Composantes connexes (facultatif)

On propose dans cet exercice de retrouver les composantes connexes d'un maillage. Ceci est une des étapes dans les méthodes de génération de maillage basé sur la méthode de Delaunay.

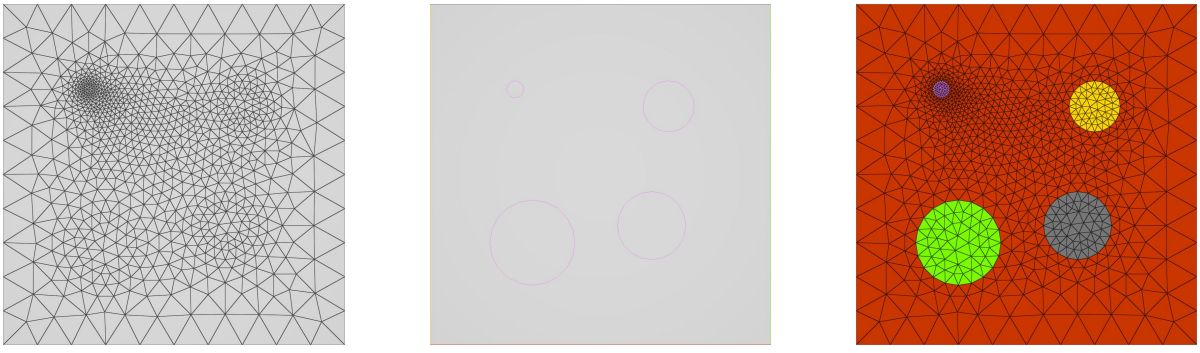


FIGURE 1 – *Maillage d'un carré (à gauche). Réseau d'arêtes internes (au milieu). Ces arêtes composent la frontière des composantes connexes. L'enjeu est de colorier tous les éléments avec une couleur (référence) par composante connexe (à droite).*

Une composante connexe est un ensemble d'éléments tel que pour deux éléments quelconques de cet ensemble, il existe un chemin à travers les arêtes en 2D (ou les faces en 3D) joignant ces deux éléments et ne traversant pas une arête frontière. Pour cet exercice, on considère un domaine carré où il existe un réseau de frontières internes (arêtes voyant deux triangles) comme illustré sur la Figure 1. On cherche à colorier tous les sous-domaines connexes. La première étape consiste à adapter le calcul des voisins en intégrant la liste des arêtes frontières dans la recherche. Il est également nécessaire d'avoir une règle de priorité pour que les arêtes frontières soient recherchées d'abord. En effet, pour les arêtes des triangles qui constituent l'interface des sous-domaines, il existe souvent un autre triangle voyant cette arête ainsi que l'arête frontière. Pour garantir que l'on ne traversera pas cette frontière, les deux triangles doivent avoir un numéro d'arête (ou zéro) dans le tableau des voisins. L'algorithme se décompose comme suit :

0. Initialisation : tous les triangles sont coloriés à zéro
1. Recherche d'un germe, *i.e.*, un élément de couleur nulle, et choisir un numéro de sous-domaine non nul.
2. Colorier par voisinage tous les éléments sans traverser de frontières.

► Retour en 1 tant qu'il reste des triangles à colorier.

Attention : Avant de démarrer, vous devez activer la lecture des arêtes frontière dans la fonction `msh_read` en mettant le paramètre `readEfr` à 1.

Pour l'étape 2, on utilise une pile qui va stocker tous les éléments déjà coloriés dont il sera nécessaire de visiter les voisins. Lorsque la pile sera vide, toute la composante connexe aura été détectée. Pour implementer une structure de pile, il faut deux fonction. Une qui ajoute un élément à la pile et une autre qui enlève un élément à la pile. On pourra s'inspirer de la structure de la table de hachage. Ceci peut être aussi fait à l'aide d'un tableau et de deux pointeurs : un pour la position du prochain élément à mettre dans la pile et un pour la position du prochain élément à sortir de la pile.

Exercice : Implémenter ensuite la structure générale de l'algorithme en C :

```
% 1. Lecture du maillage

% 2. Calculer la structure de voisins

% 3. Algorithme de coloriage (a completer) :  ndomn est l'indice du sous-domaine

ndomn = 0
for (iTri=1; iTri<=NbrTri; ++iTri)
  if ( color(iTri) == 0 )
       ndomn = ndomn + 1
       color[iTri] = ndomn
       % ajoute iTri à la pile : cela initialise la pile
       % boucle principale sur la taille de la pile
         % dépiler le prochain élément de la pile
         % Verifier les triangles voisins
         % empiler les triangles vues par une arête non frontière et mettre leur couleur à ndomn
  end
end
```

Retrouver les composantes connexes du maillage `squarecircle.mesh`.

Combien de triangles appartiennent à chaque composante connexe ?

Annexe : le format `.mesh`

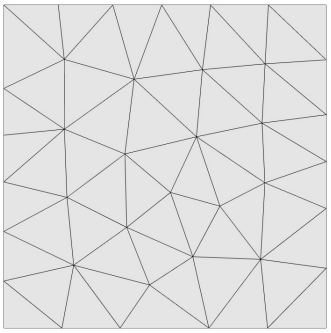
Plusieurs maillages sont donnés dans le répertoire `data` au format `.mesh`. On pourra utiliser le logiciel `vizir4` pour visualiser ces maillages :

- https://pyamg.saclay.inria.fr/vizir4.html

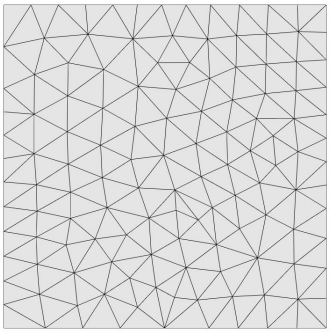
Une documentation est disponible ici :

- https://pyamg.saclay.inria.fr/download/vizir/vizir4_user_guide.pdf

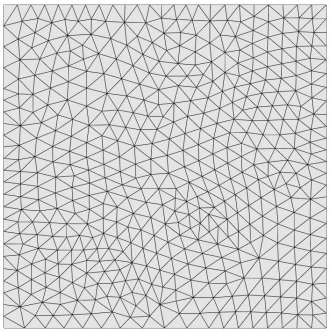
Pour valider l'efficacité de la table de hachage, on utilisera les maillages du carré unité de taille croissante `carre_4h.mesh`, ..., `carre_05h.mesh` :



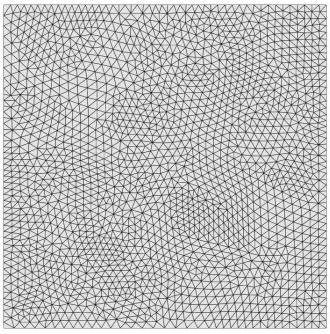
`carre4h.mesh`



`carre2h.mesh`

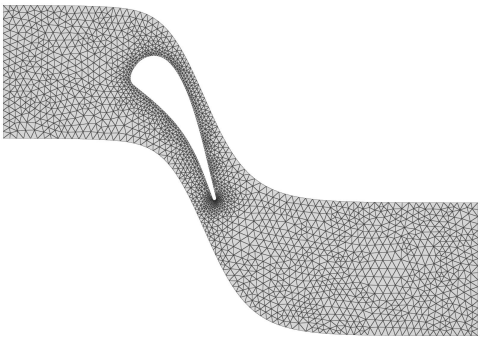


`carreh.mesh`

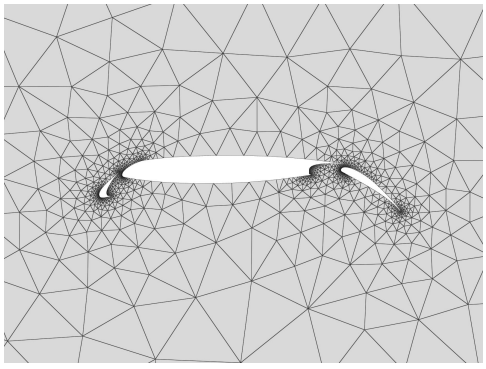


`carre05h.mesh`

Dans le cas des voisins pour retrouver la frontière d'un domaine, on utilisera les maillages `ls89.mesh` et `hlcrm.mesh`.

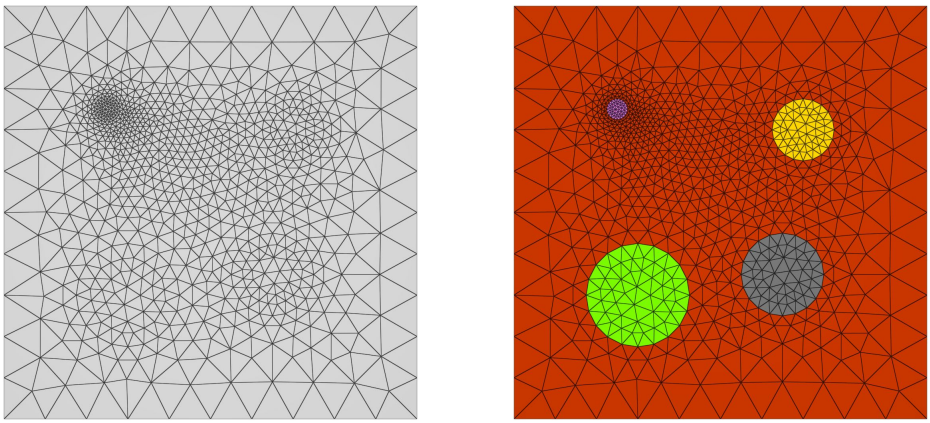


`ls89.mesh`



`hlcrm.mesh`

Pour l'exercice concernant la re-construction de sous-domaine, on utilisera le maillage `squarecircle.mesh`.



`squarecircle.mesh`

Le format `.mesh` permet de décrire des maillages non-structurés, on l'utilise dans le projet pour les maillages 2D composés de triangles. Pour définir un maillage 2D, il suffit de définir l'en-tête composée des deux mots-clés `MeshVersionFormatted 2` et `Dimension 2` qu'on laisse inchangé. Ensuite, on définit la liste de sommets `Vertices` et la liste des triangles `Triangles`. La référence `ref` (dernière valeur entière) pour les triangles et les sommets n'est pas utilisé ici, mettre 0. Le fichier doit se terminer par le mot-clé `End`. Exemple :

```
MeshVersionFormatted 2
Dimension 2

# Set of mesh vertices (x,y,ref)
Vertices
581
0.1 1. 0
0.333 12.125 0
.........

# Set of mesh triangles (v1,v2,v3,ref)
Triangles
1162
1 28 521 0
23 45 77 0
.........

End
```