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 II

1 Algorithme de Delaunay et application à la compression d'image

On s'intéresse dans cet exercice à la compression d'une image en niveaux de gris en utilisant les propriétés d'interpolation et d'unicité de la triangulation de Delaunay en 2D.

Rappel sur la triangulation de Delaunay. Étant donné un ensemble de points du plan $E = (x_i, y_i)_{i\in[\![1,N]\!]}$, la triangulation de Delaunay est une triangulation conforme d'élément simpliciaux (triangles en 2D) dont les sommets sont les points de E et vérifiant la propriété de la sphère vide : *« Étant donné un triangle K, il n'existe aucun point de la triangulation (autre que les points de K) contenu dans la cercle circonscrit à K »*. Un exemple de configuration violant ce critère est représenté sur la Figure 1. Le principal résultat (Lemme de Delaunay) est que si le critère de la sphère vide est vérifié localement, *i.e.*, pour chaque couple de triangles adjacents, alors il est vérifié globalement. Par exemple, cette triangulation permet de trouver le plus proche voisin d'un nuage de points (cf. fonction dsearchn de Matlab) ou encore l'enveloppe convexe d'un nuage de point.

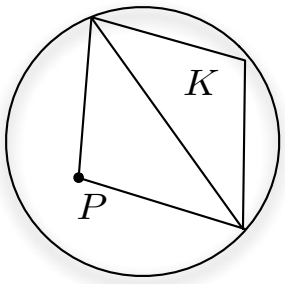


FIGURE 1 – *Maillage violant localement le critère de Delaunay, le point P est contenu dans le cercle circonscrit au triangle K.*

Si on ajoute à chaque point (x_i, y_i) de l'ensemble discret E, une solution u_i (le niveau de gris), la triangulation de Delaunay permet d'obtenir une représentation continue u de l'image tel que $u(x_i, y_i) = u_i$ en chaque point de E. u est simplement l'interpolé linéaire sur chaque triangle des valeurs discrètes de u_i. On peut donc représenter une image I sous la forme d'une fonction u de $[1, m] \times [1, n]$ à valeur dans $\mathbb{R}$. Un exemple de cette représentation est donné par la Figure 2. On utilise cette représentation dans la suite du projet.

Application à la compression/décompression d'images. L'intérêt de la triangulation de Delaunay pour la compression d'image est dû à l'unicité de cette triangulation. Si le maillage représentant la Joconde, voir Figure 3, est de Delaunay, il est suffisant de stocker seulement la liste des pixels significatifs avec leur niveau de gris. En particulier, la connectivité entre les sommets (liste de triangles) n'est pas nécessaire puisqu'elle peut être régénérée en créant la triangulation de Delaunay de la liste des pixels significatifs. Si l'image initiale comporte $m \times n$ pixels, son stockage est de l'ordre de $m \times n$, si le maillage de Delaunay comporte N_v sommets, la mémoire nécessaire pour stocker l'image compressée

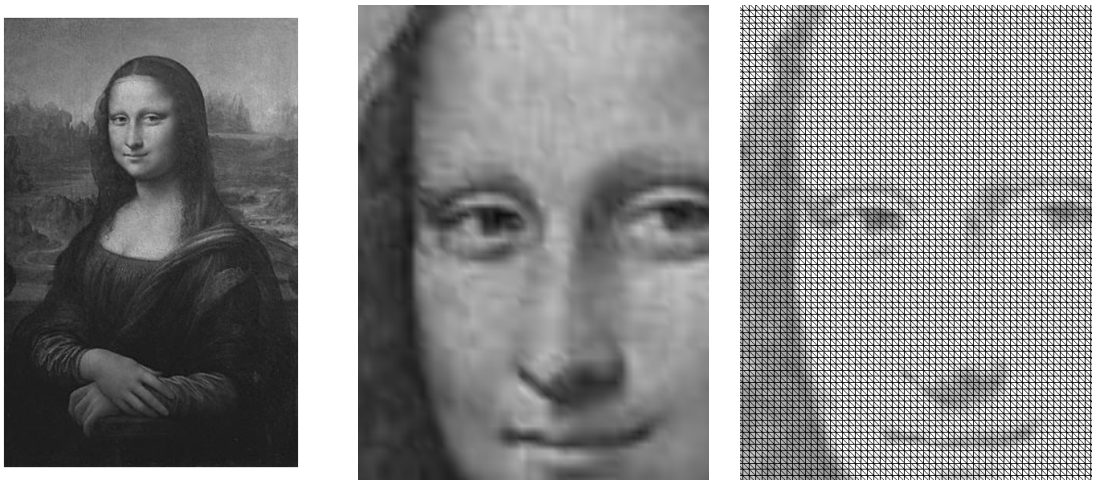


FIGURE 2 – *Représentation de l'image de la Joconde (à gauche) par un maillage non-structuré (à droite) où chaque sommet du maillage a pour solution le niveau de gris de l'image.*

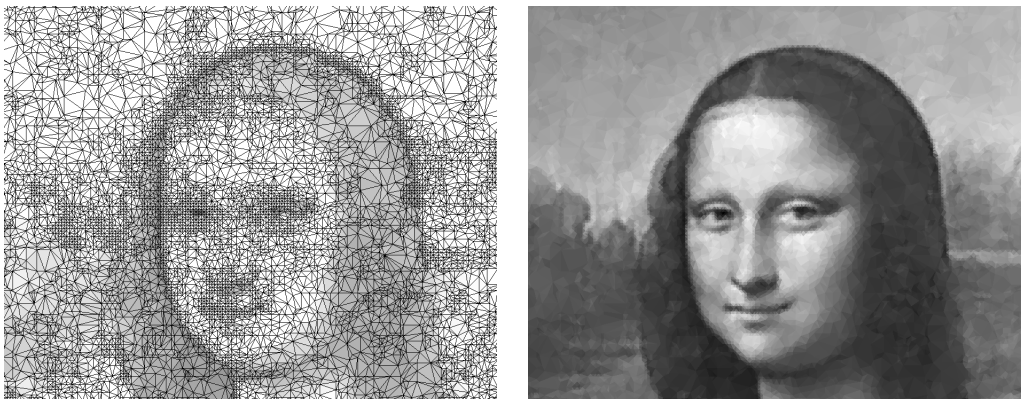


FIGURE 3 – *Représentation de l'image de la Joconde sur un maillage de Delaunay composée de* $27\,000$ *sommets au lieu des* $127\,000$ *sommets du maillage initial de la Figure 2.*

sous ce format est de l'ordre de $2N_v$ (liste des numéros des pixels significatifs et niveau de gris associé). Dans le cas de l'exemple de la Figure 3, le facteur de compression est donc de

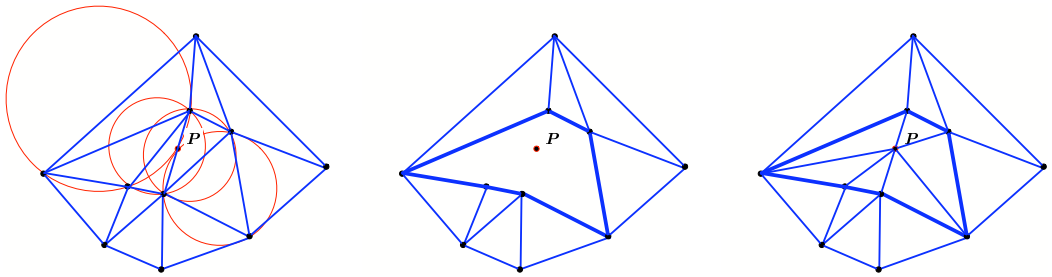
$$\alpha = \frac{127\,000}{2 \times 27\,000} = 2.3.$$

On propose le découpage du travail suivant.

Noyau de Delaunay. Cette partie concerne l'écriture d'un noyau de Delaunay en 2D. On implémentera la méthode dite de Bowyer-Watson. C'est une méthode incrémentale qui permet d'insérer un point P à partir de la triangulation courante $\mathcal{T}^n$. Les étapes sont les suivantes :

1. Localiser le point P dans $\mathcal{T}^n$,
2. Calculer la cavité $\mathcal{C}_P$ associée à P et retirer les triangles de $\mathcal{C}_p$ de $\mathcal{T}^n$
3. Créer les triangles $\mathcal{B}_P$ en étoilant autour du point P, la nouvelle triangulation est

$$\mathcal{T}^{n+1} = \mathcal{T}^n - \mathcal{C}_P + \mathcal{B}_P.$$



Etape 2a Etape 2b Etape 3

FIGURE 4 – *Insertion d'un point P dans la triangulation $\mathcal{T}^n$: Calcul de la cavité $\mathcal{C}_P$, représentation de $\mathcal{T}^n - \mathcal{C}_p$, reconnection $\mathcal{T}^{n+1} = \mathcal{T}^n - \mathcal{C}_P + \mathcal{B}_P$.*

La Figure 4 illustre les étapes 2 et 3. Il est important de proposer une structure de donnée adaptée à cet algorithme afin d'obtenir un noyau efficace.

Pour commencer, ce noyau pourra être testé indépendamment sur l'insertion d'une liste de point aléatoire du plan. Pour cela, dans un premier temps vous créez votre domaine 2D qui sera votre maillage vide de départ c'est-à-dire vous prenez comme domaine le carré unité $[0,1]\times[0,1]$ définit par ses 4 sommets et 2 triangles. Ensuite à l'aide de la fonction C `rand()` vous générez les coordonnées aléatoires des points que vous allez insérer. Il important d'initialiser toujours de la même manière la fonction `rand()` pour avoir un résultat reproductible, par exemple avec `srand(7)`.

Une courbe de complexité donnant le temps CPU en fonction du nombre de points insérés permettra de vérifier la bonne implémentation de l'algorithme. Notez que l'étape 1 peut-être traitée indépendamment dans un premier temps, elle intervient également dans la décompression des images.

Localisation et compression. Lors de la phase de compression d'image, les niveaux de gris du maillage non structuré (image compressée) sont interpolés à partir des niveaux de gris des $n \times m$ pixels (image originale représentée par le maillage structuré initial). Cela revient donc à localiser un point P dans un maillage non structuré (algorithme identique à l'étape 1 du noyau de Delaunay). On utilisera une technique de parcours local où à partir d'un triangle initial, on se déplace dans le maillage en traversant les arêtes des triangles nous rapprochant de P. Un parcours typique est représenté sur la Figure 5. Le (les) choix de l'arête à traverser est donné par le signe des coordonnées barycentriques du point P vis-à-vis du triangle courant K. On rappelle que la coordonnée barycentrique d'un point P vis-à-vis d'une arête orientée P_1P_2 est l'aire signée du triangle PP_1P_2. Une coordonnée barycentrique négative est un choix possible d'arête à traverser. La Figure 6 illustre les différents choix. Lorsque les trois coordonnées barycentriques sont positives, le point est localisé et son niveau de gris est interpolé en utilisant les niveaux de gris des trois sommets de K. Lorsque plusieurs arêtes peuvent être traversées, un choix aléatoire est fait afin d'éviter une boucle infinie où les mêmes triangles sont visités indéfiniment, cf. Figure 7.

Traitement d'images. Cette partie se concentre sur le choix des pixels les plus significatifs. Cette partie est ouverte. On conseille d'envisager une méthode naïve (1 pixel sur 5 par exemple) avant de développer des méthodes plus fines : détection de contours, contraste, contrôle de l'erreur d'interpolation entre deux images, ... Afin de valider la qualité de l'image compressée, on pourra calculer le rapport signal sur bruit (Peak signal to noise ratio) donné par :

$$PSNR = 10\,\log_{10}\left(\frac{d^2}{q_c}\right),$$

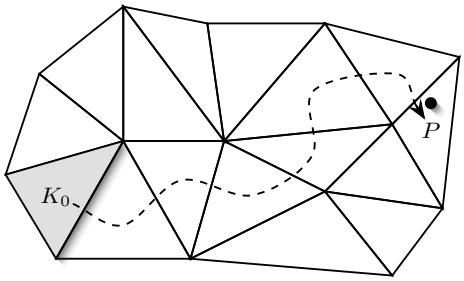


FIGURE 5 – *À partir du triangle K_0, on se déplace dans le maillage en traversant les arêtes qui nous rapprochent de P. Notez que plusieurs chemins sont possibles. Il faut donc faire un choix aléatoire lorsque plusieurs chemins sont possibles.*

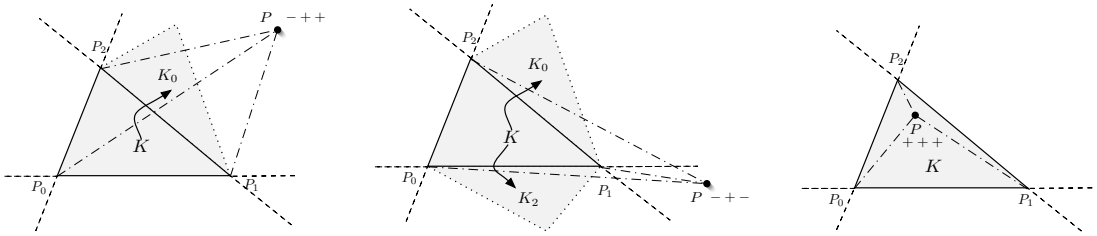


FIGURE 6 – *Choix possible d'arête en fonction du signe des coordonnées barycentriques pour différentes configurations.*

où q_c représente l'erreur quadratique moyenne :

$$q_c = \frac{1}{m\,n} \sum_{i=1}^{m} \sum_{j=1}^{n} |I(i,j) - I_c(i,j)|^2,$$

avec I le niveau de gris et d l'amplitude max de l'image. On a $d = 255$ pour une image en niveaux de gris. Les valeurs caractéristiques de $PSNR$ sont de l'ordre de 30 à 50 décibels.

Pour la lecture d'une image, on utilisera la fonction suivante qui permet de transformer une image couleur, en un maillage avec une solution scalaire qui représente les niveaux de gris. Deux images (basse et haute résolution) de la Joconde sont fournies sous forme de maillages/solutions dans le repertoire `data/`. Si vous souhaitez utiliser d'autres images, utiliser le script matlab ci-dessous pour convertir une image en maillage avec un fichier solution en niveau de gris.

```
function [coor,tri,edg,crn,sol] = im2mesh(file)


% load image
im     = imread(file);
info   = imfinfo(file);
Width  = info.Width;
Height = info.Height;


% convert image to a grey-level image from rgb
% grey = 0.3*r+0.59*g+0.11*b
```

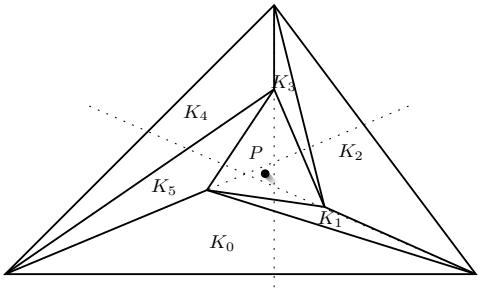


FIGURE 7 – *Configuration où le point P n'est jamais atteint si le même choix des arêtes à traverser est fait.*

```
im = 0.3*im(:,:,1) + 0.59*im(:,:,2) + 0.11*im(:,:,3);
%imshow(im);

nrow = Height;
ncol = Width;

tri  = [];
edg  = [];
crn  = [];

coor  = zeros(2,nrow*ncol);
sol   = zeros(1,nrow*ncol);
tri1  = zeros(3,(nrow-1)*(ncol-1));
tri2  = zeros(3,(nrow-1)*(ncol-1));
edg   = zeros(3,2*(ncol-1)+2*(nrow-1));
crn   = [ 1, ncol, 1 + ncol*(nrow-1), ncol*(nrow-1) + ncol ];

idx = 1;
for i=1:nrow

  ind = idx:(idx+ncol-1);
  coor(1,ind ) = 1:ncol;
  coor(2,ind ) = nrow-i+1;
  sol(ind)     = im(i,1:ncol);
  idx = idx + ncol ;

end

idx = 1;
indj = 0:(ncol-2);

for i=0:(nrow-2)

   ind  = idx:(idx+length(indj)-1);
   in   = indj + i*ncol+1;

   tri1(1,ind) = in;
   tri1(3,ind) = in+1;
   tri1(2,ind) = in+ncol+1;

   tri2(1,ind) = in;
   tri2(3,ind) = in+ncol+1;
   tri2(2,ind) = in+ncol;
   idx = idx+length(indj) ;

end

tri  = [tri1,tri2];
```

```
i1 = crn(1);
i2 = crn(2);
i3 = crn(3);
i4 = crn(4);
edg(1,:)  = [i1:(i2-1), i3:(i4-1), i1:ncol:(i3-ncol), i2:ncol:(i4-ncol) ] ;
edg(2,:)  = [(i1+1):i2, (i3+1):i4, (i1+ncol):ncol:i3, (i2+ncol):ncol:i4 ] ;
```

Exercice. Avant de se concentrer sur le cas des images, on peut réaliser un algorithme d'insertion de Delaunay plus générique. On donne le canevas suivant en pseudo-code Matlab :

```
% But:  Insertion de points par un algorithme de Delaunay

% Lire un maillage et utiliser seulement la liste des points
[dim,coor] = readmesh('carre_grossier.mesh');

% on souhaite creer une triangulation des points de coor

% 1. Creation du maillage de boite englobante, compose de 2 triangles
xmin = min(coor(1,:));
xmax = max(coor(1,:));
ymin = min(coor(2,:));
ymax = max(coor(2,:));

dx  = abs(xmax-xmin);
dy  = abs(ymax-ymin);

xmin = xmin - 0.5*dx;
xmax = xmax + 0.5*dx;
ymin = ymin - 0.5*dy;
ymax = ymax + 0.5*dy;

coor =  [ [xmin,xmax,xmax,xmin;ymin,ymin,ymax,ymax]  ,coor];
tri  =  [ [1,2,4]',[2,3,4]'];
voi  =  [ [2,0,0]',[0,1,0]'];


% 2. Pour chaque point :  insertion du point par Delaunay
for i=5:size(coor,2)
  disp(['Insertion du point ' num2str(i) ]);

  % Insertion du point par Delaunay
  % 2.1 Localiser le point
  % 2.2 Creer la cavite et calculer la frontiere de la cavite
  % 2.3 Creer boule
  % 2.4 Mettre a jour le tableau de triangles (et des voisins)
  [coord,tri] =  delaunay(coord,tri,voi,ip);

end
```

On re-utilisera la structure de table de hachage afin de calculer le bord de la cavité.

Exercice. Existe-t-il une partie quadratique dans cet algorithme ? Si oui, la modifier et comparer les temps CPUs avec l'approche non optimisée.

Exercice. Utiliser cet algorithme pour compresser et décompresser les images.

Annexe : le format `.mesh`

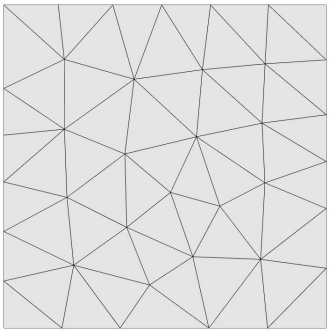
Plusieurs maillages sont donnés dans le répertoire `data` au format `.mesh`. On pourra utiliser le logiciel `vizir4` pour visualiser ces maillages :

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

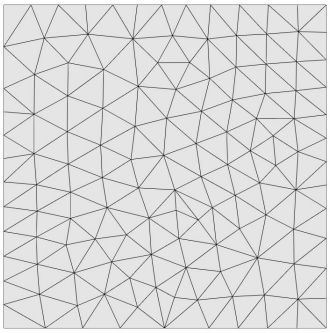
Une documentation est disponible ici :

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

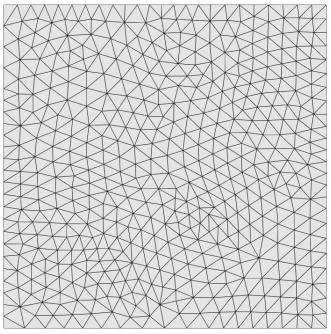
Pour valider l'efficacité de la table de hachage, on utilisera les maillages du carré unité de taille croissante `carre_4h.mesh`, ..., `carre_05h.mesh` :



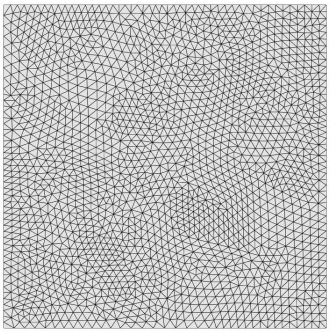
`carre4h.mesh`



`carre2h.mesh`

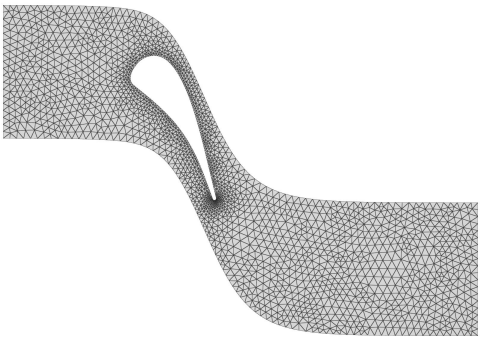


`carreh.mesh`

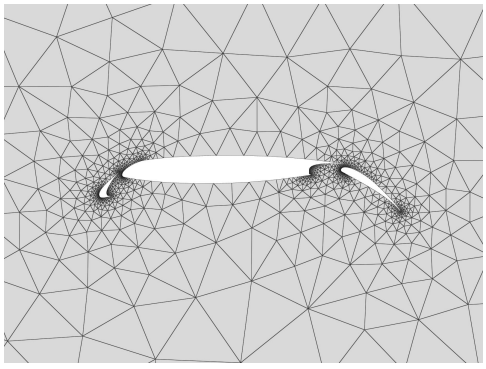


`carre05h.mesh`

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

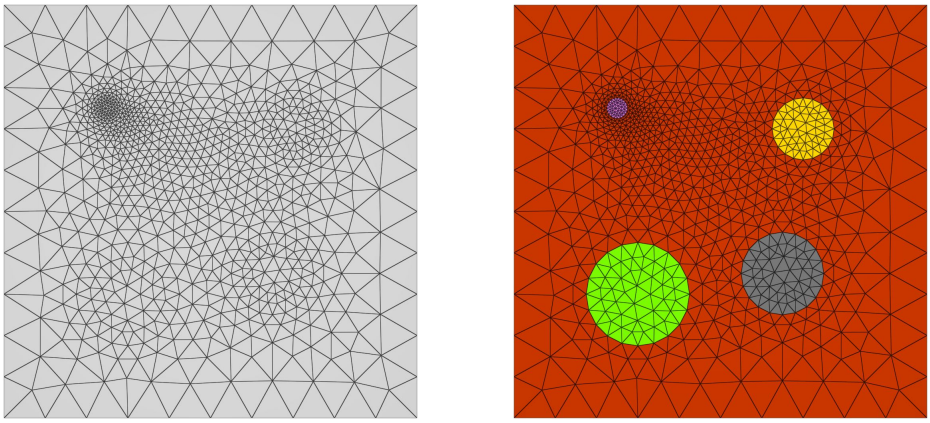


`ls89.mesh`



`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
```

Le niveau de gris des images sera stocké dans un fichier au format `.sol` de la forme suivante :

```
MeshVersionFormatted 2
Dimension 2
```

```
# Set of grey-level value for each vertex
SolAtVertices
581
1 1
12.
255.
.........

End
```