MEC6212: GENERATION DE MAILLAGES

Travail pratique: Triangulation de Delaunay contraint: traitement des collisions

29 février 2024

Énoncé

Dans ce travail on complète l'algorithme de maillage Delaunay contraint du travail précédent en y ajoutant le traitement des collisions. Cette opération est réalisée par la fonction **delFRNT3** décrite schématiquement à la Fig. 3, et comprend essentiellement les étapes suivantes :

Localisation de *posFRNT* On localise le sommet sur le front avec le plus petit angle, position où sera construit l'élément ;

Partie finissante avec un triangle On analyse le front et teste si le front se referme avec trois arêtes. Alors, on construit un élémente et alors le front devient vide et on le ferme.

Partie finissante avec un quadrangle On analyse le front et teste si le front se referme avec quatre arêtes. Alors on forme deux triangles selon le critère de la sphère vide. Alors le front devient vide et on le ferme.

.

Construction d'un nouvel element Si le front n'est pas finissant, on construit un élément candidat. Deux situations sont possibles,

- le triangle candidat est vide, alors le triangle est construit, suivi de la modification du front. (Travail pratique précédent)
- le cercle du triangle circonscrit n'est pas vide, ce qui est considéré comme une collision. Alors, le front est scindé en deux parties.

L'objectif est de détecter et de corriger les collisions du front avec lui-même ou bien avec une frontière interne, situation qui produit un chevauchement et donc un maillage non-valide.

1 Contexte

Un exemple de cette situation apparaît lorsque le domaine est non-simplement connexe, c-à-d avec des trous tel qu'illustré à la Fig. 1

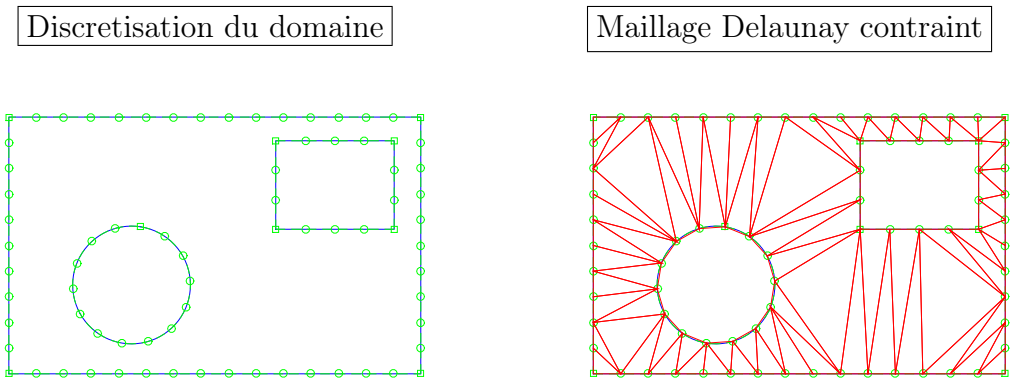
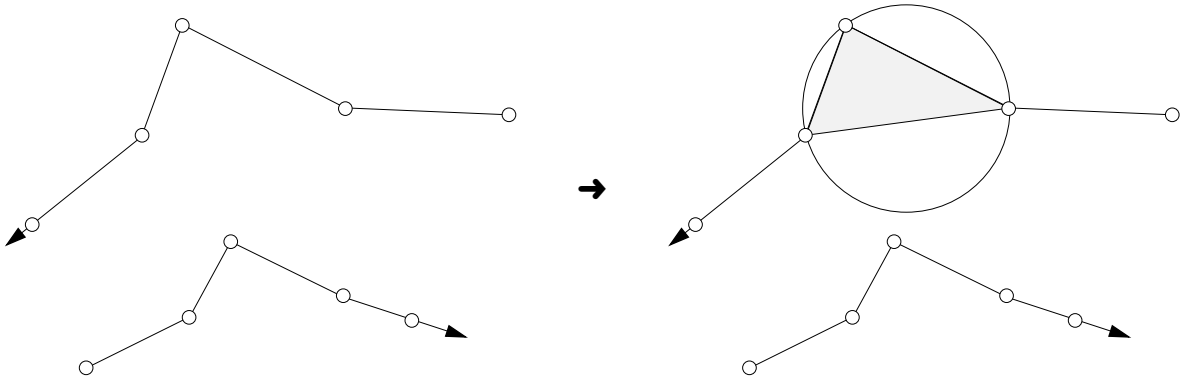


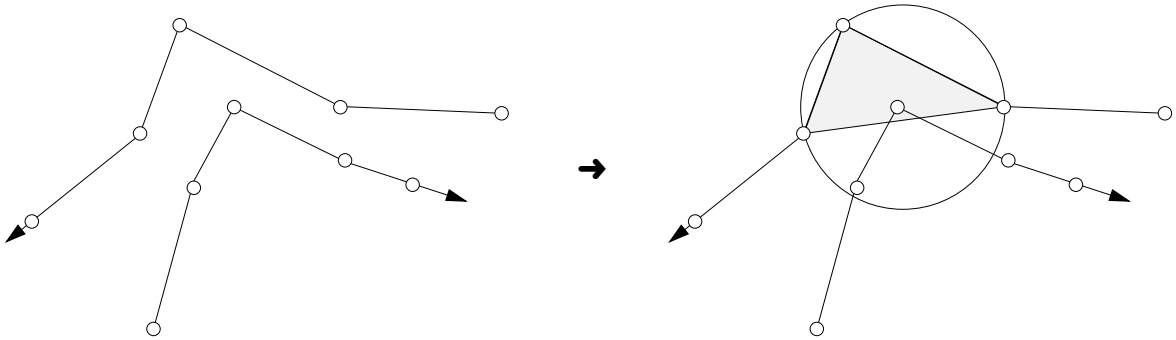
FIGURE 1 – Delaunay contraint pour domaine non-simplement connexe

On illustre trois différentes possibilités lors d'une telle collision :

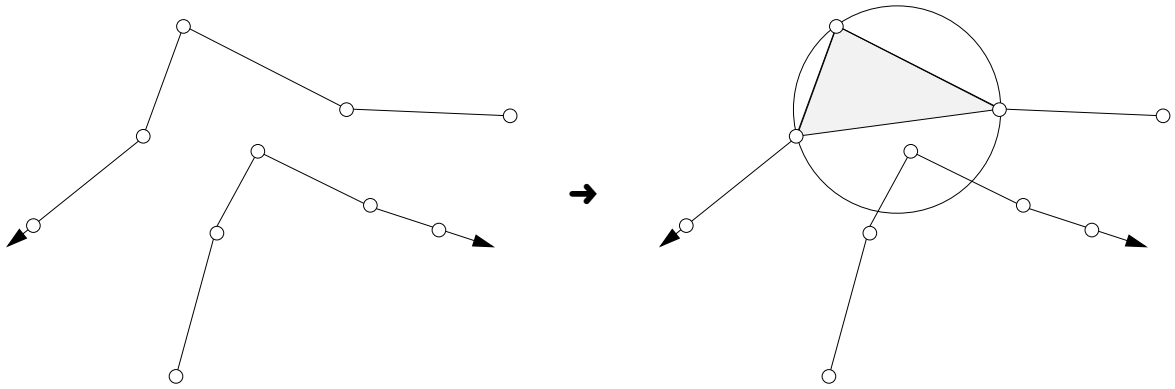
1. le triangle candidat est vide, **ainsi** que son cercle circonscrit ;



2. le triangle candidat contient un somment du front, ce qui produit un maillage non-valide ;



3. le triangle est vide, mais le cercle circonscrit contient un sommet : ce qui n'est pas strictement une collision. Cela produira un triangle valide mais avec des éléments étirés lors des étapes subséquentes.



La Fig. 2 illustre comment on peut corriger ces deux dernières situations en tirant une diagonale du somme candidat vers le sommet du front inscrit dans le cercle circonscrit.

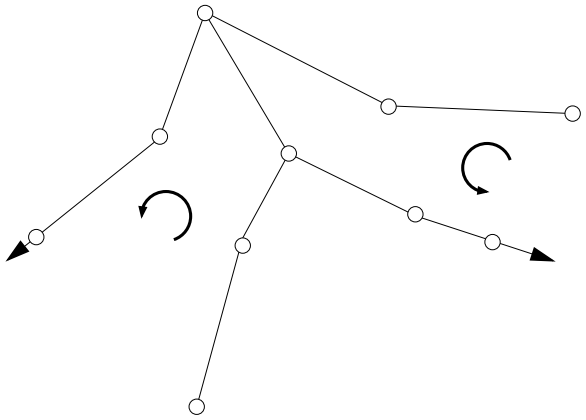


FIGURE 2 – Modification qui scinde le front en deux parties lors d'une collision

Cette construction a pour effet de scinder le front en deux parties, de chaque coté de la diagonale, et nécessite une ré-organisation du front réalisée par la fonction **FRNTdiag** qui établie une nouvelle configuration suite à la collision.

La détection de la collision ainsi que le traitement du front sont initiés par la fonction **delFRNT3** (Voir Fig. 3).

```
function delFRNT3(hObject, handles)
%-------------------------------------------------------------------------
global x y nbNOD ARE nbARE ARElong
global E nbELM ELMgeo
global FRNT FRNTdim FRNTactif FRNTlong FRNTangl  FRNTindex
%-------------------------------------------------------------------------
while FRNTactif > 0
    [~,posANGL]=min(FRNTangl(1:FRNTactif));%---localisation de posFRNT
    posFRNT=FRNTindex(posANGL);
    posSOM=FRNT(posFRNT,1);
    preFRNT=FRNT(posFRNT,2);
    preSOM=FRNT(preFRNT,1);
    prepreFRNT=FRNT(preFRNT,2);
    prepreSOM=FRNT(prepreFRNT,1);
    suiFRNT=FRNT(posFRNT,3);
    suiSOM=FRNT(suiFRNT,1);
    suisuiFRNT=FRNT(suiFRNT,3);
    suisuiSOM=FRNT(suisuiFRNT,1);
    if prepreSOM==suisuiSOM%------   partie du FRNT forme un quadrangle
        ------------------------------------
        ------ votre code: TP precedent
        ------------------------------------
    elseif preSOM==suisuiSOM%---------- partie du FRNT forme un triangle
        ------------------------------------
        ------ voir TP precedent ------
        ------------------------------------
    else%------------------------------- cas general: avec test decollision
        minFRNT=videELM2(preSOM,posSOM,suiSOM);
        if minFRNT==0%--------------------- le triangle candidat est vide
       ------------------------------------------
       ------ votre code  TP precedent ------
       ------------------------------------------
            afficheFRNT_EvS(hObject, handles)
            pause
        else
        %---- le triangle candidat non vide: on construit la diagonale
        %---------------------------------------- et le front est scinde

        FRNTdiag(minFRNT,preSOM,posSOM,suiSOM,posFRNT,preFRNT);

       afficheFRNT_EvS(hObject, handles)
       pause
       end
    end
end
```


FIGURE 3 – Fonction **delFRNT3**

2 Mise en oeuvre

1. La collision est détectée par la fonction :

   minFRNT=videELM2(preSOM,posSOM,suiSOM)

   où les arguments *(preSOM,posSOM,suiSOM)* sont les sommets du triangle cible. Cette fonction retourne *minFRNT* qui représente la position intersectée, ou bien 0, si le triangle est vide.

2. Le traitement de la collision est réalisé par la fonction,

   FRNTdiag(minFRNT,preSOM,posSOM,suiSOM,posFRNT,preFRNT)

   La Fig. 4 montre le protocole d'appel et prend en arguments la configuration de la collision, notament, la position du front *minFRNT* qui entre en collision avec le triangle candidat.

```
function FRNTdiag(minFRNT,preSOM,posSOM,suiSOM,posFRNT,preFRNT)
global FRNT FRNTdim FRNTactif FRNTlong FRNTangl FRNTindex
global ARE AREbcl nbARE ARElong
global x y nbNOD

minSOM=FRNT(minFRNT,1);
suiminFRNT=FRNT(minFRNT,3);
suiminSOM=FRNT(suiminFRNT,1);
preminFRNT=FRNT(minFRNT,2);
preminSOM=FRNT(preminFRNT,1);
%------------------------------ ajout d'une nouvelle position-1
FRNTdim=FRNTdim+1;
FRNTactif=FRNTactif+1;
        ------------------------------------------------
        ---------------- a completer-------------------
        ------------------------------------------------
%------------------------------------- ajout nouvelle position-2
FRNTdim=FRNTdim+1;
FRNTactif=FRNTactif+1;
        ------------------------------------------------
        ---------------- a completer-------------------
        ------------------------------------------------
```

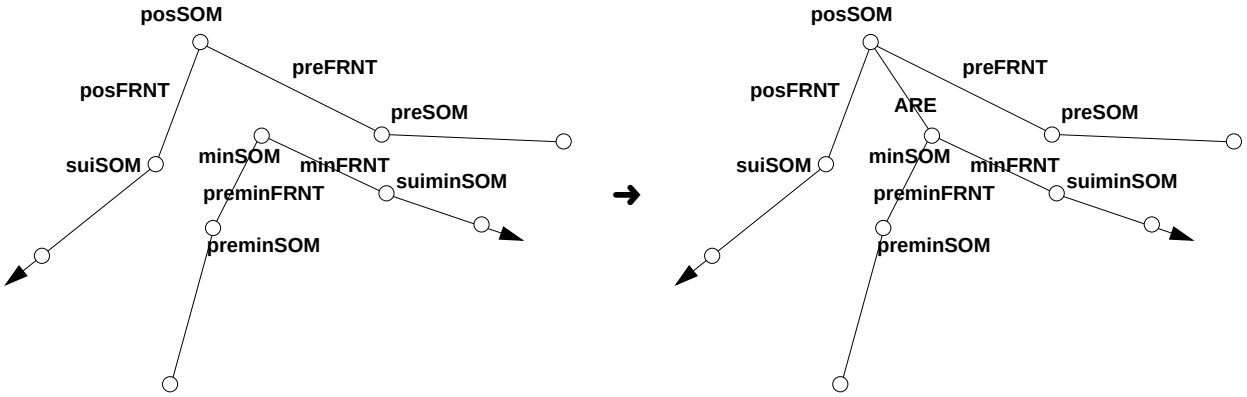
FIGURE 4 – Fonction **FRNTdiag**

Ces deux fonctions sont appellées par,

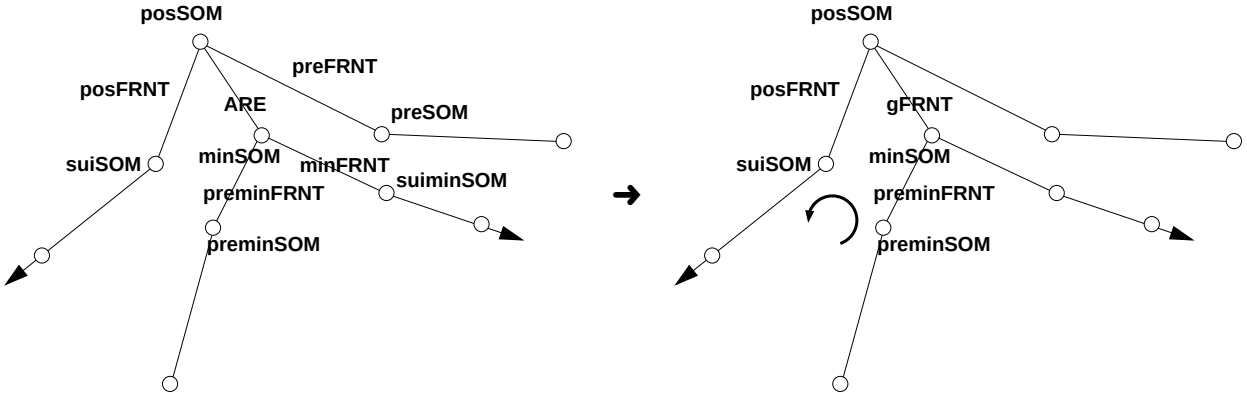
function delFRNT3(hObject, handles)(Voir Fig. 3)

L'algorithme comprend les étapes suivantes :

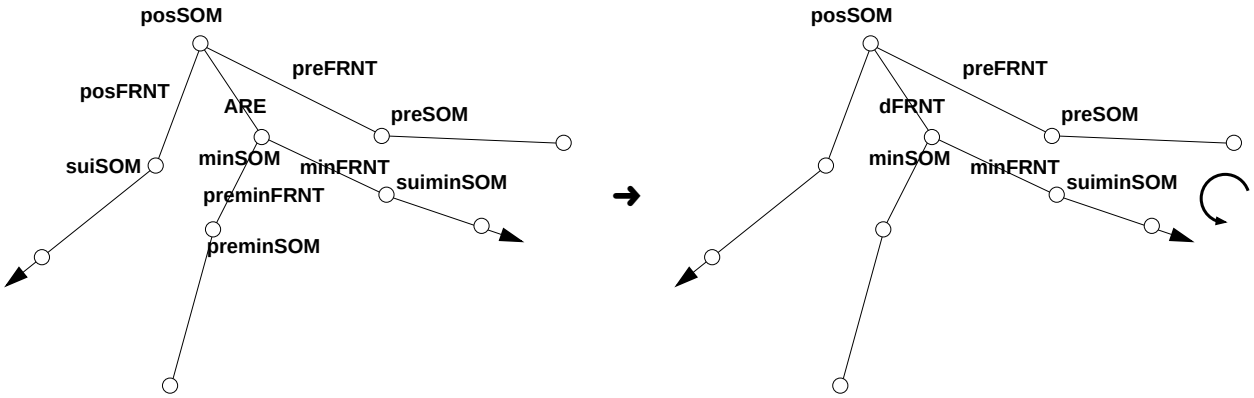
1. **Le front est scindé en deux parties par l'insertion d'une arête** ARE :



2. **Première partie : on insère une nouvelle position *gFRNT* entre le sommet cible et la collision (coté gauche) :**



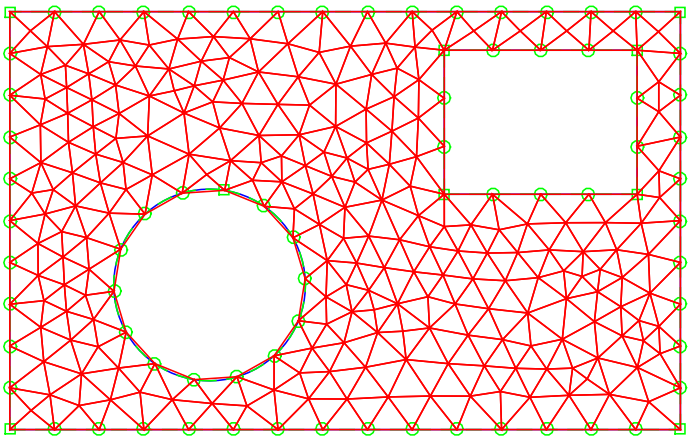
3. **Deuxième partie : on insère une nouvelle position *dFRNT* entre le sommet cible et la collision (coté droite) :**



3 Validation

1. Construire un domaine semblable à celui de la Fig. 1 ;
2. Lancer le mailleur Delaunay contraint avec la fonction **delFRNT3.m**, tel que modifiée, c-à-d **delFRNT3.p** ;

   DelaunayEvS ==> Delaunay contraint EvS==> **Demo3**
3. Valider, avec l'exécution pas-à-pas, que les collisions sont correctement résolues.
   Prendre une discrétisation grossiére .3 ou .4
4. Répéter en variant la cible de la taille des triangles. Pour cette étape, il est préferable de ne pas utiliser le mode pas-à-pas, mais de lancer l'algorithme avec le bouton **EvS ELMmax** et **delFRNT3.p** qui utilisera votre fonction.



Sujets de discussion :

1. L'algorithme de Delaunay contraint utilise une technique d'avance de front et dont la séquence procède dans l'ordre du plus petit angle, pour obtenir un premier recouvrement, suivi d'un rafinement basé sur le plus grand triangle.
   Quelles autres stratégies pourraient être utilisées ?
2. Une collision est détectée lorsque le cercle circonscrit contient un sommet du front.
   Que faire s'il y a plusieurs sommets ?
3. Robustesse ? limites ? coût ?