M2 - MPE, UPMC
Cours de C++

Teammates: MISRACHI, BEN JEBRIA
Date: 30/01/2018

Projet de C++ : Recherche rapide d'un triangle contenant un point dans un maillage

Introduction

Le but de ce projet est de mettre en place un algorithme de recherche rapide d'un triangle contenant un point dans un maillage. Une idée très basique, mais loin d'être optimisée du point de vue de sa complexité algorithmique pourrait consister à parcourir tous les triangles du maillage et vérifier si oui ou non le point y appartient (complexité en $O(n_T)$, où n_T est le nombre de triangle du maillage). Le but de ce projet est de mettre en oeuvre un algorithme de recherche plus efficace, dont la complexité est en $O(log(n_T))$

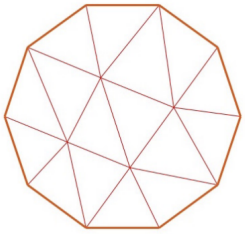

Figure 1: **Exemple de maillage conforme.**

Pour ce projet, nous allons nous limiter à l'étude de maillages conformes simples. Ce dernier sera composé de n_T triangles τ_h, telle que l'intersection de deux triangles distincts soit une arête commune, un sommet commmun ou rien, où l'ensemble des n_S sommets de τ_h est notée S_h. De plus, nous excluons de cette étude tous les maillages qui présentent des trous en leur sein ou des éléments non triangulaires.

List of Figures

Contents

1 Description globale et générale de l'algorithme de recherche

Dans cette section, nous allons décrire l'idée générale de l'algorithme de recherche rapide d'un triangle contenant un point dans un maillage. La présentation de ce "squelette" initial nous permettra de bien comprendre comment construire nos classes d'objets et leurs méthodes associées. L'idée globale de l'algorithme est de se déplacer de triangle en triangle de façon intelligente en se rapprochant grossièrement du point que l'on recherche. Cela est rendu possible par le calcul des aires signées par paire de deux sommets d'un triangle et du point dont on souhaite déterminer le triangle d'appartenance. La figure ci-dessous résume visuellement l'intuition liée à ce déplacement intelligent.

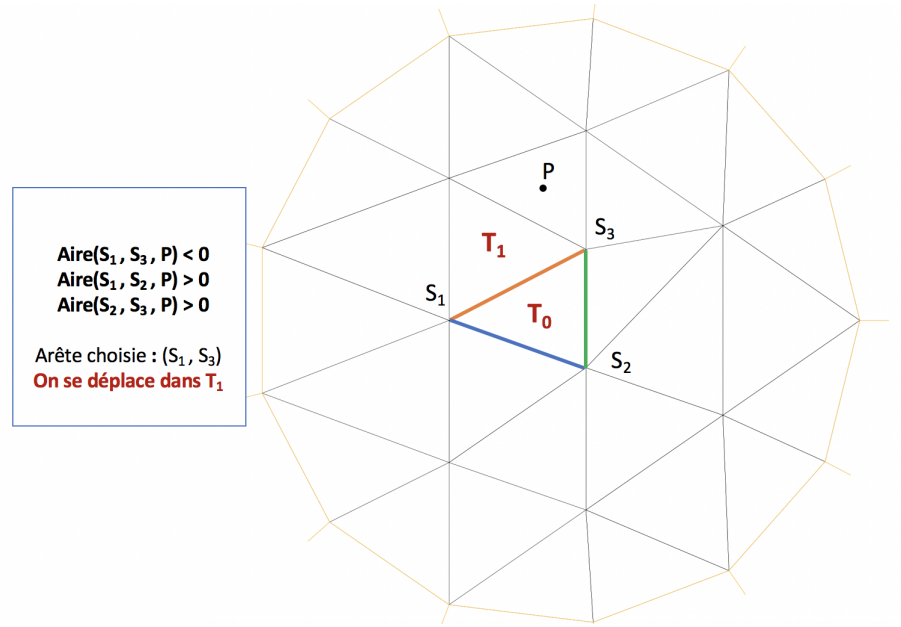

Figure 2: **Déplacement de triangle par le calcul des aires signées.**

L'arête (S_1, S_3) étant la seule à produire une aire signée négative avec le point P, on se déplace pour sur vers le triangle adjacent associé à cette arête, en l'occurence T_1. On constate qu'on se rapproche effectivement du point P.

Il faut bien évidemment pouvoir incorporer les cas plus particulier des triangles de bord, pour lesquels certaines arêtes ne possèdent pas de triangle adjacent. L'algorithme présenté page suivante résume les différentes étapes de la recherche, en incorportant ces cas extrêmes. Au vu de cet algorithme, il nous faut déterminer:

- des classes ***Maillage***, ***Triangle*** et ***Sommet***, que nous allons détailler.
- une fonction ***ConstructionArete*** qui lit notre maillage et construit les arêtes.
- une fonction ***Adjacent*** de la classe ***Maillage*** qui détermine le triangle adjacent à une arête.
- une fonction ***CalculAire*** qui calcule l'aire signée à partir de trois points de $\mathbb{R}^2$.
- une fonction ***MiseAJour*** qui correspond à l'algorithme 1 présenté ci-après.

Algorithm 1 *Recherche rapide d'un triangle contenant un point dans un maillage.*

On se donne un point P de $\mathbb{R}^2$ ▷ Point que l'on recherche
Partant du triangle K ▷ Initialisation triangle
Trouvé $\leftarrow False$ ▷ Le triangle n'est pas trouvé à ce stade

while Trouvé $== False$ **do** ▷ Tant que l'on a pas trouvé le triangle
  for (a_i, b_i), $i \in [0, 1, 2]$ les 3 arêtes du triangle K **do** ▷ Dans le sens trigonométrique
    Calculer A_i l'aire signée du triangle (a_i, b_i, P)
  if les 3 aires A_i sont positives **then**
    $P \in K$, Trouvé $\leftarrow True$ ▷ On est dans le bon triangle
  else if 2 aires A_i sont négatives **then**
    Choisir une des 2 arêtes (a_i, b_i) au hasard
    if cette arête j est au bord (pas de triangle adjacent) **then**
      Choisir l'autre arête k
      if arête au bord **then**
        "Le point est en dehors du maillage"
        Trouvé $\leftarrow True$
    else
      Se rendre dans son triangle Adjacent Q, $K \leftarrow Q$ ▷ On change de triangle
  else 1 seule aire est négative
    if arête associée au bord **then**
      "Le point est en dehors du maillage"
      Trouvé $\leftarrow True$
    else
      Se rendre dans son triangle Adjacent Q, $K \leftarrow Q$ ▷ On change de triangle

2 Description des classes *Maillage*, *Sommet*, *Triangle*

@

2.1 Classe *Sommet*

Cette classe, héritée de la classe $\mathbb{R}^2$ de C++ possède les attributs suivants:

- **lab**: un entier qui vaut 0 ou 1 pour indiquer si le sommet se trouve au bord du maillage ou non.
- **nu** : un entier qui fournit le numéro global du sommet dans le maillage
- **point** : un point de $\mathbb{R}^2$ qui contient les coordonnées du sommet.

2.2 Classe *Triangle*

Cette classe possède un unique attribut **s**. Il s'agit d'un tableau de trois pointeurs vers des objets de type ***Sommet***: les trois sommets du triangle. Nous avons également définis de nouveaux opérateurs:

- l'opérateur [**i**], avec i un numéro local $\in [0, 1, 2]$ qui renvoie le numéro global du sommet i local.
- l'opérateur (**i**), avec i un numéro local $\in [0, 1, 2]$ qui renvoie l'attribut point ($\mathbb{R}^2$) du sommet local i.

2.3 Classe *Maillage*

Cette classe possède les attributs suivants:

- $\mathbf{n_S}$: le nombre de sommets du maillage
- $\mathbf{n_T}$: le nombre de triangles du maillage
- **s** : le tableau de sommet du maillage
- **t** : le tableau de triangles du maillage

Nous avons également définis/surchargés certains opérateurs:

- l'opérateur **(i,j)** qui renvoie le numéro global du sommet j ($j$en numérotation locale) du triangle i (i en numérotation globale).
- l'opérateur **[i]** qui renvoie le triangle i (i en numérotation globale) du maillage.

Nous avons également défini certaines méthodes:

- **get_point**$(\mathbf{i}, \mathbf{j})$ qui renvoie le point de $\mathbb{R}^2$ du sommet j (j en numérotation locale) du triangle i du maillage (i en numérotation globale).
- **Adjacent** qui renvoie le numéro global du triangle adjacent à une arête (paire de sommet). Cette fonction sera détaillée dans la suite du rapport.
- **MiseAJour** qui renvoie le numéro global du triangle qui contient notre point. Cette fonction sera également détaillée plus amplement dans le suite du rapport.

3 Fonction *ConstructionArete*

La fonction ConstructionArete répertorie les arêtes du maillage et permet d'associer à chaque arête le ou les triangles qui la contiennent.

La fonction prend en argument:

- **const Maillage** & **Th** : un maillage qui ne subira aucune modification
- **int** (∗**arete**) [**2**] : un pointeur vers un tableau d'entier de taille 2. Ce tableau va être rempli avec le label des sommets globaux afin de répertorier les arêtes.
- **std** :: **multimap** < **std** :: **pair** < **int**, **int** >, **int** > & **arete_tab** : une multimap qui permet d'associer à chaque arête un ou des triangles. De plus, **std** :: **pair** < **int**, **int** >correspond à un couple de deux sommets qui nous définit donc une arête.

Dans cette fonction, on parcourt chaque triangle du maillage, puis on parcourt les 3 arêtes du triangle. Ensuite, on ajoute dans la multimap le couple arête-triangle. Enfin, on teste si l'arête est déjà dans le tableau arete, on l'ajoute le cas échéant.

Cette fonction renvoie un entier qui correspond au nombre d'arêtes dans le maillage.

4 Fonction *Adjacent*

Le but de la méthode Adjacent de la classe Maillage, c'est à partir d'un triangle T et d'une arête A donnés, de retourner l'unique triangle s'il existe, adjacent à T par A.

Cette fonction prend en argument:

- **int triangle** : un entier correspondant au triangle courant,
- **std** :: **pair** < **int**, **int** > **arete** : une paire d'entier qui représente l'arête,
- **std** :: **multimap** < **std** :: **pair** < **int**, **int** >, **int** > & **m** : une multimap contenant les numéros de tous les triangles associés à chaque arête. Cette arête est initialisée en dehors de la fonction. En effet, elle est initialisée dans la fonction main.

Cette fonction renvoie un entier qui correspond au numéro du triangle adjacent au triangle courant.

5 Fonction *Aire*

Cette fonction calcule l'aire signée d'un triangle formé par trois points de $\mathbb{R}^2$. La fonction possède trois arguments qui sont les entiers

6 Fonction *MiseAJour*

La méthode de la classe Maillage MiseAJour applique l'algorithme voulu; c'est à dire à partir des arguments :

- **R2 P** : un point P de la classe R2,
- **int point_depart** : le numéro du triangle où l'on commence la recherche,
- **std** :: **multimap** < **std** :: **pair** < **int**, **int** >, **int** > & **M** : la multimap.

Cette fonction renvoie le numéro du triangle contenant le point P.

Comme décrit à la page 6, tant que la variable booléenne **trouve** est fausse on répète ce qui suit:

- On calcule les aires signés des trois triangles formés par le point P et deux des trois sommets du triangle courant, à l'aide de la fonction **calcul_aire** appliqué aux trois points correspondants.
- Ensuite, si les trois aires sont positives, nous nous trouvons dans le bon triangle et on arrête donc l'algorithme.
- Si une seule des aires est négative, aors il n'y a qu'une seule direction possible. On applique ensuite la fonction **Adjacent** pour se déplacer dans le triangle suivant.
- Si les deux aires sont négatives, il y a alors deux directions possibles, une seule est choisie aléatoirement grâce à la fonction **rand**.

Dans le cas où le point P est en dehors du maillage l'appel à la fonction **Adjacent** retourne -1, ce qui permet de traiter également ce cas.

7 Recherche des triangles d'un maillage contenant les sommets d'un autre maillage

Dans cette question, on souhaite mettre en place un algorithme qui permette de déterminer, pour chaque sommet d'un certain maillage T_{h_2}, les triangles d'un autre maillage T_{h_1} qui le contiennent. Afin de briser la complexité algorithmique, nous allons tirer profit de l'assertion suivante: *les sommets d'un triangle sont proches car les triangles sont petits.*.

Une idée basique, mais non optimisée du point de vue de son coût algorithmique serait de parcourir chacun des sommets du maillage T_{h_2} et d'appeler la fonction ***MiseAJour*** sur le maillage T_{h_1}, pour les différents sommets de T_{h_2} en question afin de déterminer dans quel triangle de T_{h_1} ils se trouvent.

Afin de tirer profit de la remarque précédente, nous avons décidé de parcourir les sommets de T_{h_2} triangle par triangle. Effectivement, l'idée est d'appeler la fonction ***MiseAJour*** sur un des premiers sommets de ce triangle afin de déterminer le triangle de T_{h_1} qui le contient. Cet étape est à priori coûteuse puisqu'on part d'un triangle aléatoire de T_{h_1} qui est peut-être très éloigné du sommet de T_{h_2} considéré. Cependant, une fois que nous avons pu déterminer le triangle T_1 de T_{h_1} qui contient le premier sommet d'un triangle T_2 de T_{h_2}, on comprend que les autres sommets de T_2 ont de grandes chances de se trouver dans un triangle très proche de T_1

(peut-être même T_1 lui-même). Pour ces étapes, on appelle toujours la fonction ***MiseAJour***, mais depuis T_1, ce qui rend l'algorithme de recherche très peu coûteux, puisqu'en quelques itérations seulement, le bon triangle devrait être trouvé. Pour chaque triangle parcouru, on devrait donc avoir une étape à priori coûteuse et les deux autres bien plus rapides.

De plus, au fur et à mesure de la recherche, lorsque de nouveaux triangles sont parcourus, il se peut que leurs sommets aient déjà été passés en revue (puisqu'un sommet peut être dans plusieurs triangles). Dans ce cas, on choisira, pour les autres sommets non visités du triangle une initialisation intelligente du triangle de départ de T_{h_1} lors de l'appel de la fonction ***MiseAJour*** : le triangle dans lequel se trouve le sommet déjà visité.

Algorithm 2 *Recherche des triangles d'un maillage T_{h_1} contenant les sommets d'un autre maillage T_{h_2}*

M_1 multimap contenant les arêtes et triangles associés à T_{h_1}
M_2 multimap contenant les couples $(sommets \in T_{h_2}, triangles \in T_{h_1})$
"
for t triangle de T_{h_2} **do**
    if un des sommets s du triangle $t \in T_{h_2}$ a déjà été trouvé : $s \in T_1$ $(T_1 \in T_{h_1})$ **then**
        $T_{init} \leftarrow T_1$
    for s sommet de t **do**
        if s non repertioré dans la multimap M2 **then**
            $T_{final} \leftarrow MiseAJour(s, T_{init}, M_1)$
            $M_2.add((s, T_{final}))$
    if $T_{final} != -1$ **then**
        $T_{init} \leftarrow T_{final}$

8 Présentation des résultats sur cas test

- **Test de la fonction MiseAJour**:

- Cas d'un point appartenant au maillage:
  On considère le maillage Th-13-2.msh. On teste la fonction MiseAJour au point P de coordonnées (0,5; 1,2). La fonction nous renvoie que le point P appartient au triangle numéro 0 du maillage. Les coordonnées des trois sommets du triangle numéro 0 sont : (0,19386;1,24669) , (1,09693;0,623344), (1,13613;1,64597). On peut vérifier que le point P appartient bien au triangle numéro 0.
  Ainsi, la fonction MiseAJour fonctionne lorsque l'on cherche un point dans le maillage

- Cas d'un point en dehors du maillage:
  On considère le maillage Th-13-2.msh. On teste la fonction MiseAJour au point P de coordonnées (10;0) dont on sait qu'il se trouve en dehors du maillage. La fonction nous renvoie bien la réponse attendue; c'est à dire que le point se trouve en dehors du maillage.

- **Test de la fonction Question4**:
  On considère les maillages Th-13-2.msh et Th-20-1.msh correspondant respectivement aux maillage Th1 et Th2. On teste la fonction Question4 pour ces deux maillages. On sait que le maillage Th-20-1 a 45 sommets. La fonction nous renvoie bien pour chacun des 45 sommets le triangle de Th-13-2 qui lui est associé.