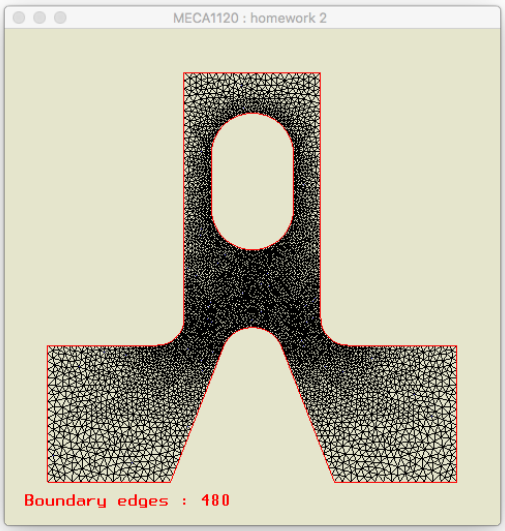
Finite elements for dummies : Déduire la frontière d'un maillage !



Il s'agit maintenant d'estimer la longueur de la frontière d'un maillage quelconque, que ce soit un profil d'aile NACA0012, un quart de passoire ou une pièce mécanique compliquée. L'objet de ce devoir est de vous faire manipuler la structure topologique du maillage qui est un graphe un peu particulier.

Il s'agit d'obtenir tous les segments sous la forme :

```
typedef struct {
   int elem[2];
   int node[2];
} femEdge;

typedef struct {
   femMesh *mesh;
   femEdge *edges;
   int nEdge;
   int nBoundary;
} femEdges;
```

L'ensemble de tous les segments est décrit dans `femEdges` qui contient le nombre total de segments, le nombre de segments frontières, un pointeur vers un maillage et un pointeur vers un tableau dont chaque élément est une structure `femEdge`. Chaque segment contiendra les indices des deux noeuds qui le forme et les indices des deux éléments qui lui sont voisins. Si le segment est sur la frontière, l'indice du second élément sera égal à `-1`. La suite des noeuds du segment sera donnée dans le même ordre que celui du premier élément (et donc dans l'ordre inverse pour le second élément si le maillage ne contient que des éléments numérotés anti-horlogiquement). On supposera que les maillages testés n'ont pas de défauts pathologiques. Les maillages sont conformes : un segment appartient uniquement à un ou à deux éléments :-) Le tableau d'appartenance des sommets aux éléments pourra être défini avec une orientation arbitraire mais commune à tous les éléments d'un maillage. Pour réaliser cette mission, voici ce qui est fourni par nos bons soins :

- Le fichier `homework.c` est une version vide du devoir à réaliser. Les cinq fonctions sont définies mais ne font pas vraiment ce qui est prévu.
- Le fichier `fem.c` a le même rôle que lors de la précédente soumission. Quelques petites fonctions y ont été ajoutées, mais cela ne devrait pas trop vous perturber. On y a inclus une fonction `femEdgesPrint` pour imprimer à l'écran la structure de données des segments. On vous conseille vivement de l'utiliser pour la mise au point de votre programme.
- Le fichier `glfem.c` permet d'afficher les résultats obtenus.
- Les fichiers `gear8.txt`, `gear12.txt`, `gear16.txt`, `gear60.txt`, `conge.txt`, `holes.txt` et `naca.txt` contiennent respectivement un maillage de triangles d'engrenages, d'une pièce quelconque, d'un quart de passoire et d'un carré contenant un profil NACA0012, tandis que l'exemple présenté au cours est le fichier `stupid.txt`. Il est vraiment conseillé d'utiliser le maillage élémentaire `stupid.txt` pour mettre au point votre code avant de le tester sur les maillages des autres pièces !

Plus précisément, on vous demande de :

1. Tout d'abord, écrire une fonction

   ```
   void   edgesExpand(femEdges *theEdges)
   ```

   qui remplit la structure de données en reprenant tous les segments définis sur tous les éléments du maillage. On parcourt chaque élément et on augmente progressivement le nombre de segments en y enregistrant les deux sommets et le numéro de l'élément courant. Seul le premier élément de la structure contient une valeur et le second est initialisé à la valeur `-1`. En d'autres mots, tous les segments sont des segments frontières à l'issue de cette étape. Se référer à l'exemple du cours pour bien comprendre à quoi doit ressembler la structure prévue.

2. Ensuite, adapter la fonction

   ```
   int    edgesCompare(const void* e0, const void *e1)
   ```

   qui doit permettre d'effectuer un tri des segments. Les segments seront classés par leur indices de noeuds. On tiendra d'abord compte du plus petit noeud et ensuite du plus grand noeud pour effectuer la sélection du classement. La fonction `edgesCompare` n'est pas adéquate et doit donc être modifiée. Par contre, la fonction `edgesSort` ne doit pas vraiment être modifiée !

3. La dernière étape de l'algorithme de génération de la structure des segments sera faite en définissant

   ```
   void   edgesShrink(femEdges *theEdges)
   ```

   qui supprime les doublons de la liste triée des segments. Elle adapte aussi le nombre des segments frontières et met à jour le second indice des éléments pour les segments non frontières. A la fin de cette étape, vous devez avoir tous les segments dans votre structure de données, pas uniquement les segments frontières.

4. Finalement, écrire une fonction

   ```
   void   double edgesBoundaryLength(femEdges *theEdges)
   ```

   qui fournit le longueur de la frontière de la pièce (ou de l'environnement de la pièce dans le cas du NACA0012). Pour ce calcul, on calculera la longueur de chaque segment simplement comme un segment de droite linéaire.

5. Un morceau de code `main.c` vous est fourni pour tester votre fonction.

```
#include "fem.h"

int main(void)
{
    femMesh  *theMesh  = femMeshRead("../data/stupid.txt");
    femEdges *theEdges = femEdgesCreate(theMesh);

    edgesExpand(theEdges);               //   femEdgesPrint(theEdges);
    edgesSort(theEdges);                 //   femEdgesPrint(theEdges);
    edgesShrink(theEdges);               //   femEdgesPrint(theEdges);
    printf("Boundary edges  : %i \n", theEdges->nBoundary);
    printf("Boundary length : %14.7e \n", edgesBoundaryLength(theEdges));

    char theMessage[256];
    sprintf(theMessage, "Boundary edges : %i", theEdges->nBoundary);
    femEdgesFree(theEdges);
    femMeshFree(theMesh);
    exit(0);
}
```

6. Vos quatre fonctions seront incluses dans un unique fichier `homework.c`, sans y adjoindre le programme de test fourni ! Ce fichier devra être soumis via le web et la correction sera effectuée automatiquement. Il est donc indispensable de respecter strictement la signature des fonctions. Votre code sera strictement conforme au langage `C` et il est fortement conseillé de bien vérifier que la compilation s'exécute correctement sur le serveur.

7. Il est impératif qu'aucune des fonctions n'arrête le flux du programme, quel que soit le problème rencontré. En cas d'arguments incompatibles, vous pouvez imprimer un message d'erreur et renvoyer une valeur nulle si nécessaire.

8. Pour votre facilité, les instructions de ré-allocation de mémoire ont été incluses dans le fichier `homework.c`. Il n'est ni utile, ni conseillé de les modifier !

9. Afin de pouvoir effectuer un test distinct de vos fonctions, des instructions de compilation ont été mis dans le fichier `homework.c`. Il est IMPERATIF de ne pas les retirer, ni de les modifier...