Politecnico di Torino

Corso di laurea triennale in matematica per ingegneria

Matteo Galla s248041

Pietro Gadaleta s245085

Lorenzo Huang s246874

Anno accademico 2018/2019

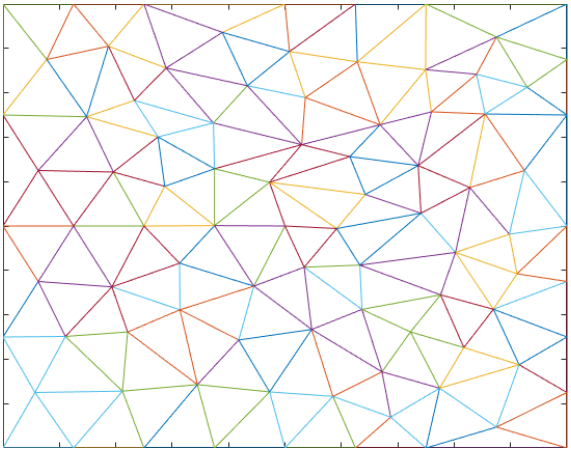
Relazione sullo sviluppo del progetto C++:

Rifinitura di una mesh triangolare

Introduzione

Il progetto che abbiamo deciso di sviluppare consiste nello scrivere una serie di funzioni capaci di rifinire un dominio bidimensionale, il quale ci viene fornito come materiale.

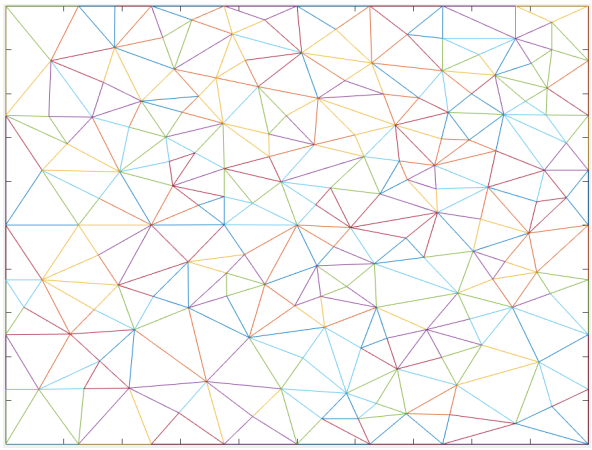
Nell'immagine possiamo vedere che il dominio preso è un quadrato 1x1; come le funzioni fornite, il quadrato viene tagliato in triangoli diversi.



Il nostro compito è quello di tagliare ulteriormente il dominio, per fare ciò abbiamo scritto una serie di funzioni con diverse funzionalità, come:

- Marcare le basi dei triangoli marcati per il raffinamento.
- Raffinare la mesh triangolare in modo da ottenere una mesh conforme. Per ottenere tale risultato una volta tagliato il lato di un triangolo lungo il suo punto medio, lo disattiviamo, iterando questo processo finché non ci sono più lati del triangolo da tagliare (tale operazione va effettuata per ogni triangolo della mesh). Contestualmente si aggiornano anche le informazioni di vicinanza per i lati della triangolazione.
- Aggiornare le informazioni di vicinanza per i triangoli.
- Raffinare uniformemente la triangolazione (ogni triangolo è diviso in 4 triangoli simili unendo i punti medi dei lati del triangolo padre).

L'immagine seguente mostra il risultato del raffinamento.



Materiali forniti

Ci vengono forniti i seguenti file:

- ConfigFile.cpp
- GenericDomain.cpp
- GenericMesh.cpp
- main.cpp
- MeshImport_Triangle.cpp
- Output.cpp
- triangle.c

- ConfigFile.hpp
- GenericDomain.hpp
- GenericMesh.hpp
- MacroDefinitions.hpp
- MeshImport_Triangle.hpp
- mytriangle.hpp
- Output.hpp
- triangle.h

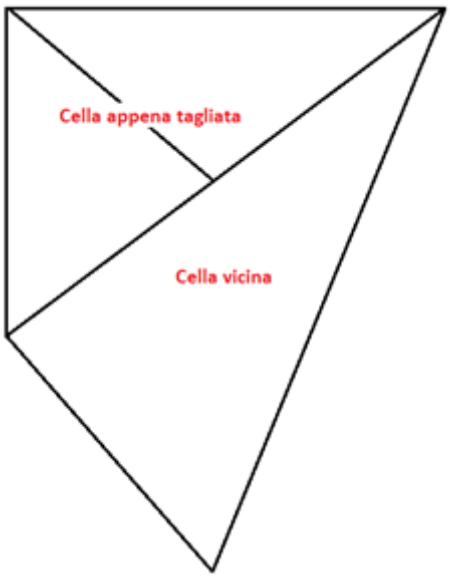
I quali contengono le classi per la generazione di una mesh triangolare in un dominio poligonale. I triangoli sono sempre “orientati” in senso antiorario.

Strategia

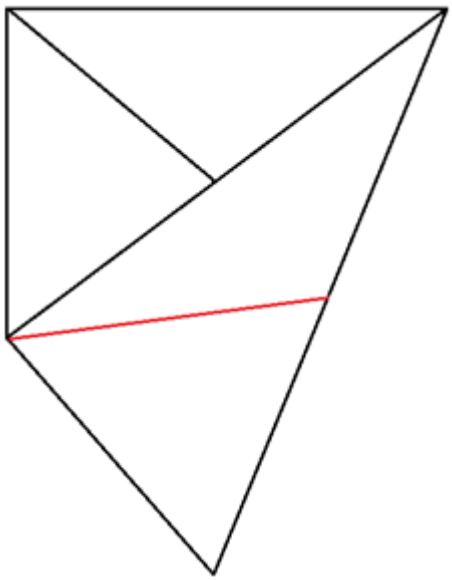
Come criterio di raffinamento abbiamo deciso di utilizzare “Longest edge bisection”: quando una cella viene selezionata per il raffinamento, come prima operazione eseguiamo una rotazione dei lati e dei vertici in modo da avere il lato più lungo come lato 0 ed il vertice iniziale di tale lato come vertice 0 del triangolo, gli altri lati e vertici saranno ordinati di conseguenza seguendo l’orientazione antioraria.

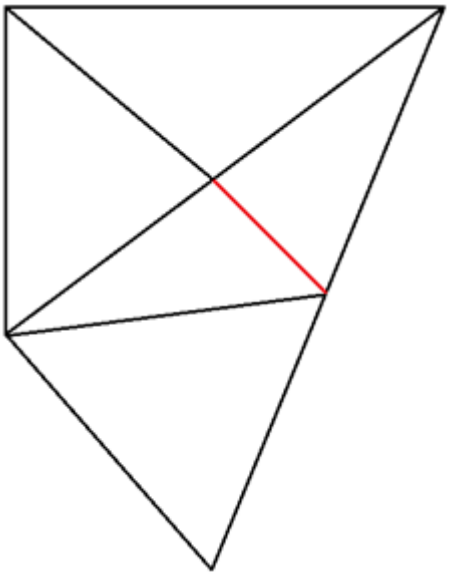
Dopo la rotazione, il lato più lungo viene tagliato generando due lati figli che saranno delimitati rispettivamente dal punto iniziale al punto medio e dal punto medio al punto finale del lato 0. Il triangolo viene invece tagliato tracciando un segmento che unisca il punto medio del lato più lungo con il vertice opposto del triangolo. La creazione di questo segmento permetterà di generare i due triangoli figli, i quali condivideranno il nuovo lato.

Dopo aver tagliato una cella ci si può ritrovare (a meno di celle sul bordo del dominio) con una mesh non uniforme: la cella che condivide il lato 0 della cella tagliata, si troverà ad avere un lato di lunghezza diversa rispetto alle due nuove celle vicine. Di conseguenza anche la cella vicina dovrà essere tagliata.
Poiché il criterio di raffinamento adottato prevede che un triangolo venga tagliato sul suo lato più lungo, potrebbe essere necessario reiterare il processo di taglio sulla cella figlia della cella vicina, anche più volte, affinché si riottenga l’uniformità.



Nell’esempio il lato più lungo della cella vicina non coincide con il lato appena tagliato, quindi essa dovrà prima essere tagliata sul lato più lungo, successivamente si taglierà la cella figlia corrispondente al lato ancora non uniforme. In questo esempio l’uniformità si riottiene già al secondo passo, ma potrebbe essere necessario ripetere l’operazione più volte.





Funzioni e pseudocodici

Per poter raffinare la mesh il nostro gruppo ha sviluppato le seguenti funzioni:

- **Funzione *RefineMesh***
  Raffina la mesh tagliando i triangoli selezionati con le apposite funzioni.

- **Funzione *Rotate***
  Ruota i punti ed i lati del triangolo.

- **Funzione *TriangleRefiner***
  Taglia il triangolo a metà.

- **Funzione *RefineUniformly***
  Taglia il triangolo in 4 triangoli.

- **Funzione *UpdateNeighbourhood***
  Aggiorna la vicinanza per i punti e per le celle.

1. Funzione RefineMesh

La funzione scorre il vettore idCellsToRefine con un ciclo while e, per ogni cella attiva, chiama le funzioni RefineTriangle e UpdateNeighbourhood(). Si tratterà più avanti delle loro funzionalità.

Pseudocodice

```
i = 0
Finchè i non raggiunge il numero di celle:
    Se la cella i-esima <= numero di celle :
        Se la cella è attiva:
            Taglia la cella attraverso la funzione RefineTriangle
Return Success
```

Tratto dal codice del progetto, nel file TriangleRefiner.cpp:

```
Output::ExitCodes TriangleRefiner::RefineMesh(){
    unsigned int i=0;
    while(i<idCellsToRefine.size()){
        if(idCellsToRefine.at(i) <= meshPointer->NumberOfCells()){
          if( (meshPointer->Cell(idCellsToRefine.at(i)))->IsActive() ){
          RefineTriangle(idCellsToRefine.at(i));
          }
        }
        i++;
    }
    idCellsToRefine.clear();
    UpdateNeighbourhood();
    return Output::Success;
}
```

2. Funzione Rotate

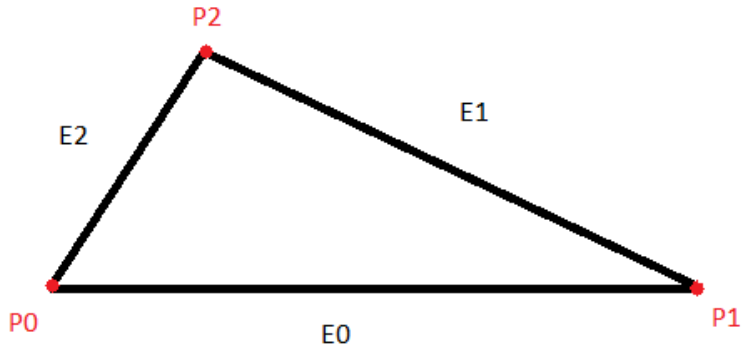
Funzione che ruota i punti ed i lati del triangolo in modo tale che il lato E0 sia il più lungo. I lati successivi vengono successivamente ordinati secondo il verso antiorario.

Prima di tutto, si prendono in esame i 3 lati E0, E1, E2 e i 3 punti P0, P1, P2 della cella *value*-esima e si calcola la loro lunghezza al quadrato (questo perché la funzione "quadrato" è monotona crescente, quindi al fine di confrontare due lati non è necessario farne la radice quadrata, risparmiando in tale modo di costo computazionale). Si confrontano, quindi, i lati per vedere quali di questi è il lato più lungo: se le 3 lunghezze sono uguali, allora si tratta di un triangolo equilatero (l'ordine in tale configurazione non è importante), altrimenti si passa a scambiare i lati, in modo da ottenere E0 come lato più lungo.

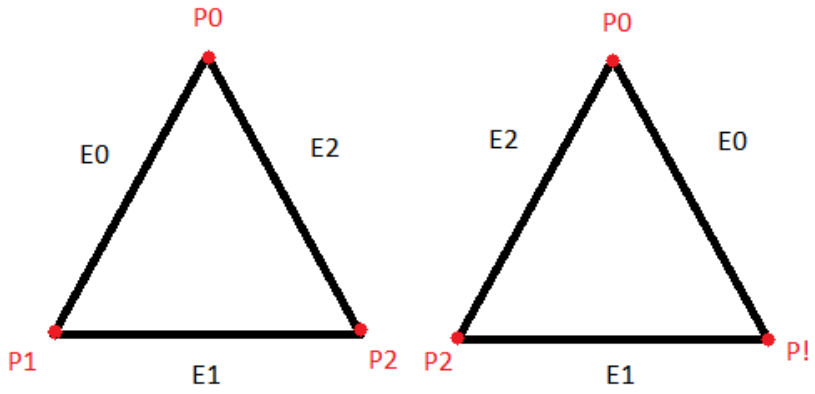
Successivamente, con i lati cambiati si devono aggiustare anche i punti assegnati affinché siano nel giusto ordine:

- P0 deve essere il punto tra i lati E0 ed E2
- P1 deve essere il punto tra i lati E0 ed E1
- P2 deve essere il punto tra i lati E1 ed E2

L'immagine mostra l'ordine assegnato.



Avendo messo i punti nella configurazione desiderata, manca la gestione dell'ordinamento di tutti i triangoli: vi sono infatti 2 possibili rappresentazioni del triangolo secondo i criteri finora utilizzati:



Adesso, è necessario orientare i triangoli in un dato verso, orario o antiorario; si è scelto il verso antiorario in conformità alla richiesta del progetto.

Nell'immagine precedente, il triangolo con l'ordine esatto è quello a sinistra.

Pseudocodice

Definisco puntatori di lati e di punti della cella value-esima *E0, *E1, *E2, *P0, *P1, *P2, *aux

Assegno l'indirizzo della value-esima a cell

Assegno l'indirizzo dei lati e punti ai puntatori

Calcolo del quadrato dei lati

Se il lato 1 è il più lungo

Scambio i lati con ausilio di aux per avere E0 come più lungo

Se invece il lato 2 è il punto lungo

Scambio i lati con ausilio di aux per avere E0 come più lungo

Se il 1° punto del lato E0 coincide con il 1° o il 2° punto del lato E2

Assegno al puntatore P0 l'indirizzo del 1° punto di E0

Se invece è il 2° punto di E0 a coincidere con il 1° o il 2° punto del lato E2

Assegno al puntatore P0 l'indirizzo del 2° punto di E0

Se il 1° punto del lato E1 coincide con il 1° o il 2° punto del lato E0

Assegno al puntatore P1 l'indirizzo del 1° punto di E1

Se invece è il 2° punto di E1 a coincidere con il 1° o il 2° punto del lato E0

Assegno al puntatore P1 l'indirizzo del 2° punto di E1

Se il 1° punto del lato E2 coincide con il 1° o il 2° punto del lato E1

Assegno al puntatore P2 l'indirizzo del 1° punto di E2

Se invece è il 2° punto di E1 a coincidere con il 1° o il 2° punto del lato E1

Assegno al puntatore P2 l'indirizzo del 2° punto di E2

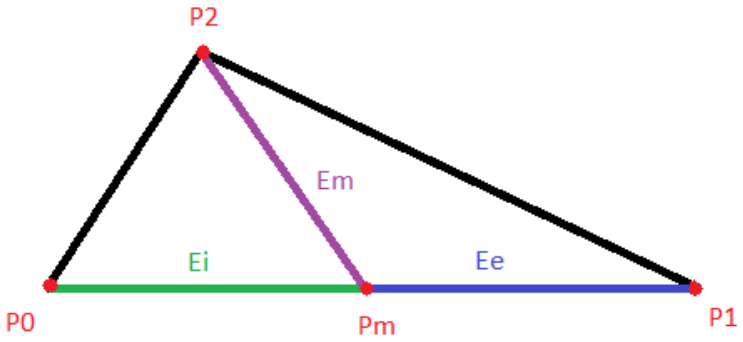
3. Funzione RefineTriangle

È la funzione che esegue il taglio del triangolo in 2 triangoli più piccoli e gestisce la corretta correlazione tra punti e lati che i triangoli condividono.

Innanzitutto, essa richiama la funzione Rotate per l'ordinamento dei lati e dei punti della cella.

Se il lato E0 è attivo, viene tagliato a metà, e al suo posto viene "sostituito" dai lati Ei ed Ee entrambi congiunti tramite il loro punto medio con il nuovo lato Em. Ei avrà come estremi i punti P0 e Pm (punto medio tra P0 e P1) ed Ee avrà come estremi i punti Pm e P1, Em invece sarà il lato che divide il triangolo in 2, con estremi Pm e P2.

L'immagine seguente riassume visivamente quanto spiegato.



Adesso E0 viene disattivato, dato che è già stata tagliata e, come conseguenza, questi ha 2 figli Ei ed Ee. I nuovi lati hanno come padre il lato tagliato E0.

In conformità alla caratteristica del dominio, non è possibile avere un triangolo con un lato formato da 2 lati e quattro vertici, per cui sarà necessario un controllo sul triangolo vicino:

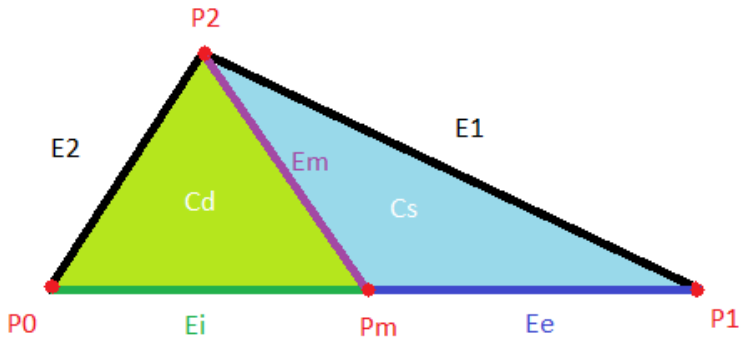
- Se E0 è attivo vuol dire che non era mai stato tagliato, perciò devo aggiungere la fra le celle da tagliare,
- Se invece E0 è inattivo, verifichiamo che anche gli altri lati (E1 ed E2) lo siano, in questo caso richiameremo la funzione RefineUniformly che descriveremo in seguito.

Successivamente, si procede alla gestione dei punti che delimitano il lato Em, il quale ha i punti P2 e Pm. Tuttavia, quest'ultimo punto può presentare un'ambiguità per come lo abbiamo trovato, perciò Pm lo prendiamo come il punto comune ai lati Ei ed Ee.

Passiamo adesso alla gestione delle 2 nuove celle, Cd e Cs, rispettivamente cella di destra e di sinistra.

- La Cd ha come punti P0, Pm, P2 e lati Ei, Em, E2;
- La Cs ha come punti Pm, P1, P2 e lati Ee Ei,Em;

L'immagine seguente riassume visivamente quanto elencato:

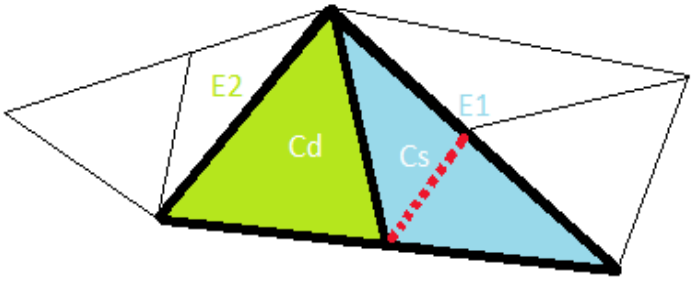


Nota la cella Cd e Cs sono viste dal punto di vista del lato Em.

Aggiorniamo anche le celle dei lati Ei, Ee ad Em:

- Ei ha la cella Cd;
- Ee ha la cella Cs;
- Em ha la cella Cd e Cs;

Non bisogna dimenticarsi di aggiornare le celle dei lati E1 ed E2 del triangolo tagliato, se prima hanno avuto la value-esima cella come una delle 2 celle, grazie al taglio avvenuto, possono avere Cs o Cd come nuovo elemento. Inoltre, per conformità della mesh, in caso il lato E1 risulti inattivo, bisogna aggiungere la cella Cs nella lista di celle da tagliare; l'inattività della cella è dovuta al taglio della cella vicina. Il controllo deve essere fatto anche per il lato E2 della cella Cd.



L'immagine sovrastante è un esempio nella quale il lato E1 ha avuto un taglio dalla cella vicina, per cui la cella Cs deve essere aggiunta alle celle da tagliare; mentre per la cella Cd non è necessario, in quanto il taglio della cella vicina non influisce sull'integrità di Cd.

Pseudocodice

Chiamo funzione Rotate(value)

Definisco *cell, *E0, *E1, *E1, *E2, *Em, *Ei, *Ee, *P0, *P1, *P2, *Pm

Se E0 è attivo

Creo Ei, Ee, Pm

Calcolo le coordinate di Pm

Assegno i punti ai lati

Disattivo il lato E0

Imposto Ei ed Ee come figli di E0

Imposto E0 come padre di Ei ed Ee

Se viene tagliata la cella di dx di E0 ed esiste la cella di sx

Aggiungi la cella di sx da tagliare

Se viene tagliata la cella di sx di E0 ed esiste la cella di dx

Aggiungi la cella di dx da tagliare

Se invece E0 è inattivo

Se E1 ed E2 non sono attivi

Chiamo la funzione RefineUniformly(value)

Termino l'elaborazione su questa cella

Definisco P0C0, P0C1, P1C0, P1C1

Se gli id di P0C0 e P0C1 oppure gli id di P0C0 e P1C1 sono uguali

P0C0 è il punto medio

Se invece gli id di P1C0 e P0C1 oppure gli id di P1C0 e P1C1 sono uguali

P1C0 è il punto medio

Assegno i punti ad Em, sistemo l'ordine dei figli di E0

Creo 2 nuove celle

Disattivo la cella tagliata

Imposto cell come padre di Cd e Cs

Imposto Cd e Cs come figli di cell

Assegno punti a Cd

Assegno punti a Cs

Assegno lati a Cd

Assegno lati a Cs

Assegno le celle ai lati Ei ed Ee

Inizializzo 2 celle di Em ed assegno le celle Cd e Cs

Se E2 ha la cella dx **e** id cella dx E2 è uguale id cell

Cd è la 1° cella di E2

Se E2 ha la cella sx **e** id cella sx E2 è uguale id cell

Cd è la 2° cella di E2

Se E1 ha la cella dx **e** id cella dx E1 è uguale id cell

Cs è la 1° cella di E1

Se E2 ha la cella sx **e** id cella sx E1 è uguale id cell

Cs è la 2° cella di E1

Se almeno uno dei lati di Cs non è attivo

Aggiungo Cs nelle celle da tagliare

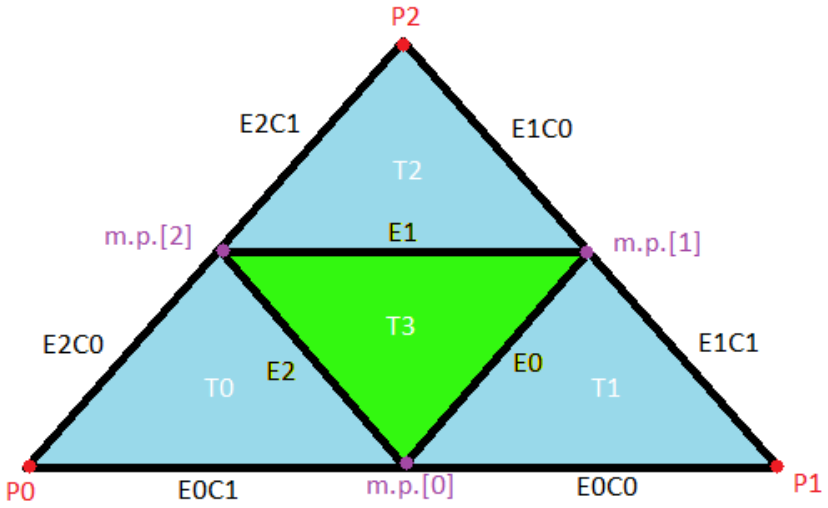
Se almeno uno dei lati di Cd non è attivo

Aggiungo Cd nelle celle da tagliare

4. Funzione RefineUniformly

È la funzione che si occupa dei casi di celle con tutti e tre i lati inattivi, si procede nel tagliare la cella in 4 celle distinte unendo i punti medi dei 3 lati.

Confrontiamo prima i punti dei lati della cella, assegnando così il punto medio. In seguito, si creano i 4 triangoli T0, T1, T2, T3, sostituti alla cella value-esima e si procede ad aggiungerli nella mesh. Infine si disattiva la cella e si gestisce la vicinanza e l'appartenenza dei lati alle celle.



m.p.: mediumpoints

Pseudocodice

Definisco *cell

Definisco vettore di puntatori edges //lati da tagliare

Definisco vettore di puntatori meduimPoints

Definisco lati *E0, *E1, *E2 //i lati "medi"

Definisco punti *C0P0, *C0P1, *C1P0, *C1P1

Definisco *E0C0, *E0C1, *E1C0, *E1C1, *E2C0, *E2C1

Per (i=0; i<3; i++)

Aggiungo ad edges il lato i-esimo

Assegno a C0P0 il 1° punto del 1° figlio dell'i-esimo elemento di edges

Assegno a C0P1 il 2° punto del 1° figlio dell'i-esimo elemento di edges

Assegno a C1P0 il 1° punto del 2° figlio dell'i-esimo elemento di edges

Assegno a C1P1 il 2° punto del 2° figlio dell'ei-esimo elemento di edges

Se C0P0 è uguale C1P0 o C0P0 uguale a C1P1

Aggiungo C0P0 in mediumPoints

Se invece C0P1 è uguale C1P0 o C0P1 è uguale C1P1

Aggiungo C0P1 in mediumPoints

Creo cella T0, lo assegno alla mesh

Assegno a T0 la cella padre cell

Assegno a cell la cella figlia T0

Assegno i punti adeguati a T0

Creo E2 e gli assegno i lati adeguati

Assegno i lati adeguati a T0 (tra questi, uno è E2)

Creo cella T1, lo assegno alla mesh

Assegno a T1 la cella padre cell

Assegno a cell la cella figlia T1

Assegno i punti adeguati a T1

Creo E0 e gli assegno i lati adeguati

Assegno i lati adeguati a T1 (tra questi, uno è E0)

Creo cella T2, lo assegno alla mesh

Assegno a T2 la cella padre cell

Assegno a cell la cella figlia T2

Assegno i punti adeguati a T2

Creo E1 e gli assegno i lati adeguati

Assegno i lati adeguati a T2 (tra questi, uno è E1)

Creo cella T3, lo assegno alla mesh

Assegno a T3 la cella padre cell

Assegno a cell la cella figlia T3

Assegno i punti a T3, i quali sono tutti i punti medi del mediumPoints

Assegno i lati E0, E1, E2 a T3

Imposto inattivo la cella value-esima

Assegno a E0C1 la cella T0

Assegno a E0C0 la cella T1

Assegno a 1OC1 la cella T1

Assegno a E1C0 la cella T2

Assegno a E2C1 la cella T2

Assegno a E2C0 la cella T0

Assegno a E0 le celle T1 T3

Assegno a E1 le celle T3 T2

Assegno a E2 le celle T1 T2

5. Funzione UpdateNeighbourhood

Ai fini della rifinitura della mesh non è necessaria, ma sistema e rende coerente la vicinanza per i punti e per le celle, restituendo alla fine una mesh con dati coerenti e consistenti. Poiché la mesh alla fine sarà costituita solo dalle celle rimaste attive, la funzione opera solo su celle attive.

Pseudocodice

Definisco **lato*, **punto*, **p, *cella*

Definisco i, j, k, h

Definisco valore booleano *trovato*

Per (i=0; i<meshPointer->NumberOfCells(); i++)

Assegno la cella i-esima a *cella*

Se *cella* è attiva

Per (j=0; j<3; j++)

Assegno il lato j-esima di *cella* a *lato*

Assegno il punto j-esima di *cella* a *punto*

Imposto *trovato* come FALSO

Se lato ha la cella di dx E la cella di dx è diversa da *cella*

Per (k=0; k < cella->NumberOfCells(); k++)

Se cella->Cell(k)!=NULL && !cella->Cell(k)->IsActive()

Rimuove la cella k-esima di *cella*

Se cella->Cell(k)!=NULL **e** cella->Cell(k)->Id() == lato->RightCell()->Id()

Imposto *trovato* come VERO

Se !trovato && lato->RightCell()->IsActive()

Aggiungo la cella dx a *cella*

Se invece lato->HasLeftCell() && lato->LeftCell()->Id() != cella->Id()

Per (k=0; k < cella->NumberOfCells(); k++)

Se cella->Cell(k)!=NULL && !cella->Cell(k)->IsActive()

Rimuove la cella k-esima da *cella*

Se cella->Cell(k)!=NULL && cella->Cell(k)->Id() == lato->LeftCell()->Id()

Imposto *trovato* come VERO

Se !trovato && lato->LeftCell()->IsActive()

Aggiungo la cella dx a *cella*

Imposto *trovato* come FALSO

Per (k=0; k<punto->NumberOfCells(); k++)

Se punto->Cell(k)!=NULL && !punto->Cell(k)->IsActive()

Rimuove il punto della cella k-esima

Se punto->Cell(k)!=NULL && punto->Cell(k)->Id() == cella->Id()

Imposto *trovato* come VERO

Se *trovato* è FALSO

Aggiungo al punto la *cella*

h=0

Finchè h<2

Assegno Il punto h-esimo del lato a p

Assegno trovato come FALSO

Per (k=0;k < p->NumberOfEdges(); k++)

Se p->Edge(k)!=NULL && !p->Edge(k)->IsActive()

Rimuovo k-esimo lato di *p*

Se p->Edge(k)->Id() == lato->Id()

Assegno trovato come VERO

Se !trovato && lato->IsActive()

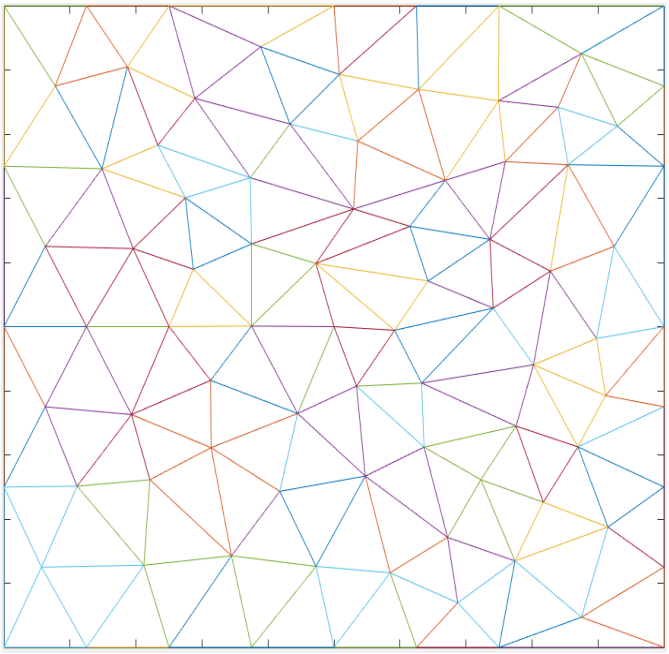
Aggiungo lato a *p*

Incremento h

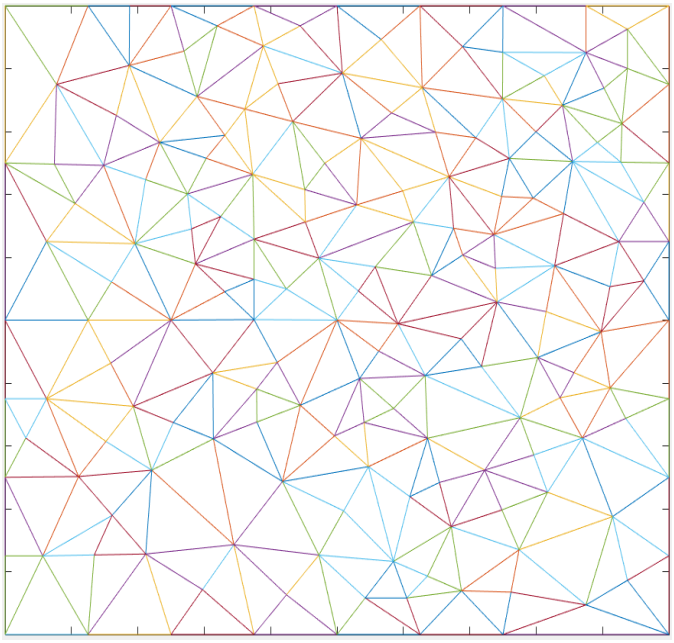
Risultato

Riportiamo infine i risultati:

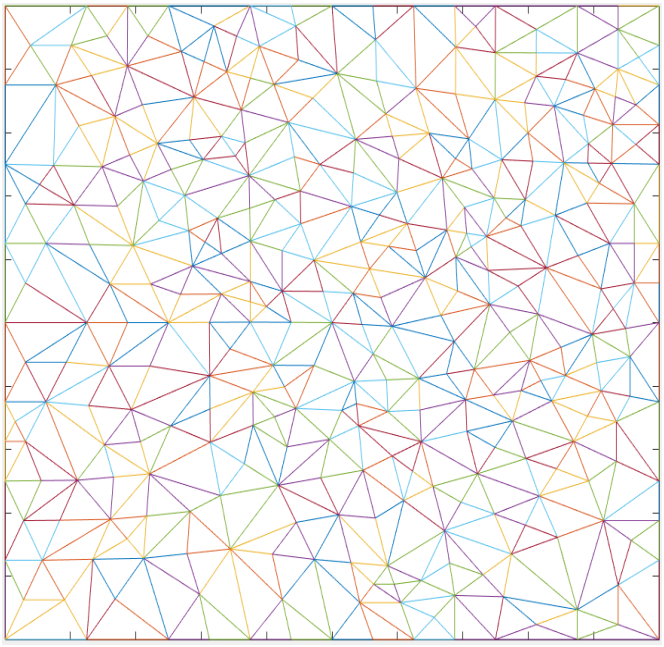
1. Immagine mesh non tagliata:



2. Immagine della mesh sottoposta a un solo ciclo di tagli:



3. Immagine mesh tagliata 2 volte:



4. Immagine della mesh dopo 4 iterazioni di tagli: