RAFFINAMENTO
(METODO COMPLESSO)

Valerio Taralli S282651
Federica Buo S281231

Indice

Si specifica che i seguenti termini saranno sostituiti per l'intera relazione:

- [1] Mesh
- [2] Test
- File con filza

1. Introduzione

Il progetto si concentra su due diverse tipologie di maglia triangolare e procede con il raffinamento di un arbitrario sottoinsieme di triangoli. In conclusione, ogni maglia ottenuta sarà più fine.

1.1. Raffinamento di una maglia triangolare

L'algoritmo è stato implementato utilizzando il metodo "bisezione del lato più lungo". Considerato un triangolo della maglia, consiste nel tracciare il lato più lungo, calcolare il suo punto medio e considerare il vertice opposto ad esso (Punta). Ne consegue l'inserimento di due nuovi sotto triangoli aventi un lato in comune, di cui estremi: punto medio e punta.
Inoltre, la triangolazione deve essere ammissibile, ovvero due triangoli opposti devono avere in comune un intero lato oppure un solo vertice. L'implementazione del metodo "Complesso" è uno dei modi per risolvere il problema: si parte dal lato massimo già ricavato e si prende in considerazione il triangolo opposto ad esso (per cui un suo lato coincide con il lato massimo precedente), dopodiché, si effettua la "bisezione del lato più lungo" e si uniscono il nuovo punto medio con quello prima calcolato. Il risultato includerà tre nuovi sotto triangoli nella maglia di partenza. Questo processo si interrompe per ciascuno dei seguenti casi: il punto medio di un triangolo è sul bordo della maglia, il lato massimo coincide con il lato più lungo del triangolo opposto o il triangolo opposto corrisponde ad uno già raffinato precedentemente.
Oltre al primo triangolo, ogniqualvolta l'algoritmo viene troncato, si sceglie il successivo soggetto da raffinare controllando il vettore delle aree e il teta, arbitrariamente scelto. Il vettore, inizialmente calcolato, è ordinato in senso crescente con l'algoritmo *HeapSort* e successivamente posto in ordine decrescente, mentre il teta è impostato manualmente all'inizio del codice e indica il quantitativo di triangoli da raffinare.

2. Programma

Il programma di raffinamento si suddivide in due parti: *Header files* e *Source files.*
In entrambi, il codice è stato scritto per due diverse tipologie di maglie: la prima con cinque marcatori distinti per lato e punti, la seconda con un solo tipo di marcatore.
Inoltre, per tutto il codice è stato considerato fisso l'orientamento dei triangoli, il cui senso è sempre antiorario.

2.1. Ordinamento

L'algoritmo scelto per l'ordinamento è l'*HeapSort*, uno dei migliori, con una complessità temporale pari a O(n*log(n)) nel caso peggiore.
In questo paragrafo si vuole appunto spiegare il funzionamento di tale algoritmo, progettato manualmente. Prima di tutto è stata definita la funzione «CreaVettoreDescrescente», necessaria a riordinare il vettore crescente, dato dalla funzione *HeapSort*, in un vettore di senso decrescente. Quindi, presi i valori memorizzati nel vettore di ingresso, crea un vettore formato di struttura «Decrescente». Progettazione?
All'interno del *namespace* sono stati anche ridefiniti alcuni operatori, i più importanti per il contesto sono: "<" e " >", la cui inversione consente all'algoritmo di ordinare sempre in modo decrescente.

La funzione «*HeapSort»* ordina quindi un vettore di elementi di tipo «T» per valori crescenti e ritorna tale oggetto.
Inizialmente, posiziona uno ad uno gli elementi del vettore scelto in un grafo seguendo l'ordine degli indici, poi verifica che il nodo parente abbia valore maggiore del nodo appena inserito. Il ciclo *for* serve a tale scopo, e riposiziona i nodi in modo tale che ogni parente abbia etichetta maggiore di ciascun figlio (massimo due).
Il ciclo *while* è utile per ripartire dall'inizio del grafo e vedere quali nodi non hanno soddisfatto la condizione precedente, in modo da ordinarli opportunamente.
In seguito, visualizza i nodi del grafo in un vettore di dimensione "n", il cui primo valore sarà il maggiore tra tutti. Quindi, con un ciclo *for,* scambia il primo elemento con quello di indice massimo ed esclude dal ciclo il vecchio primo componente.
Per soddisfare la condizione di partenza, per cui ogni parente deve essere maggiore di ciascun figlio, presta attenzione a prendere tra i due nodi figli, se entrambi maggiori, il più grande dei due (viene sempre verificata l'esistenza del nodo). Terminato il controllo delle etichette, il ciclo riparte scambiando il nuovo primo elemento con quello di indice maggiore, l'esclusione dell'ultimo avviene ogni volta.

2.2. Classe raffinamento: dichiarazione e definizione

Definito il *namespace* che racchiude tutte le dichiarazioni in un unico posto, sono state introdotte tre strutture: Punti (numero di punti, rispettive coordinate e marcatori), Lati (numero, vertici dei lati e marcatori) e Triangoli (numero, vertici e lati dei triangoli).
In Punti è stato definito un vettore di tipo *Vector2d*, per un uso meno complesso della norma. Per ogni altra struttura, è stato usato un vettore di *array* per contenere tutti gli indici dei rispettivi elementi.
In Triangoli inoltre, sono stati definiti tre importanti vettori, contenenti: lato massimo, il suo marcatore e il suo vertice opposto (Punta).
Nella dichiarazione della classe «MagliaTriangolare» si possono notare due sottocategorie: *private,* in cui gli elementi definiti non sono accessibili all'esterno della classe stessa e *public,* tutti i membri sono accessibili in ogni parte del programma.
In *private* sono state aggiunte le varie strutture (Punti, Lati, Triangoli), essendo le funzioni legate all'oggetto.
In *public*:

- «ImportaMaglia»: oggetto di tipo *bool*; importa la maglia triangolare usando le tre filze prestabilite e memorizza i dati nei rispettivi membri privati. Quindi, verifica che da ogni percorso l'importazione delle informazioni sia avvenuta con successo.

I prossimi tre oggetti di tipo *bool* hanno lo stesso algoritmo per la gestione iniziale di una filza, il quale verrà spiegato di seguito e non ripetuto: si verifica che l'apertura della filza sia avvenuta con successo, dopodiché si inserisce in una lista di stringhe tutte le righe presenti in essa e si elimina il primo elemento (definizione del tipo di dato).

- «ImportaPunti»: importa le proprietà dei punti dei triangoli dalla prima filza «Cell0Ds.csv».
  Dopo il procedimento prima specificato, se il numero dei Punti pari alla lunghezza della lista è non nullo, alloca spazio nel vettore delle coordinate dei Punti per almeno un tale valore.
  Procede poi con la separazione delle informazioni in ogni stringa della lista, avendo per ogni riga: id, marcatore e le due coordinate del punto. Considerato che la variabile «id» è una ripetizione degli indici del vettore, è introdotta solo per scartare l'informazione contenuta in essa. La funzione conclude aggiungendo le altre due variabili rispettivamente alla fine del vettore

dei marcatori e delle coordinate dei Punti. Ritorna vero se non ci sono stati errori lungo il processo di lettura.

- «ImportaLati»: importa le proprietà dei lati dei triangoli dalla filza «Cell1Ds.csv».
  A seguito dell'introduzione, se il numero dei Lati pari alla lunghezza della lista è non nullo, alloca spazio nel vettore dei vertici dei Lati per almeno un tale valore.
  Procede poi con la separazione delle informazioni in ogni stringa della lista, avendo per ogni riga: id, marcatore e gli indici dei vertici dei Lati. La funzione conclude aggiungendo le ultime due variabili rispettivamente alla fine del vettore dei marcatori e dei vertici dei Lati. Ritorna vero se non ci sono stati problemi lungo il processo di lettura.

- «ImportaTriangoli»: importa le proprietà dei triangoli dalla filza «Cell2Ds.csv».
  Dopo la gestione iniziale della filza, se il numero dei Triangoli pari alla lunghezza della lista è non nullo, alloca spazio in entrambi: vettore dei vertici e dei lati dei Triangoli per almeno un tale valore.
  Procede poi con la separazione delle informazioni in ogni stringa della lista, avendo per ogni riga: id, tre indici per i vertici e tre per i lati. In conclusione, vertici e lati vengono aggiunti alla fine dei rispettivi vettori dei Triangoli. Ritorna vero se il processo di lettura è terminato con successo.

Prima di introdurre l'algoritmo del calcolo delle aree, è necessario analizzare il ragionamento dietro ad esso: l'area di un triangolo ha formula nota "(base*altezza) /2"; se si considera come base il lato massimo, l'altezza cadrà perpendicolarmente all'interno dell'triangolo e si potranno quindi sfruttare le seguenti formule: "altezza = ipotenusa*sin(θ)" e "$v_1 \cdot v_2 = \|v_1\|_2 \|v_2\|_2 \cos(\theta)$" per ricavare rispettivamente altezza e suo angolo opposto.

- «CalcolaAreeTriangoli»: restituisce un vettore contenente oggetti di tipo *double*; memorizza in tale vettore le aree dei triangoli da esso calcolate, individuando nel mentre il lato massimo e la punta.
  Inizialmente, con un ciclo *for,* seleziona un triangolo della maglia e salva gli indici dei punti in "vertici". Con un altro ciclo *for* sostituisce le colonne della matrice «PuntiRelativi» con le coordinate di tre vettori: sapendo che un vettore è dato dalla sottrazione delle coordinate di due punti, l'algoritmo seleziona le coordinate del punto con indice precedente e le sottrae al suo successivo (la posizione degli indici avviene sempre in senso antiorario).
  Successivamente, per ogni vettore ottenuto, effettua il calcolo della norma euclidea e trova quindi le lunghezze dei lati del triangolo selezionato. Siccome si analizza un triangolo, controlla che ogni lunghezza sia non nulla.
  Dopo aver posto la base pari alla lunghezza massima, prende il lato precedente ad essa ed effettua il prodotto scalare tra il vettore relativo alla base e il vettore in verso opposto relativo al lato (fissato un piano cartesiano, i vettori partono dall'origine e hanno direzione e verso pari ai lati del triangolo in senso antiorario, quindi, per evitare che tra di essi ci siano degli angoli ottusi, cosa che in un triangolo non accade, si inverte uno dei vettori).
  Inoltre, aggiunge gli indici dei punti estremi alla base, i marcatori dei lati di quel triangolo con quella base e la punta opposta al lato maggiore rispettivamente alla fine dei vettori dei lati massimi, dei marcatori dei lati maggiori e delle punte.
  Infine, calcola l'area del triangolo usando le formule prima elencate e la inserisce alla fine del vettore delle aree. La funzione ritorna tale vettore.

- «EstraiTriangoliDaRaffinare»: restituisce un vettore contenente oggetti di tipo *unsigned int*; estrae gli indici dei triangoli da raffinare secondo un numero arbitrario teta.
  Costruisce il vettore delle aree e lo riordina in senso decrescente con il metodo *HeapSort* precedentemente spiegato.
  In seguito, con un ciclo *for*, considera i primi teta indici del vettore ordinato e seleziona la i-esima area dallo stesso vettore. Ricerca poi, con un ciclo *while*, il suo valore nel vettore delle aree (i cui indici corrispondono a quelli della filza di partenza). Una volta trovata e assicuratosi che l'indice già non compaia in «IndiciDaRaffinare», lo aggiunge alla fine di quest'ultimo vettore, che la funzione restituisce al termine del ciclo *for*.

- «TrovaTriangoloOpposto»: restituisce un oggetto di tipo *unsigned int*.
  Selezionato un triangolo e presi i vertici del suo lato massimo, la funzione ricerca nella maglia triangolare, con un ciclo *for*, il possibile triangolo opposto. Quindi, se l'indice del j-esimo triangolo è diverso da quello prima selezionato e se gli indici dei vertici del lato massimo prima introdotto sono condivisi con quelli di un lato del j-esimo triangolo, la funzione restituisce l'indice del cosiddetto triangolo opposto.

Per introdurre la prossima funzione, è necessario conoscere il ragionamento per il calcolo del punto medio. Siccome si lavora con due vettori che puntano dall'origine agli estremi del lato massimo, questi vanno proiettati sugli assi del piano cartesiano per usare le seguenti formule: "(x1 + x2) / 2" e "(y1 + y2) / 2", con x1 e x2 l'ascissa del primo e del secondo punto del lato massimo (stesso per le ordinate y).

- «Dissezionatore»: tramite l'algoritmo "bisezione del lato più lungo" in forma complessa, raffina una maglia triangolare e ritorna un oggetto contenente punti e triangoli di tale maglia.
  Inizialmente, definisce la nuova maglia raffinata, alloca spazio per almeno tre volte il numero di triangoli nel vettore dei vertici dei triangoli, quindi suppone che da ogni triangolo si ottengano altri tre sotto-triangoli dopo il raffinamento.
  Con il primo ciclo *for,* seleziona ogni triangolo dal vettore «IndiciDaRaffinare». Se questo non è raffinato (quindi è salvato nel vettore buleano con il valore zero) procede a dissezionarlo. Quindi, prende in considerazione tre elementi: indice del triangolo corrente, indici del lato massimo corrente e indice della punta. Dopodiché, calcola il punto medio e lo aggiunge al vettore delle coordinate dei punti, aumenta il numero dei punti e aggiunge il marcatore del lato massimo al vettore dei marcatori dei punti (il punto medio preserva il marcatore del lato).
  In seguito, sapendo che il primo triangolo dissezionato non ha punto medio precedente per permettere la creazione di tre sotto-triangoli, ne crea due nuovi, di vertici: punto medio ricavato, indice di uno dei punti del lato massimo a seconda del sotto-triangolo e indice della punta (opportunamente ordinati in senso antiorario). Procede aumentando il numero di triangoli e assegnando al vettore «statoTriangolo» il valore 1 (il triangolo è memorizzato come dissezionato).
  Con un *do-while*, ricerca l'indice del triangolo opposto («TrovaTriangoloOpposto»), ovvero del successivo da raffinare, e introduce delle strutture *if* per bloccare l'algoritmo nei seguenti casi:
  1) l'indice trovato coincide con il numero di triangoli, quindi il lato massimo del precedente triangolo si affaccia al bordo della maglia triangolare.
  2) il triangolo successivo è già stato dissezionato, quindi cerca nella maglia raffinata l'indice di quel "sotto-triangolo" contenente gli indici dei punti del lato massimo precedente (per semplicità si specifica "sotto-triangolo", tuttavia, siccome è di un triangolo già raffinato, dovrebbe essere considerato come un triangolo effettivo della maglia raffinata). L'algoritmo ricerca anche l'indice del punto rimanente.

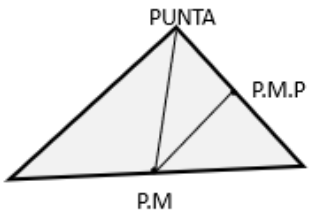
Successivamente, aggiunge al vettore dei vertici dei triangoli: il punto medio precedente, il punto rimanente e l'indice di un estremo del lato massimo precedente a seconda del sotto-triangolo (l'ordine dei dati nel codice mantengono il senso antiorario).
Siccome introduco due nuovi sotto-triangoli in una maglia già raffinata, posso cancellare il "sotto-triangolo" da cui li ho ricavati. Di conseguenza, aumento di un solo valore il numero di triangoli.

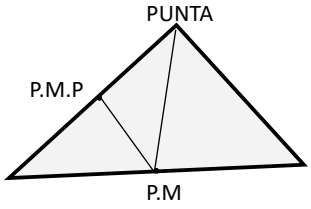
3) il lato massimo precedente coincide con il successivo, tenendo conto dell'inversione degli indici per mantenere il senso antiorario. Quindi, a causa della coincidenza con il punto medio precedente, si formano solo due sotto-triangoli in quello successivo, di cui vertici: punto medio, punta e indice di un estremo del lato massimo (sempre messi in un ordine opportuno). Conclude ponendo 1 a «StatoTriangolo» per considerarlo dissezionato.

Escluse queste possibilità, il triangolo successivo viene smembrato con la funzione «SmembraTriangolo» sotto spiegata. Quindi, calcola il punto medio del triangolo corrente e procede con il ciclo fino ad arrivare ad uno dei blocchi prima elencati.
Usciti dal ciclo *do-while*, la funzione aggiunge tutti i triangoli non dissezionati nella maglia raffinata, ponendo i vertici nel vettore dei vertici dei triangoli e aumentando il numero di triangoli totali.

Per la successiva funzione, si sviluppa il metodo di "bisezione del lato più lungo" nella forma complessa. Perciò si deve tener conto di cinque importanti oggetti: punto medio precedente, lato massimo precedente del triangolo già raffinato e punto medio, lato massimo e punta del triangolo che si sta raffinando. La costruzione dei tre sotto triangoli avviene dall'unione del punto medio corrente sia con la punta sia con il punto medio precedente.

- «SmembraTriangolo»: funzione di tipo *void;* aggiunge gli indici di tre nuovi sotto triangoli al vettore dei vertici dei triangoli.
  Per la costruzione di tali indici, la funzione controlla la posizione del lato massimo precedente, che rispetto al lato massimo corrente, in senso antiorario, può essere prima (i primi indici del lato massimo precedente e del lato massimo corrente coincidono) o dopo (i secondi indici coincidono).
  Di conseguenza, nella costruzione dei triangoli, gli indici varieranno a seconda della posizione del punto medio precedente e saranno presi in senso antiorario. LI LASCIO?
  Dopo: Prima:





- «CostruisciLati»: funzione di tipo *void*; costruisce la struttura Lati e il membro «Triangoli.LatiT».
  Con due cicli *for*, scorre tutti i triangoli e, per ciascuno di essi, studia i lati ricavati dai rispettivi vertici del triangolo scelto. Quindi, per ogni lato memorizza gli indici che lo caratterizzano ma invertiti tra loro, per mantenere il senso antiorario del triangolo opposto (per esempio per il lato di indici {2,0}, il lato stesso del triangolo opposto avrà indici {0,2}).

Successivamente, ricerca all'interno del vettore dei vertici dei lati se è già memorizzata la coppia di indici ottenuta.
Se il lato non è presente, la funzione lo aggiunge nel precedente vettore, ponendo gli indici nel giusto senso antiorario (per l'esempio: {2,0}). Quindi, aumenta il numero dei lati e aggiunge l'indice del lato considerato nel vettore dei lati dei triangoli.
La costruzione dell'oggetto implica anche la ricerca del marcatore associato. Quindi, per ogni indice del lato considerato, la funzione individua i marcatori dei punti, i quali possono non coincidere. In tale situazione, il marcatore aggiunto al vettore dei marcatori dei lati è zero. Nel caso particolare in cui uno degli indici ha un marcatore specifico per i punti agli angoli della maglia triangolare, la funzione prende quello dell'altro indice e lo aggiunge al vettore dei marcatori dei lati.
Se, invece, gli indici del lato sono presenti nel vettore dei vertici dei lati in posizione k, quest'ultimo viene aggiunto agli indici dei lati dei triangoli.

- «EsportaMaglia»: funzione di tipo *bool*; esporta in tre filze tutti i dati precedentemente ottenuti, quindi i punti, i lati e i triangoli della maglia triangolare. Il procedimento è lo stesso per tutti e tre gli oggetti.
  Quindi, crea il nome della relativa filza, verifica che la sua apertura sia avvenuta con successo e inserisce, oltre ai metadati che identificano le varie colonne, tutti i dati ricavati con le funzioni prima spiegate. Ritorna vero se la scrittura delle filze è avvenuta con successo.

- «CopiaDatiMaglia»: funzione di tipo *void*; copia i dati della maglia triangolare senza apporre modifiche, in modo da renderli accessibili al di fuori della classe.
  Quindi, introduce tre strutture su cui copia rispettivamente: i punti, i lati e i triangoli della maglia triangolare.

2.3. *Main*

In questo paragrafo saranno elencate in ordine d'uso alcune funzioni del precedente paragrafo.
Si ricorda che le maglie triangolari da raffinare sono due, quindi nel codice sono ripetuti tutti i passaggi per la seconda maglia.
Come primo passo, si importa la maglia triangolare («ImportaMaglia») e si verifica il successo dell'importazione. Dopo aver estratto i triangoli da raffinare di un teta impostato manualmente («EstraiTriangoliDaRaffinare»), si raffina la maglia triangolare («Dissezionatore») e si costruiscono tutti i lati ricavati dal raffinamento («CostruisciLati»).
Si conclude esportando la maglia in tre nuove filze («EsportaMaglia») e verificandone il successo.

3. Teste

In totale sono cinque (esclusi quelli ripetuti per la seconda maglia triangolare): due testano il funzionamento dell'ordinamento, gli altri tre, invece, l'importazione, il raffinamento e l'esportazione.

3.1. Verifica del codice

1) Considerato un vettore di riferimento già ordinato, lo si confronta con il vettore ricavato dalla funzione «*HeapSort*».
2) Considerato un vettore di riferimento ordinato in senso decrescente, lo si confronta con il vettore ricavato dalla funzione «*HeapSort*», in ordine decrescente.
3) Si verifica il successo dell'importazione e, per ogni triangolo, si controlla che gli indici dei punti estremi ai lati, presi dalla filza «Cell1Ds.csv», siano presenti all'interno di «Cell2Ds.csv», ovvero tra gli indici dei vertici dei triangoli.
4) Dopo aver controllato che l'importazione sia avvenuta con successo, si estraggono i triangoli da raffinare, si dissezionano e si costruiscono i lati. Tutti i dati vengono copiati con «CopiaDatiMaglia». La verifica è la stessa del precedente teste, infatti per ogni triangolo, si prendono gli indici dei due punti che delimitano i lati e si ricercano in indici dei vertici dei triangoli. Applicando il tutto alla maglia triangolare raffinata, il teste cede errore se quegli indici corrispondono alla fine della filza, ovvero se non sono stati trovati al suo interno.
5) Se l'importazione è avvenuta con successo, si estraggono i triangoli da raffinare, si dissezionano, si costruiscono i lati e si esporta la maglia. Infine, si controlla che l'esportazione sia avvenuta con successo, altrimenti «*ASSERT_TRUE*» riporta errore.