POLITECNICO DI TORINO

Corso di Laurea in Matematica per l'Ingegneria

Politecnico
di Torino

Relazione di Programmazione e Calcolo Scientifico

Professori	Studenti
Prof. Stefano BERRONE	Martina SALDUTTI 293376
Prof. Matteo CICCUTTIN	Martina TOSCANO 294523
Prof.ssa Gioana TEORA	
Prof. Fabio VICINI	

A.A. 23/24

Indice

Capitolo 1

Determinare le tracce di un DFN

L'obiettivo della prima parte consiste nell'identificare le tracce di un DFN: per ciascuna frattura, bisogna differenziare le tracce in passanti e non-passanti e riordinarle per lunghezza in ordine decrescente.

1.1 Strutture dati

All'interno di un apposito namespace, `Data`, sono state definite due strutture dati che consentissero di memorizzare le informazioni necessarie a lavorare con le fratture e le tracce:

- Data::**Fract**, contenente
  - *FractId*, un `unsigend int` che memorizza l'Id della frattura;
  - *vertices*, un `Eigen::MatrixXd` contenente, in ogni colonna, le coordinate dei vertici della frattura;
  - *normals*, un `Eigen::Vector3d` che salva le coordinate della vettore normale alla frattura;
  - *d*, un `double` contenente il valore del termine noto necessario per definire l'equazione del piano; tale termine coincide con un punto appartenente al piano stesso;
  - *passingTracesId*, un `std::vector` di `unsigned int` contenente gli Id di tutte le tracce passanti per la frattura;
  - *notPassingTracesId*, un `std::vector` di `unsigned int` contenente gli Id di tutte le tracce non passanti per la frattura.

- Data::**Trace**, contenente
  - *TraceId*, un `unsigend int` che memorizza l'Id della traccia;
  - *FractureIds*, un `std::array` formato da due `unsigned int`, rispettivamente gli Id delle due fratture che generano la traccia;
  - *ExtremesCoord*, un `std::array` formato da due `Eigen::Vector3d`, corrispondenti alle coordinate dei vertici della traccia;
  - *Tips*, un `std::array` formato da due `bool`, cioè due valori booleani che valgono rispettivamente 0 se la traccia è passante per la frattura e 1 se è non passante;
  - *length*, un `double` che memorizza la lunghezza della traccia.

1.2 Codice, algoritmi e funzioni

Si descrive, nel seguito, l'algoritmo utilizzato.
Vengono importati i dati forniti dai file *FR{N}_data.txt* attraverso la funzione `Data::ImportData`, dopodiché si itera su ogni coppia di fratture per determinare le tracce generate da queste attraverso la funzione `FractureOperations::findTraces`.
Prima di chiamare quest'ultima funzione, tuttavia, al fine di migliorare il costo computazionale, si evitano i controlli su alcuni casi, cioè:

- caso in cui la distanza reciproca tra due poligoni sia grande al punto da non permettere alle due fratture di intersecarsi (si usa la funzione `FractureOperations::fracDistance`);
- caso in cui le due fratture siano giacenti su piani paralleli, quì fa eccezione un caso particolare di "book case" (trattato a parte), in cui le due fratture giacciono sullo stesso piano e si intersecano solo lungo un lato, senza generare sottopoligoni.

A questo punto vengono salvate le informazioni delle tracce, divise in passanti e non, riordinate attraverso l'algoritmo di `mergeSort`, riorganizzato al fine di tener conto della lunghezza delle tracce, e generati gli output richiesti attraverso le funzioni `Data::ExportFirstFile` e `Data::ExportSecondFile`.

1.2.1 Principali funzioni in `FractureOperations`

Sono di seguito descritte le principali funzioni utilizzate per l'identificazione delle tracce:

- **`findTraces`** Prende in input le referenze di due `Data::Fract`, cioè la coppia di fratture considerate, `t`, la direzione dell'intersezione tra le due fratture e un `Data::Trace` a cui verranno aggiunte le informazioni della traccia trovata. Viene definito, successivamente risolto, il sistema per identificare uno dei punti per cui passa la retta su cui giace la traccia, in questo modo, per ogni frattura, viene chiamata la funzione *findPosition*, che identifica tutti i punti in cui la retta interseca i lati e li salva nel vettore `CandidatePoints`. Attraverso la funzione `isPointInPolygon` si controlla quali tra i candidati sono punti interni ad entrambe le fratture e si salvano questi punti in un `std::vector` chiamato *potentialPoints*, che risulterà vuoto se le fratture hanno superato il controllo della distanza (*fracDistance*) ma comunque non si intersecano. Si fa un ulteriore controllo per verificare che il vector dei punti potenziali sia effettivamente non vuoto e che i punti siano distinti e si salvano le loro coordinate in `extremePoints`. Si calcola la lunghezza della traccia, si verifica se la traccia è passante o meno per le fratture attraverso la funzione `isTracePassing`, e si salvano tutte le informazioni ottenute in `foundTrace`.

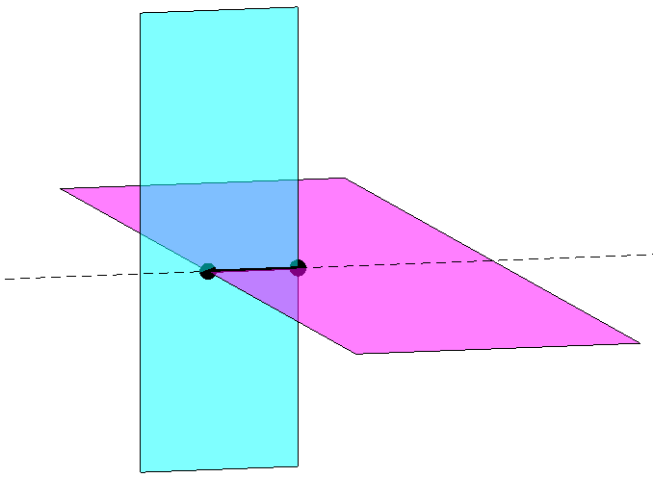

Figura 1.1: Traccia generata da due fratture

- **`findPosition`** Identifica i punti di intersezione tra la retta, definita precedentemente e i lati della frattura. Viene creata una variabile booleana `previous` che memorizza il segno del prodotto scalare tra `t` e il vettore da `P` al primo vertice di `Fracture`. Per ogni vertice del poligono, la funzione calcola una variabile booleana `current` che rappresenta il segno del prodotto scalare tra `t` e il vettore da `P` al vertice corrente. Se `current` è diverso da `previous`, allora i due vertici sono da due lati diversi rispetto alla retta e si cerca un

punto di intersezione tra la retta e il segmento di poligono corrispondente utilizzando la funzione `findExtreme`. Inoltre, la funzione verifica se la retta coincide con un lato del poligono e calcola gli ulteriori punti di intersezione che potevano essere scartati dal controllo di `previous` e current. Questi controlli possono generare più punti candidati di quelli previsti. In definitiva, i punti di intersezione validi vengono aggiunti al vettore `CandidatePoints`.

- **`findExtreme`** Calcola il punto di intersezione tra la retta e un lato della frattura. Viene costruita la matrice `A` con la prima colonna uguale a `t` e la seconda uguale alla direzione della retta sui cui giace il lato del poligono. Successivamente, viene creato un vettore `b` come differenza tra un vertice del lato e il punto `P` della retta parametrica. La funzione risolve il sistema lineare `A * paramVert = b` per ottenere i parametri di intersezione `paramVert`. Questi parametri vengono utilizzati per calcolare due punti candidati per l'intersezione: `Candidate1`, situato lungo il lato e `Candidate2`, situato lungo la retta definita da `t` e `P`. Se la differenza tra `Candidate1` e `Candidate2` è minore della tolleranza, la funzione considera valida l'intersezione, assegna `Candidate1` a `intersection` e restituisce `true`. Altrimenti, la funzione restituisce `false`, indicando che non c'è intersezione valida.

- **`isPointInPolygon`** Verifica se un punto `point` si trova all'interno di un poligono definito dai suoi vertici nella matrice `Polygon`, considerando la normale al piano del poligono, `normal`. La funzione itera su ciascun lato del poligono. Per ciascun lato, calcola il vettore `vector1` che rappresenta il lato del poligono e il vettore `vector2` che congiunge il vertice corrente con il punto da testare. Calcola quindi il prodotto vettoriale di questi due vettori, `CrossProduct`. Se il prodotto scalare tra `CrossProduct` e la normale `normal` è negativo il punto si trova al di fuori del poligono e la funzione restituisce `false`. Se il ciclo termina senza trovare un prodotto scalare negativo, la funzione restituisce `true`.

- **`isTracePassing`** Verifica se la traccia è passante per il poligono. La funzione inizializza due variabili booleane, `startOnEdge` ed `endOnEdge`, impostate inizialmente a `false`. Itera quindi su ciascun lato del poligono, definito dai vertici successivi e verifica se `traceStart` e `traceEnd` si trovano su uno dei lati del poligono utilizzando la funzione `isPointOnEdge`. Se `traceStart` si trova su un lato, imposta `startOnEdge` a `true`; se `traceEnd` si trova su un lato, imposta `endOnEdge` a `true`. La funzione restituisce `true` solo se entrambe le variabili `startOnEdge` ed `endOnEdge` sono `true`, indicando che sia il punto iniziale che il punto finale della traccia si trovano sui lati del poligono, altrimenti restituisce `false`.

- **`isPointOnEdge`** determina se un punto `point` si trova su un lato. La funzione calcola il vettore `edge` come differenza tra `V2` e `V1`, e il vettore `edgeToPoint` come differenza tra `point` e `V1`. Viene calcolata la lunghezza proiettata (`projectedLength`) del vettore `edgeToPoint` sul vettore `edge` come il rapporto tra il prodotto scalare di `edge` e `edgeToPoint` e la lunghezza del segmento `edge`. La funzione restituisce `true` se `projectedLength` è compreso tra 0 e `edgeLength` e se la norma del prodotto vettoriale tra `edgeToPoint` e `edge` è minore della tollernza, indicando che il punto `point` si trova sul segmento; altrimenti restituisce `false`.

1.2.2 Funzioni di `SortLibrary`

La funzione `merge` unisce due sottovettori ordinati di identificativi di tracce (`vecIdTraces`) in un unico vettore ordinato. Prende in ingresso il vettore degli identificativi, il vettore delle tracce (`traces`), e tre indici (`left`, `center`, `right`) che definiscono i confini dei due sottovettori da unire. La funzione confronta le lunghezze delle tracce corrispondenti agli identificativi nei due sottovettori e riempie temporaneamente un nuovo vettore (`tmp`) con gli identificativi in ordine decrescente di lunghezza. Una volta uniti i due sottovettori, il contenuto di `tmp` viene copiato nel vettore originale `vecIdTraces`.

La funzione `mergesort` implementa l'algoritmo di ordinamento merge sort su `vecIdTraces`. Prende in ingresso il vettore degli identificativi, il vettore delle tracce (`traces`), e due indici (`left` e `right`) che definiscono i confini del sottovettore da ordinare. Se `left` è minore di `right`, la funzione calcola il punto centrale (`center`) e richiama ricorsivamente se stessa per ordinare le due metà del sottovettore. Una volta ordinate le due metà, la funzione `merge` viene chiamata per unire i due sottovettori ordinati in un unico sottovettore ordinato.

La funzione `Mergesort` è l'interfaccia principale per l'algoritmo di ordinamento merge sort. Prende in ingresso il vettore degli identificativi (`data`) e il vettore delle tracce (`traces`). Se `data` non è vuoto, la funzione richiama `mergesort` passando `data`, `traces`, l'indice iniziale e l'indice finale per avviare il processo di ordinamento. Questa funzione gestisce l'inizio dell'ordinamento e garantisce che l'intero vettore venga ordinato utilizzando l'algoritmo merge sort.

Capitolo 2

Determinare i sotto-poligoni generati per ogni frattura

L'obiettivo della seconda parte è quello di determinare, per ciascuna frattura, i sotto-poligoni generati dal prolungamento e dal taglio delle tracce presenti su quella stessa frattura

2.1 Strutture dati

Innanzitutto, si definisce una nuova struttura per memorizzare i dati della mesh: `PolygonalMeshLibrary::PolygonalMesh`, contenente rispettivamente:

- per le celle 0D:
  - `Num0DsCell`: un intero che rappresenta il numero di celle 0D.
  - `coord0DsCellMap`: una mappa non ordinata che ha come chiave un intero, corrispondente all'Id della cella 0D, e come valore un `Eigen::Vector3d`, rappresentante le coordinate.
  - `Vertices_list`: la lista dei vertici individuati, con eventuali ripetizioni.
- per le celle 1D:
  - `Num1DsCell`: un intero che rappresenta il numero di celle 1D.
  - `edges_list`: una lista di array di due `Eigen::Vector3d`, rappresentante le coordinate degli estremi delle celle 1D.
  - `Cell1DMap`: una mappa non ordinata che rappresenta le connessioni delle celle 1D.
- per le celle 2D:

- `Num2DsCell`: un intero che rappresenta il numero di celle 2D.
- `subpolygons_list`: una lista di matrici `Eigen::MatrixXd` che rappresentano i vertici dei sottopoligoni, ordinati in senso decrescente.
- `Cell2DsVertices`: una mappa non ordinata che ha come chiave un intero, corrispondente all'Id della cella 2D, e come valore i vertici delle celle.
- `Cell2DsEdges`: una mappa non ordinata che ha come chiave la stessa della precedente e come valore gli spigoli della cella.

La scelta dell'`unordered_map` è dovuta ai tempi di accesso mediamente costanti, $O(1)$, per le operazioni. Questo è significativamente più veloce rispetto ai tempi di accesso $O(\log n)$ di una `map`. Inoltre, non essendo necessario mantenere l'ordine degli elementi, `unordered_map` evita il costo aggiuntivo associato alla gestione e al mantenimento della struttura ordinata, risultando in una performance migliore.

2.2 Codice, algoritmi e funzioni

Si descrive, nel seguito, l'algoritmo utilizzato.
Vengono copiate tutte le tracce in un vettore chiamato `TracesCopy`, al fine di modificare le copie e non le tracce originali, e vengono inseriti gli ID delle tracce in una lista chiamata `AllTraces`. Una coda `AllSubPolygons` viene inizializzata con la frattura corrente e poi viene chiamata la funzione `PolygonalMeshLibrary::MakeCuts` per eseguire i tagli necessari sui sottopoligoni generati dalle tracce. Dopo aver eseguito i tagli, la funzione `PolygonalMeshLibrary::CreateMesh` viene chiamata per creare la mesh poligonale. Infine, vengono aggiornati i contatori delle celle 0D, 1D e 2D della mesh e la mesh viene aggiunta al vettore `Meshes`.

2.2.1 Descrizione di `PolygonalMeshLibrary::MakeCuts`

La funzione `MakeCuts` è progettata per suddividere un poligono in più sottopoligoni. Prende in input la lista di Id delle tracce `AllTraces`, un vettore di `Data::Trace`, una mesh poligonale `PolygonMesh` e una coda di sottopoligoni `AllSubPolygons`. La funzione opera ricorsivamente come segue:

- Se la lista `AllTraces` è vuota, salva tutti i sottopoligoni presenti nella coda `AllSubPolygons` nella mesh `PolygonMesh` e ritorna `true`. Se la coda `AllSubPolygons` è vuota, la funzione ritorna `false`. Se entrambe le liste contengono elementi, la funzione seleziona il sottopoligono corrente `CurrentPolygon` dalla coda `AllSubPolygons`.
- La funzione cerca la traccia più lunga interna al `CurrentPolygon`. Per ogni traccia, verifica se i suoi estremi si trovano all'interno del poligono corrente utilizzando la funzione `FractureOperations::isPointInPolygon`. Se la traccia

è interna, la funzione verifica ulteriormente la posizione della traccia rispetto ai bordi del poligono con la funzione `checking`. Se non ci sono tracce interne, la funzione salva il sottopoligono corrente nella mesh `PolygonMesh`, rimuove il sottopoligono dalla coda e richiama `MakeCuts` ricorsivamente. Se viene trovata una traccia interna, la funzione determina i punti di intersezione tra la traccia e i bordi del poligono. Divide il poligono in due nuovi sottopoligoni lungo la traccia (Figura 2.1).

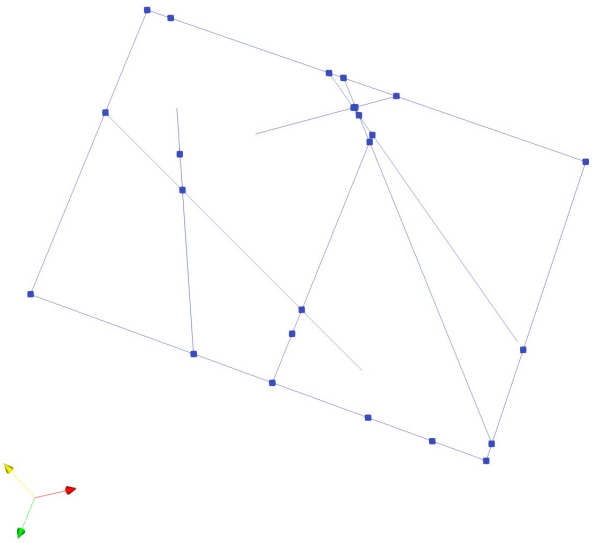

Figura 2.1: Punti di intersezione nella prima frattura di DFN10

- I nuovi sottopoligoni vengono salvati e aggiunti alla coda `AllSubPolygons` se hanno un'area valida, calcolata con la funzione `PolygonalMeshLibrary::CalculateArea`. Poi, aggiorna le estremità della traccia corrente o la rimuove dalla lista `AllTraces` se è completamente interna al poligono.
- Dopo aver aggiornato la lista di tracce e la coda di sottopoligoni, la funzione richiama ricorsivamente `MakeCuts` per continuare la suddivisione.

La funzione ritorna `true` per indicare il completamento della procedura di suddivisione (Figura 2.2).

2.2.2 Descrizione di `PolygonalMeshLibrary::CreateMesh`

La funzione `CreateMesh` è responsabile della costruzione della mesh poligonale 2D a partire dalla struttura dati `PolygonMesh`. Questa funzione si occupa di mappare

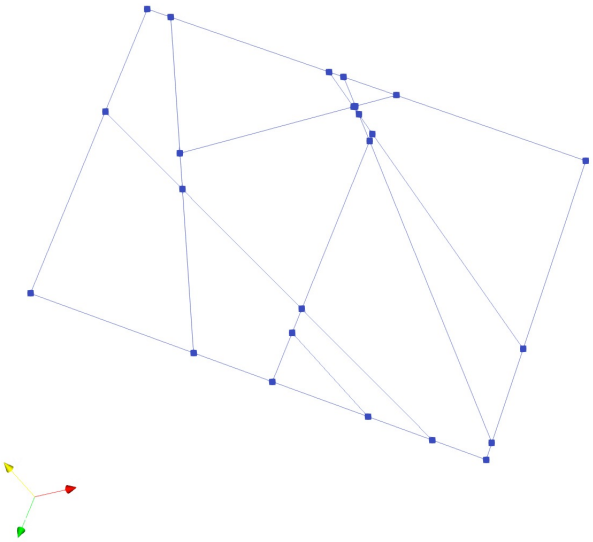

Figura 2.2: Divisione nei sottopoligoni

vertici, lati e facce del poligono con identificatori univoci, aggiornando la struttura dati del poligono con queste informazioni. La funzione opera nel modo seguente:

- Itera attraverso la lista dei vertici `Vertices_list` del poligono. Per ogni vertice, crea una mappatura tra un identificatore univoco `IdCell0d` e le coordinate del vertice, inserendola in `coord0DsCellMap`. Rimuove quindi il vertice dalla lista.

- Itera attraverso la lista i lati `edges_list` del poligono. Per ogni lato, trova i vertici estremi nella mappa `coord0DsCellMap`, recuperando i rispettivi identificatori. Crea una mappatura tra un identificatore univoco `IdCell1d` e un array contenente gli identificatori degli estremi del bordo, inserendola in `Cell1DMap`. Rimuove quindi il bordo dalla lista.

- Itera attraverso la lista dei sottopoligoni `subpolygons_list`. Per ogni sottopoligono, che è rappresentato da una matrice di coordinate, crea due vettori: `Cell2DsVertices_vector` per gli identificatori dei vertici e `Cell2DsEdges_vector` per gli identificatori dei bordi. Per ogni vertice del sottopoligono, trova il rispettivo identificatore nella mappa `coord0DsCellMap` e lo aggiunge a `Cell2DsVertices_vector`.

- Per ogni coppia di vertici consecutivi nel sottopoligono, cerca il lato corrispondente nella mappa `Cell1DMap` e aggiunge il suo identificatore a `Cell2DsEdges_vector`. Se il lato non esiste, cerca il lato con orientamento inverso.

- Inserisce le mappature dei vertici e dei lati del sottopoligono nelle rispettive strutture dati `Cell2DsVertices` e `Cell2DsEdges`, utilizzando un identificatore univoco `IdCell2d`.

La funzione costruisce così una rappresentazione completa della mesh poligonale, mappando ogni vertice, bordo e faccia su identificatori univoci che consentono di gestire efficacemente la geometria del poligono.

2.2.3 Altre funzioni

`PolygonalMeshLibrary::checking`

Questa funzione verifica se gli estremi di una traccia coincidono con i vertici di un poligono. Se entrambi gli estremi si trovano su uno stesso bordo del poligono, la funzione imposta `TraceOnEdge` a `true` e restituisce `true`. Se un estremo è interno al lato, aggiorna la sua posizione per allinearsi meglio con i vertici del poligono.

`PolygonalMeshLibrary::IsTraceInSubpolygon`

Questa funzione controlla se una traccia attraversa un sottopoligono. Utilizza il prodotto vettoriale per determinare la posizione relativa della traccia rispetto ai lati del poligono. Se la traccia interseca un lato del poligono, restituisce `true`.

`PolygonalMeshLibrary::UpdateTrace`

Questa funzione aggiorna una traccia suddividendola in due nuove tracce che connettono gli estremi originali agli estremi più vicini nel std::vector `estremiTracce`. Le nuove tracce vengono aggiunte in `AllTraces` e al vettore delle tracce `traces`.

`PolygonalMeshLibrary::CalculateArea`

Questa funzione calcola l'area di un poligono usando il metodo del determinante per i triangoli formati da un punto di riferimento e ogni coppia di vertici consecutivi. Restituisce `true` se l'area calcolata è maggiore della tolleranza tol^2.

`PolygonalMeshLibrary::SavingSubpolygon`

Questa funzione salva i vertici e i bordi di un poligono in `PolygonMesh`. I vertici vengono aggiunti alla lista `Vertices_list` e i lati alla lista `edges_list`. Il poligono viene aggiunto a `subpolygons_list`.