Report Progetto PCS 2024

s295461, Stefano Foglietti
s283463, Davide Provenzano
s284671, Sara Ilaria Maria Totino

Parte 1: Determinare le tracce di un DFN

Strutture utilizzate

Sono state definite due strutture all'interno delle quali memorizzare i dati richiesti:

DiscreteFractureNetwork

Questa struttura contiene le informazioni riguardanti le fratture contenute all'interno dei file forniti inizialmente:

- `unsigned int numFracture`, il numero di fratture contenute nel file;
- `vector<unsigned int> fractureID`, un vettore composto dai loro ID identificativi;
- `vector<unsigned int> NumVertices`, il numero di vertici noti per ogni frattura;
- `vector<MatrixXd> vertices`, le coordinate per ogni vertice.

Traces

Memorizza le caratteristiche relative alle tracce ottenute:

- `unsigned int numTraces`, il numero di tracce risultanti;
- `vector<unsigned int> traceId`, l'ID identificativo assegnato a ciascuna traccia;
- `vector<Vector2i> fractureId`, gli ID delle fratture che generano ogni traccia;

- `vector<MatrixXd> coordinates`, le coordinate di ogni traccia;
- `vector<double> length`, la lunghezza di ciascuna;
- `vector<vector<tuple<unsigned int, bool, double>>> traceReordered`, le tracce ordinate per lunghezza (decrescente), differenziate tra passanti e non-passanti.

Tolleranze

Per implementare il codice è necessario ricordare che va introdotta una tolleranza assoluta. In base al confronto da analizzare è stata poi calcolata una tolleranza relativa, definita, ad esempio per verificare che due punti coincidano:

```
(intersection1 - verify1).norm() < tol*max(intersection1.norm(), verify1.norm())
```

In generale per definire la tolleranza relativa $\mathcal{T}$

$$a == b \rightarrow \frac{|a-b|}{\max\left(|a|, |b|\right)} < \mathcal{T} \tag{1}$$

Identificare le tracce

La funzione `ImportFracture` è progettata per eseguire una serie di operazioni su un file contenente i dati delle fratture. La funzione importa i dati, calcola le intersezioni delle fratture, stampa i risultati su un file, riordina le tracce delle fratture e stampa nuovamente i risultati riordinati. Al suo interno vengono chiamate altre 5 funzioni: `ReadFracture`, `FractureIntersection`, `PrintOnFile`, `TraceReorder` e `PrintTraces`.

Prima di salvare i dati all'interno delle strutture sopra definite, le funzioni `clearDiscreteFractureNetwork` e `clearTraces` svuotano tutti gli elementi rispettivamente all'interno di `DiscreteFractureNetwork` e di `Traces`.

ReadFracture

La funzione `ReadFracture` legge un file contenente i dati delle fratture da trattare e li salva all'interno della struttura `DiscreteFractureNetwork`. Si verifica che il file si apra correttamente e che non sia vuoto. Nel `main` vengono definiti i percorsi dei file di input per le fratture (FR3, FR10, FR50, FR82, FR200, FR362). Vengono eliminate le righe non necessarie e vengono memorizzati l'ID della frattura, il numero di vertici e le loro coordinate. È necessario garantire che la struttura `DiscreteFractureNetwork` disponga di

spazio sufficiente per memorizzare tutte le informazioni che verranno lette dal file di input: non conoscendo a priori la dimensione viene inizializzata in base al numero di fratture fornite. La riserva anticipata di memoria riduce il numero di riallocazioni necessarie durante l'inserimento dei dati, migliorando l'efficienza complessiva. Le coordinate delle fratture si salvano all'interno di matrici che a loro volta vengono inserite in un vettore, per facilitarne l'accesso.

FractureIntersection

La funzione `FractureIntersection` si occupa di trovare le intersezioni tra i piani contenenti le fratture salvate.

Per evitare ricerche di intersezioni tra fratture troppo distanti tra loro è stata inserita la funzione `BBox3D`. Al suo interno è applicato l'algoritmo della Bounding Box: è un metodo di pre-elaborazione utilizzato per ridurre la complessità del calcolo delle intersezioni tra poligoni. Si crea una bounding box tridimensionale attorno a una frattura ossia un parallelepipedo rettangolo che la contiene completamente e delimita lo spazio che occupa. Vengono create due variabili vettoriali per memorizzare le coordinate minime e massime inizializzate a valori estremamente grandi e piccoli, rispettivamente. Successivamente, la funzione itera attraverso tutte le colonne della matrice `vertices`. Per ogni colonna (che rappresenta un vertice), la funzione aggiorna:

- `bbox[0]` usando `cwiseMin()`, che confronta elemento per elemento e mantiene il minimo tra i valori attuali in `bbox[0]` e le coordinate del vertice corrente.
- `bbox[1]` usando `cwiseMax()`, che confronta elemento per elemento e mantiene il massimo tra i valori attuali in `bbox[1]` e le coordinate del vertice corrente.

Per definire le intersezioni tra fratture viene creato un vettore di `tuple (BBoxVect)` per memorizzare gli ID delle fratture e i loro Bounding Boxes (`BBoxes`), ognuno rappresentato come un `Vector3d`. Per ogni frattura si cicla su tutte le fratture successive e si verifica se i loro bounding boxes si intersecano. Se viene trovata l'intersezione, vengono selezionate le due fratture (`Id1` e `Id2`) e per ognuna si calcolano i vettori `u` e `v`, cioè i vettori definiti lungo i lati della frattura che collegano rispettivamente il terzo con il primo vertice e il terzo con il secondo vertice. Si calcolano poi le normali `n` ai piani delle due fratture:

$$\frac{\vec{u} \times \vec{v}}{\|\vec{u}\|_2 \|\vec{v}\|_2} \tag{2}$$

Quindi è possibile ricavare la direzione della retta `s` di intersezione come prodotto vettoriale tra le due normali ai piani: le due normali e la direzione

della retta di intersezione vengono poi salvati all'interno della matrice `coeff`. Il termine noto `d` è calcolato come il prodotto scalare della normale `n` con un punto sul piano corrispondente (nel nostro caso viene utilizzato il terzo punto della frattura come riferimento): i due termini noti e il valore 0 (associato alla retta di intersezione) vengono salvati all'interno del vettore `term`. Per ottenere l'intersezione è infine necessario determinare l'intersezione tra i due piani delle fratture e il piano ortogonale definito da `s`. Si può quindi costruire il sistema lineare la cui soluzione è il punto di intersezione `point`:

$$\begin{cases} \vec{n}_1^\top \cdot \vec{\text{point}} = d_1 \\ \vec{n}_2 \cdot \vec{\text{point}} = d_2 \\ \vec{s}^\top \cdot \vec{\text{point}} = 0 \end{cases} \tag{3}$$

FindTraces

A questo punto, si passa a definire le singole tracce create dall'intersezione di due fratture. Inizialmente si entra in un ciclo in cui ad ogni iterazione si prende un segmento della prima e della seconda frattura tra cui si vuole trovare una traccia e successivamente si utilizza un secondo ciclo per considerare i singoli vertici delle fratture. Per ogni iterazione, vengono calcolati gli indici del vertice successivo e se il vertice attuale è l'ultimo,viene impostato sul primo vertice (0). Partendo da un vertice per ciascuna frattura, si definiscono i vettori lungo i lati della frattura come differenza tra le coordinate di due vertici consecutivi.

Si verifica che le rette definite da questi vettori non siano parallele alla retta `r` di intersezione tra i piani: se uno dei due lati è parallelo, salta l'iterazione corrente. Si definiscono poi i parametri delle rette della frattura e della retta di intersezione:

$$u_1 = \frac{(\vec{point} \times \vec{s} - \vec{P_1} \times \vec{s}) \cdot (\vec{v_1} \times \vec{s})}{\|\vec{s} \times \vec{v_1}\|^2} \tag{4}$$

$$t_1 = \frac{(\vec{P_1} \times \vec{v_1} - \vec{point} \times \vec{v_1}) \cdot (\vec{s} \times \vec{v_1})}{\|\vec{s} \times \vec{v_1}\|^2} \tag{5}$$

Tramite i due parametri è possibile calcolare i possibili punti di intersezione verificando poi che la loro posizione sia sulla retta definita precedentemente:

$$intersection = \vec{point} + t_1 \cdot \vec{s}$$
$$verify = \vec{P_1} + u_1 \cdot \vec{v_1}$$

Se i punti di intersezione calcolati coincidono con i punti di verifica entro una tolleranza e `u` è compreso tra 0 e 1, cioè si trova all'interno del segmento, i punti sono considerati validi e vengono aggiunti a `Point1` e `Point2`. Dopo aver iterato su tutti i lati delle fratture ottengo un vettore `intersectionPoints` contenente i quattro punti di intersezione trovati: due saranno dati dall'intersezione della retta risultante dall'intersezione tra i piani con la prima frattura (a e b) e gli altri due dall'intersezione della retta con la seconda frattura (c e d). Si definiscono poi le posizioni relative dei quattro punti sulla retta di intersezione tra i due piani proiettando i punti sulla retta. Per facilitare il confronto, le posizioni relative vengono riordinate in modo che la prima sia sempre minore della seconda e la terza minore della quarta. Vengono poi analizzati i diversi scenari possibili per individuare gli estremi della traccia:

- **Caso 1** Se c si trova tra a e b e b è prima di d, la traccia è formata da c e b.
- **Caso 2** Se a si trova tra c e d e d è prima di b, la traccia è formata da a e d.
- **Caso 3** Se c si trova tra a e d e d è prima di b, la traccia è formata da c e d.
- **Caso 4** Se a si trova tra c e b e b è prima di d, la traccia è formata da a e b.
- **Caso 5** Se a e c coincidono e b e d coincidono, i segmenti sono sovrapposti e la traccia è formata da una delle due coppie.
- **Caso 6** Se a e c coincidono mentre b è prima di d, la traccia è formata da a e b.
- **Caso 7** Se a e c coincidono mentre d è prima di b, la traccia è formata da c e d.
- **Caso 8** Se b e d coincidono mentre a è dopo c, la traccia è formata da a e b.
- **Caso 9** Se b e d coincidono mentre c è dopo a, la traccia è formata da c e d.

Per salvare le tracce trovate viene applicata la funzione `SaveTraces`: vengono inseriti all'interno della struttura `Traces` le coordinate degli estremi dell'intersezione, gli ID delle fratture che si intersecano e della traccia generata e la sua lunghezza. Infine, il contatore del numero di tracce (`trace.numTraces`) viene incrementato di uno.

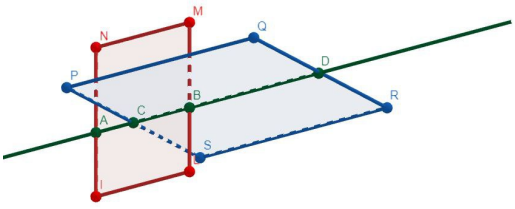

Figure 1: Retta generata dall'intersezione tra fratture (Caso 1)

PrintOnFile

Dopo aver aggiornato il salvataggio delle tracce generate, si stampano le informazioni raccolte all'interno di un file di testo. La funzione `PrintOnFile` è utilizzata per esportare i dati contenuti nella struttura `Traces` in un formato leggibile e facilmente accessibile per ulteriori analisi o archiviazioni. Le informazioni sono scritte in formato scientifico con una precisione di 16 cifre decimali per garantire l'accuratezza dei dati.

TraceReorder

Dopo aver definito le tracce generate, la funzione `TraceReorder` riordina le tracce in passanti e non passanti in ordine di lunghezza decrescente. La funzione inizia iterando su tutte le fratture presenti (`fracture.numFracture`) e all'interno di `trace.fractureId` si cercano le posizioni in cui una delle due fratture che generano la traccia corrisponde alla frattura considerata. Per ogni frattura considerata si cicla su tutti i suoi segmenti e si verifica se uno dei vertici della traccia appartiene al segmento. Si definiscono tre vettori:

- `PQ`, il vettore formato da due vertici consecutivi della frattura;
- `PA`, il vettore formato da un vertice della frattura con un estremo della traccia;
- `PB`, il segmento formato dal vertice della frattura con l'altro vertice della traccia.

Se il prodotto vettoriale tra `PQ` e `PA` oppure tra `PQ` e `PB` è zero, allora uno dei due estremi della traccia appartiene al lato della frattura, dunque un contatore è incrementato di uno. Se terminato il ciclo su tutti i segmenti, il contatore è pari a 2 vuol dire che i vertici della traccia appartengono a due segmenti della frattura, dunque la traccia è passante per frattura in esame e l'indice della traccia viene memorizzato nel vettore `passing`. Altrimenti, l'indice viene memorizzato nel vettore `notPassing`. Vengono anche poi salvate le lunghezze delle tracce rispettivamente in `lengthPassing` e in `lengthNotPassing`, mantenendo l'ordine degli id delle tracce.

Successivamente, tramite la funzione `reordering` i due vettori vengono riordinati in ordine decrescente. Viene creato un vettore di coppie (`couple`), dove ogni coppia è formata da una lunghezza e dall'ID corrispondente. Questo facilita il riordino dei dati mantenendo la relazione tra le lunghezze e i loro rispettivi ID. Dopo il riordino, la funzione aggiorna i vettori originali `idTraces` e `length` con i valori riordinati.

Infine per ogni traccia, la funzione `TraceReorder` crea una tupla contenente l'indice della traccia, un booleano che indica se la traccia è non passante (true se non passante, false se passante) e la lunghezza della traccia.

PrintTraces

Terminato il riordino delle tracce, si stampano su un file di output le informazioni relative ad esse. La funzione procede iterando attraverso tutte le fratture nella struttura `fracture`. Per ogni frattura, recupera il numero di tracce associate alla frattura corrente, stampa il suo ID, il numero di tracce presenti; per ogni traccia associata alla frattura, stampa l'ID della traccia, un booleano che indica se è passante e la sua lunghezza.

Parte 2: Determinare i sotto-poligoni generati per ogni frattura

Strutture utilizzate

In questa seconda parte è stata definita un'altra struttura che si aggiunge alle precedenti due:

PolygonalMesh

Questa struttura definisce una mesh poligonale nella quale vengono salvati i dati delle fratture finali:

- **Cell0D (Vertici)**
  - `numCell0D`: Numero di celle 0D nella mesh.
  - `cellId0D`: Vettore degli identificatori per le celle 0D.
  - `coordinates0D`: Vettore delle coordinate dei punti, dove ogni elemento è un vettore 3D (`Vector3d`) che rappresenta la posizione nello spazio 3D.
- **Cell1D (Lati)**
  - `numCell1D`: Numero di celle 1D nella mesh.
  - `cellId1D`: Vettore degli identificatori per le celle 1D.
  - `verticesId1D`: Vettore delle coppie di identificatori dei vertici che formano i lati, dove ogni elemento è un vettore 2D (`Vector2i`).
- **Cell2D (Poligoni)**

- `numCell2D`: Numero di celle 2D nella mesh.
- `cellId2D`: Vettore degli identificatori per le celle 2D.
- `numVertices2D`: Vettore del numero di vertici per ogni cella 2D.
- `numEdges2D`: Vettore del numero di lati per ogni cella 2D.
- `verticesId2D`: Vettore dei vettori di identificatori dei vertici per ogni cella 2D.
- `edgesId2D`: Vettore dei vettori di identificatori dei lati per ogni cella 2D.

Creazione delle sottofratture

L'obiettivo in questa sezione è tagliare le fratture per creare delle sottofratture e salvarle nella mesh poligonale che abbiamo definito in precedenza attraverso la funzione `FractureCut`. Nel processo si effettuano prima tagli degli tracce passanti con la funzione `createSubfracture`, in ordine di lunghezza, poi di quelle non passanti, dopo averle opportunamente estese tramite la funzione `extendTraces`. È inoltre definita la funzione `createMesh`

fractureCut

La funzione `fractureCut` taglia le fratture in `fracture` lungo le tracce in `trace`, suddividendo le fratture di partenza in sottofratture. Inizia iterando su tutte le fratture e per ognuna, vengono inizializzate strutture di dati per salvare le sottofratture e le tracce associate. Le coordinate dei vertici della frattura originale vengono estratte e memorizzate in `subfractureVertices`, e le tracce associate in `coupleTraces`. La frattura iniziale e le sue tracce vengono salvate in `subFracture`.

La funzione itera su tutte le tracce associate alla frattura corrente, identificata da `traceId`. Per ogni traccia, verifica quali sottofratture (`subFracture`) devono essere tagliate. Se la traccia non è passante, viene estesa fino a incontrare i bordi della sottofrattura tramite la funzione `extendTraces`. La funzione `createSubfracture` divide la frattura corrente in due sottofratture utilizzando la traccia corrente.

Le nuove sottofratture vengono classificate in *destra* e *sinistra* rispetto alla traccia di taglio, calcolando il prodotto vettoriale tra la traccia e un vettore dalla traccia a un vertice della sottofrattura, e confrontando il prodotto scalare tra la normale alla frattura e il prodotto vettoriale. Anche le tracce che intersecano le nuove sottofratture vengono classificate in tracce destre e sinistre. Se una traccia interseca la traccia di taglio, viene calcolato il punto

di intersezione e la traccia viene divisa in due parti, ciascuna assegnata alla sottofrattura corrispondente.

Le sottofratture e le tracce associate vengono memorizzate nella struttura `subFracture1`. Dopo aver processato tutte le tracce, `subFracture` viene aggiornata con il contenuto di `subFracture1`. Al termine del processo, le sottofratture risultanti vengono utilizzate per creare una nuova mesh poligonale con la funzione `createMesh`.

createSubfracture

La funzione ha lo scopo di suddividere una frattura in due sottofratture utilizzando un taglio su una traccia. Viene inizializzato un vettore di punti `Points` che contiene i vertici della frattura e della traccia di taglio nell'ordine in cui sono percorse lungo la frattura. Per ogni vertice della frattura, viene determinato se coincide con uno degli estremi della traccia tramite il prodotto vettoriale tra il vettore del lato della frattura e il vettore dalla frattura a ciascun estremo della traccia. A questo punto vengono identificati e salvati i punti che definiscono le due sottofratture separate. La frattura viene divisa in due parti lungo la traccia di taglio e le due sottofratture risultanti vengono memorizzate nel vettore `subfractureVertices1`.

extendTraces

Questa funzione ha lo scopo di estendere una traccia di taglio fino a intersecare i lati della sottofrattura da prendere in esame. Viene definito un vettore `vec` che rappresenta la direzione della traccia da estendere, calcolato come la differenza tra il secondo e il primo vertice della traccia; in seguito viene inizializzato un punto `point` che corrisponde al primo vertice della traccia. Si inizializza poi un vettore `extendedVertices` che conterrà i vertici della traccia estesa. Per ogni lato della sottofrattura, viene determinato se il lato e la traccia da estendere non sono paralleli. Se sono paralleli, il processo passa al lato successivo della sottofrattura,altrimenti viene calcolato il punto di intersezione tra la retta che passa per il segmento della sottofrattura e la retta che passa per il lato della traccia considerato. Se $0 \leq u \leq 1$, cioè si trova all'interno del segmento, il punto di intersezione è valido e viene aggiunto a `extendedVertices` come estremo della traccia estesa. Alla fine del processo, `extendedVertices` contiene i vertici della traccia estesa.

createMesh

Trasforma le sottofratture in input in una rappresentazione poligonale: prima di procedere con la creazione, vengono riservati gli spazi per i vertici e i lati

della mesh poligonale, per ogni sottofrattura, i vertici vengono estratti e verificati per determinare se sono già presenti nella mesh e in caso contrario vengono aggiunti. Per ciascun lato della sottofrattura, si verifica la sua presenza nella mesh e se non è già salvato, viene aggiunto. Infine per ogni sottofrattura, viene creata una cella 2D nella mesh, specificando i `verticesId2D` e `edgesId2D`.

Parte 3

Google Test

Per la quasi totalità delle funzioni utilizzate sono stati eseguiti dei `Google Test`. Tali test sono suddivisi in diverse categorie che verificano specifiche funzionalità e componenti del codice. È stata controllata la corretta esecuzione sia dei casi validi che dei casi non validi.

Esportazione su Paraview

Tramite l'utilizzo di Paraview abbiamo esportato i dati sulle fratture e sulle tracce per una visualizzazione grafica di questi. Sono state create le funzioni `importPolygonsList` e `importSegments` per la lettura e l'importazione di fratture e tracce rispettivamente con l'ausilio di tre strutture `Triangle`, `Polygons` e `Segment`. Inoltre si è utilizzata la libreria `GeDiM` per la triangolarizzazione ed esportazione effettiva degli oggetti.

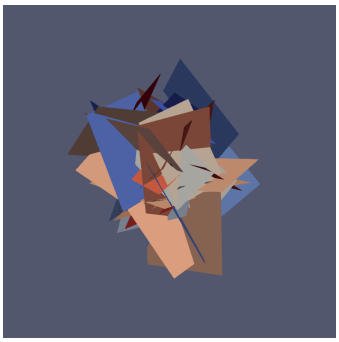

Figure 2: Frattura per FR50

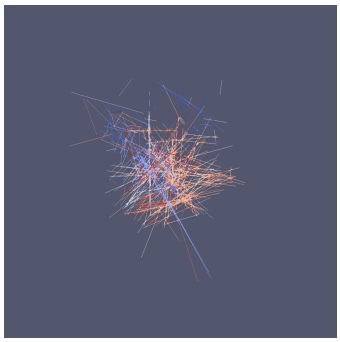

Figure 3: Tracce per FR50

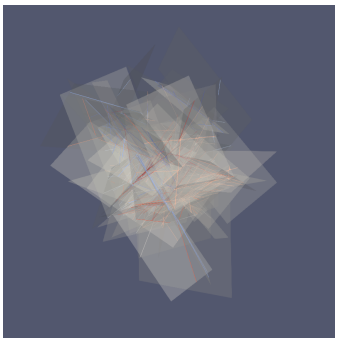

Figure 4: Fratture e tracce FR50