Politecnico di Torino

PROGETTO PCS 2024

DISCRETE FRACTURE NETWORK

Brocco Arianna, 297644

Foni Gianmarco, 298752

Maccarone Giovanna, 298226

Sommario

ABSTRACT

Il seguente report contiene le informazioni e la descrizione dello svolgimento del progetto di Programmazione e Calcolo Scientifico 2024 in cui viene gestito un DFN. Oltre al codice, analizzato nelle seguenti pagine, sono stati utilizzati i GoogleTest per eseguire test sulle singole funzioni utilizzate nella prima parte del progetto.

CODICE IN C++

STRUTTURA DATI IN INPUT

La registrazione dei dati in input è fondamentale per riuscire a manipolare le informazioni ottenute e determinare le tracce associate a ciascun poligono. È importante stabilire quali sono i dati necessari per le singole operazioni in modo da ottimizzare lo spazio di memoria, salvando esclusivamente le informazioni indispensabili. La scelta della struttura dati adibita alle singole categorie deve essere ottimale e di facile accesso: deve pertanto essere eseguita un'analisi accurata delle possibili tipologie di strutture, che è riportata in seguito.

La **struct** denominata **FractureStruct** è ideata per la gestione dei dati relativi alle fratture. In ogni file è contenuto il numero di fratture il quale viene salvato nella variabile **NumeroFratture** come **unsigned int.** Ogni frattura è contraddistinta da un numero identificativo, **unsigned int** anch'esso**,** registrato in un array di dimensione data da **NumeroFratture** ed è composta da un certo numero di vertici, salvato all'interno di un array **vector<unsigned int>** chiamato **NumeroVertici.**
Si è scelto di utilizzare gli array in quanto non si conoscono a priori le loro dimensioni, stabilite poi in base ai dati letti nei file. Una volta inizializzato l'array, è possibile ridimensionarlo tramite **resize** a seconda della necessità.
CoordinateVertici, del tipo **vector<MatrixXd>,** è un array di lunghezza corrispondente alla variabile **NumeroFratture,** in cui ogni posizione presenta una matrice contenente i vertici del poligono. **MatrixXd** è di dimensione inizialmente ignota per poi assumere dimensione 3***NumeroVertici.** Questo tipo di struttura è particolarmente efficiente per accedere con facilità ai dati poiché è sufficiente scegliere la matrice del poligono che si sta analizzando ed estrarre le colonne le cui componenti corrispondono alle coordinate x, y e z dei suoi vertici.
In vista della necessità di stampare dei file in output in cui le tracce devono apparire in ordine di lunghezza decrescente e distinte in passanti e non passanti, si è deciso di aggiungere due **vector<list<unsigned int>>** (**NumeroTracceN** e **NumeroTracceP**) contenenti, per ogni frattura,la lista degli Id delle tracce che la attraversano. Infine, nella struct, è stato aggiunto il **vector<Vector3d> NormaleFrattura** affinché le normali ai piani su cui giaciono le fratture, siano facilmente accessibili in vista del loro utilizzo nella seconda parte.

La **struct** denominata **TracesStruct** è stata ideata per gestire le informazioni relative alle tracce.
La prima variabile è **ct**, un contatore unsigned int che memorizza il numero di tracce presenti. Gli id delle tracce sono invece salvati in un **vector<unsigned int> IdTracce.** Per ogni traccia, in **vector<Matrix<double,2,3>> EstremiTracce** vengono salvate le coordinate dei suoi estremi nelle posizioni corrispondenti agli id delle tracce su una matrice in modo da avere sulla prima riga le coordinate del primo estremo e sulla seconda quelle del secondo. **LunghezzaTracce**, invece, è un **vector<double>** in cui è memorizzata la lunghezza di ogni traccia. Infine, è stata introdotto un **vector<Matrix<unsigned int,2,2>> PNP,** contenente in posizione i-esima, una matrice in cui la prima colonna fornisce gli Id delle fratture generanti la traccia, mentre nella seconda troviamo due variabili che assumono valore 0 nel caso in cui la traccia sia passante per quella determinata frattura o 1 in caso contrario. Questa struttura viene utilizzata solo nelle funzioni di Output. Un'alternativa

all'utilizzo di PNP potrebbe essere un ciclo for che controlli in NumeroTracceN e NumeroTracceP la presenza o meno dell'Id relativo ad una traccia.

La **struct Mesh,** utilizzata per la seconda parte del progetto, contiene le informazioni relative alle mesh che formano i sottopoligoni generati dal taglio delle fratture. È composta da tre sezioni: le celle 0D, le celle 1D e le celle 2D corrispondenti ai vertici, ai lati e ai poligoni formatisi. Man mano che si procede con il taglio dei poligoni vengono salvati nella struct i dati relativi ad una certa frattura, per poi procedere con le altre sovrascrivendo ogni volta la mesh.

PRIMA PARTE: FUNZIONI

IMPORT DATA

La funzione addetta alla lettura di un DFN da un file e al loro salvataggio nella struttura **FractureStruct** precedentemente descritta è denominata **ImportData**. È una funzione di tipo bool in modo da verificare se l'importazione è andata a buon fine (bool = true). La funzione riceve in input il nome del file da leggere e la struttura. Dopo aver aperto il file del DFN contenente i poligoni in analisi e restituito un errore in caso di inesistenza del suddetto documento, legge in sequenza le righe di informazioni tramite **getline.** Vengono scartate le linee prive di dati da salvare, come titoli o intestazioni, e, progressivamente,convertite in stringhe le altre. Dalle stringhe ottenute si estrapolano i dati e vengono successivamente associati alle rispettive variabili create e inizializzate nella struct in base al ruolo che rivestono. Nel processo di lettura dei file input è necessario registrare la quantità di informazioni (di vario tipo) presenti in modo da poter ridimensionare gli array della struttura adeguatamente.
L'utilizzo di un ciclo **for** permette di leggere le righe a gruppi di 6, numero che corrisponde alle righe dedicate a ogni frattura all'interno del file. Vengono ignorati i separatori ";" e vengono associati i dati di ciascun poligono alle variabili inizializzate per salvarli nella struttura. In questa prima fase, tutte le variabili sono di tipo **unsigned int** tranne quelle relative alle coordinate dei vertici inizializzate, invece, come **double**. Inoltre, per inserire in struttura queste ultime, è necessaria l'inizializzazione di matrici, riempite tramite un ciclo **for,** con le giuste componenti, in modo da associare ad ogni colonna le coordinate di un vertice.

FRACTURE PLANE

Per trovare il piano su cui giace il poligono è sufficiente utilizzare i primi tre vertici dello stesso, di seguito indicati con v_1, v_2, v_3. Fissato v_1, la funzione calcola i vettori direzionali $\boldsymbol{n_1 = v_2 - v_1}$ *e* $\boldsymbol{n_2 = v_3 - v_1}$
. Dopodiché, viene calcolata la giacitura del piano tramite il prodotto vettoriale $t = n_1 \times n_2$ ottenendo, quindi, i coefficienti *a,b,c* dell'equazione implicita del piano $ax + by + cz + d = 0$. Ultimo step: calcolare il termine noto imponendo l'appartenenza del vertice v1 al piano:

$$d = -(ax_1 + by_1 + cy_1).$$

Infine, i coefficienti che identificano il piano vengono salvate in un **Vector4d**.

PARALLEL PLANES

La funzione **ParallelPlanes** controlla se i piani su cui giacciono due fratture analizzate sono paralleli. Infatti, se lo fossero, non ci sarebbe bisogno di passare a step successivi per l'eventuale calcolo della traccia. Si sfrutta il prodotto vettoriale tra le giaciture dei piani contenenti le fratture: nel caso in cui è zero a meno di una tolleranza (relativa), i piani sono paralleli.

INTERSECTION LINE

La funzione **IntersectionLine** calcola la retta di intersezione tra due piani dati in input, essa chiama,per ciascun piano, **FracturePlane** in modo da salvarne i coefficienti identificativi. Successivamente, salva le giaciture dei piani n_1 e n_2 (ovvero le prime tre componenti dei vettori) tramite cui si ottiene la direzione tangente della retta $t = n_1 \times n_2$. La risoluzione del seguente sistema

$$\begin{pmatrix} n_{10} & n_{11} & n_{12} \\ n_{20} & n_{21} & n_{22} \\ t_0 & t_1 & t_2 \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ p_2 \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ 0 \end{pmatrix}$$

tramite l'ausilio della funzione $fullPivLu$ permette di trovare le coordinate del punto p, ovvero del punto di applicazione della retta. Quest'ultima, quindi, può essere individuata, basandosi sulla forma parametrica di una retta, tramite il punto di applicazione p e la direzione. La funzione restituisce una matrice di dimensione 2x3 avente le coordinate del punto di applicazione sulla prima riga e la tangente sulla seconda.

COMPUTE BOUNDING BOX E BBOX INTERSECTION

Le funzioni **ComputeBoundingBox** e **BBoxIntersection**, operano rispettivamente su una e due fratture date in input, e determinano,la prima, il bounding box di una frattura e, la seconda, se due fratture si intersecano. Al fine di calcolare la bounding box, il primo passo è quello di inizializzare due vettori, ***vettoreMax*** e ***vettoreMin***. ***VettoreMax*** viene inizializzato con i valori più piccoli possibili (o più negativi, se corretto), in modo che qualsiasi valore letto dai vertici della frattura sarà sicuramente maggiore e quindi aggiornerà **vettoreMax**. Questo permette di trovare i valori massimi corretti per ogni coordinata. Analogamente, **vettoreMin** viene inizializzato con i valori più grandi possibili. Questo permette di trovare i valori minimi corretti per ogni coordinata. La bounding box, individuata con i vettori appena calcolati, viene salvata in una matrice.
La seconda funzione controlla se le bounding box si intersecano in tutte e tre le dimensioni (x, y, z). Per ogni dimensione, verifica se il minimo della prima bounding box è minore del massimo della seconda bounding box, aggiustato con la tolleranza e se il massimo della prima bounding box è maggiore del minimo della seconda bounding box, aggiustato con la tolleranza. La funzione restituisce un booleano uguale a *true* se c'è intersezione.

CHECK INTERSEZIONE

La funzione **checkIntersezione** effettua un "check complessivo" sull'eventuale intersezione di due fratture. Essa chiama la maggioranza delle funzioni sopra descritte per verificare l'intersezione e l'eventuale presenza di tracce esistenti tra due fratture.

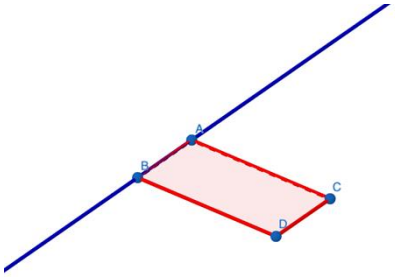
DESCENDING ORDER

La funzione **descendingOrder** è stata progettata per inserire l'identificatore della traccia (num) in una lista in ordine decrescente rispetto alla lunghezza. Viene estratta la lunghezza della traccia

corrispondente all'id num. Itor = list.begin() inizializza un iteratore (puntatore) per scorrere la lista dall'inizio.
Il ciclo while cerca il primo punto nella lista dove la lunghezza della traccia corrente (associata all'id puntato da itor) è maggiore della lunghezza della traccia num più una tolleranza tol. Questo assicura che num venga inserito prima di qualsiasi traccia con una lunghezza maggiore nella lista. Durante il ciclo, l'iteratore itor viene incrementato per puntare all'elemento successivo della lista.

CHECK TRACCIA

CheckTraccia rappresenta il corpo centrale nello svolgimento della prima parte del progetto. Essa, infatti, ha il compito di verificare se due fratture si intersecano lungo una linea e, in caso positivo, determina gli entremi della traccia di intersezione. Viene in introdotto un **vector<double>** ***ts*** in cui verranno inseriti i parametriu di intersezione tra la retta e i vari lati. Il primo passo è rappresentato dall'estrazione del punto di applicazione e della direzione della retta di intersezione dei piani su cui giacciono le fratture. Dopodiché, per ognuna delle due fratture, si crea un ciclo while che itera su tutti i lati della frattura per verificarne l'intersezione con la retta. All'interno del ciclo, si è scelto di lavorare in modulo NumeroVertici in modo da includere tutti i lati e tutti i vertici solo una volta. Dopo aver estratto le coordinate dei vertici, estremi di un lato, e averne calcolato la direzione come differenza delle coordinate dei vertici, si fa una prima distinzione in due macro-casi: la retta di intersezione e il lato sono paralleli e coincidenti (come nell'immagine di fianco) oppure non sono paralleli e si trova un punto di intersezione. Nel primo caso, considerato quando il prodotto vettoriale tra la direzione della retta è quella del lato è nullo, la traccia è rappresentata dal lato, conseguentemente vengono salvati gli estremi del lato. Il secondo caso analizzato è quello in cui il lato della frattura e la retta si intersecano in un singolo punto. Nel suddetto caso, si calcola il parametro di proporzionalità k con la seguente formula:



$$k = \frac{(point \times \text{lineDir} - \text{vertex0} \times \text{lineDir}) \cdot (\text{edgeDir} \times \text{lineDir})}{\|edgeDir \times \text{lineDir}\|^2}$$

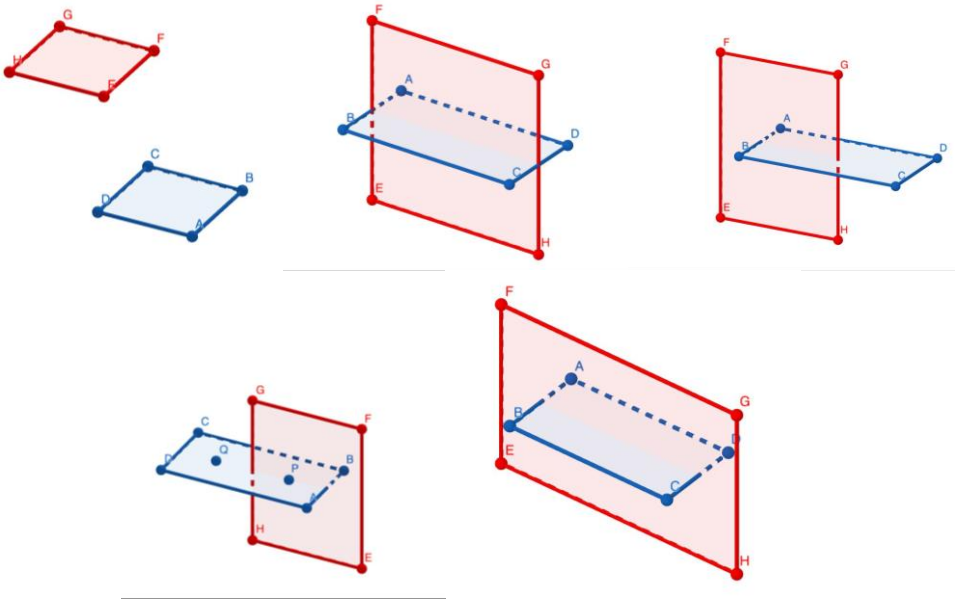
In base al valore assunto da k si distinguono le seguenti situazioni:

$$k = \begin{cases} 0 \;\; allora\ il\ punto\ di\ intersezione \text{ è } vertex0 \\ \in (0,1)\ allora\ il\ punto\ di\ intersezione \text{ è } interno\ al\ lato \\ 1\ allora\ il\ punto\ di\ intersezione \text{ è } vertex1 \end{cases}$$

Se una di queste situazioni è soddisfatta, allora si passa al calcolo del secondo parametro t con la seguente formula:

$$t = \frac{(vertex0 \times \text{edgeDir} - \text{point} \times \text{lineDir}) \cdot (\text{lineDir} \times \text{edgeDir})}{\|edgeDir \times \text{lineDir}\|^2}$$

Tale parametro indica l'esatto punto di intersezione tra la retta e il lato. Esso viene salvato in ***ts***. Un modo alternativo per verificare l'intersezione è basato sulle boundingbox ed è inserito in modalità #ifdef TEST.
Dopo aver eseguito i controlli ed aver calcolato i valori del parametro t per entrambe le fratture, si passa all'analisi delle varie casistiche di tracce, salvate tramite due booleani pass1 e pass2 con valore true se la traccia è passante per la frattura1/2. I casi presi in analisi sono illustrati nelle seguenti immagini:



Una volta determinati i punti di intersezione per ogni coppia di fratture, il vettore ***ts***, che contiene i parametri, viene ordinato in ordine crescente tramite la funzione disponibile sort (di costo computazionale O(nlogn)) e si fa riferimento alle due coordinate centrali ***ts[1]*** e ***ts[2].*** Viene creato un nuovo identificatore per la traccia (**num**) e viene aggiunto a **trac.IdTracce**; vengono calcolati e aggiunti gli estremi della traccia (p1 e p2) nel formato (**Matrix<double, 2, 3>**), utilizzando la formula della linea di intersezione. Infine, è stata aggiunta la lunghezza della traccia in **trac.LunghezzaTracce**, calcolata come la norma della differenza tra p2 e p1. L'output della funzione è un booleano che ha valore true se esiste intersezione tra le fratture.

OUTPUT FRACTURES E OUTPUT TRACES

Una volta eseguiti tutti i calcoli necessari per verificare l'intersezione e la presenza di tracce, sono state create tre funzioni di output per stampare il tutto su dei file.
La funzione **OutputFractures** si occupa di stampare i dati relativi alle fratture: per ogni frattura, è presente una lista delle tracce che la attraversano divise in due sottogruppi, passanti e non passanti; a sua volta, ogni sottogruppo presenta le diverse tracce ordinate in modo decrescente per lunghezza.
La funzione **OutputTraces** si occupa di stampare i dati relativi alle tracce: per ogni traccia vengono stampati l'id, le coordinate degli estremi e le fratture che sono coinvolte in essa.

SECONDA PARTE:

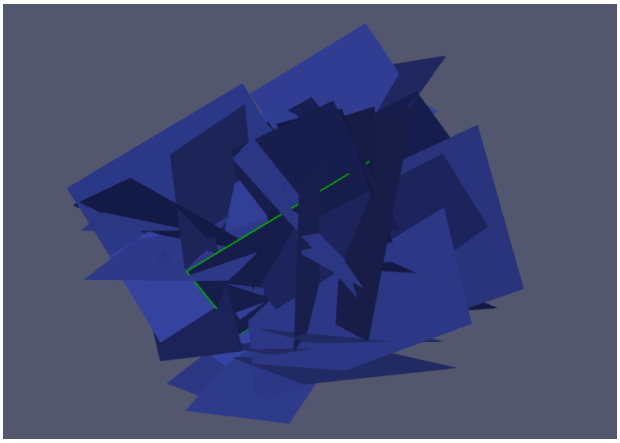
SUBPOLYGONS

La parte due si incentra sul taglio delle fratture in corrispondenza delle loro tracce, sono stati considerati esclusivamente i sottopoligoni ottenuti a partire dalle tracce passanti. I dati relativi ai sottopoligoni ottenuti vengono salvati sulla struttura mesh che ad ogni iterazione (corrispondente ad un certo poligono) sovrascrive le informazioni relative alle Cell0D, Cell1D e Cell2D.
La funzione intorno alla quale ruota questo procedimento è **subPolygons**, la cui principale differenza rispetto alle funzioni della prima parte del progetto è il fatto di prendere in input delle **list**di più agevole fruizione rispetto alle intere struct passate in precedenza.

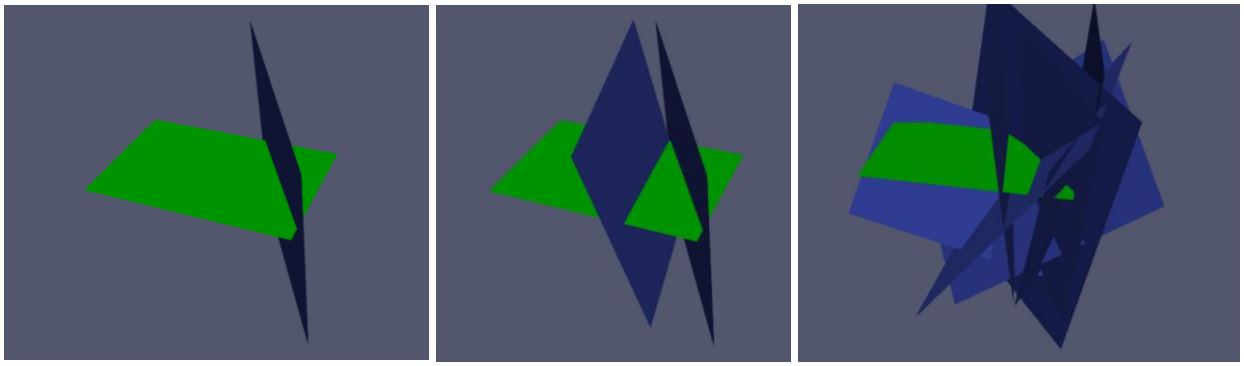
Nel **main** vengono inizializzate, dentro un **for** che cicla sulle fratture, una **list<MatrixXd> sp** le cui matrici corrisponderanno ai sottopoligoni ottenuti dal taglio di una singola frattura, un **vector<Matrix<double,2,3>> coordEstremiTracce** con memorizzate in matrici le coordinate degli estremi delle tracce, una **list<Vector3d> verticiPolygons** con i vertici della frattura e una**list<unsigned int> lista Tracce** con tutte le tracce passanti per essa. Viene poi prelevato dalla struttura delle fratture il vettore normale al piano su cui giace la frattura presa in considerazione. Successivamente, inizia il processo di taglio durante il quale **lista Tracce** verrà progressivamente svuotata dalle tracce già utilizzate fino a che non ne conterrà nessuna. Viene chiamata la funzione **subPolygons** che restituisce un booleano che assume valore true nel momento in cui il processo vaa buon fine. Il modo in cui si è scelto di operare comprende la creazione di due **list<Vector3d>destra** e **sinistra** contenenti tutti i vertici che si trovano a sinistra della prima traccia presa come riferimento e tutti quelli a sinistra.
Prendendo una coppia di vertici per volta, tramite un **while,** e cancellandoli dalla lista dei vertici del poligono considero più casi. Se uno dei due estremi della traccia appartiene al lato, cioè se il prodotto vettoriale tra la direzione della retta passante per la traccia e quella passante per un estremo di quest'ultima e il vertice considerato è nullo, a meno di una tolleranza, viene salvatodirettamente in entrambe le liste. Se il prodotto scalare del prodotto vettoriale, considerato nel caso precedente, con la normale al piano su cui giace la frattura è positivo il vertice viene messo nella lista di destra altrimenti nella sinistra.
Successivamente vengono distinte le tracce della lista rispetto a quella di riferimento tra quelle a destra, a sinistra o ancora quelle secanti da ritagliare rispetto alla principale, nello stesso modo utilizzato per suddividere i vertici. La differenza è che, in questo caso, se entrambi i vertici stanno a destra della retta di riferimento la traccia verrà messa nella lista a destra altrimenti a sinistra, se invece un vertice si trova a destra e l'altro a sinistra è necessario tagliare la traccia e mettere un segmento nella lista di destra e uno nella lista di sinistra. Per calcolare gli estremi dei due nuovi segmenti che saranno dati da i punti della traccia e il punto di intersezione con la traccia rispetto alla quale sta avvenendo la suddivisione, viene utilizzato un sistema lineare risolto con **fullPivLu().solve()**.
Raccolti tutti i dati, questi vengono salvati all'interno di **MatrixXd vertici** inserita nella lista **sp** dei sottopoligoni. La funzione **subPolygons** viene poi richiamata ricorsivamente sulle due liste, esplorando i sottopoligoni ottenuti in successione fino a terminare tutte le tracce. La condizione di terminazione si ha nel momento in cui sia sulla lista di destra che di sinistra non sono più presenti tracce rispetto cui tagliare. Si tratta, quindi, di un algoritmo di tipo ricorsivo che segue una struttura ad albero binario: si esplora l'albero alla ricerca delle foglie (i sottopoligoni) che vengono aggiuntea **sp** e si risale nell'albero fino all'ultimo nodo esplorato.
L'ultimo passo è salvare le informazioni, ricavate dal taglio, sulla mesh. E' necessario, a priori,riservare lo spazio che occorre all'interno della mesh tramite **reserve**, questo sarà per **NumeroDiLati** e **NumeroDiVertici** il numero delle matrici in **sp** e per tutte le altre variabili dalla somma del numero di colonne di ogni matrice.
Gli **Id** di vertici e lati vengono incrementati nel momento in cui questi vengono trovati e salvati in **Cell0D** e **Cell1D**. Per le operazioni sulle liste sono stati utilizzati dei puntatori che scorrono la lista dall'inizio alla fine passando da un sottopoligono all'altro.
Per riempire **Cell0DCoordinates**, corrispondenti alle coordinate dei vertici, con un primo **while** sui sottopoligoni si confronta ogni punto con quelli già memorizzati nella struct, se non è già presentelo si inserisce sia tra gli **id** che tra le **coordinate**. Con un secondo **while** si passa al salvataggio dei **lati** per cui vengono creati due **VectorXi lati** e **vertici** (a cui progressivamente si aggiungono i lati e i vertici corrispondenti al sottopoligono), si costruisce un ciclo sul numero dei lati ed estrapolandoi vertici a coppie dalla matrice corrispondente al sottopoligono, si stabiliscono gli id dei vertici che formano i lati. Dopo aver cercato tra le coordinate dei vertici quelle che formano il lato considerato,viene salvato l'id che corrisponde alla posizione del vettore del vertice in **Cell0DCoordinates**. Lacoppia degli id viene salvata in **Vector2i edge** che con **push_back** è

inserita in **Cell1DVertices**.Infine gli indici di tutti i vertici vengono salvati in **Cell2DVertices** e quelli dei lati in **Cell2DEdges.**

Esempio di taglio e salvataggio dei dati della frattura numero 3 (del file FR50) nella mesh:



Prima di usare subPolygons si considera la traccia più lunga della lista delle passanti che viene presa come riferimento nella suddivisione in lista di destra e sinistra. Si passa poi al taglio tramite la seconda traccia a destra della prima, poi la terza e così via fino alla fine della lista.



Alla fine del taglio, dopo il salvataggio dei dati nella mesh si ottiene la suddivisone della frattura in tutti i sottopoligoni ottenuti.