Progetto PCS

András Almási, Dario Chito, Alessia Catani

Sessione di luglio 2024

1 Introduzione

La simulazione di flussi in mezzi fratturati ‘e di fondamentale importanza per diverse applicazioni nel campo ambientale e idrogeologico. Ad esempio, pu‘o essere utile modellare la migrazione e l’estrazione di petrolio attraverso i sistemi di fratture del suolo per migliorare l’efficienza delle operazioni di perforazione, riducendone cos‘ı i rischi. Un possibile approccio a questi problemi ‘e quello di usare un Discrete Fracture Network (DFN), ovvero un sistema costituito da N fratture, ciascuna rappresentata da un poligono planare convesso, organizzate in una rete tridimensionale e che si intersecano lungo segmenti chiamati tracce. Ogni traccia ‘e data dall’intersezione di due soli poligoni e, rispetto a ciascuna

frattura, possiamo distinguerla in passante, quando entrambi gli estremi giac- ciono sul bordo della frattura stessa, e non-passante se ha almeno un suo estremo

all’interno della frattura stessa. L’obiettivo principale del progetto ‘e quello di sviluppare un codice C++ che analizzi un DFN e ne estragga le sue propriet‘a. In particolare, il progetto si articola in due parti:

• dopo aver letto un DFN da file, determiniamo le tracce che lo costituiscono, cercando quindi le varie intersezioni tra i poligoni convessi. Inoltre, per ciascuna frattura, differenziamo le varie tracce in passanti e non-passanti e ordiniamo i due sottoinsiemi di tracce per lunghezza in ordine decrescente;

• per ciascuna frattura del DFN determiniamo i sotto-poligoni generati dal taglio della frattura con le sue tracce, in modo tale da minimizzare il numero di sotto-poligoni che si vengono a formare. Adottiamo queste procedure per 6 esempi di DFN, costituiti rispettivamente da 3, 10, 50, 82, 200 e 362 fratture. Tali DFN differiscono non solo per il numero di fratture che contengono, ma anche per il numero di tracce che generano, variando da scenari pi‘u realistici con un numero elevato di tracce a scenari meno probabili, in cui il numero di intersezioni di fratture ‘e davvero molto esiguo.

2 Importazione del DFN

Il primo passo ‘e quello di importare un DFN da file. Ciascun file contiene il numero di fratture e la lista di fratture. Per ciascuna frattura, inoltre, ‘e ripor-

tato 1 l'identificativo e il numero dei vertici, seguiti dalla lista delle coordinate spaziali dei vertici, descritte come una matrice di dimensione 3 x numero vertici. Al fine di memorizzare le informazioni contenute nel DFN, abbiamo utilizzato una struct, dal nome "DFN", con attributi il numero di fratture e i loro identificativi, il numero di vertici e una rispettiva matrice delle loro coordinate e il numero di tracce totali che si generano. Inoltre tale struct 'e dotata anche di due particolari variabili, ovvero:

• un vettore dinamico, di nome Fractures, i cui elementi sono a loro volta delle struct, di nome Fracture, che contengono le informazioni relative ad una singola frattura;

• un vettore dinamico, di nome Traces, i cui elementi sono a loro volta delle struct, di nome Trace, che contengono le informazioni relative ad una singola traccia.

Una volta creato l'oggetto DFN, si passa all'importazione vera e propria con la funzione ImportDFN(filepath,dfn) che prende in input una struct DFN, non che il nome del file in cui si trova, e restituisce un booleano: vero se il DFN 'e stato importato correttamente, falso se accade il contrario. Tale funzione dap- prima apre il file in modalit'a lettura e, tramite il comando getline(), scorre le righe del file. Viene quindi estratto il numero di fratture presenti e convertito da stringa a unsigned int attraverso un oggetto istringstream, che consente di trattare una stringa come un flusso di input e di estrarne i dati con l'operatore di estrazione. Dopodichè, con lo stesso procedimento, legge ed estrae, per ciascuna frattura, il suo identificativo, il rispettivo numero di vertici, legge le loro coordinate e le inserisce in una matrice di dimensioni 3 x numero di vertici e calcola il baricentro attraverso una funzione scritta ad hoc. Infine, per ogni frattura, ImportDFN crea una struttura di tipo Fracture, che viene popolata con le informazioni precedentemente raccolte, e successivamente aggiunta ad un vettore dinamico che le contiene tutte.

3 L'algoritmo per individuare le fratture

Esso si trova nelle funzioni e "find_trace_coplanar(fracture1,fracture2, dfn)", contenute in "AlgFractures.cpp". Queste funzioni ritornano vero se le fratture si intersecano, falso in caso contrario Diamo un breve elenco di quelle che ci sembrano le caratteristiche più particolari dell'algoritmo, per poi dare una descrizione del processo.

- Come anche in tutto il resto del progetto che concerne i calcoli geometrici, non si è fatto uso della libreria eigen, ma si sono construite funzioni inline per calcolare prodotti scalare, vettoriale, scalare-vettore etc.
- Anche per quanto concerne la memorizzazione dei vertici, non si è usata Eigen, ma array di double della std.

- Nessuna risoluzione di sistemi lineari è invocata, le sole operazioni utilizzate sono quelle di prodotto vettoriale, scalare, riscalamento, somma, sottrazione di vettori.
- Queste scelte sono state fatte secondo il seguente principio: più ci sono note le funzione e le classi di cui facciamo uso, più l'algoritmo e la sua efficenza è sotto il nostro controllo.
- Per il particolare caso in cui due fratture siano coplanari, si è usata una funzione per trovare le tracce diversa da quella utilizzata nell'altro caso.

Nel seguito utilizziamo le seguenti notazioni:

- P_i^1 per indicare il vertice i-esimo della prima frattura.
- P_j^2 per indicare il vertice j-esimo della seconda frattura.
- B^1 e B^2 sono i baricentri della prima e della seconda frattura.
- n^1 ed n^2 sono i versori normali al piano di ciascuna frattura.
- f1 e f2 per denotare la prima e la seconda frattura rispettivamente

Spieghiamo prima l'algoritmo nel caso non coplanare. In esso abbiamo diviso la procedura in cinque passi.
Al passo uno e due si considerano i vertici della prima frattura e il piano nel quale giace la seconda frattura, individuato dal baricentro di quest'utlima e il versore normale ad essa. Si classificano i vertici della prima frattura in base al lato del piano di f2 su cui giacciono. Per fare ciò si calcolano

$$h_i = (P_i^1 - B^2) \cdot n^2$$

I P_i^1 con gli h_i dello stesso segno stanno sullo stesso lato. Chiaramente se gli h_i sono tutti dello stesso segno, le fratture non si intersecano. Se invece $h_i = 0$ per qualche i, abbiamo trovato un vertice che giace esattamente sul piano di f2. In particolare al passo uno si trova il primo vertice che passi sull'altro lato del piano rispetto a P_0^1 e al passo due il primo vertice che ritorni sullo stesso lato di P_0^1.
Al termine dei passi uno e due (a meno che non si sia trovata qualche vertice che giaccia esattamente sul piano di f2), abbiamo trovato due coppie di vertici (P_i^1, P_{i+1}^1) e (P_j^1, P_{j+1}^1) tali che i due membri di ciascuna coppia stanno uno sul lato opposto dell'altro, rispetto al piano di f2.

Nella figura è mostrata la sezione perpendicolare al piano di f2, dove la retta è porprio il piano di f2, e si ha:

$$\begin{cases} a_1 = |(P_i^1 - B^2) \cdot n^2| \\ a_2 = |(P_{i+1}^1 - B^2) \cdot n^2| \\ b_1 + b_2 = |(P_i^1 - P_{i+1}^1)| \end{cases} . \tag{1}$$

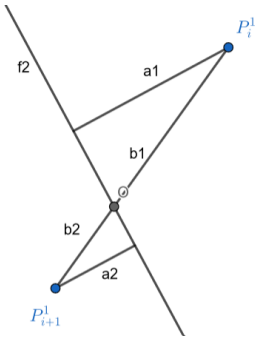

Figure 1: Sezione perpendicolare al piano di f2

Il punto O è dato da

$$P_i^1 + \alpha(P_{i+1}^1 - P_i^1) \quad \text{con } \alpha = \frac{a_1}{a_1 + a_2}$$

. In questo modo ci possiamo trovare O e O' i due putni di intersezione dei segmenti $\overline{P_i^1 P_{i+1}^1}$ e $\overline{P_j^1 P_{j+1}^1}$ con il piano di f2.
Al passo quattro classifichiamo i vertici di f2 in base al lato su cui giacciono rispetto alla retta OO'. Per fare ciò si calcolano

$$m_i = (P_i^2 - O) \cdot (\frac{(O' - O)}{||O' - O||} \times n^2)$$

I P_i^1 con gli m_i dello stesso segno stanno sullo stesso lato. Chiaramente se gli m_i sono tutti dello stesso segno, le fratture non si intersecano. Se $m_i = 0$ per qualche i, abbiamo trovato un vertice che giace esattamente sulla retta di OO'.
Una volta trovate le coppie di vertici (P_i^2, P_{i+1}^2) e (P_j^2, P_{j+1}^2) che passano uno dal lato opposto dell'altro rispetto a OO', con le stesse operazioni del passo tre ci troviamo le intersezioni G e G' dei segmenti $\overline{P_i^2 P_{i+1}^2}$ e $\overline{P_j^2 P_{j+1}^2}$ con la retta OO'.
A questo punto, se esiste la traccia delle fratture avrà per vertici due dei quattro punti collineari O, O', G, G'. Per determinare quali siano, si esegue una catena di if ... else nei quali si connsiderano i valori

$$l_1 = (G - O) \cdot (\frac{(O' - O)}{||O' - O||})$$

$$l_2 = (G' - O) \cdot (\frac{(O' - O)}{||O' - O||})$$

Se essi sono entrambi negativi, o entrambi maggiori di $||O - O'||$, tra le fratture non c'è intersezione. Individuati i vertici della traccia, viene creato un nuovo oggetto traccia, a cui viene assegnato un id, i vertici, la lunghezza della traccia, le due fratture che lo identificano e per ognuna di esse viene indicato con un

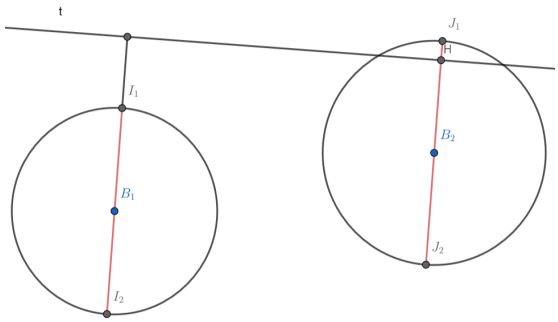

Figure 2: Veduta "dall'alto" dei due cerchi che aprossimano le fratture.

booleano se è passante o non passante. Nelle due fratture invece si aggiunge l'id della nuova traccia alla lista delle loro tracce.

Il caso coplanare è alquanto più semplice poichè ci può essere una traccia comune solo se due lati dei due poligoni si sovrappongono. Infatti si sta assumento che gli interni delle superfici dei due poligoni non si possano intersecare.
Siano n ed m il numero di vertici di f1 e f2 rispettivamente. Per ognuno dei vertici P_i^1 si calcola

$$p_{i,j} = (P^1_{(i+1)\%n} - P^1_i) \cdot (P^2_j - P^1_i)$$

Se

$$p^2_{i,j} = ||(P^1_{(i+1)\%n} - P^1_i)||^2 ||(P^2_j - P^1_i)||^2$$

significa che P_j^2 giacce sulla retta $P_i^1 P^1_{(i+1)\%n}$. Se anche per $P^2_{(j+1)\%m}$ si trova che giacce su tale retta, allora il lato $\overline{P^2_j P^2_{(j+1)\%m}}$ giace su tale retta. A questo punto, se esiste, la traccia delle due fratture avrà per vertici due dei quattro punti P_i^1, $P^1_{(i+1)\%n}$, P_j^2 e $P^2_{(j+1)\%m}$. Per trovare quali siano e se esistono, vengono ripetute le stesse identiche casistiche del passo cinque dell'algoritmo non coplanare.

4 L'esclusione preliminare della possibilità d'intersezione

Il codice di questa parte è contenuto nella funzione
"Exclude(fracture1, fracture2)" di "AlgFractures.cpp", che ritorna vero se la possibilità di intersezione è da escludersi, falso nel caso contrario.
Il processo di esclusione è fatto su quattro livelli, alla fine di ogni livello l'algoritmo ritorna vero o passa al prossimo livello. Al primo livello le due fratture vengono aprossimate come delle sfere centro il baricentro e raggio la distanza massima dei vertici dal baricentro. Nei tre piani successivi invece le fratture vengono aprossimate come dei cerchi centro il baricentro e raggio la distanza massima

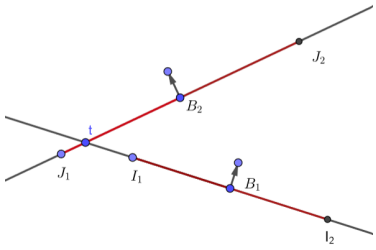

Figure 3: Proiezione dei due cerchi sul piano perpendicolare a t.

dei vertici dal baricentro.
Indichiamo con B_1 e B_2 i centri dei cerchi, R_1 e R_2 i loro raggi, n_1 e n_2 i versori normali alla superficie dei cerchi, t la retta di intersezione tra i piani di f1 e f2.
Al primo livello la possibilità di dintersezione è esclusa se si ha

$$R_1 + R_2 \leq ||B_1 - B_2||$$

Al secondo livello calcoliamo il versore $\hat{t}$ parallelo alla retta t. Vengono poi fatti dei controlli per vedere se le fratture sono parallele, e se sono coplanari. Se sono parallele ma non coplanari, viene esclusa la possibilità di intersezione, altrimenti no e il programma ritorna falso.
Nel caso più generale in cui f1 e f2 non sono parallele, la loro traccia può essere solo sulla retta di intersezione dei loro piani.
Consideriamo i punti

$$I_1 = B_1 + R_1(\hat{t} \times n_1) \qquad I_2 = B_1 - R_1(\hat{t} \times n_1)$$

$$J_1 = B_2 + R_2(\hat{t} \times n_2) \qquad J_2 = B_2 - R_2(\hat{t} \times n_2)$$

Al secondo e terzo livello la verifica che viene fatta, è vedere se le proiezioni $\tilde{I}_1\tilde{I}_2$ e $\tilde{J}_1\tilde{J}_2$ dei segmenti I_1I_2 e J_1J_2 sul piano ortogonale a $\hat{t}$ si intersecano o meno. Nel caso negatico la possibilità di intersezione è esclusa. Per fare questa verifica, si usano ragionamenti simili a quelli usati nell'algoritmo per trovare le tracce:
prima si vede se $\tilde{I}_1$, $\tilde{I}_2$ stanno sullo stesso lato della retta di $\tilde{J}_1\tilde{J}_2$. In caso negativo la possibiltà di intersezione è esclusa, altrimenti ci possiamo calcolare il punto di intersezione H di I_1I_2 con la retta t. I calcoli da fare sono gli stessi del passo tre dell'algoritmo trova-tracce.
A partire da H ci possiamo anche trovare il punto di intersezione della retta J_1J_2 con la retta t. Esso infatti è

$$H' = H + (\hat{t} \cdot (B_2 - B_1))\hat{t}$$

Al terzo livello si controlla se H' appartiene al segmento J_1J_2 o è esterno ad esso. Se è esterno, la possibilità di intersezione è esclusa.
Se si arriva al quarto livello, significa che entrambi i cerchi si intersecano con

la retta di intersezione formata dai loro piani. Questo per il fatto che le loro proiezioni sul piano ortogonale a $\hat{t}$ si intersecano. Le porzioni di t occupati dai cerchi che aprossimano f1 e f2 sono gli intervalli $I_{r_1}(H)$ e $I_{r_2}(H')$ con

$$r_1 = \sqrt{1 - ||H - B_1||^2} \qquad r_2 = \sqrt{1 - ||H' - B_2||^2}$$

Dunque se si ha $r_1 + r_2 \leq |\hat{t} \cdot (B_2 - B_1)|$ la possibilità di intersezione è esclusa. Fatti tutti questi calcoli, ci siamo chiesti, se effettivamente un processo di esclusione così elaborato possa migliorare l'efficienza. La triste risposta è no. Riportiamo nella seguente tabella i tempi di esecuzione dell'algoritmo trova-tracce per i diversi file a disposizione e per i diversi livelli di esclusione. (Livello 0 è nessuna esclusione preliminare, Livello 1 è esclusione mediante aprossimazione con sfere, gli altri due le esclusioni mediante aprossimazione con dischi.)

height	Livello 0	Livell1	Livello 2 e 3	Livello 4	Tracce totali
Fr_10	9	8	10	11	24
Fr_50	188	192	210	212	480
Fr_82	178	9	14	14	1
Fr_200	3911	3280	3857	4190	8979
Fr_362	2597	123	262	245	1

Table 1: tempi di esecuzione per i diversi livelli di esclusione. Test eseguiti su mille prove.

	Livello 1	Livello 2 e 3	Livello 4	Tracce totali
Fr_10	45	34	34	24
Fr_50	1123	655	643	480
Fr_82	1	1	1	1
Fr_200	18905	11898	11746	8979
Fr_362	1	1	1	1

Table 2: Sono riportate il numero di tracce per le quali la possibilità di intersezione non è stata esclusa per i diversi livelli di esclusione.

I dati mostrano che il Livello 4 è allolutamente ridondante, i Livelli 2 e 3 effettivamente escludono un numero non insignificante di tracce, ma i tempi di esecuzione di quest'esclusione sono addirittura maggiori dei tempi di esecuzione dell'algoritmo trova-tracce. Il Livello 1 invece è decisamente un miglioramento rispetto al non fare alcun tipo di controllo.
Anche se l'aprossimazione mediante dischi potrebbe ancora rivelarsi di qualche utilità per fratture con un numero grande di vertici, sicuramente per il caso di fratture a quattro vertici non è da utilizzare.

5 Parte due: Determinare i sotto-poligoni generati per ogni frattura

5.1 Implementazione:

Questa parte di programma, si basa sulla funzione `"Alg_process_cut()"` tale funzione richiede in input un oggetto dfn, in particolare dalla dfn vengono estrapolate tutte le fratture presenti e le relative tracce associate. La funzione `"Alg_process_cut()"` in output darà un vettore di Polygonal Mesh il quale ad ogni posizione contiene la Polygonal Mesh generata dalla relativa frattura. L'oggetto Polygonal Mesh conterrà tutti i sottopoligoni createsi dal processo di taglio delle fratture tramite le proprie tracce, il quale verrà esplicato qui di seguito. All'interno della funzione `"Alg_process_cut()"` appunto si itera su tutte le fratture presenti nella dfn e per ogni frattura viene richiamata la funzione "processFracture()", la quale prendendo in ingresso la dfn e la frattura, ha il compito di trovare le tracce associate alla frattura, distinguerle in passanti o non passanti e infine creare la coppia frattura, vettore di tracce, che sarà di fondamentale importanza nel processo di analisi delle sottofratture dal momento in cui si interromperà solo nel momento in cui il vettore di oggetti tracce risulterà vuoto, a quel punto si procede al riempimento dell'oggetto mesh. Per ogni coppia frattura-vettore di tracce viene chiamata la funzione "cutTrace()", la quale implementa l'algoritmo di taglio delle tracce, in cui avviene effettivamente la creazione delle sottofratture. La funzione "cutTrace()" prende in input la coppia frattura tracce e restituisce la medesima cosa. Per prima cosa viene richiamata la funzione "SortAssociatedTraces()" che si occupa dell'ordinamento delle tracce. Le tracce vengono ordinate al contrario rispetto alle specifiche del progetto per motivi implementativi successivi. Infatti, si itera sulle tracce presenti, si distinguono tra passanti e non passanti e si aggiungono ai relativi vettori di tracce che ci servono come oggetti ausuliari, in modo da ordinare separatamente per lunghezza crescente le due categorie. Una volta ottenuto il vettore di tracce ordinato, si prende l'ultima traccia, che sarà la traccia più lunga passante e viene salvata in un oggetto primaTraccia, dopodichè si elimina quest'ultima dal vettore tramite la funzione `"pop_back()"`. A questo punto inizia il vero processo di taglio. Le nuove coppie (Fracture, vector¡Trace¿) generate da "cutTrace()" vengono aggiunte a un nuovo vettore underFracturesNew. Se sono state create nuove fratture, il ciclo ricomincia analizzando le nuove coppie in underFracturesNew. Se non ci sono più tracce da suddividere, la coppia originale viene aggiunta a underFracturesNew e il ciclo termina. Al termine del ciclo, underFractures viene aggiornato con underFracturesNew e quest'ultimo viene svuotato per la prossima iterazione.

5.2 Algoritmo di taglio e dettagli geometrici

L'algoritmo in principio si occupa di trovare i lati sul quale avviene il taglio. Viene memorizzato un oggetto taglio il quale si salva gli indici dei vertici che delimitano il lato su cui giacciono le coordinate della traccia. Il principio

con il quale si determina se effettivamente la coordinata della traccia appartiene a quel determinato lato della frattura è sostanzialmente un riadattamento dell'algoritmo che si occupa della ricerca delle tracce. Infatti, a livello geometrico, quello che viene fatto è calcolare il vettore che va dalla prima coordinata della traccia al primo vertice della frattura e calcolare il prodotto scalare tra di esso e il vettore normale alla traccia. Una volta fatto ciò si inizia ad iterare sui vertici della frattura, chiaramente saltando il primo e per ognuno dei vertici si calcola il vettore passante per il vertice corrente e la prima coordinata della traccia. A questo punto si moltiplica il prodotto scalare trovato in precedenza con il prodotto scalare tra il nuovo vettore e la normale alla traccia, il punto è che:

• Se il prodotto scalare è positivo, significa che ci troviamo sulla stessa parte di piano, se l'immaginario fosse la frattura come piano e la traccia come retta che divide questo piano in due parti. Questo vuol dire che, se il prodotto è positivo, i due vertici della frattura non hanno "attraversato" la traccia.

• Se il prodotto scalare è negativo, significa che i due vertici si trovano uno da una parte rispetto alla traccia e uno dall'altra quindi significa che nel lato compreso tra il vertice corrente e il successivo è presente la coordinata della traccia.

Viene eseguito lo stesso procedimento per la seconda coordinata ma in questo caso partendo dal vertice successivo a quello memorizzato nell' oggetto taglio. Infine, viene effettuato un controllo per cui se il primo vertice a ritornare sul lato della traccia è il primo vertice stesso, implica che ad essere memorizzato nella posizione 1 dell'oggetto taglio deve essere l'ultimo vertice. Un altro aspetto interessante di questo codice è di come vengono elaborate le tracce non passanti: quello che si intende fare è prolungare la traccia finché non incontri uno dei lati della frattura. Quello che viene fatto è proprio controllare se entrambi gli estremi della traccia intersecano i lati della frattura, nel caso non vi fosse intersezione, l'estremo più vicino della traccia viene spostato lungo la direzione del lato, fino a quando la coordinata non cada esattamente sulla linea contenente il lato. Questo processo viene ripetuto se anche l'altra coordinata non interseca nessun lato. A livello geometrico si calcolano il vettore direzione del lato, e il vettore normale rispetto al lato, il quale viene poi normalizzato. Vengono poi calcolate due distanze: che chiameremo b1 e b2 le distanze perpendicolari create tra la traccia e la linea del lato, positive se sono sullo stesso lato del vettore normale o negative nel caso opposto. Si verifica che b1 e b2 siano diverse da zero e si calcola con le due funzioni della libreria ¡cmath¿ il massimo e il minimo per individuare l'estremo della traccia più distante. Una volta individuato l'estremo della traccia da prolungare tramite l'oggetto daProlungare si procede così di seguito, definiamo:

• a1 la distanza estremo della traccia più vicino al lato e il lato stesso

• a2 il vettore traccia

• A e B i due estremi della traccia, in particolare A è l'estremo più vicino

Il nostro obiettivo è calcolare a1, si sfrutta il fatto che i due triangoli che si creano tramite la proiezione di A e B sono simili

$$a_1 = \frac{(b_1 \cdot a_2)}{b_2 - b_1}$$

Quindi ci andiamo a calcolare il punto di intersezione con il lato:

$$I = A + (A - B) \cdot \frac{b_1}{b_2 - b_1}$$

A questo punto si riassegnano vertici e i rispettivi id alla prima nuova frattura e poi alla seconda e si aggiornano. Infine si passa alla parte delicata ossia al nuovo calcolo delle tracce, infatti le nuove sotto fratture potrebbero presentare delle tracce, tracce che il codice deve individuare e tagliare come descritto in precedenza. Per il calcolo delle nuove tracce il processo è analogo, con qualche riadattamento.

Vengono calcolati due prodotti scalari:

• p1: Prodotto scalare della differenza tra la prima coordinata della traccia t e la prima coordinata della traccia iniziale (primaTraccia) rispetto a un vettore indicatore (indicatore).

• p2: Prodotto scalare della differenza tra la seconda coordinata della traccia t e la prima coordinata della traccia iniziale rispetto allo stesso vettore indicatore.

Si controlla se il segno di p1 è diverso dal segno di p2, allora la traccia t interseca la traccia iniziale. Se c'è intersezione, viene calcolato il punto di intersezione (Intersezione) come una combinazione convessa delle coordinate di t basata sui prodotti scalari p1 e p2. Vengono create due nuove tracce t1 e t2: Entrambe hanno un ID iniziale impostato a 0.

• t1 avrà come prima coordinata il punto di intersezione (Intersezione) e come seconda coordinata la prima coordinata originale di t.

• t2 avrà come prima coordinata il punto di intersezione e come seconda coordinata la seconda coordinata originale di t. Vengono calcolate le lunghezze di entrambe le nuove tracce. Inizialmente, t1 e t2 sono marcate come "non passanti". Si controlla a quale delle due fratture appena create (suddivise dalla traccia iniziale) appartengono le nuove tracce t1 e t2: Se p1 è maggiore di 0, t1 appartiene alla prima frattura e t2 alla seconda. Se p1 è minore o uguale a 0, t1 appartiene alla seconda frattura e t2 alla prima. Per ogni nuova traccia, si verifica se essa interseca tutti i lati della frattura a cui appartiene. Questo viene fatto calcolando il prodotto vettoriale tra la differenza della coordinata della traccia e due vertici consecutivi della frattura. Per ogni coppia di vertici, calcola il prodotto vettoriale tra l'estremità della traccia e i due vertici. Verifica se il prodotto vettoriale è vicino a zero (utilizzando is_equal). Un prodotto vettoriale quasi pari a zero indica che la traccia potrebbe trovarsi esattamente su un bordo della frattura. Se questa condizione vale per tutte le coppie di vertici, la traccia viene contrassegnata come "passante".