RELAZIONE PROGETTO

Andrea Rostagno 295706

Jacopo Ferraris 294292

Daniele Semeraro 284238

Il nostro gruppo si è concentrato sulla prima parte del progetto, che consiste nel leggere un Discrete Fracture Network da file, memorizzarne le fratture e calcolarne di conseguenza le tracce, per poi classificarle in passanti e non passanti, e ordinarle per lunghezza decrescente.

Come prima cosa abbiamo impostato uno switch case: all'utente viene data la possibilità di scegliere quale DFN analizzare, inserendo in input il numero delle fratture, che viene memorizzato in una variabile. Il passo successivo è stato creare una funzione, dal nome *ImportDFN*, che leggesse il file di testo corrispondente alla scelta dell'utente. All'interno della funzione è stata implementata una variabile di tipo 'unsigned int' che fungesse da contatore: il contatore assume valori diversi in base al titolo della riga, e salva di conseguenza i dati nelle apposite variabili. La funzione aggiorna tre contenitori: *FractureId* è un vettore che salva gli identificativi di tutte le fratture, *NumVertices* è un vettore che salva il numero di vertici di ogni frattura, e *ListVertices* è un vettore di matrici, dove ogni matrice contiene le coordinate dei vertici della singola frattura.

Tali oggetti sono stati inseriti all'interno di una structure dedicata appunto alle fratture, in modo da poter organizzare meglio il codice; all'interno di essa si trova anche la seconda funzione che viene invocata, la funzione *sfere*: tale funzione itera su ogni frattura, ne calcola il baricentro e determina il raggio della sfera circoscritta al poligono. Tramite un doppio ciclo, viene verificata l'intersezione tra tutte le coppie di fratture: se la distanza tra i due baricentri è maggiore della somma dei rispettivi raggi, ciò significa che le sfere non si intersecano e di sicuro non creeranno tracce. Gli identificativi delle due fratture prese in considerazione vengono quindi immessi come coppia di 'int' nel vettore *fratturescluse*. La funzione svolge dunque una verifica preliminare importante: questa prima "scrematura" permette di risparmiare successivamente tempo di calcolo e costo computazionale relativi alla risoluzione dei sistemi lineari necessari a individuare le tracce.

Ottenuti tutti i dati riguardanti le fratture, si passa all'effettivo calcolo delle tracce, eseguito tramite l'implementazione di due funzioni, *CalcoloDirezioneTracce* e *CalcoloEstremi*, contenute all'interno di una struct nominata *Traces*, assieme a tutte le unità logiche create per sviluppare al meglio tali funzioni. La prima funzione, tramite un doppio ciclo, confronta ogni frattura con tutte le altre (ovviamente se i due id vengono trovati come coppia all'interno del contenitore *fratturescluse*, si passa direttamente al confronto con la frattura successiva) e determina se esiste un'intersezione tra le due; in caso affermativo calcola la direzione e un punto, condizioni sufficienti per tracciare una retta unica. Le fratture vengono considerati come piani, quindi vengono presi tre vertici di ciascun poligono come punti per individuare i rispettivi vettori normali ai piani (*n1* e *n2*), che vengono resi unitari normalizzandoli. Vengono inoltre calcolati i rispettivi termini noti dell'equazione del piano (*d1* e *d2*) come prodotto scalare tra versore normale e un punto del piano. A questo punto, se i due piani non sono paralleli, si

calcola il vettore direzionale *t* della retta di intersezione; dopodiché si utilizza un sistema lineare Ax=b per trovare un punto *P* che si trova su tale retta. Essendo i piani infiniti, bisogna verificare che la retta trovata (individuata proprio da direzione e punto) intersechi effettivamente due volte il bordo della frattura, iterando su vertici consecutivi del poligono e utilizzando nuovamente un sistema lineare. Se il punto trovato si trova tra i due vertici della frattura in tutte le dimensioni (x,y,z), viene incrementato il contatore *c1*. Se alla fine del ciclo il contatore *c1* ha valore uguale a 2, significa che la retta interseca due volte la prima frattura, e lo stesso procedimento viene ripetuto per la seconda frattura. A questo punto, se anche 'c2 == 2', la retta viene considerata una traccia valida tra le fratture. In tal caso, vengono memorizzate le informazioni della retta e i relativi ID delle fratture: viene incrementata la variabile *NumberOfTraces*, gli identificativi delle due fratture che si intersecano vengono immessi come coppia all'interno del vettore *IDs*, punto *P* e direzione *t* della retta vengono salvate come colonne di una matrice che si inserisce all'interno del vettore di matrici *ListCord*. Questi aggiornamenti consentono di ottenere, dopo l'esecuzione della funzione, un elenco completo delle tracce tra le fratture, incluse le loro posizioni e direzioni.

La seconda funzione, *CalcoloEstremi*, itera su tutte le tracce che sono state individuate: presa una traccia, per ciascuna delle due fratture si adopera un altro ciclo per iterare sui vertici. Preso un vertice, viene utilizzato anche il successivo per delineare il lato corrispondente, e tramite un sistema lineare si trova il punto di intersezione tra la traccia e il lato. Una volta trovato tale punto di intersezione *p*, viene effettuata una verifica per assicurarsi che sia compreso tra le coordinate dei vertici del lato della frattura. Se p è compreso tra i vertici del lato della frattura lungo tutte e tre le dimensioni, allora viene considerato un punto di intersezione valido e viene aggiunto alla matrice delle intersezioni *intersez*. Infine, si ha un ciclo 'for' che scorre attraverso i quattro punti di intersezione memorizzati nella matrice *intersez*. Per ogni punto, verifica se si trova tra gli altri due punti di intersezione con una certa tolleranza. Se la condizione è soddisfatta, il punto viene memorizzato come uno degli estremi della frattura. Il ciclo si assicura di trovare e memorizzare esattamente due estremi, necessari per definire la frattura. Una volta terminato tale procedimento, è possibile stampare su file il risultato di quanto analizzato finora.

L'ultimo passo è costituito dalla funzione *Ordinamento*. Come prima cosa, viene contato il numero di tracce presenti su ogni frattura e lo si stampa ad output in un altro file di testo. Successivamente vengono etichettate come tracce passanti e non passanti tramite la funzione *CalcoloPassante* per poi calcolarne la lunghezza e memorizzarla in uno due vettori distinti, *lungP* e *lungNP*; entrambi i vettori vengono ordinati in maniera decrescente tramite l'algoritmo *BubbleSort* visto in classe. La funzione *CalcoloPassante* prende i primi due vertici di una frattura, controlla che l'estremo della traccia sia in linea con essi e dopodichè verifica che ne sia compreso. Questo procedimento viene ripetuto con ogni coppia di vertici della frattura e su entrambi gli estremi della traccia. Al termine delle due verifiche se ha avuto un riscontro positivo su entrambe, alla variabile *pass* verrà attribuito il valore di '0'(passante) altrimenti '1'(non passante). A questo punto, due cicli distinti stampano su file l'id di ogni traccia, una variabile booleana denominata *Tips* (impostata a 'false' se la traccia è passante, 'true' se non lo è), e la lunghezza: la distinzione dei cicli fa in modo che sul file vengano sovrascritte prima le tracce passanti, e successivamente le non. All'interno di questi cicli, inoltre, è stata posta particolare attenzione sull'assicurarsi di avere la corretta lunghezza di ogni traccia e di non

stampare più volte la stessa traccia, nel caso in cui ce ne fossero molteplici di eguale lunghezza.

La verifica della correttezza del codice è stata fatta tramite i GoogleTest; l'aggiunta di un altro eseguibile all'interno del codice ha permesso di testare tutte le unità logiche implementate, per assicurarsi che memorizzassero tutti i dati e le informazioni in maniera corretta. Per questioni di comodità e velocità, i test sono stati effettuati nel caso del DFN più piccolo, ovvero quello composto da tre fratture, e hanno ottenuto tutti esito positivo.