Progetto di PCS

Bello Renato, Bernandoni Sofia, Chiodo Martina

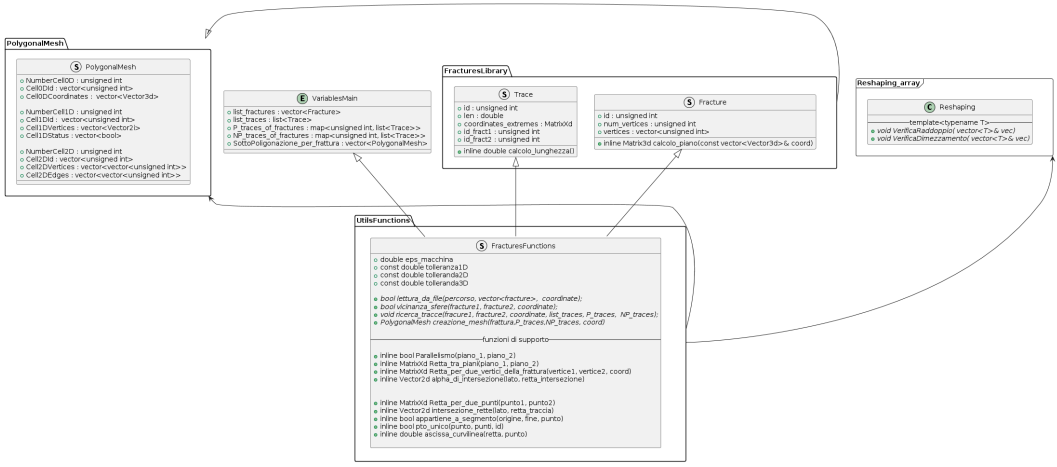
June 14, 2024

Politecnico
di Torino

Table of Contents

Documentazione UML

1 Progetto 1



Strutture Dati

1 Progetto 1

- **Fracture**: raccoglie informazioni sulla frattura, in questo senso è costituita da:
  - *id*, dato di tipo unsigned int;
  - *num_vertici*, dato di tipo unsigned int;
  - *vertices*, dato di tipo vector<unsigned int>;

  A questa struttura è associato il metodo proprio *calcolo_piano*.
- **Trace**: raccoglie informazioni sulle tracce, in tal senso è costituita da:
  - *id*, dato di tipo unsigned int;
  - *len*, dato di tipo double;
  - *id_frc1*, dato di tipo unsigned int;
  - *id_frc2*, dato di tipo unsigned int;
  - *coordinates_extremes*, dato di tipo MatrixXd;

  A questa struttura è associato il metodo proprio *calcolo_lunghezza*.
- **FracturesFunctions**: struttura dati contenente le funzioni usate nel codice, permette di definire la tolleranza per tutte le funzioni.

vicinanza_sfere

1 Progetto 1

vicinanza_sfere è un metodo definito dentro Utils.hpp, fornisce un controllo sulle fratture per capire se queste ultime si intersecano oppure no.
Il metodo prende in input due fratture e ne calcola i baricentri tramite la formula:

$$B = \left(\frac{1}{N} \sum_{i=1}^{N} x_i, \frac{1}{N} \sum_{i=1}^{N} y_i, \frac{1}{N} \sum_{i=1}^{N} z_i \right) \tag{1}$$

Successivamente calcola il raggio del cerchio che circoscrive la frattura, risolvendo il problema:

$$R = max\{R|\ R = \|B - p_i\|_2 \text{ dove } p_i \text{ è un vertice della frattura}\} \tag{2}$$

Ed in fine confronta i raggi trovati per entrambe le fratture. Se la somma dei raggi è minore della distanza allora le due fratture non si intersecano e possiamo quindi non fare conti su di esse.

ricerca_tracce

1 Progetto 1

ricerca_tracce è un metodo contenuto dentro Utils.hpp, calcola l'intersezione tra due fratture e classifica le tracce come passanti o non passanti.
Il metodo prende in input due fratture e calcola la i piani dove esse giacciono:

$$\pi = \left\{ \underline{x} \in \mathbb{R}^3 \middle| \, \underline{x} = P_0 + \alpha_0 \underline{v} + \alpha_1 \underline{u}, \ \alpha_0, \alpha_1 \in \mathbb{R} \right\} \tag{3}$$

$$\pi' = \left\{ \underline{x} \in \mathbb{R}^3 \middle| \, \underline{x} = P'_0 + \alpha_0 \underline{v'} + \alpha_1 \underline{u'}, \ \alpha_0, \alpha_1 \in \mathbb{R} \right\} \tag{4}$$

Successivamente ricava la retta di intersezione tra i due piani:

$$r = \left\{ \underline{x} \in \mathbb{R}^3 \middle| \, \underline{x} = P + \alpha \underline{t}, \ \alpha \in \mathbb{R} \right\} \tag{5}$$

Ed in fine ciclando sui vertici delle due fratture possiamo trovare i punti di intersezione tra la retta (5) e le fratture. I punti di intersezione sono 4 e sono tutti disposti lungo la retta (5), in base all'appartenenza (se alla frattura 1 o 2) dei due punti più interni possiamo calcolare la traccia e classificarla.

funzionamento ricerca_tracce

1 Progetto 1

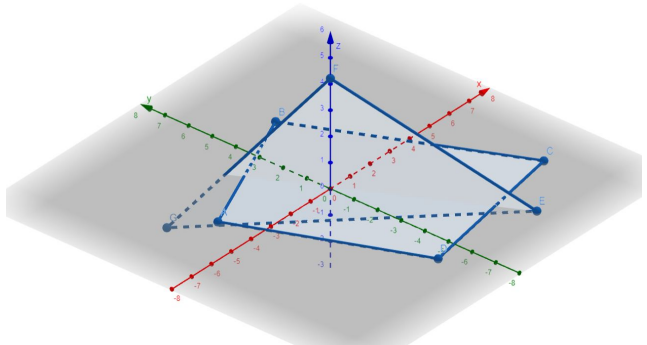


Figure: Le due fratture

funzionamento ricerca_tracce

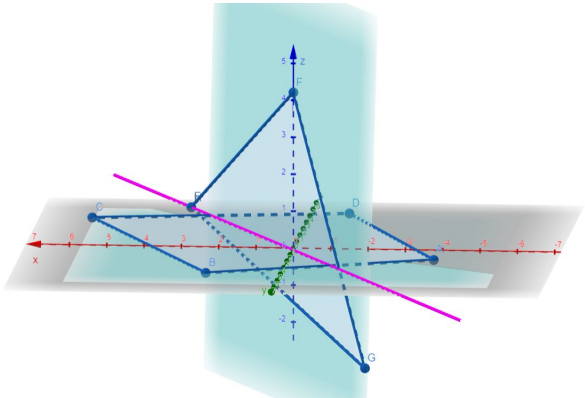


Figure: I due piani e la retta di intersezione

funzionamento ricerca_tracce

1 Progetto 1

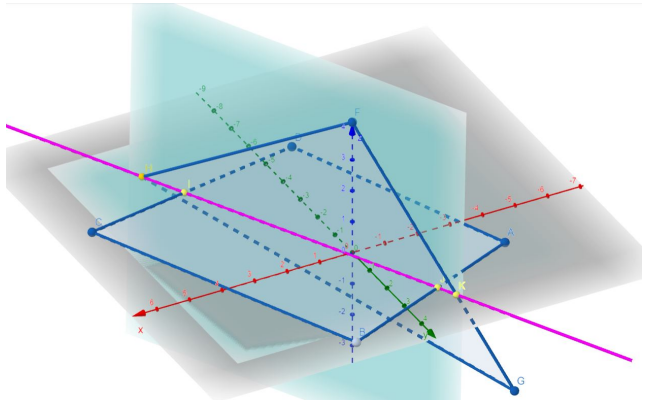


Figure: I punti β sulla retta

funzionamento ricerca_tracce

1 Progetto 1

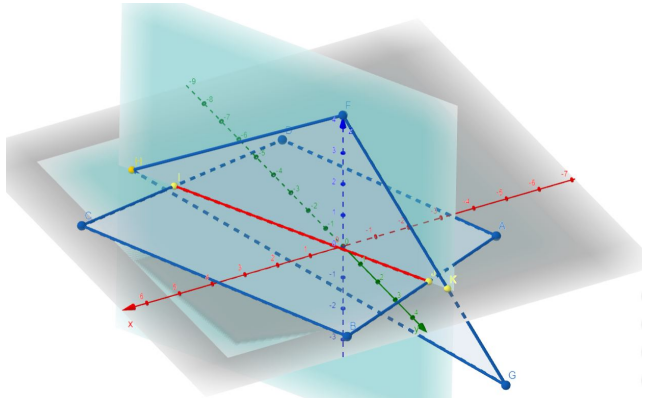


Figure: la traccia

Table of Contents

Ricorsione

2 Progetto 2

Per risolvere il secondo progetto abbiamo lavorato ricorsivamente definendo due funzioni:

- **creazione_mesh**: definita dentro *FracturesFunctions*, serve ad inizializzare la mesh e, nel caso in cui la frattura abbia delle traccie interne, chiama la seconda funzione;
- **divisione_sottopol**: funzione che si occupa di dividere la frattura padre in due fratture figlie, tagliando con una traccia. Costruisce, quindi, le due nuove mesh e riassegna le tracce a ciascuna frattura figlia. Questa funzione viene chiamata ricorsivamente, la condizione di arresto si ha quando la frattura figlia non ha più tracce all'interno.

Export in ParaView per DFN3

2 Progetto 2

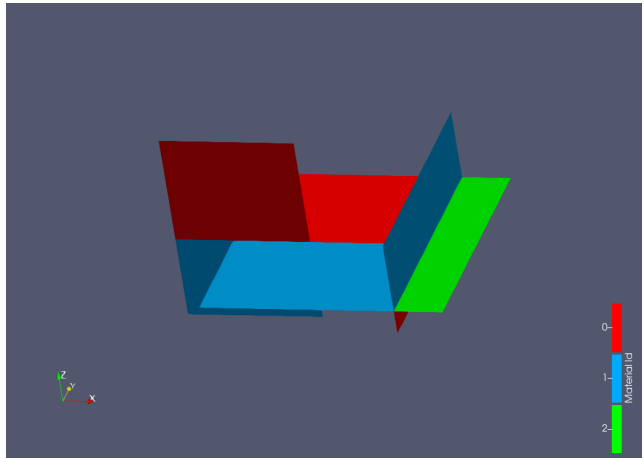


Figure: export in ParaView di DFN3

Export in ParaView per DFN10

2 Progetto 2

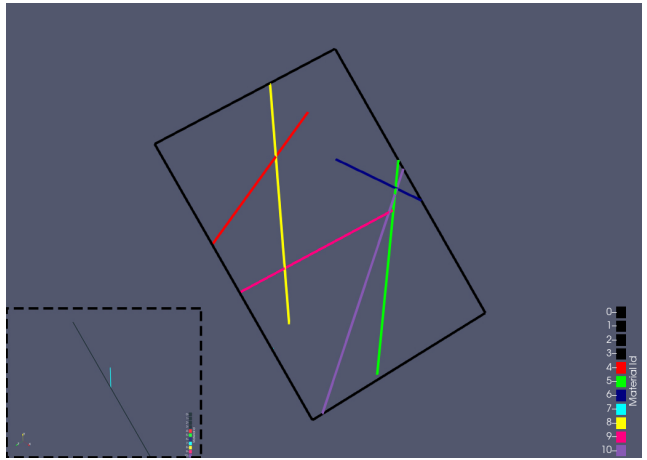


Figure: export in ParaView per una frattura di DFN10, prima del taglio con le tracce

Export in ParaView per DFN10

2 Progetto 2

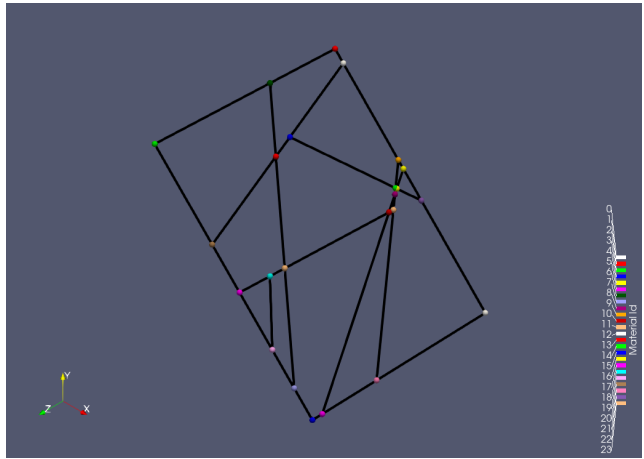


Figure: frattura una volta tagliata

Export in ParaView per DFN200

2 Progetto 2

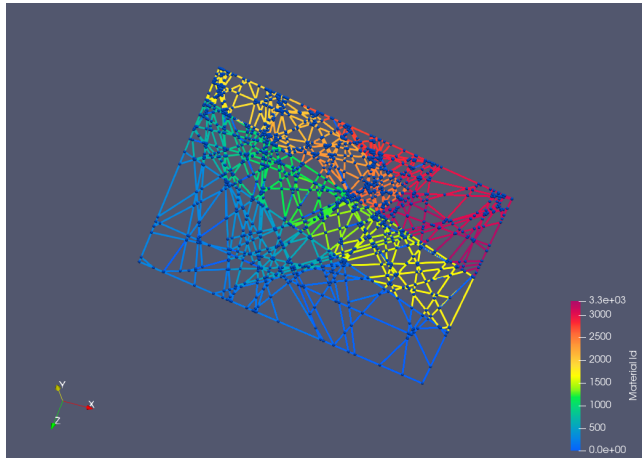


Figure: la frattura 150 con i relativi tagli

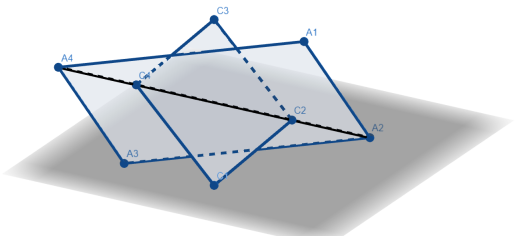
Table of Contents

Test

3 Test

Abbiamo cercato di fare almeno due test per ogni funzione, concentrandoci sulle funzioni più complesse o con più casi limite.

- Per quanto riguarda il primo progetto, abbiamo testato la funzione vicinanza_sfere nel caso in cui le sfere siano tangenti o coincidenti;
- Abbiamo testato il primo progetto su due fratture particolari, ovvero delle fratture nel quale la retta di intersezione tra i piani passa attraverso 4 vertici in totale:



- Abbiamo testato la funzione *ricerca_tracce* nel caso in cui una traccia abbia lunghezza nulla;
- Nel secondo progetto abbiamo testato la funzione *Retta_per_due_punti* nel caso in cui i punti passati in input siano uguali;
- Abbiamo testato la funzione *Appartiene_a_segmento* quanto i punti in ingresso sono l'origine o la fine;
- Abbiamo verificato che la funzione *creazione_mesh* funzionasse correttamente sia nel caso in cui una traccia appartenga a due sotto fratture, sia nel caso in cui un estremi di una traccia coincida con un vertice della frattura;

In totale 32 test.