Discrete Fracture Network

Relazione

Flavio Gasparini 294531
Rosita Tozzi 293598

Matematica per l'ingegneria
Programmazione e calcolo scientifico

Anno Accademico: 2023-2024

1 Classi utilizzate

Sono state create tre classi, *Frattura*, *Traccia* e *MeshPoligonale*, per rendere più agevole la risoluzione del problema e la manipolazione dei dati.

1.1 Frattura

La classe Frattura, rappresentante l'oggetto geometrico frattura, ha un insieme di attributi e metodi visibili nella documentazione UML.
In particolare i vettori *TracceNoPass* e *TraccePass* contengono gli Id delle tracce appartenenti alla Frattura. Abbiamo scelto di utilizzare un vettori di interi, al posto di un vettore di Traccia, per evitare di avere copie dell'oggetto Traccia salvate in memoria. Il recupero delle informazioni relative ad ogni traccia è possibile accedendo tramite l'Id ad un vettore contente tutte le tracce determinate nel progetto, tenendo conto che la posizione nel vettore coincide con l'Id stesso. Abbiamo valutato essere la scelta ottimale essendo l'accesso diretto in un vettore efficiente. I due attributi *VecNormale* e *termineNotoPiano* servono invece a definire il piano in cui giace la frattura. È stato scelto di inserirli come attributi in modo tale da non doverli calcolare più volte. La scelta di utilizzare vettori di *Eigen* per le coordinate dei vertici è motivata dalla necessità di effettuare operazioni matematiche su di esse. Infine ogni oggetto Frattura ha come attributo un oggetto di tipo MeshPoligonale, inserito per rendere più agevole lo svolgimento della seconda parte del progetto. Inoltre il vettore di interi *IdVertici* contiene gli Id delle Cell0D che costituiscono i vertici della frattura. Tale vettore serve a distinguere i vertici dalle altre Celle0D della mesh.

1.2 Traccia

La classe Traccia contiene tutte le informazioni relative all'intersezione tra due fratture. Gli attributi e i metodi sono riportati nella documentazione UML.
Anche in questo caso per memorizzare gli estremi della traccia sono stati utilizzati dei vettori di *Eigen*, in modo da poter agevolare le operazioni matematiche. Sono stati utilizzati degli array di STL per salvare i vertici di una traccia, le fratture che la generano e l'informazione passante/non passante (*VerticiTraccia*, *FrattureTraccia* e *Tips*), in quanto sappiamo che la loro lunghezza è sempre pari a 2. L' array *FrattureTraccia* salva solo gli id delle fratture, in modo tale da non avere lo stesso oggetto Frattura salvato più volte in memoria. Anche in questo caso tutte le informazioni sulla frattura possono essere recuperate accedendo ad un vettore di oggetti Frattura tramite l'Id. Per ogni traccia viene inoltre salvata anche la lunghezza,in modo da poter accedere direttamente al suo valore durante l'ordinamento, senza andarlo a calcolare ogni volta.

1.3 MeshPoligonale

Questa classe rappresenta una struttura dati per memorizzare una mesh poligonale. La mesh è suddivisa in tre tipi di celle: celle 0D (nodi), celle 1D (lati) e celle 2D (poligoni). Gli attributi nello specifico sono presenti nella documentazione UML. Per le coordinate dei nodi sono state memorizzate anche in questo caso in dei vettori di *Eigen*. Per le celle 1D, si utilizzano array di STL per rappresentare le coppie di nodi che formano i lati, poiché la dimensione è nota a priori. Per le celle 2D, dove la dimensione dei vertici e dei lati non è nota a priori, si utilizzano vettori di STL per memorizzare gli ID dei vertici e dei lati che compongono i poligoni. Ogni oggetto Frattura ha come attributo una mesh poligonale, generata a partire dalle sue tracce.

2 Scelta Tolleranza

Per lo svolgimento di tutto il progetto è stata necessaria l'introduzione di una tolleranza, lavorando in precisione finita. Per farlo abbiamo implementato due funzioni *setTol1D()* e *setTolProdotto()*. La prima chiede in input all'utente una tolleranza e restituisce il massimo tra essa e la tolleranza di default. La seconda invece restituisce il massimo tra la tolleranza di default e il prodotto della tolleranza restituita dalla funzione precedente per se stessa. La sua introduzione è stata necessaria per poter effettuare confronti di lunghezze al quadrato. Per evitare infatti il calcolo della radice quadrata, essendo quest'ultimo oneroso, tutte le distanze sono state calcolate al quadrato con la necessaria introduzione di una *"tolleranza al quadrato"*. Ad esempio, per confrontare se due punti coincidono, abbiamo usato il comando *squaredNorm()* della libreria *Eigen*. Le funzioni di tolleranza sopra elencate vengono chiamate una sola volta all'inizio del programma; dopodiché il loro valore viene salvato ed utilizzato in tutte le funzioni successive, eliminando il costo di chiamata.

3 Progetto 1

3.1 Concetto generale per la risoluzione del progetto

Con lo scopo di individuare, salvare e stampare le tracce che si formano dall'intersezione tra più fratture, abbiamo realizzato un algoritmo composto da varie funzioni in modo tale da adempiere a questo compito nel modo più ottimale possibile. Il programma legge le fratture date in input tramite file e le salva. Successivamente va a lavorare con coppie di frattura per stabilire se tra di esse esiste un'intersezione ed eventualmente le immagazzina. Infine effettua le varie stampe e ordinamenti richiesti.

3.2 Risoluzione e funzioni implementate

3.2.1 Importo fratture

Per prima cosa è stata effettuata la lettura da file e sono state salvate le fratture individuate. Per fare ciò è stata implementata la funzione *importoFratture()*. Essa prende come input il nome del file e, leggendo riga per riga, salva le informazioni riguardanti le singole fratture in oggetti di tipo Frattura e li memorizza in un vettore. La scelta dell'utilizzo di un vettore è stata dettata dal fatto che nella prima riga del file è indicato il numero di fratture totali e quindi, una volta nota questa informazioni, è possibile effettuare il *resize* del vettore senza il rischio di dover allocare ulteriore memoria in seguito. Prima di salvare ogni frattura è stato effettuato anche un controllo per verificare se i vertici fossero sovrapposti. Tale operazione è stata fatta tramite la funzione *testLunghezzaLati()* che controlla se la distanza tra due vertici è minore della tolleranza. Se la frattura non passa questo test viene scartata.

3.2.2 Controllo paralleli

Dopo aver salvato le fratture sono state individuate le possibili tracce tra le fratture. Sono stati inseriti alcuni controlli per poter scartare alcune coppie di fratture con determinate caratteristiche evitando di fare dei calcoli superflui ed ottimizzando il programma.
Per prima cosa abbiamo verificato che i due piani contenenti le fratture non fossero paralleli, perché altrimenti non ci sarebbe potuta essere intersezione. Per farlo abbiamo calcolato il prodotto vettoriale tra le normali ai piani e controllato che fosse maggiore della tolleranza. In caso contrario, essendo i due piani paralleli, le due fratture sono scartate.

3.2.3 Controllo del Centroide

Una volta stabilito che i due piani non sono paralleli, non siamo ancora certi che ci sia intersezione tra le fratture. Abbiamo effettuato infatti un altro controllo denominato tecnica del *Centroide*. Esso permette di scartare le coppie di poligoni troppo distanti fra loro per intersecarsi .
Per farlo abbiamo implementato una funzione chiamata *ControlloCentroide()*. Per prima cosa è stato individuato il punto medio di ogni poligono calcolando la media dei vertici. Dopodiché è stata calcolata la massima distanza fra il punto medio e i vertici. Costruendo una sfera avente come centro il punto medio e come raggio la massima distanza, il poligono sarà sicuramente contenuto nella sfera. Perciò è stato controllato che la distanza al quadrato fra i due punti medi, ovvero i centri delle due sfere, fosse minore della somma dei raggi, elevata al quadrato, in modo da determinare se la vicinanza dei poligoni era tale da garantire un'intersezione.

3.2.4 Individuazione del lato del poligono intersecante

Superati i primi due controlli siamo andati ad individuare i punti di intersezione tra i lati di una frattura e il piano contenente l'altra. Ovviamente il controllo è stato eseguito per entrambe le fratture. Per trovare l'intersezione è stata effettuata inizialmente una scrematura dei lati da andare a controllare. Non sapendo infatti il numero dei lati del poligono, non è conveniente risolvere un sistema lineare per ogni lato, dato il loro costo computazionale elevato. Sono perciò stati eseguiti dei controlli che utilizzano il prodotto scalare per individuare la coppia di vertici che definisce il lato intersecante.
Per farlo abbiamo implementato la funzione *SiIntersecano()*. Essa prende in input le due fratture (dove la prima è quella di cui si vogliono calcolare le intersezioni e la seconda quella di cui si considera il piano), un array che andrà a contenere le coordinate dei punti di intersezione, la tolleranza ed un booleano che segnala se un lato della frattura appartiene al piano. L'algoritmo prevede la sostituzione dei vertici della prima frattura nell'equazione del piano della seconda per studiarne il segno. Infatti sappiamo che se un punto appartiene al piano soddisfa l'equazione del piano stesso, altrimenti la sostituzione restituisce un valore maggiore o minore di zero in base a se si trova nel semispazio positivo e negativo. Di conseguenza il lato che interseca il piano sarà quello i cui vertici si trovano in due semispazi diversi.
Per semplicità in questo paragrafo useremo valore zero o diverso da zero, sottintendendo che stiamo lavorando in precisione finita e l'uso di una tolleranza.
Per ogni vertice della frattura viene studiato il valore restituito dopo averlo sostituito nell'equazione del piano. Se esso è nullo vengono controllati il punto precedente e il punto successivo per verificare se è solo un punto a toccare il piano o un intero lato. Nel caso sia solo un punto viene ulteriormente controllato che non sia solo un punto a toccare l'altra frattura ma ci sia stato un effettivo passaggio oltre il piano.Nel caso non ci sia, la funzione restituisce falso a segnalare che non si genera nessuna traccia. Altrimenti è stato trovato un lato intersecante e vengono memorizzate le coordinate del vertice sul piano e di quello successivo.

Se il valore restituito dall'equazione del piano è diverso da zero, viene confrontato con il valore restituito dal vertice precedente. Se i due valori hanno segno discorde vuol dire che i vertici si trovano da parti opposti del piano e ho trovato gli estremi del lato intersecante. L'algoritmo restituisce sempre 4 vertici che individuano i due lati della frattura che intersecano il piano, accertandosi sempre di salvare due lati distinti. I diversi casi sono riportati nella Figura 1.

3.2.5 Calcolo della retta contenente la traccia

Il calcolo della retta sulla quale giace la traccia è implementato direttamente nella funzione che calcola le tracce, denominata appunto *CalcoloTracce()*. Tale scelta è dovuta al fatto che risultava

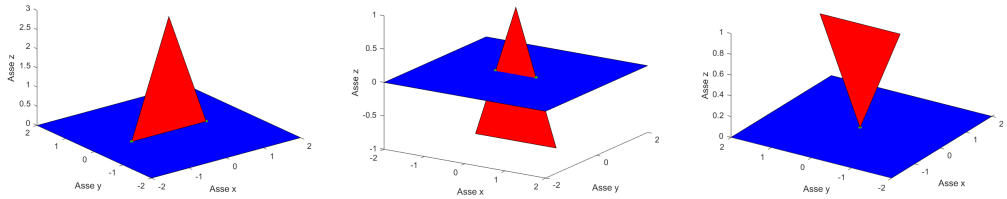

Figura 1: Nella prima figura un lato appartiene al piano, nella seconda la frattura passa attraverso il piano e nell'ultima l'intersezione è solo un punto.

poco conveniente chiamare più volte una funzione che conteneva solo alcune righe di codice. La direzione della retta viene determinata calcolando il prodotto vettoriale tra le normali dei piani contenenti le fratture. Per poter scrivere l'equazione della retta è necessario anche un punto della retta stessa, che viene determinato andando a risolvere un sistema lineare tra le equazioni dei piani e l'equazione della retta con termine noto nullo. Essendo il sistema lineare quadrato viene risolto tramite la decomposizione PA=LU.

3.2.6 Calcolo dei punti di intersezione

Per trovare gli eventuali punti di intersezione tra le due fratture sono state utilizzati i lati trovati tramite la funzione *SiIntersecano()* precedentemente descritta e la retta di intersezione tra i due piani di cui sopra. Per prima cosa è stata individuata la retta contenente un lato calcolando la sua direzione e utilizzando come punto uno dei due vertici. Dopodiché, per trovare il punto di intersezione tra le due rette, è stato risolto un sistema lineare 3X2, avente come colonne le due direttrici delle rette e come termine noto la sottrazione di due punti distinti appartenenti ognuno ad una retta. Il sistema è stato risolto usando una scomposizione QR, essendo il sistema lineare non quadrato. Tutto ciò è stato implementato nella funzione *IncontroTraRette()*. Una volta trovati questi punti sono stati passati alla funzione che calcolava gli estremi della traccia. Entrambe le funzioni sono richiamate all'interno della funzione *CalcoloTracce()*.

3.2.7 Individuazione estremi traccia

Una volta calcolati i 4 punti di intersezione, siamo passati a determinare gli estremi della traccia. Si noti che a questo punto non siamo ancora certi che le due fratture si intersecano, in quanto potrebbero essere vicine, ma non al punto da intersecarsi, oppure toccarsi in un unico punto. Siamo partiti dai punti di intersezione tra i lati delle fratture e la retta su cui giace la traccia e abbiamo implementato la funzione *EstremiTraccia()* che, come vedremo, sfrutta il fatto che i punti sono allineati. Per prima cosa abbiamo studiato le varie casistiche che potevano verificarsi:

- *Due punti distinti* Nel caso in cui la traccia risulti passante per entrambe le frattura allora ci sono solo due punti distinti.
- *Tre punti distinti* In alcuni casi due dei quattro punti di intersezione coincidono. Tale punto è un estremo della traccia.
- *Tutti punti distinti* Nel caso più generico tutti i punti sono distinti.

Prima di richiamare la funzione *EstremiTraccia()* abbiamo effettuato dei controlli per poter verificare se si trattava del primo caso. Superato questo controllo la funzione agisce in maniera

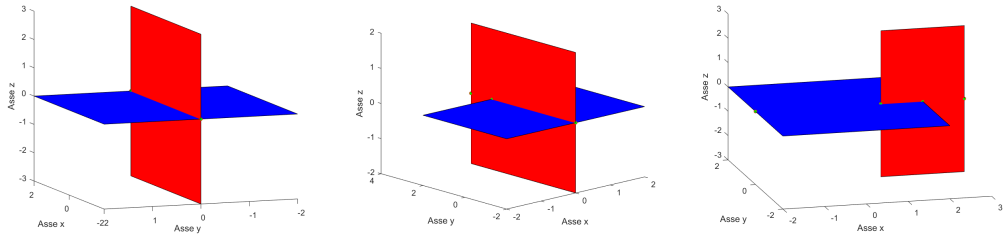

Figura 2: Nella prima figura ci sono solo due punti distinti, nella seconda tre e nell'ultima tutti.

differente in base a se ci sono 3 o 4 punti distinti. Nel caso di 3 punti a partire dal punto in comune si determina l'altro estremo andando a calcolare le distanze dei punti rimanenti dal punto selezionato e prendendo quello con distanza minima. Se il secondo estremo appartiene alla prima frattura, la traccia risulta passante per la prima e non passante per la seconda, se appartiene alla seconda frattura viceversa. Nel caso in cui tutti i punti sono tutti distinti, abbiamo individuato gli estremi della traccia ordinando i punti e prendendo quelli centrali. Per farlo abbiamo sfruttato il prodotto scalare, tenendo conto del fatto che i punti sono allineati e che quelli centrali hanno due punti punti alla loro destra o sinistra. Per determinare la natura della traccia abbiamo notato che se i due estremi appartengono entrambi allo stesso poligono è passante per esso e non passante per l'altro. Se invece i due estremi appartengono a due fratture diverse, la traccia risulta non passante per entrambe. In entrambi i casi prima di salvare la frattura siamo andati a determinare se si trattava di una finta intersezione, ovvero i casi riportati nella Figura 3. Nel caso di tre punti

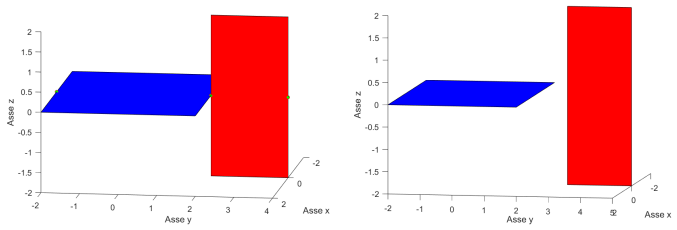

Figura 3: Casi scartati.

distinti è bastato controllare, sempre mediante il prodotto scalare, che gli altri due punti si trovano da lati opposti rispetto all'estremo della traccia già individuato. Nel caso di tutti punti distinti abbiamo controllato che, una volta ordinati, non si trovassero prima i due punti di una frattura e poi quelli dell'altra.
Per poter ottimizzare l'algoritmo e diminuire i tempi di esecuzione nel caso di tutti punti distinti abbiamo deciso di effettuare il controllo con il prodotto scalare solo per i primi tre punti ed eventualmente selezionato il quarto per esclusione. Sempre all'interno della funzione *CalcolaTracce()* le tracce trovate vengono salvate e memorizzate nei rispettivi contenitori scelti in precedenza.

3.2.8 Algoritmo di Ordinamento MergeSort

La funzione *MergeSort()* utilizzata è stato un adattamento del generico algoritmo MergeSort. La nostra implementazione ordina gli oggetti di tipo traccia in base al loro attributo *Lunghezza*.

Abbiamo scelto di utilizzare l'algoritmo MergeSort perché il suo costo computazionale è sempre $O(n \log n)$, che è il minimo costo per gli algoritmi di ordinamento.

3.2.9 Stampa su file

Per la stampa su file sono state implementate due funzioni *stampaTracce()* e *stampaTracceFratture()* che prendono in input rispettivamente il vettore di tutte le tracce e il vettore di tutte le fratture. La prima funzione esegue semplicemente la stampa seguendo l'ordine di inserimento delle tracce nel vettore. Nel secondo caso invece per ogni frattura viene ordinato il vettore delle tracce passanti e non passanti, tramite *MergeSort()* e viene eseguita la stampa.

4 Progetto 2

4.1 Concetto generale

A partire dalle fratture e tracce individuate nel progetto 1, abbiamo proseguito con la creazione di una mesh per ogni frattura. Per farlo abbiamo implementato una serie di funzioni che permettono di tagliare la frattura lungo la direzione della traccia e di individuare gli elementi che andranno a costituire la mesh. Abbiamo deciso di trattare i sotto poligoni che si originano dal taglio di una frattura come delle nuove fratture, ognuna con le proprie tracce. Prima di iniziare il taglio di una frattura i suoi vertici vengono convertiti in Cell0D ed inseriti all'interno della mesh.

4.2 Risoluzione e funzioni implementate

4.2.1 Funzione ricorsiva taglia

Avendo deciso di interpretare i sotto poligoni come delle fratture, abbiamo implementato una funzione ricorsiva *Taglia()* per il taglio lungo le tracce. La funzione in questione prende come input la frattura sulla quale si vogliono effettuare i tagli, la frattura madre e la tolleranza. La frattura madre rappresenta la frattura della quale vogliamo costruire la mesh e coincide con la prima frattura passata come input se non sono mai stati effettuati tagli.
Nella funzione vengono analizzati i casi in cui la frattura ha delle tracce o meno. Nel primo caso, distinguiamo se ci sono delle tracce passanti o sono solo non passanti. In base alla tipologia viene chiamata una funzione che va ad individuare le sotto fratture. In entrambi i casi il taglio viene effettuato lungo la prima traccia disponibile, tenendo conto che esse sono ordinate come spiegato in precedenza. Tale funzione, di cui parleremo in seguito, restituisce le due fratture figlie, sulle quali viene richiamata la funzione *Taglia()*. Se la frattura non ha tracce, può essere convertita in una cella 2D della mesh della frattura madre. A questo scopo viene chiamata una funzione *converteInCelle()*. L'algoritmo termina nel momento non ci sono più sottofratture da tagliare perché sono state tutte convertite in celle ed è stata costruita la mesh. Le sottofratture che vengono create durante il taglio sono memorizzate nello stack frame e vengono rimosse quando la chiamata a funzione su quella specifica sottofrattura termina.

4.2.2 Calcolo sottopoligoni

Le funzioni *Calcolo sotto poligoni* sono state create per riuscire a suddividere la frattura in due sottofratture distinte, prolungando se opportuno, la traccia che andiamo a considerare. Sono state ideate due diverse funzioni che si occupano di gestire in maniera distinta le tracce passanti e non passanti. Entrambe le funzioni utilizzano come concetto quello di definire all'interno della frattura

due semipiani, separati dalla traccia con cui attuiamo la suddivisione. Le funzioni si occupano di assegnare ad ogni sottofrattura (identificata con il semipiano che la contiene) i vertici e tutte le altre tracce che lo compongono.
Per determinare in quale semipiano si trovi il vertice viene preso come riferimento il valore (salvato nella variabile *segno*) del prodotto scalare tra la normale al piano della frattura e il vettore generato dal prodotto vettoriale tra il vettore direzione della traccia e il vettore che congiunge un vertice della traccia con il vertice della frattura in questione. Infatti il prodotto vettoriale individua un vettore perpendicolare al piano della frattura, che ha una direzione sopra o sotto il piano a seconda del verso dei due vettori. Essendo il vettore della traccia costante, il verso del vettore del prodotto è deciso solamente dalla posizione del vertice nella frattura. Il prodotto scalare con il vettore del piano permette di trasformare questa informazione in un numero con un segno, gestibile dall'algoritmo.
Se il valore della variabile *segno* assume un valore nullo (ovvero minore della tolleranza scelta) possono esserci diverse cause, tra le quali la lunghezza della traccia nulla, coincidenza dei vertici della traccia e della frattura oppure il parallelismo tra il vettore della traccia e il vettore congiungente i vertici della traccia e della frattura. Il primo caso viene trattato controllando prima dell'inserimento se la lunghezza delle tracce è minore della tolleranza. Il secondo caso e il terzo caso vengono invece controllato prima di calcolare il valore della variabile *segno*, assegnando il vertice della frattura ad entrambi i semipiani. Dopodiché durante l'algoritmo il valore della variabile *segno* sarà positivo o negativo. L'algoritmo confronta i segni del valore *segno* corrente con il valore del vertice precedente controllando se è avvenuto un passaggio da un semipiano all'altro. Nel caso di segni discordi, la funzione che si occupa delle tracce passanti sceglie come punto intermedio tra i due semispazi il vertice della traccia giacente sul lato tra il vertice attuale e al precedente; questo è dovuto al fatto che il punto di intersezione tra traccia e lato è dato dal vertice di una traccia. Nel caso di tracce non passanti viene invece calcolato il punto facendo l'intersezione tra la retta della traccia e il lato formato dal vertice considerato e il precedente. Se il valore di *segno* rimane invece uguale viene semplicemente aggiunto il vertice della frattura al semipiano del vertice precedente. Questi controlli vengono effettuati fino a quando non si inseriscono due punti di passaggio tra i semipiani; dopodiché i vertici vengono inseriti nel semipiano del segno dell'ultimo vertice. Alla fine del controllo su tutti i vertici, se i punti di passaggio sono minori di due, viene controllato se c'è un cambio di segno tra il primo vertice e l'ultimo.
Il salvataggio dei vertici delle due sottofratture avviene inserendo in ordine di lettura i vertici della frattura in due distinti vettori a seconda del semipiano considerato. Parallelamente avviene anche il salvataggio di ogni Id dei vertici salvati. Durante il salvataggio di punti che non sono vertici della frattura viene chiamata la funzione *RicercaVerticeId()*. Essa scorre il vettore delle Cella0D della FratturaMadre e va controllare se il punto che andiamo inserire è stato già registrato con un ID precedentemente oppure è necessario creare una nuova Cella0D in cui salvare il vertice. In questo modo, dopo aver tagliato la frattura lungo tutte le sue tracce, ogni sottopoligono avrà i propri vertici con un Id ben definito e univoco a seconda della sua posizione.
L'algoritmo tratta anche il caso degenere, ovvero quello di avere una traccia appartenente ad un lato. Il taglio di questa frattura non crea due frattura ma aumenta semplicemente il numero di vertici del poligono che andiamo ad analizzare. Il caso è stato gestito inserendo un booleano che indica che il taglio ha prodotto una sola sottofrattura. Per verificare se la traccia si trovi sul lato viene controllato se, quando il valore della variabile *segno* è minore della tolleranza, il lato giace sul lato compreso tra il vertice precedente e il vertice corrente. In caso positivo vengono salvati i nuovi vertici della frattura e il booleano *TracciaSUlBordo* assume valore *true*.
Dopo aver effettuato tutti i controlli e riempito i vettori che rappresentano i semipiani vengono calcolate le posizioni delle tracce rimanenti. Effettuando un procedimento simile a quello precedente con un valore *segnoTracciaVertice*, viene calcolato a quale semipiano appartengono i vertici della

traccia.Se la traccia è tutta contenuta in un semipiano, allora l'indice che la identifica viene inserito nel vettore corrispondente al semipiano. Se invece i vertici appartengono a semipiani diversi, allora viene calcolato il punto di intersezione tra la traccia presso cui viene effettuato il taglio e la traccia di cui stiamo considerando la posizione. Per queste due nuove tracce viene controllato che la loro lunghezza non sia minore della tolleranza e vengono inserite nel vettore di Tracce totali. Una volta terminata la creazione di una mesh su una frattura tutte le nuove tracce che sono state create vengono eliminate, non essendo più utilizzate, per liberare memoria. Dopo questi controlli vengono salvati le posizioni delle traccia in due vettori distinti, a seconda del semipiano a cui essi appartengono.
Infine vengono create le due fratture, associando i vertici e le tracce alla sottofrattura del semipiano corrispondente. Le due fratture vengono restituite tramite un vettore di oggetti *Frattura*.

4.2.3 Convertire in celle

La funzione *converteInCelle()* prende come input la frattura da convertire e la frattura madre. Essa converte i lati della frattura in Celle 1D e la frattura stessa in una Cella 2D.I vertici non vengono convertiti in quanto sono già stati inseriti in precedenza nella funzione che calcola i sotto poligoni. Prima di assegnare un Id ad un lato viene effettuato un controllo per verificare se nella mesh è già presente un lato con quegli estremi. Ciò può verificarsi siccome la traccia lungo la quale viene effettuato il taglio diventa un lato di entrambe le fratture figlie. Una volta memorizzati tutti i lati, viene salvata la cella 2D.

4.2.4 Stampa su file

Per la stampa su file è stata implementata una funzione *stampaMesh()* che prende in input il vettore di tutte le fratture. Per ognuna di esse vengono stampate tutte le informazioni sulla mesh contenuto nell'attributo *SottoPoligoni*.

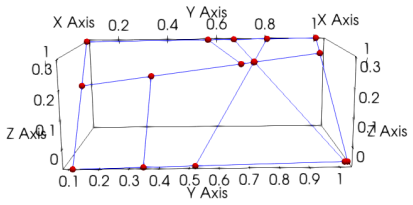

Figura 4: Mesh realizzata con Paraview

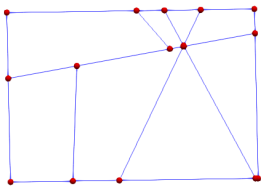

Figura 5: Mesh vista dall'alto realizzata con Paraview