Tetraedrizzazione 3D su C++

# Punto 1.

Per ciascuna frattura individuare l'elenco dei tetraedri tagliati dalla frattura. (Se la frattura tocca il tetraedro in un vertice, un lato o ricopre una intera faccia il tetraedro non si intende tagliato. Se la frattura ricopre parte di una faccia o lato invece o se ha un vertice all'interno di un lato o faccia del tetraedro, questo e da ritenersi tagliato).

1. Per prima cosa è chiamata la funzione **input\_file\_Tetraedrizzazione** che apre i file e salva tutti i dati in variabili globali come nel 2D.
2. Poi è creata la struttura globale touch, con la stessa struttura di quella creata per il 2D, attraverso la funzione **touch\_3D.** Per ogni nodo sono quindi savati

* I LATI per cui esso è uno degli estremi
* Le FACCE per cui è uno dei vertici
* I NODI ad esso collegato tramite qualche arco
* I TETRAEDRI che lo posseggono come uno dei quattro nodi.

Questa struttura ci sarà molto utile successivamente.

1. Si creano due figure grazie alla funzione **Draw\_3D**, una per la visualizzazione 3D dei tetraedri e del poligono e una, in due dimensioni, utilizzata successivamente per la sottopoligonizzazione della frattura (vedi punti successivi).
2. Si apre il ciclo i che scorre i poligoni e si inizializza la struttura dove saranno salvate le intersezioni fra la frattura e i vari poligoni:

la struttura global si chiama **intersect\_tetr**, intersect\_tetr(i) si riferisce alla i-esima frattura. intersect\_tetr(i).array(m) analizza il m-esimo tetraedro intersecato.

-num\_tetr: individua il numero dell’m-esimo tetraedro intersecato

- intersect\_edge: sottostruttura intersect\_tetr(i).array(m). intersect\_edge

-num\_edge: numero del lato tagliato

-coord: nodi intersecati per quel lato. Tali nodi possono essere o quelli dati o quelli aggiungi durante il progetto come nuovi punti di intersezione (spiegazione successiva).

- intersect\_face: sottostruttura intersect\_tetr(i).array(m). intersect\_face. Analogo a intersect\_edge ma riferito alle facce

-num\_face

-coord

1. Si calcola la normale alla frattura n: non c’è la sicurezza che si tratti di un piano quindi, tenendo fisso il primo vertice del poligono, si calcolano 2 normali (essendoci in questo caso 4 vertici), cioè si crea un ciclo j che parte da 2 e finisce al numero di nodi della faccia-1. Per farlo per la normale j-esima si calcola il prodotto vettoriale fra v1 e v2, dove v1 collega il vertice j al primo e v2 il vertice j+1 al primo. Poi per trovare n si normalizza.
2. Poi si calcola l’area del poligono, dividendo il poligono in triangoli, e facendo la norma della metà del prodotto vettoriale.
3. A differenza del 2D, che analizza direttamente tutti i lati, per il 3D è stata fatta una scelta differente. Si apre un ciclo j per trovare la PRIMA FACCIA INTERSECATA. Trovata la prima faccia, dovuta o all’intersezione in un punto interno alla faccia o in un punto appartenente a uno dei suoi lati, sono inseriti in intersect\_tetr i tetraedri che condividono tale faccia (nel primo caso) o tale lato (nel secondo caso) ed è aggiornato **tetra\_tail**. Tetra\_tail è un vettore globale che contiene i vicini dei tetraedri tagliati, cioè quei tetraedri che successivamente saranno da controllare. In particolare, ogni volta che si trova un tetraedro tagliato, si cercano i suoi vicini e, se non sono già presenti, si aggiungono in coda a tetra\_tail. Il fatto che tetra\_tail è continuamente aggiornato è il motivo per cui dopo, quando lo si scorre, non si utilizza il ciclo for, ma un ciclo while. Il ciclo for infatti prevede di fissare la lunghezza del ciclo e non variarla.

Quindi si apre il ciclo j che scorre le facce per cercare la prima e di esce, grazie a un break, quando troviamo la faccia.

**Face\_control**: una volta che una faccia è stata controllata si vuole che non sia più ricontrollata in cicli successivi, quindi si è creato questo vettore global di controllo per evitare calcoli inutili. In questo vettore ogni posizione corrisponde a una faccia ed inizialmente è inizializzato a 0. Quindi

Face\_control(j)=**0** -> da controllare faccia j

Face\_control(j)=**2**-> faccia j controllata

Face\_control(j)=**1**->faccia j tagliata e inserita

La faccia è controllata attraverso la funzione control\_face

1. **Control\_face** è quella funzione che data una faccia e una frattura, trova tutte le intersezioni, le salva in intersect\_tetra ed aggiorna tetra\_tail. Funziona come segue:
2. Si setta a zero il vettore **control\_poly:** è un vettore lungo quanti sono i lati della frattura inizializzato a zero. Questo vettore ci servirà successivamente, per il momento basta notare che se un lato del poligono è tagliato si setta per quel lato a 1.
3. **F** è la normale alla faccia.
4. Si trovano i lati di quella faccia e si salvano in **edge\_face**
5. Per prima cosa si è deciso di dividere due casi, se i piani (faccia e frattura) sono o non sono paralleli:

If il piano della faccia e il piano della frattura sono PARALLELI (cross(n,f)==0

E’ necessario che siano complanari, altrimenti sicuramente non ci sono intersezioni. Se sono complanari, quindi se il prodotto scalare fra n e il lato che collega un punto della faccia a un punto della frattura fa zero, continuo i controlli.

Controlliamo poi se i vertici della faccia sono interni al poligono attraverso la funzione **inner\_point**. A inner\_point si pass il nodo da controllare e l’area della faccia. La funzione calcola l’area dei triangoli formati da due vertici del poligono collegati e dal nodo, quindi trova tante aree quante sono le possibili coppie. In questo caso, ad esempio, la frattura ha 4 vertici. Quindi c’è 1-2, 2-3, 3-4 e 4-1. Quindi trova tanti triangoli quanti sono i lati della frattura. Se il punto è interno al poligono la somma delle aree di questi triangolini è uguale all’area della faccia. Come output inner\_point restituisce flag:

-1: punto esterno non complanare

1: punto interno

0: punto complanare ma esterno

if tutti vertici della faccia interni al poligono isequal(flag,[1,1,1])

se la faccia è completamente interna al poligono non è tagliata. Tuttavia se uno dei vertici della faccia corrisponde a un vertice della frattura lo si considererà successivamente in una delle facce non parallele.

Else alcuni dentro e altri fuori o tutti fuori

Si scorrono con l i lati della faccia. Cosi come per le facce, è inutile ricontrollare più volte uno stesso lato, quindi si utilizza il vettore globale **edge\_control** che funziona esattamente come face\_control. Per verificare in uno di questi vettori di controllo se il lato/la faccia sono stati controllati si utilizza **already\_control**. Questa funzione restituisce zero se è ancora da controllare, 1 se è già stato controllato o inserito. Quindi si continuano i controlli per il lato edge\_face(l) solo se flag è 0.

Il ciclo for scorre con k i lati del poligono, quindi

L -> numero lato faccia

K -> numero lato poligono

If lato faccia e lato poligono paralleli

Si trovano **ki** e **kf** che sono le coordinate parametriche del punto di inizio e di fine rispetto al lato del poligono.

If kf >=1 e ki<1

Il lato del poligono è interno al lato, quindin tale lato è intersecato. Si cercano i tetraedri che lo condividono e si inseriscono tramine **insert\_only\_tetra**. Questa funzione riceve in input il un determinato tetraedro e lo inserisce nella struttura intersect\_tetr. In particolare controlla che non sia già presente: se lo è restituisce la linea array in cui lo ha trovato, altrimenti lo aggiunge e restituisce la linea in cui è stato aggiunto. In caso lo aggiunga nella funzione sono inizializzati anche gli altri campi della struttura necessari per trovare correttamente le intersezioni. Tornati in control\_poly si verifica per ogni tetraedro intersecato se il lato è già stato inserito e, in caso negativo, si aggiunge in intersect\_edge.

If ki<=0

Aggiungo entrambi gli estremi della traccia

Else ki>0

Aggiungo solo l’estremo finale

End

Elseif ki<=0 e kf>0

Ragionamento speculare. Aggiungo solo l’estremo ki, altrimenti saremmo già entrati nel primo caso.

End

Else lato faccia e lato poligono NON paralleli

Trovo il vettore x contenente s e t.

If intersezione nel pt di inizio del lato e in un estremo del lato del poligono

Trovo i tetraedri che condividono quel lato e li inserisco. Per ognuno di essi inserisco il lato e come coordinata il prunto di inizio del lato.

Elseif intersezi. nel pt di fine del lato e in un estremo del lato del poligono

Caso speculare al precedente.

Elseif si intersecano in un punto interno ad entrambi

Trovo il punto di intersezione e lo aggiungo con add\_node. Inserisco anche qui i tetraedri, il lato e il punto trovato.

End

End

If poligono interno faccia

Il poligono potrebbe essere completamente interno alla faccia, quindi bisogna inserire la faccia come tagliati e inserire in intersect i tetraedri che la condividono. Si settano a uno tutti i lati del control\_poly.

End

End punti dentro o fuori

Else i piani NON sono PARALLELI

Inizialmente si scorrono i lati della faccia e si trovano le tre coordinate parametriche di intersezione con il piano del poligono: nel caso il lato della faccia in parallela al poligono (prodotto scalare fra il lato e la normale al poligono è nullo) non cerchiamo il punto di intersezione, poiché ne troverebbe o infiniti o nessuno, ma si pone la coordinata a Inf. Si tratta di un valore ‘speciale’ che permetterà successivamente di distinguere quel lato da quelli con un solo punto di intersezione.

Si apre il ciclo w che scorre i lati della faccia:

if piano poligono NON è parallelo al lato della faccia e

if t(w) è compreso fra 0 e 1

trovo il punto le coordinate del punto di intersezione X e chiamo la funzione inner\_point per controllare che sia interno al/sul bordo del poligono.

If punto di intersezione interno al poligono

If il vertice del lato poggia sul piano del poligono x(1) vale 0 o 1

Controllo se X corrisponde a un vertice del poligono attraverso il vettore flag\_vertex\_poly. flag\_vertex\_poly vale o il numero del vertice o zero se non poggia su un vertice.

If l’estremo del lato poggia su un vertice del poligono flag\_vertex\_poly ~=0

(NB: Non è considerato tagliato il caso in cui un vertice della faccia poggia sul poligono in punto diverso da uno degli estremi del poligono stesso).

Il lato è tagliato e si inserisce in intersect tetra tutti i tetraedri che lo condividono, specificando il lato e il punto X (cioè uno dei due estremi del lato). Si aggiorna control\_poly.

Elseif l’estremo del lato NON poggia su un vertice del poligono

Il lato non è considerato tagliato. Tuttavia se X cade sul perimetro del poligono bisogna salvarsi in control\_poly che il lato su cui poggia è tagliato. Per controllare se il punto X cade sul lato c del poligono (P(c) è il punto c-esimo del poligono) si controlla che il prodotto vettoriale fra P(c)-X e P(c+1)- P(c) sia zero, quindi che siano paralleli.

Inoltre aggiungere X a special\_edge.

**SPECIAL\_EDGE**: special edge è un vettore nel quale si inseriscono quei punti del che appartengono al poligono ma che in teoria non fanno in modo che i tetraedri a cui appartengono siano tagliati. Tuttavia, se quel tetraedro presenta altre intersezione, quei punti sono da considerare tra i tagliati. In mezzo ai controlli non si conoscono tutti i tetraedri tagliati, quindi si salvano questi nodi speciali in questo vettore. Così, alla fine, si potranno aggiungere in intersect\_tetra se e solo se i tetraedri a cui appartengono sono già nella struttura.

End

Else intersezione interna al lato faccia (e interna al poligono)

Se X appartiene al bordo del poligono (prodotto vettoriale p1-p2 e p1-X nullo) si aggiorna control\_poly.

X è aggiunto con add\_node e si aggiorna intersect\_tetra con tutti i tetraedri che condividono quel lato

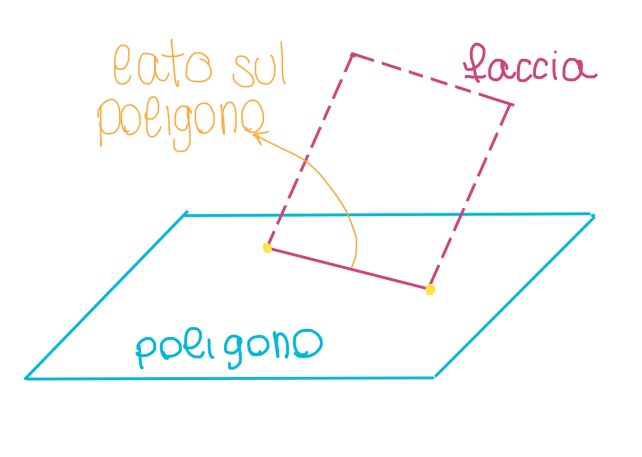
End

End punto interno al poligono

End interno al lato

Else lato della faccia parallelo al poligono e complanare (sono complanari se n e p1-x1 sono perpendicolari, cioè hanno prodotto vettoriale nullo)

Se un punto del lato appartiene al piano del poligono allora il lato della faccia è complanare al poligono e, in questo caso, si controlla con inner\_point se gli estremi del lato sono interni al poligono.



If lato del tetraedro interno o sul bordo

Se il lato è completamente interno al poligono devo considerarlo tagliato solo se uno dei due vertici del lato del tetraedro corrisponde a un vertice del poligono.

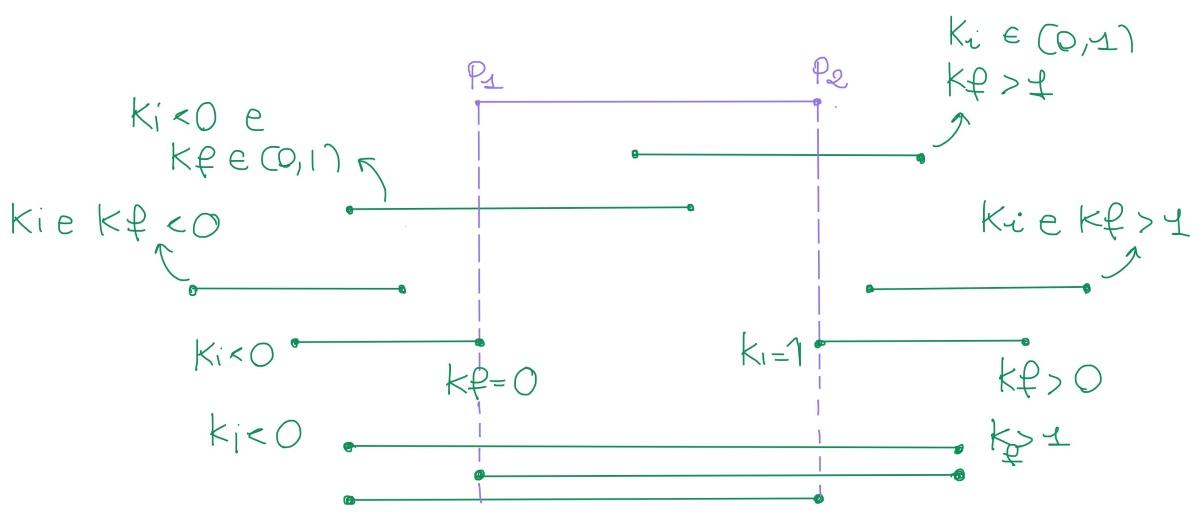
Quindi si scorrono i vertici del poligono e si controlla se corrispondono a uno dei due vertici del lato. In caso negativo il lato non è da inserire. Si aggiorna anche control\_poly.

Else lato esterno o un estremo fuori e uno dentro

Scorro i lati del poligono

If lato faccia parallela lato poligono e coincidenti (prodotto vettoriale nullo)

Come nel cado dei piani paralleli si trovano k1 e kf: coordinata parametrica del vertice del lato del tetraedro rispetto al lato del poligono



if kf>=1-toll && ki<=1-toll

Lato intersecato, si aggiorna intersect\_tetra e control\_poly

Else lato faccia non parallela lato poligono

Mi salvo in **inner\_point** quale dei due estremi del lato della faccia è interno e -1 se sono entrambi esterni.

Calcolo le coordinate parametriche di intersezione fra le due rette perché so che non sono parallele

If s e t comprese fra 0 e 1

Calcolo il punto di intersezione

If il punto di intersezione appartiene al bordo del poligono, cioè è proprio inner\_point

Considero il punto intersecato solo se coincide con un vertice del poligono, altrimenti lo inserisco in special\_edge.

End

End

End

End

End

End

RIASSUNTO CONTROL\_FACE

If il piano della faccia e il piano della frattura sono PARALLELI (cross(n,f)==0

if tutti vertici della faccia interni al poligono isequal(flag,[1,1,1])

Else alcuni dentro e altri fuori o tutti fuori

If lato faccia e lato poligono paralleli

If kf >=1 e ki<1

Elseif ki<=0 e kf>0

End

Else lato faccia e lato poligono NON paralleli

If intersezione nel pt di inizio del lato e in un estremo del lato del poligono

Elseif intersezi. nel pt di fine del lato e in un estremo del lato del poligono

Elseif si intersecano in un punto interno ad entrambi

End

End

If poligono interno faccia

End

End punti dentro o fuori

Else i piani NON sono PARALLELI

if piano poligono NON è parallelo al lato della faccia e

if t(w) è compreso fra 0 e 1

If punto di intersezione interno al poligono

If il vertice del lato poggia sul piano del poligono x(1) vale 0 o 1

If l’estremo del lato poggia su un vertice del poligono flag\_vertex\_poly ~=0

Elseif l’estremo del lato NON poggia su un vertice del poligono

End

Else intersezione interna al lato faccia (e interna al poligono)

End

End punto interno al poligono

End interno al lato

Else lato della faccia parallelo al poligono e complanare (sono complanari se n e p1-x1 sono perpendicolari, cioè hanno prodotto vettoriale nullo)

If lato del tetraedro interno o sul bordo

Else lato esterno o un estremo fuori e uno dentro

If lato faccia parallela lato poligono e coincidenti (prodotto vettoriale nullo)

if kf>=1-toll && ki<=1-toll

End

Else lato faccia non parallela lato poligono

If s e t comprese fra 0 e 1

If il punto di intersezione appartiene al bordo del poligono,

End

End

End

End

End

End

TAGLIATA SOLO FACCIA NO LATI

Con un metodo simile a quello precedente, calcoliamo la coordinata di intersezione fra il lato del poligono e il piano della faccia del tetraedro.

Scorriamo i lati del poligono: per ognuno controlliamo con control\_poly se non è stato già intersecato. Infatti, se è già ha avuto intersezioni, è impossibile che intersechi l’interno della faccia. Se quel lato del poligono non ha intersecato è chiamata la funzione **intersect\_onlyface\_noedge.** Questa funzione si occupa di trovare le intersezioni nel caso il poligono entri nella faccia del tetraedro senza intersecare nessun lato

1. If lato poligono e faccia tetraedro paralleli

Si calcola l’area della faccia e ci si controlla se i vertici del lato del poligono sono interni alla faccia.

If vertice del poligono interno alla faccia

Di controlla che tale vertice non sia sul bordo e nel caso non lo sia inserisco i tetraedri che condividono quella faccia.

End

Else lato poligono e faccia tetraedro NON paralleli

Se t(w) compreso fra 0 e 1 troviamo il punto di intersezione e si controlla sempre che non sia sul bordo. Se non è sul bordo controllo che il punto sia interno e, in caso affermativo, lo si inserisce.

End

A questo punto, abbiamo trovato la prima faccia e scorso tutto tetra\_tail. Quindi sono state trovate tutte le intersezioni.

1. **classify\_vertex\_poly**: scorro i tetraedri intersecati e controllo se i vertici del poligono sono interni al tetraedro. Quelli che sono interni li devo aggiungere a quel tetraedro se non sono mai stati aggiunti. Inizializzo per tutti i tetraedri intersecati la nuova struttura dove inserirli: **intersect\_vertex\_poly**.

Per farlo per ogni tetraedro intersecato controllo con una tecnica simile a quella delle aree, ma con i volumi, se il nodo del poligono è interno a quel tetraedro.

Se è interno a un tetraedro bisogna controllare se appartiene a una delle facce. Troviamo le quattro facce del tetraedro e lo scopriamo con il metodo delle aree. Se appartiene a una faccia bisogna inserirlo per tutti i tetraedri intersecati che condividono tale faccia, altrimenti solo per quel tetraedro.

1. Bisogna analizzare i vertici in special\_edge come è stato spiegato precedentente attraverso **analyze\_special\_edge:**

per ogni nodo in special\_edge cerchiamo tutti i tetraedri a cui appartiene (share\_tetra). Poi scorriamo i tetraedri intersecati e vediamo se un certo tetraedro intersecato è fra i share\_tetra.

Trovato il tetraedro bisogna trovare il lato giusto in cui inserirlo. Quindi trovo la posizione del vertice special\_edge in nel tetraedro intersecato con position e cerco qual è il lato che collega position con position+1. Trovato il lato eedge, se è già presente fra i lati intersecati aggiorniamo solo coord, altrimenti bisogna aggiungerlo e aggiornare coord.

# Punto 2.

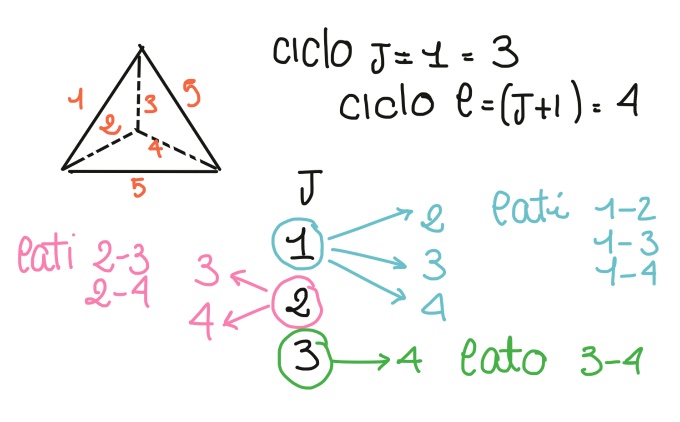
Per ciascuna frattura individuare l'elenco dei tetraedri che condividono almeno un vertice

con un tetraedro tagliato e per questi memorizzare anche quali vertici o lati sono condivisi

con un tetraedro tagliato.

Si inizializza un ramo di intersect\_tetra, riservato a trovare i tetraedri vicini: **intersect\_tetr(i).array(m).share**

Ha 3 campi:

* share\_num\_tetr: numero del tetraedro vicino
* share\_nodes\_tetr: vertici condivisi
* share\_edge\_tetr: lati condivisi

Si scorrono i tetraedri intersecati con m, e con j si scorrono i suoi vertici.

Per ogni vertice di ogni tetraedro intersecato si cercano gli altri tetraedri con cui è condiviso (evitando lui stesso) e si controlla se condivide con essi quel nodo. In caso positivo si aggiunge quel tetradro fra i condivisi e il relativo nodo. Prima di fare ciò si effettuano tutti i controlli per vedere s equel tetraedro non era già stato trovato.

Dopo che sono stati scorsi i vertici dell’m-esimo tetraedri intersecato si fa un controllo simile con i lati: per fare ciò si fanno i seguenti due cicli:

Trovato il lato edge\_tetra si scorrono tutti i tetraedri e controllo se quel tetraedro contiene i due nodi di quel lato. In caso positivo si aggiorna intersect\_tetr(i).array(m).share con edge\_tetra e il relativo tetraedro vicino.

# Punto 3.

**Poligonizzazione della frattura:** viene aperta la funzione **POLIGONIZZAZIONE\_POLY:** Tale funzione ruota innanzitutto gli estremi della frattura su un piano, tramite la matrice di rotazione data dal vettore perpendicolare alla frattura, il vettore di un lato del poligono e un altro vettore perpendicolare ai due. A questo punto bisogna inserire in uno stesso vettore tutti i punti di intersezione che sono stati trovati tra la frattura e i tetraedri tagliati: si scorrono i tetraedri tagliati e per ognuno si scorrono le intersezioni con i lati, le intersezioni con le facce e i punti di intersext\_vertex\_poly (ovvero i punti del poligono interni a quel tetraedro) e una volta ruotati si inseriscono tutti nel vettore vertex\_newcoord, in modo che nelle prime tre colonne ci siano le componenti x’, y’ e z’ (che sarà nulla per la base scelta) dei punti e nella quarta ci sia il numero del nodo. Ora è necessario prendere in considerazione l’immagine del tetraedro sulla frattura solo e questa ha più di due punti, per cui se ha esattamente tre punti non è necessario ordinarli perché comunque inseriti formeranno un triangolo; se sono più di tre è necessario chiamare **ORDERING\_VERTEX.**

Tale funzione trova tra tutti i punti in vertex\_newcoord quello che ha la coordinata x’ minore (p1) e divide tutti i restanti punti in due gruppi, quelli che hanno la y maggiore o uguale di quella di p1 e i restanti, in due vettori diversi vertex\_newcoord\_up e vertex\_newcoord\_down. Prima di iniziare l’analisi devo ordinare i punti del vettore down i modo che abbiano le coordinate y crescenti mentre quelli di up decrescenti, perché essendo che la funzione min trova nel vettore il primo punti che ha la coordinata x minore dobbiamo essere sicuri che sia anche il punto che ha la y maggiore per i punti down e la y minore per i punti up. Inizio a scorrere i punti down ricercando ogni volta quello con la x minore, togliendolo dal vettore e inserendolo in polig\_poly la struttura i poligonizzazione del poligono. Si scorre poi up allo stesso modo ma inserendo i punti nella struttura al contrario perché ora quelli con la x minore saranno gli ultimi del poligono.

**Division\_intersect\_tetra:** funzione che trova la parte di frattura interna al tetraedro. Scorre i tetraedri intersecati, scorre poi tutti i lati del tetraedro e controlla se il lato e parallelo al piano della frattura, nel caso lo sia, controlla poi se sono complanari. Se sono complanari inserisce entrambi gli estremi del lato in vertex\_even\_poly; nel caso invece non siano neanche paralleli trova le coordinate parametriche dell’intersezione tra piano e retta passante per il lato, controlla quindi che l’intersezione sia interna al lato e interna al poligono e se sì le metto in vertex\_even\_poly che quindi contiene ora tutti i punti di intersezione tra il tetraedro e il piano della frattura. Per trovare ora il poligono formato dalla parte della frattura interna al tetraedro prende i punti di intersect\_edge, intesect\_face ed eventuali punti di del poligono interni al tetraedro e li salva in section\_poly\_inner. A questo punto ordina sia vertex\_even\_poly che section\_poly\_inner.

**Poliedrizzazione dei tetraedri:** Viene chiamata la funzione **poliedrizzazione:** essa scorre i tetraedri intersecati, per ognuno sappiamo che l’immagine del tetraedro sul poligono sarà una faccia di entrambi i poliedri per cui la salviamo in intersect\_tetr(i).array(w).poliedrizzazione.upper\_poly\_faces.coord e in intersect\_tetr(i).array(w).poliedrizzazione.lower\_poly\_faces.coord i punti in vertex\_even\_poly. Ora per ogni tetraedro è necessario trovare le sue facce e poi scorrerle. Per ogni faccia tramite il prodotto scalare dividiamo i suoi punti in tre vettori, quelli che sono sopra i piano (upper) , quelli sul piano (middle) e quelli sotto (lower). Analizziamo tali vettori: se upper è vuoto e lower no o al contrario, la faccia è tutta da una parte e non viene tagliata per cui è da inserire come faccia di uno dei due poliedri. Se invece upper e lower sono entrambi vuoti i punti sono tutti sul piano, la faccia poggia sul piano del poligono quindi il tetraedro non deve essere poligonizzato ed esco dal for. Se invece sono un po’ sopra e un po’ sotto il piano inserisco nel poliedro inferiore i punti di lower e in quello superiore i punti di upper, mentre quelli di middle in entrambi. Ora sempre in questo caso cerco anche quali punti di vertex\_even\_poly sono interni alla faccia considerata tramite il calcolo dell’area già implementato precedentemente e se si, li inserisco nelle facce di entrambi i poliedri. Una volta trovate tutte le facce dei due poligoni le ordino se hanno più di tre punti come fatto precedentemente.