Triangolazione 2D su C++

I ragionamenti che tratteremo in questa parte di progetto si basano sui ragionamenti e sui calcoli precedentemente trattati nella parte di progetto sulla triangolazione 2D su Matlab. La principale differenza è la struttura su cui abbiamo salvato i dati. Su Matlab ci siamo basati su vettori e matrici e salvavamo gli indici, ad esempio quelli dei triangoli intersecati. Su C++ ci siamo basati sulle classi GenericMesh e GenericDomain e abbiamo salvato i puntatori, ad esempio quelli dei triangoli intersecati.

L’analisi delle intersezioni delle tracce si svolge in due livelli:

1. Analisi delle intersezioni con i triangoli della triangolazione originale
2. Analisi dei sottopoligoni dei triangoli della triangolazione originale intersecati

Questa analisi su due livelli permette di ridurre il numero di sottopoligoni controllati nel caso siano presenti molte tracce.

# Librerie utilizzate

Sono utilizzate alcune delle librerie principali del C++: iostrem, fstrem, vector e cmath. Sono utilizzate anche alcune librerie date dal docente: GenericDomain.hpp, GenericMesh.hpp, Eigen.hpp, ConfigFile.hpp e Output.hpp. È stata creata anche una libreria MyMesh.hpp.

La libreria MyMesh.hpp contiene due classi:

* *class GenericCellTree* è figlia della classe GenericCell che è figlia della classe GenericTreeNode contenute entrambe nella libreria GenericMesh. Questa classe contiene un vettore (vector<GenericCellTree\*> leaves) per salvare le foglie dell’albero e un puntatore alla radice (GenericCellTree\* grandfather) dell’albero. La classe contiene anche le funzioni per l’utilizzo delle strutture dati presenti nella classe padre e nella classe nonno, ad esempio si possono leggere e aggiungere i figli (salvati nel vettore childs) e il padre (salvato in father). È presente anche la funzione NumberOfGrandchilds() che permette di calcolare il numero di figli, figli dei figli e così via presenti nel sottoalbero del nodo che chiama la funzione.
* *class GenericIntersect* serve a salvare le informazioni sui triangoli e sui poligoni intersecati. Salva il puntatore al triangolo o poligono intersecato, la coordinata parametrica dell’intersezione sulla traccia, il tipo di contatto (lato o nodo), l’elemento intersecato.

Il tipo di contatto è un intero:

* 1 = intersezione con un lato
* 2 = intersezione con un nodo
* 3 = intersezione con un lato, la traccia è parallela al lato
* 4 = intersezione con un nodo, la traccia è parallela ad uno dei lati il cui nodo è estremo
* 5 = la traccia è totalmente interna al triangolo/poligono

L’elenco degli elementi intersecati è salvato in un vettore di puntatori void, così si possono salvare sia i puntatori ai lati che quelli ai nodi che sono di classi diverse. Tramite il tipo di contatto salvato si può risalire alla classe originale utilizzando un casting.

Sono presenti anche le funzioni per visualizzare e agire sulle strutture dati della classe.

# Salvataggio dati in input

Il dominio della triangolazione è salvato in un elemento della classe GenericDomain2D.

I nodi della triangolazione vengono salvati nel vettore *node* di puntatori ad elementi GenericPoint.

I lati della triangolazione vengono salvati nel vettore *edge* di puntatori ad elementi GenericEdge, salvando i puntatori ai nodi che sono gli estremi del lato. Nei nodi vengono salvati anche i puntatori ai lati di cui il nodo è estremo.

I triangoli vengono salvati nel vettore *ele* di puntatori ad elementi GenericCellTree (figlia della classe GenericCell). Dentro ogni elemento vengono salvati i nodi del triangolo in senso antiorario e i lati che hanno per estremi quei nodi. Nei vettori dei nodi e in quello dei lati vengono salvati anche i puntatori al triangolo.

Dentro il vettore *tracce* di puntatori ad elementi GenericDomain1D sono salvate le tracce che intersecano il dominio.

Scorro tutte le tracce e per ciascuna traccia eseguo i seguenti comandi:

# Analisi della triangolazione

Calcolo la normale alla traccia.

Scorro tutti i lati della triangolazione originale e per ciascuno calcolo (tramite la funzione *scalar\_product*) il prodotto scalare tra la normale alla traccia e la congiungente tra il primo estremo della traccia e i ciascuno degli estremi del lato. Se l'estremo del lato sta sulla retta della traccia, il prodotto scalare viene 0. Si distinguono 4 casi:

1. I prodotti scalari sono discordi: gli estremi del lato sono nei due semipiani diversi generati dalla traccia, quindi c’è intersezione tra lato e retta della traccia (traccia stessa o suo prolungamento).
2. I prodotti scalari sono entrambi nulli: entrambi gli estremi del lato sono sulla retta della traccia, quindi potrebbe esserci intersezione.
3. Un prodotto scalare è nullo e l’altro no: uno degli estremi del lato è sulla traccia e l’altro no.
4. I prodotti scalari sono concordi: entrambi gli estremi del lato sono nello stesso semipiano generato dalla traccia, quindi sicuramente non c’è intersezione tra lato e traccia.

Nel primo caso troviamo il vettore x che contiene le coordinate parametriche s e t date dall’intersezione traccia-lato, riguardanti rispettivamente il lato (x[0]=s) e la traccia (x[1]=t), considerando che le due rette non possono essere parallele in questo caso. Se sia s che t sono compresi fra 0 e 1 (estremi compresi) possiamo avere la sicurezza che esista un punto di intersezione fra la traccia e il lato. Quindi passiamo alla funzione *insert* (funzione spiegata meglio al fondo) le coordinate parametriche affinché questa funzione inserisca nel vettore intersect\_triangle tutti i triangoli che condividono quel lato, la coordinata parametrica dell’intersezione e il lato/nodo intersecato.

Nel secondo caso entrambi gli estremi sono sulla retta della traccia, quindi si non possono trovare le coordinate parametriche di intersezione tra traccia e lato perché sono infinite. Allora vengono calcolati ki e kf che rappresentano rispettivamente la coordinata parametrica del punto di inizio e di fine del lato rispetto alla retta della traccia; sono stati trovati sostituendo l’estremo del lato considerato nell’equazione parametrica della traccia e risolvendo il conseguente sistema. Vengono ordinati ki e kf così da fare controlli più agevoli. Sono poi stati analizzati i diversi casi:

* ki>0 e kf<1: il lato è completamente interno alla traccia, quindi il lato non è considerato tagliato.
* kf<=0 o ki>=1: il lato inizia dopo la fine della traccia o finisce dell’inizio della traccia quindi il lato e esterno alla traccia e non è tagliato.
* (kf>=1 e ki<1) o (ki<=0 e kf>0): se il lato inizia prima della fine della traccia abbiamo già la sicurezza, avendo escluso gli altri casi, che kf sarà maggiore o uguale a 1. Analogamente se il lato finisce dopo l’inizio della traccia, ki non potrà essere maggiore di 0 (altrimenti sarebbe già entrato nel primo caso).

Quindi c’è la sicurezza che almeno un estremo è da inserire. Per sfruttare la funzione insert, che controlla che s=x[0] sia un valore accettabile (fra 0 e 1), bisogna passare x[0]. Tuttavia in questo caso non si calcola perché verrebbe infinito, quindi si può passare alla funzione un qualunque valore di x[0] accettabile e come t=x[1] o 0 o 1, a seconda se analizziamo ki o kf. Ad esempio x[0]=0.5 va bene come valore di s. Se il lato è intersecato esattamente in un estremo allora bisognerà inserire o 0 o 1 a seconda dell’estremo intersecato, viene tenuto conto anche se ki e kf sono stati scambiati o no.

Nel terzo caso un nodo del lato è sulla traccia. Troviamo il vettore x che contiene le coordinate parametriche s e t date dall’intersezione traccia-lato, riguardanti rispettivamente il lato (x[0]=s) e la traccia (x[1]=t), considerando che le due rette non possono essere parallele in questo caso. Vengono analizzati i triangoli che hanno quel lato. Tramite la funzione check\_node viene controllato se il prolungamento della traccia interseca il lato del triangolo che non ha per estremo il nodo intersecato. In caso affermativo viene inserito il triangolo nel vettore intersect\_triangle con tutte le informazioni riguardanti l’intersezione tramite la funzione insert.

Nel quarto caso non viene effettuata nessuna operazione perché il lato considerato non è intersecato.

Dopo aver scorso tutti i lati della triangolazione originale sono salvati tutti i triangoli della triangolazione originale intersecati.

Se non sono stati trovati triangoli intersecati vuol dire che tutta la traccia è interna ad un triangolo. Scorro tutti i triangoli e per ogni triangolo calcolo la sua area tramite la norma del prodotto vettoriale divisa per due. Poi viene calcolata la somma delle aree dei triangoli che hanno per vertici un estremo della traccia e due vertici consecutivi del poligono. Se l’area del triangolo e la somma delle aree sono uguali allora la traccia è interna al triangolo e quindi il triangolo viene considerato intersecato (il tipo di contatto è 5, la coordinata di intersezione è impostata a -1 e il puntatore all’elemento intersecato è impostato a NULL).

Ora il programma, scorrendo tutti i triangoli intersecati, si divide in due procedure: una per i triangoli attivi (ovvero quelli che non sono mai stati intersecati prima) e una per i triangoli passivi (ovvero quelli che sono stati intersecati da almeno una traccia in precedenza).

# Analisi dei triangoli attivi tagliati (funzione triangle\_active)

I triangoli attivi vengono salvati nel vettore *intersect\_poly* che conterrà tutti i triangoli attivi e sotto-poligoni intersecati.

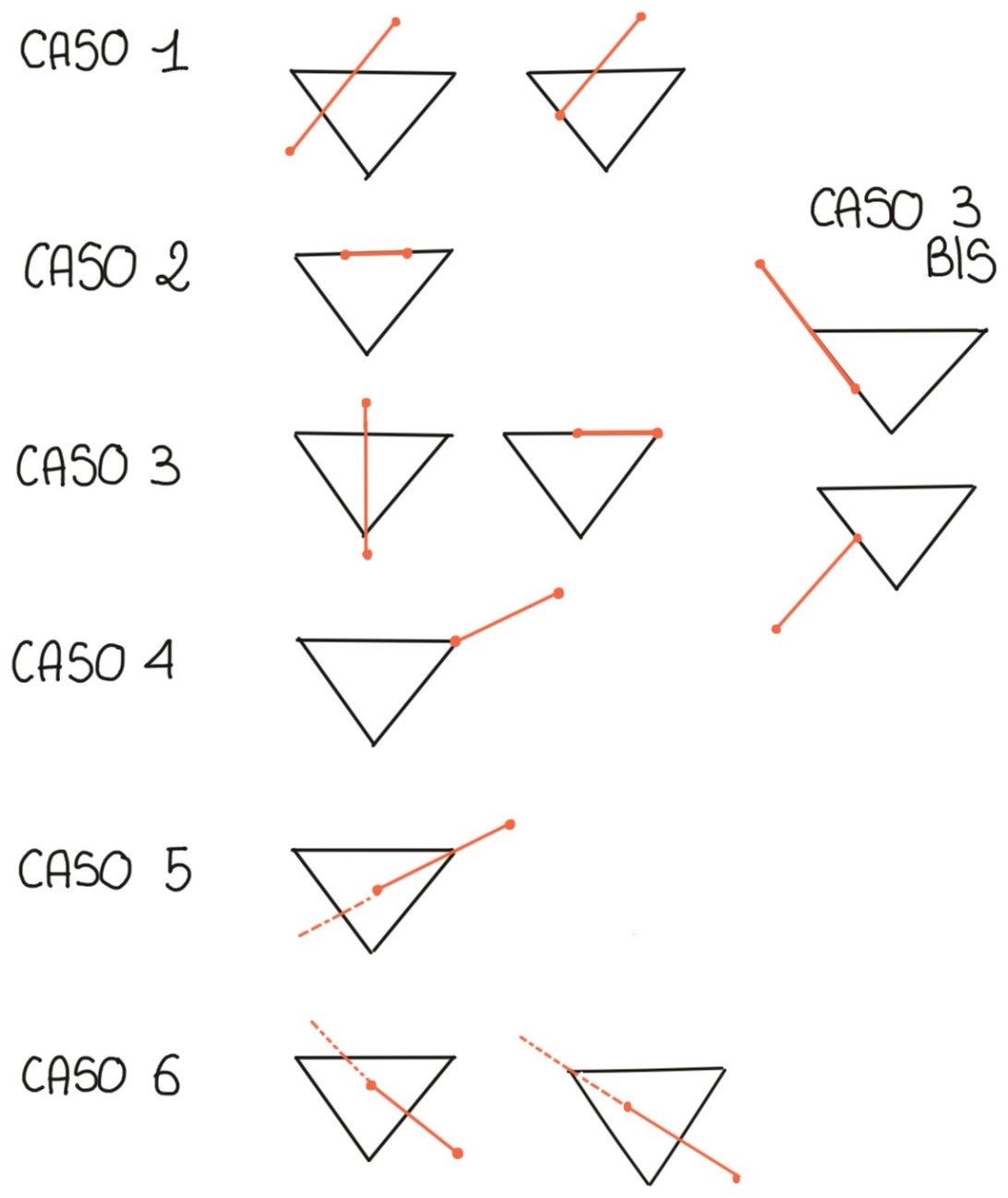
La funzione triangle\_active riceve in ingresso il riferimento al vettore contenente le informazioni sulle intersezioni, l’indice del triangolo intersecato e della traccia, il riferimento del vettore contenente i nodi (che restituirà come return) e quello dei lati.

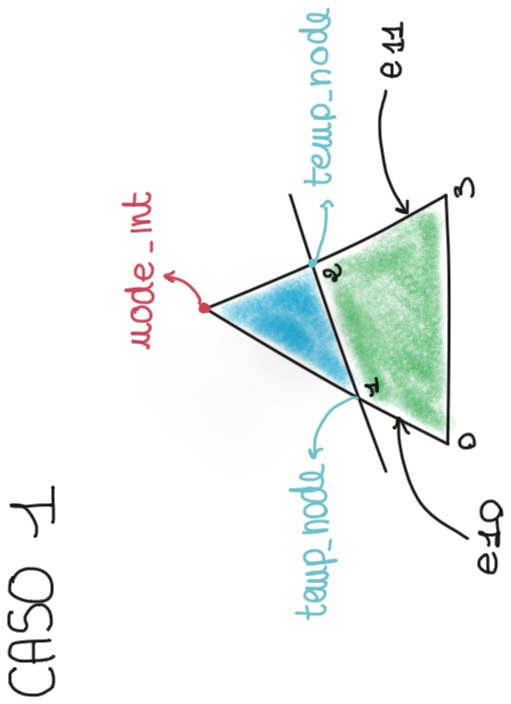
Sono possibili 7 diversi casi:

1. Ci sono due lati diversi intersecati
2. Ci sono due intersezioni sullo stesso lato
3. Ci sono due intersezioni, una con un lato e l’altra con un nodo

3bis. C’è un lato intersecato con un estremo della traccia

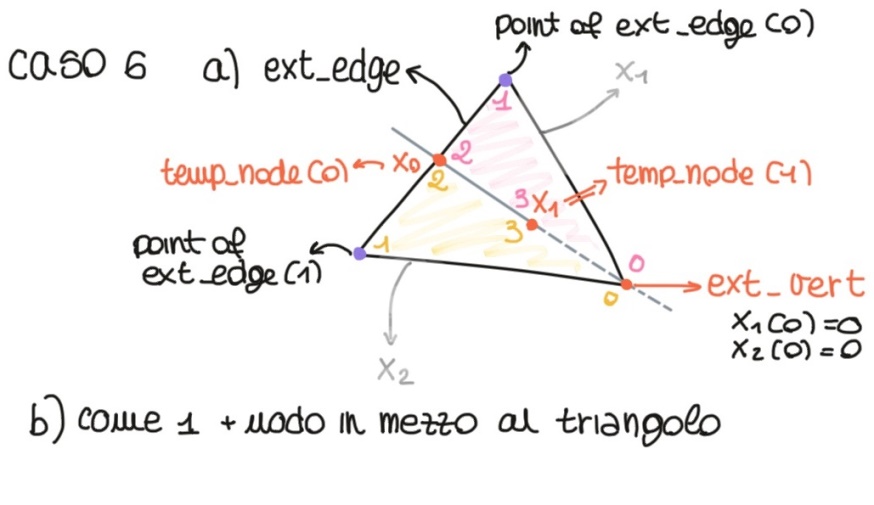
1. C’è un nodo intersecato con un estremo della traccia
2. C’è un lato intersecato e la traccia finisce dentro il triangolo
3. C’è un nodo intersecato e la traccia finisce dentro il triangolo



I 7 casi svolgono azioni simili, ma per chiarezza viene dato un cenno di ragionamento per ciascun caso:

1. I triangolo sarà diviso in un triangolo e in un quadrilatero. Vengono calcolate le coordinate delle intersezioni con i due lati. Viene cercata la presenza di questi nodi all’interno del vettore *node*, in caso di assenza vengono creati nuovi nodi. In due vettori ausiliari *aux\_vert* vengono salvati i nodi che formano i nuovi poligoni controllando che siano nell’ordine corretto (a prescindere dal senso orario o antiorario). Tramite la funzione anticlockwise vengono ordinati i nodi in senso antiorario (tramite l’uso del prodotto vettoriale). In altri due vettori ausiliari *temp\_edge* vengono salvati i lati che formano i nuovi poligoni prima ricercandoli tra i lati esistenti e in caso di assenza creandone di nuovi (i lati sono in senso antiorario perché sono cercati quando i nodi erano già in senso antiorario, il primo nodo sarà uno dei due estremi del primo e dell’ultimo lato, il secondo nodo sarà uno degli estremi del primo e del secondo lato e così via). Dopo aver creato i quattro vettori ausiliari con le informazioni sui lati e nodi dei nuovi poligoni, queste vengono inserite in due nuovi figli del triangolo originale ripetendo due volte il seguente procedimento: viene creato un nuovo oggetto della classe GenericCellTree, viene impostato il suo stato su attivo (true), viene aggiungo il puntatore al padre e il puntatore alla radice e vengono aggiunte seguendo il senso antiorario i puntatori ai nodi e ai lati del nuovo poligono; al padre viene aggiunto il puntatore al figlio e viene aggiunto lo stesso puntatore anche come foglia perché childs può contenere solo elementi costanti, mentre leaves no. Infine viene impostato lo stato del triangolo originale su disattivo (false).
2. Il triangolo diventerà un poligono con 5 vertici. Vengono calcolate le coordinate delle intersezioni con il lato. Viene cercata la presenza di questi nodi all’interno del vettore *node*, in caso di assenza vengono creati nuovi nodi. In un vettore ausiliario *aux\_vert* vengono salvati i nodi che formano il nuovo poligono controllando che siano nell’ordine corretto (i nodi saranno gli stessi della triangolo iniziale con l’aggiunta delle intersezioni). Tramite la funzione anticlockwise vengono ordinati i nodi in senso antiorario. In un altro vettore ausiliario *temp\_edge* vengono salvati i lati che formano il nuovo poligono. Vengono inserite le informazioni sui lati e sui nodi del nuovo poligono in un nuovo figlio del triangolo originale seguendo il procedimento descritto nel primo caso.
3. Il triangolo sarà diviso in due triangoli. Viene calcolata la coordinata dell’intersezione con il lato. Viene cercata la presenza di questo nodo all’interno del vettore *node*, in caso di assenza viene creato un nuovo nodo. In due vettori ausiliari *aux\_vert* vengono salvati i nodi che formano i nuovi poligoni controllando che siano nell’ordine corretto. Tramite la funzione anticlockwise vengono ordinati i nodi in senso antiorario. In altri due vettori ausiliari *temp\_edge* vengono salvati i lati che formano i nuovi poligoni. Vengono inserite le informazioni sui lati e sui nodi dei nuovi poligoni in due nuovi figli del triangolo originale seguendo il procedimento descritto nel primo caso.

3bis. Il triangolo diventerà un poligono con 4 vertici. Il procedimento è identico al caso 2, ma con un’intersezione al posto che due.

1. Il figlio del triangolo intersecato sarà una copia identica del triangolo originale perché l’intersezione sul nodo non comporta un cambiamento di nodi o di lati.
2. Il triangolo sarà diviso in due triangoli. Il procedimento è simile al caso 3, ma l’intersezione con il lato questa volta è generata dal prolungamento della traccia. Oltre ai nodi salvati nel caso 3 viene aggiunto anche l’estremo della traccia che è interno al triangolo.
3. In questo caso la traccia passa per un lato e termina dentro al triangolo. La risoluzione si suddivide in due sotto-casi:
   1. L’intersezione del prolungamento della traccia è in un vertice del triangolo: allora la sottopoligonalizzazione è simile al caso 5.
   2. L’intersezione del prolungamento della traccia è in un lato del triangolo: allora la sottopoligonalizzazione è simile al caso 1, ma con l’aggiunta dell’estremo della traccia che è interno al triangolo.

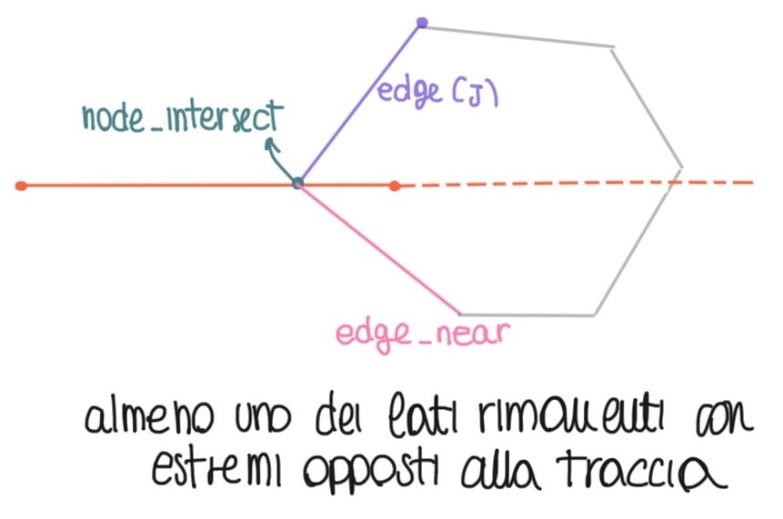
# Analisi dei triangoli disattivi tagliati (funzione child\_active)

Per ogni triangolo disattivo intersecato scorro le foglie (salvate in leaves) per cercare quelle intersecate.

La funzione child\_active riceve il puntatore alla foglia, il riferimento al vettore contenente le tracce, quello ai poligoni intersecati (intersect\_poly), l’indice della traccia in considerazione e la sua normale.

Per risparmiare righe di codice è stato aggiunto temporaneamente al fondo dell’elenco dei nodi del poligono il primo nodo, idem per i lati.

Vengono analizzati i lati del poligono sapendo che per costruzione il primo e il secondo nodo sono gli estremi del primo lato, il secondo e terzo nodo sono gli estremi del secondo lato e così via. Per ogni lato viene fatta un’analisi simile a quella fatta in precedenza ai lati della triangolazione originale, iniziando tutto dal calcolo dei prodotti scalari tra la normale alla traccia e le congiungenti tra il primo estremo della traccia e i ciascuno degli estremi del lato. Questa volta se:

* I prodotti scalari sono discordi viene chiamata la funzione *insert\_child* per inserire il poligono intersecato nel vettore *intersect\_poly*. La funzione ha un codice molto simile alla funzione *insert* usata in precedenza.
* I prodotti scalari sono entrambi nulli vengono calcolati ki e kf come in precedenza. In base ai risultati vengono passati adeguati parametri alla funzione *insert\_child* per inserire le informazioni in *intersect\_poly*. A differenza di tutte le altre chiamate di insert o di intert\_child questa volta il tipo di contatto è 3 se viene intersecato un lato e 4 se viene intersecato un nodo. Questo è stato fatto per poter gestire al meglio la sottopoligonalizzazione effettuata successivamente.
* Un prodotto scalare è nullo e l’altro no controllo se almeno uno dei lati (che non ha per estremo il nodo intersecato) ha gli estremi dai lati opposti della traccia o del suo prolungamento. Se non ci sono lati che soddisfano questa richiesta vuol dire che la traccia passa solo su quel vertice e non interseca il poligono. Nel caso passasse da quel vertice e poi su un lato parallelo alla traccia non lo considero ora ma nel caso dei prodotti scalari entrambi nulli quando analizzo il lato parallelo. Se il controllo ha dato esito positivo passo adeguati parametri alla funzione *insert\_child* per inserire le informazioni in *intersect\_poly*.

Se non sono stati trovati foglie intersecate vuol dire che tutta la traccia è interna ad una foglia. Scorro tutte le foglie e per ogni foglia calcolo la sua area sommando le aree dei triangoli con vertici il nodo 0 e due nodi consecutivi del poligono (le aree sono calcolate tramite la norma del prodotto vettoriale divisa per due). Poi viene calcolata la somma delle aree dei triangoli che hanno per vertici un estremo della traccia e due vertici consecutivi del poligono. Se l’area del poligono e la somma delle aree dei triangoli con l’estremo della traccia sono uguali allora la traccia è interna al triangolo e quindi il triangolo viene considerato intersecato (il tipo di contatto è 5, la coordinata di intersezione è impostata a -1 e il puntatore all’elemento intersecato è impostato a NULL).

Ora dentro *intersect\_poly* sono salvate le foglie tagliate in coda ai poligoni precedentemente tagliati.

# Analisi dei sottopoligoni tagliati (funzione sotto\_poly\_leaves)

Si scorre *intersect\_poly* dalla prima foglia intersecata all'ultima del triangolo in considerazione per sottopoligonare le foglie.

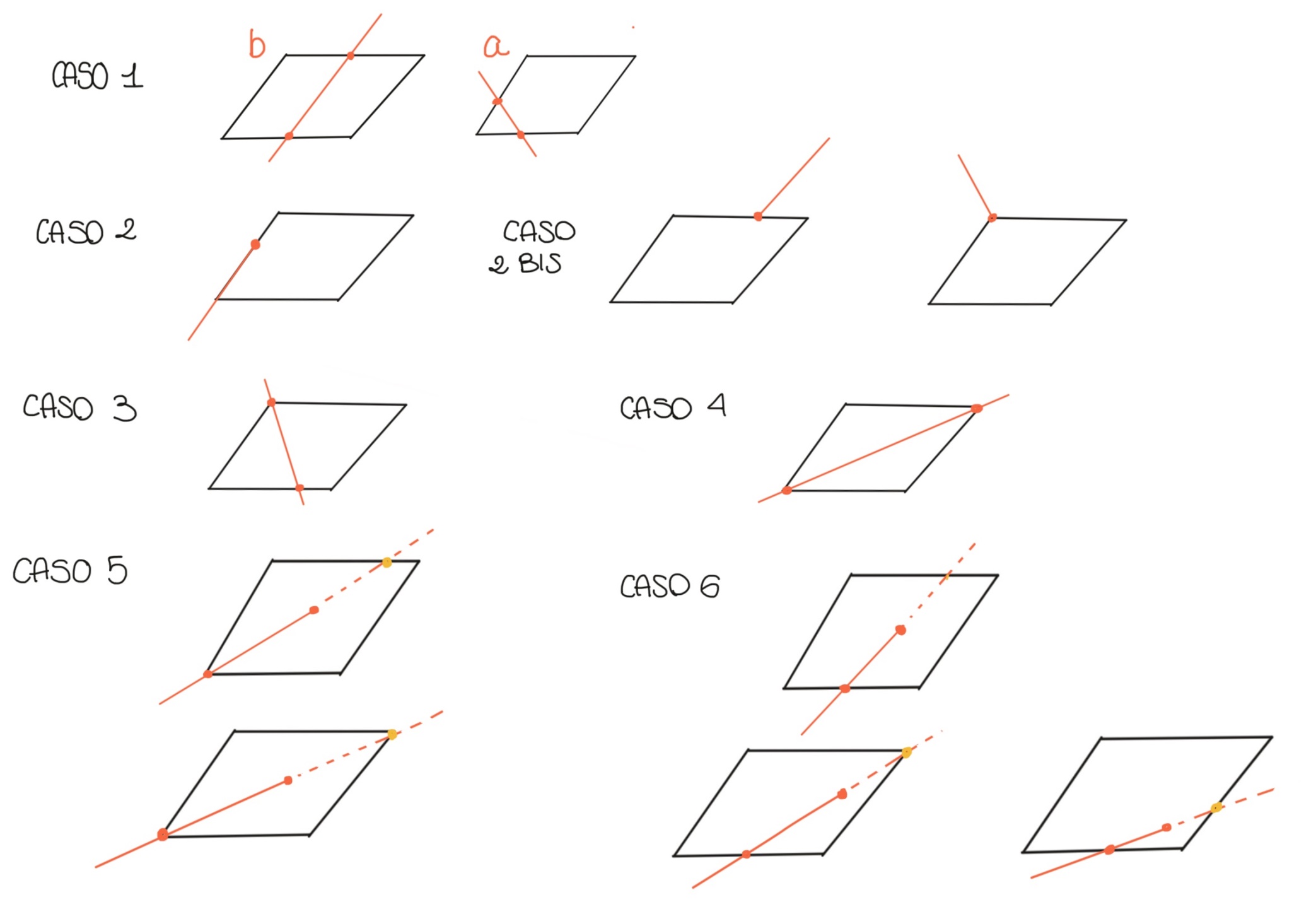
La funzione *sotto\_poly\_leaves* riceve in ingresso il riferimento all’elemento del vettore intersect\_poly che stiamo considerando e quindi contenente tutte le informazioni sulle intersezioni, riceve anche il riferimento al vettore delle tracce con il suo indice e quello al vettore dei nodi e dei lati.

Sono possibili 7 diversi casi:

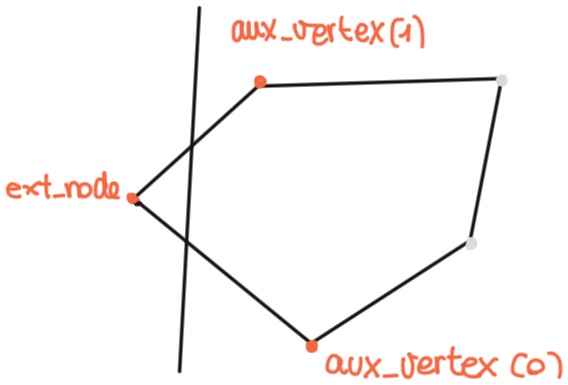
1. Ci sono due lati diversi intersecati
2. Ci sono intersezioni su lati paralleli alla traccia

2bis. C’è un lato o un nodo intersecato con un estremo della traccia

1. Ci sono due intersezioni, una con un lato e l’altra con un nodo
2. Ci sono due intersezioni con due nodi
3. C’è un nodo intersecato e la traccia finisce dentro il triangolo
4. C’è un lato intersecato e la traccia finisce dentro il triangolo



I 7 casi svolgono azioni simili, ma per chiarezza viene dato un cenno di ragionamento per ciascun caso:

1. Il poligono verrà suddiviso in due sottopoligoni. Vengono calcolate le coordinate delle intersezioni con i due lati. Viene cercata la presenza di questi nodi all’interno del vettore *node*, in caso di assenza vengono creati nuovi nodi. Viene controllato se i lati intersecati sono contigui o no:
   1. Se i lati sono contigui: verranno a crearsi un poligono e un triangolo formato dalle intersezioni e dal nodo in comune ai due lati intersecati. Vengono creati due vettori ausiliari *aux\_vert* in cui verranno salvati i nodi che formano i nuovi poligoni. Prima vengono inseriti i nodi del triangolo nel primo vettore. Poi viene controllato se il primo punto è il punto in comune tra i due lati intersecati, se il primo punto è uno degli estremi dei lati intersecati. In base ai controlli vengono già aggiunti dei punti al vettore dei vertici. Poi vengono aggiunti tutti i nodi del poligono escludendo quelli già eventualmente inseriti e controllando sempre se i nodi fanno parte di lati intersecati con conseguente inserimento anche dei punti di intersezione.
   2. Se i lati non sono contigui: vengono creati due vettori ausiliari *aux\_vert* in cui verranno salvati i nodi che formano i nuovi poligoni. Verrà usata una variabile flag per capire in quale poligono si stanno inserendo i vertici e in caso di passaggio dal lato intersecato viene cambiata questa flag. Innanzitutto viene controllato se l’ultimo lato sia uno dei lati intersecati, se sì aggiungo gli estremi del lato e il punto di intersezione nei poligoni corretti. Poi vengono aggiunti tutti i nodi del poligono escludendo quelli già eventualmente inseriti e controllando sempre se i nodi fanno parte di lati intersecati con conseguente inserimento anche del punto di intersezione e gestione della flag.

Non viene usata la funzione anticlockwise perché i nodi sono già in senso antiorario per costruzione. Viene usata la funzione edge\_poly per cercare se i lati del poligono esistono già, altrimenti vengono creati, poi i lati vengono salvati in due vettori ausiliari temp\_edge. Dopo aver creato i quattro vettori ausiliari con le informazioni sui lati e nodi dei nuovi poligoni, queste vengono inserite in due nuovi figli del poligono intersecato ripetendo due volte il seguente procedimento: viene creato un nuovo oggetto della classe GenericCellTree (l’indice del nodo dato al costruttore è il numero di nodi dell’albero), viene impostato il suo stato su attivo (true), viene aggiungo il puntatore al padre e il puntatore alla radice e vengono aggiunte seguendo il senso antiorario i puntatori ai nodi e ai lati del nuovo poligono; al padre viene aggiunto il puntatore al figlio e alla radice viene aggiunto lo stesso puntatore come foglia. Infine viene impostato lo stato del poligono intersecato su disattivo (false).

1. In questo caso la traccia è parallela ad uno o più lati. Sicuramente se il primo tipo di contatto è 3 o 4 anche il secondo, se c'è, è 3 o 4. Ci sarà solo un sottopoligono. I tipi di contatto 4 (nodo) non modificano la sottopoligonalizzazione quindi non vengono considerati, quindi vengono considerati solo quando gli estremi della traccia finiscono su un lato (3). Si distinguono ora 3 casi in base al numero di intersezioni con lati che ci sono:
   1. Ci sono solo intersezioni sui nodi: quindi il figlio sarà una copia identica del triangolo originale perché l’intersezione sul nodo non comporta un cambiamento di nodi o di lati.
   2. C’è un’intersezione su un lato: viene calcolata la coordinata dell’intersezione con il lato. Viene cercata la presenza di questo nodo all’interno del vettore *node*, in caso di assenza viene creato un nuovo nodo. Viene controllato se l’ultimo lato sia il lato intersecato, se sì aggiungo gli estremi del lato e il punto di intersezione. Poi vengono aggiunti tutti i nodi del poligono escludendo quelli già eventualmente inseriti e controllando sempre se i nodi fanno parte del lato intersecato con conseguente inserimento anche del punto di intersezione. Vengono analizzate le informazioni sui lati e sui nodi del nuovo poligono e poi inserite in un nuovo figlio del triangolo originale seguendo il procedimento descritto nel primo caso.
   3. Ci sono due intersezioni sui lati: vengono calcolate le coordinate delle intersezioni con i due lati. Viene cercata la presenza di questi nodi all’interno del vettore *node*, in caso di assenza vengono creati nuovi nodi. Si suddivide la procedura in tre ulteriori sotto-casi:
      1. I lati intersecati sono lo stesso lato: trovo le coordinate parametriche delle intersezioni rispetto al lato. Viene controllato se l’ultimo lato sia il lato intersecato, se sì aggiungo gli estremi del lato e i punti di intersezione nell’ordine corretto. Poi vengono aggiunti tutti i nodi del poligono escludendo quelli già eventualmente inseriti e controllando sempre se i nodi fanno parte del lato intersecato con conseguente inserimento anche dei punti di intersezione nell’ordine corretto.
      2. I lati intersecati sono diversi, ma contigui (e quindi hanno un nodo in comune): viene controllato se il primo punto è il punto in comune tra i due lati intersecati, se il primo punto è uno degli estremi dei lati intersecati. In base ai controlli vengono già aggiunti dei punti al vettore dei vertici. Poi vengono aggiunti tutti i nodi del poligono escludendo quelli già eventualmente inseriti e controllando sempre se i nodi fanno parte di lati intersecati con conseguente inserimento anche dei punti di intersezione.
      3. I lati intersecati sono diversi e non contigui: viene controllato se l’ultimo lato sia uno dei lati intersecati, se sì aggiungo gli estremi del lato e il punto di intersezione nell’ordine corretto. Poi vengono aggiunti tutti i nodi del poligono escludendo quelli già eventualmente inseriti e controllando sempre se i nodi fanno parte di lati intersecati con conseguente inserimento anche del punto di intersezione.

Vengono analizzate le informazioni sui lati e sui nodi del nuovo poligono e poi inserite in un nuovo figlio del triangolo originale seguendo il procedimento descritto nel primo caso.

2bis. In questo caso la traccia tocca i lati in un solo punto con un solo estremo. Quindi c'è solo un sottopoligono. I tipi di contatto 2 (nodo) non modificano la sottopoligonalizzazione quindi non verranno considerati, verranno considerati solo quando gli estremi della traccia finiscono su un lato. Quindi si distinguono due sotto-casi:

1. C’è solo l’intersezione sul nodo: quindi il figlio sarà una copia identica del triangolo originale perché l’intersezione sul nodo non comporta un cambiamento di nodi o di lati.
2. C’è un’intersezione su un lato: viene calcolata la coordinata dell’intersezione con il lato. Viene cercata la presenza di questo nodo all’interno del vettore *node*, in caso di assenza viene creato un nuovo nodo. Viene controllato se l’ultimo lato sia il lato intersecato, se sì aggiungo gli estremi del lato e il punto di intersezione. Poi vengono aggiunti tutti i nodi del poligono escludendo quelli già eventualmente inseriti e controllando sempre se i nodi fanno parte del lato intersecato con conseguente inserimento anche del punto di intersezione. Vengono analizzate le informazioni sui lati e sui nodi del nuovo poligono e poi inserite in un nuovo figlio del triangolo originale seguendo il procedimento descritto nel primo caso.
3. Il poligono verrà suddiviso in due sottopoligoni. Viene calcolata la coordinata dell’intersezione con il lato. Viene cercata la presenza di questo nodo all’interno del vettore *node*, in caso di assenza viene creato un nuovo nodo. Vengono creati due vettori ausiliari *aux\_vert*. Verrà usata una variabile flag per capire in quale poligono si stanno inserendo i vertici e in caso di passaggio dal lato o dal nodo intersecato viene cambiata questa flag. Innanzitutto viene controllato se l’ultimo lato sia uno dei lati intersecati, se sì aggiungo gli estremi del lato e il punto di intersezione nei poligoni corretti. Poi vengono aggiunti tutti i nodi del poligono escludendo quelli già eventualmente inseriti e controllando sempre se i nodi fanno parte di lato intersecato con conseguente inserimento anche del punto di intersezione e gestione della flag e se il nodo è quello intersecato con conseguente gestione della flag. Vengono analizzate le informazioni sui lati e sui nodi dei nuovi poligoni e poi inserite in due nuovi figli del triangolo originale seguendo il procedimento descritto nel primo caso.
4. Il poligono verrà suddiviso in due sottopoligoni. Verrà usata una variabile flag per capire in quale poligono si stanno inserendo i vertici e in caso di passaggio dal lato o dal nodo intersecato viene cambiata questa flag. Vengono aggiunti tutti i nodi del poligono controllando sempre se il nodo è quello intersecato con conseguente gestione della flag. Vengono analizzate le informazioni sui lati e sui nodi dei nuovi poligoni e poi inserite in due nuovi figli del triangolo originale seguendo il procedimento descritto nel primo caso.
5. In questo caso la traccia passa per un nodo e finisce dentro al poligono. Cerco l’intersezione, tra il prolungamento della traccia e un nodo o un lato del poligono, scorrendo tutti i lati del poligono che non hanno il nodo intersecato dalla traccia come estremo. Se gli estremi di un lato risultano dai due lati opposti della retta della traccia vuol dire che il lato è intersecato. Se un estremo di un lato risulta su un lato vuol dire che la retta della traccia interseca quel nodo. Il procedimento si divide in due sotto-casi distinti:
   1. L’intersezione del prolungamento della traccia è in un nodo del poligono: allora la sottopoligonalizzazione è simile al caso 4, ma con l’aggiunta dell’estremo della traccia che è interno al triangolo.
   2. L’intersezione del prolungamento della traccia è in un lato del poligono: allora la sottopoligonalizzazione è simile al caso 3, ma con l’aggiunta dell’estremo della traccia che è interno al triangolo.
6. In questo caso la traccia passa per un lato e finisce dentro al poligono. Cerco l’intersezione, tra il prolungamento della traccia e un nodo o un lato del poligono, scorrendo tutti i lati del poligono tranne il lato intersecato dalla traccia. Se gli estremi di un lato risultano dai due lati opposti della retta della traccia vuol dire che il lato è intersecato. Se un estremo di un lato risulta su un lato vuol dire che la retta della traccia interseca quel nodo. Il procedimento si divide in due sotto-casi distinti:
   1. L’intersezione del prolungamento della traccia è in un nodo del poligono: allora la sottopoligonalizzazione è simile al caso 3, ma con l’aggiunta dell’estremo della traccia che è interno al triangolo.
   2. L’intersezione del prolungamento della traccia è in un lato del poligono: allora la sottopoligonalizzazione è simile al caso 1 (ovvero con anche la distinzione dei casi in cui i lati siano contigui o no), ma con l’aggiunta dell’estremo della traccia che è interno al triangolo.

Ora tutte le foglie intersecate del triangolo in considerazione sono state sottopoligonate. I rispettivi figli sono stati aggiunti all’elenco delle foglie. Tramite un ciclo for e una resize vengolo eliminati i poligoni intersecati e che quindi non sono più foglie, così da lasciare la mesh in uno stato consistente.

# Salvataggio dei dati

Vengono stampate a video tutte le informazioni riguardanti le intersezioni (tipo, elemento e coordinata) e le informazioni riguardanti la sottopoligonalizzazione (nodi dei sottopoligoni)

Tramite la chiamata della funzione output vengono creati dei file con:

1. lo stato corrente del vettore dei nodi,
2. lo stato corrente del vettore dei lati,
3. lo stato corrente della triangolazione con le corrispettive sottopoligonalizzazioni,
4. l’elenco delle intersezioni con la traccia.

Per sapere per quali sottopoligoni toccano la traccia lungo il suo tragitto basta guardare i figli dei poligoni intersecati.

Per impostare il numero di cifre significative nelle coordinate dei nodi viene usata la funzione setprecision(2) della libreria <iomanip>.

I valori delle coordinate sono calcolate mediante la risoluzione di un sistema lineare, quindi a causa della precisione di macchina il valore 0 viene rappresentato con valori minori di 10^-14. Per questo nel salvataggio delle coordinate controllo se ci sono valori inferiori a 10^-14 e quelli vengono stampati come 0.

Nella stampa delle intersezioni controllo se il tipo di contatto è 5. In quel caso stampo come coordinata -1 e come identificativo dell’elemento identificato NULL.

1. FILE quad.out.2.node

%d %d numero di nodi, numero della dimensione (2D, 3D)

%d %f %f numero nodo, coordinata x del nodo, coord y

%d %f %f

...

1. FILE quad.out.2.edge

%d numero di lati

%d %d %d numero lato, nodo iniziale, nodo finale

%d %d %d

...

1. FILE quad.out.2.ele

%d numero di triangoli

%d %d %d numeri nodi

%d %d %d

...

%d %d %d %d numero grandfather, numero father, numero figlio, numero di nodi

%d %d ... elenco dei nodi del poligono

%d %d %d %d

%d %d ...

1. FILE quad.out.2.interpoly

%d numero di poligoni intersecati

%d %d %d numero grandfather, numero poligono, numero di intersezioni

%d %d %f tipo di contatto, lato/nodo intersecato, coordinata parametrica di intersezione

...

%d %d %d numero grandfather, numero poligono, numero di intersezioni

%d %d %f tipo di contatto, lato/nodo intersecato, coordinata parametrica di intersezione

...

A questo punto si azzerano i vettori intersect\_triangle e intersect\_poly e riparte tutto il procedimento con l’utilizzo della traccia successiva mantenendo però attive le modifiche alla mesh effettuate dalle tracce precedenti.

# Funzione insert

La funzione *int Insert(int j, int i, int p, double x1, double x2, vector<GenericEdge\*>& edge, vector<GenericIntersect\*>& intersect\_triangle, int num\_edge)* riceve in ingresso il numero del lato intersecato (j) e della traccia in considerazione (i), il numero di triangoli intersecati fino ad ora più uno (p), la coordinata parametrica del lato (x1) e quella della traccia (x2), il riferimento al vettore contenente i puntatori ai lati (edge), quello ai triangoli intersecati (intersect\_triangle) e il numero di lati della triangolazione originale (num\_edge). La funzione restituisce il numero di triangoli intersecati fino ad ora più uno.

Tramite due cicli for annidati si salvano i triangoli (uno o due) che hanno quel lato. Poi si salvano i triangoli intersecati controllando che non siano già stati inseriti e se sono già stati inseriti controlla che non sia già presente lo stesso nodo.

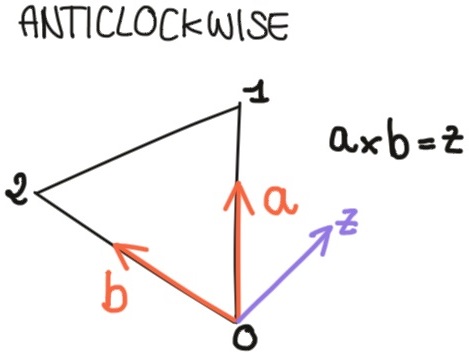
Sono suddivisi i 3 casi: intersezione con il lato (non agli estremi), intersezione nel primo estremo e intersezione nel secondo estremo.

# Funzione check\_node

La *funzione void check\_node(int index, int j, double\* n, int i, int num\_edge, vector<GenericEdge\*>& edge, vector<const GenericCell\*>& flag, vector<GenericDomain1D\*>& tracce)* riceve in ingresso l’indice dell’estremo del lato che non è sulla traccia (index), il numero del lato (j) e della traccia (i) in considerazione, la normale alla traccia (n), il numero di lati della triangolazione originale (num\_edge), il riferimento al vettore contenente i puntatori ai lati (edge), quello in cui salveremo temporaneamente i triangoli intersecati (flag) e quello alle tracce (tracce).

Tramite due cicli for annidati si salvano i triangoli (uno o due) che hanno quel lato. Viene analizzato separatamente ogni triangolo che ha quel lato, viene trovato il nodo che non è sul lato tagliato dalla traccia. Tramite un ciclo for su tutti i lati viene trovato il lato che ha per estremi i due nodi non intersecati. Tramite la funzione scalar\_product controllo se il lato che ha per estremi i due nodi non intersecati ha gli estremi ai due lati opposti della traccia allora quel triangolo è intersecato, altrimenti non lo considero.

# Funzione anticlockwise

La funzione *void anticlockwise(vector<const GenericPoint\*>& aux\_vert)* riceve in ingresso il riferimento ad un vettore di puntatori a nodi, questi rappresentano i vertici di un poligono. Per controllare se i vertici sono in senso orario o antiorario uso il prodotto vettoriale. Essendo su un piano le componenti x e y del prodotto vettoriale verranno nulle, quindi viene calcolata solo la componente z. Il prodotto vettoriale viene fatto tra i vettori che congiungono il primo nodo con due nodi consecutivi (ciclando su tutti i nodi escludendo il primo). Se la componente z del prodotto vettoriale è negativa vuol dire che i nodi sono in senso orario, altrimenti sono in senso antiorario. Se i nodi sono in senso antiorario allora viene invertito l’ordine dei vertici scambiando il primo con l’ultimo, il secondo con il penultimo e così via fino a metà dell’elenco dei vertici.