Triangolazione 2D su MATLAB

# Punto 1.

Per ciascuna traccia è stato individuato l'elenco dei triangoli tagliati dalla traccia.

1. Innanzitutto è stata creata una funzione chiamata “**Input\_file\_Triangolazione2D**” il cui scopo è quello di aprire i file dati e creare delle matrici global che contengano tutte le informazioni necessarie. In caso di errore nell’apertura di un qualche file, il programma segnala l’errore e si blocca.
2. In secondo luogo è stata realizzata la struttura **TOUCH**, anch’essa globale. Il suo scopo è quello di indicare, per ognuno dei nodi, i nodi ad esso collegato tramite un qualche lato e i triangoli cui uno dei vertici è quel nodo.
3. Poi si apre il ciclo i che scorre le tracce date, poiché le stesse operazioni saranno iterate per ognuna delle tracce date.

Subito dopo è inizializzata la struttura principale del progetto, che contiene i risultati. Essa, denominata “**intersect\_triangle**” ed è globale.

intersect\_triangle(i) apre la traccia numero i

intersect\_triangle(i).array(m) analizza il m-esimo triangolo intersecato numero

all’interno sono presenti i seguenti campi per i triangoli tagliati:

-num\_tri : individua il numero del triangolo tagliato

-coord: salva la coordinata parametrica del punto di intersezione rispetto alla traccia. (se ad esempio abbiamo intersezione nel suo punto medio sarà salvato il numero 0.5)

-intersect\_edge: specifica per ogni triangolo quali lati sono tagliati.

-vertex: contiene per ogni triangolo, tutti i vertici del triangolo più i nuovi punti di intersezione con la traccia.

-node\_pass: indica se la traccia ha attraversato un nodo e se sì, quale nodo;

-poly: una nuova sottostruttura che contiene i sottopoligoni di ogni triangolo

-tri: una nuova sottostruttura che contiene i sottotriangoli

-share: una sottostruttura che contiene per ogni triangolo tagliato, i triangoli a lui vicini e i nodi che con questi condivide.

NB: nel caso la traccia passi per un vertice del triangolo, il vertice sarà inserito i node\_pass ma non saranno salvati i lati che condividono quel nodo. Quindi in intersect\_edge si considerano i lati attraversati non nel propri estremi.

P, inizializzata a 1, è una variabile che indica il primo posto libero nella struttura intersect per inserire i prossimi triangoli tagliati.

N è la normale alla traccia.

1. Successivamente si apre il ciclo j che scorre tutti i lati in edge per trovare tutte le intersezioni con la traccia numero i.

Per analizzarli prima di tutto viene chiamata la funzione **SCALAR** : questa funzione calcola il prodotto scalare tra la normale alla traccia e il vettore congiungente l'estremo 1 della traccia e i vari vertici del lato. Restituisce s1 e s2, calcolati rispetto al nodo d’inizio e di fine del lato j.

1. 5a) Se s1 e s2 sono discordi (senza considerare lo zero) continuo l’analisi. Il fatto che s1 e s2 siano discordi indica che i due vertici del lato si trovano uno nel primo dei due semipiani individuati dalla retta passante per la traccia, e uno nell’altro semipiano.

Troviamo il vettore verticale x che contiene le coordinate parametriche s e t date dall’intersezione traccia-lato, riguardanti rispettivamente il lato (x(1)=s) e la traccia (x(2)=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 ad **INSERT** le coordinate parametriche affinché questa funzione inserisca in intersect tutti i triangoli che condividono quel lato in num\_tri (trovati grazie alla funzione intersect e la struttura touch), la coordinata parametrica in coord e il lato in intersect\_edge. La funzione si occupa anche di incrementare il contatore p.

5b) s1 e s2 sono entrambi nulli: in questo caso non può provare a calcolare s e t perché lato e traccia sono paralleli, in particolare appartengono alla stessa retta. Si hanno infiniti punti di intersezione. Quindi si procede diversamente: sono 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. Sono poi stati analizzati i diversi casi:

if (ki>=0+toll) && (kf<=1-toll)

se ki e maggiore di 0 (non compreso) e kf minore di 1 (non compreso) significa che il lato è completamente interno alla traccia. In questo caso il lato NON è considerato tagliato quindi non è stato inserito nessun comando.

elseif kf<=0+toll || ki>=1-toll

se il lato inizia dopo la fine della traccia o finisce dopo la fine della traccia quindi il lato e esterno alla traccia e non è tagliato.

elseif (kf>1-toll && ki<1-toll) || (ki<0+toll && kf>0+toll)

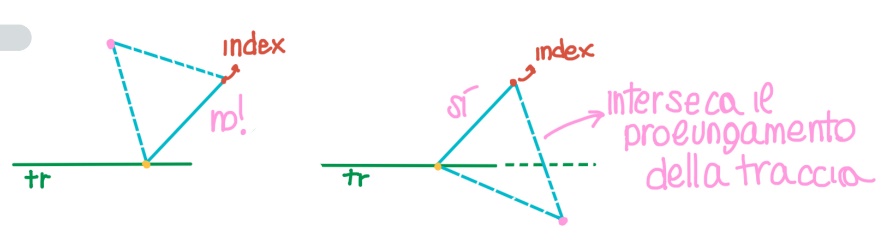
se il lato inizia prima della fine della traccia abbiamo già la sicurezza, avendo escluso gli altri casi, che kf sarà maggiore o uguale a 1. Analogamente se il lato finisce dopo l’inizio della traccia, ki non potrà essere maggiore di 0 (altrimenti sarebbe già entrato nel primo caso).

Quindi c’è la sicurezza che almeno un estremo è da inserire. Per sfruttare la funzione insert, che controlla che s=x(1) sia un valore accettabile (fra 0 e 1), bisogna passare s1. tuttavia in questo caso non si calcola x(1), che verrebbe inf, quindi si può passare alla

funzione un qualunque valore di x(1) accettabile e come t=x(2) o 0 o 1, a seconda se analizziamo ki o kf. Ad esempio x(1)=0.5 va bene come valore di s.

Se il lato è intersecato esattamente in un estremo allora bisognerà inserire node\_pass e non sarà considerato intersecato il lato. Quindi bisogna passare alla funzione un valore di s non compreso fra 0 e 1, affinché non sia accettabile. Va bene quindi passare x(1)=-1 se ki=0 e/o kf=1.

5c) o s1 o s2 nulli (non entrambi, altrimenti sarebbe già entrato prima): lato non parallelo alla traccia. Un vertice del lato appoggerà sulla retta a cui appartiene la traccia. Possiamo calcolare s e t, chiamando poi la funzione **CHECK\_NODE.**

Check\_node è una funzione che riceve in input il nodo che non si trova sulla traccia, il lato analizzato (j), il numero di traccia, la normale alla traccia e il numero di lati. Per prima cosa ricerca i due(o uno) triangoli che hanno il lato j in comune e trova quale è il nodo che si trova sulla traccia mettendolo in node\_on\_trace. Vengono scorsi i triangoli trovati e si ricercano i due nodi che non appartengono al nodo j (uno di un triangolo e uno dell’altro). Per ognuno dei due punti trovati ricerca quale sia il lato che collega tale punto all’estremo del lato che non si trova sulla traccia. Trovato il lato viene richiamata la funzione scalar che controlla gli estremi del lato precedentemente trovato in modo che, se i due prodotti scalari siano discordi, il lato e quindi il triangolo, sono tagliati. Check\_node restituisce il vettore flag che contiene l’indice del triangolo tagliato che verrà poi inserito nella struttura intersect\_triangle.

1. Finito di analizzare tutti i lati si controlla che sia stato trovato almeno un triangolo. Se p vale ancora 1 significa che non sono state trovate intersezioni, quindi viene chiamata la funzione **TRACCIA\_ALL\_INNER**. Questo controllo si basa sulla considerazione che la traccia non può essere completamente esterna all’insieme di triangoli, quindi alla fine del programma almeno un triangolo sarà tagliato.

Questa funzione scorre tutti i triangoli. Per ognuno di essi calcola l’area: la metà del prodotto vettoriale fra due dei tre lati. Lavorando in due dimensioni si è posta uguale a zero la componente lungo z. Non essendoci intersezioni con i lati entrambi gli estremi della traccia sono interni allo stesso triangolo. Si è quindi diviso il triangolo in tre sottotriangoli, ciascuno individuato dai due dei tre vertici del reale triangolo e da un estremo della traccia (è indifferente quale). Si calcola l’area dei tre sottotriangoli e si sommano. Se l’estremo della traccia è esterno al triangolo sicuramente quest’ultima area sarà maggiore della prima calcolata. Quindi il controllo di basa sul vedere per quale triangolo le due aree coincidono. Trovato il triangolo si inserisce il triangolo lasciando -1 la coordinata (non essendoci intersezione con la traccia) e ovviamente non sono inseriti né il vertice né il lato. In questa funzione è anche analizzato il secondo punto del progetto in questo caso particolare.

Finito ciò si ottiene in intersect\_tri quanto chiesto dal primo punto. Come ultima cosa aggiungiamo alla matrice dei nodi, tutti i nuovi nodi che si sono venuti a creare dalle intersezioni traccia-lati e contemporaneamente aggiungiamo in intersect\_triangle(i).array(m).vertex i vertici del triangolo stesso e i nuovi nodi di intersezione. Nel caso uno di questi nuovi punti di intersezione corrisponda proprio ad un vertice del triangolo, aggiorniamo node\_pass inserendo il numero del nodo per cui passa la traccia.

# Punto 2.

Per ciascuna traccia salvare l'elenco dei triangoli che condividono almeno un vertice con un triangolo tagliato e per questi memorizzare anche quali vertici sono condivisi con un triangolo tagliato.

Per risolvere il punto due è stata creata la funzione **point2\_2D**:

1. Inizialmente si apre il ciclo m che scorre i triangoli intersecati, e per ogni triangolo è inizializzata la parte della struttura che si occupa dei triangoli vicini: intersect\_triangle(i).array(m).share

intersect\_triangle(i).array(m).share(z).share\_num\_tri: salva il numero dello z-esimo trinagolo vicino all’m-esimo intersecato.

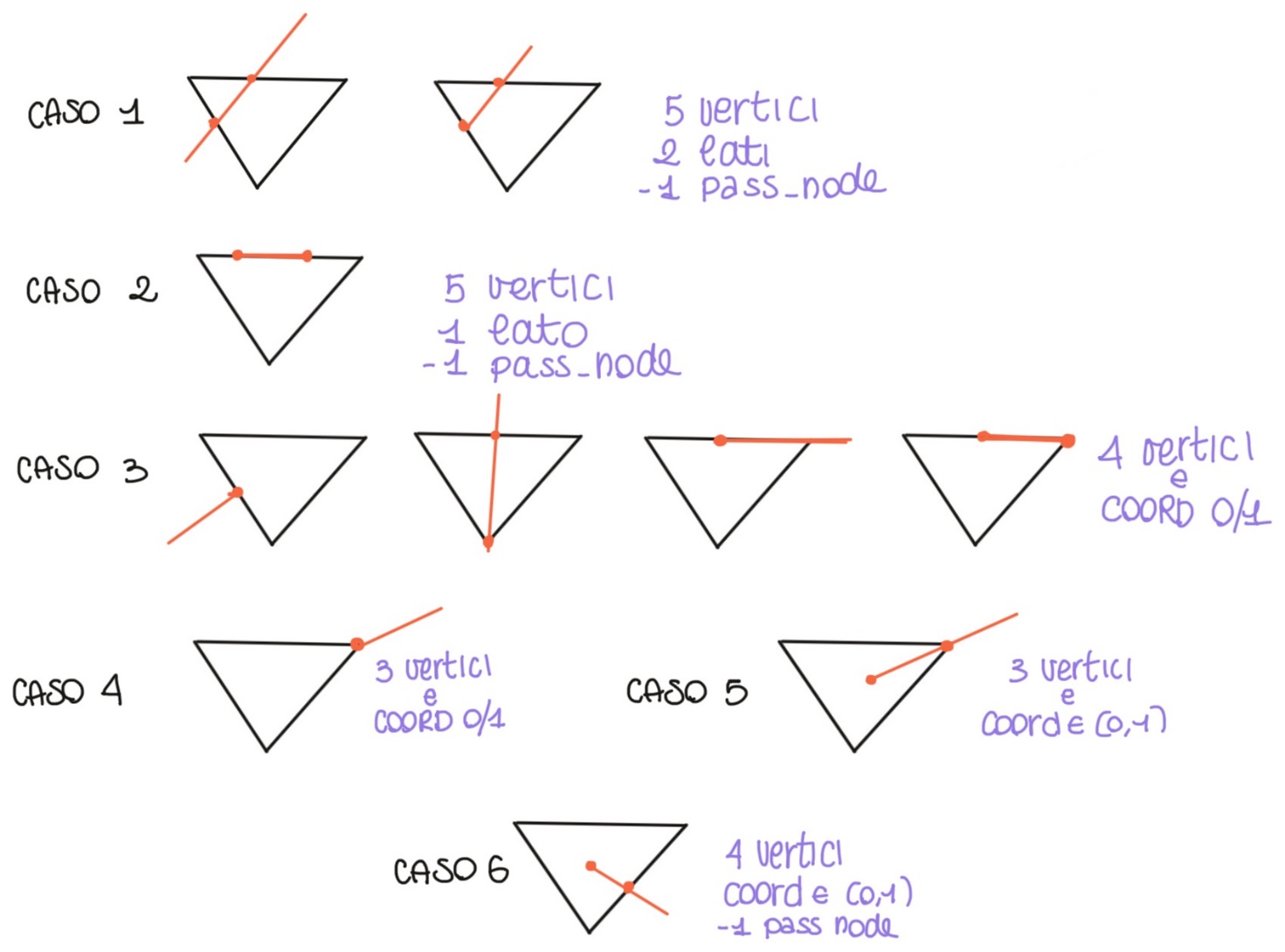
intersect\_triangle(i).array(m).share(z).share\_nodes\_tri: sono salvati i nodi in comune fra m-esimo tagliato e lo z-esimo vicino.

1. Scorro i vertici dell’m-esimo triangolo tagliato tramite j e per ognuno dei nodi scorro tutti i triangoli con k. Dopo aver verificato che non si tratti dello stesso triangolo, controllo se uno dei vertici del triangolo k coincide con il j-esimo del triangolo intersecato. Se tale condizione è verificata si aggiorna la struttura intersect\_triangle. In particolare se il triangolo trovato k è già stato inserito per intersect\_triangle(i).array(m) si aggiornato solo i nodi, per non aggiungerlo nuovamente.

# Punto 3. e 4.

Per ciascun triangolo tagliato creare i sotto-poligoni ottenuti tagliando il triangolo con la traccia. Se la traccia termina nel triangolo prolungare il taglio fino ad un lato/punto del triangolo.

1. La sottopoligonizzazione (e successivamente la sottotriangolazione) sono da effettuare solamente se la traccia non è totalmente interna a un triangolo, quindi inizialmente controllo che p sia maggiore di 1.
2. Scorro i triangoli tagliati tramite m e, per ognuno dei tagliati, mi salvo in **size\_vertex** il numero dei punti totali del triangolo e in **size\_intersect\_edge** il numero di lati tagliati.
3. Sono stati individuati 6 casi diversi, ognuno analizzato separatamente:



1. 5 vertici e 2 lati:

* **Vert5\_Edge2\_poly**: si formano un triangolo è un poligono. Dapprima si salvano le coordinate del triangolo: si trova il vertice in comune ai due lati tagliati e si prendono i due nuovi punti formatisi e i salvano in aux\_vert\_poly, vettore ausiliario. Si salvano anche le coordinate cartesiane di tali punti in aux\_vert\_poly\_coord. A questo punto viene chiamata la funzione **ANTICLOCKWISE** che ordina i punti di un poligono qualsiasi in senso antiorario: controlla quindi che non siano in senso orario, altrimenti li ordina in senso antiorario tramite due funzioni implementate in MATLAB. Una volta terminata la funzione si salvano i nodi in senso antiorario nella struttura intersect\_triangle(i).array(m).poly(1).vert\_poly.

In seguito si passa al salvataggio del quadrilatero: si salva in ext\_vert il vertice del triangolo che abbiamo già usato per formare il triangolo precedente, in questo modo possiamo ricercare il primo vertice del poligono prendendo il primo lato intersecato e trovando quale è l’estremo che non è salvato in ext\_vert. Poi viene preso il nodo di intersezione tra il primo lato e la traccia, vedendo quale dei due nuovi punti di intersezione giace sul lato considerato, di conseguenza viene preso l’altro punto di intersezione e per ultimo il terzo vertice del triangolo mai considerato. Questo poligono viene quindi salvato in intersect\_triangle(i).array(m).poly(2).vert\_poly.

* **Vert5\_Edge2\_tri**: controlla il numero di punti di poly(j).vert\_poly: se sono tre stiamo analizzando il triagolo quindi deve essere lasciato invariato e salvato in intersect\_triangle(i).array(m).tri(end+1).vert\_tri. Altrimenti se sono quattro, salvo in due triangoli diversi i vertici in poly(j).vert:poly (v1 v2 v4) e (v2 v3 v4) perché so già che sono corretti e in senso antiorario.

1. 5 vertici e 1 lato:

* **Vert5\_Edge1:** dobbiamo salvare tre triangoli. Troviamo il nodo del triangolo che non appartiene al lato intersecato (ext\_vert). Si considerano poi i quattro punti che si trovano tutti sullo stesso lato: si calcolano le distanze tra il primo estremo del lato e i due estremi della traccia. In questo modo possiamo salvare nella matrice ausiliaria aux\_vertex\_poly i tre triangoli, salvando nella colonna 1 per tutte le righe ext\_vert; poi controllando quale delle due distanze calcolate prima sia minore si salvano di conseguenza gli altri nodi del triangolo:

C

B

D

A

Se dist(AB)<dist(AC) ----> -------->

E

Se dist(AC)<dist(AB) ----> così nel disegno B e C sarebbero invertite

Una volta ottenuta tale matrice ogni riga viene passata ad anticlockwise per ordinare in triangoli in senso antiorario e poi salvarli nella struttura. La triangolazione è quindi già questa.

D

C

B

1. 4 vertici e la traccia non finisce dentro il triangolo:

tale condizione si verifica controllando che node\_pass sia diverso da -1 oppure che se node\_pass è -1, qundi nessun vertice del triangolo è intersecato, la coordinata parametriche dell’intersezione della traccia deve essere o 1 o 0 cosicché sappiamo che la traccia si è fermata su di un lato e non ha proseguito oltre.

A

* **Vert4\_noInt\_poly:** trova sempre il vertice che non si trova sul lato intersecato (ext\_vert). Si prendono quindi i due triangoli che hanno come vertici (A,B,C) e (A,C,D), dove A=ext\_vert e C=intersect\_triangle(i).array(m).vertex(4). La triangolazione anche qua è già òa poligonizzazione stessa.

1. 3 vertici e la traccia finisce su un vertice:

questa condizione si verifica quando ho tre nodi in vertex e l’unica coordinata parametrica di intersezione è 1 o 0.

In questo caso poligonizzazione e sottotriangolazione non sono da implementare in quanto entrambi sono il triangolo di partenza.

**N.B** Nel progetto, nella trattazione di questo caso, abbiamo scelto di considerare tagliati solo quei triangoli in cui il prolungamento della traccia intersecasse il lato opposto al nodo per cui passa; infatti i controlli di check\_node e del caso in cui lato e traccia siano paralleli, considerano solo il caso pocanzi detto.

1. 3 vertici e la traccia finisce nel triangolo:

C

questa condizione si verifica quando oltre ad avere 3 vertici in vertex la coordinata parametrica di intersezione tra lato e traccia è compresa tra 0 e 1 esclusi.

* **Vert3\_siInt:** la poligonizzazione verrà implementata prolungando la traccia fino a lato opposto.

D

E

In questo caso ext\_vert è esattamente node\_pass; cerchiamo quindi i vertici del triangolo per cui non passa a traccia. Si ricerca l’intersezione tra il prolungamento della traccia e il lato opposto ad ext\_vert, si trovano le coordinate cartesiane e si aggiunge come nuovo nodo in node. I triangoli che si vengono a creare sono quindi gli stessi del caso 3.

B

A

La sottotriangolazione viene invece implementata collegando l’estremo della traccia interna al triangolo ai vertici del triangolo. Si ricerca quale dei due estremi della traccia sia quello interno al triangolo guardando la coordinata parametrica dell’intersezione con il prolungamento della traccia: se è minore di toll vuol dire che è il primo estremo interno altrimenti il secondo; tale nodo viene inserito come nuovo nodo in node. Vengono quindi salvati i nuovi triangoli nella matrice

ordinati i vertici in senso antiorario passando riga per riga alla funzione anticlockwise, e poi inseriti nella struttura intersect\_triangle(i).array(m).tri.

1. 4 vertici e la traccia finisce nel triangolo:

condizione che si verifica quando si hanno 4 vertici, node\_pass=-1 e la coordinata parametrica di intersezione è compresa tra 1 e 0 esclusi.

* **Vert4\_siInt:** In questo caso la traccia verrà prolungata e sarà trovata l’intersezione con l’altro lato, tenendo in considerazione che il nuovo punto di intersezione potrebbe essere un vertice del triangolo. Ext\_vrt contiene il vertice che non è sul alto su cui passa la traccia mentre in diff\_edge ci sono gli altri due. Si trovano quindi le coordinate parametriche delle intersezioni del prolungamento della traccia con i due lati non tagliati. Se almeno una delle coordinate parametriche riferite al lato (x1(1) e x2(1)) è zero vuol dire che sta attraversando un vertice del triangolo e quindi si richiama la funzione Vert4\_noInt\_poly. Nel caso in cui x1(1) sia compreso tra 1 e 0 esclusi sarà il primo lato ad essere intersecato altrimenti il secondo. In entrambi i casi si salva prima il triangolo che si è formato ordinando i nodi in senso antiorario, e poi si salva il quadrilatero. Tale poligono viene creato con lo stesso criterio usato nel caso 1 anche se qui sappiamo già quali nodi stanno su quali nodi per le ricerche effettuate in precedenza. Anche qui i nodi vengono ordinati in senso antiorario. La sottotriangolazione viene effettuata ottenendo quattro triangoli, frutto del collegamento dell’estremo della traccia interno al triangolo con i tre vertici e del pezzetto di traccia interno al triangolo. I nodi vengono salvati in questa matrice i nodi vengono quindi oridnati in senso antiorario e ogni triangolo viene salvato nella struttura.

A

C

D

B

E

# Output dei dati

A questo punto le richieste sono state completate e il programma termina dopo stampato in vari file tali dati:

FILE quad.2.node

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

%2d %5f %5f numero nodo, coordinata x del nodo, coord y

%2d %5f %5f

...

%--------------------------------------------------------------------

FILE intesect\_triangle.i.base i = numero della traccia

%d %d numero della traccia, numero di triangoli intersecati

%d numero del triangolo intersecato

%d %f ... numero di intersezioni, coordinate parametriche

| dell'intersezione

%d %d numero del 1° lato intersecato, numero del 2° lato

| intersecato (se non c'è viene messo -1)

%d vertice per cui passa la traccia, se non c'è viene messo -1

...

%--------------------------------------------------------------------

FILE intesect\_triangle.i.share i = numero della traccia

%d %d numero della traccia, numero di triangoli intersecati

%d %d numero del triangolo intersecato, numero di triangoli che

| condividono almeno un vertice

%d %d %d numero del triangolo che condivide almeno un vertice,

| numero del primo nodo in comune, numero del secondo

| nodo in comune (-1 se c'è solo un vertice in comune)

...

%--------------------------------------------------------------------

FILE intesect\_triangle.i.poly i = numero della traccia

%d %d numero della traccia, numero di triangoli intersecati

%d %d numero del triangolo intersecato, numero di poligoni

%d %d ... numero di vertici del poligono, numeri dei nodi, ...

...

%--------------------------------------------------------------------

FILE intesect\_triangle.i.tri i = numero della traccia

%d %d numero della traccia, numero di triangoli intersecati

%d %d numero del triangolo intersecato, numero di sottotriangoli

%d %d %d numero dei vertici del sottotriangolo