UNIVERSITÀ DEGLI STUDI DI PADOVA
DIPARTIMENTO DI INGEGNERIA INDUSTRIALE
Department of Industrial Engineering

Corso di Laurea Triennale in Ingegneria Aerospaziale

Identificazione del percorso di cricca con simulazioni basate sulla teoria *Peridynamics*

Relatore: prof. MIRCO ZACCARIOTTO
Correlatore: prof. UGO GALVANETTO

Laureando: ALBERTO SCOMPARIN
1118778

ANNO ACCADEMICO 2017-2018

Ai miei genitori, per avermi sempre sostenuto
Ai miei fratelli e parenti, che non dubitano mai di me
Ai miei amici, che sono al mio fianco nelle sfide
e le rendono un gioco
A chi indaga il mondo
con curiosità inesauribile

Indice

Sommario

Il corpo di questa stesura è composto da tre parti principali: teoria *Peridynamics*, stesura dell'algoritmo, e risultati. In particolare il contenuto è così suddiviso:

1. Introduzione: come il sottoscritto è entrato a contatto con l'argomento;
2. Teoria *Peridynamics*: breve spiegazione della teoria alla base dell'implementazione;
3. *Stesura dell'algoritmo*: cuore dell'elaborato, tratta del percorso seguito per la stesura e il test del codice;
4. Simulatore: viene delineato lo script finale commentando brevemente il codice;
5. Esperimento: viene illustrato l'esperimento con cui è stato testato l'algoritmo;
6. Discussioni: questioni non discusse altrove che meritano menzione;
7. Risultati;
8. Conclusioni

Capitolo 1

Introduzione

Il presente elaborato è frutto di un'interessante proposta del professor Zaccariotto di svolgere un algoritmo Matlab che permetta di visualizzare l'evoluzione di una cricca con simulazioni basate su una teoria recente nell'ambito della Meccanica dei Solidi, detta *Peridynamics*.

Già introdotto al Metodo degli Elementi Finiti *FEM* nei corsi di Aerodinamica e Complementi di Strutture Aerospaziali, la proposta è stata subito incalzante.

Nel corso del laboratorio di Aerodinamica, il Metodo degli Elementi Finiti è stato applicato alla fluidodinamica computazionale *CFD* seguendo un percorso dai principi primi: è stato assegnato infatti il compito di implementare un algoritmo da zero - basandosi sulle equazioni di Poisson e Laplace applicate ad elementi isoparametrici - che simulasse l'andamento di un fluido all'interno della girante di una turbomacchina. I risultati sono stati soddisfacenti e la disciplina mi è rimasta piacevolmente impressa.

Nel corso dell'esame di Complementi invece, un approccio manuale del *FEM* è stato applicato al problema del telaio iperstatico, riconducendo telai semplici ad una matrice di rigidezza che rappresentasse la relazione tra spostamenti e carichi. Nel laboratorio di Complementi è stata affrontata la simulazione *FEM* al calcolatore con il motore *Patran* e il software *Nastran*.

La simulazione di fenomeni fisici al calcolatore si è rivelata un aspetto interessante: l'implementazione di una teoria recente - alternativa e complementare al Metodo degli Elementi

Finiti - è risultata una conseguente conclusione del percorso di studi.

Questa stesura, essendo una tesi per la laurea Triennale, si presenta dapprima come un'introduzione ad un aspetto basilare e primitivo del grande spettro di argomenti legati alla *Peridynamics*; successivamente come un tentativo di comprenderne i principi e il funzionamento; in ultima analisi come una presentazione dei risultati ottenuti.

Capitolo 2

Teoria *Peridynamics*

Nascita

Nella disciplina della Meccanica dei Solidi il problema di prevedere la progressione di una cricca lungo un materiale risulta complesso se indagato con la Meccanica Classica ed equazioni parziali alle differenze *PDE*, in particolare per quanto riguarda la definizione delle condizioni al contorno per un dominio che cambia man mano che le cricche si aprono. Per approfondimenti si rimanda a Lapidus and Pinder (2011), Hillerborg, Modéer, and Petersson (1976), Belytschko and Black (1999) e Bolander Jr and Saito (1998).

Come tentativo di risolvere tale complessità - alla luce di precedenti risultati in Streit and Finnie (1980), Zhou et al. (1996), Ravi-Chandar (1998), Ramulu and Kobayashi (1985) e Xu and Needleman (1994) - in Silling (2000) viene formulata una teoria basata su equazioni integrali facilmente implementabili numericamente al calcolatore. Tale metodo risulta semplice quanto efficace quando vengono a formarsi discontinuità spontanee: queste risultano parte della soluzione, non una complessità che ridefinisce il sistema.

Esempi di applicazione del Metodo degli Elementi Finiti assieme alla teoria di Silling si approfondiscono ad esempio in Macek and Silling (2007), Gerstle, Sau, and Silling (2007) e Kilic and Madenci (2010).

Fondamenti

La teoria sviluppata da Silling è nominata da lui stesso *peridinamica*, dalle radici in greco corrispondenti a *vicino* e *forza*. Essa si basa infatti su un concetto di prossimità, detto orizzonte, entro il quale una particella interagisce con altre particelle di un corpo.

Mentre la Meccanica Classica si concentra principalmente su particelle confinanti a due a due, riferendosi alle derivate spaziali delle loro posizioni, la teoria peridinamica entra nel gruppo di quelle cosiddette non-locali, poiché più particelle a una certa distanza possono interagire tra loro.

Il fenomeno fisico può essere interpretato come in figura: ogni particella dell'oggetto *vede* una porzione dell'oggetto entro una certa distanza, con la quale interagisce.

L'equazione di equilibrio di ogni particella, identificata dalla propria posizione iniziale $\mathbf{x}_i$ rispetto a un sistema di riferimento prefissato, è data dalla seguente:

$$\rho \cdot \mathbf{a}(\mathbf{x}_i, t) = \int_{H_{x_i}} \mathbf{f}\left(\mathbf{u}(\mathbf{x}_j, t) \; - \; \mathbf{u}(x_i, t) \; , \quad \mathbf{x}_j - x_i\right) \; \cdot \; dV_j \; + \; \mathbf{b}(x_i, t) \tag{2.1}$$

Dove $\mathbf{a}$ e $\mathbf{u}$ sono rispettivamente campi vettoriali di accelerazione e spostamento della particella $\mathbf{x}_i$ al momento t, mentre ρ è la densità del materiale. L'intorno $H_{\mathbf{x}_i}$ è un cerchio di raggio δ definito attorno alla particella denominato *orizzonte*, dV_j è il volume equivalente della particella $\mathbf{x}_j$ interna all'orizzonte e $\mathbf{b}$ è il carico esterno per unità di volume applicato alla particella $\mathbf{x}_i$.

Per riferire in termini vettoriali la posizione relativa tra due particelle, si assegna a due vettori ξ e η i valori di:

$$\xi = \mathbf{x}_j - \mathbf{x}_i \qquad\qquad \eta = \mathbf{u}(\mathbf{x}_j, t) - \mathbf{u}(\mathbf{x}_i, t) \tag{2.2}$$

Per ogni istante t la posizione relativa tra le due particelle sarà data da $\xi + \eta$.

La forza $\mathbf{f}$, detta *pairwise force*, per particolari materiali presenta interessanti proprietà descritte in Silling (2000), tra le quali simmetria e conservazione di momento angolare, quest'ultimo dovuto al fatto che la forza è definita dalla sola posizione attuale delle particelle.

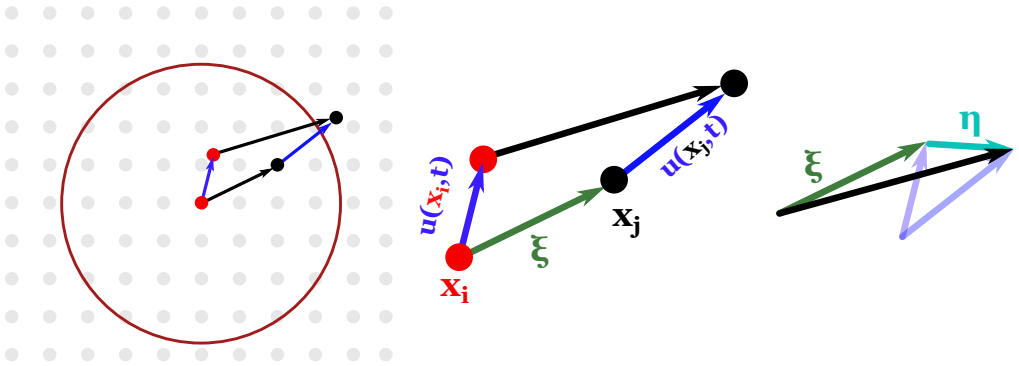

Figura 2.1: Posizioni relative dei nodi nell'orizzonte. In blu sono indicati i vettori spostamento.

Linearizzazioni

Per poter impiegare agevolmente le equazioni peridinamiche, si assumono alcune proprietà del materiale di cui è composto l'oggetto e la natura delle forze in gioco. Ciò comporta la possibilità di linearizzare le equazioni introdotte e implementarle nel codice.

Un materiale si definisce *microelastico* se la forza $\mathbf{f}$ deriva da un potenziale dato da:

$$\mathbf{f}(\eta, \xi) = \frac{\partial \omega(\eta, \xi)}{\partial \eta} \tag{2.3}$$

Un potenziale lineare microelastico può essere definito come:

$$\omega(\eta, \xi) = \frac{c(|\xi|)|\xi| s^2}{2} \tag{2.4}$$

Dove per s si intende l'allungamento relativo:

$$s = \frac{|\xi + \eta| - |\xi|}{|\xi|} \tag{2.5}$$

La funzione $c(|\xi|)$ è detta *micromodulo* ed ha significato di resistenza elastica di legame; per legame si intende relazione tra particelle. Per un micromodulo lineare la forza diventa allora:

$$\mathbf{f}(\eta, \xi) = \begin{cases} \frac{\xi+\eta}{|\xi+\eta|} c(|\xi|) s, & \xi \leq \delta \\ \mathbf{0}, & \xi > \delta \end{cases} \tag{2.6}$$

Indicando che se due particelle sono una fuori dall'orizzonte dell'altra queste non interagiscono.

Il valore del micromodulo in questa stesura viene preso costante, e dai risultati in Ha and Bobaru (2010) il suo valore è dato da:

$$c_0 = \frac{6E}{\pi\delta^3(1-\nu)}$$

Il potenziale ω è un potenziale centrale e tale assunzione implica, per materiali microelastici lineari e isotropici, un coefficiente di Poisson ν pari a $\frac{1}{3}$ per i casi di *plane stress* ed $\frac{1}{4}$ per i casi di *plane strain*.

Il valore del micromodulo diventa pertanto, come riportato in Zaccariotto et al. (2015):

$$c_0 = \frac{9 \cdot E}{\pi \cdot t \cdot \delta^3} \tag{2.7}$$

Dove con t si tiene conto anche dello spessore della lamina, precedente considerato come unitario.

Per introdurre una condizione di *failure* nel modello, i legami devono potersi rompere o cessare di agire: ciò accade una volta raggiunto un valore limite, detto allungamento relativo critico s_0.

Si introduce una grandezza, detta energia per unità di lunghezza della frattura (o per unità di area nel caso tridimensionale) G_0, come energia necessaria ad ottenere la completa separazione tra due parti di un corpo.

In Silling and Askari (2005) viene misurata tale quantità derivandola dalla seguente espressione:

$$G_0 = \int_0^{\delta}\int_0^{2\pi}\int_z^{\delta}\int_0^{cos^{-1}(\frac{z}{\xi})}\left[\frac{c(\xi)\xi {s_0}^2}{2}\right]\xi sin(\phi)d\phi d\theta d\xi dz \tag{2.8}$$

Più precisamente:

$$G_0 = \int_0^{\delta} dz \int_0^{2\pi} d\theta \int_z^{\delta} \frac{c(\xi)\cdot\xi^2 {s_0}^2}{2} d\xi \int_0^{cos^{-1}(\frac{z}{\xi})} sin\phi d\phi \tag{2.9}$$

Sostituendo il micromodulo con c_0 e riordinando per s_0 si ottiene la quantità:

$$s_0 = \sqrt{\frac{4\pi G_0}{9E\delta}} \tag{2.10}$$

Una volta superato l'allungamento relativo critico s_0 si spezza il legame eliminando la *pairwise force* che agisce tra le due particelle.

Definiti:

- la forma dell'oggetto e della cricca iniziale;
- i carichi;
- la precisione di discretizzazione Δ;
- il rapporto m tra raggio dell'orizzonte δ e Δ;
- tutti i parametri fisici come modulo di elasticità E, densità del materiale ρ e velocità di propagazione delle onde elastiche nel mezzo v;
- le tre quantità c_0, G_0, s_0;
- la durata dell'esperimento;

la simulazione è totalmente definita e l'algoritmo qui presentato è in grado di eseguirla.

Capitolo 3

Struttura dell'algoritmo

Con l'obiettivo di creare un ambiente di elaborazione da zero basato sulla teoria peridinamica, l'algoritmo è sviluppato suddividendo i passaggi logici in corrispondenti *function* che permettono di variare i loro argomenti e di avere agevolmente risultati diversi a seconda della necessità.

Ogni *function* è modificabile singolarmente, anche se lo script finale dovrebbe essere sufficiente a variare i parametri desiderati.

Nel corso di questo elaborato verranno introdotte le variabili sia con simbolo matematico (per riferimento alla teoria peridinamica), sia con nome della variabile all'interno del codice. Esempio: Δ e `delta`.

Lo spirito di sviluppo seguito è stato quello di rendere le singole parti di codice più versatili possibili: già dalla funzione dedicata alle forme dell'oggetto `shapes.m` è possibile definirne di diversi tipi - o aggiungerne se si desidera - e il resto dell'elaborato deve adattarsi di conseguenza.

Attualmente l'adattabilità è garantita (testata estensivamente) fino all'applicazione della teoria peridinamica presente nella parte di iterazione nel tempo; il post processing è stato invece sviluppato soltanto orientandosi sui risultati ottenuti da Ha e Bobaru (2010).

L'algoritmo si sviluppa in tre macro fasi: una in preparazione all'iterazione dove si applica la teoria peridinamica, una dedicata all'iterazione, infine l'ultima dedicata all'elaborazione dei risultati.

Definizione della griglia di nodi e dei *bond*

Per qualsiasi implementazione numerica di una simulazione fisica risulta necessario discretizzare il dominio del problema, che corrisponde in questo caso alla geometria dell'oggetto in questione.

Essendo progettato in un contesto bidimensionale, l'oggetto è una lastra o lamina sagomata. Ogni figura andrà interpretata come la rappresentazione del comportamento di una lastra, di un certo spessore `thickness` definito a priori.

Una volta nota la forma e le dimensioni dell'oggetto, la prima parte dell'algoritmo trasforma tali parametri in una lista di punti (nodi) con un coefficiente correttivo dovuto alla discretizzazione (pesi assoluti). Ogni nodo rappresenta una porzione fisica dell'oggetto a cui viene associata una massa e una dimensione puntiforme.

In letteratura vi sono diversi approcci all'individuazione di tali nodi, da approcci *meshless* come Nguyen et al. (2008) all'utilizzo di diagrammi Voronoi in Okabe et al. (2009).

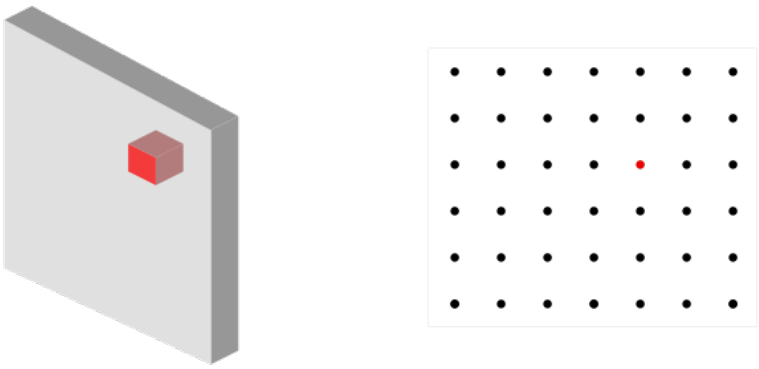

Figura 3.1: Una lastra di materiale viene convertita in una serie di punti rappresentanti prismi di materiale.

Definizione oggetto come poligonale

Il bordo della lastra è riducibile ad una curva chiusa nel piano, e successivamente discretizzata in una poligonale chiusa. Questa permette di avere una relazione diretta tra i punti di due vettori `xShape` e `yShape` quella che sarebbe la forma continua dell'oggetto.

Se l'oggetto è già sagomato come una poligonale non vi sono imprecisioni di discretizzazione in questa prima fase, ma se è sagomato con archi curvi ciò implica la necessità di riduzione dell'arco ad una successione di piccoli segmenti. Per coerenza con la parte successiva del programma, si consiglia una precisione proporzionale alla lunghezza di riferimento `delta` della discretizzazione del dominio interno al contorno appena definito.

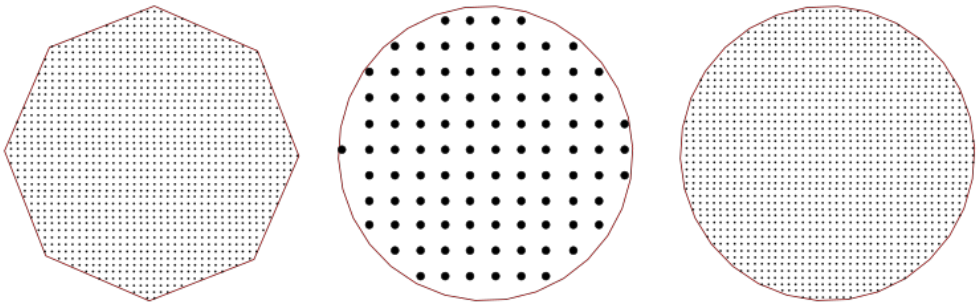

Figura 3.2: Prima figura: discretizzazione precisa, poligonale grezza; seconda figura: discretizzazione grezza, poligonale precisa; terza figura: discretizzazione precisa, poligonale precisa.

Quella di poligonale chiusa non è affatto una limitazione: è possibile definire oggetti non semplicemente connessi, o separati. È sufficiente definire la poligonale con segmenti coincidenti in due porzioni diverse della percorrenza per ottenere forme anulari, oppure creare dei collegamenti filiformi di spessore nullo per ottenere due oggetti separati.

Poligonale, curva e forma saranno usate come sinonimi in questa prima parte dell'elaborato.

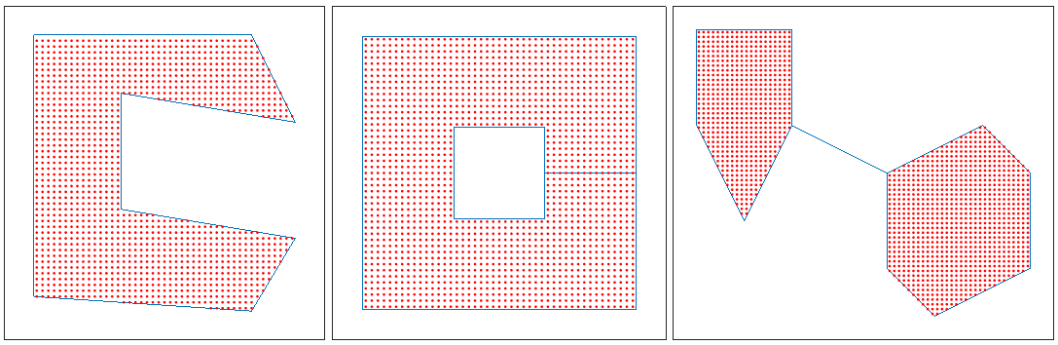

Figura 3.3: Oggetto concavo, oggetto cavo, oggetti separati.

Individuazione nodi oggetto

Definita come `delta` la lunghezza di riferimento per la discretizzazione, viene popolato di punti un rettangolo che circoscrive l'intera forma. Il rettangolo ha dimensioni date dalla minima e massima ascissa e ordinata tra tutti i punti della poligonale.

Per evitare di posizionare i punti iniziali e finali sul contorno della forma, la posizione di partenza della popolazione viene scelta traslata di $[\frac{\Delta}{2}, \frac{\Delta}{2}]$ rispetto all'origine data da `[min(xShape), min(yShape)]`.

Una volta ricoperto il rettangolo di punti, vengono topologicamente individuati quelli interni alla forma grazie al *Teorema della curva di Jordan*: data una curva chiusa non autointersecante, questa divide il piano $\mathbb{R}^2$ in due regioni distinte. La regione interna alla curva rappresenta la serie di punti dell'oggetto.

Inizialmente è stato realizzato un algoritmo ad hoc che sfruttasse una proprietà topologica conseguente al *Teorema della curva*: se un punto si trova nella regione interna, il segmento tra tale punto e un punto esterno (in questo caso l'origine del rettangolo) interseca la poligonale un numero dispari di volte, mentre se si trova nella regione esterna un numero pari di volte.

Esiste una funzione di Matlab, chiamata `inpolygon()`, che individua i punti interni a una poligonale più efficientemente e che copre più casi, come forme poligonali molteplicemente connesse.

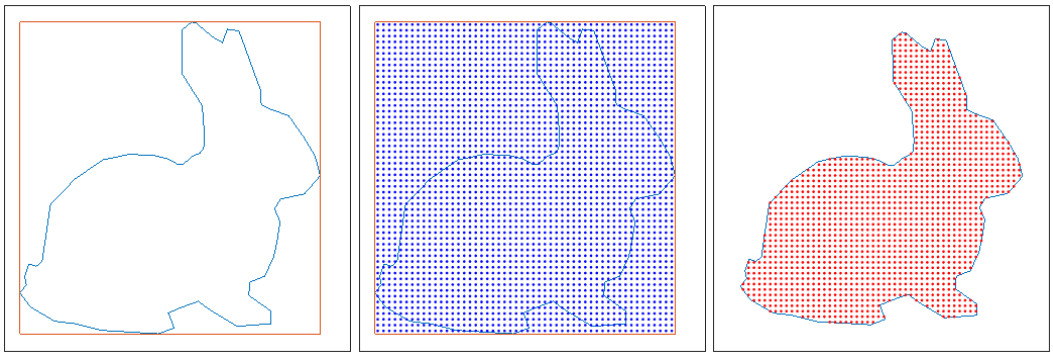

Figura 3.4: Vengono illustrati i vari passaggi per la determinazione dei nodi interni per una forma arbitraria.

Fornendo a questa funzione la lista dei nodi `[xNodes, yNodes]` e la lista dei punti della poligonale `[xShape, yShape]`, essa restituisce un vettore booleano `in` indicando se il punto si trova all'interno della poligonale o meno.

Si può così costruire una diversa lista di nodi `[xObjNodes, yObjNodes]`, questa volta tutti appartenenti all'oggetto.

Individuazione nodi contorno

Come si è visto, i nodi rappresentano dei parallelepipedi massa concentrati su dei punti. Per quanto riguarda i nodi interni alla figura tale semplificazione è accettabile, mentre per i nodi sul contorno si introducono errori dovuti alla diversità tra il parallelepipedo a base quadrata e il prisma a base poligonale che in realtà si dovrebbe considerare.

Occorre individuare i nodi al contorno: si costruisce una matrice a partire dal vettore delle posizioni `[xNodes, yNodes]` e dal vettore `in` indicativo dell'essere interno o esterno alla forma, moltiplicandole. Si ottiene una matrice che avrà come valore `0` per tutti i punti esterni alla figura, mentre un indice per i punti interni.

In modo da avere una relazione diretta con il vettore dei nodi dell'oggetto `[xObjNodes, yObjNodes]` gli indici della matrice vengono sostituiti con indici progressivi a partire da `1`.

Il semplice criterio utilizzato per definire se i nodi si trovano al contorno - salvato nel vettore booleano `isEdge` - è: se il nodo è immediatamente circondato da altri nodi nelle quattro direzioni allora è interno, viceversa si trova al contorno. Le quattro direzioni sono date dai coefficienti della matrice: `(i+1,j)`, `(i,j+1)`, `(i-1,j)`, `(i,j-1)`.

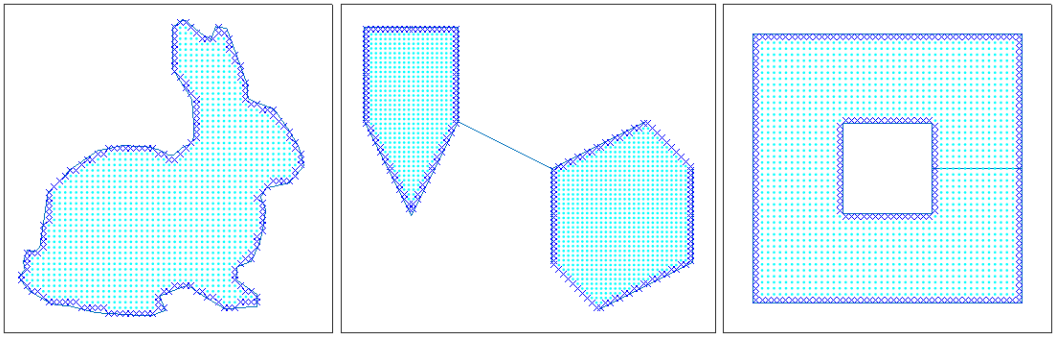

Figura 3.5: Il criterio di individuazione del contorno funziona per qualsiasi forma venga assegnata all'algoritmo. I punti indicatin blu sono individuati come punti del contorno.

Coefficiente di volume materiale

I nodi del contorno possono trovarsi a rappresentare più o meno materiale di un nodo interno alla figura. Ciò è dovuto al fatto che la discretizzazione avviene a intervalli regolari, mentre la poligonale può avere forma qualsiasi.

Più è fine la discretizzazione, meno l'effetto di questo errore è percepibile; tuttavia considerare tale imprecisione riduce di un ordine di grandezza (per discretizzazioni non troppo fini) l'errore relativo di trascurare le masse equivalenti del contorno.

L'algoritmo di questa fase richiede più passaggi rispetto alle fasi precedenti:

- Assegnazione zona di influenza;
- Riduzione area di influenza dovuta a nodi interni e ad altri nodi del contorno;
- Riduzione area di influenza dovuta alla poligonale.

Per **zona di influenza** si intende un'area porzione dell'area totale di sezione dell'oggetto. I punti interni al contorno hanno necessariamente una zona di influenza unitaria, ovvero la

loro area è già definita come Δ^2.

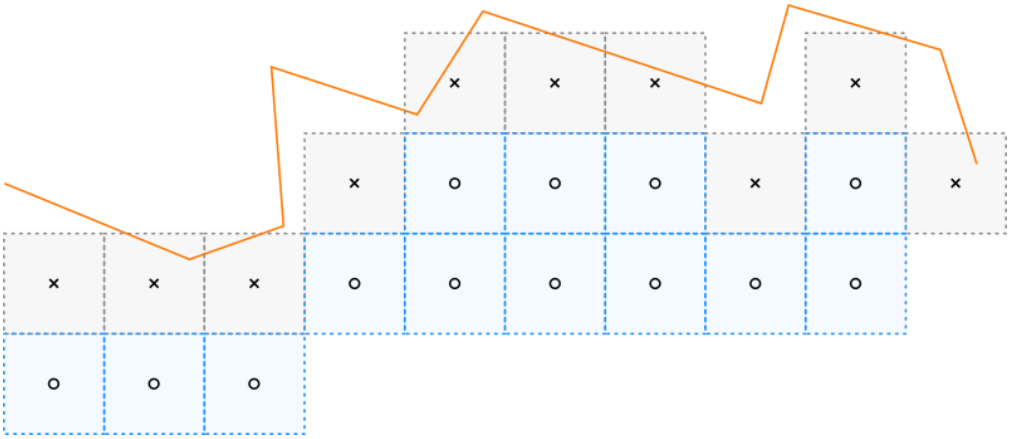

Figura 3.6: Dettaglio che riporta una porzione di oggetto con i nodi esterni e la poligonale della forma.

Si prende un'area quadrata con lato doppio della discretizzazione Δ e centrata nel punto del nodo al contorno: il lato doppio è stato scelto poiché oltre a tale dimensione la discretizzazione avrebbe dovuto prendere un nodo più esterno a quello attualmente al contorno.

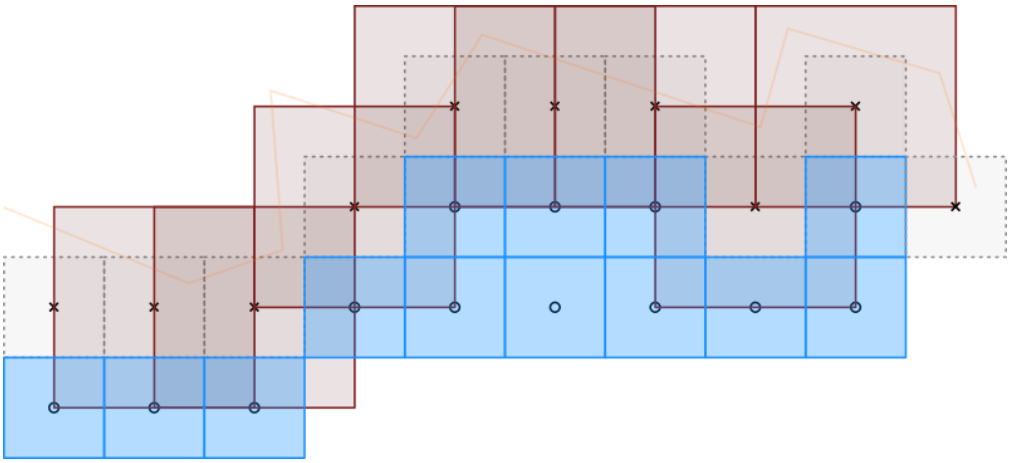

Figura 3.7: Dettaglio che riporta una porzione di oggetto con i nodi esterni e la poligonale della forma.

Tale area va ridotta considerando i nodi interni e quelli al contorno.

Per evitare una sovrapposizione di zone di influenza (che equivale all'introduzione di nuova massa), dove queste vengono sovrapposte si divide l'area a metà tra i due nodi. In base alla disposizione dei nodi, le aree assumono forme diverse.

Per definire nell'algoritmo le poligonali di queste zone di influenza si è scelto un criterio semplice: si percorre il perimetro confrontando se i nodi presenti in esso sono dell'oggetto; in tal caso la distanza dal centro dei punti relativi ai nodi della poligonale `1-2-3-4-5-6-7-8` va dimezzata.

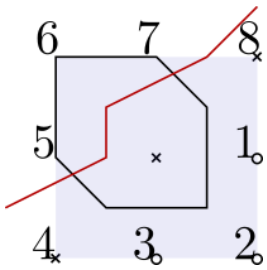

Figura 3.8: Esempio di percorrenza per la determinazione della zona di influenza ridotta dai nodi adiacenti.

Le forme risultanti non si sovrappongono con gli altri nodi del contorno.

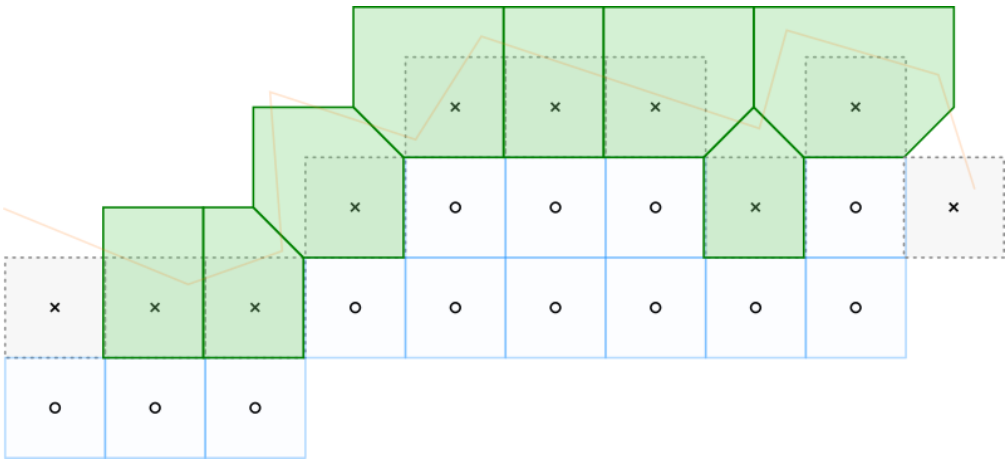

Figura 3.9: Dettaglio che riporta una porzione di oggetto con i nodi esterni e la poligonale della forma.

L'ultima riduzione va fatta confrontando le zone di influenza con la poligonale della forma

dell'oggetto `[xShape, yShape]`.

Per individuare i punti della poligonale effettiva della zona di influenza finale si impiega nuovamente la funzione `inpolygon()`.

- Si individuano i punti della poligonale provvisoria interni alla forma dell'oggetto;
- Si cercano i punti interni alla zona di influenza della poligonale della forma;
- Si individua l'intersezione tra le due poligonali grazie alla funzione `InterX.m`;
- Si crea la poligonale finale, ordinando i punti in senso orario rispetto al loro centroide.

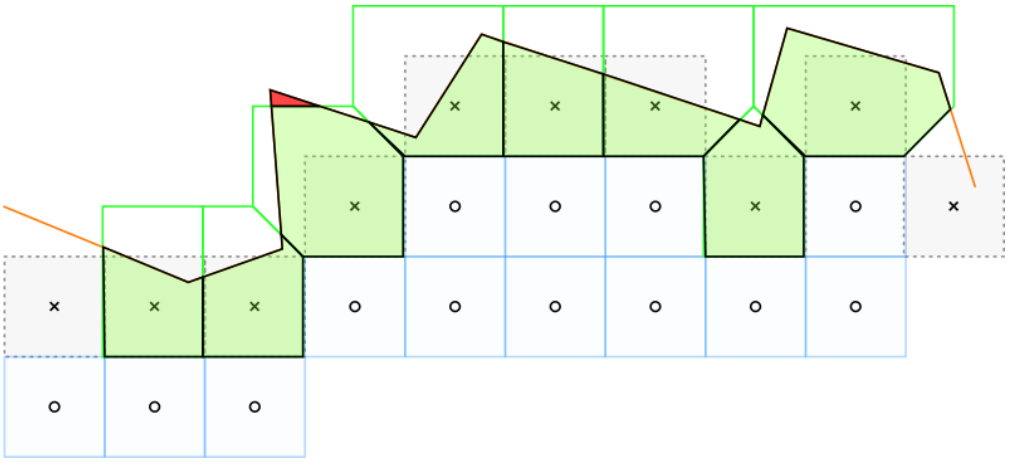

Figura 3.10: Le zone di influenza vengono ridotte in base alla poligonale della forma. L'area in rosso rappresenta l'unico caso di questo procedimento che induce ad un errore; ciò capita per poligonali molto concave.

Il coefficiente da assegnare al nodo, nota ora la sua zona di influenza, è il rapporto tra l'area della poligonale e l'area di un nodo interno di valore Δ^2. Tali rapporti vengono salvati nel vettore `weight`.

La discretizzazione dell'oggetto è ora completa: la forma della poligonale è stata trasformata in due vettori con la posizione dei nodi `[xObjNodes, yObjNodes]`, e l'effetto dell'imprecisione al contorno viene corretto con il vettore `weight`.

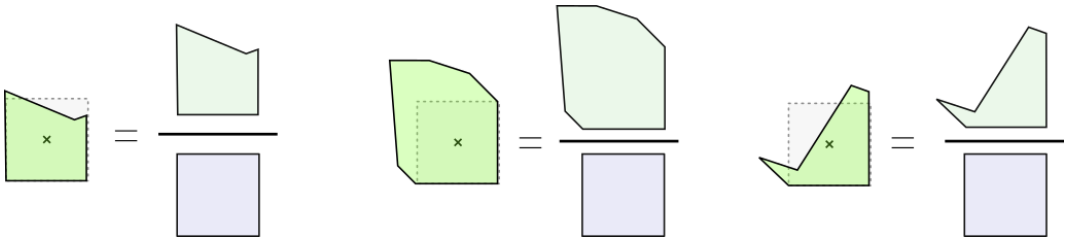

Figura 3.11: Il vettore `weight` mantiene al suo interno il rapporto tra le aree indicate.

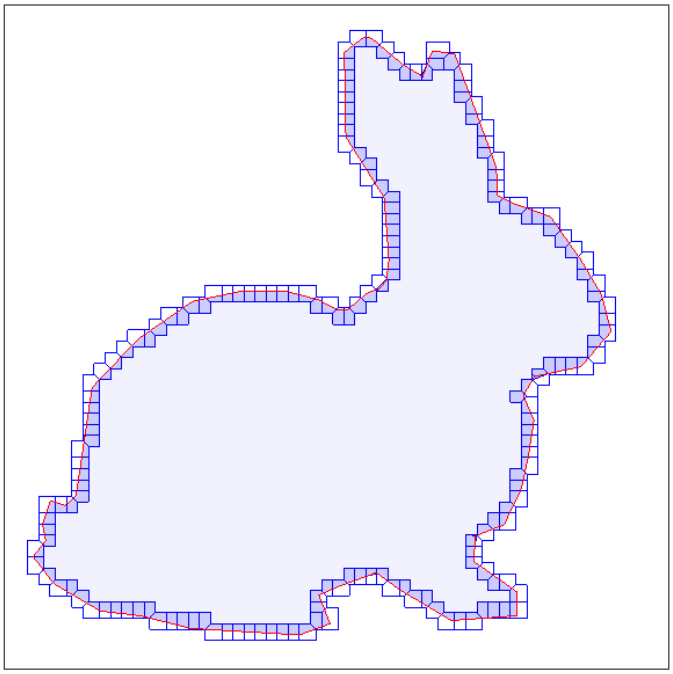

Figura 3.12: Algoritmo applicato ai nodi esterni di una figura arbitraria. I contorni blu rappresentano le zone di influenza dimezzate, le aree in azzurro rappresentano le zone di influenza ridotte dalla poligonale della forma.

Struttura dati per nodi e legami tra nodi

Una volta definita la dimensione dell'orizzonte locale come multiplo `m` della discretizzazione si necessita di una struttura di dati per immagazzinare le relazioni tra un nodo e quelli appartenenti al suo orizzonte. Tale struttura è costituita di tre matrici, caratterizzate da ogni riga corrispondente a un nodo:

- la prima immagazzina gli indici dei nodi interni all'orizzonte rispetto al nodo in questione `relativeNode`
- la seconda attribuisce a tali nodi un coefficiente relativo alla loro posizione nell'orizzonte `relativeNodeWeight`
- la terza indica la posizione del nodo centrale nella riga della prima matrice del nodo relativo `relativeRelativeNode`

Orizzonte locale

Si definisca l'orizzonte come poligonale di un poligono regolare di un numero di lati > 100 con apotema `m*delta` e come origine il punto del nodo in questione. È immediato con `inpolygon` individuare i nodi interni all'orizzonte e inserire tali indici nella riga della prima matrice.

Dato che non tutti i nodi hanno il massimo delle relazioni possibili con altri nodi (si pensi a un nodo al contorno) e la matrice deve essere definita con un numero fisso di colonne, si riempie la riga con gli indici e si azzera la porzione restante con degli zeri.

Coefficiente relativo nodi orizzonte

Nella letteratura un esempio completo per la caratterizzazione dei coefficienti dei nodi relativi interni all'orizzonte è di Yu (2011): nel terzo capitolo vi è una trattazione estensiva. Ciò che accade è che il nodo centrale percepisce diversamente l'effetto dei nodi vicini al contorno dell'orizzonte: questi devono essere corretti con un coefficiente dato dall'area effettiva che il nodo relativo rappresenta nel contesto dell'orizzonte. La seconda matrice tiene conto di tale effetto e associa al nodo nella stessa posizione della prima matrice tale coefficiente, generalmente unitario.

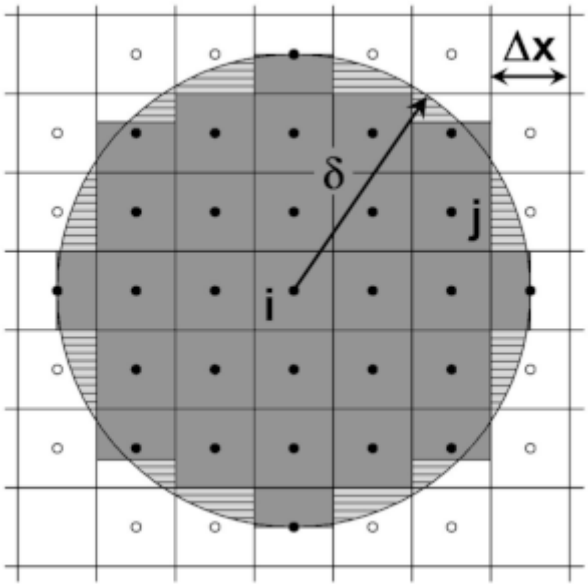

Figura 3.13: Come indigato da Yu (2011), ogni nodo al contorno rappresenta un'area diversa da quella unitaria.

Come valori dei nodi al ciglio dell'orizzonte sono stati presi quelli studiati in Scudellaro (2012).

Richiamo di un nodo a se stesso

Una terza matrice entra in gioco quando successivamente si andranno a spezzare le relazioni di forza tra i nodi (legami). Per effettuare la rottura di un legame, infatti, si è scelto di azzerare il coefficiente relativo nella seconda matrice, mantenendo nella prima matrice l'indice del nodo relativo. In questo modo è possibile sapere il rapporto tra legami finali e legami iniziali semplicemente conoscendo il numero di elementi non nulli nelle righe delle due matrici una volta conclusa l'iterazione.

Perché la rottura sia efficace, devono essere azzerati i coefficienti in entrambe le direzioni, a due a due.

Si salva, nella terza matrice, la posizione dell'indice del nodo centrale x_i nella riga del nodo relativo x_j della prima matrice, così da sapere quale posizione azzerare nella seconda matrice.

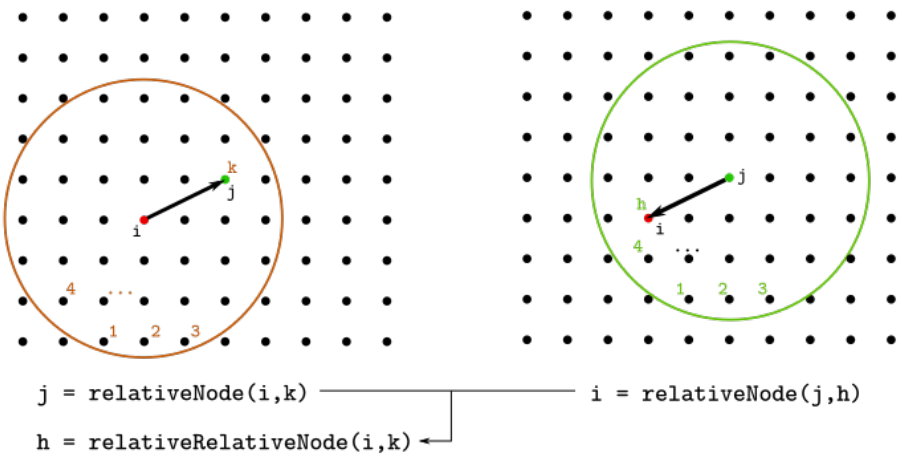

Figura 3.14: Grazie alla matrice `relativeRelativeNode` è possibile sapere la posizione dell'indice nodo x_i nella riga del nodo x_j.

Condizioni iniziali

Rottura legami

Può essere richiesto di inserire una cricca come condizione iniziale.

Definita la cricca come una poligonale, preferibilmente aperta, i segmenti dati dalla lista di punti `[xCrack, yCrack]` vengono confrontati con ogni legame presente nell'oggetto.

È stata creata una lista di legami univoci a partire dalla matrice `relativeNodes` individuando i nodi a due a due ed eliminando i duplicati, visto che ogni legame interessa due punti di vista.

Non appena `InterX` individua una intersezione tra il legame e la cricca, il legame viene spezzato, azzerando i `relativeNodeWeight` nella posizione del nodo relativo a ciascuno dei due nodi interessati.

Questo metodo permette di ottenere cricche di forme qualsiasi, tenendo conto che maggiore è la complessità della cricca e maggiore la definizione della discretizzazione dell'oggetto, più

tempo richiederà questa parte di algoritmo.

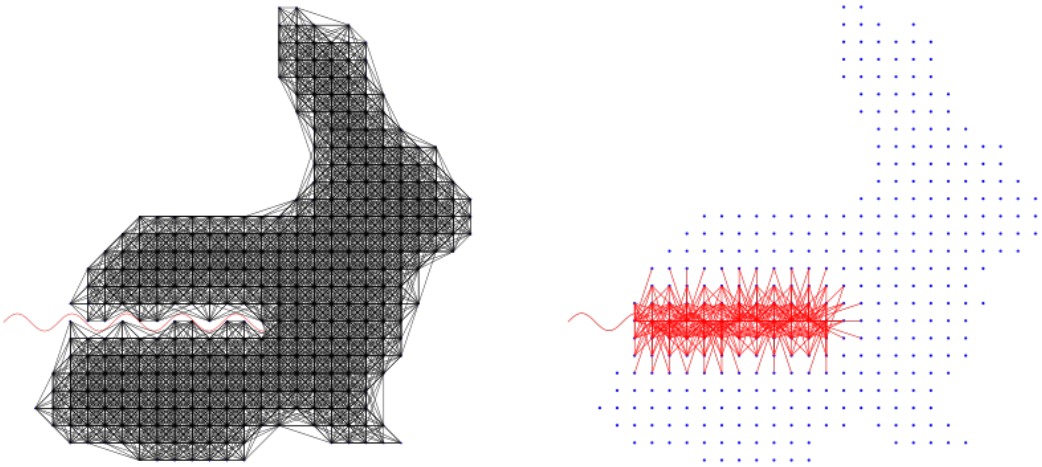

Figura 3.15: In seguito all'applicazione di una cricca iniziale: legami intatti di un oggetto qualsiasi a sinistra, legami rotti dalla cricca iniziale a destra.

Carichi applicati e condizioni di vincolo

I carichi esterni in questa istanza sono costanti, ma nulla vieta di fornire alla funzione `loadsAndConstrains` una variabile di tempo e di aggiornarli nell'iterazione peridinamica per ogni passo temporale.

I carichi vengono descritti come due vettori `bX` e `bY` di dimensione uguale alla lista dei punti, in modo da poter associare ad ogni nodo il carico che vi è applicato. Per i nodi a cui viene applicato il carico si consiglia di rendere i legami ad essi associati indistruttibili, poiché l'impulso di forza potrebbe rompere tali legami prima di permettere la distribuzione del carico.

I vincoli possono essere solo verticali o orizzontali: si introducono due vettori booleani `cX` e `cY` che dicano se lo spostamento del nodo `[uX, uY]` è consentito nel corso della simulazione.

Modulo per la soluzione con teoria peridinamica

Definizione parametri fisici

Noti i dati sperimentali e i parametri fisici della lastra fragile, vengono inseriti nella funzione `physicalQuantities` per essere interpretati dal resto del programma.

Vengono inseriti inoltre i parametri che determinano la rottura di un legame quali energia di rottura G_0, micromodulo C_0, e allungamento limite per la rottura s_0.

Viene calcolata la durata del passo iterativo temporale Δt come:

$$\Delta t = \frac{v}{10 \cdot \Delta x} \tag{3.1}$$

Ove v rappresenta la velocità di propagazione delle onde elastiche nel mezzo, riconducibile alla velocità del suono nello stesso mezzo. Con un passo iterativo così calcolato la simulazione permette una precisione temporale sufficiente a simulare il comportamento reale della lastra.

Iterazione nel tempo

Sezione più importante di tutta la stesura, l'iterazione nel tempo rappresenta il punto vero e proprio in cui viene applicata la teoria peridinamica.

In questa implementazione si è impiegato l'algoritmo di iterazione nel tempo detto *Verlet-Velocity* introdotto per la prima volta in Verlet (1967) e riadattato per il contesto peridinamico, che prevede l'introduzione di un termine di velocità di mezzo passo iterativo.

Inizializzati i vettori spostamento `[uX, uY]`, velocità `[vX, vY]` come nulli e accelerazione `[aX, aY]` come carico esterno per unità di massa, l'algoritmo segue qualche semplice passo:

1. Calcolo della velocità di mezzo passo:

$$\mathbf{v}_{HS} = \mathbf{v}(\mathbf{x}_i, t) + \frac{\Delta t}{2}\mathbf{a}(\mathbf{x}_i, t) \tag{3.2}$$

2. Calcolo della posizione al passo successivo:

$$\mathbf{u}(\mathbf{x}_i, t+1) = \mathbf{u}(\mathbf{x}_i, t) + \Delta t \cdot \mathbf{v}_{HS} \tag{3.3}$$

3. Calcolo delle forze di interazione con gli altri nodi (_pairwise force_) e del carico esterno.

   La forza f è definita in questo contesto con micromodulo costante, linearmente dipendente all'allungamento $s = \frac{|\xi+\eta|-|\xi|}{|\xi|}$ e parallela alla sua direzione:

$$\mathbf{a}(\mathbf{x}_i, t+1) = \frac{1}{\rho}\left[\sum_j \mathbf{f}(\mathbf{u}(\mathbf{x}_j, t+1) - \mathbf{u}(x_i, t+1), \mathbf{x}_j - x_i) \cdot \Delta V_j + \mathbf{b}(x_i, t)\right] \tag{3.4}$$

4. Calcolo della velocità al passo successivo:

$$\mathbf{v}(\mathbf{x}_i, t+1) = \mathbf{v}_{HS} + \frac{\Delta t}{2}\mathbf{a}(\mathbf{x}_i, t+1) \tag{3.5}$$

Per tanti passi di iterazione t quanti corrispondono all'intervallo di iterazione richiesto `intervalTime` diviso per il tempo massimo di passo iterativo `deltaT`. Si consiglia di non fornire un tempo eccessivo di simulazione per volta, per evitare attese troppo lunghe senza avere conferme grafiche.

Visualizzazione percorsi di cricca

Massimo allungamento di legame per nodo

Se si mantiene traccia dell'allungamento massimo dei legami intatti per ogni nodo nel corso dell'iterazione, è visibile l'andamento della distribuzione delle onde elastiche nel corso del tempo.

Si salva, in un vettore `maxStretch` l'allungamento massimo per passo iterativo di ogni nodo, così da indicare ogni nodo con il proprio legame più allungato.

Calcolo vettore con indici di danno per nodi

La struttura dati introdotta permette di ricondursi facilmente al numero dei legami spezzati e al numero di legami inizialmente presenti per ogni nodo.

Si definisce come coefficiente di danno il valore dato da:

$$damageIndex = 1 - \frac{n_{bonds,end}}{n_{bonds,start}} \tag{3.6}$$

Che nella letteratura viene espresso come:

$$\varphi(x) = 1 - \frac{\int_H \mu \cdot dH_{\hat{x}}}{\int_H dH_{\hat{x}}} \tag{3.7}$$

Dove H rappresenta l'orizzonte, $dH_{\hat{x}}$ il nodo relativo e μ il fatto se sia spezzato o meno.

Per calcolare $n_{bonds,end}$ e $n_{bonds,start}$ si sfruttano le funzioni `length()` e `find()` applicate alle righe della matrice `relativeNode` e `relativeNodeWeight` in modo da ottenere il numero di valori non nulli.

Tale coefficiente è indicativo dell'integrità del nodo e fornisce una visualizzazione chiara dell'andamento della cricca.

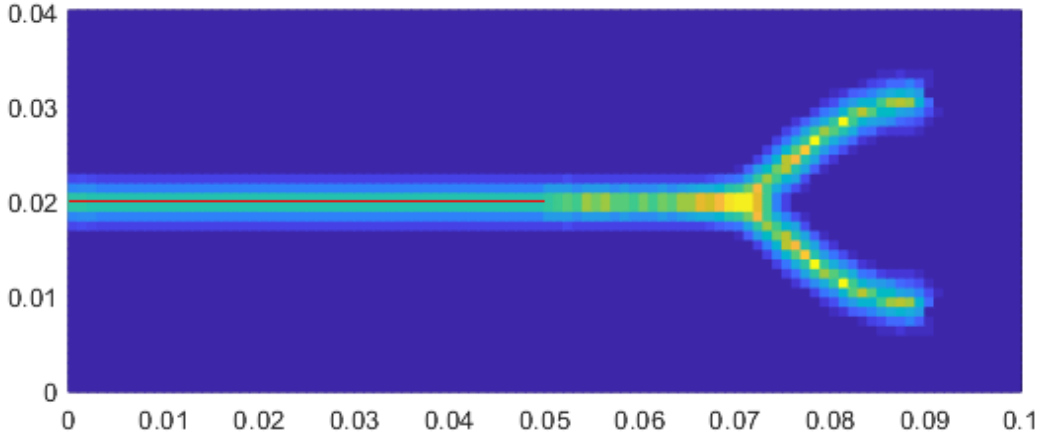

Figura 3.16: Distribuzione dei coefficienti di danno per una lamina di vetro Duran 50 con discretizzazione di 1mm dopo $46\mu s$ dall'applicazione del carico.

Curve di livello di funzione bidimensionale interpolante

Per fornire un percorso di cricca chiaro, si reinterpreta il vettore coefficiente di danno `damageIndex` come una funzione bidimensionale nelle direzioni x e y. Tale funzione viene definita a partire da una `meshgrid` (formata essenzialmente da due matrici `XX` e `YY`) che ricopre il rettangolo circoscritto all'oggetto con punti corrispondenti ai nodi fin'ora studiati.

Confrontando ogni punto della mesh con i nodi dell'oggetto, si assegna il valore alla funzione bidimensionale `Z`, uguale al coefficiente di danno del nodo in questione.

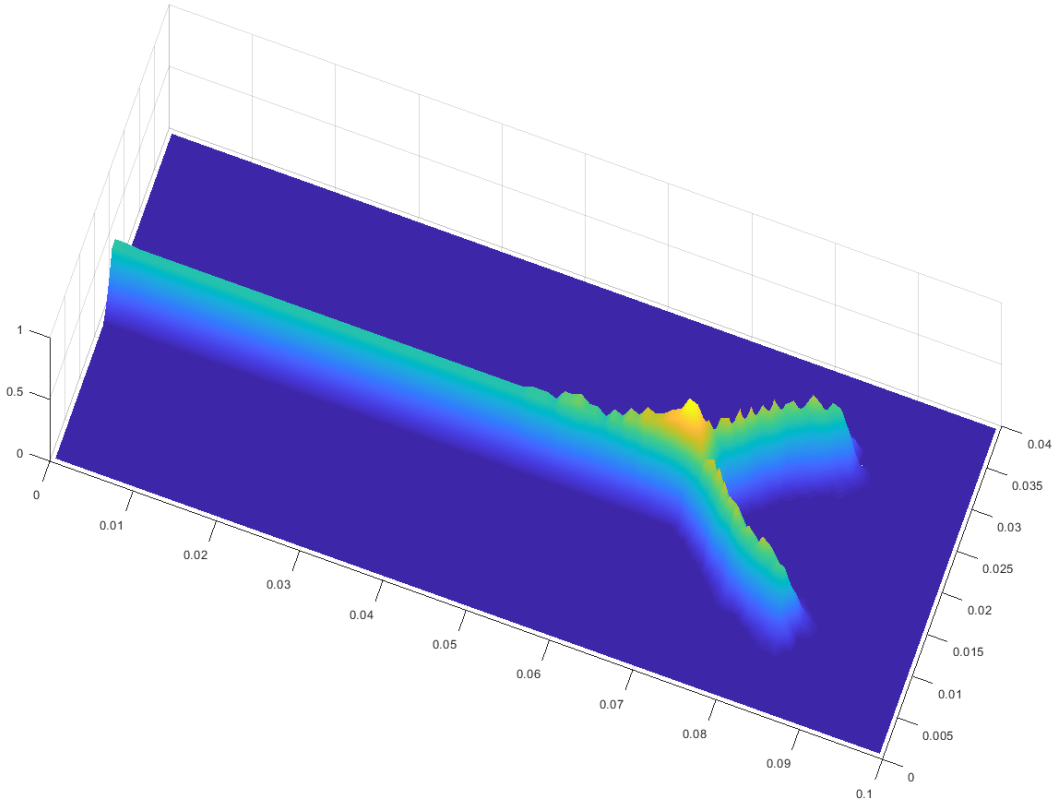

Figura 3.17: Interpolazione tridimensionale della distribuzione dei coefficienti di danno.

È possibile raffinare ulteriormente la superficie con un'interpolazione cubica della funzione `Z` in una `meshgrid` più fine (impostata di default a 10 punti per ogni lunghezza `delta`).

Il valore della curva di livello viene investigato a partire dalla cricca nelle condizioni di

partenza: si cerca infatti il valore che dia una curva di livello che circondi il più vicino possibile la poligonale della cricca.

Individuato il valore si mantiene per tutto il periodo di iterazione.

Capitolo 4

Simulatore

Il simulatore finale è composto da poche righe che richiamano le funzioni presentate nella Struttura dell'Algoritmo.

Si libera l'ambiente dalle variabili precedenti:

```
clear all;
```

Si introduce la forma dell'oggetto come poligonale. A libreria è stato creato uno script che raccoglie alcune forme, quella dell'esperimento è la 5:

```
[xShape, yShape] = shapes(shape,scale,xTranslate,yTranslate);
```

Scelto il passo di discretizzazione, si individuano i nodi dell'oggetto:

```
delta = 0.001;
[xObjNodes, yObjNodes, nNodes, isEdge] = nodes(delta, xShape, yShape);
```

Si calcolano i coefficienti di influenza assoluti dei nodi dell'oggetto:

```
[weight, errNoEdge, errEdge] = weights(...
    xShape, yShape,...
    delta, nNodes, ...
```

```
    xObjNodes, yObjNodes,...
    isEdge);
```

La funzione restituisce anche l'errore relativo tra la somma delle aree associate ai nodi e l'area racchiusa dalla poligonale di forma, serve per confrontare l'aver considerato le zone di influenza o l'aver assunto tutti i coefficienti come unitari.

Noto il rapporto tra il raggio dell'orizzonte e il passo di discretizzazione, si popolano la prima, seconda e terza matrice della struttura dati già presentata. Si usa un m di $3,5$ per individuare i nodi, mentre di 3 per tutti i calcoli relativi a energia di frattura e allungamento critico.

```
m=3.5;
[bondsPerNode, relativeNode, relativeNodeWeight,...
    relativeRelativeNode] = horizonNodesFinder(...
    m, delta, nNodes, ...
    xObjNodes, yObjNodes);
```

Si individuano i legami e il numero di legami tra i nodi

```
[bond, bonds] = bondFinder(relativeNode, nNodes);
```

Si definisce e introduce la poligonale della cricca. Anche in questo caso è stata costruita una piccola libreria con forme di cricche di base

```
crackType = 1;
[xCrack, yCrack] = cracks(crackType);
```

Si spezzano i legami intersecati dalla cricca iniziale

```
bondsCrackedUpdate;
```

Si introducono le quantità fisiche di riferimento del materiale di cui è composto l'oggetto

```
physicalQuantities;
```

Si esegue l'iterazione peridinamica

```
iteration(restart,iterationTime);
```

Si visualizzano i risultati

```
plots;
```

Capitolo 5

Esperimento

Per verificare la validità dell'algoritmo sviluppato, è stato testato confrontandolo con la stessa simulazione effettuata in Ha and Bobaru (2010).

Installazione

L'esperimento consiste in una lastra di vetro *Duran 50* delle dimensioni di $100mm$ di larghezza l per $40mm$ di altezza h che presenti una cricca iniziale. La densità del vetro *Duran 50* è di $2235\frac{kg}{m^3}$, il suo modulo di elasticità E è di $65GPa$ e l'energia di frattura per unità di area G_0 di $204\frac{J}{m^2}$. Tali dati vengono riportati in Ha and Bobaru (2010) e sono stati ricavati dai risultati sperimentali di Döll (1975).

Applicazione carico

Viene applicato alla lastra un carico σ di $12MPa$ sul lato superiore e inferiore. Per applicare tale carico nella simulazione questo va distribuito sui nodi dell'oggetto, prestando attenzione alle quantità dimensionali.

Considerando la lunghezza di discretizzazione Δ, la superficie superiore della lastra può essere suddivisa in $\frac{l}{\Delta}$ porzioni di superficie, sulle quali viene applicata una risultante di:

$$\mathbf{F} = \sigma \cdot \Delta \cdot t \tag{5.1}$$

Dato che il carico nell'algoritmo è espresso come forza per unità di volume, si divide il carico per il volume associato al nodo all'estremità della lastra:

$$\mathbf{b} = \frac{\mathbf{F}}{\Delta V} = \frac{\sigma \cdot \Delta \cdot t}{\Delta \cdot \Delta \cdot t} = \frac{\sigma}{\Delta} \tag{5.2}$$

Inserendo i valori si ottiene un carico per nodo di:

$$\mathbf{b} = \frac{12MPa}{0.001m} = \frac{12 \cdot 10^6 N}{m^2} \cdot \frac{1}{10^{-3}m} = 12 \cdot 10^9 \frac{N}{m^3}$$

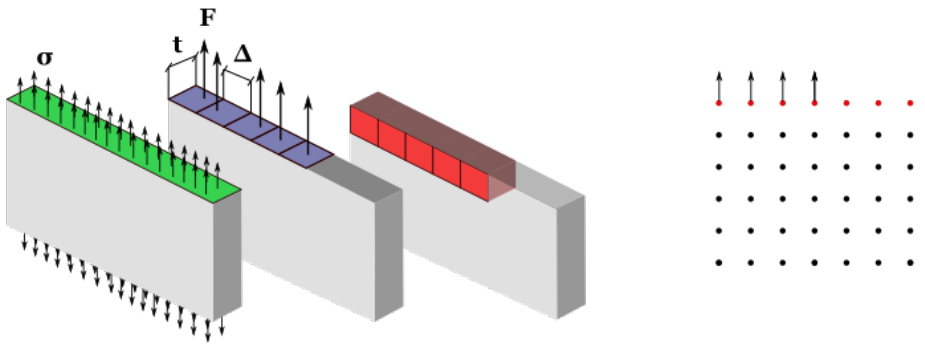

Figura 5.1: Illustrazione dei passaggi da svolgere per ottenere il carico per ogni nodo.

Applicazione cricca iniziale

Applicando una cricca iniziale da sinistra verso destra, a metà dell'altezza della lastra fino ad arrivare al centro della lastra si rompono i legami che intersecano la cricca.

Andando ad analizzare i legami intatti, si individua la cricca come mancanza della presenza di legami intatti.

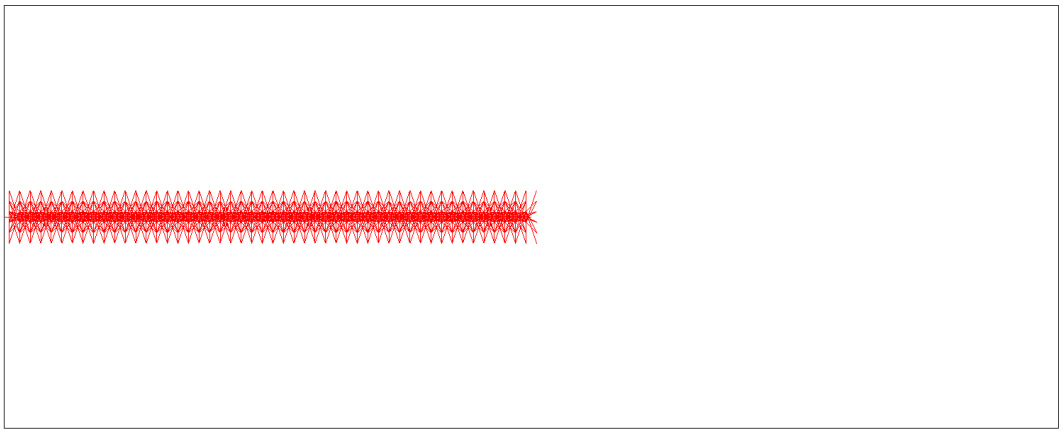

Figura 5.2: Legami spezzati all'inserimento della cricca iniziale.

Evoluzione temporale

Prendendo vari fotogrammi dei massimi allungamenti per nodo nel corso dell'iterazione, si può studiare l'evoluzione dell'esperimento.

A partire da dove viene applicato il carico, questo viene distribuito fino alla punta della cricca dove - in poco tempo - si raggiunge lo sforzo massimo rispetto a tutti gli altri nodi.

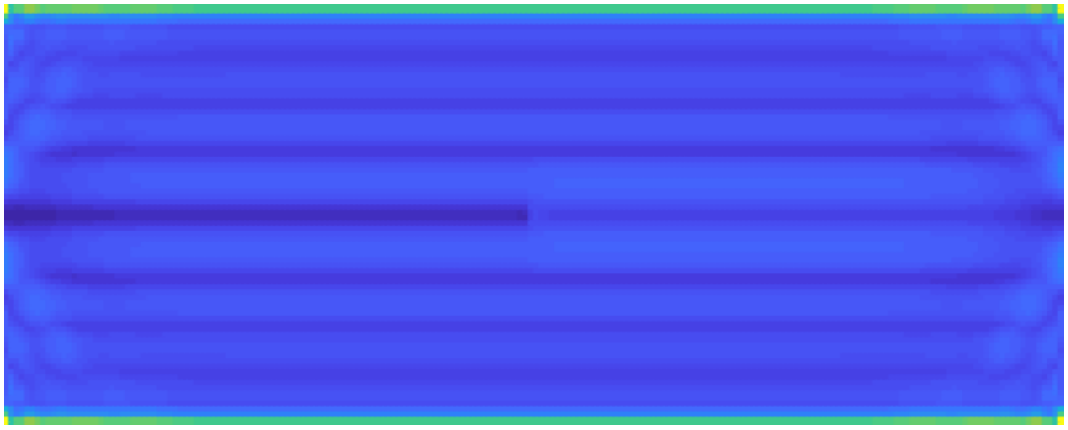

Figura 5.3: Il carico provoca delle onde elastiche che si propagano dai lati superiore e inferiore.

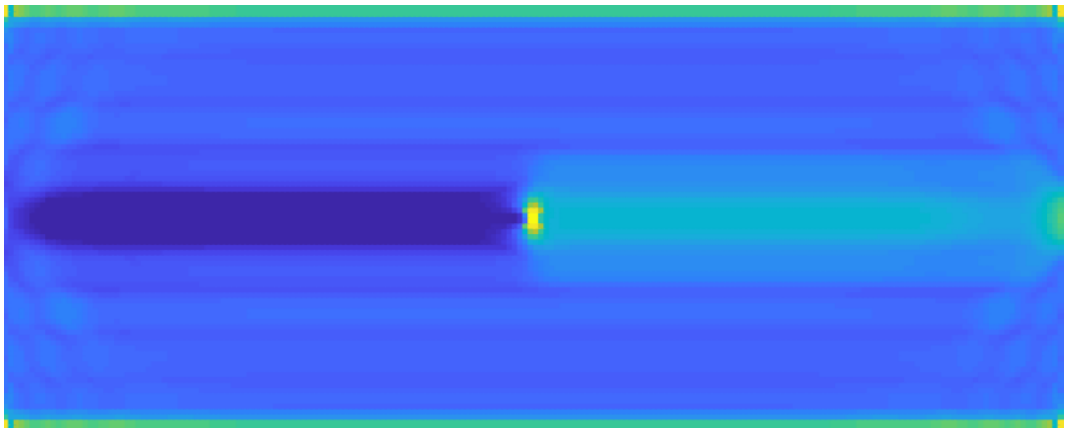

Figura 5.4: Il massimo punto di deformazione si concentra sulla punta della cricca.

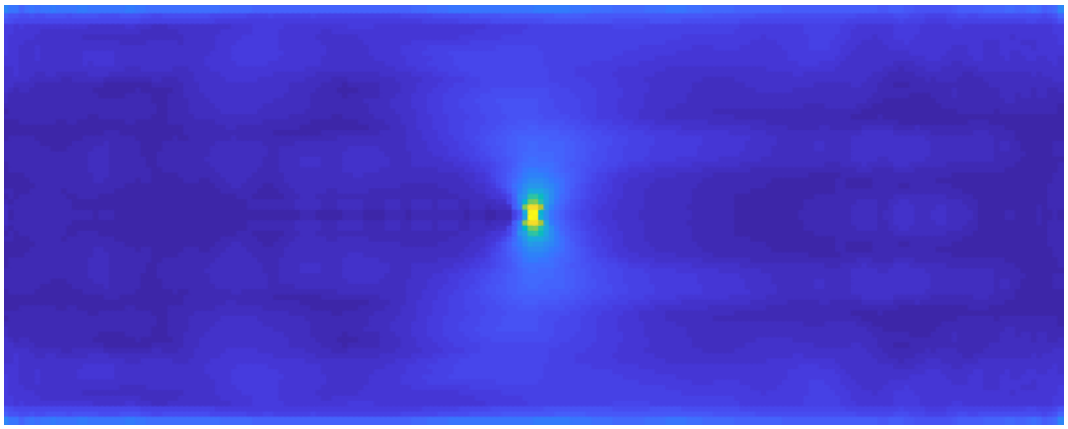

Figura 5.5: La punta della cricca è il punto più in tensione di tutta la lastra, e la cricca sta per aprirsi.

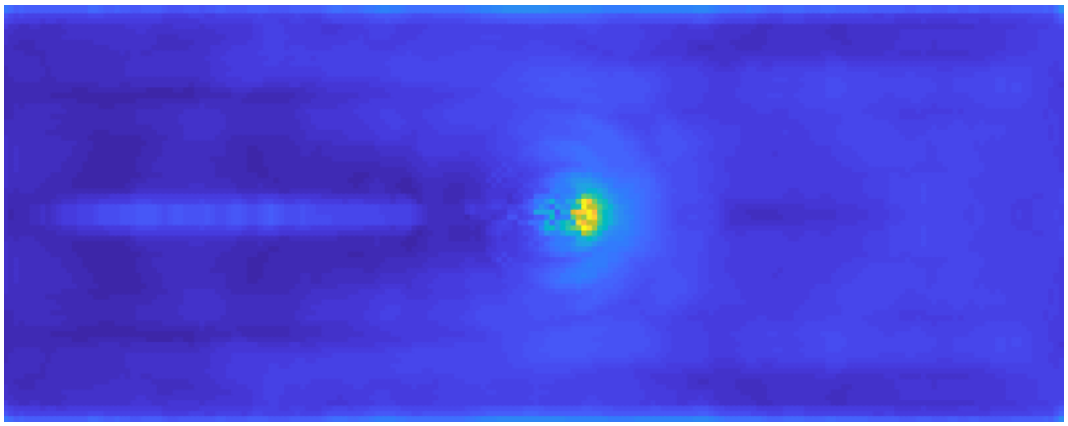

Figura 5.6: La cricca si sta aprendo e si originano onde sferiche dai punti di rottura dei legami.

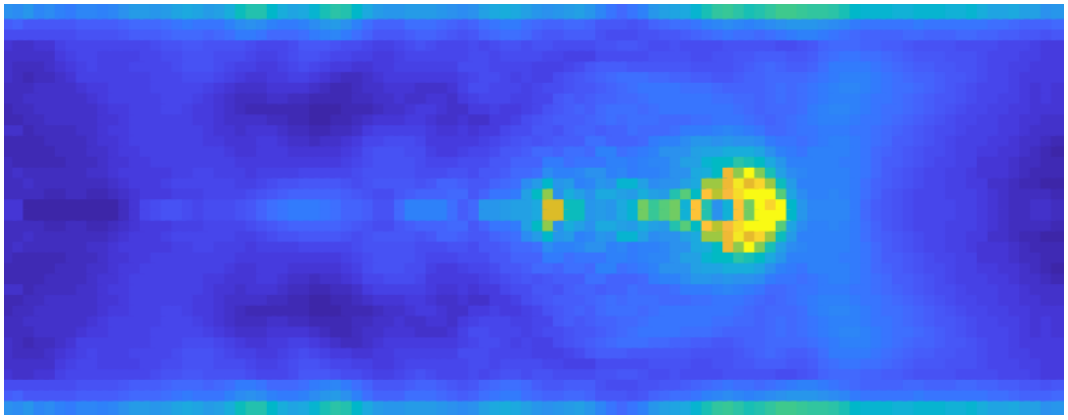

Figura 5.7: La cricca sta per dividersi.

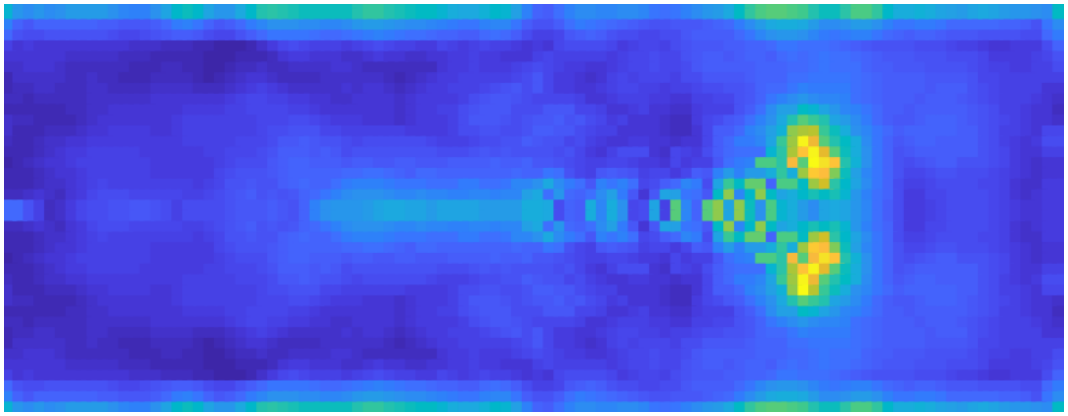

Figura 5.8: La cricca si è divisa e continua a propagarsi.

Capitolo 6

Discussioni

Nel suo stato attuale l'algoritmo è in grado di eseguire in sicurezza soltanto δ-convergenza, ovvero mantenendo il rapporto m tra orizzonte locale δ e discretizzazione Δ costante.

Mentre la prima parte (pre-iterazione) è stata estensivamente testata con forme diverse e insolite, quelle di iterazione ed elaborazione sono state eseguite concentrandosi più sulla realizzazione di risultati compatibili con quelli di Ha e Bobaru.

Riducendo la lunghezza di discretizzazione si risente del fenomeno di imprecisione numerica che rende instabile l'andamento della cricca poco prima di raggiungere il punto di apertura a Y, provocando asimmetrie nella forma della cricca.

Questo fenomeno è stato riscontrato anche in Ha and Bobaru (2011):

> We also performed computations in which the origin of the system of coordinates is not at the center of the sample (coordinates are no longer symmetric with respect to the crack line), and the results showed an increased asymmetry in the crack branching paths.

Osservando infatti in prossimità dell'apertura della cricca si nota un'oscillazione del percorso e un'apertura asimmetrica della cricca anche se l'oggetto simmetrico viene posto sotto un carico simmetrico. Ciò accade quando il centro del problema dinamico (in questo

esperimento la punta della cricca iniziale) non si trova sull'origine del sistema di riferimento: tali perturbazioni leggere sono date dalla non simmetria delle coordinate espresse con valori discretizzati $\mathbb{F}$ rispetto ai numeri reali $\mathbb{R}$.

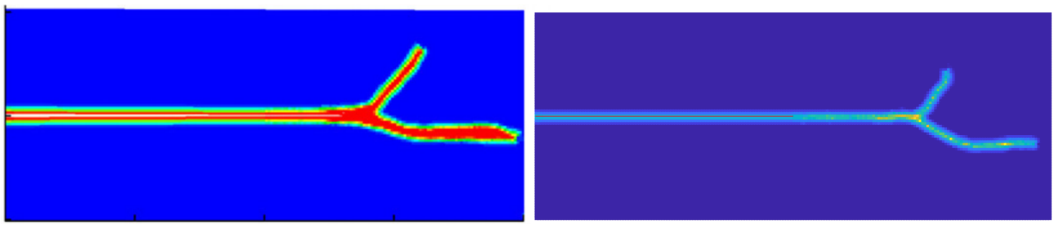

Figura 6.1: Confronto tra i risultati ottenuti da Bobaru (sinistra) con quelli ottenuti dall'algoritmo di questa stesura (destra).

Capitolo 7

Risultati

I risultati sono concordi con quanto riportato in *Studies of dynamic crack propagation and crack branching with peridynamics* di Ha e Bobaru nell'esempio della lastra di vetro Duran 50. Applicando una tensione di $12MPa$ sulle estremità superiore e inferiore della lastra di $40mm$ di altezza e $100mm$ di larghezza in presenza di una cricca nella sua metà sinistra lunga $50mm$, dopo circa $10\mu s$ si comincia ad aprire la cricca e a circa $30\mu s$ si apre a forma di Y, raggiunta un'ascissa attorno ai $70mm$.

Interpolando la matrice dei coefficienti di danno come una funzione bidimensionale e ricavandone la curva di livello con valore $0,41$ si evidenzia in una esplicita curva l'andamento della cricca.

Più la discretizzazione effettuata è fine, più stretta e convergente alla cricca effettiva sarà la curva di livello.

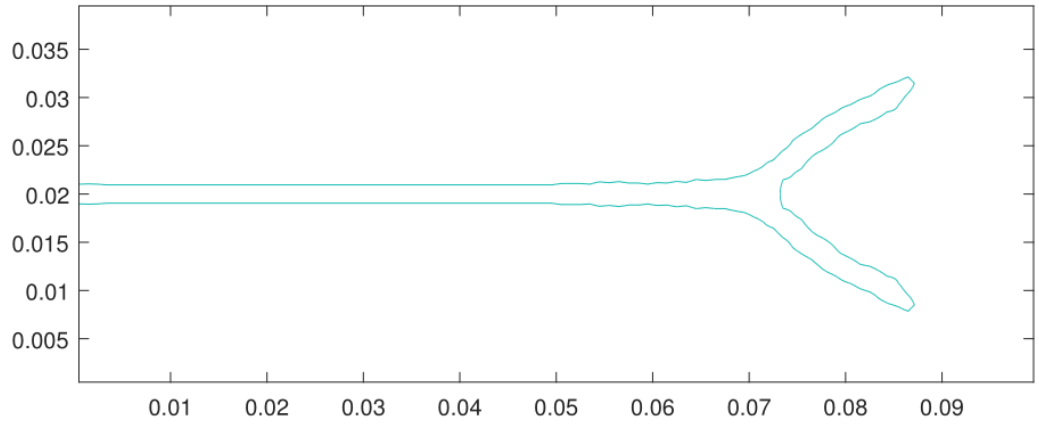

Figura 7.1: Curva di cricca finale: tale curva rappresenta la zona in cui si è propagata la cricca nel corso della simulazione.

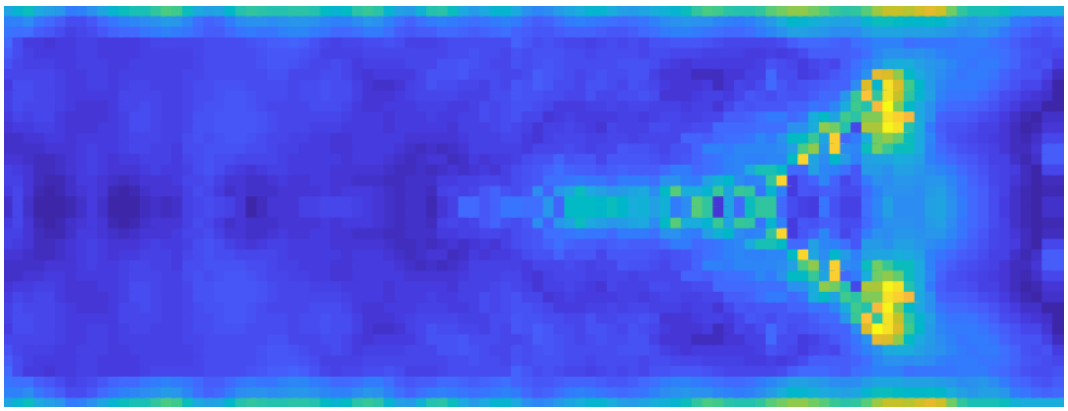

Figura 7.2: Allungamenti relativi per ogni nodo dopo 46μs.

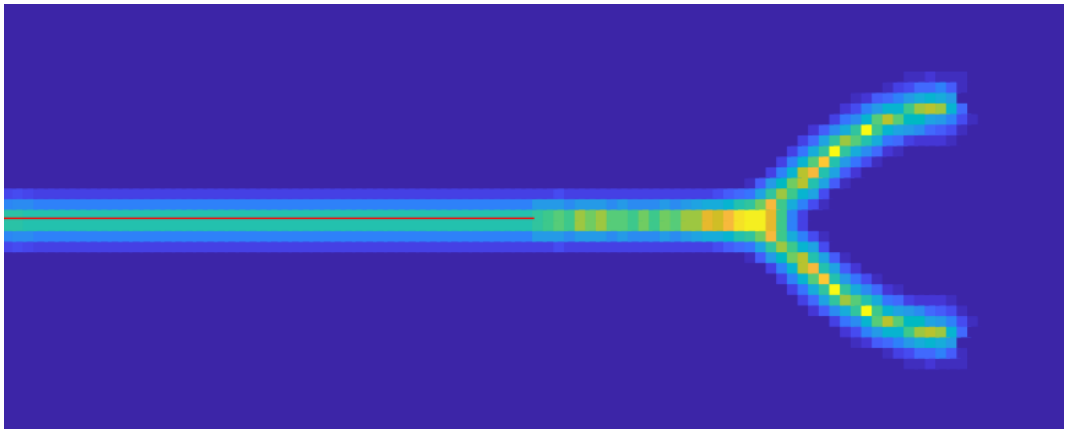

Figura 7.3: Distribuzione dei coefficienti di danno per una lamina di vetro Duran 50 con discretizzazione di 1mm dopo $46\mu s$ dall'applicazione del carico.

Capitolo 8

Conclusioni

La versione *bond based* della teoria *Peridynamics* è stata proposta in questa stesura ed implementata per mezzo di un codice sviluppato in ambiente Matlab; tuttavia ciò non limita la compatibilità dei risultati tra diverse implementazioni e ne illustra le capacità, a prescindere dal motore di processione e dalle scelte di strutture di dati.

Per come è stato strutturato l'algoritmo, una volta definita la forma dell'oggetto, i parametri fisici e dove vengono applicati i carichi, è facilmente adattabile per testare la sua compatibilità con altri tipi di esperimenti numerici.

Inoltre le quantità vettoriali sono state definite con un array dedicato a ogni dimensione: basandosi su ciò sarà possibile partire da questo algoritmo per passare alla terza dimensione e simulare oggetti tridimensionali.

La prima parte di algoritmo dedicata alla discretizzazione è stata testata su forme arbitrarie e non convenzionali come forme non semplicemente connesse o concave. Con opportuni accorgimenti è possibile anche simulare il comportamento di oggetti separati o disconnessi.

La sezione di iterazione nel tempo con applicazione della teoria peridinamica, basata sull'algoritmo *Verlet-Velocity*, è stata inserita in una `function` dedicata per rendere eseguibili iterazioni lunghe con passaggi intermedi, come ad esempio il salvataggio di immagini plot periodiche per creare animazioni video.

L'analisi dell'esempio proposto da Ha and Bobaru (2010) condotta grazie al codice sviluppato ha permesso di riprodurre in maniera accurata i risultati presenti in letteratura.

Il passo finale è stato quello di fornire una visualizzazione diretta della cricca a partire da curve di livello, potendo in future applicazioni analizzare i punti di tali curve per studiare effetti come la velocità di propagazione della cricca.

L'algoritmo nella sua interezza è disponibile pubblicamente nella *repository* all'indirizzo: github.com/apmox/peridynamics

Capitolo 9

Ringraziamenti

Ringrazio amici e colleghi compagni di corso, il loro supporto in questi anni è stato incommensurabile.

Ringrazio i docenti per la loro chiarezza e disponibilità, in particolare il professor Zaccariotto per la sua pazienza, precisione e cura.

Ringrazio particolarmente i miei genitori per avermi permesso di seguire questo percorso e per continuare a sostenermi sempre.

Bibliografia

Belytschko, Ted, and Tom Black. 1999. "Elastic Crack Growth in Finite Elements with Minimal Remeshing." *International Journal for Numerical Methods in Engineering* 45 (5): 601–20.

Bolander Jr, JE, and Shigehiko Saito. 1998. "Fracture Analyses Using Spring Networks with Random Geometry." *Engineering Fracture Mechanics* 61 (5-6): 569–91.

Döll, W. 1975. "Investigations of the Crack Branching Energy." *International Journal of Fracture* 11 (1): 184–86.

Gerstle, Walter, Nicolas Sau, and Stewart Silling. 2007. "Peridynamic Modeling of Concrete Structures." *Nuclear Engineering and Design* 237 (12-13): 1250–8.

Ha, Youn Doh, and Florin Bobaru. 2010. "Studies of Dynamic Crack Propagation and Crack Branching with Peridynamics." *International Journal of Fracture* 162 (1-2): 229–44.

Ha, Youn Doh, and Florin Bobaru. 2011. "Characteristics of Dynamic Brittle Fracture Captured with Peridynamics." *Engineering Fracture Mechanics* 78 (6): 1156–68.

Hillerborg, Arne, Mats Modéer, and P-E Petersson. 1976. "Analysis of Crack Formation and Crack Growth in Concrete by Means of Fracture Mechanics and Finite Elements." *Cement and Concrete Research* 6 (6): 773–81.

Kilic, Bahattin, and Erdogan Madenci. 2010. "Coupling of Peridynamic Theory and the Finite Element Method." *Journal of Mechanics of Materials and Structures* 5 (5): 707–33.

Lapidus, Leon, and George F Pinder. 2011. *Numerical Solution of Partial Differential Equations in Science and Engineering.* John Wiley & Sons.

Macek, Richard W, and Stewart A Silling. 2007. "Peridynamics via Finite Element Analysis." *Finite Elements in Analysis and Design* 43 (15): 1169–78.

Nguyen, Vinh Phu, Timon Rabczuk, Stéphane Bordas, and Marc Duflot. 2008. "Meshless Methods: A Review and Computer Implementation Aspects." *Mathematics and Computers in Simulation* 79 (3): 763–813.

Okabe, Atsuyuki, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. 2009. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* Vol. 501. John Wiley & Sons.

Ramulu, M, and AS Kobayashi. 1985. "Mechanics of Crack Curving and Branching—a Dynamic Fracture Analysis." In *Dynamic Fracture*, 61–75. Springer.

Ravi-Chandar, Krishnaswamy. 1998. "Dynamic Fracture of Nominally Brittle Materials." *International Journal of Fracture* 90 (1-2): 83–102.

Scudellaro, L. 2012. "Nonlocal Theories for the Study of Continuum: A Finite Elements Method Implementation." Università degli studi di Padova.

Silling, Stewart A. 2000. "Reformulation of Elasticity Theory for Discontinuities and Long-Range Forces." *Journal of the Mechanics and Physics of Solids* 48 (1): 175–209.

Silling, Stewart A, and Ebrahim Askari. 2005. "A Meshfree Method Based on the Peridynamic Model of Solid Mechanics." *Computers & Structures* 83 (17-18): 1526–35.

Streit, R, and Il Finnie. 1980. "An Experimental Investigation of Crack-Path Directional Stability." *Experimental Mechanics* 20 (1): 17–23.

Verlet, Loup. 1967. "Computer" Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules." *Physical Review* 159 (1): 98.

Xu, X-P, and Alan Needleman. 1994. "Numerical Simulations of Fast Crack Growth in Brittle Solids." *Journal of the Mechanics and Physics of Solids* 42 (9): 1397–1434.

Yu, Kebing. 2011. “Enhanced Integration Methods for the Peridynamic Theory.” PhD thesis, Kansas state university.

Zaccariotto, Mirco, Fabio Luongo, U Galvanetto, and others. 2015. “Examples of Applications of the Peridynamic Theory to the Solution of Static Equilibrium Problems.” *The Aeronautical Journal* 119 (1216): 677–700.

Zhou, SJ, PS Lomdahl, R Thomson, and BL Holian. 1996. “Dynamic Crack Processes via Molecular Dynamics.” *Physical Review Letters* 76 (13): 2318.