UNIVERSITÀ DEGLI STUDI DI CAGLIARI

FACOLTÀ DI SCIENZE

Corso di Laurea Triennale in Informatica

Implementing Topological Sheet Operators in Hexahedral Meshes for the Cinolib Library

Relatore
Dott. Gianmarco Cherchi

Studente
Federico Meloni
Matr. N. 65469

ANNO ACCADEMICO 2019/2020

Nowdays volumetric meshes nowadays represent a powerful tool to perform, among other things, physical simulations on models of real objects, such as mechanical components, vehicles or even buildings, anticipating any defects or malfunctions in a very precise way.
This thesis focuses on the implementation of various operations aimed at simplifying and/or improving hexmeshes (volumetric meshes composed exclusively by hexahedra), such as insertion and collapse of a hexahedral sheets inside the mesh, and their integration into the open source Cinolib library, which allows the visualization and manipulation of surface meshes and volume meshes.

Le mesh volumetriche oggigiorno rappresentano un potente strumento per effettuare, tra le altre cose, simulazioni fisiche su modelli di oggetti reali, come componenti meccaniche, veicoli o addirittura edifici, andando ad anticipare eventuali difetti o malfunzionamenti in maniera molto precisa.
Il lavoro di tesi verte sull'implementazione di varie operazioni volte alla semplificazione e/o perfezionamento di hexmesh (mesh volumetriche composte esclusivamente da esaedri), quali inserimento e collassamento di un livello di esaedri all'interno della mesh, e la loro integrazione nella libreria open source Cinolib, che permette la visualizzazione e manipolazione di mesh di superficie e mesh volumetriche.

Indice

1 Introduzione **1**

2 State of the art **5**
2.1 Mesh Volumetriche 5
2.2 Cinolib 8

3 Implementazione **11**
3.1 Selezione degli esaedri 11
3.1.1 Edge di partenza 11
3.1.2 Catena di edge 11
3.1.3 Catena di esaedri 12
3.2 Inserimento 13
3.2.1 Preparazione 13
3.2.2 Esecuzione 13
3.2.3 Split semplice 14
3.2.4 Split doppio 15
3.2.5 Split triplo 16
3.2.6 Finalizzazione 16
3.3 Collassamento 16
3.3.1 Preparazione ed esecuzione 16
3.3.2 Collassamento semplice 17
3.3.3 Collassamento doppio 18
3.3.4 Singolarità 20
3.3.5 Finalizzazione 20

4 Risultati **21**
4.1 Risultati della selezione di un livello 21
4.2 Risultati dell'inserimento di un livello 22
4.3 Risultati del collassamento di un livello 22
4.4 Considerazioni complessive 23

5 Conclusioni **25**

6 Sviluppi futuri **27**

References **29**

Capitolo 1

Introduzione

Le mesh volumetriche oggigiorno rappresentano un potente strumento per effettuare, tra le altre cose, simulazioni fisiche su modelli di oggetti reali, come componenti meccaniche, veicoli o addirittura edifici, andando ad anticipare eventuali difetti o malfunzionamenti in maniera molto precisa.

Mentre le mesh di superficie, utilizzate in numerosi ambiti come videogiochi e animazione, rappresentano solo l'esterno del modello, le mesh volumetriche rappresentano anche la struttura interna della mesh e proprio questa caratteristica permette il loro utilizzo in ambiti come la Finite Element Analysis.

Mesh di superficie e volumetriche si pongono nel contesto della Computer Graphics, branca dell'informatica che si occupa di sintetizzare e manipolare digitalmente contenuti visivi, sia bidimensionali che tridimensionali.

In questa tesi si analizzeranno due algoritmi sviluppati per la libreria Cinolib [4] per la manipolazione di hexmesh, cioè mesh volumetriche [1] composte esclusivamente da esaedri. Il primo algoritmo andrà a trattare dell'aggiunta di un livello di esaedri all'interno di una hexmesh, mentre il secondo tratterà la rimozione di un livello da una hexmesh. I due algoritmi condividono una parte iniziale di selezione degli esaedri di partenza, a cui è stata dedicata una sezione a parte.

L'obiettivo finale di questo lavoro è quello di creare delle funzionalità che possano tornare utili a chi andrà a lavorare su mesh volumetriche di esaedri e che si possano integrare come chiamate di libreria in Cinolib.
Cinolib è una libreria open source ed header only scritta in C++ che permette la visualizzazione e la manipolazione di mesh di superficie e mesh volumetriche.

La funzione di inserimento prevede la creazione di un livello che funge da "cuscino" tra due livelli pre-esistenti della mesh: a livello teorico si crea uno spazio tra due livelli e si inserisce all'interno di questo spazio un terzo livello, ma, come vedremo in seguito, a livello pratico per raggiungere lo stesso obiettivo si passerà per una strada differente. Nella figura 1.1, il procedimento è schematizzato in 2D.

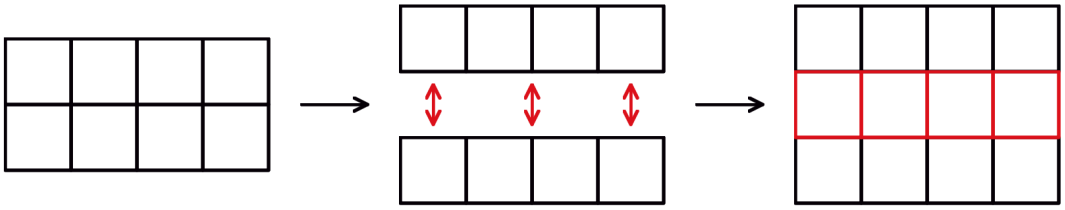

Figura 1.1: *Inserimento di un livello dal punto di vista teorico*

La funzione di collassamento, invece, prevede l'eliminazione di un livello di esaedri dalla mesh, andando ad occupare lo spazio lasciato vuoto con i livelli di esaedri adiacenti al livello appena eliminato. Come nel caso precedente, in seguito vedremo che per arrivare al risultato la pratica differisce leggermente dalla teoria. Di seguito, nella figura 1.2, l'algoritmo è schematizzato in 2D.

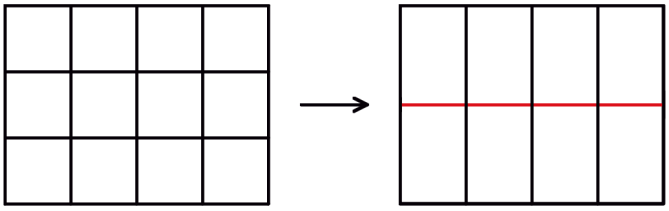

Figura 1.2: *Collassamento di un livello dal punto di vista teorico*

Queste operazioni risultano estremamente utili nel momento in cui vogliamo andare a ottimizzare, rifinire o semplificare una mesh.
A tal proposito, numerosi lavori in letteratura hanno al loro interno operazioni topologiche di sheet insertion e sono stati fatti numerosi studi al riguardo, per esempio l'articolo "Selective Padding for Polycube-Based Hexahedral Meshing" [2], espone un approccio per migliorare la qualità di esaedri appartenenti a una hex-mesh estratta da policubi usando operatori dello stesso genere.

Altro esempio può essere la pubblicazione "An approach to achieving optimized complex sheet inflation under constraints" [3], del 2016, dove si presenta un algoritmo per l'inserimento applicato a sheet complessi (vedi figura 1.3).

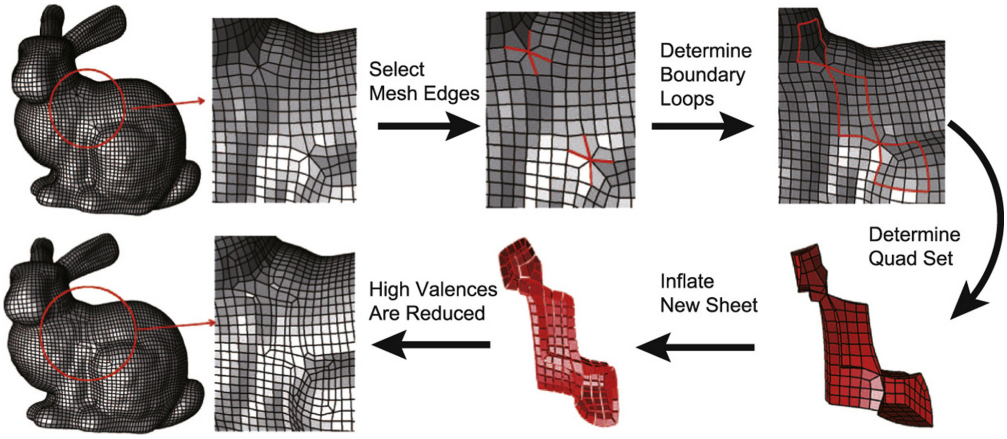

Figura 1.3: *Schema della pipeline introdotta in [3]*

Altri metodi per il miglioramento della qualità degli esaedri, sempre mediante l'utilizzo di operatori di questo tipo, sono esposti della pubblicazione "Hex mesh topological improvement based on frame field and sheet adjustment" [5].

Capitolo 2

State of the art

2.1 Mesh Volumetriche

Una mesh è una struttura dati che ci permette di rappresentare, in maniera discreta un oggetto, nello spazio tridimensionale attraverso delle primitive quali vertici, edge, facce e poliedri (nel caso delle mesh volumetriche).
Al contrario delle mesh di superficie, che rappresentano solo la superficie esterna di un oggetto tridimensionale, le mesh volumetriche rappresentano anche la struttura interna degli oggetti [1].

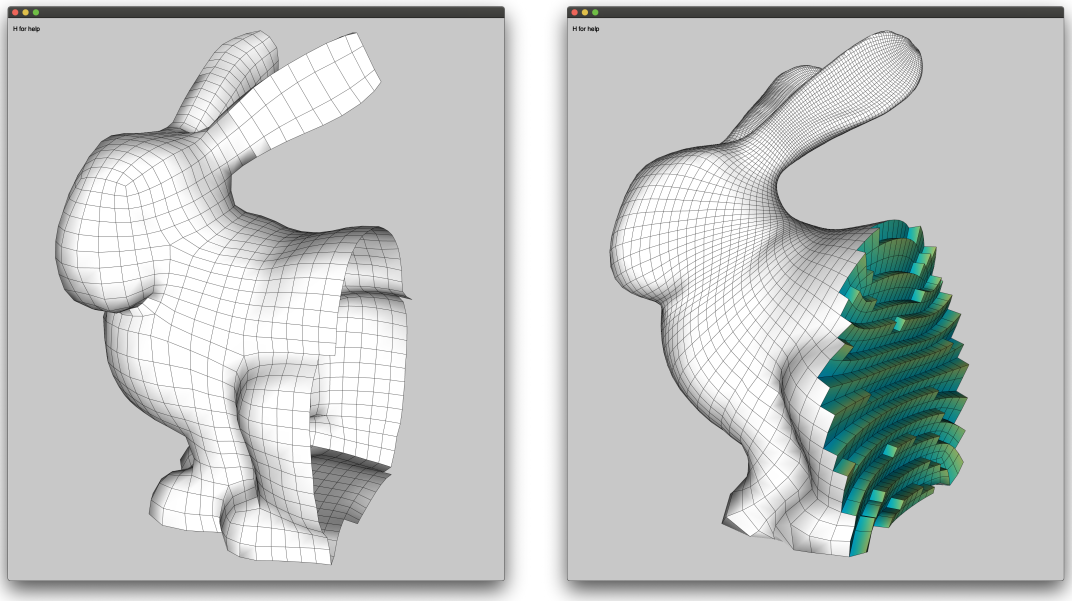

Figura 2.1: *Confronto tra l'interno di una mesh di superficie (sinistra) e di una mesh volumetrica (destra).*

Le mesh sono definite da:

- **Geometria**: l'insieme dei vertici che compongono il modello, ognuno dei quali è caratterizzato da una posizione in $\mathbb{R}3$ descritta tramite un vettore tridimensionale;

- **Connettività (topologia)**: edge, facce e poliedri sono i vari modi in cui i vertici possono connettersi tra di loro;

- **Attributi**: un qualsiasi valore (scalare o vettoriale) che varia linearmente sulla superficie, associato agli elementi della mesh (es. normale).

Caratteristiche generali: Data una geometria che definisce i vertici della mesh, la parte più importante della mesh è la topologia.
La topologia va a definire i vari livelli (e dimensioni) di connessione tra i vertici e di conseguenza le adiacenze tra le varie parti della mesh.
Gli edge (letteralmente "spigoli") sono la connessione tra due vertici (1 dimensione), le facce sono poligoni formati da un insieme di edge (2 dimensioni) e i poliedri non sono quindi altro che un insieme di facce che delimitano uno spazio chiuso (3 dimensioni).
Questo ultimo passo nella topologia è ciò che ci permette di descrivere la struttura interna di una mesh ed è ciò che, come anticipato prima, distingue le mesh volumetriche dalle mesh di superficie.
Ognuno di questi elementi definiti dalla connettività ha delle adiacenze, nello specifico:

- Un **vertice** può essere adiacente a un altro vertice (collegati da un edge), a una faccia (fa parte di essa) o a un poliedro (fa parte di esso)

- Un **edge** può essere adiacente a altro edge (hanno un vertice in comune), a una faccia (fa parte di essa) o a un poliedro (fa parte di esso)

- Una **faccia** può essere adiacente a un altra faccia (hanno un edge in comune) o a un poliedro (fa parte di esso)

- Un **poliedro** può essere adiacente ad un altro poliedro (hanno una faccia in comune)

Da sottolineare che queste relazioni di adiacenza sono simmetriche.

Classificazione: Generalmente andiamo a distinguere le mesh volumetriche in categorie basate sul poliedro che le vanno a comporre.
I poliedri più utilizzati sono i **tetraedri** e gli **esaedri**, ma possiamo trovare mesh volumetriche composte da poliedri aventi un qualsiasi numero di vertici e volume mesh con diversi tipi di poliedri contemporaneamente, queste mesh vengono definite ibride (es. una mesh può essere composta sia da tetraedri che da esaedri).
Le mesh composte da tetraedri sono chiamate **tetmesh**, sono le più semplici da costruire e ogni altro poliedro può essere scomposto in tetraedri. Le mesh composte da esaedri (anche detti cuboidi) sono dette **hexmesh**, più complicate e difficili da gestire, e sono quelle su cui vertono gli algoritmi scritti in questa tesi.

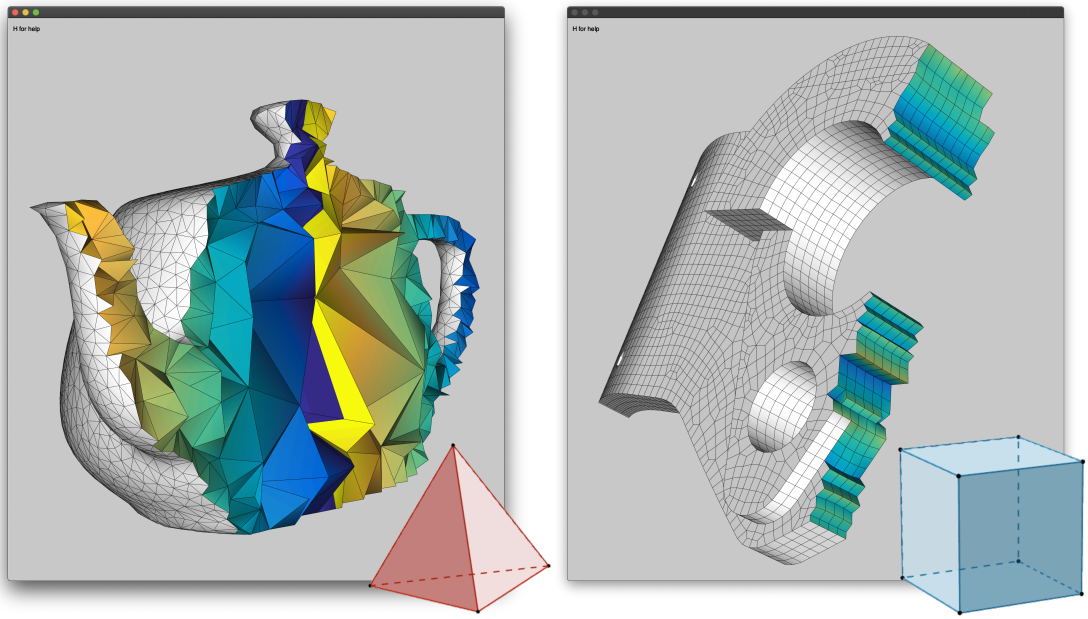

Figura 2.2: *Confronto tra le sezioni di una tetmesh (sinistra) e una hexmesh (destra)*

Campi di utilizzo: Mentre le mesh di superficie sono usate in numerosi campi come videogiochi e modellazione, le mesh volumetriche vengono principalmente utilizzate per simulazioni fisiche, in particolare Finite Element Analysis.
Grazie alle mesh volumetriche si possono simulare e verificare virtualmente le proprietà strutturali degli oggetti rappresentati, un esempio sono le simulazioni di carico o le simulazioni termodinamiche.
Un altro campo di utilizzo in cui le mesh volumetriche stanno iniziando ad essere utilizzate è l'animazione, in quanto riescono a simulare una fisica degli oggetti in scena più realistica.

Per essere utilizzata in delle simulazioni una mesh volumetrica deve avere elementi di buona "qualità", cioè la loro forma dev'essere lontana dall'essere degenere (piatta o concava) ed essere il più possibile simile ad un cubo perfetto ($SJ \simeq 1$). In linea generale, una hexmesh è più difficile da costruire, ma offre simulazioni più efficienti (a parità di risoluzione) o più accurate (a parità di tempo di esecuzione) rispetto a una tetmesh rappresentante il medesimo oggetto.

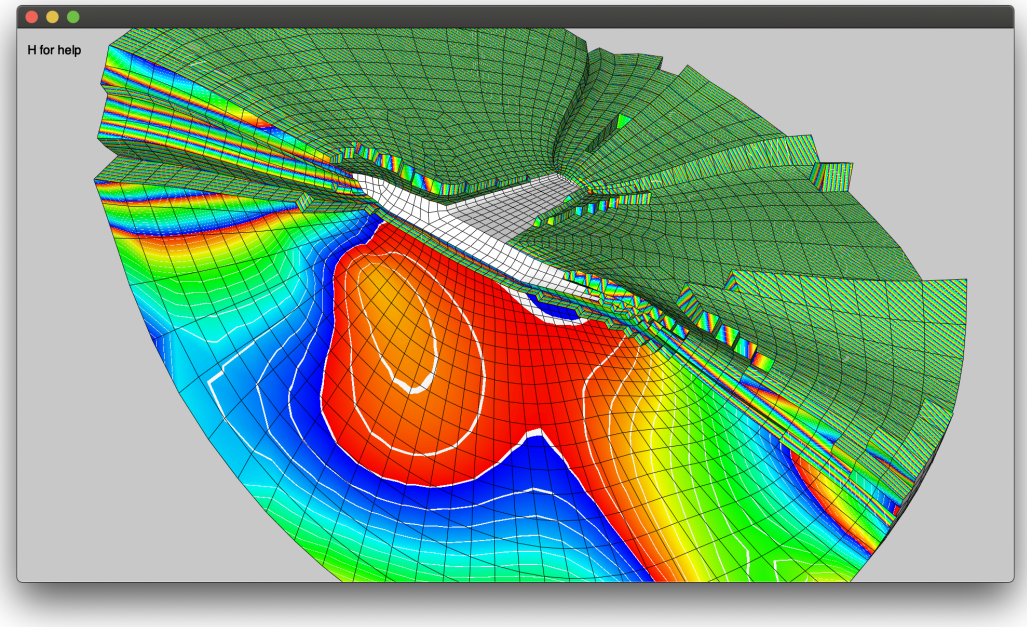

Figura 2.3: *Simulazione di carico su un aereo di linea*

2.2 Cinolib

Cinolib è una libreria header only, scritta in C++, preposta alla visualizzazione e manipolazione via codice di mesh di superficie e mesh volumetriche generiche (non solo trimesh/quadmesh e tetmesh/hexmesh ma qualsiasi tipo di mesh) [4].
Grazie alla sua struttura gerarchica Cinolib permette di implementare algoritmi e operatori una sola volta ed applicarli a numerosi tipi di mesh, in questo modo si va ad evitare la ripetizione del codice e si facilità la manutenzione del software e il bug fixing.

Architettura: In Cinolib la topologia di ogni mesh è rappresentata da un insieme di liste, una per ogni tipo di connettività. Questa rappresentazione permette di

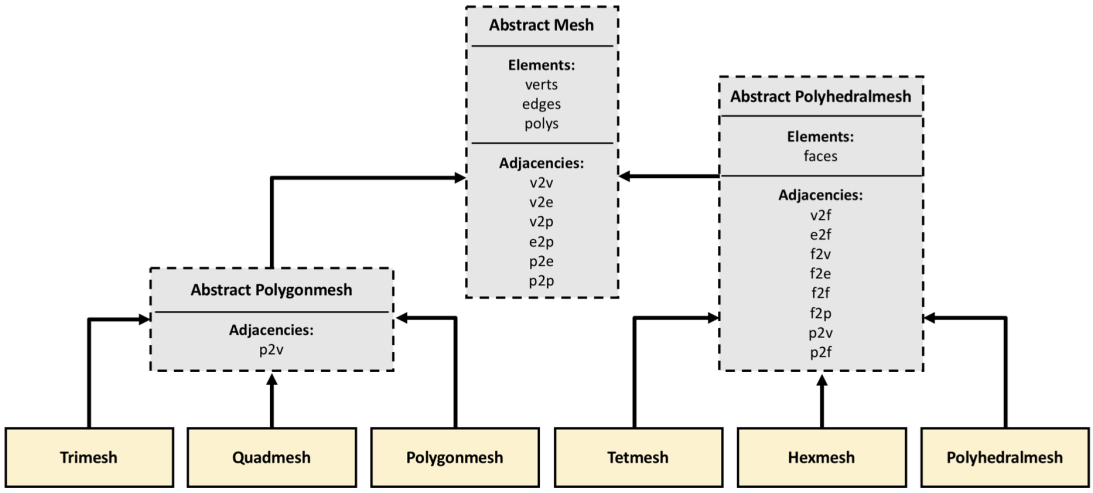

Figura 2.4: *Struttura della libreria Cinolib.*

poter gestire le adiacenze sia delle mesh di superficie che delle mesh volumetriche con lo stesso codice, in quanto gli algoritmi di adiacenza sono gli stessi.
Entrando nello specifico in Cinolib troviamo 3 livelli di classi:

- **Primo livello**, **concetti generali di tutte le mesh**: Mesh di superficie e mesh volumetriche condividono numerosi concetti, per questo motivo all'apice della gerarchia troviamo `Abstract Mesh`, al cui interno troviamo la definizione degli elementi (vertici, edge e poly) e delle loro adiacenze. Da notare che l'elemento "poly" rappresenta, nel caso delle mesh di superficie, i poligoni, mentre, nel caso delle mesh volumetriche rappresenta, i poliedri;

- **Secondo livello**, **introduzione concetto di mesh di superficie e mesh volumetriche**: al secondo livello della gerarchia troviamo la definizione di mesh di superficie generali e di mesh volumetriche generali attraverso due classi che estendono `Abstract Mesh`: `Abstract Polygonmesh` e `Abstract Polyhedralmesh`. In entrambe le classi viene definita l'adiacenza tra poly e vertici, differente tra mesh di superficie e mesh volumetriche, e viene introdotto il concetto di faccia nelle mesh volumetriche e le relative adiacenze (al contrario delle mesh di superficie, i poly non rappresentano i poligoni, quindi le facce, ma i poliedri).

- **Terzo livello**, **tipologie specifiche di mesh**: alla base della gerarchia troviamo le classi che definiscono specifiche tipologie di mesh e gli algoritmi ad esse associate, in particolare troviamo `Trimesh`, `Quadmesh` e `Polygonmesh` per le mesh di superficie e `Tetmesh`, `Hexmesh`, `Polyhedralmesh` per le mesh volumetriche.

Funzionalità Cinolib al suo interno implementa una larga varietà di tools e algoritmi, utili per le operazioni più frequenti.
Alcune tra le funzionalità che troviamo:

- Operatori differenziali discreti (es. gradiente, divergenza e laplassiana)
- Generazione e elaborazione di mesh
- Campi
- Distanze e path
- Visualizzazione/rendering
- Algoritmi per la stampa 3D
- Parser per i principali formati di mesh file

Capitolo 3

Implementazione

In questo capitolo si analizzeranno i due algoritmi di inserimento e di collassamento di un livello di esaedri dal punto di vista implementativo. In entrambe le operazioni si parte con la selezione del livello, si procede con l'inserimento/collassamento, si cancellano gli esaedri originali e si aggiorna la mesh.

3.1 Selezione degli esaedri

Per la selezione degli esaedri è stato sviluppato un algoritmo di flooding a partire da un edge.
In un primo momento si andrà a creare una catena di edge topologicamente paralleli, poi grazie a questa si andrà a creare la catena di esaedri rappresentante il livello.

3.1.1 Edge di partenza

L'edge da cui partire per la creazione del livello può essere selezionato in due maniere:

- Cinolib fornisce un sistema di picking che permette di selezionare da UI un edge che poi può essere passato alla funzione di selezione del livello;
- indice dell'edge da cui si vuole partire passato via codice come parametro.

3.1.2 Catena di edge

Dato un edge di partenza, saranno necessarie due strutture dati per selezionare la catena di edge: una coda e un set, entrambi popolati inizialmente dall'edge di partenza. La coda servirà a tenere traccia degli edge su cui ancora bisogna

effettuare i controlli, mentre il set definirà gli edge già visitati, quindi la catena.
Per ogni edge prelevato dalla coda si andranno a identificare le facce adiacenti e, per ognuna di esse, si andrà a cercare l'edge opposto a quello prelevato dalla coda. Quest'ultima operazione è possibile andando a prendere l'unico edge nella faccia che non è adiacente a quello in esame. Se l'edge opposto non fa parte del set degli edge già visitati allora viene inserito sia nella coda, sia nel set, altrimenti si passa alla faccia successiva. Una volta finite le facce adiacenti all'edge si passa all'edge successivo, fino all'esaurimento della coda.
All'esaurimento della coda il set rappresenterà la catena di edge.

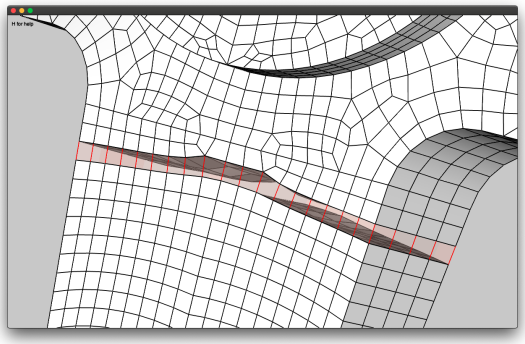

Figura 3.1: *Esempio di catena di edge in una mesh volumetrica*

3.1.3 Catena di esaedri

Data la catena di edge calcolata in precedenza, per calcolare il livello di esaedri sarà necessario anche in questo caso l'utilizzo di un set.
Scorrendo la catena di edge calcolata in precedenza, per ogni edge, si inseriscono nel set tutti gli esaedri adiacenti, il set ci garantirà l'assenza di doppioni.

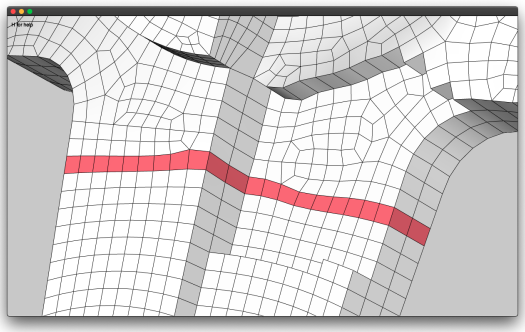

Figura 3.2: *Sezione di una mesh volumetrica da cui è stato selezionato un livello di esaedri*

3.2 Inserimento

A livello teorico, come abbiamo già visto con la figura 1.1, l'obiettivo dell'algoritmo è creare uno spazio tra due livelli di esaedri e inserirne un terzo, a livello pratico possiamo arrivare allo stesso risultato dividendo in due un livello pre-esistente, in quanto si va a creare un livello aggiuntivo in mezzo alla mesh.
Questo "split" si concretizza ulteriormente nell'operazione di rimozione del livello selezionato e la sua sostituzione con due livelli che occupano lo stesso spazio. Per fare ciò bisognerà servirsi della catena di esaedri, di quella di edge e altri espedienti.

3.2.1 Preparazione

Come preparazione alla creazione dei due nuovi livelli è necessario creare nuovi vertici.
Per ogni edge della catena viene creato e associato (grazie a una mappa) un vertice, posizionato al centro dell'edge a cui viene associato. Questi edge saranno sicuramente necessari, ma potrebbe essere necessario crearne di altri.

3.2.2 Esecuzione

Il livello può assumere varie conformazioni, fino a intersecarsi con se stesso. In base al numero di volte che il livello attraversa un esaedro identifichiamo 3 casi:

- **Split semplice**: l'esaedro è attraversato solo in una direzione dal livello;
- **Split doppio**: l'esaedro è attraversato in due direzioni dal livello, significa che per quell'esaedro il livello sta intersecando se stesso;
- **Split triplo**: l'esaedro è attraversato in tre direzioni dal livello, significa che per quell'esaedro il livello sta intersecando se stesso due volte.

Per ogni esaedro nel livello si procede allo split andando a identificare a quale delle tre categorie appartiene. Per riconoscere la categoria di appartenenza, preso un vertice qualsiasi, si conta quanti edge localmente adiacenti al vertice facciano parte della catena di edge: ogni edge che appartiene alla catena corrisponde a una direzione di attraversamento del livello (es. se due degli edge localmente adiacenti al vertice appartengono alla catena di edge allora l'esaedro dovrà subire uno split triplo, come mostrato nella figura 3.3)

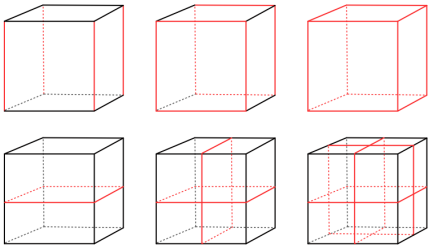

Figura 3.3: *Le colonne rappresentano in ordine: split semplice, doppio e triplo. In alto troviamo gli edge facenti parte della catena per ogni categoria, mentre in basso lo split scheme.*

3.2.3 Split semplice

Se un esaedro è attraversato in una sola direzione dal livello, come spiegato nel paragrafo precedente, vuol dire che non è sito di intersezione del livello con se stesso, quindi verranno creati due esaedri al posto di quello attuale.
Per la creazione dei nuovi esaedri si vanno a prendere le due facce che non hanno edge appartenenti alla catena e si useranno come riferimento per i due nuovi esaedri. Per riconoscerle basterà controllare, per ogni faccia, che due edge adiacenti non facciano parte della catena, poiché se uno fa parte della catena ne farà parte anche quello opposto.
Una volta identificate le due facce, per ognuna di queste si crea una corrispondenza tra ogni vertice della faccia e l'edge localmente adiacente al vertice facente parte della catena. Questa corrispondenza (per via transitiva grazie alla mappa) equivale alla corrispondenza tra vertice della faccia e uno dei nuovi vertici.
Dopo questo passaggio si conoscono gli otto vertici per ognuno dei due esaedri da creare.

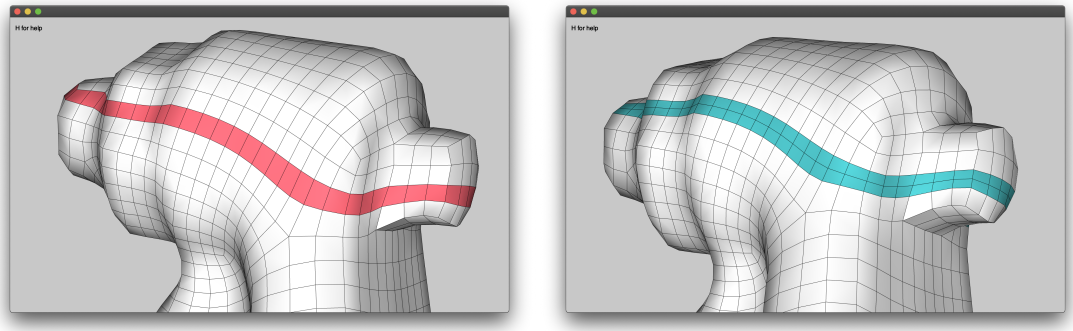

Figura 3.4: *In rosso il livello prima dello split, in blu dopo*

3.2.4 Split doppio

Un esaedro attraversato dal livello per due direzioni è sito di un intersezione del livello con se stesso, al suo posto verranno creati 4 esaedri.
In questo caso avremo due facce con tutti gli edge facenti parte della catena, mentre le altre quattro facce avranno solo due edge ciascuna appartenenti alla catena.
I vertici creati in corrispondenza di ogni edge facente parte della catena non sono sufficienti per uno split doppio, è necessario creare altri due vertici in corrispondenza delle due facce con tutti gli edge appartenenti alla catena (vengono utilizzati i centroidi delle due facce come nuovi vertici in maniera arbitraria).
Una volta creati i due vertici si identificano due facce opposte tra quelle con soli due vertici appartenenti alla catena (quale tra le due coppie di facce è indifferente) e sulla base di ogni edge non facente parte della catena di queste due facce si crea uno dei quattro esaedri che andranno a sostituire quello in questione. Da sottolineare il fatto che identificare queste due facce non sia strettamente necessario, ma averle come riferimento semplifica e riduce i controlli da effettuare per mettere nel giusto ordine i vertici al momento della creazione dei nuovi esaedri.
Dato uno di questi edge, per ognuno dei suoi vertici sappiamo che:

- è localmente adiacente a due edge facenti parte della catena: corrispondenza a due dei nuovi vertici grazie alla mappa;
- fa parte di una delle due facce dell'esaedro con tutti gli edge facenti parte della catena: corrispondenza a uno dei due vertici aggiuntivi appena creati.

Da queste informazioni ricaviamo otto vertici, che ordiniamo grazie alle facce identificate in precedenza e creiamo il nuovo esaedro.
Questo processo viene iterato quattro volte, una per ogni edge del poliedro al di fuori della catena.

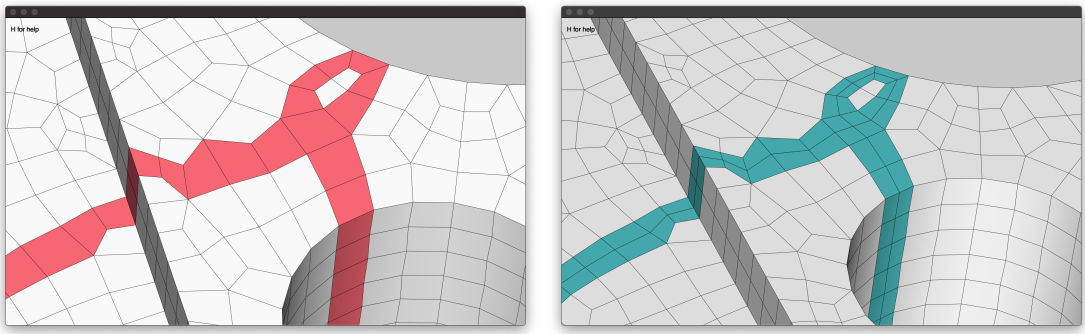

Figura 3.5: *In rosso il livello prima dello split, in blu dopo*

3.2.5 Split triplo

Se un esaedro è attraversato dal livello per tre direzioni vuol dire che è sede di una doppia intersezione del livello con se stesso, quindi andrà suddiviso in otto sub-esaedri.
Ogni edge di questo esaedro fa parte della catena, quindi prenderemo dalla mappa tanti vertici quanti edge nel poliedro.
In questo caso si creano ben sette vertici supplementari, uno in corrispondenza del centroide di ogni faccia e uno in corrispondenza del centroide dell'esaedro.
Per ogni vertice del poliedro si andrà a costruire un esaedro con otto vertici di cui:

- uno è il vertice stesso;
- tre sono quelli associati alle tre facce a cui è localmente adiacente;
- tre sono quelli associati ai tre edge a cui è localmente adiacente;
- uno è il centroide dell'esaedro;

3.2.6 Finalizzazione

Una volta creati tutti i nuovi esaedri che andranno a sostituire lo spazio occupato dall'ormai vecchio livello si procede all'eliminazione di quest'ultimo e poi all'aggiornamento della mesh.

3.3 Collassamento

Dal punto di vista teorico, dato un livello di esaedri, questo algoritmo prevede il collassamento di ogni esaedro facente parte del livello, cioè cancellare il livello e "ricongiungere" le due parti della mesh, come visto nella figura 1.2. Dal punto di vista pratico quello che si andrà a fare è, dato un livello di esaedri, andare a riconoscere gli esaedri adiacenti al livello (sopra e sotto), eliminarli insieme al livello e creare due livelli di esaedri in sostituzione ai tre eliminati.

3.3.1 Preparazione ed esecuzione

Al contrario dell'algoritmo precedente, sarà necessario eseguire il collassamento degli esaedri in un certo ordine. Come nell'algoritmo precedente si va a controllare, per ogni esaedro facente parte del livello, per quante direzioni è attraversato dal livello e, a priori, si dividono gli esaedri sulla base di questa analisi in tre liste differenti (vedi figura 3.6).

Questa gestione è necessaria in quanto il collassamento deve avvenire prima per gli esaedri attraversati dal livello in tre direzioni, poi per quelli attraversati in due direzioni e successivamente per quelli attraversati in una sola direzione, poiché più sono le direzioni per cui il livello passa più esaedri dovremmo far collassare contemporaneamente (gli esaedri appartenenti alla catena e adiacenti per la direzione del collasso vengono gestiti tutti insieme), come mostrato nella figura 3.6.
Il controllo viene effettuato in maniera identica all'algoritmo precedente: si prende un vertice qualsiasi e si controlla quanti edge localmente adiacenti fanno parte della catena, in base a questo parametro si distribuiscono nelle liste di priorità gli esaedri. Una volta tutti gli esaedri sono stati inseriti nella lista di appartenenza si ciclano le tre liste in base alla priorità e si procede al collassamento.

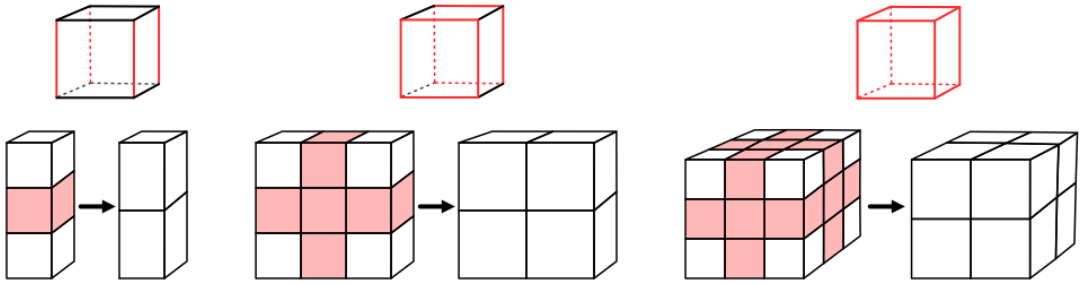

Figura 3.6: *Le colonne rappresentano in ordine: collassamento semplice, doppio e triplo. In alto troviamo gli edge facenti parte della catena per ogni categoria, mentre in basso lo schema del collassamento (in rosso gli esaedri del livello).*

3.3.2 Collassamento semplice

Dato un esaedro che deve subire un collassamento semplice, saranno necessari quattro nuovi vertici, che andranno a formare la base in comune dei due nuovi esaedri. Questi vertici aggiuntivi vengono creati al centro degli edge dell'esaedro che fanno parte della catena (la posizione è ancora una volta arbitraria), in modo simile all'algoritmo precedente.
Una volta creati i nuovi vertici, si devono identificare gli esaedri adiacenti esterni al livello, questi serviranno come riferimento per la creazione dei nuovi esaedri. Per trovarli si vanno a cercare i due esaedri (non è detto che ci siano) adiacenti all'esaedro attraverso le due facce che non hanno edge facenti parte della catena. Per ognuno di questi esaedri esterni al livello si prende la faccia opposta a quella condivisa con l'esaedro che fa parte del livello e, per ognuno dei suoi vertici, si crea una corrispondenza con i vertici aggiuntivi che sono stati creati in precedenza.
Per trovare queste corrispondenze, che ci permetteranno di trovare i vertici per

completare l'esaedro nel giusto ordine, da ogni vertice della faccia troviamo il vertice localmente adiacente che appartiene alla faccia in comune con l'esaedro di partenza. Poi da quel vertice si va a trovare l'unico edge che fa parte della catena di edge.
Fatte queste associazioni abbiamo tutti i vertici e le informazioni per metterli nel giusto ordine, quindi possiamo creare i due esaedri.

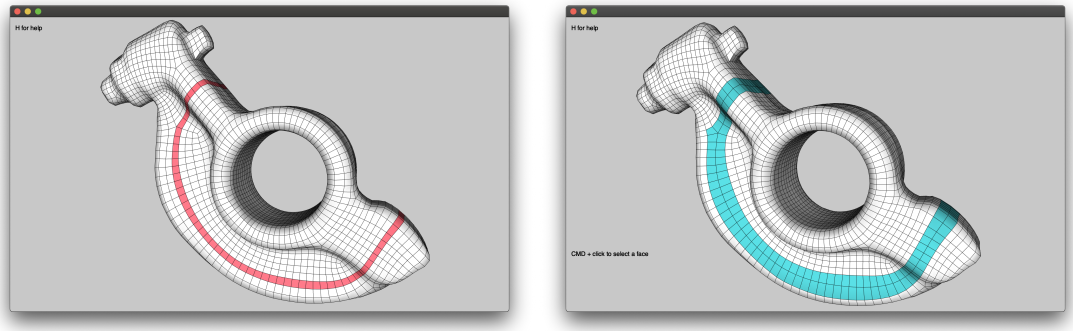

Figura 3.7: *A sinistra la selezione del livello, a destra il risultato del collassamento del livello (dove possiamo notare un collassamento doppio)*

3.3.3 Collassamento doppio

Un esaedro che deve subire un collassamento doppio è attraversato dal livello per due direzioni, per la creazione dei nuovi esardri saranno necessari dai 6 ai 10 nuovi vertici. Quando ci troviamo in questo caso abbiamo 8 edge facenti parte della catena e 4 esterni alla catena, per ognuno di questi ultimi andiamo a identificare gli esaedri adiacenti:

- gli esaedri adiacenti che fanno parte del livello vengono gestiti in blocco insieme all'attuale

- gli esaedri non facenti parte del livello servono come riferimento e base per i nuovi esaedri

Una volta identificati gli esaedri adiacenti facenti parte del livello si può procedere nella creazione dei vertici aggiuntivi:

- un vertice sul centroide (posizione arbitraria) di faccia con tutti e 4 gli edge facenti parte della catena, per un totale di 2 vertici

- un vertice al centro di ogni edge facente parte della catena appartenente appartenente agli esaedri adiacenti appartenenti al livello identificati in precedenza, ma non appartenenti all'esaedro di partenza, per un totale di vertici che varia da 4 a 8 in base al numero di esaedri adiacenti appartenenti al livello identificati.

Per ogni esaedro identificato non facente parte della catena si identificano i due esaedri adiacenti che fanno parte del livello e, a partire dalle due facce al centro della quale abbiamo creato due dei nuovi vertici, si iniziano a identificare i vertici e le corrispondenze per creare i nuovi esaedri.

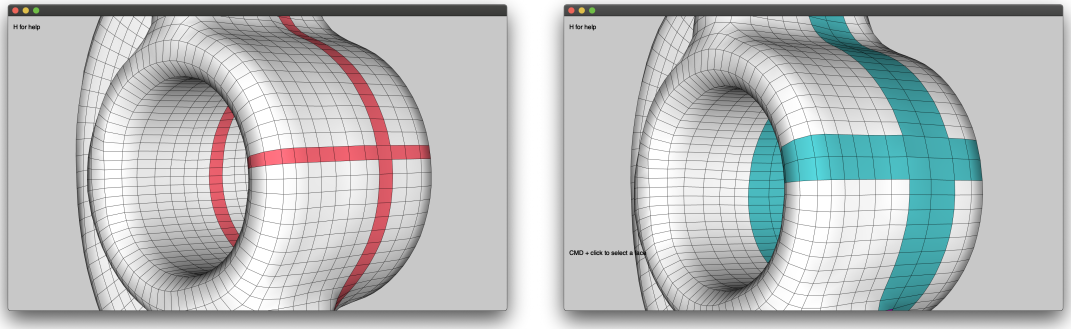

Figura 3.8: *A sinistra la selezione del livello, a destra il risultato del collassamento del livello (dove possiamo notare un collassamento doppio)*

Quindi, per ogni faccia con tutti gli edge appartenenti alla catena (a cui ricordiamo corrisponde un vertice), si identificano le due facce adiacenti che appartengono ai due esaedri identificati immediatamente prima (non la faccia comune ai due esaedri). Per ognuna di queste due facce riconosciamo l'esaedro appartenente alla catena che non appartiene anche all'esaedro di partenza, in quanto abbiamo creato un vertice associato a quell'edge.
Dopo di che andiamo ad identificare la faccia, adiacente ad entrambe queste due facce, appartenente all'esaedro fuori dal livello, in quanto l'unico vertice di questa faccia che non appartiene anche alle altre due facce sarà il quarto vertice, che ci permetterà, insieme agli altri quattro trovati in modo speculare, di creare il nuovo esaedro.

3.3.4 Singolarità

Nel caso di collasso semplice può capitare, in presenza di singolarità della mesh, che ci siano esaedri che condividono con un esaedro del livello un edge o un vertice, ma non sono adiacenti a nessun livello del vertice. Quando si va a collassare bisogna andare ad aggiornare anche questi esaedri andando a creare un nuovo esaedro corrispondente in cui andiamo a sostituire quell'edge/vertice con i nuovi corrispondenti creati nella fase di collasso.

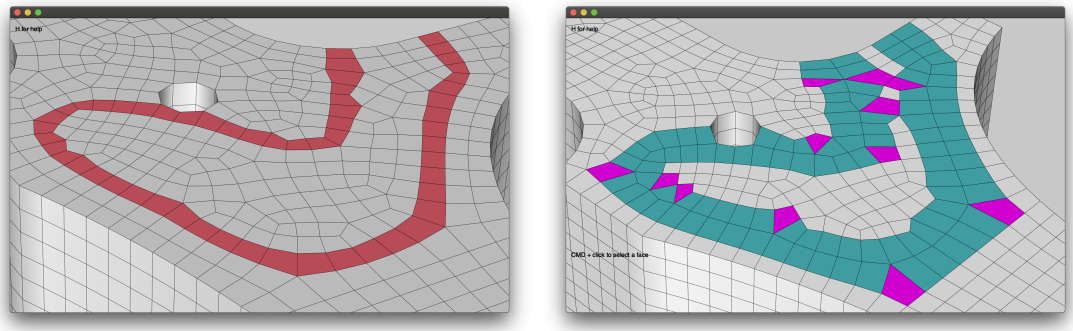

Figura 3.9: *In rosso evidenziato il livello prima del collasso, in celeste dopo. In magenta la gestione degli esaedri che aveano un edge in comune con un esaedro del livello*

3.3.5 Finalizzazione

Ogni esaedro originale coinvolto nelle operazioni di collassamento, quindi gli esaedri facenti parte del livello e i vari esaedri adiacenti coinvolti, vengono eliminati, quindi si può procedere all'aggiornamento della mesh.

Capitolo 4

Risultati

In questo capitolo analizziamo alcuni esempi di utilizzo delle funzionalità implementate con i relativi dati.

4.1 Risultati della selezione di un livello

Di seguito alcuni esempi di selezione del livello di esaedri, in mesh volumetriche più o meno complesse, con i livelli evidenziati in rosso.
Oltre ad essere funzionale per i due metodi di inserimento e collassamento, l'identificazione di un livello di esaedri può essere utile per varie analisi, un esempio può essere la distribuzione delle forze in un pezzo meccanico

Figura	Tempo	#Edge catena	#Esaedri livello
Sinistra	0.149239s	13946	12977
Destra	0.093074s	10037	9192

Tabella 4.1: *Statistiche esempi di selezione del livello*

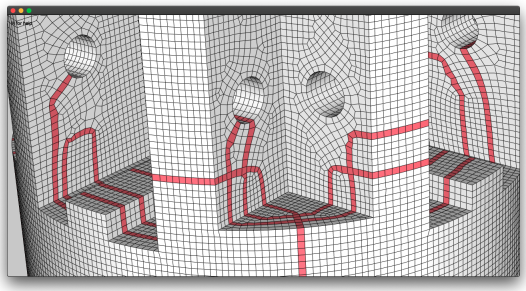

Figura 4.1: *Componente meccanico*

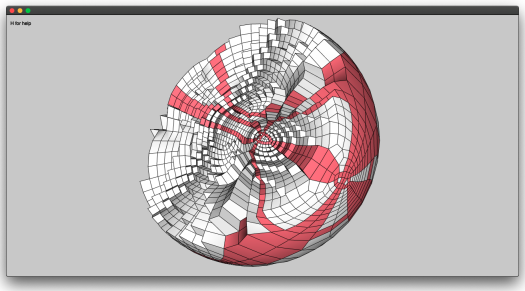

Figura 4.2: *Simulazione di carico*

4.2 Risultati dell'inserimento di un livello

Di seguito esempi di inserimento del livello, in blu/verde gli esaedri creati nello split semplice.
Questa operazione può essere utilizzata per il refinement e rendere più definita la mesh volumetrica.
In media, ponendo come X il numero di esadri nel livello, andiamo a cancellare X esaedri e a creare circa 2X esaedri.

Figura	Tempo	#Esaedri livello	#Esaedri creati
Sinistra	0.038315s	330	672
Destra	1.128874s	12743	25486

Tabella 4.2: *Statistiche esempi di inserimento del livello*

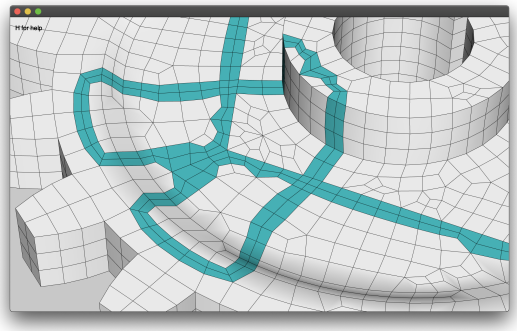

Figura 4.3: *Ingranaggio*

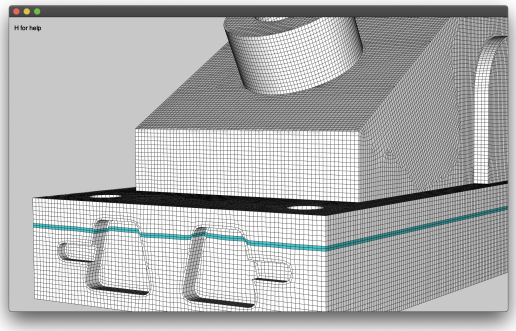

Figura 4.4: *Componente meccanico*

4.3 Risultati del collassamento di un livello

A seguire alcuni risultati del collassamento del livello, evidenziati i nuovi esaedri.
In maniera opposta all'operazione precedente, è molto utile per semplificare una mesh volumetrica.
È l'operazione più pesante tra le tre, in quanto, ponendo come X il numero di esaedri nel livello, vengono eliminati circa 3X esaedri e creati circa 2X esaedri.
Un ottimizzazione potrebbe essere quella di trovare un nuovo approccio in modo da andare a ridurre queste operazioni di eliminazione e di creazione di esaedri, cercando di raggiungere 2X eliminazioni e 1X creazioni.

Figura	Tempo	#Esaedri livello	#Esaedri creati
Sinistra	0.124095s	440	904
Destra	0.621560s	1861	3722

Tabella 4.3: *Statistiche esempi di collassamento del livello*

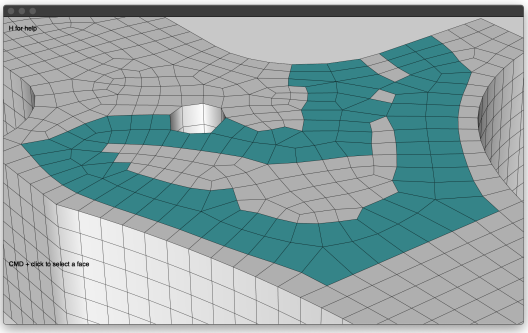

Figura 4.5: *Componente meccanico*

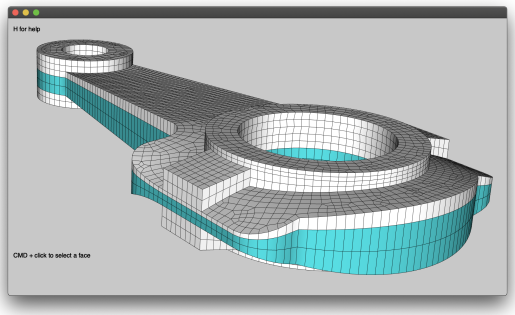

Figura 4.6: *Componente meccanico*

4.4 Considerazioni complessive

Come possiamo notare il tempo varia in base al numero di esaedri nel livello, ma ciò che fa veramente la differenza è il numero di operazioni di cancellazione e creazione degli esaedri.
Se andiamo a vedere nello specifico l'operazione più veloce è, senza ombra di dubbio, la selezione del livello, in quanto non prevede nessun operazione di creazione o eliminazione di poliedri, mentre le altre due operazioni prevedono creazione e cancellazione.
In particolare l'inserimento del livello prevede un numero di operazioni di creazione e cancellazione di esaedri pari a tre volte la grandezza del livello, mentre il collassamento addirittura cinque volte la grandezza del livello.

Capitolo 5

Conclusioni

In questa tesi è stata presentata l'implementazione della selezione, dell'inserimento e del collassamento di un livello di esaedri in una mesh volumetrica, operazioni che permettono di andare a raffinare o semplificare un modello volumetrico e che possono tornare molto utili in caso di manipolazione di una mesh volumetrica.

Come abbiamo potuto notare dai risultati le prestazioni dipendono principalmente dal numero di esaedri che si va eliminare e creare, se nel caso dell'inserimento di un livello questa operazione è stata ottimizzata il più possibile. Per quanto riguarda il collassamento di un livello sicuramente si potrà in futuro andare a trovare un algoritmo che permetta di effettuare questa operazione in maniera consistente andando a minimizzare il numero di operazioni di inserimento e cancellazione.

Altro aspetto importante del lavoro è stato accertarsi che la qualità degli esaedri rimanesse buona, che le normali delle facce fossero sempre rivolte verso l'esterno e che il segno del volume rimanesse consistente rispetto con esaedri originali, fattore dipendente dal winding delle facce. Per raggiungere questo obiettivo è stato fondamentale tenere riferimenti all'interno della mesh e creare corrispondenze in base alla connettività che legava i vari elementi della mesh, in questo modo si è riusciti ad avere nel giusto ordine i vertici ogni qual volta si dovesse andare a creare un nuovo esaedro. Andare a creare un nuovo esaedro utilizzando una lista di vertici ordinata nel modo sbagliato può generare esaedri dal volume negativo, con le normali delle facce rivolte verso l'interno o peggio esaedri degeneri.

Per quanto riguarda la copertura di più casi possibili nel funzionamento, nel caso della selezione e dell'inserimento del livello i risultati sono stati soddisfacenti, nel caso del collassamento del livello c'è ancora del lavoro da fare.

Allo stato attuale il collassamento del livello non prevede ancora la gestione di alcuni corner case, quali split triplo (in via di sviluppo) e particolari casi in cui il livello "curvando" si ritrova ad essere adiacente a se stesso (figura 5.1).

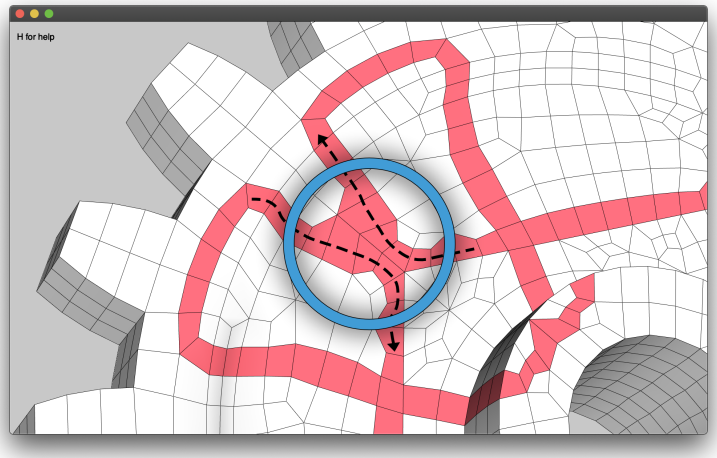

Figura 5.1: *Il livello selezionato è evidenziato in rosso, come possiamo notare il livello risulta adiacente a se stesso*

Il corner case in figura 5.1 risulta essere un grosso problema, in quanto il livello potrebbe assumere una conformazione a "serpentina" e andare ad affiancare se stesso diverse volte e non solo una come in figura, per questo motivo c'è bisogno di ulteriore tempo per lo sviluppo.
Per fortuna questi due corner case sono abbastanza rari, quindi nella maggior parte delle occasioni sarà possibile utilizzare l'operazione di collassamento del livello senza problemi.

Allo stato attuale dei lavori ci si può ritenere soddisfatti dei risultati ottenuti fin ora, ma, come spiegato in precedenza, c'è ancora del lavoro da fare per rendere ottimali queste operazioni, in particolar modo l'operazione di collassamento.

Capitolo 6

Sviluppi futuri

Le operazioni implementate potranno avere numerosi sviluppi futuri, tra cui l'inclusione definitiva come parte integrante della libreria Cinolib.
L'obiettivo finale dello sviluppo di queste funzionalità è l'integrazione nella libreria in modo da renderle chiamate di libreria, in questa maniera chi andrà a gestire hexmesh con l'ausilio di Cinolib non dovrà implementare da se queste funzioni.

Lavoro preliminare per far diventare queste funzionalità chiamate di libreria è sicuramente una fase di testing su grossi dataset di modelli, questo permette di avere più dati dal punto di vista delle tempistiche e della memoria occupata, ma anche di scovare eventuali bug, ove presenti.

Altri sviluppi saranno quindi un ottimizzazione del codice per migliorare le performance, magari trovando approcci che comportino tempistiche migliori o meno consumo di memoria durante l'esecuzione (che comunque va a influire sulle tempistiche in quanto minore lettura e scrittura in memoria in genera corrisponde anche a migliori tempistiche), un esempio potrebbe essere trovare un algoritmo che permetta in modo consistente di ridurre di un terzo gli esaedri eliminati e della metà quelli creati.

Una volta le funzionalità descritte saranno incluse nella libreria si potrà procedere allo sviluppo di altri sheet operator da includere, un esempio può essere creare un tunnel di esaedri a partire da una faccia.

Bibliografia

[1] Volume Mesh Wikipedia.

[2] G. Cherchi, P. Alliez, R. Scateni, M. Lyon, D. Bommes. Selective Padding for Polycube-Based Hexahedral Meshing. 2019.

[3] Jinming Chen, Shuming Gao, Rui Wang, Haiyan Wu. An approach to achieving optimized complex sheet inflation under constraints. 2016.

[4] Marco Livesu. CinoLib: A generic programming header only C++ library for processing polygonal and polyhedral meshes. 2019.

[5] Rui Wang, Shuming Gao, Zhihao Zheng, Jinming Chen. Hex mesh topological improvement based on frame field and sheet adjustment. 2017.

Elenco delle figure

1.1 Inserimento di un livello dal punto di vista teorico 2
1.2 Collassamento di un livello dal punto di vista teorico 2
1.3 Schema della pipeline introdotta in [3] 3

2.1 *Confronto tra l'interno di una mesh di superficie (sinistra) e di una mesh volumetrica (destra).* 5
2.2 Confronto tra le sezioni di una tetmesh (sinistra) e una hexmesh (destra) 7
2.3 Simulazione di carico su un aereo di linea 8
2.4 *Struttura della libreria Cinolib.* 9

3.1 Esempio di catena di edge in una mesh volumetrica 12
3.2 Sezione di una mesh volumetrica da cui è stato selezionato un livello di esaedri 12
3.3 Le colonne rappresentano in ordine: split semplice, doppio e triplo. In alto troviamo gli edge facenti parte della catena per ogni categoria, mentre in basso lo split scheme. 14
3.4 In rosso il livello prima dello split, in blu dopo 14
3.5 In rosso il livello prima dello split, in blu dopo 15
3.6 Le colonne rappresentano in ordine: collassamento semplice, doppio e triplo. In alto troviamo gli edge facenti parte della catena per ogni categoria, mentre in basso lo schema del collassamento (in rosso gli esaedri del livello). 17
3.7 A sinistra la selezione del livello, a destra il risultato del collassamento del livello (dove possiamo notare un collassamento doppio) . 18
3.8 A sinistra la selezione del livello, a destra il risultato del collassamento del livello (dove possiamo notare un collassamento doppio) . 19
3.9 In rosso evidenziato il livello prima del collasso, in celeste dopo. In magenta la gestione degli esaedri che aveano un edge in comune con un esaedro del livello 20

4.1 Componente meccanico 21

4.2 Simulazione di carico 21
4.3 Ingranaggio 22
4.4 Componente meccanico 22
4.5 Componente meccanico 23
4.6 Componente meccanico 23

5.1 Il livello selezionato è evidenziato in rosso, come possiamo notare il livello risulta adiacente a se stesso 26

Elenco delle tabelle

4.1 Statistiche esempi di selezione del livello 21
4.2 Statistiche esempi di inserimento del livello 22
4.3 Statistiche esempi di collassamento del livello 23

Ringraziamenti