Alma Mater Studiorum · Università di Bologna

SCUOLA DI INGEGNERIA E ARCHITETTURA

Corso di Laurea Magistrale in Ingegneria Informatica

TESI DI LAUREA in
Fondamenti di Computer Graphics M

IMPLEMENTAZIONE DI UN ALGORITMO SDF PER LA DETERMINAZIONE DELLO SPESSORE DI COMPONENTI MECCANICI

Candidato:
Lorenzo Guidi

Relatore:
Chiar.mo Prof.
Giulio Casciola

Co-relatori:
Ing. Roberto Raffaeli
Giacomo Ferrari

Anno Accademico 2013-2014
III Sessione

Introduzione

Il processo di stampaggio di materie plastiche é eseguito attraverso l'utilizzo di una pressa in grado di produrre oggetti con una forma definita partendo da materiali plastici. Tale trasformazione avviene attraverso un processo termico denominato plastificazione che rende fluido il materiale e prosegue con un'iniezione sotto alta pressione in una cavitá stampo. Questo processo termina con l'estrazione del prodotto dopo che ha ultimato il suo raffreddamento o raggiunto delle caratteristiche tali da poter continuare il suo raffreddamento in aria senza compromettere le sue funzionalitá meccaniche, fisiche ed estetiche. Le caratteristiche strutturali delle materie plastiche ne permettono la modellazione per azione del calore e della pressione. Queste materie sono impiegate in ogni settore per le loro caratteristiche di leggerezza, facile lavorabilitá e basso costo.
Per stabilire il costo di produzione di un nuovo prodotto, l'azienda interessata a produrlo deve analizzare le molteplici voci che lo compongono per giungere alla stesura di un preventivo. Gli aspetti principali che concorrono alla definizione del costo di produzione di un prodotto sono:

1. analisi delle caratteristiche geometriche dell'oggetto;

2. stima del tempo di lavorazione delle parti che compongono la forma dell'oggetto all'interno dello stampo: in funzione dell'analisi svolta alla fase precedente viene ricavato il tempo necessario per asportare il volume di materiale che serve a creare la cavitá che compone la forma dell'oggetto;

3. stima del tempo di progettazione: esso dipende dalla forma finale del manufatto;

4. valutazione degli accessori: utilizzando anche in questa fase l'analisi delle caratteristiche, vengono valutati il numero di estrattori necessari, la loro tipologia, il tipo di raffreddamento ed altri elementi riguardanti la produzione;

5. valutazione delle lavorazioni secondarie: tali lavorazioni sono necessarie per raggiungere il livello qualitativo posto come obiettivo;

6. costo prove e consegna: a seconda della tipologia di prodotto, vengono definite un determinato numero di prove stampo che permettono di stabilire i corretti parametri di lavorazione della pressa per il ciclo di stampaggio, assicurare che ogni componente nello stampo funzioni in maniera adeguata ed infine verificare la qualitá del prodotto finale ottenuto.

Come si puó notare ci sono due fasi la cui precisione dei risultati dipende dalla prima fase, quella relativa all'analisi delle caratteristiche geometriche dell'oggetto che si vuole produrre e di cui si vuole fare una stima dei costi di produzione.
Il lavoro effettuato in questa tesi si occupa proprio della fase che analizza le caratteristiche geometriche principali dell'oggetto, come area proiettata, spessore medio, massimo e minimo, volume e dimensioni principali.
Per poter stimare al meglio il costo di produzione di stampi che permettono di realizzare l'oggetto con la forma voluta e quindi di stimare il costo di produzione degli oggetti stessi si é sentita la necessitá di implementare in un sistema di gestione ed analisi di modelli 3D una funzione che consenta di misurare lo spessore di un qualsiasi oggetto. In particolare il focus é stato posto su pezzi industriali in plastica i quali, per loro costruzione, non sono pieni bensí cavi rendendo lo spessore localmente molto variabile. Questa repentina variabilitá dello spessore fa sí che la sua stima richieda una misurazione di grande precisione.

L'obiettivo di questa tesi é quindi inserire nel programma di gestione ed analisi di modelli 3D una funzione per il calcolo dello spessore di componenti meccanici che non fornisca solamente una stima approssimativa delle dimensioni dell'oggetto in alcune parti, ma misurazioni a distanze ravvicinate, molto precise e locali. Tra le possibili funzioni esistenti in letteratura si é scelto di implementare la Shape-Diameter Function. Questo algoritmo consiste nel lanciare un cono di raggi per ogni punto in cui si vuole analizzare lo spessore cercando intersezioni con la mesh e misurando lo spessore di quel punto come la media dei valori distanza restituiti dai raggi stessi.
Lo studio della Shape-Diameter Function é stato suddiviso in 3 fasi principali:

1. analisi della funzione SDF proposta in MeshLab;

2. implementazione funzione SDF: in questa fase é stata implementata una particolare versione della funzione SDF, rimodellata innanzitutto sulla struttura dell'applicativo in cui é stata inserita, utilizzando le strutture dati ed i costrutti piú efficienti sia del programma che del liguaggio utilizzato. In seguito sono state apportate modifiche alla struttura dell'algoritmo stesso che é stato in parte ripensato. La riprogettazione é stata eseguita per renderne l'implementazione piú conforme all'ambiente applicativo in cui viene utilizzata e sulla base delle caratteristiche piú richieste nel suo utilizzo, cioé la caratterizzazione delle proprietá geometriche di modelli di componenti meccanici, in particolare plastici;

3. test e validazione dei risultati: in tale fase sono stati effettuati test sull'efficienza dell'algoritmo utilizzando tre principali indicatori:

   (a) tempo di esecuzione;

   (b) percentuale di vertici dai cui raggi é stata trovata almeno un'intersezione con la mesh rispetto al totale di vertici di cui é composta la mesh stessa;

   (c) misurazione delle caratteristiche geometriche: consiste nel calcolo dello spessore minimo, massimo, medio e della frequenza di inter-

valli di spessore sulla base di una parametrizzazione dell'intervallo tra il minimo ed il massimo valore calcolato.

Questi test sono stati effettuati su un insieme di modelli di riferimento, preesistenti nell'applicativo di riferimento.
Le ultime due fasi sono state eseguite in parallelo al fine di tenere sotto controllo l'effetto di ogni singola modifica al codice sugli indicatori di efficienza utilizzati, in modo da implementare soltanto le variazioni che portassero ad un miglioramento generale dell'algoritmo, cioé che permettessero di:

- abbassare il tempo di esecuzione mantenendo invariata la quantitá e qualitá di risultati ottenuti;
- aumentare la precisione dell'algoritmo mantenendo il tempo di esecuzione (pressoché) invariato;
- migliorare l'algoritmo per quanto riguarda il tempo di esecuzione o la sua precisione senza avere importanti effetti negativi sull'altro indicatore.

Nei capitoli successivi vengono descritte le varie fasi eseguite durante il lavoro:

- nel capitolo 1 vengono accennate le funzioni di calcolo dello spessore di un modello tridimensionale proposte fino ad oggi e viene analizzato brevemente lo stato dell'arte riguardo applicativi sul mercato che permettano di studiare le caratteristiche geometriche degli oggetti in analisi;
- nel capitolo 2 viene analizzata nel dettaglio la funzione SDF con particolare riferimento all'implementazione proposta da MeshLab;
- nel capitolo 3 vengono presentate le principali modifiche apportate alla struttura della funzione e dell'algoritmo al fine di migliorarlo per lo specifico ambiente di utilizzo basandosi sugli indicatori prima descritti;

- nel capitolo 4 vengono descritti i test eseguiti sulla versione finale del codice presentando risultati che confermano il miglioramento delle prestazioni.

Indice

Capitolo 1

Stato dell'arte

La partizione di modelli 3D e l'estrazione del loro scheletro strutturale sono due algoritmi molto importanti nella computer graphics. I primi algoritmi che hanno permesso l'estrazione di queste caratteristiche sono basati su attributi della superficie nei confini dell'oggetto come la curvatura, la planaritá, la direzione della normale, le distanze geodetiche ed altri. In alternativa furono proposti algoritmi basati sulla topologia, sull'analisi spettrale o su attributi globali come la distanza geodetica media (AGD).
Questi algoritmi peró soffrono di un problema importante, ovvero sono troppo dipendenti dalla posizione nello spazio 3D e topologia dell'oggetto su cui sono eseguiti, quindi cambiare la posa di un qualsiasi oggetto potrebbe causare come risultato una partizione completamente differente.
Ció ha portato allo studio di nuovi algoritmi che permettessero di superare queste problematiche ed ottenere risultati piú soddisfacenti.
Furono proposti algoritmi di differente impostazione, un esempio é un algoritmo [1] che estrae dal modello punti caratteristici utilizzati per calcolare una funzione di mapping invariante che rivela le parti importanti dell'oggetto e permette di estrarre lo scheletro dello stesso per poi utilizzarlo [2] per calcolare una segmentazione gerarchica della mesh stessa. Oppure un algoritmo [3] che utilizza un ridimensionamento multidimensionale per la creazione di una rappresentazione della mesh indipendente dalla posa poi usata per trovare

punti caratteristici ed operare una decomposizione gerarchica. O ancora un algoritmo [4] che utilizza una procedura iterativa operando simultaneamente una decomposizione approssimata convessa dell'oggetto [5] ed estraendo uno scheletro usando l'asse principale di ogni parte.
I due metodi principali peró si basano sul MAT (medial axis transform) o sulla voxelizzazione. I primi, come [6], hanno alta precisione ma soffrono di grande difficoltá nell'estrazione ed utilizzo della struttura MAT, i secondi, come [7] e [8], creando una discretizzazione sono soggetti ad errori e dipendono fortemente dalla grandezza della griglia che viene creata.
In seguito vennero proposti algoritmi basati sul volume degli oggetti invece che sugli attributi della superficie, permettendo di superare il problema della dipendenza dalla posa. Uno di essi [9] crea una bolla sferica in ciascun vertice misurando il modo in cui essa interseca la superficie, oppure un altro [10] utilizza una decomposizione approssimata convessa partizionando ogni modello in una gerarchia in modo da preservare la convessitá e compattezza delle parti degli oggetti. Tra questo tipo di algoritmi spiccava l'algoritmo Shape-Diameter Function (SDF) [11] che esamina il diametro del modello nella vicinanza di ogni punto sulla sua superficie. In particolare questo algoritmo calcola il volume della mesh mediante la ricerca di intersezioni raggio-mesh, operazione molto conosciuta nel ray-tracing. Molto brevemente tale funzione consiste nel lanciare, per ogni vertice, un cono di raggi verso l'interno della mesh, calcolando cosí lo spessore della mesh in ogni singolo vertice.
Sul World Wide Web é possibile trovare due principali implementazioni della funzione SDF, la prima é possibile trovarla e scaricarla direttamente dal sito web dell'inventore della funzione stessa, ovvero Lior Shapira [12]. La seconda implementazione [13] é inserita all'interno di MeshLab, programma gratuito ed open-source italiano che permette di creare, modificare ed analizzare modelli tridimensionali composti da mesh.

Per quanto riguarda gli strumenti commerciali che forniscono funzionalitá di misurazione delle caratteristiche geometriche di un modello 3d é possibile

citare Materialise Magics Rp 16. Questo programma consente di effettuare operazioni basate su modelli geometrici con estensione *.stl. Questo formato rappresenta un componente tridimensionale dotato di spessori ad ognuno dei quali vengono associate due superfici: una interna ed una esterna. All'interno del programma vi é il comando 'Spessore delle pareti' che esegue un'analisi sullo spessore del manufatto localizzando eventuali pareti dotate di uno spessore troppo sottile che potrebbe creare problemi sia nel processo di stampaggio, sia nel processo di realizzazione dello stampo. I risultati ottenuti vengono mostrati mediante una mappa di colori che permette di identificare i vari valori dello spessore, che puó essere misurato manualmente, anche in determinate zone del pezzo. Per ottenere dei risultati occorre definire uno spessore massimo sia per l'analisi, sia per la mappa dei colori.

Capitolo 2

Algoritmo Shape-Diameter Function

In questo capitolo viene inizialmente analizzata la Shape-Diameter Function nella sua caratterizzazione generale, poi viene descritta passo passo l'implementazione presente in MeshLab inserita nel programma stesso come filtro. Essendo MeshLab un programma open-source é stato possibile accedere alla documentazione relativa al codice utilizzandolo come base per l'implementazione eseguita in questa tesi.

2.1 Descrizione generale

La Shape Diameter Function permette di calcolare lo spessore della mesh calcolando lo spessore di ogni singolo vertice di cui essa é composta.
Per ottenere lo spessore di un singolo vertice l'algoritmo esegue una serie di passi, infatti:

- crea un cono di raggi con origine nel punto corrispondente al vertice attualmente in analisi e sviluppato in direzione opposta alla normale del vertice stesso, in modo da puntare all'interno della mesh e non all'esterno. Il motivo é intuitivo: lo spessore é la misurazione della grandezza interna di un oggetto quindi é necessario analizzare l'interno

della mesh non l'esterno. Questo ipotetico cono viene creato generando un certo numero di raggi che hanno tutti origine nel vertice e direzione ognuno differente ma sempre interna al cono;

- per ogni raggio del cono, esegue una ricerca del punto di intersezione con la mesh piú vicino nella direzione del raggio stesso. La tecnica utilizzata é il ray casting: si fa il cast del raggio nella sua specifica direzione e si controlla se esso tocca la mesh. Se viene trovato almeno un punto di intersezione, si controlla la normale in quel punto: se essa punta nella stessa direzione della normale del vertice origine del raggio si ignora l'intersezione. L'uguaglianza tra le normali é definita come una differenza tra i rispettivi angoli minore di 90 gradi. Questa operazione viene eseguita per eliminare intersezioni con la parte esterna della mesh cioé false intersezioni. Se il punto passa il controllo allora viene inserito in un insieme di risultati trovati per quel vertice;

- avendo calcolato i valori di distanza di tutti i raggi del cono di uno stesso vertice, il valore SDF di quel vertice é definito come la media pesata delle lunghezze dei raggi (ognuna intesa come distanza tra origine del raggio e punto d'intersezione trovato da quel raggio) che rimangono entro una deviazione standard dal valore medio di tutte le lunghezze. Il peso che ogni raggio ha all'interno della media é l'inverso dell'angolo tra il raggio ed il centro del cono, perché i raggi con angoli piú larghi sono piú frequenti quindi hanno meno peso.

L'algoritmo SDF ripete questa esecuzione per tutti i vertici della mesh restituendo come risultato il valore SDF calcolato per ogni singolo vertice. I singoli valori rappresentano ognuno lo spessore di uno specifico vertice e nel loro insieme lo spessore della mesh.
Questa definizione del SDF é invariante a traformazioni a corpo rigido della mesh ed é invariante a qualsiasi trasformazione che non altera localmente la forma volumetrica. Tra queste sono incluse le deformazioni articolate dell'oggetto e movimenti basati sul suo scheletro.

2.2 Descrizione versione MeshLab

In questa parte vengono descritte nel dettaglio le principali caratteristiche della funzione SDF implementata in MeshLab da cui é stato preso spunto per lo sviluppo dell'algoritmo di questa tesi.

L'implementazione MeshLab dell'algoritmo SDF segue nella logica esecutiva quanto descritto al paragrafo 2.1, e per quanto riguarda i risultati agisce restituendo l'insieme di risultati ottenuti su ogni raggio di ogni vertice insieme ai valori di distanza massimo e minimo calcolati.

2.2.1 Discretizzazione della mesh

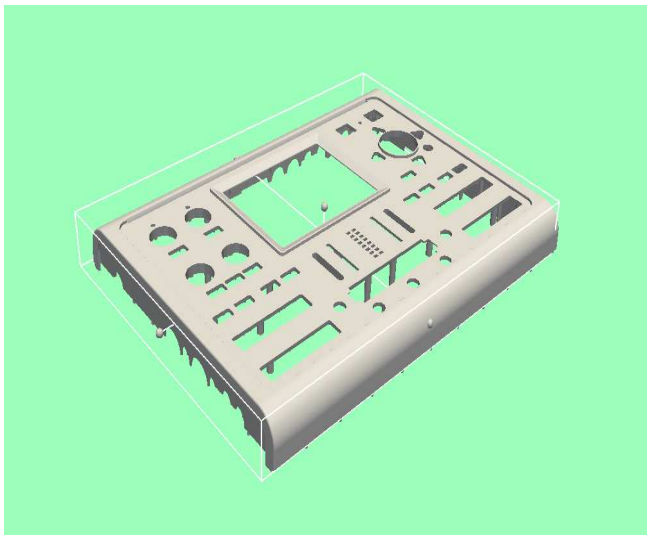

Figura 2.1: Mesh con il proprio bounding box continuo

Il concetto fondamentale alla base di questa implementazione della funzione SDF é la discretizzazione della mesh in voxel (piccoli cubetti) di approssimazione che permettono di costruire strutture di indirizzamento per facilitare e velocizzare la ricerca delle face su cui testare l'intersezione.
La discretizzazione della mesh si compone di due fasi, la prima consiste nel creare un bounding box discreto delle stesse dimensioni del bounding box

continuo della mesh e suddividerlo in voxel cioé cubetti tridimensionali che possono avere grandezza differente in ogni dimensione e che nel loro insieme coprono tutto lo spazio coperto dal bounding box continuo. Nella seconda fase viene effettuata una clusterizzazione delle face di cui é composta la mesh. Questo processo consiste nel calcolare e tenere traccia, per ciascuna face, dei voxel in cui giace il loro bounding box.

2.2.1.1 Discretizzazione del bounding box della mesh

Per discretizzare il bounding box continuo della mesh, l'algoritmo calcola:

- il punto minimo ed il punto massimo del bounding box continuo della mesh negli assi x, y, z (grandezze definite (bbMinX; bbMinY; bbMinZ) e (bbMaxX; bbMaxY; bbMaxZ));
- la dimensione assoluta del bounding box nei 3 assi x, y, z: calcolata, per ogni asse, come la differenza tra il valore massimo ed il valore minimo del bounding box nello specifico asse (grandezze dimX, dimY, dimZ);
- la lunghezza della diagonale del bounding box stesso (maxDist);
- il numero di face di cui é composta la mesh (facesCount);
- un fattore denominato k ($\leq$1) che relaziona il numero di face della mesh all'area del bounding box continuo della stessa: calcolato come $(\frac{facesCount}{dimX*dimY*dimZ})^{\frac{1}{3}}$;

Da queste misurazioni vengono poi calcolati i dati che vanno a comporre la discretizzazione vera e propria:

- il numero intero di parti in cui dividere il bounding box della mesh nei tre assi (un valore intero per asse: sizX, sizY, sizZ): per un singolo asse si calcola come il valore intero del risultato della moltiplicazione tra la grandezza del bounding box continuo in quell'asse ed il fattore k;

- la grandezza, per ogni singolo asse, delle parti in cui viene diviso il bounding box (una per asse: voxelX, voxelY, voxelZ): per un singolo asse é calcolata come il rapporto tra la dimensione del bounding box continuo in quell'asse ed il numero di parti in cui esso é diviso in quell'asse. Ció é intuitivo: avendo ad esempio un'ampiezza in x del bounding box pari a dimX e volendola dividere in sizX parti, ogni parte ha dimensione $voxelX = \frac{dimX}{sizX}$.

Mediante questi dati é possibile costruire un bounding box discreto che ha queste caratteristiche:

- grandezza totale equivalente a quella del bounding box continuo;
- suddivisione di ogni asse secondo i dati calcolati:
  - in x: suddivisione della distanza dimX in sizX parti di dimensione voxelX;
  - in y: suddivisione della distanza dimY in sizY parti di dimensione voxelY;
  - in z: suddivisione della distanza dimZ in sizZ parti di dimensione voxelZ;
- nel loro complesso queste suddivisioni compongono un reticolo tridimensionale che definisce n = sizX*sizY*sizZ voxel ognuno di area = voxelX*voxelY*voxelZ.

Ogni asse é diviso in un numero intero di parti ognuna delle quali referenziabile mediante coordinate intere. Da ció deriva che ogni voxel di cui il bounding box é composto puó essere identificato da una terna intera di coordinate, un valore per ogni asse. Questi valori hanno sempre come minimo il valore 0 mentre il massimo varia per ogni asse ed é corrispondente al numero di parti in cui é diviso quell'asse -1 (esempio: in x varia tra 0 e sizX-1).
Ognuno di questi voxel discretizza l'area del bounding box continuo corrispondente all'area del bounding box discreto equivalente che esso ricopre.

Infatti ponendo come cellX, cellY, cellZ il valore delle 3 coordinate intere che identificano un qualsiasi voxel, esso comprende la seguente area del bounding box continuo:

- in x: tra (cellX*voxelX+bbMinX) e ((cellX+1)*voxelX+bbMinX);
- in y: tra (cellY*voxelY+bbMinY) e ((cellY+1)*voxelY+bbMinY);
- in z: tra (cellZ*voxelZ+bbMinZ) e ((cellZ+1)*voxelZ+bbMinZ).

É intuitivo che ad esempio spostandosi in avanti in x, quindi aumentando cellX, il voxel che viene referenziato comprende il medesimo spazio in y e z e lo spazio successivo in x e cosí via per gli altri assi.

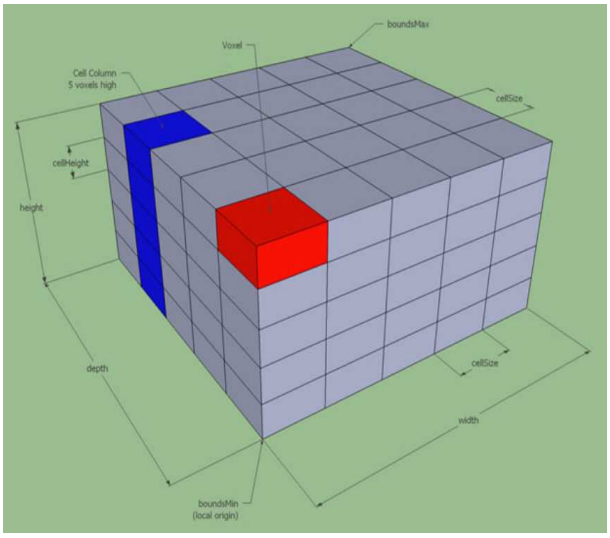

Figura 2.2: Generico bounding box discreto con i dimensionamenti

Esempio:

Dato il bounding box continuo della mesh:

- minimo: (bbMinX = -10,0; bbMinY = 0,0; bbMinZ = -8,0)
- massimo: (bbMaxX = 90,0; bbMaxY = 55,0; bbMaxZ = 15,0)

- dimensione assoluta: (dimX = 100,0; dimY = 55,0; dimZ = 23,0)
- lunghezza della diagonale: maxDist = $\sqrt{(100,0)^2 + (55,0)^2 + (23,0)^2}$ = $\sqrt{10000 + 3025 + 529} = \sqrt{13554} = 116{,}42164$

e dato il numero totale di face della mesh: 2500
viene calcolato:

- k = $(\frac{facesCount}{dimX*dimY*dimZ})^{\frac{1}{3}} = (\frac{2500}{100*55*23})^{\frac{1}{3}} = 0{,}27036$

da cui si ottiene:

- sizX = Int(dimX*k) = Int(100*0,27036) = 27
- sizY = Int(dimY*k) = Int(55*0,27036) = 15
- sizZ = Int(dimZ*k) = Int(23*0,27036) = 6
- voxelX = $\frac{dimX}{sizX} = \frac{100}{27} = 3{,}70370$
- voxelY = $\frac{dimY}{sizY} = \frac{55}{15} = 3{,}66667$
- voxelZ = $\frac{dimZ}{sizZ} = \frac{23}{6} = 3{,}83334$

Da ció si deduce che il bounding box continuo della mesh viene diviso in:

- x: 27 parti di ampiezza 3,70370;
- y: 15 parti di ampiezza 3,66667;
- z: 6 parti di ampiezza 3,83334

Quindi in totale il bounding box discreto é composto da 27*15*6 = 2430 voxel, ognuno corrispondente ad una differente combinazione di valori interi degli assi entro i limiti della grandezza del bounding box discreto, quindi:

- x puó assumere valori da 0 a 26;
- y puó assumere valori da 0 a 14;

- z puó assumere valori da 0 a 5

Da cui deriva che:

- Data la terna di valori (0; 0; 0), il corrispondente voxel copre l'area del bounding box continuo tra:
  (0*3,70370 - 10; 0*3,66667 + 0; 0*3,83334 - 8) = (-10; 0; -8)
  e
  (1*3,70370 - 10; 1*3,66667 + 0; 1*3,83334 - 8) = (-6,29630; 3,66667; -4,16666)

- Il voxel della terna (1; 1; 1) invece copre l'area tra:
  (1*3,70370 - 10; 1*3,66667 + 0; 1*3,83334 - 8) = (-6,29630; 3,66667; -4,16666)
  e
  (2*3,70370 - 10; 2*3,66667 + 0; 2*3,83334 - 8) = (-2,59260; 7,33334; -0,33332)

- Infine la terna (26; 14; 5) identifica un voxel che copre l'area del bounding box continuo tra:
  (26*3,70370 - 10; 14*3,66667 + 0; 5*3,83334 - 8) = (86,29620; 51,33338; 11,16670)
  e
  (27*3,70370 - 10; 15*3,66667 + 0; 6*3,83334 - 8) = (90; 55; 15)

É evidente che ogni possibile configurazione dei valori delle coordinate identifica un differente voxel quindi una diversa area del corrispettivo bounding box continuo da esso ricoperta.

2.2.1.2 Clusterizzazione del bounding box delle face

Per rendere piú efficiente e veloce il processo di ricerca delle intersezioni tra raggi e mesh é utile creare una struttura dati che indicizzi ogni voxel e

che permetta di sapere velocemente quali siano le face il cui bounding box giace in un certo voxel.
L'elemento cardine é una struttura dati, il link, composto da:

- face: il riferimento alla struttura dati di una face;
- index: un intero che identifica un singolo voxel in modo univoco

Questa struttura dati consente di indicare che il bounding box della face di cui contiene il riferimento giace nell'area del bounding box continuo della mesh occupata nel discreto dal voxel identificato da index.
Ovviamente il bounding box di una face puó giacere anche in piú di un voxel, in tal caso viene creato un link per ognuno di questi voxel contenente sempre il riferimento alla medesima face ed ognuno il proprio index.
L'index rappresenta una valida alternativa alla terna per l'identificazione dei voxel, infatti esso é un intero univoco voxel per voxel calcolabile mediante una certa funzione a partire dalla specifica terna del voxel. Ponendo quindi come (cellX; cellY; cellZ) la terna che identifica un qualsivoglia voxel, l'indice che permette di identificare univocamente quello stesso voxel si calcola come: index = cellX + ((cellY + cellZ * sizY) * sizX).
Per costruzione della funzione, l'indice cresce (diminuisce) di:

- 1: al crescere (diminuire) di cellX;
- multipli di sizX: al crescere (diminuire) di cellY;
- multipli di sizX*sizY: al crescere (diminuire) di cellZ.

In questo modo i voxel sono numerati in modo crescente per asse x, poi per asse y ed infine per asse z, quindi:

- spostarsi di 1 voxel in avanti (indietro) in x porta l'indice a crescere (diminuire) di 1;
- spostarsi di 1 voxel in avanti (indietro) in y porta l'indice a crescere (diminuire) di un valore pari a sizX;

- spostarsi di 1 voxel in avanti (indietro) in z porta l'indice a crescere (diminuire) di una quantitá pari a sizX*sizY.

Esempio:

Dati sizX = 2, sizY = 4, sizZ = 3 quindi voxel totali = 24

cellZ	cellY	cellY	funzione: cellX + ((cellY + cellZ * sizY) * sizY)	index
0	0	0	0 + ((0 + 0 * 4) * 2)	0
0	0	1	1 + ((0 + 0 * 4) * 2)	1
0	1	0	0 + ((1 + 0 * 4) * 2)	2
0	1	1	1 + ((1 + 0 * 4) * 2)	3
0	2	0	0 + ((2 + 0 * 4) * 2)	4
0	2	1	1 + ((2 + 0 * 4) * 2)	5
0	3	0	0 + ((3 + 0 * 4) * 2)	6
0	3	1	1 + ((3 + 0 * 4) * 2)	7
1	0	0	0 + ((0 + 1 * 4) * 2)	8
1	0	1	1 + ((0 + 1 * 4) * 2)	9
1	1	0	0 + ((1 + 1 * 4) * 2)	10
1	1	1	1 + ((1 + 1 * 4) * 2)	11
1	2	0	0 + ((2 + 1 * 4) * 2)	12
1	2	1	1 + ((2 + 1 * 4) * 2)	13
1	3	0	0 + ((3 + 1 * 4) * 2)	14
1	3	1	1 + ((3 + 1 * 4) * 2)	15
2	0	0	0 + ((0 + 2 * 4) * 2)	16
2	0	1	1 + ((0 + 2 * 4) * 2)	17
2	1	0	0 + ((1 + 2 * 4) * 2)	18
2	1	1	1 + ((1 + 2 * 4) * 2)	19
2	2	0	0 + ((2 + 2 * 4) * 2)	20
2	2	1	1 + ((2 + 2 * 4) * 2)	21
2	3	0	0 + ((3 + 2 * 4) * 2)	22
2	3	1	1 + ((3 + 2 * 4) * 2)	23

Figura 2.3: Esempio calcolo index dei voxel

Per fare il clustering del bounding box delle face di cui é composta la mesh é necessario ovviamente clusterizzare il bounding box di ogni singola face, una alla volta. Tale procedimento é composto dai seguenti passi:

- calcolo del bounding box continuo della face: si ottiene il punto minimo ed il punto massimo del bounding box continuo della face (bbFaceMinX; bbFaceMinY; bbFaceMinZ) e (bbFaceMaxX; bbFaceMaxY; bbFaceMaxZ);

- calcolo dei voxel in cui giace il bounding box continuo: si calcola la terna intera che identifica il voxel che contiene il punto minimo del bounding box continuo della face e la terna che identifica il voxel che contiene il punto massimo. Ecco i passi che vengono eseguiti:
  - si trasla il punto minimo del bounding box continuo della face di un valore pari al valore minimo del bounding box continuo della mesh. In questo modo si trova il punto a cui corrisponderebbe il punto minimo del bounding box continuo della face ponendo che quello della mesh coincida con (0; 0; 0).
    t = (bbFaceMinX - bbMinX; bbFaceMinY - bbMinY; bbFaceMinZ - bbMinZ);
  - il punto traslato viene diviso, in ogni asse, per l'ampiezza dei voxel in quello stesso asse, ottenendo 3 valori in virgola mobile. La terna di valori interi che identificano il voxel si ottengono semplicemente dalla parte intera dei valori appena calcolati.
    BbFaceMinInt = (Int($\frac{tX}{voxelX}$); Int($\frac{tY}{voxelY}$); Int($\frac{tZ}{voxelZ}$));
  - questo procedimento viene ripetuto col punto massimo del bounding box continuo della face.
- per ogni combinazione possibile delle coordinate intere dal valore minimo al valore massimo trovati nei 3 assi (quindi per ogni voxel su cui giace il bounding box discreto della face):
  - creazione di un link con:
    - face = face attuale (sempre la stessa per questi link);
    - index = l'indice che identifica lo specifico voxel, costruito secondo la funzione vista precedentemente.
  - inserimento del link in una lista

Esempio:

Dato il punto minimo del bounding box continuo della mesh:

- minimo: (bbMinX = -10,0; bbMinY = 0,0; bbMinZ = -8,0)

la dimensione dei voxel nei 3 assi:

- voxel = (3,70370; 3,66667; 3,83334)

e il numero di voxel nei 3 assi:

- siz = (27; 15; 6)

Si calcola:

- bounding box continuo della face:
  - minimo: (bbFaceMinX = -10,0; bbFaceMinY = 11,5; bbFaceMinZ = 0,0)
  - massimo: (bbFaceMaxX = -9,0; bbFaceMaxY = 15,0; bbFaceMaxZ = 7,5)
- voxel discreto corrispondente al punto minimo:
  t = (bbFaceMinX - bbMinX; bbFaceMinY - bbMinY; bbFaceMinZ - bbMinZ) = (-10,0 - -10,0; 11,5 - 0,0; 0,0 - -8,0) = (0,0; 11,5; 8,0)
  quindi:
  - bbFaceMinInt = (Int($\frac{tX}{voxelX}$); Int($\frac{tY}{voxelY}$); Int($\frac{tZ}{voxelZ}$)) = (Int($\frac{0,0}{3,70370}$); Int($\frac{11,5}{3,66667}$); Int($\frac{8,0}{3,83334}$)) = (0; 3; 2)
- voxel discreto corrispondente al punto massimo:
  t = (bbFaceMaxX - bbMinX; bbFaceMaxY - bbMinY; bbFaceMaxZ - bbMinZ) = (-9,0 - -10,0; 15,0 - 0,0; 7,5 - -8,0) = (1,0; 15,0; 15,5)
  da cui:
  - bbFaceMaxInt = (Int($\frac{tX}{voxelX}$); Int($\frac{tY}{voxelY}$); Int($\frac{tZ}{voxelZ}$)) = (Int($\frac{1,0}{3,70370}$); Int($\frac{15,0}{3,66667}$); Int($\frac{15,5}{3,83334}$)) = (0; 4; 4)
- creazione dei link: per questa face vengono creati i link con questi indici:

- cellZ = 2, cellY = 3; cellX = 0 → index = 0 + ((3 + 2*15) * 27) = 891
- cellZ = 2, cellY = 4; cellX = 0 → index = 0 + ((4 + 2*15) * 27) = 918
- cellZ = 3, cellY = 3; cellX = 0 → index = 0 + ((3 + 3*15) * 27) = 1296
- cellZ = 3, cellY = 4; cellX = 0 → index = 0 + ((4 + 3*15) * 27) = 1323
- cellZ = 4, cellY = 3; cellX = 0 → index = 0 + ((3 + 4*15) * 27) = 1701
- cellZ = 4, cellY = 4; cellX = 0 → index = 0 + ((4 + 4*15) * 27) = 1728

Dalla struttura dell'algoritmo di clusterizzazione delle face deriva che:

- ogni singola face é presente nella lista di link un numero di volte (almeno una volta ovviamente) equivalente al numero di voxel in cui giace il proprio bounding box;
- qualsiasi indice puó essere presente piú volte, tante quante sono le face il cui bounding box giace in quel voxel. Questo accade perché affinché il bounding box di una face giaccia in un voxel basta che occupi una minima parte dell'area continua che il voxel ricopre, quindi ogni face avente una piccolissima parte del proprio bounding box nell'area occupata dal voxel causa l'inserimento di un link con l'indice relativo a quel voxel;
- qualsiasi indice puó anche non essere presente, nel caso in cui nel voxel che esso identifica non giaccia nessuna parte del bounding box di nessuna face;
- tutti i link col riferimento alla stessa face rappresentano la clusterizzazione del bounding box di quella face, quindi tutti i link nel loro insieme rappresentano la clusterizzazione del bounding box di tutte le face.

Una volta che la lista di link é stata creata, i suoi elementi vengono ordinati per indice crescente per facilitarne l'inserimento nella griglia.

2.2.1.3 Costruzione griglia di indirizzamento spaziale

Creati tutti i link che costituiscono la clusterizzazione di tutte le face della mesh, é utile organizzarli in una struttura adeguata che permetta, dato l'indice identificativo di un certo voxel, di estrarre facilmente le eventuali face il cui bounding box giace in quello stesso voxel. Questo é molto utile in seguito durante la fase di ricerca di intersezioni per reperire rapidamente le face da testare in base alla specifica parte della mesh in cui l'algoritmo si é spostato.
La struttura dati costruita é una semplice griglia composta da tanti elementi quanti sono i voxel in cui é suddiviso il bounding box discreto della mesh. Il primo elemento della griglia corrisponde al voxel di indice 0, il secondo corrisponde al voxel di indice 1 e cosí via fino all'ultimo che corrisponde al voxel di massimo indice possibile.
La modalitá di riempimento delle celle della griglia é la seguente: si prende il primo link e lo si inserisce nella prima cella, se l'indice del link non é uguale a quello della cella si avanza alla cella successiva e vi si inserisce lo stesso link e cosí via finché si giunge alla cella avente lo stesso indice del link. Se invece l'indice del link e della cella corrispondono si avanza nella lista di link passando tutti quelli aventi l'indice attuale fino ad arrivare al link con indice successivo tra quelli della lista (non per forza l'indice esattamente successivo perché ci possono essere voxel vuoti, ma l'indice successivo tra gli indici presenti nella lista). Ottenuto il link di indice successivo si passa alla cella immediatamente successiva che viene riempita con quel link, poi di nuovo si testa l'uguaglianza tra indici e si agisce in modo differente a seconda che gli indici siano uguali o no. E cosí via.
Riassumendo, ogni cella della griglia viene inizializzata con uno ed un solo link secondo questa modalitá:

- celle il cui indice é presente nella lista di link: vi viene inserito il solo

link o il primo link (se piú di uno) della lista ordinata avente quell'indice; si puó dire che quel link diventa rappresentativo di tutti i link aventi lo stesso indice e quindi che la face referenziata da quel link diventa rappresentativa di tutte le face giacenti nel voxel corrispondente a quell'indice;

- celle il cui indice non é presente nella lista di link: vi viene inserito il primo link avente indice successivo a quello della cella attuale tra quelli presenti nella lista o l'ultimo link se si tratta di celle avente indice maggiore dell'indice dell'ultimo link della lista. Quindi in queste celle viene inserito un link contenente il riferimento ad una face il cui bounding box discreto non giace in quel voxel. Tale situazione é inevitabile volendo riempire queste celle che sono quelle corrispondenti a voxel vuoti.

Esempio:
Poniamo che nella cella di indice 891 giacciano la face 0 e la face 1, nelle celle di indice 892 e 893 nessuna face, nella cella di indice 894 la face 2. I link cosí creati sono:

- link(face0, 891)
- link(face1, 891)
- link(face2, 894)

e poniamo che siano giá stati ordinati.
In quegli indici la griglia é quindi cosí composta:

- elemento 891: link(face0, 891)
- elemento 892: link(face2, 894)
- elemento 893: link(face2, 894)
- elemento 894: link(face2, 894)

Come si puó vedere gli elementi 891 e 894 hanno il link di una face il cui bounding box giace in quel voxel, mentre gli elementi 892 e 893 hanno il link di una face che non giace nel voxel.

2.2.2 Ricerca intersezioni

Dopo aver costruito la griglia di indicizzazione spaziale, l'algoritmo deve calcolare lo spessore della mesh in ogni suo vertice mediante ricerca di intersezioni tra i raggi originati dai vertici e le face della mesh.
Lo scheletro delle iterazioni che l'algoritmo compie per cercare le intersezioni é il seguente:

- per ogni vertice della mesh:
  - creazione del raggio principale avente origine in prossimitá del vertice e direzione opposta alla normale del vertice;
  - generazione del cono di raggi: creazione di un ipotetico cono avente origine nel punto corrispondente all'origine del raggio principale, direzione di sviluppo opposta alla normale del vertice e raggio del cerchio di base di ampiezza definita a propri. All'interno del cono viene generato un qualsivoglia numero di raggi, tutti aventi origine corrispondente all'origine del cono ma ognuno una diversa direzione, intorno alla direzione di sviluppo del cono, che consenta che il raggio, durante il lancio, passi all'interno della base del cono;
  - per ogni raggio generato:
    - ricerca di almeno un'intersezione valida tra il raggio e le face della mesh, prima testando la face nel voxel che contiene il punto di partenza del raggio per poi allontanarsi di voxel in voxel nella direzione del raggio. La ricerca puó avere risultato:
      - positivo: se é stata trovata almeno un'intersezione valida;

- negativo: se l'algoritmo é uscito dal bounding box discreto senza trovare intersezioni valide.

* inserimento della distanza nella lista di risultati: la distanza inserita é:
  - un valore ≥ 0: se la ricerca ha trovato almeno un'intersezione valida, il valore é la distanza valida trovata;
  - un valore indefinito: se la ricerca non ha ottenuto alcuna intersezione valida.

Il risultato dell'algoritmo é la lista delle distanze sopra descritta unitamente alle statistiche corrispondenti alle distanze massima e minima.

2.2.2.1 Generazione del cono di raggi

Per la generazione del cono di raggi l'algoritmo SDF permette di definire sia la quantitá di raggi che la grandezza del diametro che deve avere il cerchio che costituisce la base del cono in cui vengono generati. Definiti questi valori la modalitá di generazione dei raggi per un qualsiasi vertice v rimane la medesima per tutti i vertici.
Innanzitutto viene definito un raggio principale avente come direzione la direzione opposta alla normale di v e come origine il punto corrispondente a v piú uno scostamento nella direzione del raggio stesso. Lo scostamento si ottiene moltiplicando ciascuna componente della direzione del raggio per una grandezza pari alla diagonale del bounding box / 1000. Lo scostamento serve per evitare di ottenere intersezioni con una delle face che insiste sul vertice stesso. La direzione viene infine normalizzata.

Esempio:
Sia:

- lunghezza della diagonale del bounding box: maxDist = 116,42164

e:

- v = (-5,411642; 22,834926; 8,94179)
- vNormal = (-0,1; 0,3; -0,5)

Allora:

- rayDir = direzione opposta a vNormal = (0,1; -0,3; 0,5)
- minDist = maxDist/1000 = 0,11642
- rayOrigin = (vX + minDist*rayDirX; vY + minDist*rayDirY; vZ + minDist*rayDirZ) = (-5,411642 + 0,11642*0,1; 22,834926 + 0,11642*-0,3; 8,94179 + 0,11642*0,5) = (-5,4; 22,8; 9,0)

quindi il raggio ha:

- rayOrigin = (-5,4; 22,8; 9,0)
- rayDir (normalizzata) = (0,16903; -0,50709; 0,84515)

La normale di un vertice é un vettore perpendicolare al piano su cui giace il vertice e che ha come origine il vertice stesso. La normale di un vertice di una mesh ha ovviamente origine nel vertice stesso ed ha direzione dipendente dalla composizione delle face che insistono su quel vertice. In particolare essa viene calcolata in modo che vada verso la parte superiore delle face cioé che, pensando di guardare le face da un qualsiasi punto nella direzione della normale, sia visibile la parte esterna delle face non la parte interna.
Lo scopo dell'algoritmo peró é di misurare lo spessore della mesh, quindi la sua dimensione interna in ogni punto. Per fare questo é necessario che i raggi vengano lanciati verso la parte interna della mesh, quindi delle face. A tal fine bisogna impostare come direzione del raggio principale il vettore opposto alla normale del vertice, in modo tale che non punti verso la parte esterna delle face che insistono sul vertice quindi della mesh ma verso il loro interno permettendoci di misurare la distanza tra vertici e mesh nei punti interni della stessa.
Il raggio principale creato rappresenta la base per i raggi successivi di v,

infatti intorno ad esso viene creato un ipotetico cono che ha come origine l'origine del raggio principale, come direzione di sviluppo la direzione del raggio principale e di cui é stata definita l'apertura cioé il diametro della base.
Date le dimensioni del cono ipotetico viene generato il numero impostato di raggi, il primo é il raggio principale mentre gli altri vengono creati con queste caratteristiche:

- origine: comune a tutti i raggi del cono di v, corrispondente all'origine del cono quindi all'origine del raggio principale;

- direzione: varia per ogni raggio e viene generata mediante rotazione casuale della direzione del raggio principale rimanendo all'interno dell'apertura definita, in modo che il raggio durante il ray casting passi all'interno della base dell'ipotetico cono.

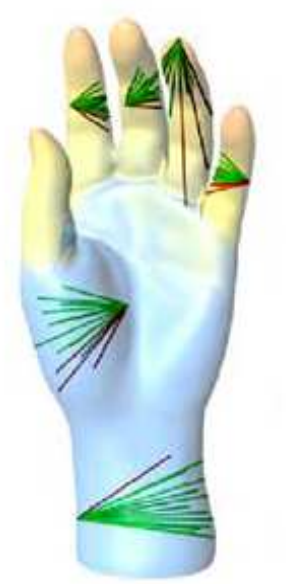

Figura 2.4: Esempio di generazione di coni di raggi in una mesh

2.2.2.2 Ray casting e ricerca intersezioni raggi-mesh

Creato il cono di raggi per lo specifico vertice v, ognuno di essi viene utilizzato per la ricerca dello spessore di v. In questo capitolo vengono analizzati i passi di cui si compone la ricerca di almeno una distanza valida tra un generico raggio r del cono di raggi del vertice v e la mesh.
Come prima cosa é importante analizzare l'utilitá della discretizzazione della mesh (paragrafo 2.2.1) e della conseguente creazione e riempimento della griglia (paragrafo 2.2.1.3).
Per riassumere, é stato creato un bounding box discreto contenente voxel che é il corrispettivo a coordinate intere del bounding box continuo che contiene la mesh, quindi ogni voxel copre la parte interna del bounding box discreto corrispondente alla stessa parte interna del bounding box continuo.
Quali sono i motivi che hanno portato a discretizzare il bounding box della mesh in voxel, indicizzare ogni voxel e discretizzare anche il bounding box di ogni face inserendo in una griglia, per ogni voxel, il riferimento ad una face tra quelle che giacciono su di esso? Principalmente sono due:

1. per calcolare nel migliore modo possibile lo spessore della mesh in v é bene trovare il punto di intersezione r-mesh piú vicino a v stesso, in modo da ottenere risultati corretti indipendentemente dalla conformazione della mesh. La motivazione é intuitiva: si ponga di avere una mesh a forma di cilindro con un certo spessore esterno ma internamente cava e che v sia un vertice nella superficie esterna. Lanciando da v un raggio con direzione parallela al piano su cui giace la base del cilindro e verso che punta all'interno del cilindro, il raggio interseca il cilindro stesso in tre punti: la superficie interna che forma lo spessore con la parte esterna da cui parte il raggio e le superfici interna ed esterna opposte che compongono l'altro spessore. Nel caso non venga cercata per prima l'intersezione piú vicina a v é possibile che l'algoritmo restituisca come risultato le superfici interna o esterna che compongono lo spessore opposto senza considerare la superficie interna che forma lo spessore con la superficie da cui parte il raggio, che é la prima intersezione che

il raggio incontra lungo la sua direzione. Questo ovviamente significa trovare il punto di intersezione errato quindi restituire come risultato una distanza, quindi uno spessore, errati.

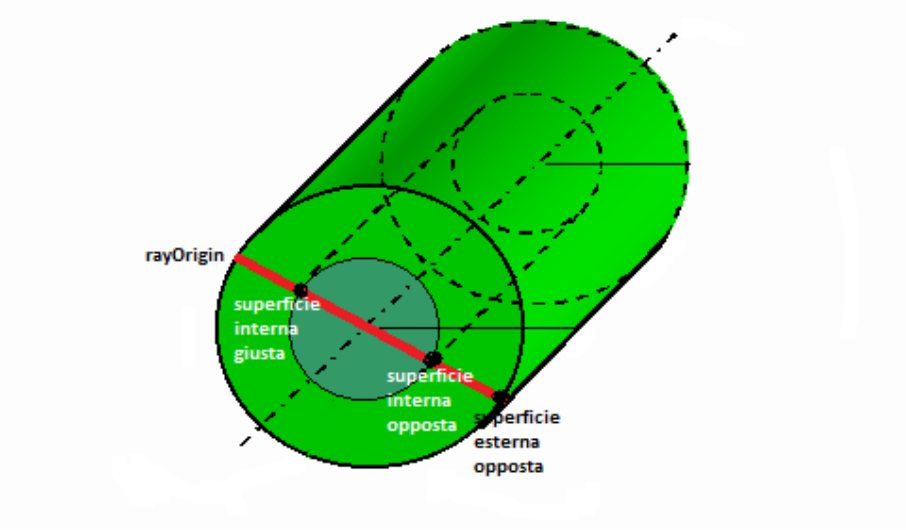

Figura 2.5: Esempio di ray casting in un cilindro cavo

Dalla situazione analizzata deriva che inizialmente é necessario cercare intersezioni con le face giacenti nell'intorno di v, ovviamente nella direzione del raggio. Avendo, per ogni voxel, il riferimento ad una face rappresentativa delle face aventi almeno una parte del proprio bounding box giacente nel voxel, é facile estrarre la face su cui cercare la prima intersezione r-mesh possibile. Infatti, una volta identificato il voxel contenente il punto v, si accede alla griglia utilizzando l'indice di quello stesso voxel per estrarre la face corrispondente. Ottenuta la face si testa se esiste un'intersezione tra r e la face stessa: in caso positivo quello é il risultato della ricerca delle intersezioni per r, in caso contrario é necessario allontanarsi da quel voxel, nella direzione di r, per andare a testare altre face;

2. il secondo motivo é quello di rendere il piú semplice possibile lo spostamento all'interno del bounding box. Per proseguire la ricerca nella

direzione del raggio é infatti sufficiente spostarsi di volta in volta ad uno dei voxel intorno a quello attuale, scegliendo il voxel successivo in base alla distanza tra voxel attuale ed i voxel nel suo intorno in rapporto all'entitá dello spostamento di r negli assi. In tal modo lo spostamento non avviene nel dominio continuo ma nella sua astrazione discreta, rendendo ogni passo molto semplice.

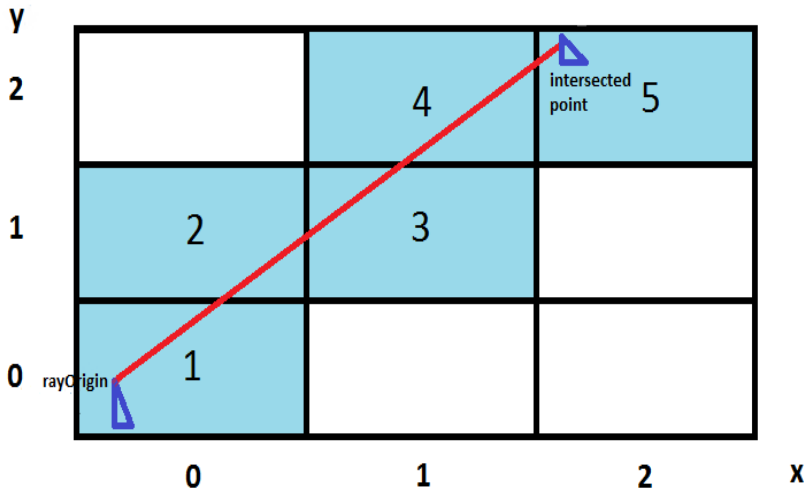

Figura 2.6: Esempio percorso di un raggio in 2D da punto di origine a punto di intersezione: i quadrati azzurri rappresentano i voxel in cui l'algoritmo si sposta nella sua ricerca di intersezioni ed i numeri indicano l'ordine di spostamento mentre i due triangoli rappresentano una face che insiste sul vertice da cui parte il raggio e la face intersecata

2.2.2.2.1 Impostazioni iniziali

All'inizio della ricerca di intersezioni per un qualsiasi raggio r del vertice v é necessario fare alcune operazioni per impostare valori utili per tutta la ricerca.

Come prima cosa é necessario normalizzare la direzione dello specifico raggio r.

Il secondo passo é la creazione dei marker: elementi a valore intero, uno per

la mesh ed uno per ogni singola face. I marker servono ad indicare se per la face attualmente in analisi é giá stata trovata una distanza valida tra r e la face stessa. Alla loro creazione, che avviene all'inizio della ricerca di intersezioni per ogni singolo raggio, vengono tutti inizializzati al valore 0. Immediatamente dopo il mesh marker viene incrementato di 1 (diventa 1) mentre i face marker restano a 0, ció indica che inizia l'iterazione di ricerca di intersezioni per uno specifico raggio (mesh marker = 1) e, ovviamente, che per nessuna face é ancora stata trovata una distanza valida (tutti i face marker = 0). Per segnalare che é stata trovata un'intersezione tra la mesh ed una face si pone il face marker corrispondente a quella stessa face uguale al valore del mesh marker, quindi 1. Dato che questi marker vengono ricreati per ogni raggio che viene lanciato non c'é alcun rischio di considerare una qualsiasi face da cui é stato ottenuto un risultato per l'ultimo raggio come giá visitata per il nuovo raggio perché, appunto, il suo valore viene resettato quando si passa al raggio successivo.
Successivamente é necessario identificare qual é il voxel che ricopre l'area del bounding box continuo contenente il punto origine del raggio; i valori delle 3 coordinate che lo identificano vengono denominati currentCellX, currentCellY, currentCellZ. Il loro calcolo avviene nella stessa modalitá vista precedentemente per calcolare il voxel contenente il punto minimo o il punto massimo del bounding box continuo di una face. D'ora in avanti con voxel attuale o voxel corrente viene identificato il voxel corrispondente nel bounding box discreto della mesh a questi tre valori. Essi cambiano di volta in volta durante l'esecuzione dell'algoritmo a seconda di qual é il voxel in cui l'algoritmo si sposta durante la ricerca di intersezioni.

Esempio:
Dati il punto origine del raggio:

- rayOrigin = (-5,4; 22,8; 9,0)

e il punto minimo del bounding box:

- bbMin: (-10,0; 0,0; -8,0)

ed infine la dimensione dei voxel nei 3 assi:

- voxel = (3,70370; 3,66667; 3,83334)

Si calcola:

- voxel discreto corrispondente al punto origine del raggio:
  t = (rayOriginX - bbMinX; rayOriginY - bbMinY; rayOriginZ - bbMinZ) = (-5,4 - -10,0; 22,8 - 0,0; 9,0 - -8,0) = (4,6; 22,8; 17,0)
  quindi:
  - currentCell = (Int($\frac{tX}{voxelX}$); Int($\frac{tY}{voxelY}$); Int($\frac{tZ}{voxelZ}$)) = (Int($\frac{4,6}{3,70370}$); Int($\frac{22,8}{3,66667}$); Int($\frac{17,0}{3,83334}$)) = (1; 6; 4)

In seguito viene calcolato il punto (nel dominio continuo) che ci poniamo come obiettivo. Inizialmente viene settato come il punto minimo continuo dell'area coperta dall'attuale voxel, cioé:

- in x: goalX = currentCellX*voxelX + bbMinX;
- in y: goalY = currentCellY*voxelY + bbMinY;
- in z: goalZ = currentCellZ*voxelZ + bbMinZ;

Il punto obiettivo a questo punto coincide col limite del voxel attuale verso la parte negativa nei tre assi, essendo il punto minimo in tutte le coordinate. Questo punto é corretto come obiettivo attuale per il raggio nel caso in cui esso abbia valore negativo per tutte le componenti del proprio vettore direzione, perché andrebbe 'indietro' nel voxel rispetto al vertice, quindi tenderebbe al valore minimo del voxel in ogni asse. Se invece il raggio ha valore positivo in almeno una delle componenti, il punto obiettivo attuale non é corretto perché almeno in un asse il raggio va 'in avanti' quindi tende al valore massimo del voxel. Per porre come punto obiettivo il limite del voxel a cui il raggio tende in base alle sue componenti direzione bisogna aggiornare il punto obiettivo negli assi in cui il raggio ha componente positiva sostituendo

al valore minimo del voxel il valore massimo. Ció si ottiene semplicemente aggiungendo ai valori prima impostati la dimensione dei voxel nei rispettivi assi.
Nel caso in cui il raggio abbia una o piú componenti nulle, in quegli assi non si modifica il valore del goal perché r non compie alcun spostamento in quell'asse quindi il valore del punto obiettivo in quell'asse é assolutamente superfluo.

Esempio:
Sia il punto minimo del bounding box:

- minimo: (bbMinX = -10,0; bbMinY = 0,0; bbMinZ = 8,0)

siano le dimensioni di ogni voxel:

- voxel = (3,70370; 3,66667; 3,83334)

e sia:

- currentCell = (1; 6; 4)

ed infine:

- rayDir = (0,16903; -0,50709; 0,84515)

Allora il punto obiettivo inizialmente é:

- goal = (currentCellX*voxelX + bbMinX; currentCellY*voxelY + bbMinY; currentCellZ*voxelZ + bbMinZ) = (1*3,70370 + -10,0; 6*3,66667 + 0,0; 4*3,83334 + -8,0) = (-6,2963; 22,0; 7,33336)

e considerando le componenti direzione diventa:

- in x: rayDirX $> 0 \rightarrow$ goalX = goalX + voxelX = -6,2963 + 3,70370 = -2,5926
- in y: rayDirY $\leq 0 \rightarrow$ goalY = 22,5
- in z: rayDir $> 0 \rightarrow$ goalZ = goalZ + voxelZ = 7,33336 + 3,83334 = 11,1667

quindi:

- goal = (-2,5926; 22,5; 11,1667)

Calcolato il punto obiettivo ora si calcola la distanza tra questo ed il punto corrispondente all'origine di r.

Esempio:
Dati:

- goal = (-2,5926; 22,5; 11,1667)
- rayOrigin = (-5,4; 22,8; 9,0)

allora:

- diff = (-5,4 - -2,5926; 22,8 - 22,5; 9,0 - 11,16667) = (-2,8074; 0,3; -2,16667)

quindi:

- dist = $\sqrt{(-2,8074)^2 + (0,3)^2 + (-2,16667)^2} = \sqrt{7,88149 + 0,09 + 4,69445}$ $= \sqrt{12,66594} = 3,55892$

L'ultimo passo prima di iniziare il ciclo di ricerca di intersezioni é il calcolo di qual é l'asse in cui r é piú vicino al goal in relazione alla grandezza delle componenti della direzione. Per fare questo si calcola, per ogni asse, quanti 'passi' deve compiere il raggio affinché, partendo dalla sua origine, possa raggiungere il goal. Ció si traduce nel calcolare, per ogni asse, la differenza tra goal e punto di origine di r e nel dividere il risultato per la specifica componente della direzione di r. Ovviamente l'asse in cui r é piú vicino al goal é l'asse corrispondente al minore numero di passi.
Il calcolo precedente vale per gli assi la cui componente direzionale del raggio ha valore assoluto maggiore di una certa tolleranza, detta epsilon, altrimenti il numero di passi viene settato al valore in virgola mobile piú alto possibile.

Questo perché r negli assi in cui ha componente direzionale di valore estremamente piccolo o addirittura zero ci metterebbe un numero di passi quasi infinito per arrivare all'obiettivo, quindi semplicemente il numero di passi viene settato al valore massimo possibile.

Esempio:
Dati:

- goal = (-2,5926; 22,5; 11,16667)
- rayOrigin = (-5,4; 22,8; 9,0)
- rayDir = (0,16903; -0,50709; 0,84515)

da cui:

- stepX = $\frac{goalX - rayOriginX}{rayDirX} = \frac{-2,5926 - -5,4}{0,16903} = \frac{2,8074}{0,16903} = 16,60888$
- stepY = $\frac{goalY - rayOriginY}{rayDirY} = \frac{22,5 - 22,8}{-0,50709} = \frac{-0,3}{-0,50709} = 0,59161$
- stepZ = $\frac{goalZ - rayOriginZ}{rayDirZ} = \frac{11,16667 - 9,0}{0,84515} = \frac{2,16667}{0,84515} = 2,56365$

quindi:

- step = (16,60888; 0,59161; 2,56365)

Quindi l'asse in cui r é piú vicino al goal é l'asse y.

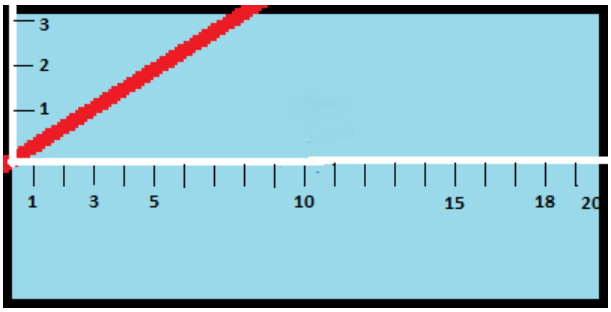

Figura 2.7: Esempio calcolo step in 2D: in rosso il raggio, in bianco le componenti x ed y del raggio; in quel voxel il raggio dal suo punto di ingresso deve muoversi di 20 passi in x e 3,5 passi in y per uscire dal voxel nei rispettivi assi; quindi l'asse in cui é piú vicino al goal é y

2.2.2.2.2 Ciclo di ricerca intersezione raggio-mesh

Finita l'impostazione di tutti i dati necessari per l'iterazione del raggio r del vertice v della ricerca delle intersezioni, é proprio a questa fase che l'algoritmo passa. Questa é la fase piú importante perché si occupa di calcolare i risultati che l'algoritmo restituisce all'utente alla fine della sua esecuzione, cioé l'insieme delle distanze calcolate. Tale fase consiste di un ciclo while:

```
while (!End() && Refresh())
    NextCell()
```

Per capire il funzionamento del ciclo é utile un'analisi dettagliata di ognuna di queste funzioni:

- End(): funzione booleana che restituisce il valore di un flag, detto end, che é true se il voxel attuale é identificato da almeno una coordinata con valore < 0 o $\geq$ al numero di parti in cui é diviso quell'asse (cioé se il voxel é al di fuori del bounding box), false altrimenti. Ovviamente ne deriva che !End() fa proseguire il ciclo while se End() restituisce false, cioé se il voxel attuale é interno al bounding box.

  Esempio:
  Dato:

  - siz = (27; 15; 6)

  se:

  - currentCell = (1; 6; 4)

  allora:

  - End() = false (currentCellX/Y/Z ≥ 0 e $<$ sizX/sizY/sizZ rispettivamente) $\rightarrow$!End() verificato $\rightarrow$ il ciclo while puó proseguire

invece se:

- currentCell = (20; 10; 6)

allora:

- End() = true (currentCellZ $\geq$ sizZ) $\rightarrow$!End() non verificato $\rightarrow$ il ciclo while termina

- Refresh(): é la funzione principale di tutto l'algoritmo, quella che testa se esiste un'intersezione tra r ed una face. Il raggio r é giá conosciuto, mentre bisogna estrarre dalla griglia il riferimento alla face da testare. In questa implementazione ad ogni iterazione il test viene eseguito sulla face contenuta nel link dell'elemento della griglia corrispondente al voxel attuale. In particolare vengono recuperati dalla griglia due link:
  - first: il link contenuto nell'elemento della griglia corrispondente al voxel attuale, cioé quello identificato (utilizzando la funzione vista al paragrafo 2.2.1.2) dai valori currentCellX/Y/Z;
  - last: il link nell'elemento della griglia immediatamente successivo all'elemento sopra descritto.

Esempio:
Ponendo di riprendere una parte di griglia da un esempio precedente:

- elemento 891: link(face0, 891)
- elemento 892: link(face2, 894)
- elemento 893: link(face2, 894)
- elemento 894: link(face2, 894)

Sia:

- siz = (27; 15; 6)

Se:

- currentCell = (0; 3; 2)

Allora:

- index = currentCellX + ((currentCellY + currentCellZ * sizY) * sizX) = 0 + ((3 + 2*15) *27) = 0 + 891 = 891

quindi:

- first = link all'elemento 891
- last = link all'elemento 892.

L'algoritmo seleziona due link ad ogni iterazione ma esegue il test di intersezione soltanto sulla face contenuta nel link copiato in first, quello corrispondente al voxel attuale. Il test di intersezione tra r e la face é costituito da un if su tre condizioni che devono tutte essere verificate affinché lo sia la condizione totale:

- la prima condizione é che il face marker della face in analisi non abbia lo stesso valore del mesh marker, cioé che sia 0 non 1. Questa condizione controlla che per quella face non sia ancora stata trovata alcuna intersezione;
- la seconda é che esista un'intersezione tra r e face. In seguito questa parte viene analizzata in dettaglio;
- la terza é che la distanza dall'origine del raggio del punto di intersezione eventualmente trovato abbia valore $\leq$ della diagonale del bounding box.

Se queste tre condizioni sono tutte vere significa che é stata trovata un'intersezione valida tra r e la face, quindi l'algoritmo inserisce la face, il punto di intersezione (rayOrigin + rayDir*t) e la distanza (t) nella lista dei risultati dell'iterazione di ricerca per l'attuale raggio. In seguito la face viene segnata come visitata ponendo il relativo faceMarker a

1 in modo che non venga piú testata per questo raggio perché sarebbe inutile in quanto é giá stato calcolato un punto di intersezione valido.
Se invece almeno una delle condizioni prima descritte non é vera significa che non é stata trovata intersezione valida quindi non viene aggiunto alcun risultato alla lista attuale e la face non viene segnata come visitata.
Qualsiasi sia il risultato della ricerca di intersezione tra r e la face, gli elementi nella lista dei risultati vengono ordinati per distanza crescente e viene settato come risultato dell'iterazione il primo elemento, cioé quello avente minore distanza. Ovviamente se non é ancora stata trovata alcuna intersezione la lista dei risultati é vuota e di conseguenza l'elemento risultato non ha alcun significato, é un elemento nullo.
La funzione termina restituendo come risultato un valore booleano che é:

- true: in due casi:
  - se la lista di risultati é vuota cioé se non é ancora stata trovata alcuna intersezione per r;
  - se la distanza dell'elemento risultato é > di dist, cioé se l'intersezione trovata non é considerata valida perché fuori dal limite goal a cui il raggio tende attualmente, ovvero se la distanza trovata é maggiore della distanza tra il punto di partenza del raggio ed il punto goal attuale.
- false: se la lista di risultati contiene almeno un elemento cioé almeno un'intersezione trovata e se essa non é oltre il goal.

Analizzando la condizione del while, il ciclo prosegue se sia !End() che Refresh() sono verificate, cioé se il voxel attuale é interno al bounding box (!End() true) e se non é stata ancora trovata alcuna intersezione valida (Refresh() true), altrimenti il ciclo termina.
Se la condizione del while é verificata, l'algoritmo prosegue eseguendo la funzione

- NextCell(): funzione che compie gli spostamenti all'interno del bounding box discreto, modificando il voxel attuale e lo step nell'asse che l'algoritmo decide di utilizzare per spostarsi che é quello corrispondente al minore passo, cioé in cui r deve fare meno spostamenti per raggiungere il voxel successivo.
  Innanzitutto vengono impostati i valori minimo e massimo di un nuovo bounding box continuo corrispondenti ai valori del bounding box del voxel attuale.

  Esempio:
  Sia il punto minimo del bounding box:

  – minimo: (bbFaceMinX = -10,0; bbFaceMinY = 0,0; bbFaceMinZ = 9,0)

  Siano le dimensioni di ogni voxel:

  – voxel = (3,70370; 3,66667; 3,83334)

  e sia:

  – currentCell = (1; 6; 4)

  Allora il punto minimo del bounding box del voxel identificato é:

  – minimo = (currentCellX*voxelX + bbMinX; currentCellY*voxelY + bbMinY; currentCellZ*voxelZ + bbMinZ) = (1*3,70370 + -10,0; 6*3,66667 + 0,0; 4*3,83334 + -8,0) = (-6,2963; 22,0; 7,33336)

  Mentre il punto massimo é:

  – massimo = minimo + voxel = (-6,2963 + 3,70370; 22,0 + 3,66667; 7,33336 + 3,83334) = (-2,5926; 25,66667; 11,16667)

  In seguito vengono effettuati calcoli sulla posizione dell'origine di r rispetto al bounding box appena calcolato, settando un punto inters in questo modo:

- se r parte da un punto interno al bounding box appena calcolato (cioé é all'interno del voxel attuale): inters = origine di r.

  Esempio:
  Dati dall'esempio precedente:

  * punto minimo: (-6,2963; 22,0; 7,33336)
  * punto massimo: (-2,5926; 25,66667; 11,16667)

  Sia:

  * rayOrigin = (-5,4; 22,8; 9,0)

  Allora:

  * origine r interna a bounding box, quindi: inters = (-5,4; 22,8; 9,0)

- altrimenti se r parte da un punto esterno al bounding box appena calcolato in un qualsiasi asse, cioé se rayOrigin non é interno al voxel attuale: inters viene calcolato in vari passi:

  1. calcolo di maxStep:
     * negli assi in cui il punto origine di r 'e esterno al bounding box e la cui componente direzionale é diversa da 0: il valore maxStep viene calcolato in questo modo:
       · maxStep = (valore del bounding box ecceduto in quell'asse - rayOrigin in quell'asse) / rayDir in quell'asse;
     * negli altri assi:
       · maxStep = -1
  2. si sceglie il maxStep di piú alto valore:
     * se é < 0 (cioé la componente direzionale di r nell'asse o negli assi in cui il punto origine di r é fuori dal bounding box ha valore nullo): niente;
     * altrimenti si calcola inters in questo modo:

- nell'asse corrispondente al maxStep piú alto: inters = valore del bounding box ecceduto in quell'asse;
- negli altri assi: inters = rayOrigin in quell'asse + maxStep piú alto*rayDir in quell'asse. Se inters eccede il valore minimo o massimo del bounding box in quell'asse allora non é possibile trovare un intersezione tra rayOrigin ed il bounding box e inters potrebbe non essere completamente settato.

Esempio:
Dati:

* punto minimo: (-6,2963; 25,66667; 7,33336)
* punto massimo: (-2,5926; 29,33334; 11,16667)

Sia:

* rayOrigin = (-5,4; 30; 7,0)
* rayDir = (0,1; 0; 0,5)

Allora:

* rayOrigin esterno a bounding box in y e z: 30 > 29,33334 e 7,0 < 7,33336

Quindi bisogna calcolare maxStep:

* maxStepX = -1: perché rayOriginX interno a bounding box in X (-6,2963 < -5,4 < -2,5926)
* maxStepY = -1: perché rayOriginY esterno a bounding box in Y (30 > 29,33334 > 25,66667) ma rayDirY = 0
* maxStepZ = (valore bounding box ecceduto in Z - rayOriginZ) / rayDirZ = (7,33336 - 7,0) / 0,5 = 0,66672: perché rayOriginZ esterno a bounding box in Z (7,0 < 11,16667 < 7,33336) e rayDirZ non nulla.

Il maxStep di valore piú alto é ovviamente quello nell'asse Z, quindi:

* intersX = rayOriginX + maxStepZ * rayDirX = -5,4 + 0,66672 * 0,1 = -5,4 + 0,066672 = -5,33332
* intersY = rayOriginY + maxStepZ * rayDirY = 30 + 0,66672 * 0 = 30
* intersZ = valore ecceduto bounding box in Z = 7,33336

Controllo su inters assicurandosi che negli assi diversi da Z sia interno a bounding box:

* intersX: -6,2963 < -5,33332 < -2,5926 → OK
* intersY: 30 > 29,33334 > 25,66667 → NO

Successivamente viene calcolata la distanza tra inters e rayOrigin di r, se tale distanza é > della diagonale del bounding box della mesh viene settato il flag end a true e End() dopo restituirá true facendo terminare la ricerca perch'e l'algoritmo é uscito dal bounding box della mesh. Se invece la distanza é $\leq$ della diagonale del bounding box della mesh l'algoritmo si sposta in un nuovo voxel modificando currentCell e goal nell'asse corrispondente al minore valore di step (cioé nell'asse in cui r deve fare meno spostamenti lungo quella direzione per giungere dalla sua origine al punto obiettivo):

- se componente di rayDir in quell'asse > 0 (il raggio in quell'asse va 'avanti'):
  * goal = goal + voxel
  * currentCell = currentCell + 1
  * step = $\frac{goal - rayOrigin}{rayDir}$
- altrimenti (componente < 0, quindi il raggio in quell'asse va 'indietro'):

* goal = goal - voxel
* currentCell = currentCell - 1
* step = $\frac{goal - rayOrigin}{rayDir}$

Infine viene ricalcolata la distanza assoluta tra rayOrigin di r e goal e controllato che il nuovo voxel attuale sia interno al bounding box, cioé viene impostato il flag end (quindi il valore restituito dalla funzione End()) a true se il nuovo voxel ha la coordinata modificata < 0 o $\geq$ al numero di voxel possibili in quell'asse, false altrimenti.

Esempio:
Dati:

- voxel = (3,70370; 3,66667; 3,83334)
- rayOrigin = (-5,4; 22,8; 9,0)
- rayDir = (0,16903; -0,50709; 0,84515)
- currentCell = (1; 6; 4)
- goal = (-2,5926; 22,5; 11,16667)
- step = (16,60888; 0,59161; 2,56365)

L'asse con minore step é y, quindi bisogna modificare i valori in quell'asse. RayDirY é < 0 quindi:

- goalY = 22,5 - 3,66667 = 18,83333
- currentCellY = 6 - 1 = 5
- stepY = $\frac{goalY - rayOriginY}{rayDirY} = \frac{18,83333 - 22,8}{-0,50709} = 7,82241$

Quindi:

- goal = (-2,5926; 18,83333; 11,16667)
- currentCell = (1; 5; 4)
- step = (16,60888; 7,82241; 2,56365)

L'asse avente il minore step ora é z. Poi si aggiorna:

- diff = (rayOriginX - goalX; rayOriginY - goalY; rayOriginZ - goalZ) = (-5,4 - -2,5926; 22,8 - 18,83333; 9,0 - 11,16667) = (-2,8074; 3,96667; -2,16667)

quindi:

- dist = $\sqrt{(-2,8074)^2 + (3,96667)^2 + (-2,16667)^2} = \sqrt{7,88149 + 15,73447 + 4,69445}$ = $\sqrt{28,31041}$ = 5,32075

Infine:

- End() = false (currentCellX/Y/Z $\geq$ 0 e $<$ sizX/sizY/sizZ rispettivamente) $\rightarrow$!End() successivo verificato

L'aggiornamento appena visto del valore dei parametri impostati all'inizio dell'iterazione di ricerca ed utili per quella specifica fase permettono, come giá detto, lo spostamento dell'algoritmo da un voxel al successivo all'interno del bounding box discreto della mesh per proseguire nella ricerca di intersezioni nella nuova area raggiunta.
Terminata questa funzione l'algoritmo torna a valutare la condizione del while.

Se invece la condizione del ciclo while non é verificata l'algoritmo esce dal ciclo while senza eseguire la funzione NextCell().

2.2.2.2.3 Inserimento risultato ricerca intersezioni e proseguimento/terminazione algoritmo

Una volta che l'algoritmo é uscito dal ciclo while analizzato precedentemente, la ricerca di intersezioni per l'attuale coppia r-v é terminata.
A questo punto l'algoritmo deve inserire nella lista di output il risultato della ricerca per tale coppia. A tal fine viene testato il flag end, due casi possibili:

1. end = false: vuol dire che l'algoritmo ha trovato un'intersezione valida senza uscire dal bounding box, quindi l'elemento risultato attuale

impostato contiene la distanza trovata che viene aggiunta alla lista di risultati dell'algoritmo;

2. end = true: l'algoritmo é uscito dai voxel del bounding box senza trovare alcuna intersezione valida, quindi viene aggiunto alla lista di risultati un valore indefinito (distanza indefinita per indicare che non é stata trovata alcuna intersezione tra r e la mesh).

Inserito il risultato per il raggio r del vertice v nella lista di risultati, se mancano altri raggi da testare nel cono creato su v l'algoritmo passa alla ricerca di intersezioni tra la mesh ed il successivo raggio r+1 dello stesso vertice v.
Se invece r é l'ultimo raggio del cono di raggi di v l'algoritmo analizza la lista di risultati ottenuti per l'attuale vertice eseguendo alcune correzioni prima di definire il valore dello spessore del vertice. L'algoritmo elimina i valori indefiniti ottenuti, poi elimina gli outlier: prende il valore minimo ed il valore massimo, sia 0% il minimo e 100% il massimo vengono eliminati i valori inferiori al 10% e maggiori del 90%. Tra le distanze rimanenti viene calcolato il valore medio e quello é lo spessore calcolato per quel vertice. Impostato lo spessore di v si possono verificare due casi:

- v non é l'ultimo vertice della mesh: l'algoritmo passa al vertice successivo ed effettua la ricerca delle intersezioni per il vertice v+1, cioé crea il cono di raggi ed esegue i test di intersezione tra mesh ed ogni raggio del cono creato;

- v é l'ultimo vertice della mesh: l'algoritmo ha inserito nella lista di risultati il valore calcolato per ogni raggio lanciato dai vertici.

Terminato il riempimento della lista di risultati l'algoritmo calcola la distanza maggiore e minore presente nella lista considerando le sole distanze valide cioé escludendo i valori indefiniti.
I valori calcolati insieme alla lista di distanze calcolate permettono di creare un istogramma rappresentante lo spessore generale della mesh.

Capitolo 3

Ottimizzazione algoritmo

L'algoritmo SDF implementato in MeshLab é stato analizzato ed utilizzato come riferimento per implementare una versione progettata sulla base dell'obiettivo che il suo utilizzo si pone cioé la caratterizzazione delle proprietá geometriche di modelli di componenti meccanici, per lo piú elementi in plastica. Durante l'implementazione di questa particolare versione della funzione SDF l'algoritmo é stato ampiamente modificato testando, passo dopo passo, l'impatto di ogni possibile variazione sulla precisione e sul tempo di esecuzione dell'algoritmo. Ció é stato fatto per assicurarsi di rimodellare l'algoritmo inserendo, ove possibile, modifiche che permettessero di elevare la qualitá complessiva dell'implementazione utilizzando come dati di confronto sia la quantitá e qualitá delle distanze ottenute sia il tempo di esecuzione complessivo.
Le modifiche apportate possono essere divise in due tipologie:

1. scelte implementative specifiche vincolate dalle caratteristiche strutturali dell'applicativo in cui la funzione é stata inserita;

2. modifiche strutturali apportate all'algoritmo stesso.

3.1 Implementazione algoritmo SDF in linguaggio VB.NET

La struttura dell'algoritmo descritta al capitolo precedente é stata ripresa ed implementata nel programma di gestione ed analisi di mesh piú volte citato. Questo programma é scritto in codice Visual Basic .NET quindi durante l'implementazione é stato fatto un lavoro di ottimizzaziones mediante l'utilizzo di strutture dati ed una strutturazione dell'algoritmo stesso su misura per il linguaggio di programmazione utilizzato.

3.2 Modifiche alla struttura dell'algoritmo

Nella logica dell'algoritmo sono state riscontrate alcune criticitá importanti, tra cui:

- necessitá/possibilitá di velocizzare l'esecuzione dell'algoritmo;
- necessitá di migliorare i risultati restituiti dall'algoritmo.

É alla risoluzione di queste problematiche che é stata riservata grande attenzione in seguito all'analisi dei risultati ottenuti con una prima versione dell'implementazione della funzione SDF in quanto inizialmente i test eseguiti sulle mesh campione evidenziavano risultati non soddisfacenti, sia dal punto di vista del tempo di esecuzione che degli spessori calcolati. Infatti, come spiegato al capitolo successivo, l'algoritmo riusciva a calcolare la distanza per approssimativamente meno della metá dei vertici delle mesh campione.
Di seguito sono descritte le principali modifiche apportate alla struttura della funzione.

3.2.1 Clusterizzazione del bounding box delle face e riempimento della griglia di indirizzamento spaziale

Il primo importante cambiamento é stato effettuato sulla griglia, infatti analizzandone il riempimento sono sorti tre punti di discussione:

1. inizialmente vengono creati per ogni voxel tanti link quante sono le face il cui bounding box occupa una qualsiasi parte di quel voxel, ma nella fase di ricerca delle intersezioni vengono utilizzati solo i link inseriti nella griglia che sono soltanto uno per ogni voxel;

2. inserendo nella griglia solo un link per ogni voxel é estremamente probabile che vengano perse delle face su cui testare le intersezioni (l'unica possibilitá di non perdere delle face é che in ogni voxel giaccia al massimo una sola face);

3. inserire nella griglia, in corrispondenza di voxel vuoti, face il cui bounding box non giace nel voxel rischia di provocare test di intersezione inutili.

Queste problematiche riscontrate nella struttura dell'algoritmo hanno portato alla modifica della modalitá di riempimento della griglia di indirizzamento spaziale, infatti in ogni elemento della griglia non viene inserito piú uno ed un solo link ma una lista di link di dimensione variabile. Questo ha permesso di eliminare le fasi di inserimento dei link in una lista e successivo ordinamento di essa perché divenute inutili, infatti ora i link vengono creati ed immediatamente inseriti nella griglia.
Ecco i passi compiuti dall'algoritmo per ogni face della mesh:

- (come prima) calcolo del bounding box continuo della face;

- (come prima) calcolo dei voxel in cui giace il bounding box continuo;

- per ogni voxel cosí trovato:

  - (come prima) creazione di un link con:
    * face = face attuale (sempre la stessa per questi link);
    * index = l'indice che identifica lo specifico voxel attuale
  - (nuovo) inserimento del link nella lista all'elemento della griglia di indice corrispondente ad index.

Da queste modifiche deriva che:

- come detto precedentemente, non viene creata né ordinata la lista di link;
- la lista presente in un singolo elemento della griglia puó essere:
  - vuota: se nell'area coperta dal corrispondente voxel non giace il bounding box di alcuna face. Di conseguenza gli elementi corrispondenti a voxel in cui non giace il bounding box di alcuna face non hanno al loro interno link con riferimento ad una face il cui bounding box non giace in quel voxel ma sono semplicemente vuoti;
  - composta da n link: un link per ogni face il cui bounding box giace in quel voxel. Da questo deriva che gli elementi corrispondenti a voxel in cui giace il bounding box di piú di una face non si limitano piú a contenere il riferimento ad una ed una sola di queste face, scelta come rappresentativa, ma contengono i link di tutte le face appartenenti a quel voxel. Ció permette di calcolare l'intersezione su tutte le face giacenti in quel voxel.

3.2.2 Ricerca intersezioni

La struttura della funzione SDF é stata modificata anche nelle parti relative alla ricerca delle intersezioni tra i raggi e la mesh, in particolare i cambiamenti piú significativi sono stati operati sulla parte riguardante la generazione del cono di raggi e quella relativa alla ricerca di intersezioni tra le face e la mesh.

3.2.2.1 Generazione del cono di raggi

Per quanto riguarda la generazione del cono di raggi é stato eliminato lo scostamento del punto origine del raggio principale rispetto al punto corrispondente al vertice su cui viene creato. Dato che il punto di origine del raggio principale corrisponde al punto di origine del cono e di tutti i raggi creati al suo interno, é facilmente deducibile che grazie a questa modifica ogni raggio creato ha come origine esattamente il vertice.
Lo scostamento nella direzione del raggio, pur essendo minimo, sposta il punto di partenza del raggio, quindi causa una variazione dei valori di distanza di volta in volta calcolati dall'algoritmo. La modifica apportata permette quindi di rendere piú precisi gli spessori calcolati facendo partire ogni raggio relativo ad un certo vertice proprio dal punto corrispondente al vertice stesso. D'altra parte inserire uno scostamento del punto di origine dei raggi permette di evitare di trovare intersezioni con le face aventi tra i propri vertici lo stesso vertice su cui vengono creati i raggi. Quindi l'eliminazione di questo scostamento puó provocare come effetto collaterale il calcolo di intersezioni con le face create intorno a quel vertice. Questo problema é stato risolto inserendo un controllo nella funzione di calcolo dell'intersezione tra un raggio ed una face, cioé viene testato se i vertici della face corrispondono al vertice da cui parte il raggio. Se l'algoritmo calcola almeno una corrispondenza si comporta come se tra il raggio e la face non ci fosse alcuna intersezione, cioé restituisce risultato negativo al test di intersezione. Se invece nessuno dei vertici dell face é il medesimo vertice da cui ha origine il raggio l'esecuzione prosegue normalmente cercando una possibile intersezione tra raggio e face.

Un'altra modifica importante effettuata é la drastica diminuzione del numero di raggi creati per ogni cono e quindi usati per la ricerca di intersezioni. Come si puó evincere dai test inseriti nel capitolo successivo, la modifica apportata ha permesso di diminuire drasticamente il tempo di esecuzione dell'algoritmo senza avere un importante impatto negativo sulla quantitá di distanze corrette calcolate né sulla loro qualitá.

3.2.2.2 Ciclo di ricerca intersezione raggio-mesh

La variazione relativa al riempimento della griglia che ora é composta in ogni elemento da una lista di link ha i principali effetti su questa parte dell'algoritmo. Non essendoci piú uno ed un solo link in ognuno degli elementi della griglia, durante la fase di reperimento delle face relative al voxel attuale in cui l'algoritmo si é mosso non vengono piú presi i link presenti agli elementi della griglia corrispondenti al voxel attuale ed al successivo. Infatti ogni elemento della griglia ora é composto da una lista di link, quindi ogni qualvolta bisogna reperire le face su cui effettuare il test viene presa la lista completa inserita nell'elemento corrispondente al voxel attuale. Questo permette di effettuare il test di intersezione su tutte e le sole le face giacenti nel voxel attuale. Di conseguenza per ogni voxel il test di intersezione non viene effettuato una ed una sola volta cioé su una ed una sola face ma n volte, tante quante sono le face che giacciono nel voxel attuale. Ovviamente se il voxel attuale non contiene il bounding box di alcuna face il test di intersezione per questo voxel non viene effettuato nessuna volta, mentre se contiene una sola face il test viene effettuato una sola volta come nella versione precedente.
Nella parte relativa ai test é stata inserita la dimostrazione che queste modifiche hanno portato un grande miglioramento all'algoritmo, infatti hanno garantito un notevole aumento della percentuale di risultati corretti restituiti dall'algoritmo in quanto per via dell'aumento del numero di face, quindi della superficie della mesh, su cui vengono eseguiti i test delle intersezioni.

La seconda modifica apportata in questa parte del codice é la modalitá di etichettatura di una singola face come visitata per il raggio attuale. In seguito al cambiamento di questa parte una face viene etichettata come visitata in questi casi:

- se é stata trovata un'intersezione e il punto intersecato é ad una distanza dall'origine del raggio non superiore alla distanza che si ha tra la stessa origine del raggio ed il punto obiettivo attuale, cioé se il punto di intersezione trovato é considerato valido. La face viene segnata come

visitata perché per essa é stato trovato un punto di intersezione valido ed ogni successivo test tra lo stesso raggio e la stessa face darebbe il medesimo risultato quindi sarebbe inutile da ripetere;

- se non é stata trovata intersezione, anche in questo caso ogni successivo test tra lo stesso raggio e la stessa face darebbe il medesimo risultato e la ripetizione sarebbe inutile.

L'unico caso in cui la face non viene segnata come visitata é quello in cui é stata trovata un'intersezione ed il punto corrispondente é piú distante dall'origine del raggio rispetto al punto obiettivo attuale, quindi la distanza non viene considerata valida. La face in questo caso non viene etichettata come visitata perché altrimenti in caso di spostamento ad un successivo voxel il quale contenesse tra le face in esso giacenti anche quella stessa face, essa non verrebbe testata perché considerata come giá testata. Questo potrebbe provocare errori perché avendo spostato il goal a causa dello spostamento al nuovo voxel il punto di intersezione tra raggio e face potrebbe ora essere compreso nell'area tra il punto di origine del raggio ed il nuovo punto obiettivo. Di conseguenza l'eventuale distanza che questa face permetterebbe di calcolare non sarebbe considerata come risultato valido potendo provocare o il fallimento della ricerca di intersezione tra il raggio e la face oppure il calcolo di una distanza piú ampia in modo erroneo.
La variazione alle regole di etichettatura di una face permette sopratutto di evitare, ove possibile, la ripetizione di test di intersezione inutili in particolare per mesh in cui si verifica piú volte lo spostamento di voxel in voxel nella direzione del raggio.
Per poter implementare questa differenziazione di casi per decidere se segnare una face come visitata oppure no é stato necessario modificare la parte relativa alla condizione che regola l'inserimento dell'intersezione trovata e dell'etichettatura della face. La condizione é stata scomposta in 3 parti:

1. se la face é etichettata come visitata: si passa alla face successiva o se questa é l'ultima face della lista inserita in quello specifico elemento

si prosegue con l'ordinamento dei risultati, selezione del risultato di minore distanza e terminazione della funzione;

2. se la face non é etichettata si esegue il test di intersezione, due casi:
   (a) test positivo: si controlla che la distanza calcolata non abbia valore superiore alla diagonale del bounding box ed alla distanza tra l'origine del raggio ed il punto obiettivo attuale:
      i. in caso positivo: si inseriscono i dati ottenuti nella lista di risultati per l'attuale raggio e la face viene segnata come visitata;
      ii. se invece la distanza ha valore maggiore della diagonale del bounding box: non viene aggiunto nessun risultato ma la face viene segnata come visitata perché la distanza tra origine del raggio ed intersezione é maggiore della distanza tra punto minimo e massimo del bounding box quindi é una distanza sempre non valida da non ricalcolare perché ottenuta con almeno uno dei due punti esterno al bounding box della mesh.
   (b) test negativo: non viene aggiunto nessun risultato e la face viene etichettata come visitata.

Come si puó notare da questi casi é escluso quello in cui é stato trovato un punto di intersezione avente distanza non superiore alla diagonale del bounding box ma superiore alla distanza tra origine del raggio e punto obiettivo. Questo perché, come detto prima, questa distanza non puó essere attualmente valida perché oltre l'intorno ora in analisi ma nemmeno definitivamente scartata perché potrebbe diventare una distanza valida nel caso in cui l'algoritmo si spostasse a ricercare intersezioni per quello stesso raggio nel voxel contenente quel punto di intersezione.

3.2.2.2.1 Test di intersezione raggio-face

Le ultime modifiche importanti al codice sono state apportate alla funzione di test di intersezione tra un raggio ed una face.

Innanzitutto viene controllato che il punto da cui parte il raggio non sia il punto corrispondente ad uno qualsiasi dei vertici della face. Se uno dei tre vertici corrisponde all'origine del raggio la funzione restituisce immediatamente risultato negativo al test perché l'eventuale intersezione trovata non sarebbe corretta. Questa evenienza é possibile avendo eliminato lo scostamento del punto origine del raggio rispetto al vertice come descritto precedentemente. Se invece nessuno dei vertici della mesh corrisponde al vertice da cui parte il raggio l'algoritmo prosegue eseguendo il test geometrico di intersezione raggio-mesh.

Capitolo 4

Test e validazione risultati

Di pari passo con la fase relativa alla modifica del codice per ottimizzarlo e renderlo piú conforme allo scopo di questa tesi, ad ogni iterazione di modifica sono corrisposti una serie di test su alcune mesh utilizzate come benchmark. La validazione di ogni variazione quindi é passata necessariamente dall'analisi e riscontro positivo dei relativi test.

4.1 Indici di valutazione

Per valutare l'impatto di ogni modifica sull'efficacia ed efficienza dell'algoritmo sono state utilizzate tre metriche, divise in due tipologie:

- metriche di precisione:
  - percentuale di distanze valide ottenute: percentuale di valori corretti (quindi non valori indefiniti) ottenuti rispetto al numero di totale di vertici di cui é composta la mesh in analisi;
  - caratteristiche geometriche rilevate dai risultati: analisi dello spessore minimo e massimo e della frequenza degli spessori. In particolare la frequenza é stata calcolata creando 101 intervalli: il primo corrisponde ai valori indefiniti, gli altri sono generati dividendo in 100 intervalli i valori da 0 allo spessore massimo riscontrato. Ov-

viamente ciascun intervallo contiene il valore corrispondente alla frequenza nei risultati delle distanze che esso contiene;

- metrica temporale:
  - tempo di esecuzione dell'algoritmo: di volta in volta é stato monitorato il valore cercando di diminuirlo ove possibile o al piú mantenerlo pressoché invariato

Di volta in volta sono state validate le modifiche i cui test garantissero risultati riconducibili ad uno di questi casi:

- miglioramento dal punto di vista della precisione o del tempo di esecuzione senza avere impatto negativo sull'altro tipo di metrica;
- netto miglioramento in una delle due tipologie di metriche senza avere ripercussioni negative importanti sull'altra.

4.2 Mesh di testing

Per la valutazione di ogni variazione del codice sono stati eseguiti test su alcune mesh presentate nelle figure seguenti:

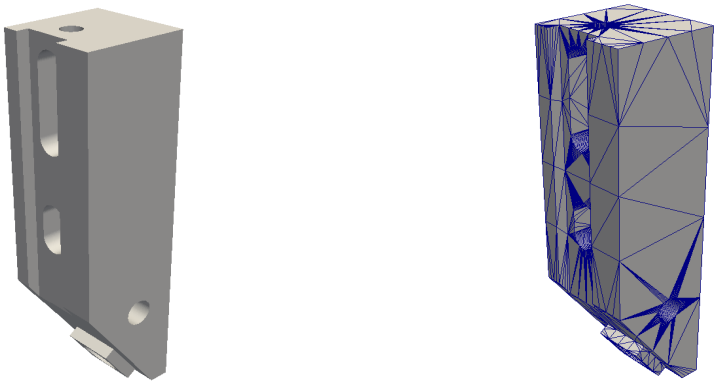

Figura 4.1: Mesh 1: numero vertici: 1115, numero face: 2246

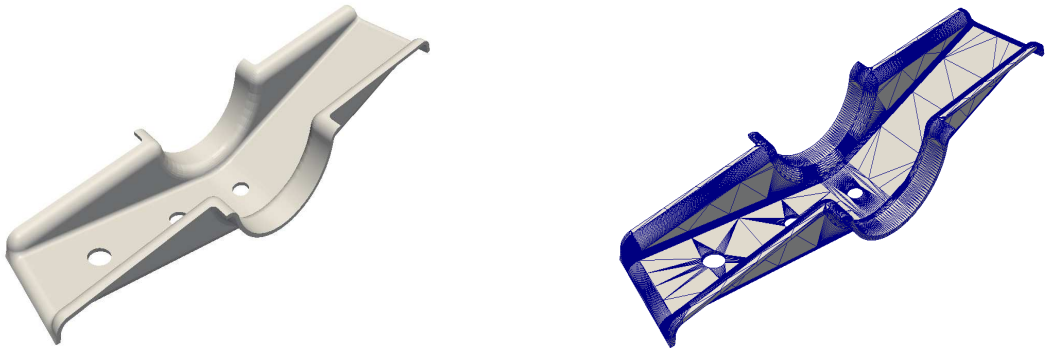

Figura 4.2: Mesh 2: numero vertici: 17054, numero face: 34116

Figura 4.3: Mesh 3: numero vertici: 65660, numero face: 131580

4.3 Test

In ogni test inserito vengono mostrati:

- tempo di esecuzione;
- quantitá assoluta di distanze calcolate e di valori indefiniti e percentuale di entrambe le grandezze rispetto al numero totale di vertici della mesh;
- distanza minima e distanza massima rilevate;

- intervallo piú frequente: range di valori dell'intervallo piú frequente tra i 101 intervalli creati.

Di seguito ai dati vengono inserite due immagini:

1. istogramma che illustra le frequenze per tutti i 101 intervalli creati;

2. immagine che raffigura con colorazione variabile lo spessore calcolato nelle varie parti della mesh. I colori possono assumere infinite gradazioni variabili tra:

   - blu: nessuno spessore calcolato;
   - rosso: spessore di valore importante.

Le immagini base in cui non é stata calcolata alcuna intersezione valida sono le seguenti:

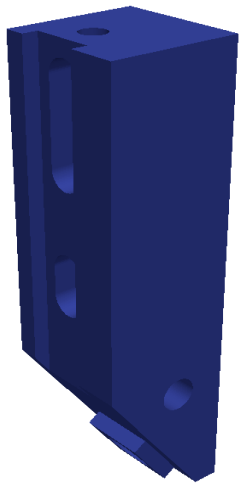

Figura 4.4: Mesh 1 con colorazione senza calcolo spessore

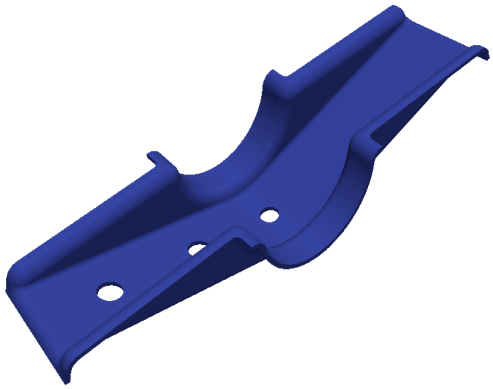

Figura 4.5: Mesh 2 con colorazione senza calcolo spessore

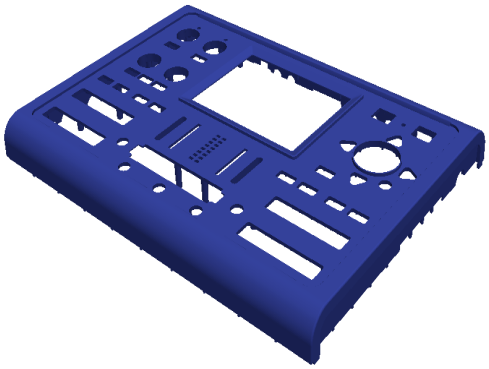

Figura 4.6: Mesh 3 con colorazione senza calcolo spessore

4.3.1 Test iniziali

In questa parte vengono presentati i risultati dei test eseguiti sulla prima versione dell'algoritmo implementata.

Mesh1:

Tempo di esecuzione: 0m, 29s, 850ms

474 distanze (0 zero) - 641 undefined → 42% - 58%

Distanza minore: 1,11224

Distanza maggiore: 76,38637

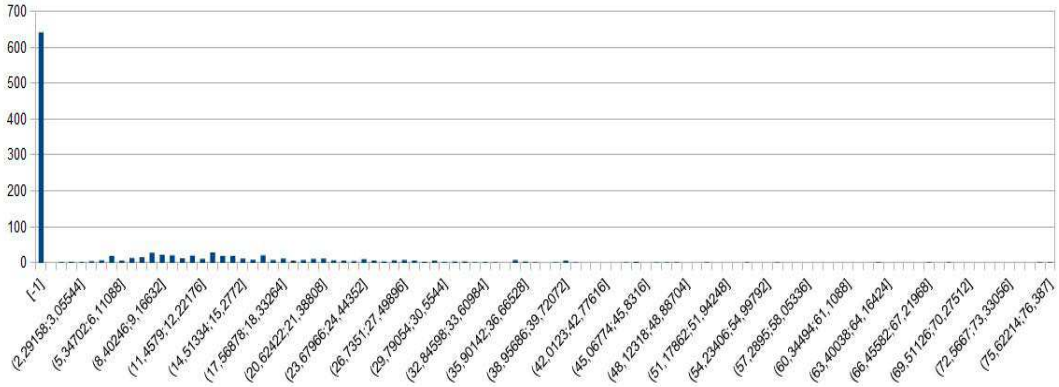

Figura 4.7: Istogramma frequenza valori di distanza mesh 1: intervallo piú frequente: [-1], frequenza: 641

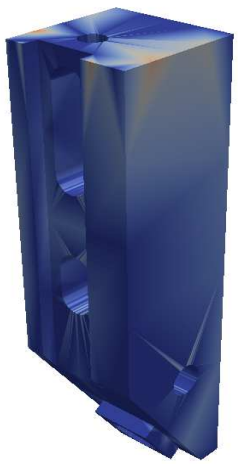

Figura 4.8: Colorazione spessore per mesh 1

Mesh2:

Tempo di esecuzione: 10m, 32s, 422ms

776 distanze (0 zero) - 16278 undefined → 4% - 96%

Distanza minore: 1,11052

Distanza maggiore: 102,00722

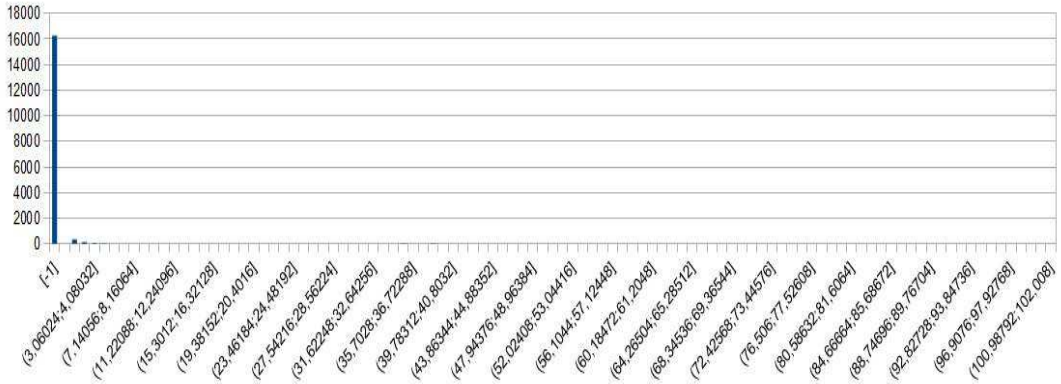

Figura 4.9: Istogramma frequenza valori di distanza mesh 2: intervallo piú frequente: [-1], frequenza: 16278

Figura 4.10: Colorazione spessore per mesh 2

Mesh3:

Tempo di esecuzione: 53m, 10s, 51ms

21790 distanze (0 zero) - 43870 undefined → 33% - 67%

Distanza minore: 0,08833

Distanza maggiore: 396,52818

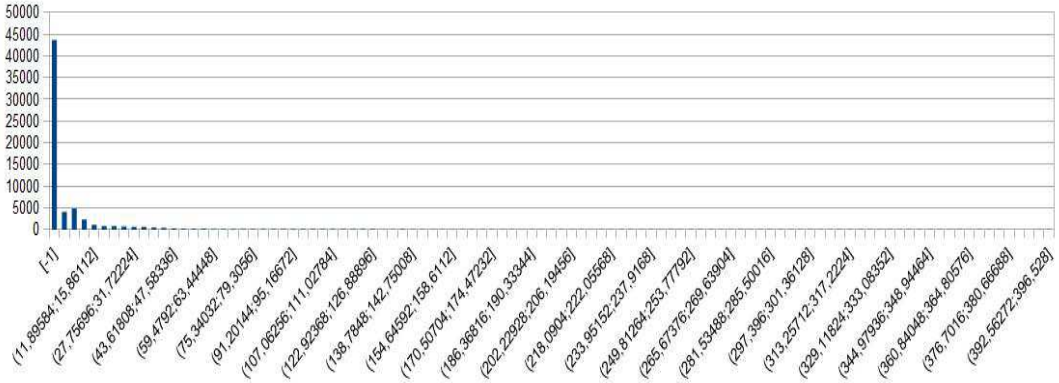

Figura 4.11: Istogramma frequenza valori di distanza mesh 3: intervallo piú frequente: [-1], frequenza: 43870

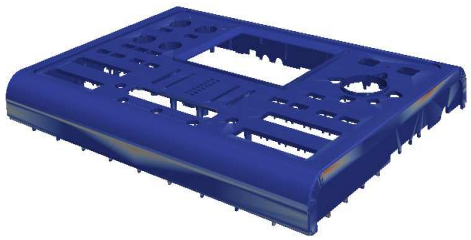

Figura 4.12: Colorazione spessore per mesh 3

Come si puó vedere, la strutturazione delle mesh utilizzate per i test rende poco preciso l'algoritmo. Infatti nelle tre mesh utilizzate l'intervallo di valori piú frequente é sempre il valore -1, corrispondente all'indicazione di nessuna distanza trovata per i relativi vertici.
Il tempo di esecuzione chiaramente dipende dalle dimensioni della mesh, intese come numero di vertici e di face. Mentre la prima mesh ha un tempo di esecuzione accettabile (circa 30 secondi) le altre esigono una quantitá di tempo per eseguire assolutamente troppo elevata per essere utilizzate in un programma di analisi delle caratteristiche di modelli 3D.

4.3.2 Test riduzione numero raggi

Il problema del tempo di esecuzione é stato analizzato per primo. É' stato deciso di provare a diminuire il numero di raggi lanciati in ogni cono in modo da ridurre notevolmente il numero di iterazioni di ricerca intersezioni eseguite dall'algoritmo, sperando di non perdere in precisione in modo significativo.

4.3.2.1 Test utilizzando 10 raggi

Mesh1:
Tempo di esecuzione: 0m, 2s, 355ms
463 distanze (0 zero) - 652 undefined → 41% - 59%
Distanza minore: 1,11229
Distanza maggiore: 76,39103

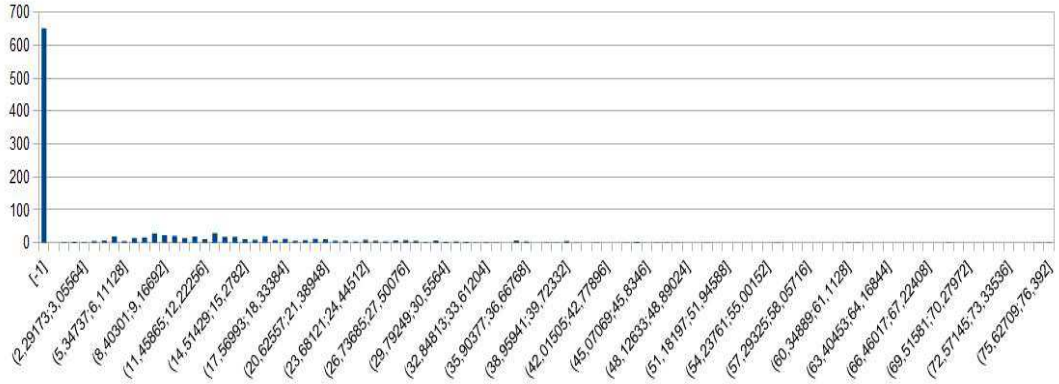

Figura 4.13: Istogramma frequenza valori di distanza mesh 1 con 10 raggi: intervallo piú frequente: [-1], frequenza: 652

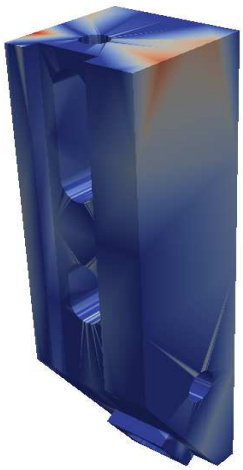

Figura 4.14: Colorazione spessore per mesh 1 con 10 raggi

Mesh2:

Tempo di esecuzione: 1m, 5s, 582ms

736 distanze (0 zero) - 16318 undefined → 4% - 96%

Distanza minore: 1,11080

Distanza maggiore: 102,02367

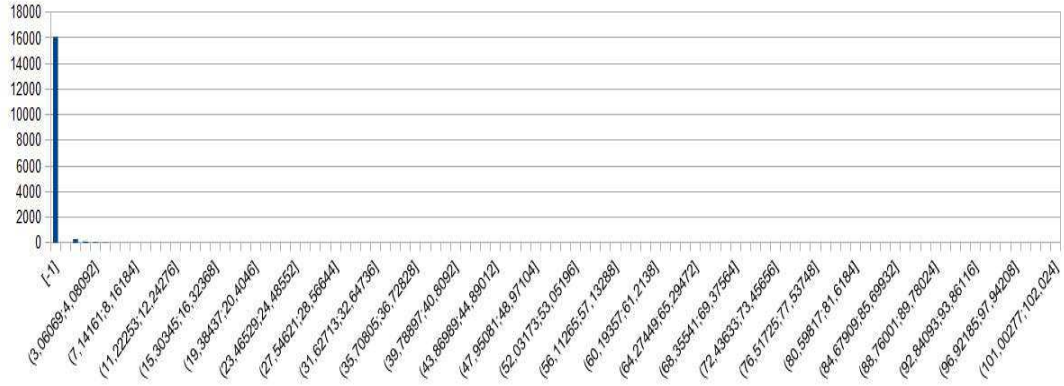

Figura 4.15: Istogramma frequenza valori di distanza mesh 2 con 10 raggi: intervallo piú frequente: [-1], frequenza: 16318

Figura 4.16: Colorazione spessore per mesh 2 con 10 raggi

Mesh3:

Tempo di esecuzione: 5m, 9s, 3ms

20926 distanze (0 zero) - 44734 undefined → 31% - 69%

Distanza minore: 0,08838

Distanza maggiore: 396,51962

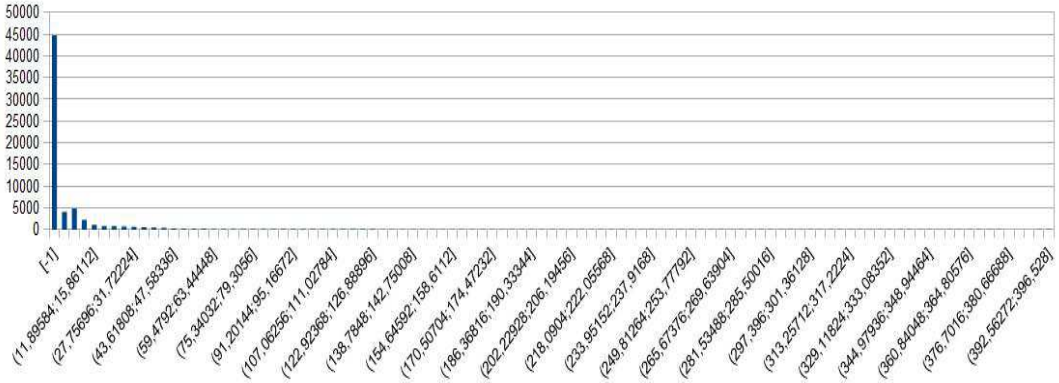

Figura 4.17: Istogramma frequenza valori di distanza mesh 3 con 10 raggi: intervallo piú frequente: [-1], frequenza: 44734

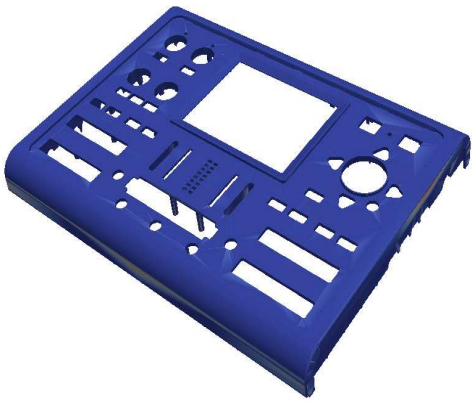

Figura 4.18: Colorazione spessore per mesh 3 con 10 raggi

4.3.2.2 Test utilizzando 1 raggio

Mesh1:

Tempo di esecuzione: 0m, 0s, 436ms

451 distanze (0 zero) - 664 undefined → 40% - 60%

Distanza minore: 1,11225

Distanza maggiore: 75,18313

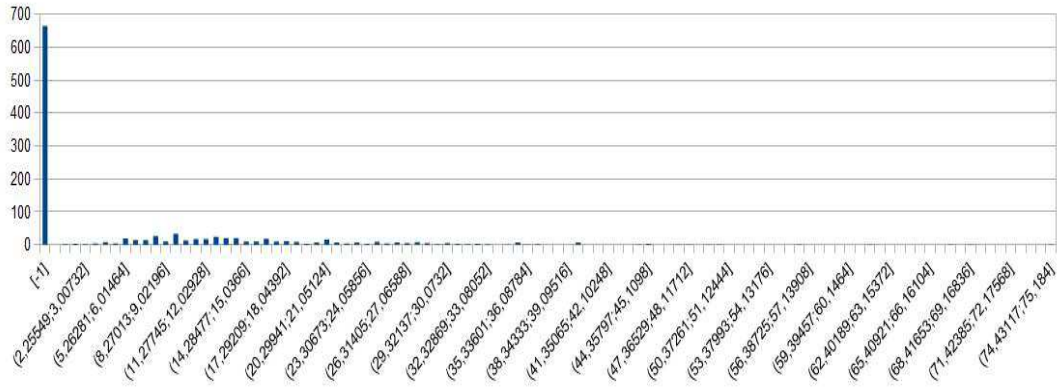

Figura 4.19: Istogramma frequenza valori di distanza mesh 1 con 1 raggio: intervallo piú frequente: [-1], frequenza: 664

Figura 4.20: Colorazione spessore per mesh 1 con 1 raggio

Mesh2:

Tempo di esecuzione: 0m, 9s, 128ms

668 distanze (0 zero) - 16386 undefined → 3% - 97%

Distanza minore: 1,11046

Distanza maggiore: 101,96346

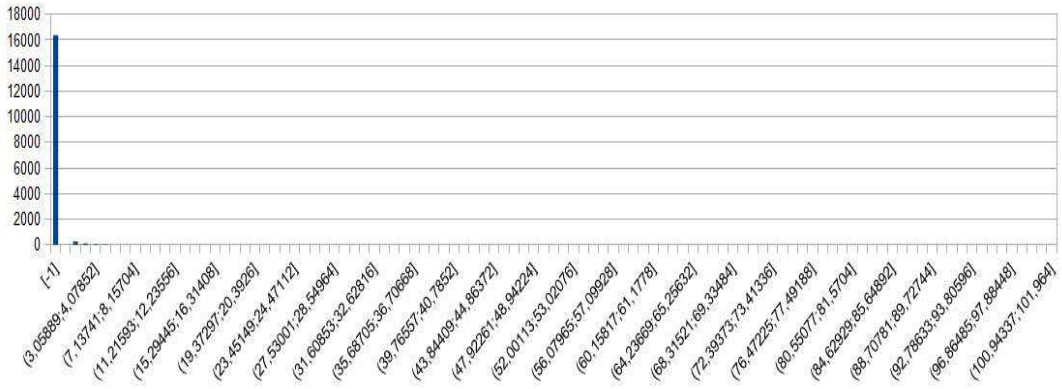

Figura 4.21: Istogramma frequenza valori di distanza mesh 2 con 1 raggio: intervallo piú frequente: [-1], frequenza: 16386

Figura 4.22: Colorazione spessore per mesh 2 con 1 raggio

Mesh3:

Tempo di esecuzione: 0m, 39s, 343ms

19522 distanze (0 zero) - 46138 undefined → 29% - 71%

Distanza minore: 0,088329
Distanza maggiore: 396,53570

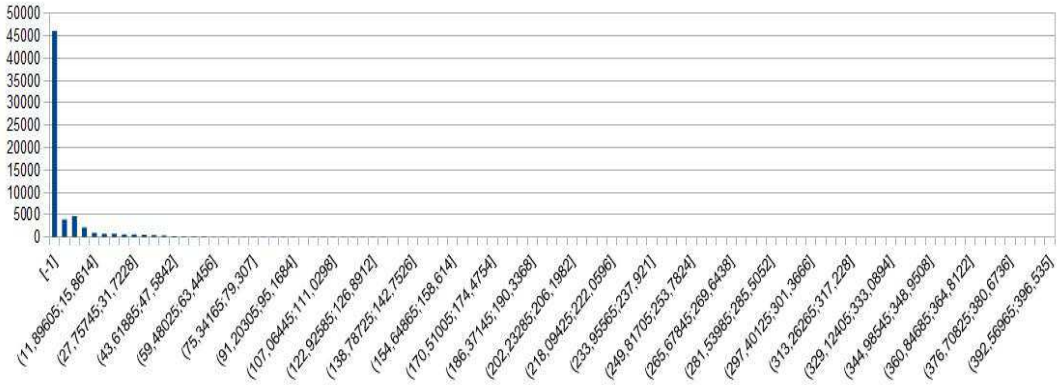

Figura 4.23: Istogramma frequenza valori di distanza mesh 3 con 1 raggio: intervallo piú frequente: [-1], frequenza: 46138

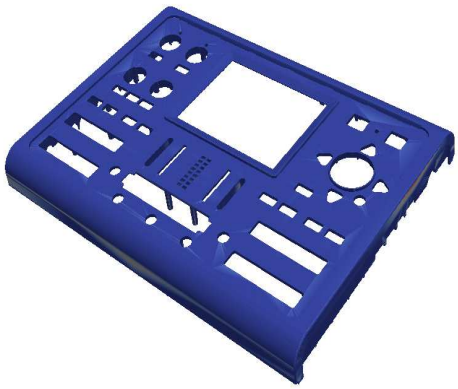

Figura 4.24: Colorazione spessore per mesh 3 con 1 raggio

Questi risultati dimostrano che é stato ottenuto l'obiettivo prefissato per questa modifica, cioé:

- per la prima mesh: utilizzando 100 raggi l'algoritmo impiega 30 secondi ed ottiene distanze per il 42% dei vertici, per 10 raggi il tempo impiegato é di 2 secondi ottenendo il 41% di distanze, mentre per 1 raggio meno di 1 secondo ed il 40% di risultati;

- per la seconda: il calcolo con 100 raggi necessita di 10 minuti ed ottiene il 4% dei risultati, per 10 raggi 1 minuto ed il 4%, mentre per 1 raggio 10 secondi ed il 3% di intersezioni calcolate;

- infine per la terza: 53 minuti col 33% di risultati per 100 raggi, 5 minuti ed il 31% dei risultati con 10 raggi ed infine 40 secondi col 29% per 1 raggio

É evidente il netto miglioramento dal punto di vista del tempo di esecuzione dell'algoritmo con calo della percentuale di risultati corretti che non subisce importanti variazioni.

I test successivi quindi, dove non specificato, vengono eseguiti utilizzando un solo raggio.

4.3.3 Test ampliamento lista di link

I test seguenti sono relativi alla piú importante modifica dell'implementazione dal punto di vista della precisione dell'algoritmo cioé la variazione riguardante il riempimento della griglia.

Mesh1:
Tempo di esecuzione: 0m, 1s, 671ms
1114 distanze (0 zero) - 1 undefined → 99% - 1%
Distanza minore: 0,93311
Distanza maggiore: 67,21366

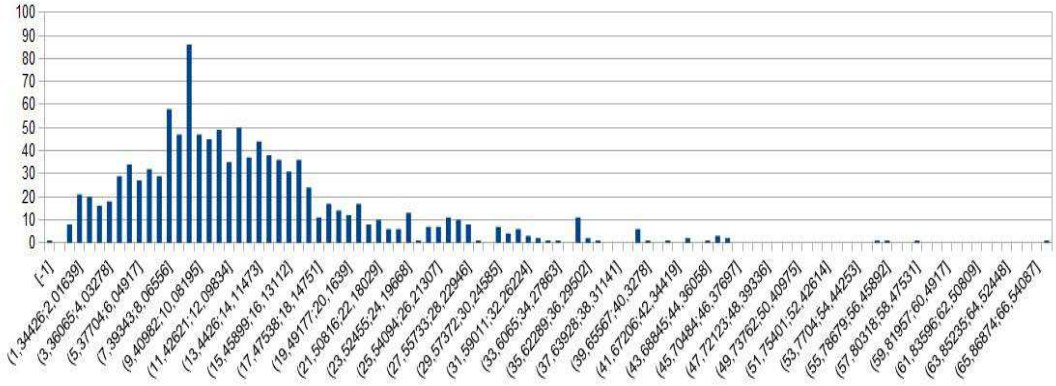

Figura 4.25: Istogramma frequenza valori di distanza mesh 1 inserendo tutti i link: intervallo piú frequente: [8,73777 → 9,40991], frequenza: 86

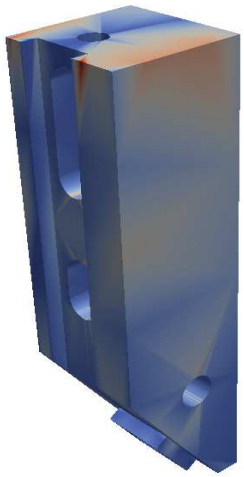

Figura 4.26: Colorazione spessore per mesh 1 inserendo tutti i link

Mesh2:

Tempo di esecuzione: 2m, 2s, 394ms

17050 distanze (0 zero) - 4 undefined → 99% - 1%

Distanza minore: 0,05302

Distanza maggiore: 11,86257

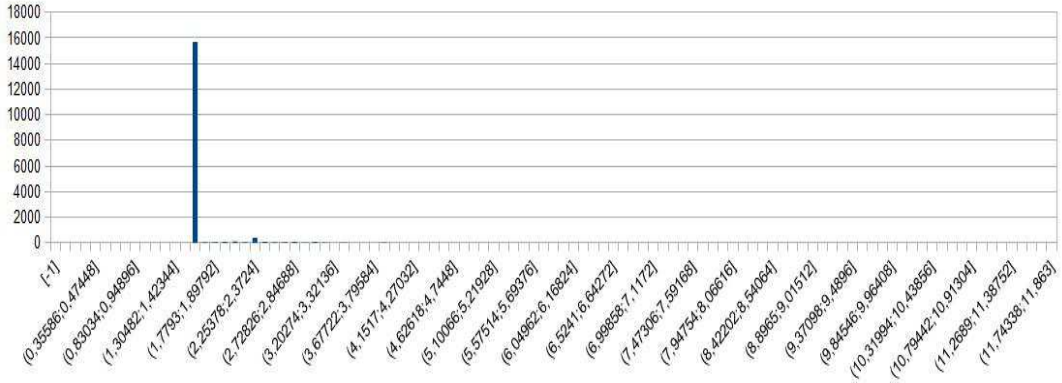

Figura 4.27: Istogramma frequenza valori di distanza mesh 2 inserendo tutti i link: intervallo piú frequente: [1,54213 → 1,66076], frequenza: 15688

Figura 4.28: Colorazione spessore per mesh 2 inserendo tutti i link

Mesh3:

Tempo di esecuzione: 4m, 16s, 150ms

65646 distanze (0 zero) - 14 undefined → 99% - 1%

Distanza minore: 0,00055

Distanza maggiore: 217,34981

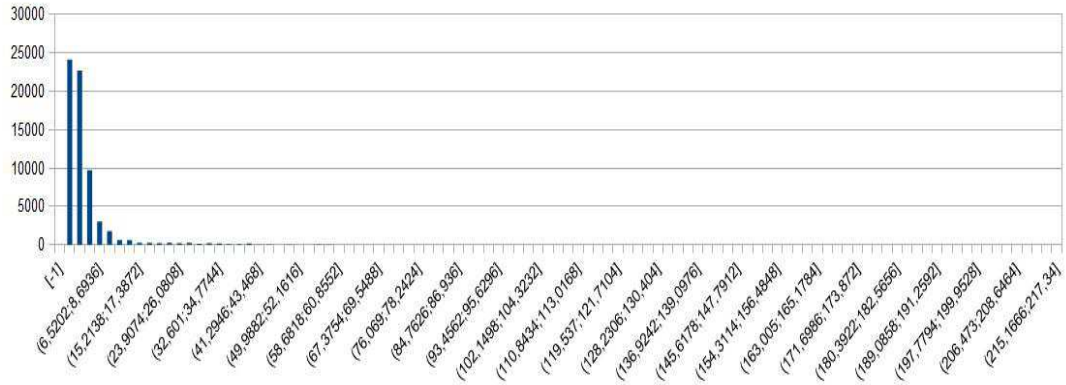

Figura 4.29: Istogramma frequenza valori di distanza mesh 3 inserendo tutti i link: intervallo piú frequente: $[0 \rightarrow 2{,}17349]$, frequenza: 24140

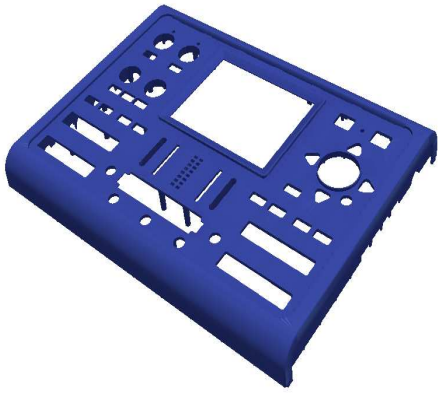

Figura 4.30: Colorazione spessore per mesh 3 inserendo tutti i link

Dai risultati dei test eseguiti é facilmente visibile il miglioramento di precisione dell'algoritmo. Nei test precedenti la percentuale piú alta di distanze calcolate si assestava a circa il 40% e la minore intorno al 3%, con questa modifica é stato possibile raggiungere per tutte le mesh una percentuale del 99%. Questo é dovuto al fatto che inserendo in ogni elemento tutte le face

giacenti nel corrispondente voxel, durante la sua esecuzione l'algoritmo controlla le intersezioni con tutte le face dei voxel in cui si sposta di volta in volta senza tralasciarne alcuna.
Questa modifica si ripercuote peró sul tempo di esecuzione, infatti ogni mesh ora richiede molto piú tempo per eseguire rispetto al test precedente. Questo aumento del tempo di esecuzione peró é ben bilanciato dall'importantissimo aumento di precisione sopra discusso.
I risultati di questo test risultano quindi assolutamente soddisfacenti.

4.3.4 Test su origine raggio coincidente col vertice

I test di questo paragrafo sono relativi all'eliminazione dello scostamento nell'impostazione del punto di origine del raggio che ha portato a far coincidere tale punto esattamente con il vertice. Lo scostamento, pur evitando di trovare intersezioni non corrette con le face intorno al vertice, puó spesso portare a calcolare distanze non precisissime dato che avvicina il punto di partenza del raggio al punto intersezione. Per evitare che l'algoritmo consideri come corretta l'intersezione con una face intorno al vertice é stata inserito un controllo che si assicuri che nei tre vertici della face non sia presente il vertice origine del raggio, altrimenti non viene eseguito il test di intersezione per la specifica face.
Per questo test vengono inseriti soltanto i dati riassuntivi per ciascuna mesh senza inserire i relativi istogrammi e le immagini per visualizzare gli spessori in quanto le differenze rispetto alla configurazione precedente sono minime e sono rilevabili dalla distanza minima e massima e dai valori dell'intervallo piú frequente.

Mesh1:
Tempo di esecuzione: 0m, 1s, 900ms
1114 distanze (0 zero) - 1 undefined → 99% - 1%
Distanza minore: 1,03401
Distanza maggiore: 67,31456

Intervallo piú frequente: [8,75089 → 9,42403], frequenza: 81

Mesh2:
Tempo di esecuzione: 2m, 3s, 727ms
17054 distanze (0 zero) - 0 undefined → 100% - 0%
Distanza minore: 0,07907
Distanza maggiore: 12,09290
Intervallo piú frequente: [1,69300 → 1,81393], frequenza: 15369

Mesh3:
Tempo di esecuzione: 4m, 37s, 872mss
65648 distanze (0 zero) - 12 undefined → 99% - 1%
Distanza minore: 0,00165
Distanza maggiore: 131,25747
Intervallo piú frequente: [1,31257 → 2,62514], frequenza: 17344

4.3.5 Confronti finali

Nei test relativi alla prima modifica effettuata alla versione iniziale é stato diminuito il numero di raggi creati in ognuno dei coni dato che inizialmente l'utilizzo di 100 raggi rendeva molto inefficiente l'algoritmo avendo un tempo di esecuzione altissimo sopratutto per la seconda e la terza mesh.
Nei test seguenti sono state poi validate varie modifiche strutturali che hanno reso l'algoritmo molto piú preciso e consistente.
Tra l'ultimo test eseguito e quello che segue sono state apportate altre piccole modifiche che hanno reso l'algoritmo piú veloce.
In questo paragrafo si vuole confrontare la precisione e qualitá dei risultati ottenuti dall'algoritmo nella sua versione finale per diverse quantitá di raggi, cioé 1, 5, 10 e 100 raggi. Questo viene fatto per dimostrare che le modifiche apportate all'algoritmo validate nei test precedenti hanno migliorato l'algoritmo in modo tale da rendere le differenze che si creano aumentando

il numero di raggi quasi impercettibili. Lo scopo di questi test é la conferma che é possibile raggiungere ottimi risultati mantenendo basso il numero di raggi quindi il tempo di esecuzione.

4.3.5.1 Test utilizzando 100 raggi

Mesh1:

Tempo di esecuzione: 1m, 23s, 777ms
1114 distanze (0 zero) - 1 undefined → 99% - 1%
Distanza minima: 1,03397
Distanza massima: 67,32261

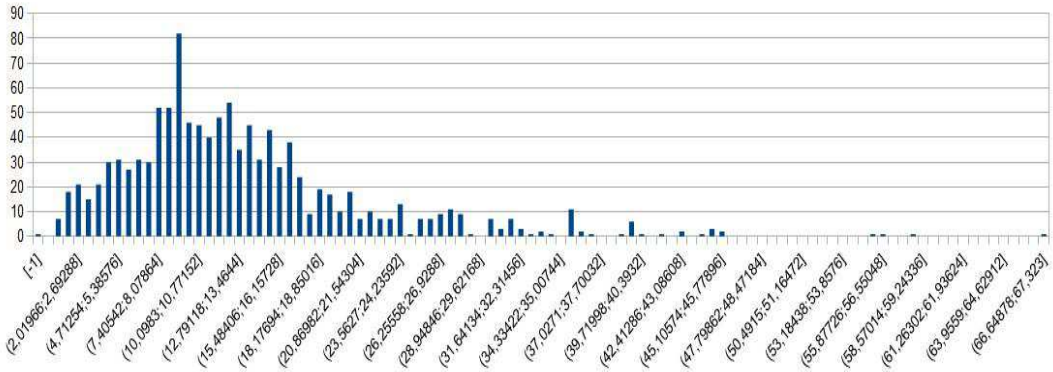

Figura 4.31: Istogramma frequenza valori di distanza mesh 1 con 100 raggi: intervallo piú frequente: [8,75193 → 9,42516], frequenza: 82

Mesh2:

Tempo di esecuzione: 2h, 14m, 53s, 355ms
17054 distanze (0 zero) - 0 undefined → 100% - 0%
Distanza minima: 0,07905
Distanza massima: 12,09451

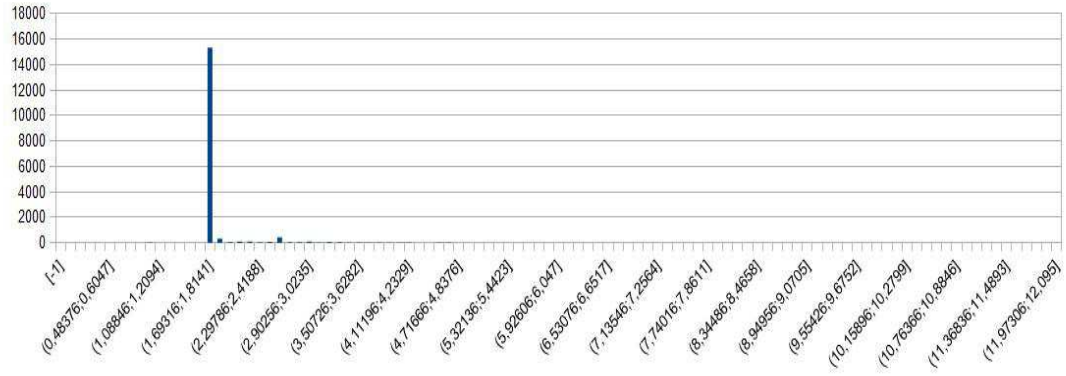

Figura 4.32: Istogramma frequenza valori di distanza mesh 2 con 100 raggi: intervallo piú frequente: [1,69323 → 1,81417], frequenza: 15390

Mesh3:

Tempo di esecuzione: 5h, 26m, 12s, 345ms

65648 distanze (0 zero) - 12 undefined → 99% - 1%

Distanza minima: 0,00165

Distanza massima: 131,25752

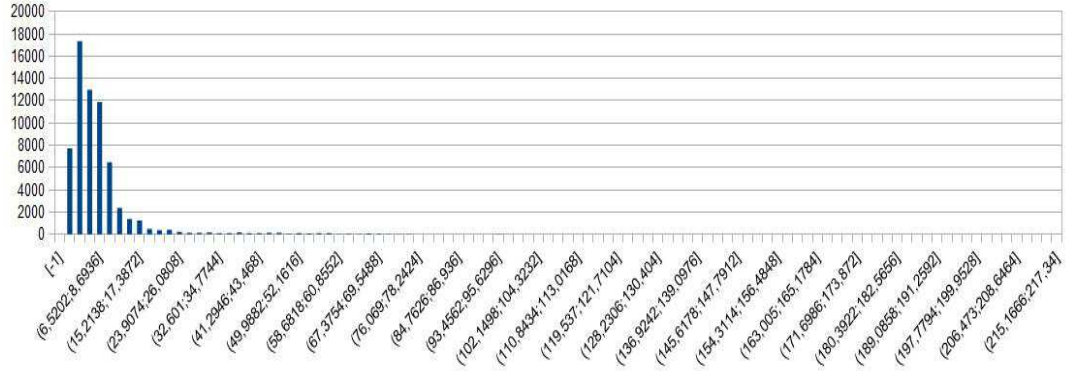

Figura 4.33: Istogramma frequenza valori di distanza mesh 3 con 100 raggi: intervallo piú frequente: [1,31257 → 2,62515], frequenza: 17348

4.3.5.2 Test utilizzando 10 raggi

Mesh1:

Tempo di esecuzione: 0m, 9s, 874ms

1114 distanze (0 zero) - 1 undefined → 99% - 1%

Distanza minima: 1,03390

Distanza massima: 67,34657

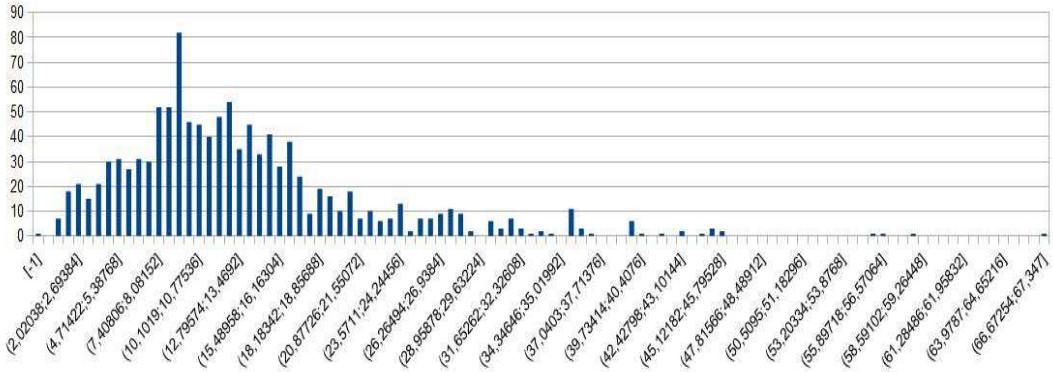

Figura 4.34: Istogramma frequenza valori di distanza mesh 1 con 10 raggi: intervallo piú frequente: [8,75505 → 9,42852], frequenza: 82

Mesh2:

Tempo di esecuzione: 12m, 11s, 233ms

17054 distanze (0 zero) - 0 undefined → 100% - 0%

Distanza minima: 0,07905

Distanza massima: 12,09729

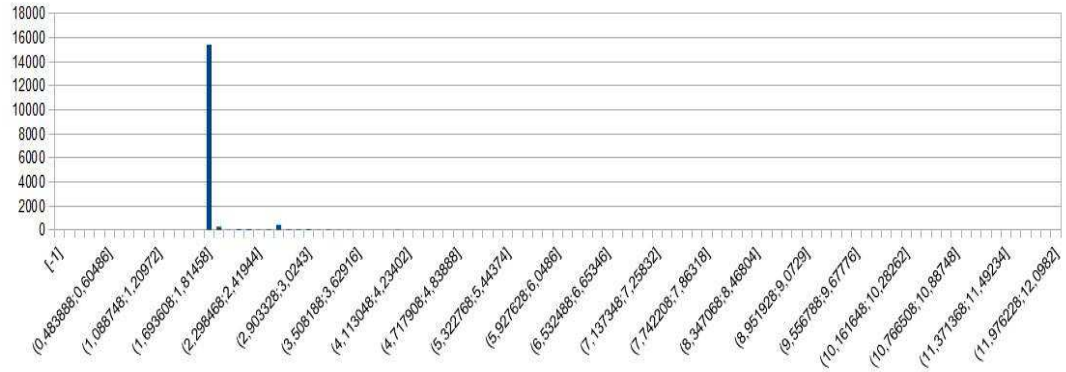

Figura 4.35: Istogramma frequenza valori di distanza mesh 2 con 10 raggi: intervallo piú frequente: [1,69362 → 1,81459], frequenza: 15430

Mesh3:

Tempo di esecuzione: 24m, 27s, 608ms

65648 distanze (0 zero) - 12 undefined → 99% - 1%

Distanza minima: 0,00165

Distanza massima: 131,25953

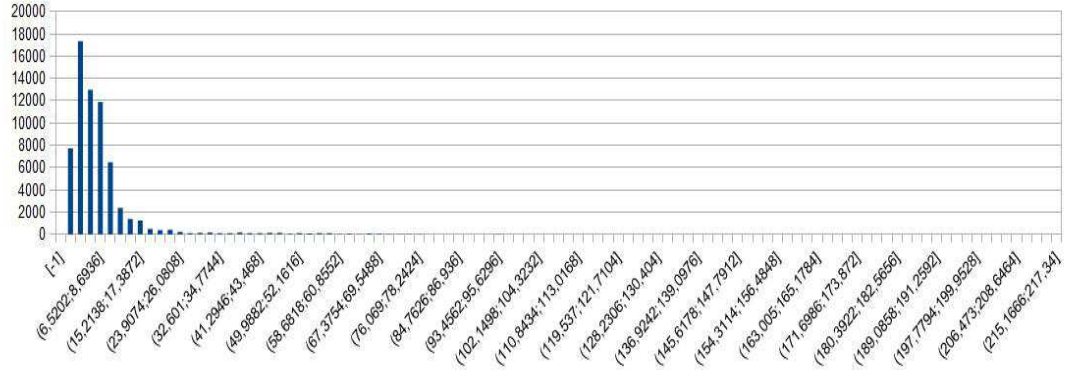

Figura 4.36: Istogramma frequenza valori di distanza mesh 3 con 10 raggi: intervallo piú frequente: [1,31259 → 2,62519], frequenza: 17343

4.3.5.3 Test utilizzando 5 raggi

Mesh1:

Tempo di esecuzione: 0m, 4s, 568ms

1114 distanze (0 zero) - 1 undefined → 99% - 1%

Distanza minima: 1,03398

Distanza massima: 67,29823

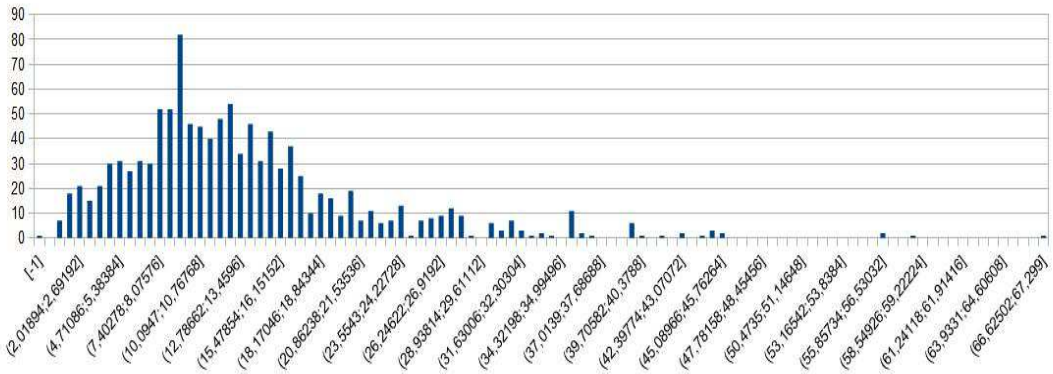

Figura 4.37: Istogramma frequenza valori di distanza mesh 1 con 5 raggi: intervallo piú frequente: [8,74877 → 9,42175], frequenza: 82

Mesh2:

Tempo di esecuzione: 5m, 56s, 633ms

17054 distanze (0 zero) - 0 undefined → 100% - 0%

Distanza minima: 0,07903

Distanza massima: 12,09482

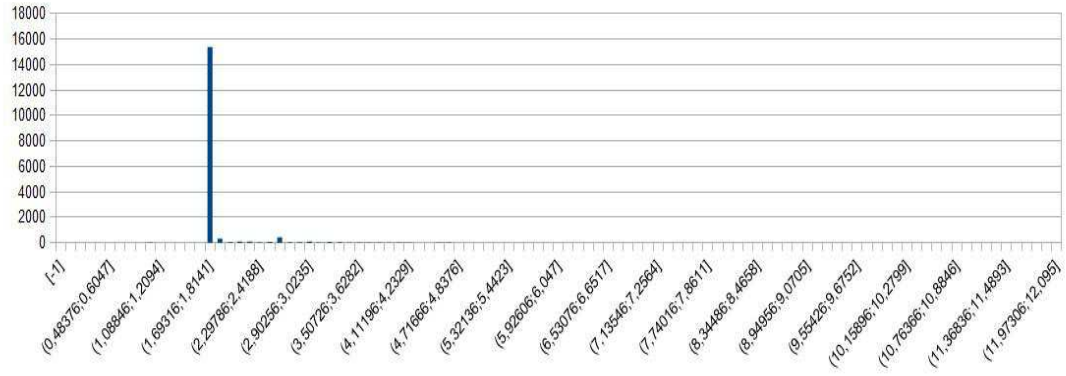

Figura 4.38: Istogramma frequenza valori di distanza mesh 2 con 5 raggi: intervallo piú frequente: [1,69327 → 1,81422], frequenza: 15392

Mesh3:

Tempo di esecuzione: 11m, 28s, 616ms

65648 distanze (0 zero) - 12 undefined → 99% - 1%

Distanza minima: 0,00165

Distanza massima: 131,25911

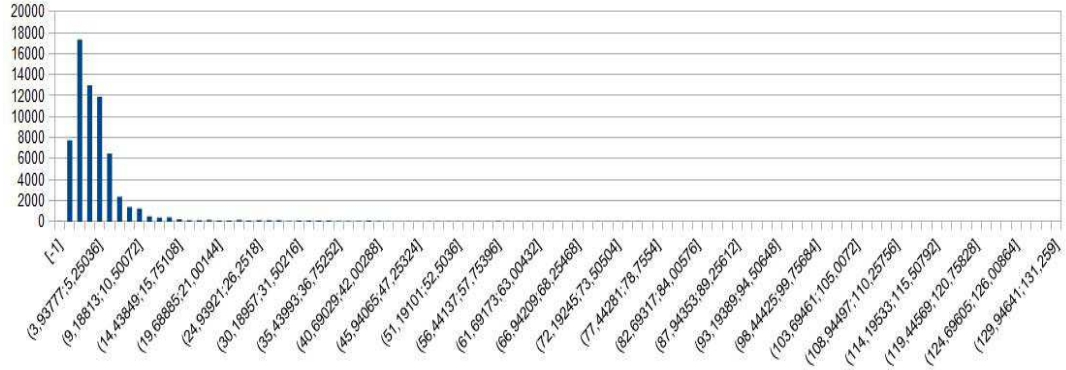

Figura 4.39: Istogramma frequenza valori di distanza mesh 3 con 5 raggi: intervallo piú frequente: [1,31259 → 2,62518], frequenza: 17347

4.3.5.4 Test utilizzando 1 raggio

Mesh1:

Tempo di esecuzione: 0m, 0s, 818ms

1114 distanze (0 zero) - 1 undefined → 99% - 1%

Distanza minore: 1,034010

Distanza maggiore: 67,31456

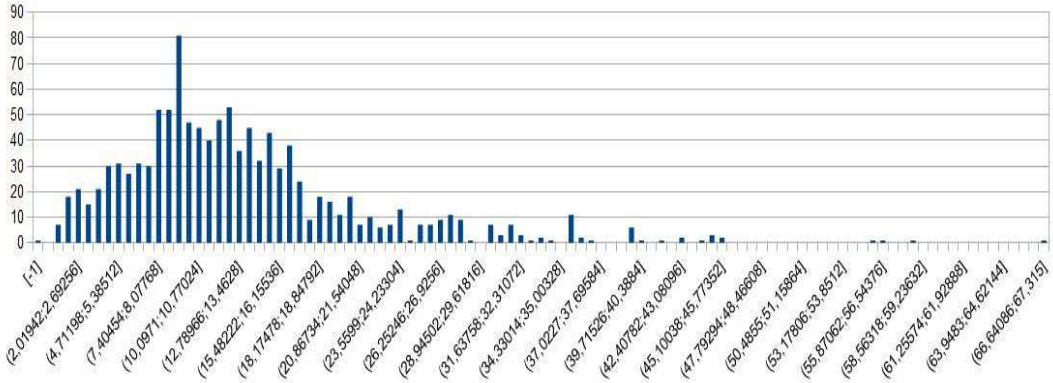

Figura 4.40: Istogramma frequenza valori di distanza mesh 1 con 1 raggio: intervallo piú frequente: [8,75089 → 9,42403], frequenza: 81

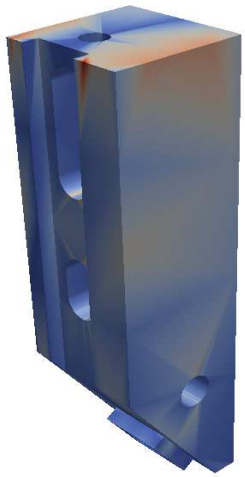

Figura 4.41: Colorazione spessore per mesh 1

Mesh2:

Tempo di esecuzione: 1m, 15s, 962ms

17054 distanze (0 zero) - 0 undefined → 100% - 0%

Distanza minore: 0,07907

Distanza maggiore: 12,09290

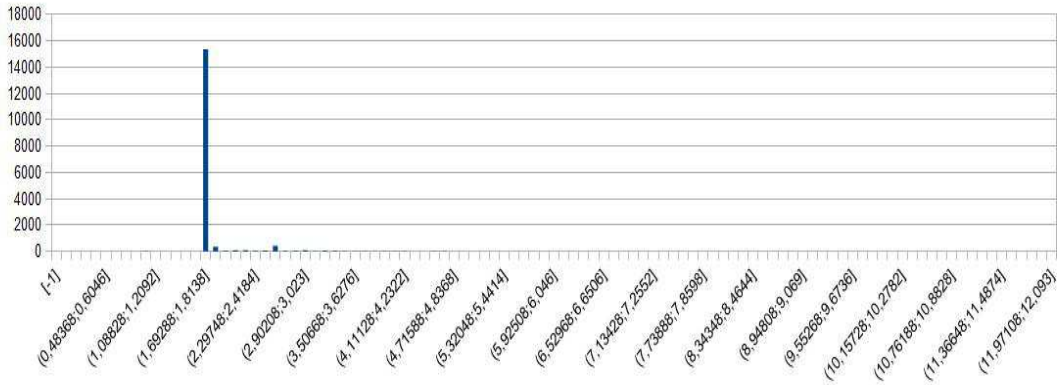

Figura 4.42: Istogramma frequenza valori di distanza mesh 2 con 1 raggio: intervallo piú frequente: [1,69300 → 1,81393], frequenza: 15369

Figura 4.43: Colorazione spessore per mesh 2

Mesh3:

Tempo di esecuzione: 2m, 39s, 794ms

65648 distanze (0 zero) - 12 undefined → 99% - 1%

Distanza minima: 0,00165

Distanza massima: 131,25747

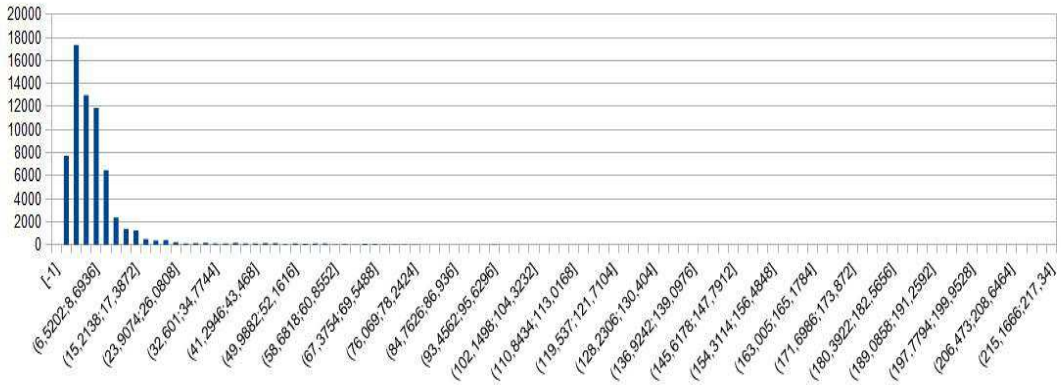

Figura 4.44: Istogramma frequenza valori di distanza mesh 3 con 1 raggio: intervallo piú frequente: [1,31257 → 2,62514], frequenza: 17344

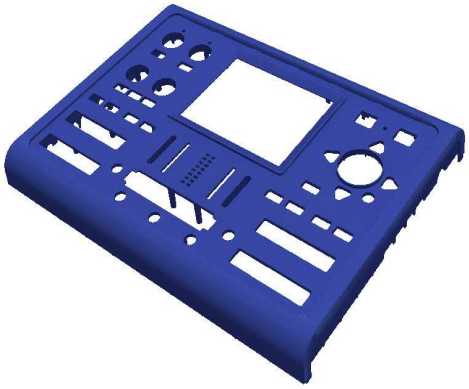

Figura 4.45: Colorazione spessore per mesh 3

Si é deciso di inserire le immagini relative alla colorazione degli spessori soltanto per la versione ad un solo raggio in quanto dai dati é chiaro che le differenze tra le versioni sono veramente minime e non apprezzabile alla vista.
I dati appena presentati dimostrano che l'algoritmo non differisce nei risultati in modo importante dal numero di raggi, é quindi importante mantenere il numero di raggi molto basso per mantenere un'alta velocitá di esecuzione dell'algoritmo.

4.4 Esempi di intersezioni

Di seguito vengono inserite alcune immagini che raffigurano esempi di intersezioni tra raggi e face. In tali immagini in blu é disegnata la mesh di base, in rosso le face che stanno intorno al vertice da cui parte il raggio, in grigio la face intersecata ed ovviamente la linea visibile rappresenta il raggio.

4.4.1 Intersezioni valide

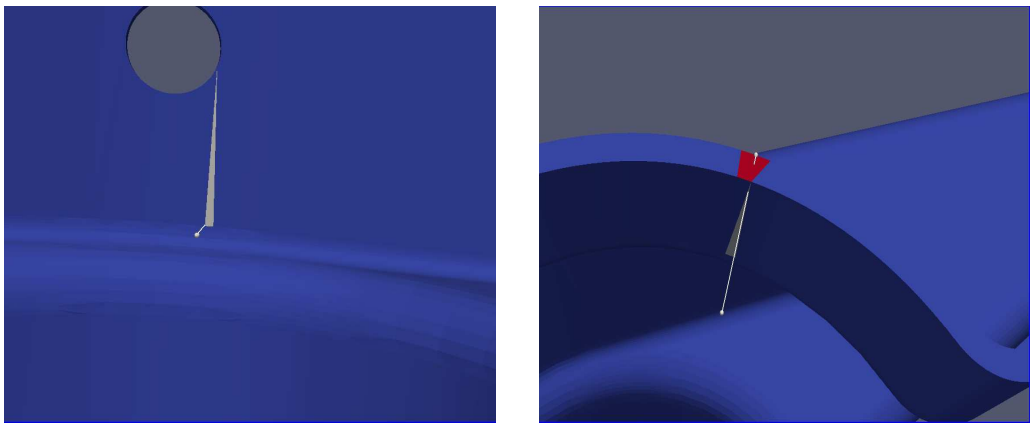

Figura 4.46: Esempi di intersezioni valide

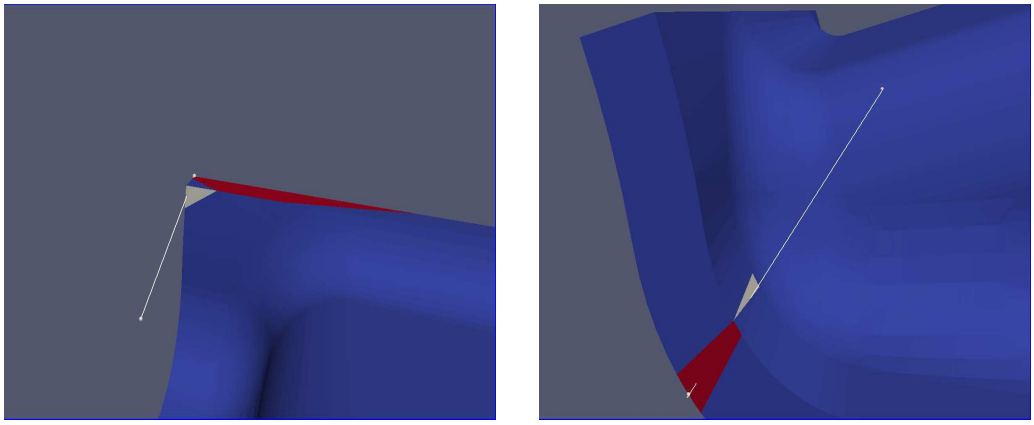

Figura 4.47: Altri esempi di intersezioni valide

Dalle immagini si nota che questi raggi presi come esempio intersecano la face corretta lungo la loro direzione.

4.4.2 Intersezioni indefinite

Questa immagine mostra il caso di una distanza non trovata cioé del calcolo di un valore indefinito.

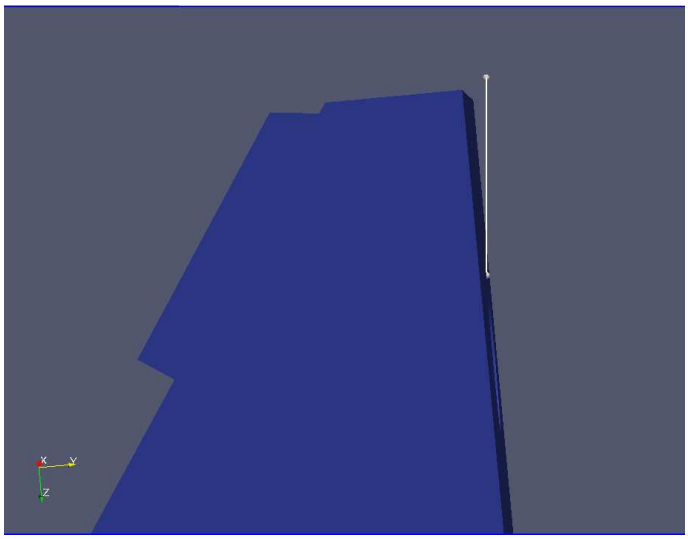

Figura 4.48: Esempio di intersezione indefinita

Dall'immagine si vede immediatamente che in effetti il raggio non interseca alcuna face, quindi in questo caso la distanza é giustamente indefinita.

Conclusioni

Il calcolo del costo di produzione di un oggetto modellato mediante il processo di stampaggio é un processo lungo e regolato da tante variabili. Probabilmente la principale é il dimensionamento dell'oggetto stesso in modo da calcolare il costo di produzione di uno stampo che abbia le caratteristiche geometriche richieste dal componente ed il costo di produzione di ogni singola unitá del componente stesso. Per rendere la stima dei costi il piú possibile precisa é quindi necessario dimensionare adeguatamente in ogni sua parte l'oggetto che si vuole produrre. In letteratura sono stati sviluppati molteplici algoritmi per l'analisi delle dimensioni di componenti. Tra essi l'algoritmo Shape Diameter Function permette di calcolare lo spessore dell'oggetto calcolando lo spessore del modello 3D dell'oggetto stesso in molteplici punti mediante ray casting. In questa tesi é stata implementata una versione di questo algoritmo, modificandolo ove necessario o possibile al fine di renderlo piú efficiente e preciso possibile per una sua implementazione in un programma di gestione di modelli 3D di oggetti industriali. In particolare il focus é stato posto sulla precisione e localitá delle distanze calcolate dall'algoritmo in quanto i modelli principalmente sono costituiti da oggetti in plastica che, per costruzione, sono molto sottili ed hanno rapidi e locali cambiamenti di forma.

L'obiettivo posto é stato raggiunto in quanto l'algoritmo calcola distanze per un'elevata percentuale di vertici restituendo fallimento di intersezione nei casi in cui il raggio in effetti non interseca alcuna parte della mesh. La precisione dell'algoritmo é stata aumentata sia come percentuale di intersezioni calco-

late che come accuratezza delle distanze restituite. É stato anche possibile abbassare notevolmente il tempo di esecuzione limitando il piú possibile il numero di raggi lanciati da ogni cono, fino ad un limite di un raggio soltanto per ogni cono. Ovviamente questa modifica ha leggermente abbassato la precisione complessiva dell'algoritmo, quindi probabilmente il migliore trade-off tra precisione e tempo di esecuzione é rappresentato da un range compreso approssimativamente tra 2 e 5 raggi per ogni cono.
Questa implementazione rappresenta un ottima base per il calcolo dello spessore generale e locale di qualsiasi oggetto ben riprodotto dal suo modello 3D. Il principale miglioramento possibile é l'ulteriore perfezionamento della precisione dell'algoritmo inserendo una funzione di correzione dell'errore che modifichi il valore di distanza calcolato basandosi sull'angolo che si crea tra il raggio e la normale della face intersecata. Questa modifica non sarebbe banale trattandosi di modelli in tre dimensioni ma consentirebbe di ottenere valori ancora piú precisi senza dover considerare un errore sul calcolo a causa dell'angolo con cui il raggio interseca la face.

Bibliografia

[1] J.Tierny, J.P.Vandeborre, M.Daoudi; *3d Mesh Skeleton Extraction Using Topological and Geometrical Analyses*, 2006. In: 14th Pacific Conference on Computer Graphics and Applications (Pacific Graphics 2006), pp. 85-94. Taipei, Taiwan

[2] J.Tierny, J.P.Vandeborre, M.Daoudi; *Topology driven 3D mesh hierarchical segmentation*, 2007. In: IEEE International Conference on Shape Modeling and Applications (SMI 2007). IEEE, Lyon

[3] S.Katz, G.Leifman, A.Tal: *Mesh segmentation using feature point and core extraction*, 2005. Visual Comput. (Pacific Graphics) 21(8-10), 865-875

[4] J.M.Lien, N.M.Amato: *Simultaneous shape decomposition and skeletonization*, 2005. Tech. rep., Texas AM University

[5] J.M.Lien, N.M.Amato: *Approximate convex decomposition of polygons*, 2004. In: SCG 2004: Proceedings of the twentieth annual symposium on Computational geometry, pp. 17-26. ACM Press, New York, NY, USA

[6] T.Dey, J.Giesen, S.Goswami: *Shape segmentation and matching with flow discretization*, 2003. In: Proceedings of the Workshop on Algorithms and Data Structures (WADS 2003). Lect. Notes Comput. Sci., vol. 2748, pp. 25-36. Springer, Berlin/Heidelberg

[7] S.Svensson, G.S.di Baja: *Using distance transforms to decompose 3d discrete objects*, 2002. Image Vis. Comput. 20(8), 529-540

[8] S.C.Zhu, A.L.Yuille: *Forms: A flexible object recognition and modeling system*, 1996. Int. J. Comput. Vis. 20(3), 187-212

[9] M.Mortara, G.Patané, M.Spagnuolo, B.Falcidieno, J.Rossignac: *Blowing bubbles for multi-scale analysis and decomposition of triangle meshes*, 2003. Algorithmica 38(1), 227-248

[10] V.Kraevoy, D.Julius, A.Sheffer: *Shuffler: Modeling with interchangeable parts*, 2007. Visual Comput.

[11] L.Shapira, A.Shamir, D.Cohen-Or: *Consistent mesh partitioning and skeletonisation using the shape diamtere function*, 2007. Springer-Verlag, 249-259

[12] http://www.cs.tau.ac.il/ liors/research/projects/sdf/

[13] http://3dgraphicsprogramming.blogspot.it/2011/07/gpu-accelerated-shape-diameter-function.html e http://3dgraphicsprogramming.blogspot.it/2011/08/meshlab-plugin-development-depth.html