Tesi di Laurea Triennale in
INGEGNERIA DELL'INFORMAZIONE:

Algoritmi di ricostruzione di superfici 3D a partire da nuvole di punti

Relatore:
Pietro Zanuttigh

Candidato:
Gianmarco Doro

Correlatore:
Enrico Cappelletto

Anno Accademico 2012/2013

Sommario

Lo scopo del presente testo, è quello di presentare in modo semplice i risultati di un'analisi svolta sugli algoritmi di meshing 3D. Nella prima parte, viene presentata una breve descrizione della procedura che permette di ottenere una mesh a partire da nuvole di punti.
Nella parte centrale, l'attenzione è stata posta sugli algoritmi di meshing, presentando nella maniera più semplice possibile il funzionamento, i vantaggi e gli svantaggi di ciascun algoritmo, confrontandoli tra loro. Infine, per la parte di testing, sono state utilizzate nuvole di punti acquisite con il sensore 3D Kinect. Le nuvole di punti sono state infine elaborate tramite l'utilizzo della libreria PCL 1.6.0

Indice

1 La grafica 3D

La computer grafica 3D è un ramo della computer grafica che si occupa dell'elaborazione di modelli tridimensionali da parte di un computer.

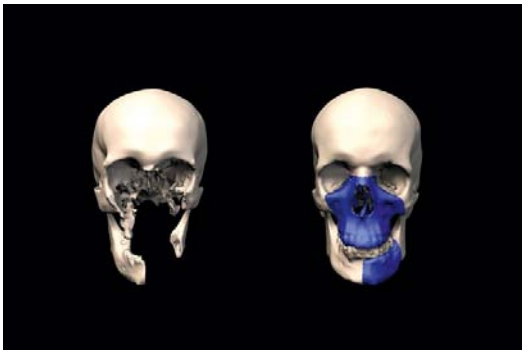


Figura 1: Esempio di ricostruzione 3D di un cranio

Negli ultimi anni l'utilizzo della grafica 3D ha preso piede in una moltitudine di ambiti diversi. Vi sono applicazioni di carattere artistico come nella grafica pubblicitaria, nell'industria cinematografica e della televisione, nei videogame e nelle pubblicazioni editoriali, ma non solo. Con il raffinarsi della tecnologia si sta iniziando a fare largo uso della modellizzazione 3D anche in ambito tecnico o scientifico trovando applicazioni in una moltitudine di campi come la medicina, l'ingegneria civile ed industriale, la geologia e molto altro. L'utilizzo della grafica 3D permette una rappresentazione tridimensionale di un oggetto o di una scena tramite modelli matematici che l'approssimano il più efficacemente possibile.

Figura 2: Ricostruzione 3D a partire da fotografie aeree

Per ricreare una rappresentazione tridimensionale di un oggetto reale, è necessario utilizzare dei sensori, i quali, con tecniche diverse che andremo ad analizzare nei capitoli seguenti, raccolgono una serie di dati dall'ambiente. Sarà poi compito dell'elaboratore andare ad elaborare questi dati per ottenere una superficie tridimensionale in grado di rappresentare efficacemente il campione reale di partenza.
L'operazione di acquisizione dati può essere svolta tramite l'utilizzo di sensori come i laserscanner. Queste apparecchiature, proprio a causa delle loro elevate prestazioni in termini di precisione, posso rivelarsi molto costose e quindi non adatte a tutti gli utilizzi. Nei casi in cui la precisione non è un fattore cruciale per i propri scopi, è possibile sfruttare sensori a costi più contenuti quali il Time of Flight. Altri esempi di sensori a basso costo come il Microsoft Kinect o l'Asus Xtion possono essere, però, problematici in quanto hanno accuratezza e l'affidabilità non eccezionali e possono presentarsi svariati problemi di varia natura quali la presenza di rumore significativo, o la dipendenza della precisione dal colore dell'oggetto che si cerca di ricostruire. Recentemente è divenuto un interessante oggetto di studi come passare dai dati raccolti dai sensori alle superfici finali nonostante le problematiche presentate dai sensori. È possibile fare uso di diversi algoritmi che operano in maniera diversa l'uno dall'altro e sarà obiettivo di questo testo l'analisi dei punti a favore e degli eventuali limiti posti da questi. L'obiettivo è individuare i tipi di algoritmi che sono meno affetti dai limiti dei sensori. Per fare ciò si è partito da misure raccolte tramite sensori a basso costo come il Kinect, ma le conclusioni possono essere estese in maniera analoga ad altri sensori.

2 I Sensori

Esistono molteplici tipi di sensori che permettono l'acquisizione di immagini in 3D. Di seguito saranno analizzati i più comuni sensori, focalizzando l'attenzione su quelli a basso costo.

2.1 Il Kinect e l'Asus Xtion

Il Kinect e l'Asus Xtion hanno un principio di funzionamento molto simile e si differenziano principalmente per le dimensioni.

Figura 3: Microsoft Kinect e Asus Xtion

I due dispositivi sono dotati di una fotocamera che permette di fare delle normali foto a colori fino ad una risoluzione di 1280x1024 e in contemporanea un depth map differenziale. Presentano entrambi due modalità di funzionamento: ad alta risoluzione sono in grado di effettuare acquisizioni ad una frequenza di 30 fps, mentre in bassa risoluzione arrivano fino a 60fps

Questa depth map viene generata tramite un emettitore agli infrarossi in grado di proiettare sulla scena una griglia di punti (dell'ordine di 300'000 punti per scatto) con una frequenza ben precisa. Tramite un sensore calibrato su quella frequenza, i due dispositivi sono in grado di identificare a che distanza si trova ciascuno punto della griglia. Questa depth map verrà poi fusa da un elaboratore alla fotografia a colori. L'errore di cui questi dispositivi sono affetti può essere dovuto a diverse fonti, quali le proprietà delle superficie

dell'oggetto, le condizioni di luce, la distanza dell'oggetto ed una calibrazione non adeguata tra la fotocamera e il sensore di profondità. I sensori a causa della loro architettura, catturano in maniera accurata gli oggetti che si trovano al centro del loro campo visivo, mentre le zone ai margini subiscono un effetto di distorsione che compromette l'accuratezza della depth map. Per risolvere questo problema è necessario ricorrere ad un'accurata calibrazione. Una volta che il dispositivo ha salvato in memoria una rilevazione di un oggetto campione, è in grado di confrontare questa rilevazione con le successive in modo da poter eliminare parte del rumore bianco che lo affligge. La distorsione ai margini del campo visivo non verrà risolta, ma è possibile comunque tenerne conto tramite un valore che rappresenta l'accuratezza della rilevazione del singolo punto in corrispondenza alla sua posizione all'interno della scena. Va inoltre posta una particolare attenzione alla distanza che l'oggetto ha dal sensore; sebbene idealmente il sensore dovrebbe rilevare oggetti che si trovano tra 0,5m e 5m, in realtà presenta un errore random che aumenta quadraticamente con la distanza. A distanze superiori ai 3m, inoltre, anche la bassa risoluzione contribuisce con un errore bianco che potrebbe rivelarsi non indifferente. Per evitare questi problemi è sufficiente effettuare rilevazioni di oggetti che si trovano tra 1 e 3 metri di distanza dal sensore. Una volta tenuto conto di tutte le possibili fonti di errore, tutti i punti della depth map vanno associati a punti della fotografia a colori, ottenendo la cosiddetta nuvola di punti, cioè una serie di vertici in un sistema di coordinate tridimensionali a ciascuno dei quali viene associata un'informazione relativa al suo colore. Questa sarà la base di partenza per andare a ricostruire al computer, in un passo successivo del processo, la superficie originale.

Device	Pro	Contro
Microsoft Kinect	Motorizzato Stabilità con hardware diversi Ha un SDK ufficiale dedicato	Dimensioni Peso Alimentazione esterna
Asus XTion	Dimensioni più compatte Più leggero Alimentato da USB	Meno diffuso Non ha il motore Non funziona con USB 3.0

2.2 Il sensore a tempo di volo

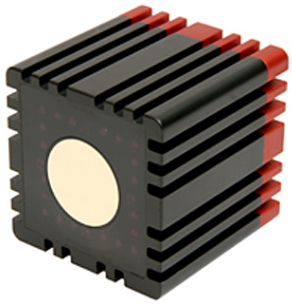


Figura 4: Il time of flight sensor

Il sensore a tempo di volo invia un segnale infrarosso sinusoidale e in base allo sfasamento di fase che si misura nel segnale di ritorno si è in grado di definire a che distanza si trova l'ostacolo. Unitamente a delle telecamere in grado di ottenere l'informazione sul colore, può svolgere la stessa funzione del Kinect con il vantaggio che l'errore è indipendente dalla distanza dell'oggetto. Purtroppo questo sensore ha un costo decisamente maggiore rispetto ai precedenti e presenta anche numerosi altri limiti: l'errore nelle rilevazioni è causato da fattori differenti, come il rumore intrinseco del dispositivo, il materiale di cui è composto l'oggetto e la bassa risoluzione (176x144) che sgrana l'immagine sulla distanza. Va inoltre considerato l'errore di wriggling, che è dovuto al fatto che la sinusoide ricevuta non è mai perfettamente sinusoidale e questo causa una distorsione in tutta l'immagine. Tuttavia questo errore sistematico può essere ridotto confrontando il segnale di ritorno con un'onda quadra invece che con il segnale originale. Apportando dei piccoli accorgimenti che permettono di ridurre l'impatto dei vari errori, è dunque possibile aumentare notevolmente la precisione di questo dispositivo che arriva ad ottenere una precisione superiore a Kinect e Xtion.

3 La pipeline di elaborazione

Una volta che tramite l'utilizzo di sensori si è ottenuta una serie di immagini con associato un certo valore di profondità, è necessario elaborare queste immagini. Per giungere al risultato finale dovrà essere svolta una serie di operazioni fondamentali:

- Acquisizione immagini e depth map tramite sensori
  - Il punto di partenza per l'elaborazione consisterà nell'andare ad effettuare delle rilevazioni con l'ausilio di sensori opportunamente calibrati. Tramite un tool di acquisizione è possibile fare in modo che vengano prese in sequenza diverse immagini di una stessa scena. Queste permettono di vedere la scena da punti di vista diversi in modo da avere informazioni anche su punti che non sarebbero visibili con una sola fotografia. I dati raccolti vengono poi inviati al computer per essere uniti ed elaborati.
- Elaborazione delle nuvole di punti
  - Le varie acquisizioni dovranno essere unite in un'unica nuvola di punti. Questo permette di avere in un'unica nuvola più informazioni possibili sull'oggetto. Unendo immagini da angolazioni diverse, ci permette di ottenere un modello 3D molto più accurato e completo di quello ottenibile con l'ausilio della nuvola di punti di una singola immagine. Le nuvole vengono prima filtrate per eliminare i punti problematici e cercare di ridurre gli eventuali errori dovuti al rumore. I passi successivi consistono nel calcolare la normale, effettuare l'associazione tra colore e la depth map. Viene poi eseguito un calcolo della rilevanza dei singoli punti e viene effettuata una selezione degli n punti più rilevanti. Ora è necessario unire la nuvola di punti a quelle elaborate precedentemente e per fare ciò viene sfruttato l'algoritmo ICP pair-wise. Dopo che tutte le immagini sono state elaborate e unite in un'unica nuvola di punti è il momento di effettuare la vera e propria operazione di meshing.
- Rendering
  - Le nuvole di punti sono ancora una rappresentazione discreta della scena ed è necessaria una trasformazione in superfici continue.

Per fare ciò esistono innumerevoli algoritmi che affrontano il problema con approci diversi. A causa della loro natura, alcuni di questi algoritmi hanno dei punti forti e dei limiti che li caratterizzano. Questo costringe l'utilizzatore a dover operare una scelta in base ai propri scopi e alle necessità specifiche del dispositivo da cui derivano i dati. Nel seguente capitolo verranno elencati e analizzati tutti gli algoritmi presenti nella libreria PCL 1.6.0

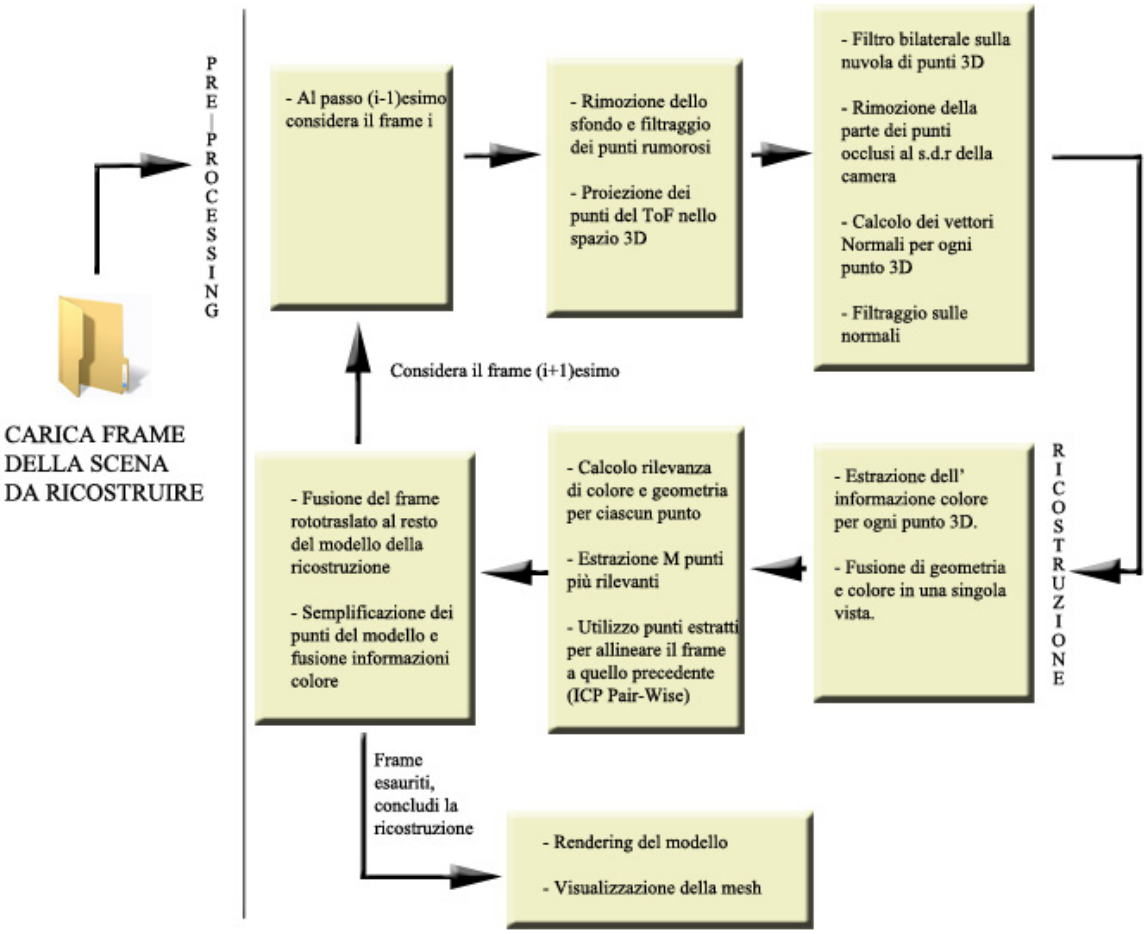


Figura 5: Il sistema di elaborazione

4 Algoritmi di meshing della PCL 1.6.0

4.1 Greedy Projection Triangulation

Questo è un algoritmo iterativo sviluppato da Zoltan Csaba Marton [1] che inizia unendo tre punti in un triangolo iniziale e poi collega altri triangoli adiacenti fino a che tutti i punti non sono stati presi in esame. È stato sviluppato cercando di ottimizzare i tempi di elaborazione del meshing. Esistono diversi tipi di algoritmi di tipo Greedy (come il Ball-pivoting [2] o l'algoritmo di Boyer e Petitjean [3]) ma seguono tutti lo stesso principio dando priorità a fattori diversi. L'algoritmo si basa sulla distanza per capire se un punto fa parte della superficie precedente o di un'altra, e unisce solamente le superfici che rispecchiano certi parametri impostabili dall'utilizzatore, quali l'angolo massimo tra le normali di due superfici e la massima lunghezza dei lati del triangolo. A questo punto può succedere che siano rimasti dei punti che non fanno parte della superficie che è stata generata. In tal caso, se possibile, l'algoritmo genera un nuovo triangolo di partenza e ricomincia ad espanderlo. Questo procedimento viene ripetuto fin quando non sarà più possibile creare i triangoli di partenza. La distanza massima che un punto deve avere dal triangolo più vicino è uno dei parametri che vanno settati per riuscire ad ottenere un risultato ottimale.

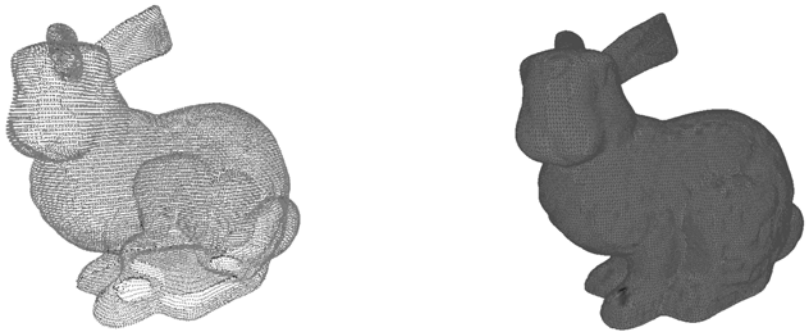


Figura 6: Esempio di meshing di una nuvola di punti generata al computer

Una delle caratteristiche principali di questo metodo è rappresentato dal fatto che non modifica né elimina i punti che riceve in input, né esegue interpolazioni con le superfici. Questo permette di ottenere il massimo della precisione nella creazione delle superfici. Inoltre se la densità dei punti è variabile, questo non rappresenta un problema, infatti questo algoritmo è in grado di adattarsi a densità differenti. Nascono, però, alcuni problemi nel caso in cui le nuvole siano affette da rumore, poiché altera la precisione

con cui vengono ricavate le normali ai triangoli e il risultato può non essere ottimale.

- **Radius:**Per ovviare a questo problema è necessario aumentare il raggio entro il quale vengono presi i vertici dei triangoli. Così facendo il programma farà una media e cercherà di assegnare ad ogni triangolino una normale il più possibile simile a quella dei vicini. Questo, però, se da una parte risolve il problema delle normali su una superficie piana, non funziona altrettanto bene lungo gli spigoli o gli spazi angusti: andando a modificare le normali su un raggio molto ampio è possibile che questi punti critici vengano smussati o modificati significativamente. Poiché il Kinect presenta innumerevoli problemi nel riconoscere gli spigoli, può essere conveniente porre una maggiore attenzione in quei punti critici sacrificando le superfici piane che risulteranno un po' rumorose. Di seguito è rappresentata una mesh raffigurante un bancone del laboratorio e uno zoom del piano mette in luce le problematiche appena discusse.

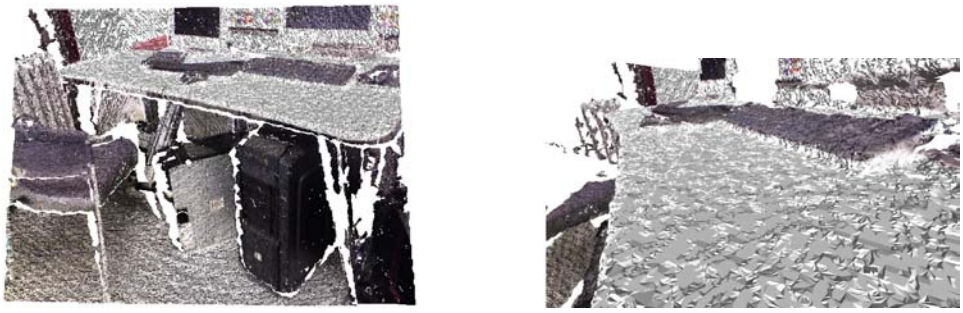


Figura 7: Esempio di meshing di una nuvola di punti ottenuta tramite Kinect

  Nella seconda immagine, è chiaramente visibile come l'aver aumentato il raggio ha permesso di rendere certi tratti del piano di lavoro orientati nella stessa direzione. Purtroppo questo causa delle problematiche nel bordo del tavolo. In ogni caso è necessario trovare il raggio ottimale per i propri scopi.

- **Maximum Number of Neighbours:** Per evitare di appesantire eccessivamente l'algoritmo, è inoltre possibile impostare il numero massimo di vicini che verranno uniti al triangolo attuale. Avranno la precedenza i punti più vicini alla superficie.

Pro	Contro
Può gestire i colori Può gestire densità di punti variabili Precisione elevata	Trade-off tra precisione e dimensione dei poligoni

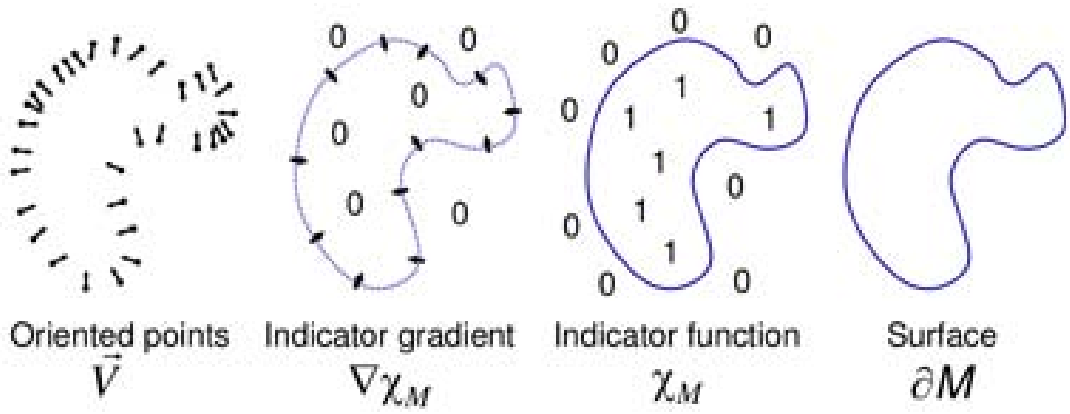
4.2 Organized Fast Mesh

Questo algoritmo punta ad un'elevata velocità di elaborazione delle nuvole di punti. È in grado di ricostruire in tempo reale le superfici a partire da nuvole di punti organizzate, cioè impostate come una struttura (o una matrice), dove i punti sono ordinati in file e colonne. Purtroppo, il nostro tool di acquisizione fornisce una nuvola di punti non strutturata e questo ci impedisce di utilizzare questo algoritmo.
In ogni caso, proprio a causa della sua velocità, la mesh che si ottiene utilizzando questo algoritmo è di scarsa qualità e uno qualsiasi degli altri algoritmi proposti è in grado di fornire una mesh di gran lunga più definita.
Anche la quantità di parametri che è possibile settare è estremamente limitata. L'unico parametro che influisce significativamente sul risultato finale è la lunghezza massima dei lati dei poligoni.

Pro	Contro
Estremamente veloce	Non gestisce i colori Necessita di point cloud organizzate

4.3 Poisson Surface Reconstruction

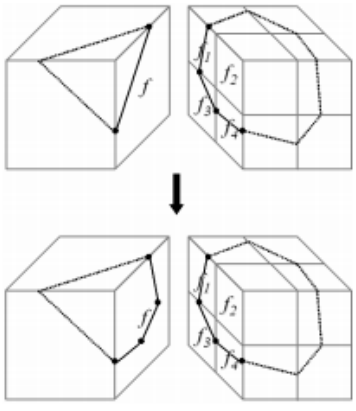
Questo algoritmo è stato sviluppato da Kazhdan, Bolitho e Hoppe [4]. Necessita in input di un dataset che abbia associato ad ogni punto della point cloud la sua normale rispetto alla superficie. In genere, nel far ciò, si fa una media con le normali vicine in modo da aumentare la robustezza al rumore. Partendo dall'insieme di tutte le normali, le quali sono legate al gradiente della superficie, l'algoritmo permette di ricavare la forma dell'oggetto iniziale risolvendo quello che viene visto come un Problema di Poisson, da cui il nome. Per sua natura, però, questo algoritmo può venire utilizzato solo nell'elaborazione di oggetti watertight ovvero chiusi. Questo può risultare problematico quando andiamo a prendere le nuvole di punti di input tramite Kinect, infatti non sempre gli oggetti che andiamo a ricostruire hanno questa proprietà. Nello specifico, utilizzi come la ricostruzione delle pareti di una stanza fotografata dall'interno posso dare risultati non voluti. Se la superficie finale non è watertight, l'algoritmo cerca comunque di trovare il modo di chiuderla, spesso aggiungendo superfici dove non dovrebbero esserci. L'output di questo algoritmo è una isosuperficie, una via di mezzo tra la nuvola di punti e la mesh finale. Quest'ultima andrà poi elaborata tramite algoritmi come Marching Cubes per raggiungere il risultato finale. Le immagini seguenti illustrano in 2D il procedimento seguito dal Poisson Surface Reconstruction Algorithm:

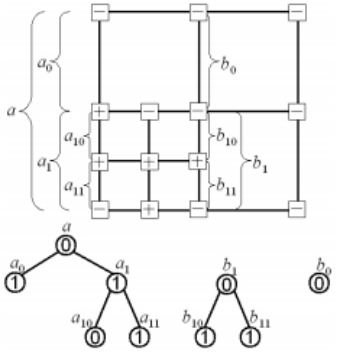


Nella sua implementazione PCL, questo algoritmo ha purtroppo una grave limite: non è in grado di gestire le informazioni sul colore e anche a partire da nuvole di punti che contengono tale informazioni, la mesh finale sarà comunque priva di colore.

4.4 Marching Cubes

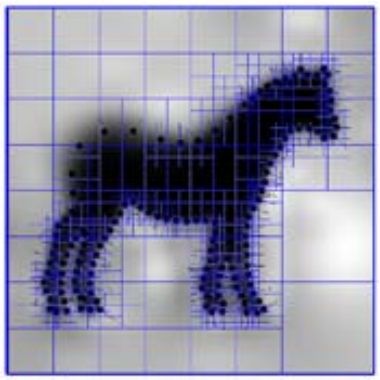
Marching Cubes è un algoritmo sviluppato da Lorensen e Cline [5] e implementato nella libreria PCL da Kazhdan, Klein, Dalal, e Hoppe [6] che richiede come input una isosuperficie. Prima di utilizzarlo, quindi, sarà necessario che la nuvola di punti sia stata precedentemente elaborata da un altro algoritmo che dia in output un'isosuperficie come il Poisson Surface Reconstruction Algorithm. In questo caso, per arrivare alla mesh finale, viene usato un approccio di tipo divide and conquer: l'area circostante l'isosuperficie viene divisa in cubetti chiamati Voxels, i quali conterranno al proprio interno parte della superficie. Inizialmente si sa solo come l'isosuperficie interseca i lati dei voxels e sarà compito dell'algoritmo ricostruire come la superficie è fatta all'interno del cubo.
In prossimità dei punti dove la superficie cambia con un gradiente maggiore, la dimensione dei voxel viene dimezzata una o più volte a seconda della rapidità di variazione delle normali dei punti. Avendo dei voxel troppo grandi si corre il rischio di perdere molte delle informazioni che contengono. Viene dunque sfruttato il cosiddetto octree ovverosia un albero che tiene traccia del numero di suddivisioni che hanno subito i voxel e quindi della loro dimensione. Ogni nodo dell'albero corrisponderà ad un voxel e conterrà delle informazioni che riguardano la quantità di punti che sono presenti al suo interno. Nelle seguenti immagini un esempio spiega il funzionamento di questa procedura:

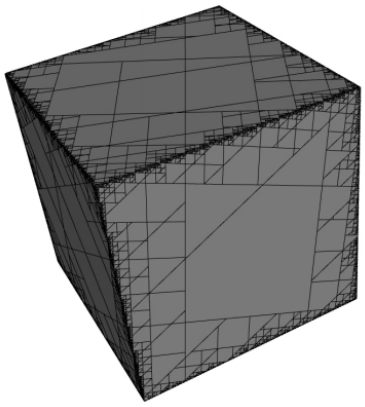




Come si può vedere un maggior numero di suddivisioni porta ad una maggior precisione nella ricostruzione, ma allo stesso tempo anche ad una complessità di calcolo maggiore. Per quantità elevate di punti i tempi potrebbero aumentare considerevolmente. Sarà quindi necessario porre attenzione nel settare i parametri dell'algoritmo per evitare che l'elaborazione da

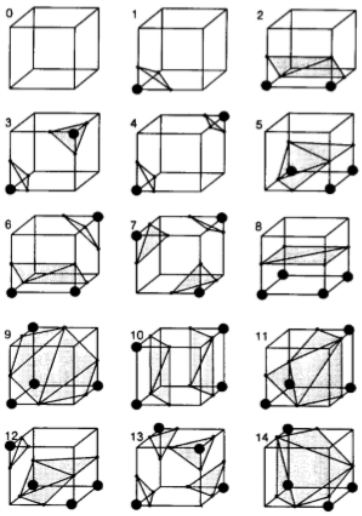
parte del sistema diventi troppo lunga. Il risultato di tutta la procedura sarà dunque qualcosa di simile alle seguenti immagini dove a sinistra è presentato un esempio in 2D mentre a destra è presentata una possibile suddivisione per un cubo.





Come è possibile notare nella seconda immagine, i poligoni sono stati suddivisi maggiormente in prossimità degli spigoli, mentre nelle facce, dove la normale cambia pochissimo da un punto all'altro, è sufficiente una minor precisione.

L'intersezione della superficie con un cubo può avvenire in molti modi, ma per motivi di simmetria questi sono riconducibili soltanto in 14 casi differenti. Partendo dai Voxels e sapendo come questi vengono intersecati dalla superficie, l'algoritmo trova poi il modo di effettuare un'interpolazione affiancandoli tra loro e cercando di unire le varie superfici interne ai cubetti in un'unica superficie finale che sarà poi l'output dell'algoritmo. È tuttavia possibile sfruttare i parametri di input in modo da ottenere il risultato desiderato. Qui di seguito saranno elencati i parametri più importanti settabili dall'utente:



- **Depth:** permette di modificare l'altezza massima dell'octree. Valori più alti offrono una ricostruzione più precisa, mentre valori minori migliorano il tempo di elaborazione. Nelle immagini seguenti sono rappresentate diverse ricostruzioni della stessa point cloud con diverse depth:

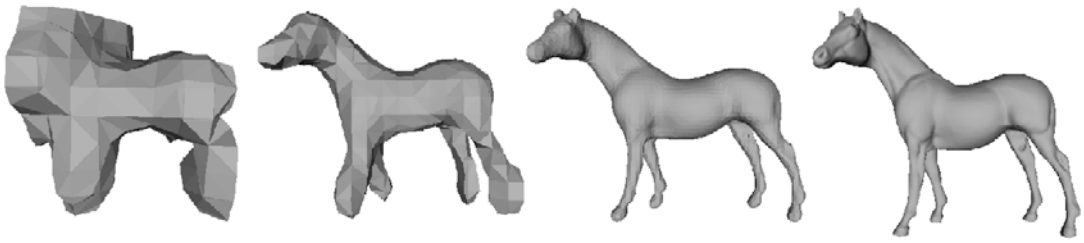


Figura 8: Marching Cubes con Depth = 3, 4, 6, 10

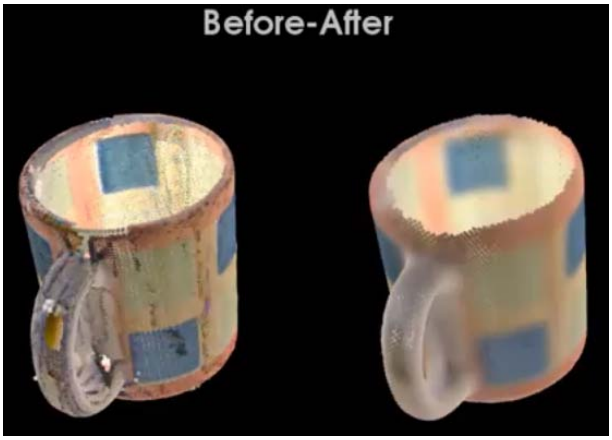
- **Samples per node:** Questo parametro permette di settare quanti punti saranno contenuti in ciascun cubo prima che questo venga suddiviso in cubi più piccoli. Diminuire questo parametro permette di suddividere l'isosuperficie in molti più cubetti, a rischio, però, di aumentare enormemente il tempo di elaborazione per un vantaggio che può essere difficilmente visibile. D'altro canto, aumentare il valore di questo parametro permette di smussare le curve e questo, se fatto in modo appropriato, può risultare in un effetto di soppressione del rumore.

Pro	Contro
Permette la scelta della precisione	Non gestisce i colori
Risultato finale molto pulito	Poisson e Marching Cubes vanno usati insieme

5 Algoritmi di preprocessing e postprocessing delle PCL 1.6.0

5.1 Moving Least Squares

Sviluppato da Cuccuru, Gobbetti, Marton, Pajarola e Pintus [7], Moving Least Squares non è esattamente un algoritmo di meshing, ma se inserito all'interno di altri algoritmi è in grado di modificare e uniformare la distribuzione dei punti del file di partenza in modo da renderli più omogenei.



Questo procedimento da un certo punto di vista può risultare utile, in quanto permette di ridurre il rumore causato dal Kinect. Purtroppo, però, il modificare la posizione dei punti porta la point cloud a non rispecchiare più la realtà come è stata catturata dal sensore: in questo modo si andrebbe ad effettuare un'operazione di meshing su una nuvola di punti che è passata attraverso uno stadio che l'ha drasticamente modificata. A seconda degli utilizzi questo può essere utile ma anche deleterio per l'affidabilità del risultato.

L'algoritmo cerca di uniformare il più possibile la disposizione dei punti in modo che le normali che vengono stimate risultino orientate coerentemente le une con le altre. Nell'immagine iniziale è possibile vedere come i punti vengono disposti se viene applicato l'algoritmo ad una point cloud rumorosa. È evidente che il risultato è stato quello di eliminare buona parte del rumore lungo le superfici laterali dell'oggetto, a discapito, ancora una volta, dei bordi.

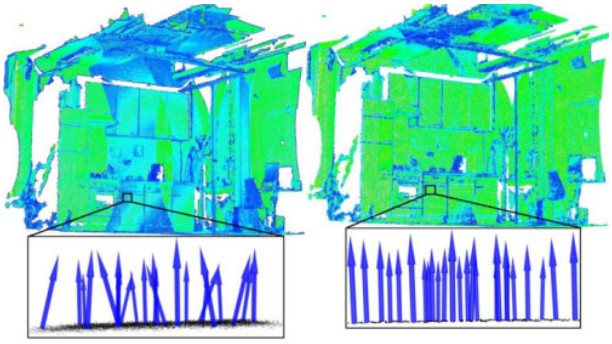
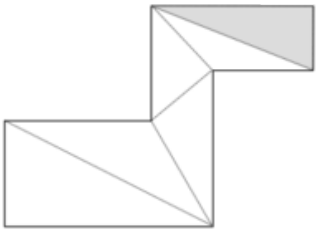


Figura 9: La disposizione delle normali prima e dopo l'algoritmo

Pro	Contro
Diminuisce l'impatto del rumore	I colori risultano sfocati
Migliora il risultato finale	Rallenta gli algoritmi nei quali viene implementato

5.2 Ear Clipping

Il metodo dell'ear clipping permette la suddivisione di un poligono semplice in triangoli.

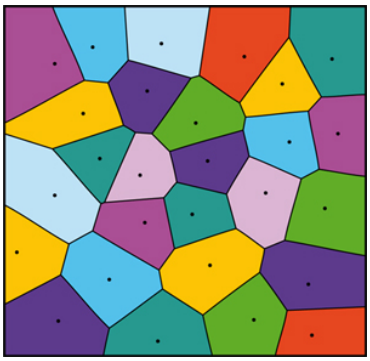


L'algoritmo si basa sul fatto che ogni poligono semplice con almeno 4 vertici e privo di buchi ha almeno due orecchie, che sono triangoli con due dei lati facenti parte dei lati del poligono e il terzo completamente al suo interno. L'algoritmo consiste nel trovare tali orecchie rimuovendole dal poligono di partenza finché non rimarrà un solo triangolo. In questo modo la superficie di partenza è ora suddivisa in diversi triangoli. Questo può essere utile per suddividere una mesh, ottenuta tramite altri algoritmi, in triangoli più piccoli. In questo modo è possibile sapere con certezza di che poligoni è composta la superficie finale, nel caso sia necessario. Il problema di questo algoritmo risiede nel fatto che il poligono da suddividere non deve avere buchi al suo interno. Questo nel caso di una rilevazione fatta tramite sensori rumorosi non è sempre assicurato ed è quindi necessario prendere delle precauzioni: bisognerà assicurarsi di avere una affidabilità tale nella rilevazione che in fase di meshing non ci sia il rischio che vengano lasciati dei buchi. Questo può essere ottenuto anche tramite un'accurata scelta dei parametri dell'algoritmo di meshing.

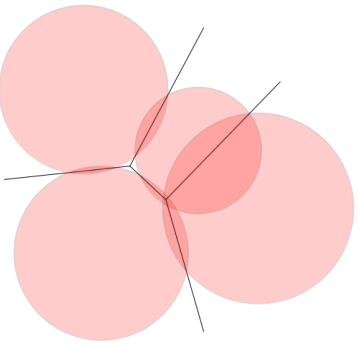
6 Algoritmi di meshing esterni alla PCL 1.6.0

6.1 Power crust

L'algoritmo Power crust, ad opera di N. Amenta, S. Choi, R. K. Kolluri [8] si basa sull'utilizzo in sequenza di una serie di strumenti matematici che, uniti, permettono di ottenere la superficie finale desiderata.



Il primo di questi è il diagramma di Voronoi: un particolare tipo di decomposizione di uno spazio metrico determinata dalle distanze rispetto ad un insieme finito di punti, in questo caso la nuvola di punti di ingresso. Dati due punti vicini tra loro, si trova la superficie posta ad uguale distanza tra di essi e la si interseca con le superfici generate dalle altre coppie di punti. Una volta concluso il procedimento lo spazio attorno ai punti verrà diviso in sottosezioni ciascuna contenente un solo punto. Da questo diagramma (composto dalle superfici che delimitano ciascuna sottosezione) verrà dunque preso un sottoinsieme dei vertici di Voronoi chiamato sottoinsieme dei poli. Questi ultimi andranno a formare il MAT (ovvero Medial Axis Transform) approssimato, che può essere visto come uno scheletro della nostra superficie finale. Attorno ai punti del MAT approssimato andranno costruite delle sfere chiamate Sfere Polari. Queste sfere avranno sulla loro superficie i punti della nuvola di partenza, ma non avranno nessun punto al loro interno. Il loro raggio massimo determina il peso di ciascun polo.



Viene dunque calcolato il Power Diagram delle sezioni di Voronoi, ovverosia l'area di spazio che è contenuta tra ciascuna superficie di Voronoi e lo spicchio di sfera che si trova più vicino ad esso. Etichettiamo dunque alcune parti del Power Diagram come facenti parte dell'interno della superficie finale. Le porzioni di sfera che divideranno il Power diagram delle superfici interne da quelle interne formeranno la Power Crust ossia la nostra superficie finale. Qui di seguito sono reppresentate delle rappresentazioni in 2D del procedimento per facilitarne la comprensione.

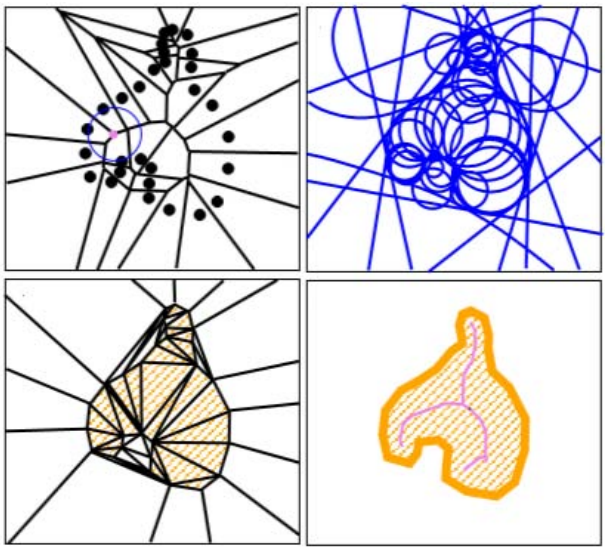
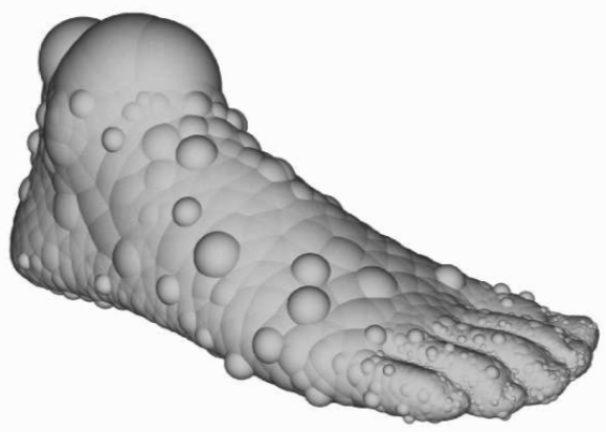
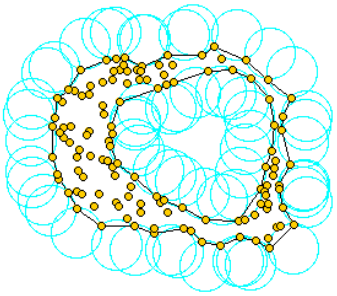


Figura 10: Diagramma di Voronoi, Power Diagram, Power Crust, Risultato finale

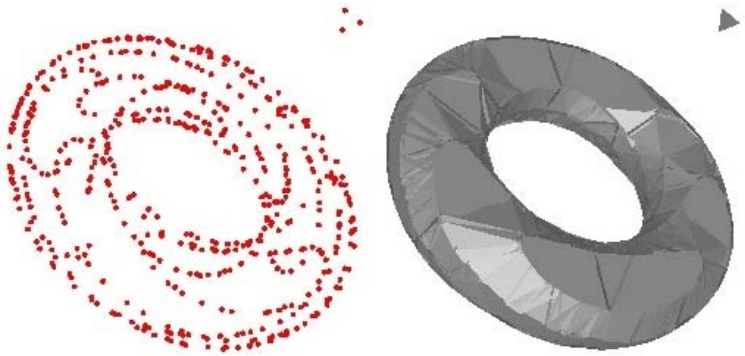
Questo algoritmo, purtroppo, oltre a non poter utilizzare le informazioni sul colore, non è in grado di gestire bene né le nuvole di punti rumorose né le superfici watertight. La natura dell'algoritmo, anzi, tende ad amplificare gli errori della point cloud in un modo che è bene evidente nella seguente immagine: i punti che a causa del rumore si discostano verso l'esterno della nuvola di punti vengono ricostruiti come delle sfere, mentre all'altezza della caviglia, dove non sono presenti punti, vengono create delle sfere che tentano di chiudere la superficie in un modo similare a quanto visto con Poisson.



6.2 Alpha shapes



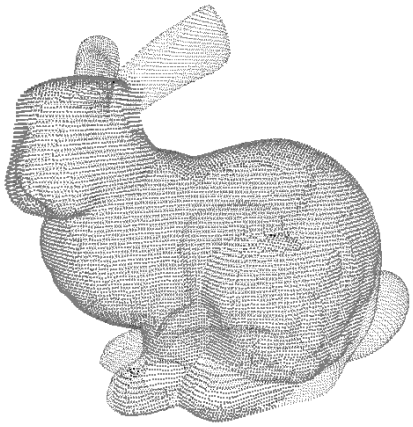
Questo algoritmo sviluppato da Edelsbrunner e Mucke [10] a partire da una nuvola di punti effettua il meshing in una maniera simile alla scultura tradizionale. Tale algoritmo considera come superficie iniziale una zona di R^3 che contiene tutti i punti della nuvola di punti e poi vi elimina, utilizzando una sfera di raggio fissabile, tutte quelle zone che non contengono all'interno della sfera dei punti appartenenti alla nuvola. In questo modo, del volume iniziale rimarrà sono una piccola parte. La superficie che contiene questa area è chiamata Alpha Shape, e sarà anche l'output dell'algoritmo. Purtroppo, questo procedimento, oltre che molto oneroso computazionalmente è anche molto piuttosto impreciso nell'elaborare nuvole di punti con densità variabile. C'è il rischio, infatti, di creare buchi nella superficie dove i punti sono più radi, e di perdere informazioni nelle zone dove i punti di margine di due superfici differenti sono molto vicini, unendoli. Nel seguente esempio è possibile vedere come una nuvola di punti viene ricostruita:



7 Risultati

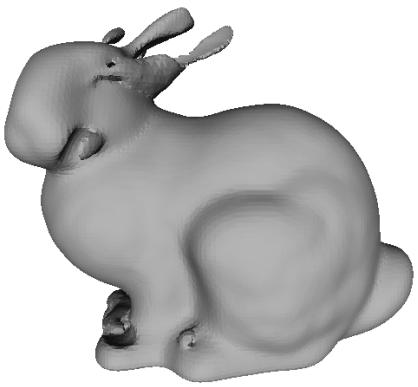
Qui di seguito sono rappresentate e poste a confronto delle mesh ottenute tramite l'utilizzo dei diversi algoritmi. Per ottenere questi risultati si è utilizzato un Kinect opportunamente calibrato e le librerie open source PCL 1.6.0. Si è scelto inoltre di mostrare anche una mesh non colorata prodotta dall'algoritmo Greedy in modo da facilitare un confronto qualitativo con gli algoritmi Marching Cubes - Poisson

In questo primo caso la nuvola di punti era quella di un coniglio; è stata generata al computer ed è watertight, quindi adatta ad essere elaborata da ogni algoritmo.



(a) Point Cloud iniziale

(b) Greedy Projection Triangulation



(c) Marching Cubes - Poisson

(d) PowerCrust

Le due mesh ottenute hanno caratteristiche differenti: il Greedy Projection Triangulation ha ricostruito la superficie molto velocemente e in maniera abbastanza fedele, ma ha lasciato dei buchi in certi punti come nelle orecchie e nelle zampe del coniglio. Questi buchi sono dovuti ad un errato calcolo della direzione delle normali. Per provare a correggere questi errori è necessario modificare il ViewPoint, ossia il punto verso il quale vengono orientate tutte le normali alle superfici. Nel far ciò, può darsi che si riesca a correggere la ricostruzione nei punti critici appena descritti, ma questo andrebbe a scapito di altri. Per ciascuna nuvola di punti sarà necessario trovare il punto dove il ViewPoint causa la minor quantità di errori possibile. Generalmente il baricentro è il punto più affidabile, ma, come è possibile vedere dalle figure precedenti, anche questo può causare degli errori.

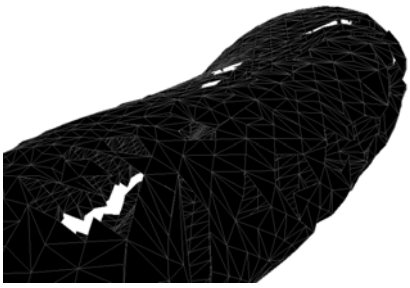


Figura 11: Particolare dell'orecchio

L'algoritmo composto da Marching Cubes e Poisson, invece, sebbene dia un risultato più apprezzabile in termini visivi, fallisce anch'esso nel ricostruire i punti più critici. Le orecchie non vengono ricostruite bene e sulle zampe vengono creati degli artefatti. Anche in questo caso il problema è dovuto al calcolo delle normali e sono valide le considerazioni appena fatte.
In definitiva in questo caso l'algoritmo Greedy potrebbe essere la scelta migliore per la ricostruzione, in quanto gli artefatti generati dall'algoritmo di Poisson sono molto evidenti e di difficile soluzione.
L'algoritmo Powercrust, infine, ha prodotto una mesh che, sebbene sia meno affetta dai problemi derivati dal calcolo delle normali, è molto sbozzata per la natura stessa dell'algoritmo che l'ha generata.

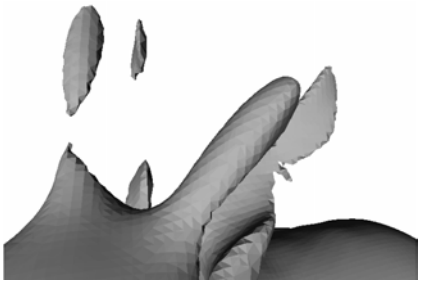


Figura 12: Punti critici delle orecchie

Le immagini a pagina seguente riguardano una nuvola di punti watertight acquisita con il Kinect e quindi rumorosa. Essendo watertight, anche in questo caso possiamo usare entrambi gli algoritmi, con la differenza che l'algoritmo Greedy permette la gestione dei colori.

(a) Point Cloud iniziale

(b) Greedy Projection Triangulation

(c) Greedy Projection Triangulation in bianco e nero

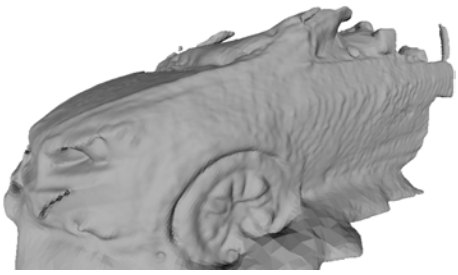
(d) Marching Cubes - Poisson

È evidente che il rumore introdotto dal Kinect rende molto più difficoltoso il procedimento di meshing. Ancora una volta Poisson introduce degli artefatti su collo e capelli dovuti ad un errato calcolo delle normali, ma d'altro canto esegue un'operazione di interpolazione che elimina parzialmente il rumore presente in tutto il corpo. Questo tipo di rumore invece è molto presente nel caso dell'algoritmo Greedy e distorce il volto in più punti. L'algoritmo migliore dipende dunque dalle necessità dell'utente.

(e) Point Cloud iniziale

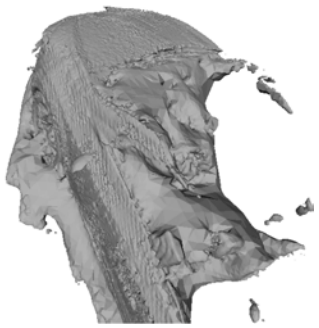
(f) Greedy Projection Triangulation

(g) Greedy Projection Triangulation in bianco e nero



(h) Marching Cubes - Poisson

In questo caso è stata usata una nuvola di punti rumorosa e non watertight. La rilevazione tramite il sensore è stata complicata anche dal fatto che l'oggetto era illuminato da luce solare (la quale complica la rilevazione dei punti infrarossi da parte del sensore) e dal fatto che l'oggetto che stiamo andando a ricostruire originariamente aveva una superficie riflettente, che rende difficoltoso il calcolo della depth map. Queste problematiche sono ben evidenti nella mesh costruita con l'algoritmo Greedy, nella quale il colore non è uniforme, ma presenta delle fasce nere. Nonostante ciò, questo tipo di ricostruzione è comunque preferibile ad una ottenuta tramite Poisson.

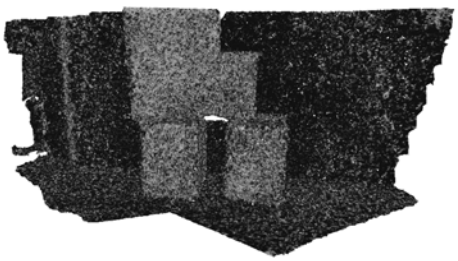


L'algoritmo di Poisson, infatti, non è in grado di gestire né i colori né le superfici non watertight. Una volta ricevuta in input una superficie aperta, l'algoritmo tenterà di chiuderla in tutti i modi, andando a creare degli artefatti non desiderati, chiaramente visibili in prossimità della ruota. L'immagine qui a fianco mostra invece l'altro lato della macchina, che non essendo stato fotografato dal Kinect, è stato chiuso dall'algoritmo di Poisson.

Infine, viene presentato un confronto tra i risultati dei vari algoritmi applicati alla ricostruzione di interni. È stato usato il Kinect per catturare una nuvola di punti che rappresentasse le pareti di una stanza vista dall'interno. La superficie che otterremo non potrà essere Watertight, pertanto ci aspettiamo che l'algoritmo di Poisson produca degli artefatti che non permetteranno una ricostruzione fedele.

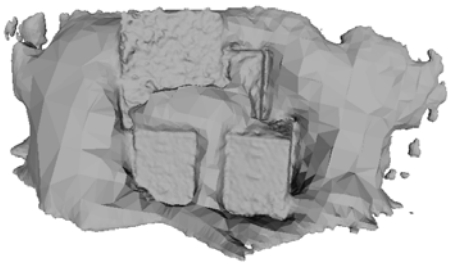
(i) Scacchiera - Greedy

(j) Interno di una stanza - Greedy

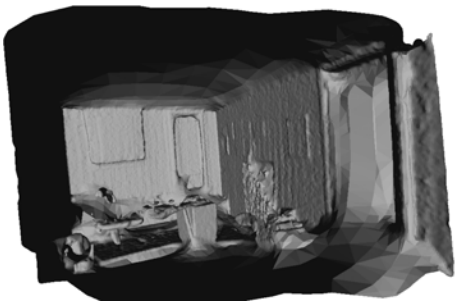


(k) Scacchiera - Greedy in bianco e nero

(l) Interno di una stanza - Greedy in bianco e nero



(m) Scacchiera Poisson



(n) Interno di una stanza - Poisson

Come si può vedere, nel caso della singola parete, l'algoritmo Greedy la ricostruisce in maniera decisamente più fedele, mentre Poisson la deforma in maniera tale da renderla quasi irriconoscibile. Tuttavia nel ricostruire tutte e quattro le pareti interne di una stanza l'effetto è diverso. La chiusura viene operata semplicemente aggiungendo pavimenti e soffitti e molti oggetti vengono collegati al suolo. Il greedy, invece, si limita a rappresentare i dati in ingresso il più fedelmente possibile e questo porta ad avere poligoni colorati

scollegati dal resto della scena. In ogni caso, per la ricostruzione di interni, è generalmente preferibile far ricadere la propria scelta su un algoritmo Greedy.

8 Conclusione

In questo testo sono stati fatti un'analisi ed un confronto dei vari algoritmi proposti dalla PCL 1.6.0. Come si è visto, ci sono molti fattori che possono giocare un ruolo fondamentale nella scelta dell'algoritmo migliore. Questa dovrebbe ricadere generalmente sugli algoritmi Marching Cubes - Poisson se si sta ricostruendo superfici watertight e nelle quali l'informazione sul colore non è un fattore importante. Se queste condizione non sono rispettate, allora è preferibile l'utilizzo di un algoritmo di tipo Greedy che, invece, è in grado di gestire egregiamente le superfici aperte e i colori. In ogni caso sarà compito dell'utente un'analisi del comportamento dei due algoritmi in ogni caso specifico.

Riferimenti bibliografici

[1] Z. C. Marton, R. B. Rusu, M. Beetz,
On fast surface reconstruction methods for large and noisy point clouds.
Robotics and Automation, 2009. ICRA'09. IEEE International Conference on. IEEE, 2009.

[2] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, G. Taubin,
The Ball-Pivoting Algorithm for Surface Reconstruction.
Visualization and Computer Graphics, IEEE Transactions on, 5(4), 349-359.

[3] S. Petitjean, E. Boyer,
Regular and non-regular point sets: properties and reconstruction.
Computational Geometry, 19(2), 101-126.

[4] M. Kazhdan, M. Bolitho, H. Hoppe,
Poisson Surface Reconstruction.
In: Proceedings of the fourth Eurographics symposium on Geometry processing. 2006.

[5] W. E. Lorensen, H. E. Cline,
Marching cubes: a high resolution 3D surface construction algorithm.
In ACM Siggraph Computer Graphics (Vol. 21, No. 4, pp. 163-169). ACM.

[6] M. Kazhdan, A. Klein, K. Dalal, H. Hoppe,
Unconstrained isosurface extraction on arbitrary octrees.
In Symposium on Geometry Processing (pp. 125-133).

[7] G. Cuccuru, E. Gobbetti, F. Marton, R. Pajarola, R. Pintus,
Fast low-memory streaming MLS reconstruction of point-sampled surfaces.
Proceedings of Graphics Interface 2009. Canadian Information Processing Society, 2009.

[8] N. Amenta, S. Choi, R. K. Kolluri,
The Power Crust.
Proceedings of the sixth ACM symposium on Solid modeling and applications. ACM, 2001.

[9] N. Amenta, S. Choi, R. K. Kolluri,
The Power Crust, Union of Balls and Medial Axis Transform.
Computational Geometry, 19(2) (2001), 127-153.

[10] H. Edelsbrunnher, E. P. Mucke,
Three-dimensional alpha shapes.
ACM Transactions on Graphics (TOG) 13(1) (1994): 43-72.

[11] E. Cappelletto, P. Zanuttigh, G. M. Cortelazzo,
Handheld scanning with 3D cameras.
Proceedings of MMSP2013 (2013)

[12] K. Khoshelham, S. O. Elberink,
Accuracy and resolution of Kinect depth data for indoor mapping applications.
Sensors 12(2) (2012): 1437-1454.

[13] J. Hyvärinen,
Surface reconstruction of point clouds captured with Microsoft Kinect.
(2012)

[14] C. Dal Mutto, P. Zanuttigh and G. M. Cortelazzo,
Time-of-Flight Cameras and Microsoft Kinect.
Springer, 2012.