Università degli studi di Padova
Facoltá di Ingegneria

Tesi di Laurea Triennale in

INGEGNERIA DELL'INFORMAZIONE

Ricostruzione di scene 3D a colori

Relatore
Prof. Pietro Zanuttigh
Co-relatore
Cappelletto Enrico

Candidato
Luca Palmieri

Anno Accademico 2012/2013

Dedica riga prima
Dedica riga seconda

Abstract

La tesi tratta il miglioramento di un programma di ricostruzione di modelli tridimensionali. Tale programma, attraverso l'acquisizione della scena utilizzando sensori di profondità (depth-cameras) come il Kinect [4] è in grado di elaborare i dati fino a completare una riproduzione della geometria della scena. Il lavoro è stato progettato come un'implementazione di un'estensione successiva che sopperisce alla mancanza di un'accurata gestione del colore nel modello: si è quindi andati a lavorare sull'informazione di colore che i sensori forniscono in modo da rendere il programma più robusto e la riproduzione più fedele.

Indice

Capitolo 1

Sommario

La fedele riproduzione delle tre dimensioni della vita quotidiana in un modello bidimensionale visibile in uno schermo è sempre stato un ambito molto seguito e molto complesso, e numerosi sono stati i tentativi di ottenere soddisfacenti risultati sul piano tecnologico: generalmente tutti consistono nell'acquisizione di un certo numero di viste tridimensionali appunto e nella loro fusione in un modello completo. La recente introduzione introduzione di telecamere come il Kinect o le Time Of Flight cameras [4] hanno velocizzato e semplificato l'acquisizione di viste tridimensionali: questo ha notevolmente incrementato il numero di viste che si possono acquisire in poco tempo grazie al framerate molto elevato (di solito intorno $30 fps$) e questo rappresenta un aspetto positivo che si porta con sè però un limite computazionale, infatti richiede grandi sforzi a livello di elaborazione dei dati e di memoria da utilizzare; inoltre, queste innovative tecnologie hanno dei limiti nella precisione e nell'affidabilità nella ricostruzione della scena dovuti al rumore nel segnale, alla risoluzione limitata in alcuni punti e ad altri errori tecnologici, che vanno tenuti in considerazione e possono essere limitati e/o ammortizzati dagli algoritmi di ricostruzione che vengono applicati alle viste. Il campo della ricostruzione di scene in tre dimensioni in questo momento attraversa una fase di intenso sviluppo; tra le varie tecnologie e progetti che si stanno sviluppando nel campo della ricostruzione il KinectFusion di Microsoft permette un'accurata ricostruzione, ma il suo algoritmo richiede una grande quantità di memoria per il salvataggio dei dati necessari, e questo ne limita notevolmente la sua rapidità e il suo utilizzo per scene di grandi dimensioni. [1] Il progetto di KinectFusion è probabilmente l'esempio più famoso dell'utilizzo di questo tipo di tecnologie, ma sicuramente non è l'unico: si cita anche l'estensione dello stesso in un progetto svolto dal MIT di Boston, chiamato Kintinuous [2], che lavora sulla mappatura di nuvole di punti ancora dense e sviluppa una soluzione parallela per l'allineamento dei punti delle varie viste;

Fig. 1.1: Modello 3D eseguito con KinectFusion

ma esistono anche applicazioni meno astratte e più concrete, infatti stanno nascendo le prime applicazioni commerciali come *Reconstructme*[8] , un potente scanner tridimensionale in grado di elaborare modelli in tempo reale o *Skanect*[9], che facilita ulteriormente quello che può a prima vista sembrare un'operazione vasta e complessa come riprodurre un modello tridimensionale grazie ad un interfaccia user-friendly; entrambi permettono di ottenere una completa rappresentazione di un oggetto in tre dimensioni completo di colore, anche se la qualità della riproduzione non sempre è soddisfacente. Conseguenza di tutto ciò è lo studio e lo sviluppo di algoritmi che migliorino la fedeltà della ricostruzione del modello, obiettivo non ancora raggiunto ad un livello soddisfacente. Partendo da un programma funzionante che prevede la ricostruzione attraverso una pipeline di operazioni che forniscano robustezza al rumore e accuratezza in fase di fusione delle viste, la tesi si riserva di proporre un'implementazione alternativa delle informazioni di colore che i sensori già citati forniscono in modo da migliorare la precisione del modello finale, alzando il margine di resistenza a problemi che si possono presentare come il rumore del segnale o l'eccessiva luminosità e/o riflessione di alcune superfici che possono contribuire a peggiorare la qualità dell'immagine finale. La tesi illustra i passaggi fondamentali del lavoro analizzando e motivando le operazioni e le scelte effettuate nella successiva fase di implementazione.

Capitolo 2

Progettazione

La tesi si è sviluppata fondando le basi su un progetto precedentemente sviluppato dal laboratorio LTTM (Laboratorio di Tecnologia e Telecomunicazioni Multimediali) che operava la ricostruzione di scene 3D a partire dai dati acquisiti dal Microsoft Kinect o da dispositivi simili, come l'Asus Xtion Pro Live, insieme al sensore della Microsoft tra i sensori a disposizione del laboratorio. L'algoritmo gestisce ed elabora le immagini in nuvole di punti tridimensionali in modo da ottenere una fedele riproduzione degli oggetti, ed il punto in cui i margini di miglioramento erano i più elevati è stato individuato nella gestione delle informazioni di colore, ed è quindi lì che si è andati ad operare.

2.1 Il programma pre-esistente

Il programma di base è stato sviluppato nel linguaggio C++ sfruttando l'editor della Microsoft Visual Studio per la scrittura del codice e l'esecuzione. Nel procedimento l'algoritmo usufruisce della Point Cloud Library [6] che fornisce una notevole scelta di pattern di esecuzione utili nel campo delle nuvole di punti. L'algoritmo sviluppato dal laboratorio è personalizzato e prevede una pipeline di ricostruzione della scena al fine di ottimizzare la riproduzione che si divide in varie fasi:

2.1.1 Pre-Processing

La prima fase dell'algoritmo sopracitato è destinata alla preparazione dei dati. Consiste in alcune necessarie operazioni preliminari che permettono poi l'esecuzione vera e propria dell'algoritmo: vengono utilizzate le informazioni ricevute dallo strumento usato per acquisire i frame delle immagini (depth-cameras come il Microsoft Kinect o l'Asus Xtion Pro Live, vedi capitolo 3), quindi viene applicato un filtro bilaterale sulla depth-map dell'immagine per ridurre il rumore e l'imprecisione dei dati raccolti e per dare maggiore robustezza alla nostra ricostruzione. Si calcolano quindi le normali alla superficie rispetto alla posizione del dispositivo che acquisisce le immagini, e, unendo queste informazioni alla mappa a colori dell'immagine viene creata una nuvola di punti (densa) colorata.

2.1.2 Estrazione dei punti salienti

Data la grande quantità di punti immagazzinati nella fase di acquisizione delle immagini vi è bisogno di una delicata scelta riguardo quali punti effettivamente utilizzare nel successivo allineamento delle nuvole di punti. La scelta viene fatta assegnando due valori di importanza/distintività ad ogni punto secondo due diversi criteri (uno basato sulla geometria e uno basato sul colore) e poi scegliendo tutti i punti in cui il valore maggiore tra i due supera una certa soglia (diventando quindi un punto saliente). I due criteri si basano sull'importanza che un punto riveste nella scena, quindi saranno privilegiati punti che separano due zone dell'immagine, punti di confine, punti che costituiscono la forma degli oggetti. L'immagine viene divisa in quadranti e per ogni quadrante vengono scelti un numero prefissato di punti salienti che andranno a formare quella porzione di immagine. Se un quadrante non contiene abbastanza punti salienti (dovuto al fatto che magari è una porzione di immagine piatta è monocolore) viene incrementato il numero di punti salienti selezionabili per gli altri quadranti.

2.1.3 L'algoritmo Iterative-Closest-Point

Una volta effettuata la scelta dei punti che andranno a formare le varie nuvole di punti viene utilizzato l'algoritmo ICP per allineare le varie nuvole di punti che andranno poi a creare il modello: l'algoritmo funziona in modo iterativo come il nome suggerisce e si basa sull'utilizzo di un Kd-Tree in cui vengono memorizzate le tre componenti spaziali del punto (x, y, z) e su cui vengono calcolate le corrispondenti distanze, quindi viene cercato, per ogni iterazione, di avvicinare i punti tra loro più vicini in modo da allineare due nuvole di punti senza perdita di informazione; l'algoritmo funziona finché ogni punto non è stato avvicinato. La versione proposta in laboratorio è stata modificata da in uno studio precedente [3] aggiungendo le due componenti del colore convertite nello spazio Cielab (a, b) normalizzate e moltiplicate per una costante dal valore ottenuto sperimentalmente, in modo da allineare le nuvole di punti in modo più accurato sfruttando la somiglianza del colore nei diversi punti.

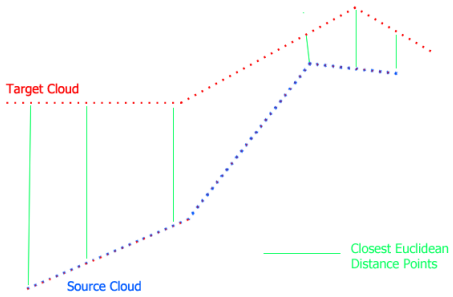

Fig. 2.1: ICP

2.1.4 La fusione del colore e della geometria

Una volta effettuato l'allineamento delle nuvole di punti si ottiene un modello completo e tridimensionale della scena che deve essere completato con la fusione delle informazioni di colore. Infatti, se le informazioni geometriche sono state allineate dall'esecuzione dell'ICP ogni punto dispone ancora di diverse informazioni di colore in quanto un punto acquisito da diversi frame ottenuti da diverse acquisizioni del dispositivo può risultare di colore diverso, fatto dovuto alle diverse angolazioni da cui è stato osservato e quindi dalla diversa riflessione della luce o dal rumore. Nella versione di partenza ad ogni punto viene assegnato il colore del frame acquisita con il prodotto tra la normale alla superficie e la normale alla camera con il valore più elevato, quindi quello preso frontalmente.

Capitolo 3

Hardware

Tutto il progetto è calibrato sui sensori che forniscono le informazioni, depth-cameras a basso costo facilmente reperibili sul mercato, in particolare sul sensore della Microsoft Kinect, e il suo simile Asus Xtion Pro Live, entrambi dal funzionamento analogo.

3.1 Microsoft Kinect

Fig. 3.1: Microsoft Kinect

3.1.1 Struttura

Il Kinect, il sensore della Microsoft, è dotato di telecamera RGB e di un doppio sensore di profondità a raggi infrarossi composto da uno scanner laser a infrarossi e da una telecamera sensibile alla stessa banda. La telecamera RGB ha una risoluzione di 1280×1024 pixel, mentre quella a infrarossi usa una matrice di 640×480 pixel; inoltre dispone di un array di microfoni utilizzato dal sistema per la calibrazione dell'ambiente in cui ci si trova, mediante l'analisi della riflessione del suono sulle pareti e sull'arredamento, in modo da poter riconoscere i comandi vocali.

3.1.2 Funzionamento

Il sensore della Microsoft basa la sua ricostruzione della scena tridimensionale su un immagine a colori e su una disparity map che rappresenta la geometria tridimensionale della scena.

Fig. 3.2: Immagine a colori

Fig. 3.3: Disparity map

Nella stereopsi una disparity map è praticamente una rappresentazione bidimensionale delle varie distanze tra i punti dell'immagine fornita dall'utilizzo incrociato delle due telecamere, in cui viene elaborata la distanza che un punto ha in base alla differenza tra la posizione che il punto ha nell'immagine sinistra e la posizione che ha invece in quella destra: una volta completato una mappa interna di tutte le distanze dei punti, l'immagine viene riprodotta colorata utilizzando scale diverse di grigi per indicare la distanza che un oggetto ha dall'osservatore. Come si nota spesso, nel modello a tre dimensioni molti punti non sono presenti: sono i cosiddetti punti occlusi, punti che compaiono in una delle due immagini che costituiranno poi la disparity map, ma non nell'altra, quindi punti la cui posizione tridimensionale non è calcolabile con l'algoritmo del sensore per cui vengono scartati nella riproduzione. Per ottenere l'effettiva geometria della scena, il Kinect invece proietta un pattern predefinito sulla scena utilizzando la telecamera a raggi infrarossi di cui dispone, in modo da utilizzare frequenze non appartenenti allo spettro

del visibile e quindi di non disturbare l'utente che utilizza il dispositivo o la scena che si sta acquisendo, e opera una triangolazione tra la posizione che i punti del pattern hanno assunto effettivamente nella scena e la posizione che avrebbero se il pattern fosse proiettato su una superficie piana posta ad una certa distanza (informazioni predefinite già presenti nel dispositivo) ottenendo un'immagine grezza delle distanze, che attraverso la calibrazione del sensore, fornisce la completa geometria dell'immagine tridimensionale che, unita all'immagine a colori, forma la vista completa.

3.2 Asus Xtion Pro Live

Fig. 3.4: Asus Xtion Pro Live

Struttura

Il dispositivo targato Asus è molto simile al Kinect per struttura, sebbene sia notevolmente più piccolo e più leggero, quindi più maneggevole, ma non per questo meno potente: dispone anch'esso di una telecamera RGB e di un sensore ad infrarossi per la misurazione delle distanze, con risoluzione analoga al Kinect, 640×480 pixel, ed è in grado di riconoscere una vasta gamma di segnali audio, dai comandi vocali ad altri tipi suoni. [5] Per quanto concerne il funzionamento, l'Asus Xtion Pro Live ha le stesse caratteristiche del già citato Kinect e basa il suo funzionamento sulla stessa tecnica del pattern di luce esposta nel paragrafo precedente.

Capitolo 4

Analisi e progettazione dell'algoritmo

Al fine di proporre la fusione del colore in maniera alternativa, sfruttando tutte le informazioni che il sensore è in grado di fornirci, si progetta il lavoro in modo tale da garantire la robustezza al rumore e quindi una precisione migliore: analizzando i problemi più riscontrati troviamo sicuramente la suscettibilità della telecamera all'illuminazione esterna.
Caratteristica di molte acquisizioni è infatti la presenza di luce (naturale o artificale) che va ad inficiare l'immagine che verrà poi acquisita dalla telecamera del sensore: questo problema si verifica soprattutto in acquisizioni in ambiente esterno (ben consapevoli della loro maggior sporadicità, dato il tipo di sensore e l'uso per cui di solito viene adoperato), ma anche in presenza di superfici altamente riflettenti (come schermi, specchi, occhiali ecc) che spesso si trovano nelle scene da riprodurre, e quindi va considerato in fase di ricostruzione.

4.1 La funzione affidabilità

Base del lavoro è di conseguenza la pianificazione e l'elaborazione di un'adeguata funzione affidabilità: si è elaborato un algoritmo che per ogni punto possa darci un'indicazione di quanto le informazioni (di colore) di quel punto possano essere affidabili, e quindi quanto possano essere tenute in considerazione. Tale funzione sarà poi utilizzata all'interno della fusione dell'informazione di colore, in quanto ci consente di migliorare l'accuratezza e la precisione del colore nel modello finale. Siano R la funzione affidabilità, n_p le normali del punto (essendo tridimensionale il punto ha tre componenti), L la luminanza e σ^2 la varianza del punto, allora la funzione affidabilità è stata

progettata in funzione di 3 parametri:

$$R = f(\mathbf{n}, L, \sigma^2)$$

La funzione affidabilità è stata pensata per assumere valori compresi tra 0 e 1 in modo da essere più facilmente utilizzabile e comprensibile ($R \to 1$ significa affidabilità elevata, $R \to 0$ significa affidabilità nulla), quindi definiamo la nostra funzione come la somma delle tre componenti indipendenti l'una dall'altra.

$$R = R_n + R_L + R_\sigma \qquad\qquad R \in [0, 1]$$

4.1.1 La normale del punto

La normale **n** del punto viene calcolata come il prodotto scalare tra la normale alla superficie del punto in questione (n_p) e la normale alla superficie della camera (n_{cam}) nel momento in cui sta acquisendo il punto. La normale fornisce un'indicazione di quanto sia inclinata la superficie a cui appartiene il punto che stiamo registrando e quindi quanto quel punto possa essere affidabile e utile in fase di ricostruzione. Valutiamo come inaffidabile (in realtà meno importante) un punto la cui normale si discosti troppo dalla perpendicolare al piano, identificandolo come punto non affidabile, quindi andiamo ad operare un filtro sulle normali, settando a zero l'affidabilità di un punto che non superi una certa soglia. Attribuiremo poi un valore di affidabilità direttamente proporzionale al valore della normale, infatti più il valore si avvicina ad 1, la normale perpendicolare, più il punto è stato ripreso frontalmente e più siamo sicuri che l'informazione che riceviamo sia effettivamente giusta (quindi affidabile)

$$R_n = \frac{1}{3}\,(n_p * n_{cam}) \qquad\qquad R_n \in \left[0, \frac{1}{3}\right]$$

4.1.2 La luminanza

Viene calcolata componente R_L in funzione della Luminanza del punto.

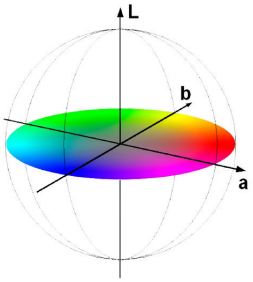

Fig. 4.1: Cielab color space

A questo punto risulta doverosa una piccola premessa sullo spazio di colore utilizzato: le operazioni sul colore vengono effettuate nello spazio di colore Cielab, in cui i canonici tre valori RGB (Red, Green e Blue) vengono convertiti in L (lightness), a e b, due valori di colore. La differenza fondamentale consiste nel fatto che matematicamente la somiglianza tra due colori è direttamente proporzionale ai valori dei due parametri a e b, mentre questo non vale per lo spazio RGB, in cui due colori simili possono avere valori diversi.
Questo rende molto più facile l'operazione matematica di confronto tra due colori e la loro relativa quantificazione di uguaglianza/diversità. La luminanza è il terzo valore che porta più informazioni sulla luce e sulla chiarezza dell'immagine (una zona bianca avrà alta luminanza, una zona nera avrà bassa luminanza), ma in generale un alto valore di L significa che nel punto c'è una grande quantità di luce il che, soprattutto sulle superfici altamente riflettente, è sinonimo di presenza di rumore nel segnale che arriva alla camera, quindi di inaffidabilità. Queste semplici considerazioni, ci portano a formulare una variazione dell'affidabilità legata linearmente, ma con una proporzionalità inversa, alla misura di luminanza del punto; inoltre si è deciso di inserire una certa soglia di partenza (normalmente $L \in [0, 100]$, ma nel progetto è stata normalizza a $[0, 255]$ come i valori RGB) in modo da evitare che un punto scuro venga sottopesato solo per il suo colore, e il risultato è:

$$R_L = \begin{cases} \frac{1}{3}, & \text{con } \ 0 < L \leq 200 \\ \frac{1}{3}\frac{(255-L)}{55}, & \text{con } \ 200 < L < 255 \end{cases} \qquad R_L \in \left[0, \frac{1}{3}\right]$$

4.1.3 La varianza locale

Utilizziamo come terzo parametro per la nostra funzione la varianza locale del punto rispetto ai suoi simili adiacenti. Consideriamo la matrice di lato 5, sufficiente per ottenere una buona valutazione abbastanza resistente al rumore, ma non troppo per evitare inutili appesantimenti dell'algoritmo, in cui il nostro punto risulta centrato, e su quella calcoliamo la varianza. In questo caso la varianza, applicata sulle indicazioni di colore del punto (questo ci è permesso dalla conversione del colore nello spazio Cielab) ci fornisce una

misura di quanto il punto sia omogeneo rispetto ai punti che lo circondano (se si trova quindi al centro di una zona di colore omogeneo o se sia al confine dove due zone di colore diverso si incontrano) e quindi regoliamo la nostra affidabilità con una proporzionalità inversa rispetto al valore di varianza che otterremo, in quanto è più probabile che un punto che si avvicina ai suoi vicini sia effettivamente del colore proposto, quindi chiamando σ^2 la varianza locale del punto, definiamo:

$$R_\sigma = \frac{1}{3}\left(\frac{1}{1+\sigma^2}\right) \qquad\qquad R_\sigma \in \left[\frac{1}{1+\sqrt{255}}, \frac{1}{3}\right]$$

Capitolo 5

La fusione delle informazioni di colore

La parte più significativa del lavoro è stata svolta nella fusione ed ha visibilità nell'elaborazione del modello finale, quando viene effettivamente calcolato il colore da assegnare ad ogni punto.
Una volta progettato l'algoritmo, esso è stato implementato nella parte relativa alla fusione, andando a modificare il calcolo del colore da assegnare al punto: per costruire la nostra valutazione su solide basi matematiche è stata utilizzata la già citata Point Cloud Library attraverso un algoritmo che semplifica la trattazione e la ricerca su grandi moli di dati. Il procedimento è molto complesso e può influire negativamente sulla velocità di calcolo del programma, ma è necessario per arrivare ad un'immagine più nitida e corretta.

5.1 Statistical Outlier Removal

Le nuvole di punti costituiscono un pilastro fondamentale del progetto, ma costituiscono anche un peso in quanto a dimensioni che influiscono a livello computazionale. Per ovviare a questi problemi e per costruire un ulteriore filtro al rumore nella fusione delle nuvole viene applicata una rimozione dei punti seguendo un criterio statistico ben preciso, ovvero costruendo una Gaussiana delle distanze tra ogni punto ed eliminando tutti i punti troppo distanti (generalmente dovuti ad errori o rumore del segnale) o troppo vicino (operando in questo caso quindi un filtro sulla densità di punti) permettendo di avere un modello maggiormente uniforme e omogeneo.

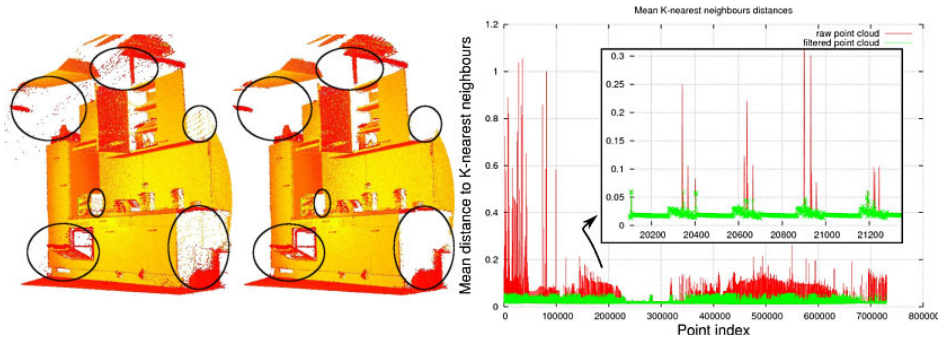

Fig. 5.1: Esempio grafico di Statistical Outlier Removal

Si è pensato quindi di andare ad utilizzare tutti questi punti che vengono rimossi ad ogni esecuzione del filtro statistico: sebbene probabilmente i punti dovuti ad errore o al rumore non contengano informazioni significative, è molto probabile che i punti molto vicini eliminati dal filtro di densità portassero informazioni sul colore molto simili a quelle dei punti sopravvissuti al filtro: quindi viene rioperato il filtro, questa volta però salvando su un'apposita struttura dati soltanto i punti sporchi, cioè quelli cancellati.

5.2 KdTree

Viene quindi implementato un KdTree, una struttura ad albero, sul modello incrementale delle nuvole di punti che vengono di volta in volta aggiunte, che permette la ricerca in tempi asintoticamente molto brevi.

Tale scelta viene pensata in modo da cercare di non appesantire ulteriormente l'elaborazione dei dati, in quanto il computer dovrà lavorare su quantità elevatissime di dati quando si iniziano a utilizzare un buon numero di acquisizioni. Sfruttando appunto questa struttura, si compie una ricerca all'interno della struttura ad albero rappresentante il modello pulito dagli outlier in cui ogni outlier appunto trova il suo più vicino corrispondente. In questo modo i punti cancellati non vengono completamente persi, ma alcune loro informazioni (effettivamente vengono salvate solo le infor-

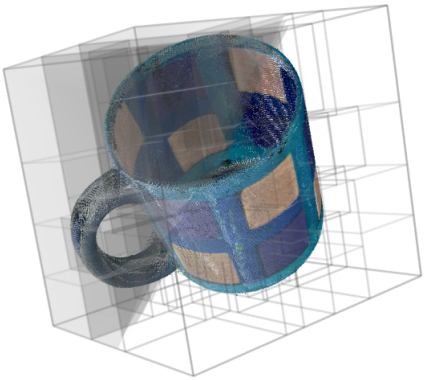

Fig. 5.2: Esempio di kdtree su una tazza

mazioni di colore e la loro affidabilità, le coordinate e le normali vengono perse) vengono conservate e sono vincolate al punto a loro più vicino, consentendogli l'accesso a più dati, senza appesantire il punto in sè. Alla fine del procedimento in cui sono state aggiunte tutte le nuvole, ogni punto sarà quindi legato a tutti i punti cancellati al quale era più vicino, potendo così disporre di una quantità maggiore (e più efficace) di informazioni, in quanto è lecito pensare che in punti estremamente vicini il colore sia, se non estremamente, comunque in maniera significativa, simile.

5.3 Calcolo finale del colore

La parte conclusiva è senz'altro la parte più importante, che va a concludere il lavoro e va ad unire e creare un collegamento tra entrambe le fasi del lavoro: infatti viene operata la scelta finale del colore per ogni punto (il colore che poi si vede effettivamente nel modello tridimensionale) e la scelta si compie con un criterio che è stato già discusso in precedenza, affidandoci cioè a quanto l'informazione di colore può effettivamente essere credibile, quindi come intuitivamente si può anche pensare, grazie alla misura di affidabilità preparata appositamente nella prima parte. La scelta viene operato tenendo conto di tutte le informazioni di cui possiamo disporre, grazie all'applicazione dell'algoritmo sopracitato, sfruttando quindi il colore del punto e tutti i colori dei punti cancellati durante l'esecuzione del programma. Per arrivare ad una scelta che riesca a soddisfare i requisiti di precisione e robustezza al rumore si è pensato di utilizzare una media pesata appunto sull'affidabilità di ogni colore preso dai punti cancellati, in modo che colori poco affidabili contribuiscano in maniera minore ed appunto colori magari erroneamente acquisiti vengano ammortizzati dalle maggiori informazioni analizzate e dal loro peso che in teoria, essendo dato dall'affidabilità, dovrebbe essere minore. Chiamando c il colore (che sarà dato dai tre valori canonici RGB), denotando con v i punti cancellati associati ad ogni punto e contenuti in un apposito vettore V, con colore c_v e affidabilità R_v:

$$c = \frac{\left(\sum_{v \in V} c_v R_v\right) + c_p w_p}{\left(\sum_{v \in V} R_v\right) + w_p}$$

Capitolo 6

Risultati Sperimentali

6.1 Risultati generali

I risultati ottenuti nei test in laboratorio mostrano un buon livello di fedeltà nella riproduzione delle scene ricostruite con l'algoritmo proposto.

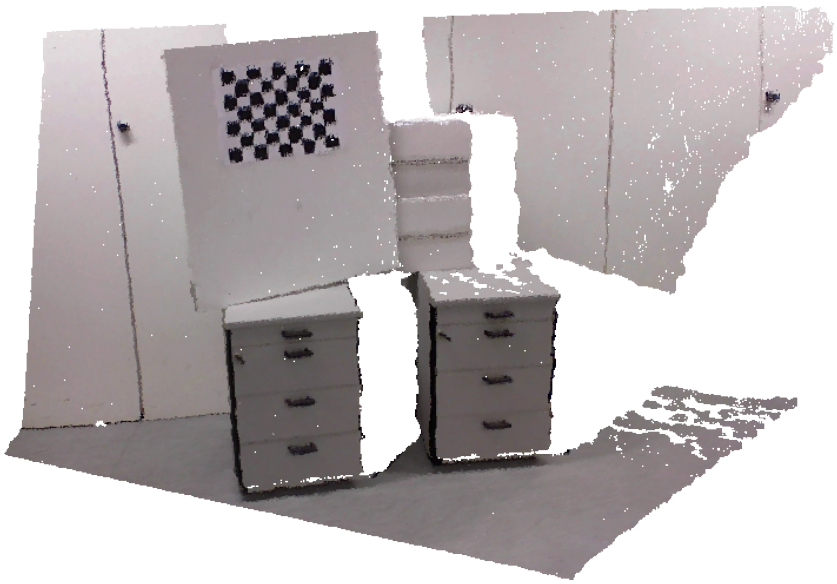

Fig. 6.1: un esempio di modello tridimensionale

Le figure infatti vengono riprodotte con una qualità soddisfacente e a parte qualche imperfezione risultano apprezzabili anche esteticamente.

6.2 Confronto

I risultati per poter avere un riscontro devono essere confrontati con la versione di partenza del progetto. Prendiamo un paio di modelli che sono stati elaborati nei test di laboratorio, dove si possono osservare le modifiche

Fig. 6.2: Modello ricostruito con il progetto di partenza

Nel primo test i risultati non sono evidenti, sebbene si possa notare qualche piccolo miglioramento negli angoli del modello che vengono puliti coprendo alcuni buchi, ma nel complesso la riproduzione non subisce un cambiamento notevole.

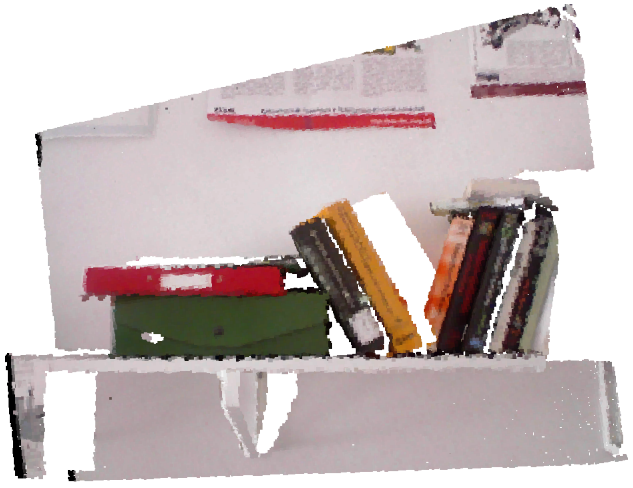

Fig. 6.3: Modello ricostruito con la versione modificata

Fig. 6.4: Modello ricostruito con il progetto di partenza

Il secondo test offre dei risultati migliori, infatti la differenza tra le due versione è abbastanza chiara in questa occasione

Fig. 6.5: Modello ricostruito con la versione modificata

I risultati ricalcano su vari punti le aspettative: non si era infatti alla ricerca di un algoritmo che stravolgesse la riproduzione del modello, né di un algoritmo che né cambiasse il modo di operare, bensì lo scopo delle modifiche era quello di acquisire robustezza al rumore e alla luce (i problemi principali che si affrontano nell'utilizzo dei sensori di cui è stato trattato) e migliorare la valutazione del colore in alcuni punti. Riguardo alla valutazione del colore, il miglioramento per questi semplici test non è apprezzabile a livello visivo,

anche se a livello numerico la differenza tra i valori assunti dai diversi punti non è nulla: giustamente il colore non viene modificato completamente poichè l'algoritmo di partenza riesce a riprodurre il colore in un modo accettabile, ma esso viene solamente migliorato nel dettaglio.
Nel primo test i risultati sono meno evidenti: ciò è dovuto alla mancanza di particolari problemi nell'acquisizione, ed in mancanza di illuminazione o rumore la versione modificata non deve fare nulla di diverso da quella di partenza, se non lavorare sul colore che in scene così piccole non va praticamente ad essere modificato.
Nel secondo test invece i risultati appaiono più nitidi: la luce era presente nella scena in quantità maggiore, e alcuni punti risultano mancanti a causa di rumore nel segnale (o ad errori umani nell'acquisizione) e questi due aspetti sono quelli su cui si è lavorato, quindi si visualizza anche ad occhio un miglioramento, soprattutto nel colore dello sfondo e degli oggetti su di esso, che appaiono più omogenei e più reali.

6.3 Conclusioni

Il lavoro effettuato porta qualche vantaggio in termini di robustezza al rumore e bassa sensibilità alla presenza di luce nella scena da riprodurre, e riesce a migliorare la rappresentazione del colore su sfondi e oggetti di larghe superfici in cui si possono notare sensibili differenze. Il progetto sfrutta molto le risorse del computer: richiede un tempo di esecuzione maggiore rispetto alla versione di partenza, in quanto alcune operazioni sono molto costose in termini computazionali, ma sono necessarie per l'apporto delle modifiche; potranno essere ottimizzate in futuri sviluppi dell'algoritmo.

Appendice A

La personalizzazione del punto di lavoro

Alla base di tutto il lavoro effettuato vi è stato un processo di preparazione ed impostazione del programma in modo da consentire effettivamente l'introduzione delle modifiche progettate. Si è quindi pensato di creare una struttura punto (un diverso tipo di punto per capirci), poiché la versione di partenza del programma utilizzava un punto che conteneva al suo interno solamente le indicazioni di posizione (x, y, z) e quelle di colore (RGB), mentre nella nuova versione vi era bisogno di contenere più informazioni quali le normali del punto e soprattutto la sua affidabilità per effettivamente avere la possibilità di eseguire i passi sopra descritti, quindi utilizzando la PCL è stata creata una nuova struttura che ricalcava la precedente, ma introduceva delle nuove informazioni, risultando più efficace a discapito di una maggior pesantezza in fase di calcolo in quanto la dimensione (in byte) della nuova struttura è inevitabilmente maggiore.

Appendice B

Implementazione attraverso i template

Il lavoro, soprattutto in funzione della nuova struttura che siamo andati a costuire appositamente, ha richiesto una modifica del programma alla base: infatti si è compiuta una completa templatizzazione delle classi costituenti il codice in modo da poter utilizzare un tipo di dato generico che poi sarà esattamente la nostra struttura personalizzata.

B.1 Cos'è un template?

Un template in C++ è uno speciale parametro che viene utilizzato per generalizzare il tipo di oggetto che un metodo dovrà gestire. Garantisce ampia flessibilità al progetto in quanto poi è possibile scegliere e cambiare gli oggetti effettivamente utilizzare nelle classi senza avere problemi di conseguenza: infatti una classe (e i conseguenti metodi) templatizzata è in grado di gestire un tipo di oggetti generico. L'utilizzo di template in grandi progetti inoltre fornisce la possibilità di cambiare il punto in modo semplice ed efficace, potendo cambiare solo una definizione che renderà la modifica effettiva solo in fase di esecuzione (ciò garantisce più libertà nella fase di progettazione).

B.2 Il motivo della scelta

Si capisce quindi l'importanza che questa fase iniziale ha avuto nel lavoro, consentendoci di apportare e di mettere in pratica la personalizzazione del punto fondamentale ai fini degli algoritmi progettati e implementati, altrimenti difficilmente realizzabili a meno di complesse e articolate operazioni che

sarebbero andate inevitabilmente a discapito della complessità temporale del programma in maniera abbastanza pesante.

Appendice C

Gli strumenti di lavoro

Data la grande quantità di lavoro svolta in fase di programmazione in laboratorio, viene effettuata anche una piccola panoramica sugli indispensabili strumenti messi a disposizione dello studente ed effettivamente utilizzati per produrre il lavoro finale, dal software di cui si è potuto usufruire alla libreria, componente fondamentale nell'elaborazione delle nuvole di punti e nell'uso di algoritmi di ricerca e costruzione di strutture ad albero sulle stesse.

C.1 VisualStudio

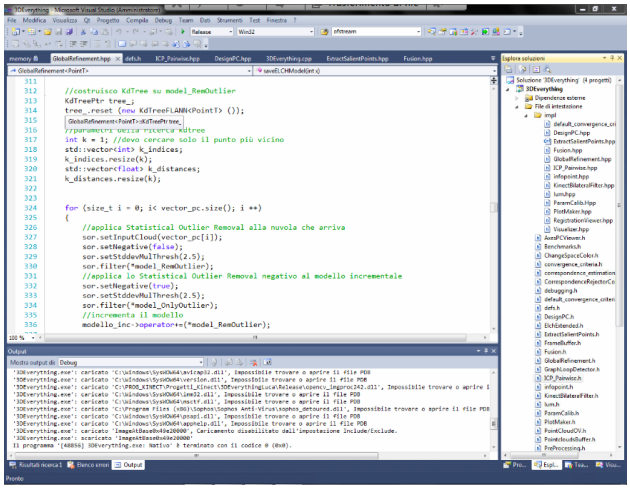

Fig. C.1: screenshot di Visual Studio

Il software della Microsoft è stato scelto come il più efficace e il più adatto per il lavoro. Permette la compilazione del codice, sviluppato interamente

nel linguaggio C++, collegandosi alla libreria PCL, e l'esecuzione tramite un'apposita finestra.

C.2 Il tool di acquisizione

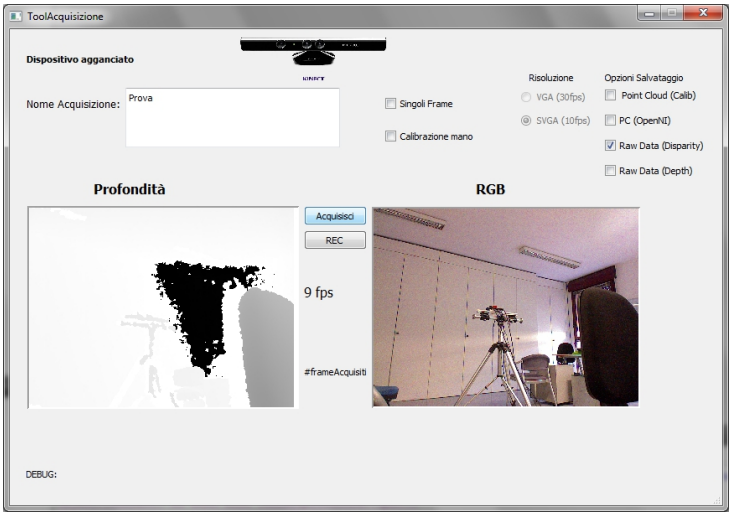

Fig. C.2: screenshot di 3DFusion

3DFusion è un programma sviluppato dal laboratorio che permette tramite un'interfaccia semplice di gestire l'acquisizione con diverse opzioni (singoli frame, continuo) e di salvare i dati nel formato che si preferisce (raw data, disparity map). L'acquisizione della scena tramite il Kinect è stata eseguita attraverso l'utilizzo di questo tool.

C.3 Librerie Utilizzate

La Point Cloud Library è la libreria Open Source più ampia e aggiornata per quanto riguarda la modellizzazione e la ricostruzione di superfici tridimensionali e tutto ciò che è attinente all'argomento [5].

Il progetto sfrutta notevolmente questa libreria che fornisce i metodi necessari per la gestione di nuvole di punti dense o sparse che siano, come l'utilizzo di filtri statistici come lo Statistical Outlier Removal e la costruzione e l'utilizzo di un KdTree, già citati precedentemente.

Fig. C.3: logo della PCL

Bibliografia

[1] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon, *"Kinectfusion: Real-time dense surface mapping and tracking,"* in *Proc. of* IEEE ISMAR, October 2011.

[2] T. Whelan, H. Johannsson, M. Kaess, J. Leonard, and J. McDonald, *"Robust tracking for real-time dense RGB-D mapping with Kintinuous"* MIT, Tech. Rep., Sep 2012.

[3] E. Cappelletto, P. Zanuttigh, G. M. Cortelazzo (2013), *Handheld scanning with 3D cameras*, 2013 MMSP IEEE Paper

[4] C. Dal Mutto, P. Zanuttigh and G. M. Cortelazzo (2013), *Time-of-Flight Cameras and Microsoft KinectTM - A user perspective on technology and applications*, Springer

[5] Asus website, Xtion Pro Live page, *http://www.asus.com/Multimedia/Xtion_PRO_LIVE*

[6] Point Cloud Libraries, *http://pointclouds.org*

[7] A.Fusiello, (2009), *Visione Computazionale - Appunti delle lezioni*, Creative Commons

[8] Reconstructme, *http://reconstructme.net/*

[9] Skanect, *http://skanect.manctl.com/*