POLITECNICO DI TORINO

Laurea Magistrale in Ingegneria del Cinema e dei Mezzi di Comunicazione

Tesi di Laurea Magistrale

Studio e implementazione di tecniche avanzate per l'ottimizzazione della percezione visiva e del senso di profondità nella realtà aumentata per la radiologia interventistica: un'analisi sul sistema Endosight.

Relatori:
Prof. Bottino Andrea
Dott. Strada Francesco

Candidato
Panza Gianluca
Numero di Matricola: s290135

Anno accademico 2022/2023

Indice

SOMMARIO

All'interno del documento che segue si esplora l'utilizzo della realtà aumentata (AR) nel campo della radiologia interventistica, con un focus particolare sull'ottimizzazione della percezione visiva e del senso di profondità di uno specifico tool digitale utilizzato in questo ambito: Endosight. L'obiettivo principale è migliorare l'efficacia e la sicurezza delle procedure mediche relative alla radiologia interventistica, attraverso l'implementazione di differenti tecniche rendering e la creazione di interfacce utente intuitive.

Dopo un'introduzione generale sull'AR e le sue applicazioni nel settore medico, la tesi analizza lo stato dell'arte della radiologia interventistica e le tecnologie in uso, evidenziando vantaggi e svantaggi di alcuni sistemi utilizzati attualmente. Si esamina inoltre il sistema Endosight, l'unica soluzione innovativa per questa specifica branca della radiologia, che offre ausilio agli operatori sanitari con sistemi di realtà aumentata. Viene quindi analizzata la sua pipeline di rendering e le sue caratteristiche tecniche, riportandone le principali criticità.

All'interno della sezione "Materiali e Metodi" vengono dettagliate tutte le soluzioni adottate rispetto alla preesistente pipeline di render di Endosight, basata sul game engine Unity. Questa sezione descrive quindi l'utilizzo di una pipeline grafica URP, la creazione di shaders realistici tramite triplanar projection e l'implementazione di un algoritmo che permette di visualizzare le mani dell'operatore, le quali normalmente sarebbero occluse dalle mesh 3D delle strutture anatomiche compositate sul video.

A tal fine, viene illustrata la creazione di una libreria DepthUtility in C++, che sfrutta gli algoritmi di stereo matching di OpenCV, alcune tecniche di filtraggio dei dati per il calcolo della xyzMap (una mappa che contiene le informazioni di posizione dei pixel della scena, decritti nel sistema di riferimento della camera) e il suo utilizzo in Unity.

Successivamente vengono riportate le logiche implementate all'interno dell'ambiente Unity, grazie alle quali si è riusciti a generare una mesh della mano a partire dai dati di triangolazione estratti dalla libreria sopra citata.

Sommario

La mesh della mano, all'interno del sistema sviluppato, si comporta come classificatore dei pixel dell'immagine rasterizzata, riuscendo a compensare gli errori di occlusione sopra citati, attraverso la gestione dello stencil buffer messo a diposizione da Unity.

La tesi presenta quindi i risultati quantitativi, analizzando la bontà degli algoritmi finalizzati alla visualizzazione delle mani dell'utente.

A tal fine vengono presentati dei grafici Recall and Precision, i quali mostrano l'efficacia del classificatore, con un valore massimo di F-Measure pari al 91%, raggiunto a seguito del perfezionamento dei parametri passati ai metodi di Open CV.

Successivamente si discutono i limiti dell'applicazione, riguardanti principalmente la complessità computazionale e la perdita delle caratteristiche real time del sistema, a seguito delle modifiche introdotte.

In conclusione, vengono presentate alcune ipotesi di miglioramento del sistema sviluppato in questo progetto.

1 INTRODUZIONE

In questa sezione della tesi vengono affrontati tutti i concetti generali, che permettono di delineare un quadro propedeutico alla comprensione delle nozioni riportate all'interno del documento.

1.1 INTRODUZIONE ALLA REALTÀ AUMENTATA (AR)

La realtà aumentata (AR) è una tecnologia che sovrappone informazioni virtuali, come immagini, testo o modelli 3D, all'ambiente reale dell'utente, arricchendo così la percezione del mondo circostante.

L'AR rientra all'interno del più ampio concetto di mixed reality (MR), ovvero il modello che, secondo Paul Milgram[1], descrive la fusione/interazione del mondo reale con quello virtuale, all'interno del quale l'utente può interagire sia con gli oggetti reali che con quelli virtuali.

In un ambiente di Mixed Reality, gli elementi virtuali possono essere manipolati e influenzati dal mondo reale e viceversa.

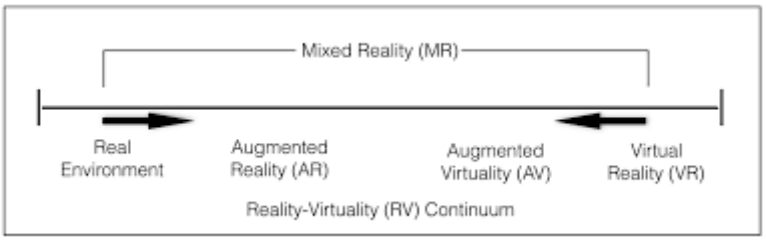

Figura 1: Reality-Virtuality continuum (Milgram & Kishino, 1994).

[1] In Milgram, P., Takemura, H., Utsumi, A., and Kishino, F. (1994). Augmented reality: a class of displays on the reality-virtuality continuum. Proc. SPIE 2351, 282–292. doi: 10.1117/12.197321

In *Classification Space for Augmented Surgery, an Augmented Reality Case Study* [2] viene proposta la classificazione OPAS, ossia un modello per classificare sistemi interattivi, tra cui anche quelli di realtà aumentata.

Si compone di quattro elementi principali:
Oggetto (O), Persona (P), Adattatore (A) e Sistema (S).

1. **Oggetto** (O): Elementi reali come un trapano, una penna o un foglio di carta, manipolati dall'utente o da un robot per svolgere un compito. Anche le persone possono essere considerate oggetti, purché non interagiscano direttamente con il sistema.
2. **Persona** (P): L'utente che interagisce con il sistema.
3. **Adattatore** (A): Collega il mondo reale ("*mondo degli atomi*") al mondo virtuale ("*mondo dei bit*"). Esempi di adattatori sono il mouse, la tastiera, lo schermo, i localizzatori ultrasonici o elettromagnetici.
4. **Sistema** (S): Componente informatico in grado di memorizzare, recuperare e trasformare dati, appartenente al mondo virtuale o sintetico.

La principale difficoltà è distinguere tra adattatori (A) e oggetti (O). Ad esempio, un mouse è un adattatore poiché trasforma i movimenti fisici dell'utente in movimenti nel mondo virtuale mostrato sullo schermo. Tuttavia, se l'utente utilizza il mouse come fermacarte, il mouse diventa un oggetto.

All'interno del modello i componenti OPAS interagiscono scambiando informazioni tra loro in modo unidirezionale, graficamente rappresentabili come segue:

Analizziamo come esempio uno strumento chirurgico, la cui posizione viene tracciata tramite l'ausilio di un localizzatore; in questo caso il localizzatore si comporterà da adattatore (A) per l'oggetto (O) strumento chirurgico.

[2] Emmanuel Dubois, Laurence Nigay, Jocelyne Troccaz, Olivier Chavanon, Lionel Carrat, Classification Space for Augmented Surgery, an Augmented Reality Case Study, A. Sasse and C. Johnson (eds.), *Proceedings of Interact'99, IOS Press*, (1999), Edinburgh (UK), p. 353-359.

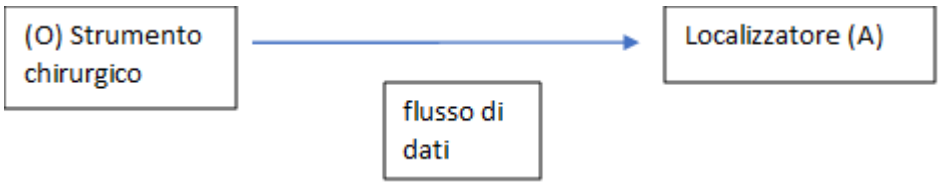

Figura 2

A questo punto è importante specificare se l'utente del sistema sta eseguendo un'azione al fine di modificare il mondo reale o al fine di modificare lo stato delle informazioni mantenute dal computer. Pertanto, si può considerare il compito di (P) in termini di operazioni con target su:

1. **Mondo reale** (interazione persona-mondo reale)
2. **Computer** (interazione persona-computer)

Questi concetti sono stati presentati da Milgram e Kishino come "realtà aumentata" (1) e "virtualità aumentata" (2). Nella realtà aumentata, l'interazione con il mondo reale è migliorata dal calcolatore, mentre nella virtualità aumentata, l'interazione con il computer è migliorata dagli oggetti e dalle azioni nel mondo reale.

L' "aumento" offerto dal sistema AR può assumere diverse forme. Ad esempio, può essere migliorata l'esecuzione di un certo task e/o la percezione da parte dell'utente del mondo reale.

Attraverso l'uso della classificazione OPAS, gli sviluppatori e i ricercatori possono esaminare i diversi aspetti dell'interazione tra utenti e sistemi, tra cui l'obiettivo dei compiti supportati dal sistema, il tipo di "aumento" offerto e la relazione tra i componenti del sistema.

In sintesi, la classificazione OPAS fornisce un quadro concettuale utile per analizzare e progettare sistemi interattivi e realtà aumentata (e non solo), facilitando la comprensione delle diverse sfaccettature dell'interazione tra utente e sistema e promuovendo lo sviluppo di soluzioni più efficaci ed efficienti.

Per implementare la realtà aumentata, sono necessari diversi componenti chiave, tra cui:

1. **Dispositivi di visualizzazione**: Gli elementi digitali possono essere sovrapposti all'ambiente reale utilizzando diversi dispositivi di visualizzazione, come smartphone, tablet, visori e proiettori. I visori AR, come Microsoft HoloLens o Google Glass, sono esempi di dispositivi indossabili che permettono un'esperienza AR più immersiva.

2. **Tecnologie di tracking e localizzazione**: Per garantire la corretta sovrapposizione degli elementi virtuali sull'ambiente reale, è fondamentale utilizzare sistemi di tracking e localizzazione accurati. Questi sistemi possono utilizzare bussole, giroscopi, accelerometri, telecamere, GPS e altri dispositivi per rilevare la posizione e l'orientamento dell'utente e degli oggetti nell'ambiente.

3. **Elaborazione dei dati e rendering**: L'elaborazione dei dati e il rendering degli elementi digitali sono compiti essenziali per una buona esperienza AR. Questi processi possono essere eseguiti da dispositivi mobili, computer o server remoti, a seconda della potenza di calcolo necessaria e delle specifiche dell'applicazione AR.

4. **Interazione e interfaccia utente**: L'interazione tra l'utente e gli elementi virtuali nella realtà aumentata può avvenire attraverso diversi metodi, come il tocco, la voce, i gesti o dispositivi di input specializzati. Le interfacce utente AR devono essere progettate per essere intuitive e facilmente comprensibili, permettendo all'utente di interagire con gli elementi digitali in modo naturale e fluido.

La realtà aumentata ha trovato applicazione in numerosi settori, tra cui intrattenimento, istruzione, ingegneria, architettura e medicina.

Nel contesto medico, l'AR offre potenziali benefici per la formazione, la diagnosi e la pianificazione delle procedure chirurgiche, oltre a migliorare la precisione e l'efficacia degli interventi medici, grazie all'accesso a informazioni aggiuntive, in tempo reale durante le procedure.

1.2 REALTÀ AUMENTATA NEL SETTORE MEDICO SANITARIO

L'adozione di tecnologie di realtà aumentata ha avuto un grande impatto sul settore medico sanitario, offrendo strumenti innovativi che migliorano la precisione, l'efficacia e la sicurezza degli interventi clinici. L'AR ha migliorato anche la visualizzazione di dati aggregati e la possibilità, da parte di specialisti, di partecipare a distanza ad alcuni interventi chirurgici. Diverse tecnologie di AR sono state sviluppate e applicate in vari ambiti della medicina, tra cui la formazione, la diagnosi, la pianificazione e l'esecuzione di procedure chirurgiche.

Di seguito un breve excursus su alcune tecnologie AR utilizzate oggi in ambito medicale:

1. AccuVein: è un dispositivo di realtà aumentata che consente di visualizzare in tempo reale le vene sotto la pelle del paziente. Grazie all'uso di una tecnologia di imaging a infrarossi, questo strumento aiuta i medici a individuare con precisione le vene per prelievi di sangue, iniezioni e cateterismi, riducendo il rischio di complicanze e migliorando l'esperienza del paziente. Link: https://www.accuvein.com/

2. EchoPixel: è una tecnologia di AR che permette ai medici di visualizzare e interagire con immagini mediche 3D, come tomografie computerizzate (TC) e risonanze magnetiche (RM), in uno spazio tridimensionale. Questo strumento migliora la comprensione delle strutture anatomiche complesse, facilita la diagnosi e la pianificazione chirurgica, e permette una formazione più efficace per i medici in formazione. Link: https://echopixeltech.com/

3. Proximie: è una piattaforma di AR che consente ai chirurghi di collaborare e guidare interventi chirurgici a distanza. Attraverso una connessione Internet sicura, i chirurghi possono condividere la loro visione in tempo reale del campo operatorio, sovrapporre immagini virtuali e fornire indicazioni utilizzando gesti e annotazioni. Questo strumento può migliorare la qualità delle procedure chirurgiche, fornire supporto nella formazione e consentire la collaborazione tra specialisti in tutto il mondo. Link: https://www.proximie.com

4. Touch Surgery: è un'applicazione di AR che fornisce simulazioni chirurgiche interattive per aiutare i medici a perfezionare le loro competenze tecniche e a prepararsi per interventi specifici. L'app presenta una vasta gamma di procedure chirurgiche, con dettagliate istruzioni passo-passo e una rappresentazione visuale 3D dell'anatomia del paziente. Touch Surgery è uno strumento di formazione utile per i medici in formazione e un valido supporto per i chirurghi esperti nella pianificazione di interventi complessi. Link: https://live.touchsurgery.com/

1.3 RADIOLOGIA INTERVENTISTICA

"La radiologia interventistica è una specializzazione medica che prevede l'esecuzione di una serie di procedure di imaging per ottenere immagini dell'interno del corpo. Il radiologo interventista interpreta attentamente queste immagini per diagnosticare lesioni e malattie e per eseguire una serie di procedure mediche interventistiche [3] ".

In questo ambito vengono utilizzate tecniche di imaging come raggi X, risonanza magnetica (MRI), fluoroscopia (una procedura a raggi X che consente di vedere gli organi interni in movimento), tomografia computerizzata (TC) ed ecografie.

I radiologi interventisti eseguono un'ampia gamma di procedure, come il trattamento dei tumori, la biopsia degli organi o il posizionamento di stent, inserendo nel corpo piccoli strumenti e sottili tubi di plastica (cateteri) attraverso un'arteria o una vena. Le immagini pre-acquisite vengono utilizzate per guidare i cateteri e gli altri strumenti specifici, nell'area esatta in cui deve essere eseguita la procedura o il trattamento. Ciò riduce la necessità di interventi chirurgici tradizionali (aperti) o a cielo coperto (laparoscopici), poiché il trattamento può essere somministrato tramite un piccolo tubo di plastica delle dimensioni di una cannuccia.

I continui progressi nella tecnologia stanno ampliando la gamma di condizioni che possono essere trattate dalla radiologia interventistica.

[3] Definizione presa da: https://www.insideradiology.com.au/interventional-radiology/

I radiologi interventisti eseguono una vasta gamma di procedure, tra cui:

- Angioplastica e inserimento di stent
- Paracentesi ascitica
- Drenaggio biliare
- Iniezione bursale
- Stent carotideo
- Ecografia e iniezione del tunnel carpale
- Iniezione di corticosteroidi nella radice nervosa cervicale guidata dall'immagine
- Biopsia epatica guidata dall'immagine
- Iniezione epidurale lombare di corticosteroidi guidata dall'immagine
- Iniezione di corticosteroidi nella radice nervosa lombare guidata dall'immagine
- Filtri della vena cava inferiore
- Iniezione articolare
- Nefrostomia
- Aspirazione pleurica
- Ablazione per radiofrequenza
- Trattamento endovascolare del vasospasmo SAH
- Terapia radiante interna selettiva [SIRT]: SIR-Spheres®
- Embolizzazione del midollo spinale (AVM/DAVF)
- Aspirazione fine dell'ago tiroideo (FNA)
- Chemoembolizzazione transarteriosa (TACE)
- Embolizzazione delle fibromi uterini
- Ablazione delle vene varicose
- Dispositivi di chiusura vascolare
- Accesso venoso
- Vertebroplastica

1.4 DOMINIO APPLICATIVO DI ENDOSIGHT

La tecnologia Endosight può essere classificata all'interno del concetto di CAMI (Computer-Aided Medical Interventions).

I CAMI sono sistemi di supporto informativo e di controllo che integrano le tecnologie informatiche e di imaging nella pratica medica. Questi sistemi migliorano la precisione e la sicurezza delle procedure chirurgiche, riducono i rischi per i pazienti e consentono di eseguire interventi meno invasivi.

I CAMI possono essere suddivisi in tre categorie principali [4]:

1. sistemi di pianificazione preoperatoria, che aiutano a definire la strategia chirurgica;
2. sistemi per guidare l'intervento, che supportano il chirurgo durante la procedura;
3. sistemi di valutazione post-operatoria, che permettono di analizzare i risultati dell'intervento.

I sistemi CAMI rappresentano un'importante sfida nello sviluppo di interfacce utente, poiché richiedono un'interazione efficace tra l'utente (il medico) e il sistema stesso.

Questa interazione può essere influenzata da diversi fattori, come l'affinità dell'utente con l'utilizzo di strumenti digitali, le caratteristiche del sistema, le specifiche del contesto in cui viene utilizzato e la bontà dell'implementazione dell'interfaccia stessa. Pertanto, è fondamentale progettare interfacce utente intuitive ed efficienti per queste tipologie di sistemi.

In particolare, Endosight rientra all'interno dei primi due punti della classificazione CAMI, risultando un valido strumento per la pianificazione preoperatoria e uno strumento di supporto durante l'intervento.

Nel paragrafo successivo, analizzeremo più nello specifico il sistema Endosight.

[4] Emmanuel Dubois, Laurence Nigay, Jocelyne Troccaz, Olivier Chavanon, Lionel Carrat, Classification Space for Augmented Surgery, an Augmented Reality Case Study, A. Sasse and C. Johnson (eds.), *Proceedings of Interact'99, IOS Press*, (1999), Edinburgh (UK), p. 353-359.

1.5 IL SISTEMA ENDOSIGHT

Endosight è un software di navigazione in realtà aumentata e navigazione virtuale, progettato per assistere i radiologi interventistici per svolgere operazioni chirurgiche poco invasive su organi e strutture anatomiche nel tronco.

Il suo funzionamento si basa inizialmente sulla realizzazione di una tomografia assiale computerizzata (TAC) del paziente in analisi.

Figura 3: Esecuzione di una Tomografia Assiale Computerizzata (TAC)

Successivamente, attraverso l'interfaccia del software, il medico può identificare il bersaglio dell'intervento (che chiameremo "lesione") sulla TAC e pianificare l'inserimento dell'ago di ablazione, come avviene per le procedure di chirurgia interventistica standard.

Endosight interviene in ausilio del chirurgo già in questa fase preoperatoria; infatti, questo strumento rende possibile la visualizzazione una rappresentazione tridimensionale delle strutture anatomiche del paziente.

Esse vengono ricostruite tramite segmentazione delle immagini TAC e triangolazione (intesa come generazione di una mesh poligonale a facce triangolari) delle nuvole di punti ottenute.

La segmentazione delle immagini è importante per ottenere una suddivisione semantica dei voxel[5] delle immagini DICOM[6]. Dal contorno di ogni volume segmentato viene costruita una mesh indipendente, che rappresenta la singola struttura anatomica; è quindi possibile intervenire sulle singole strutture tramite l'interfaccia. È possibile, ad esempio disattivare la renderizzazione di una parte di esse.

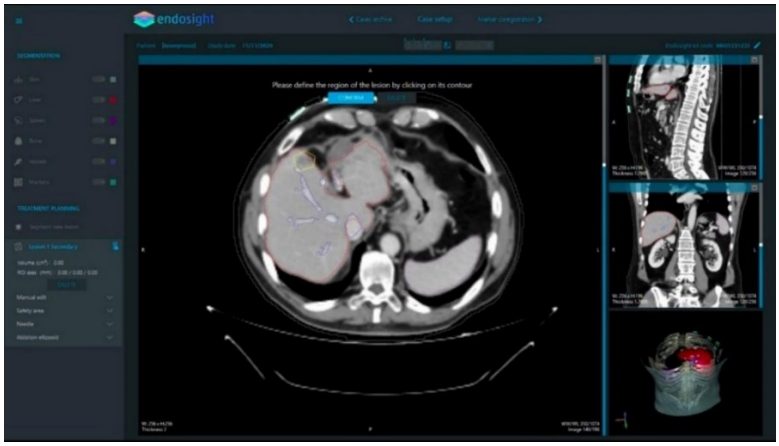

Figura 4: Interfaccia per il planning preoperatorio di Endosight

Il chirurgo è quindi in grado di pianificare l'intervento, evitando di lesionare sezioni anatomiche vitali o di intralcio ai fini dell'operazione stessa, come è possibile osservare in Figura 5.

[5] o "volume pixel", è l'equivalente tridimensionale di un pixel. Mentre un pixel rappresenta un punto di colore o intensità in un'immagine bidimensionale, un voxel rappresenta un volume di colore o intensità in uno spazio tridimensionale.

[6] Digital Imaging and Communications in Medicine, è uno standard che regola la trasmissione, lo scambio, la condivisione, l'archiviazione e la visualizzazione di informazioni di imaging biomedico e relative informazioni associate.

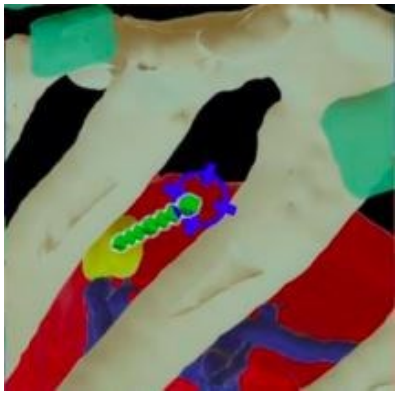

Figura 5: pianificazione della traiettoria di ingresso dell'ago di ablazione

Durante la fase operatoria, Endosight, sfruttando il proprio sistema di realtà aumentata, sovrappone correttamente al tronco del paziente le sue strutture anatomiche, ricostruite dalle scansioni TAC.

Vengono contemporaneamente virtualizzati gli strumenti medicali (come aghi o sonde). Le immagini risultanti vengono mostrate al medico su un monitor o su occhiali di realtà aumentata, a seconda delle preferenze.

Figura 6: Modalità visore, si può notare la stereo camera che permette la visione "see through"

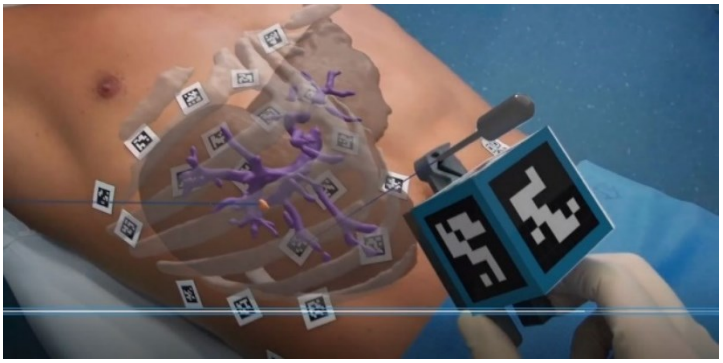

Figura 7:POV (Point Of View) del chirurgo in modalità visore. [l'immagine in questione non rispecchia l'originale visualizzazione AR di Endosight, ma è un'immagine renderizzata offline, con scopo illustrativo]

Per visualizzare correttamente l'ago e le strutture anatomiche del paziente nelle loro posizioni spaziali corrette, è necessario un algoritmo di tracking e alcuni punti di riferimento fisici, rappresentati dai markers ARUCO posizionati sull'addome del paziente, come rappresentato in Figura 8.

Figura 8: ARUCO markers per la stima della posa delle strutture anatomiche in computer grafica

2 STATO DELL'ARTE

In questa sezione viene analizzato lo stato dell'arte delle tecnologie digitali utilizzate nel settore della radiologia interventistica, soffermandosi poi sul sistema Endosight: qui verranno descritte sommariamente le logiche di funzionamento di questo strumento, in modo tale da fornire le nozioni necessarie a comprendere le motivazioni delle scelte conseguite durante lo sviluppo del progetto di tesi. Infine, lo stato di sviluppo della pipeline di rendering di Endosight sarà analizzato nello specifico, evidenziando le principali criticità e le soluzioni ad esse proposte (nonché implementate).

2.1 ASSISTENZA DIGITALE PER LA RADIOLOGIA INTERVENTISTICA

Nel campo della radiologia interventista e dell'ablazione esistono differenti aziende che offrono soluzioni digitali di ausilio al radiologo interventista; ne prenderemo in analisi due:

1. **CT Navigation di Imactis**: si tratta di una soluzione che facilita la pianificazione e l'esecuzione di procedure radiologiche guidate da immagini. Utilizza le immagini TC (Tomografia Computerizzata) per fornire una guida in tempo reale durante l'intervento, con l'obiettivo di migliorare la precisione e la sicurezza del processo.

   Sebbene CT Navigation possa ottimizzare il flusso di lavoro dei radiologi, di fatto, si tratta di un sistema che permette soltanto di muoversi all'interno delle slice delle immagini diagnostiche, attraverso il tracciamento del tool di ablazione.

   CT Navigation non fa quindi uso di realtà aumentata per migliorare ulteriormente la visualizzazione delle strutture anatomiche, né prevede una visualizzazione di un modello tridimensionale delle strutture anatomiche, che permetterebbe di navigare le stesse in uno spazio tridimensionale, senza essere vincolati alla visualizzazione messa a disposizione dalle immagini diagnostiche.

Figura 9: Procedura di pianificazione preoperatoria con CT Navigation; il medico pre-visualizza la traiettoria del tool di ablazione navigando tra le slice dell'immagine diagnostica.

2. **CAS-One IR di Cascination**: è un sistema di navigazione per la radiologia interventista che impiega la tecnologia di fusione di immagini per assistere i medici durante le procedure di ablazione tumorale. Il sistema combina diverse modalità di imaging, come l'ecografia, la tomografia computerizzata (TC) e la risonanza magnetica (RM), per fornire una rappresentazione 3D precisa della zona di intervento. CAS-One IR consente una pianificazione accurata della procedura e il controllo dei risultati dell'ablazione. Tuttavia, anche questo sistema non utilizza la realtà aumentata per fornire un'esperienza visiva più avanzata e un maggiore senso di profondità durante l'intervento.

Figura 10: Procedura operatoria mediante CAS-One; il medico verifica l'effettivo raggiungimento dell'area interessata mediante la duplice visualizzazione DICOM e 3D, navigabile, quest'ultima nelle tre dimensioni.

In sintesi, sia CT Navigation di Imactis che CAS-One IR di Cascination offrono soluzioni utili per assistere i radiologi nelle procedure interventiste e di ablazione. Tuttavia, a differenza di Endosight, nessuna di queste tecnologie implementa la realtà aumentata come strumento di ausilio all'intervento.

La realtà aumentata, come utilizzata in Endosight, offre ulteriori vantaggi nella visualizzazione delle strutture anatomiche e nella percezione del senso di profondità, potenzialmente migliorando ulteriormente la sicurezza e l'efficacia degli interventi radiologici.

2.2 IL SISTEMA DI REALTÀ AUMENTATA DI ENDOSIGHT

2.2.1 Hardware utilizzato

Per analizzare Endosight a livello hardware dobbiamo distinguere tra le due modalità di utilizzo possibile:

1. **Modalità "see through"** tramite headset.
2. **Modalità desktop**.

La modalità see through (1) è permessa da un headset indossabile dal chirurgo.

L'headset presenta, installata sulla sua parte frontale, una stereo camera, come mostrato in Figura 6. Quest'ultima si muove solidalmente con la testa dell'utente, inquadrando la porzione di spazio che verrebbe "inquadrata" dagli occhi dell'utente.

La stereo camera genera uno stream video stereoscopico che viene visualizzato sui monitor dell'headset, insieme alle mesh sintetiche delle strutture anatomiche.

Questa modalità permette un'esperienza maggiormente immersiva rispetto alla modalità desktop, in quanto l'utente è in grado di muovere il punto di osservazione, semplicemente muovendo la testa; inoltre, la visione stereoscopica, ottenibile tramite headset, è in grado di restituire il senso di profondità della scena.

Per capire come una visione stereoscopica permette di ottenere l'informazione di profondità, si può fare riferimento al paragrafo "Triangolazione: fondamenti teorici" presente nel capitolo Materiali e Metodi di questa tesi.

La modalità see through permette potenzialmente di migliorare la precisione con cui l'utente opera durante l'operazione chirurgica, proprio grazie ad una maggiore percezione della profondità.

La modalità desktop (2) permette la visualizzazione in realtà aumentata dall'interfaccia desktop di Endosight. In questa modalità la stereo camera è fissa nello spazio (riferimento alla Figura 11, sulla destra) ed inquadra il torso del paziente.

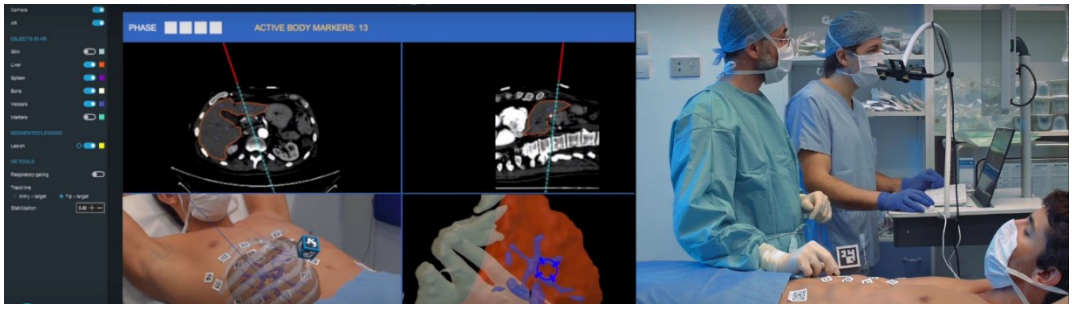

Figura 11: Endosight in modalità desktop. La stereo camera è posizionata su uno stand e non sull'headset indossato dal chirurgo.

In questa modalità si perdono i vantaggi della visione stereoscopica e la metafora di navigazione della scena muta, diventando di fatto una camera fissa.

L'unico vantaggio di questa modalità consiste in una modalità di visualizzazione aggiuntiva: il punto di vista dell'ago (che si può osservare nell'angolo in basso a destra dell'interfaccia in Figura 11).

2.2.2 Software e logiche di funzionamento

Il sistema di realtà aumentata di Endosight si basa sul game engine Unity, il quale permette di gestire il rendering real time di oggetti sintetici e l'implementazione di alcune logiche specifiche tramite script.

Osserviamo in estrema sintesi come viene gestita la realtà aumentata all'interno di Unity:

Lo stream video proveniente dalle due camere viene rettificato e successivamente "sdoppiato" per intraprendere due percorsi differenti:

1. Uno stream stereoscopico a colori viene utilizzato come image texture di due piani, sfasati tra di loro sull'asse orizzontale, della stessa distanza che intercorre tra i due sensori della stereo camera.
   Lo stream proveniente dalla camera di sinistra è proiettato sul piano di sinistra mentre lo stream di destra sul piano di destra.
2. Un flusso video in scala di grigi (a singolo canale) viene invece utilizzato per il tracciamento dei markers ARUCO.
   In funzione della posizione relativa di questi markers con il sistema di visione, viene calcola la posa, ovvero le trasformazioni spaziali (roto-traslazioni) che verranno applicate in tempo reale ai gameObjects contenenti le strutture anatomiche del paziente.

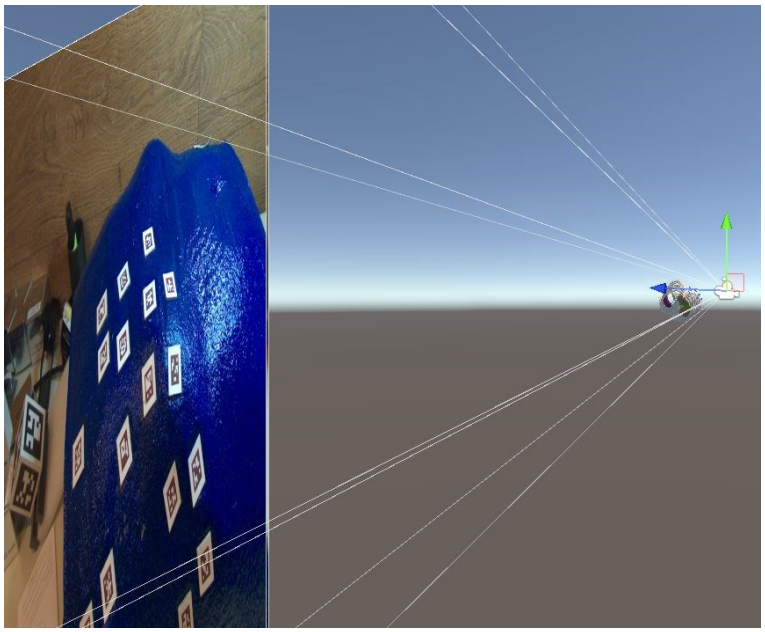

Figura 12: come appare la scena all'interno di Unity

I markers sul torso del paziente possono essere disposti pressoché randomicamente; infatti, i dati sulla loro posizione ed orientamento vengono acquisiti mediante una fase di scansione chiamata co-registrazione.

Osservando la scena in modalità viewport, si può notare come le camere di Unity e i piani immagine siano fermi rispetto al sistema di riferimento globale, mentre gli unici elementi

che mutano sono la posizione delle strutture anatomiche e la texture dei due piani immagine, la quale si aggiorna coerentemente con lo stream video.

Questa implementazione permette, tramite le due camere di Unity, di visualizzare le strutture anatomiche del paziente come perfettamente solidali con il suo busto.

2.3 LA QUALITÀ GRAFICA DI ENDOSIGHT

In questa sezione analizzeremo la pipeline grafica della sezione di realtà aumentata di Endosight, evidenziandone le criticità e proponendo delle soluzioni, le quali saranno approfondite all'interno della sezione Materiali e Metodi.

Le principali criticità riguardano il senso di profondità che viene restituito dalle immagini di realtà aumentata: come vedremo nel dettaglio nei prossimi paragrafi, gli shader privi di dettagli, la mancata implementazione di un comportamento fisicamente accurato dei materiali (ombre e riflessioni), nonché le occlusioni degli oggetti sintetici, ai danni di elementi reali, i quali si dovrebbero trovare più vicini alla camera, concorrono ad un impoverimento della potenziale percezione di profondità.

Con lo scopo di migliorare la comprensione delle criticità e le motivazioni che hanno spinto verso l'adozione delle soluzioni illustrate all'interno di Materiali e Metodi, è stato inserito nel documento il successivo paragrafo, che riguarda un breve approfondimento teorico sul funzionamento cognitivo che concorre alla generazione del senso di profondità visiva nell'essere umano.

2.3.1 Percezione di profondità: accenni teorici

La percezione della profondità è un elemento chiave della nostra visione tridimensionale del mondo, permettendo al cervello di determinare le distanze e le posizioni relative degli oggetti nello spazio. Questa percezione è guidata da una serie di fattori visivi che possono essere classificati in tre categorie: indizi monoculari, indizi binoculari o stereopsici, e indizi temporali.

1. **Indizi monoculari**: Gli indizi monoculari sono quegli indizi sulla profondità e sulla distanza che possono essere rilevati con un solo occhio. Alcuni degli indizi monoculari più importanti sono:

   1.1 Prospettiva lineare: Si riferisce al fenomeno per cui linee parallele sembrano convergere a un punto di fuga con l'aumentare della distanza. Questo può essere osservato, ad esempio, guardando lungo una strada dritta, dove le linee del marciapiede sembrano convergere in lontananza.

   1.2 Gradiente di texture: Quando osserviamo una superficie tessellata o una superficie con una texture ripetitiva, noteremo che la dimensione delle tessere o la densità della texture sembra aumentare con la distanza. Questo fornisce un importante indizio sulla profondità e sulla distanza degli oggetti.

   1.3 Sfumato atmosferico o perspettiva aerea: In scene molto ampie, gli oggetti lontani appaiono più sfocati e meno saturi a causa dell'interferenza dell'atmosfera. Questo effetto è particolarmente visibile in giornate con foschia o nebbiose, ma può essere percepito anche in condizioni di visibilità ottimale.

   1.4 Ombreggiatura e illuminazione: Le ombre proiettate dagli oggetti possono fornire indizi sulla loro forma tridimensionale e sulla loro posizione relativa. Inoltre, le variazioni di illuminazione su una superficie possono suggerire la sua forma tridimensionale.

   1.5 Interposizione o occlusione: Quando un oggetto blocca parzialmente la vista di un altro oggetto, possiamo dedurre che l'oggetto che blocca la vista sia più vicino a noi. Questo indizio è noto come interposizione o sovrapposizione.

2. **Indizi binoculari o stereopsici**: Questi indizi sfruttano la distanza tra i nostri due occhi per percepire la profondità. Quando osserviamo un oggetto, l'immagine che cade sulla retina di ciascuno dei nostri occhi è leggermente diversa a causa del diverso angolo di visione. Il cervello rielabora queste due immagini per

ottenere una percezione di profondità. Questo fenomeno è noto come disparità binoculare o stereopsi.

3. **Indizi temporali**: Questi si riferiscono alla percezione della profondità derivante dal movimento. Ad esempio, la parallasse del movimento si verifica quando osserviamo un ambiente in movimento: gli oggetti vicini sembrano muoversi più velocemente di quelli distanti, fornendo un indizio sulla loro posizione relativa. Un altro esempio è la percezione del movimento in profondità attraverso l'accomodazione e la convergenza degli occhi, meccanismi che il sistema visivo utilizza per mettere a fuoco gli oggetti a diverse distanze.

In conclusione, la percezione della profondità è un processo complesso che coinvolge una combinazione di diversi indizi visivi. Questa percezione consente al cervello di costruire un'immagine tridimensionale del mondo a partire da informazioni bidimensionali, consentendoci di navigare e interagire efficacemente con il nostro ambiente.

Risulta concluso questo l'approfondimento teorico; i prossimi paragrafi riguarderanno l'analisi delle caratteristiche del sistema Endosight.

2.3.2 Qualità della geometria

Come precedentemente evidenziato, la topologia delle mesh generate tramite triangolazione della nuvola di punti , risulta essere caotica, poco pulita e caratterizzata da un numero eccessivo di facce.

Inoltre, la densità dei triangoli non è omogenea su tutta la mesh.

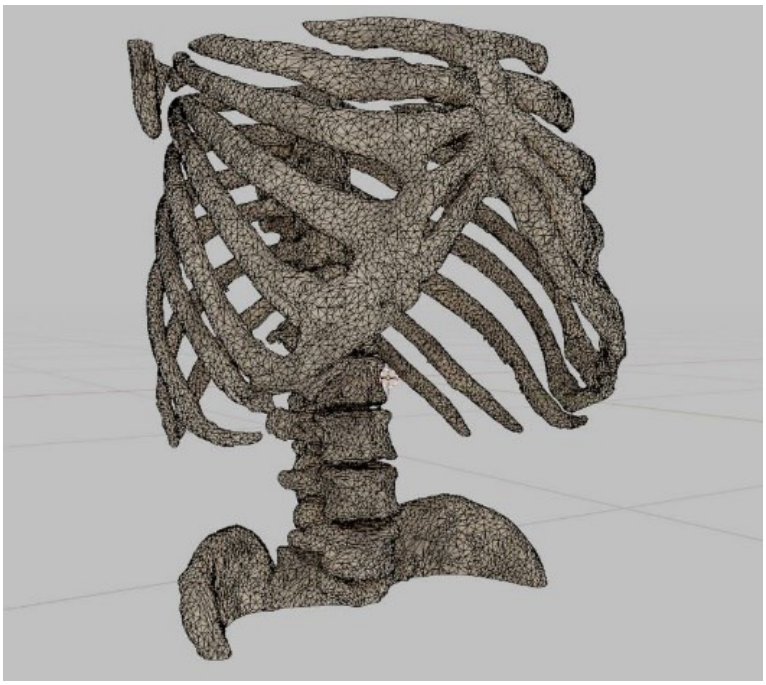

Figura 13: topologia della mesh "Bones". Si contano 59.185 vertici e 119.741 triangoli

Essendo la geometria generata automaticamente dall'algoritmo di Endosight, risulta molto complesso ottenere una topologia pulita, coerente ed omogenea, come la si potrebbe ottenere tramite modellazione manuale.

Portando la geometria sul software di modellazione 3D Blender, è possibile effettuare il merging dei vertici, in funzione della distanza relativa tra gli stessi.

Impostando una distanza di merging di 0.005m, equivalenti allo 0.25% della dimensione maggiore dell'oggetto, notiamo una diminuzione di 1127 vertici (equivalenti al 1.9% del totale), senza un apprezzabile mutamento della forma della mesh.

Questo indica che molti vertici risultano essere molto vicini, se non addirittura sovrapposti.

Questa caratteristica intrinseca degli elementi tridimensionali, unitamente all'inesistenza all'intero della pipeline di Endosight, di un algoritmo che generi proceduralmente delle mappe UV, ha comportato, durante il progetto, che i materiali venissero gestiti tramite delle tecniche di shading UV-independent.

Approfondiremo questo argomento all'interno della sezione Materiali e Metodi.

2.3.3 Qualità dei materiali e Unity rendering pipeline

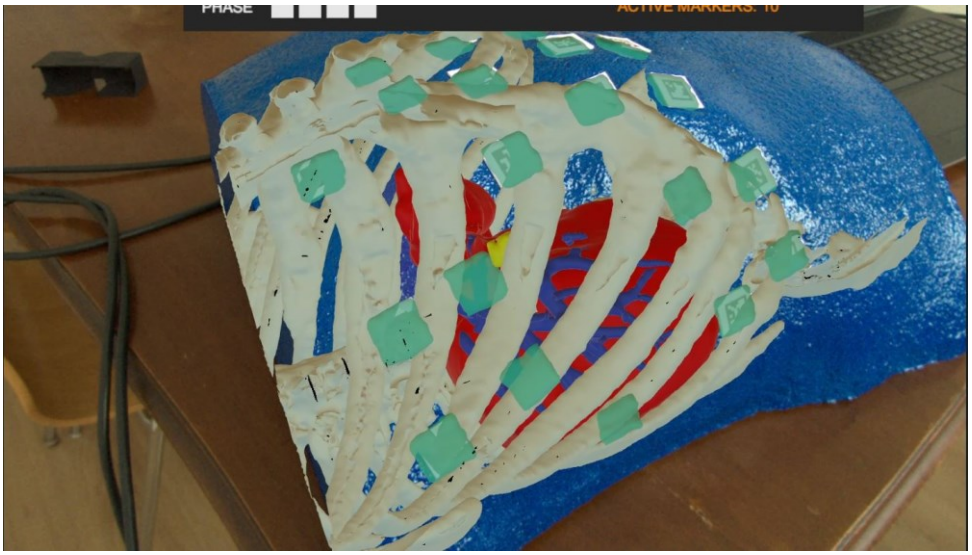

Figura 14: Qualità dei materiali applicati alle mesh

La pipeline di rendering che è stata implementata in Endosight è la “built-in rendering pipeline”; affronteremo nel dettaglio questo aspetto nel capitolo Materiali e Metodi, all’interno della sezione dedicata alle soluzioni applicate per ottenere una resa più realistica dei materiali.

Lo shader dei materiali invece risulta veramente basilare: si tratta di un semplice colore RGB applicato ad un Unlit shader.

Un "Unlit Shader" in Unity è un tipo di shader che non tiene conto dell'illuminazione o dell'ombreggiatura della scena. Questo significa che non calcola effetti di luce o ombra, e mostra i colori e le texture esattamente come sono, indipendentemente dalla direzione della luce o dalla posizione dell'oggetto.

I materiali non sfruttano quindi le potenzialità del PBR (Physically Based Rendering) per ottenere una restituzione realistica del comportamento fisico della luce.

2.3.4 Gestione delle trasparenze

Per le finalità applicative che abbiamo descritto nel capitolo precedente, è importante che gli oggetti sintetici, contenuti all'interno di altri oggetti sintetici (come, ad esempio, i vasi sanguigni del fegato) possano essere visti dall'esterno.

A tal fine è stato sfruttato uno dei sistemi di ottimizzazione per il rendering real time di Unity: le facce aventi normale opposta alla direzione di vista della camera, in Unity non vengono renderizzate.

L'idea inizialmente implementata è stata quella di flippare tutte e normali delle facce delle mesh, affinché si potesse vedere all'interno di queste ultime.

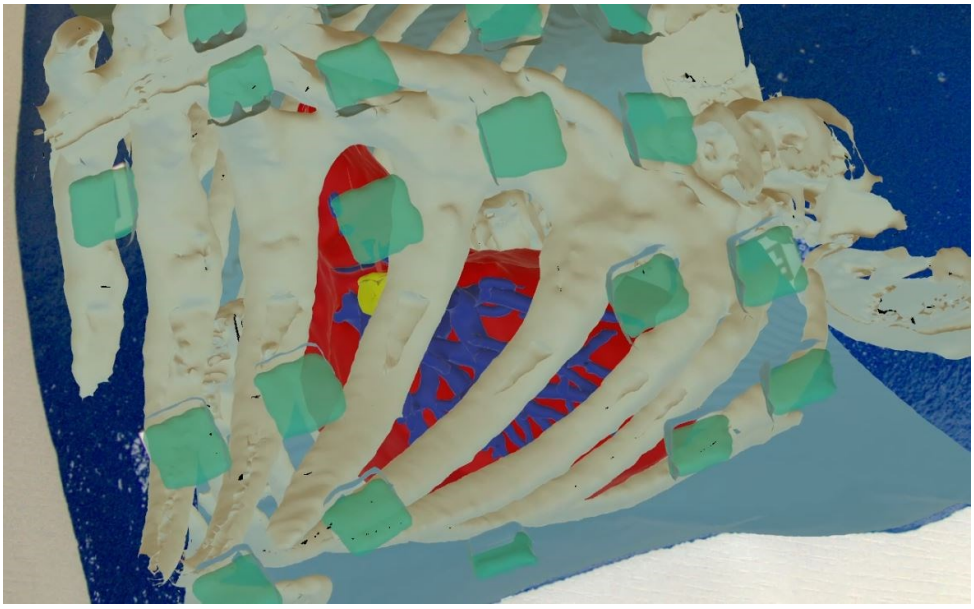

Figura 15: visualizzazione AR in Endosight. Le normali invertite permettono di visualizzare l'interno degli oggetti renderizzati

Questa soluzione genera però due problematiche principali:

1. Viene persa l'informazione riguardante le facce rivolte verso la camera, il che rende complesso capire la reale forma degli organi interni.
2. Non viene propriamente gestita la trasparenza, quindi gli oggetti più vicini alla camera continuano ad occludere totalmente quelli più lontani.

2.3.5 Occlusioni non corrette

Utilizzando Endosight per visualizzare le strutture anatomiche del paziente in realtà aumentata, risulta evidente quanto la visualizzazione dei modelli 3D, che sono renderizzati al di sopra della mano di chi sta operando, affligga non solo il senso di realismo e la corretta sensazione di profondità della scena, ma renda complesso compiere azioni con la propria mano, in quanto essa risulta completamente occlusa alla vista.

L'effetto diventa molto più invasivo se viene renderizzata la mesh della pelle del paziente.

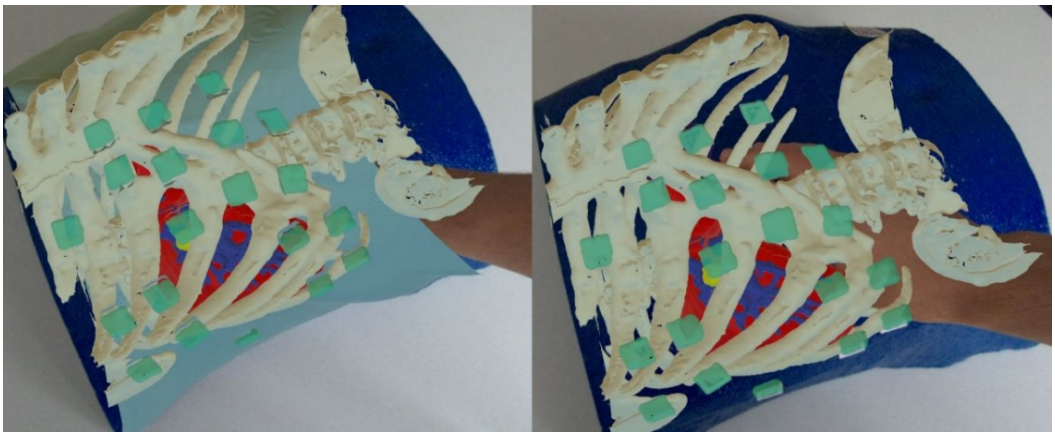

Figura 16: visualizzazione monoscopica in AR delle strutture anatomiche; con e senza pelle.

La scomparsa della mano viene leggermente mitigata quando si adopera il tool di ablazione (Figura 17). I suoi movimenti vengono infatti tracciati grazie a cinque markers ARUCO per la posa di un cubetto in computer grafica che simuli la posizione del tool (Figura 17).

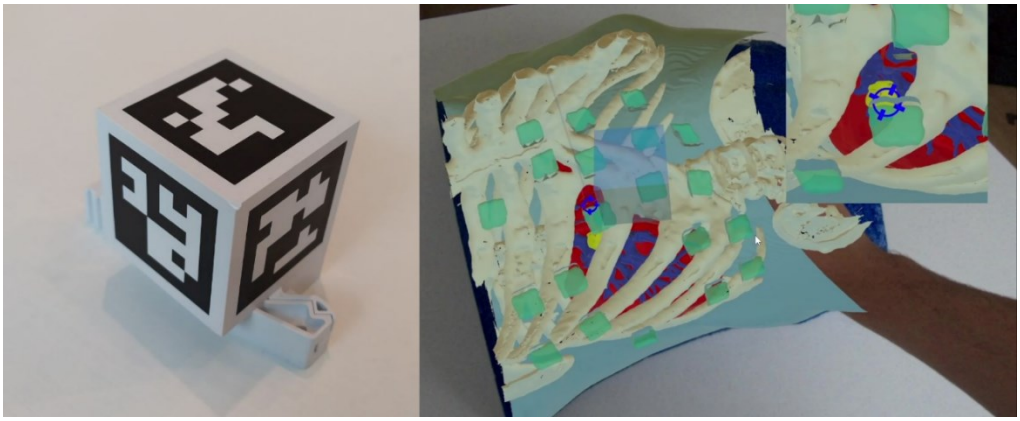

Figura 17: tool per il tracciamento dell'ago di ablazione e renderizzazione del cubetto in CG (in blu).

Al fine di migliorare l'esperienza di utilizzo e conseguentemente la bontà dei risultati ottenuti in fase operatoria, è risultato chiaro come questa problematica fosse l'aspetto su cui riversare maggiore attenzione durante il progetto di tesi.

La soluzione a questo aspetto è ampliamente dettagliata all'interno del capitolo Materiali e Metodi.

3 MATERIALI E METODI

Di seguito analizzeremo nel dettaglio tutte le modifiche apportate al sistema Endosight al fine di migliorare la resa visiva e il senso di profondità percepito durante il suo utilizzo.

3.1 CREAZIONE DI SHADERS REALISTICI

In questa sezione analizziamo le soluzioni adottate per l'ottenimento di una resa visiva maggiormente realistica dei materiali applicati agli oggetti generati in computer grafica.

Ricordiamo che texture povere e prive di dettaglio non affliggono soltanto la qualità grafica, ma rendono complesso per il sistema visivo umano riconoscere le features necessarie ad una corretta stereopsi.

Al fine di migliorare la comprensione dei concetti, verranno eseguiti degli approfondimenti teorici sulle nozioni di Render Pipeline, Shader Graph e Tri-planar projection.

3.1.1 Universal Render Pipeline (URP)

La pipeline di rendering è un concetto fondamentale nell'ambito della grafica computazionale; esso descrive il processo sequenziale attraverso il quale una scena tridimensionale virtuale viene convertita in un'immagine bidimensionale.

Questo processo, svolto oggigiorno principalmente dalla GPU (Graphics Processing Unit), coinvolge una serie di passaggi che vanno dalla definizione dei dati geometrici iniziali fino alla produzione dell'immagine finale visualizzata sullo schermo.

All'interno del contesto del motore grafico di Unity, possiamo evidenziare la possibilità di adozione di *tre differenti pipeline di rendering* [7]:

[7] Da documentazione ufficiale Unity presente al seguente link: https://docs.unity3d.com/Manual/render-pipelines.html

1. Built-in Rendering Pipeline.
2. Universal Rendering Pipeline.
3. High-Definition Rendering Pipeline.

Per le finalità di questo documento di tesi, affrontiamo le principali differenze tra la pipeline (1) e la pipeline (2);

La Universal Render Pipeline (URP), precedentemente conosciuta come Light Weight Render Pipeline (LWRP), è una soluzione di rendering avanzata e altamente scalabile offerta da Unity.

È stata progettata per supportare un'ampia gamma di piattaforme hardware, dai dispositivi mobili alle console di gioco e ai PC di fascia alta, e offre un equilibrio tra prestazioni e qualità dell'immagine.

Uno degli aspetti distintivi della URP è la sua struttura semplificata e ottimizzata che possibilmente riduce *l'overhead del driver*[8] e offre una pipeline di rendering più snella rispetto alla Built-in Render Pipeline.

Inoltre, grazie all'uso di Shader Graph e agli strumenti di post-elaborazione integrati, URP consente agli sviluppatori di creare shader ed effetti visivi personalizzati senza dover scrivere codice shader complesso.

Un altro vantaggio significativo della URP è la sua capacità di sfruttare gli eventi di rendering, come beginCameraRendering(), beginFrameRendering(), endCameraRendering() e endFrameRendering(). Questi eventi offrono agli sviluppatori un controllo molto preciso sul flusso di rendering, consentendo di eseguire del codice in specifici momenti del processo di rendering.

D'altra parte, la Built-in Render Pipeline di Unity, che è stata la soluzione di rendering predefinita di Unity per molti anni, non offre gli stessi eventi di callback per il rendering come quelli presenti nella URP.

Un altro aspetto da considerare è che la Built-in Render Pipeline non è più attivamente sviluppata da Unity, con l'eccezione di correzioni di bug critici e problemi di sicurezza.

[8] Si riferisce all'insieme di operazioni di gestione, manutenzione e altre attività amministrative che il software del driver deve svolgere per funzionare correttamente.

Ciò significa che non riceverà nuove funzionalità o miglioramenti, il che potrebbe limitare le sue capacità rispetto alla URP o alla High Definition Render Pipeline (HDRP) nel lungo termine; ciò non è compatibile con le esigenze di un prodotto commercializzato come Endosight.

In termini di prestazioni, la URP è generalmente più veloce della Built-in Render Pipeline su hardware più recente, grazie alla sua architettura ottimizzata. Tuttavia, le prestazioni specifiche possono variare a seconda delle specifiche del progetto e dell'hardware utilizzato.

Alla luce di queste differenze e volendo creare degli shader fotorealistici, ma facilmente editabili e personalizzabili, è stata abbandonata la Built-in Pipeline a favore della Universal Render Pipeline. Come vedremo successivamente, sono state ampiamente sfruttate le potenzialità degli shader graphs messi a disposizione dalla URP per creare dei materiali procedurali indipendenti dalle coordinate UV dei modelli 3D.

3.1.2 URP Graph Editor

Nell'ambito del progetto Endosight, l'implementazione di materiali procedurali è stata una scelta chiave, motivata dalla natura unica e specifica delle mesh anatomiche generate. Infatti, queste mesh sono create attraverso un processo di triangolazione di una nuvola di punti, derivata da immagini TAC del paziente.

Queste immagini sono poi segmentate per isolare le diverse sezioni anatomiche.

Dato che il passaggio da immagini DICOM a modelli 3D è automatizzato da Endosight, non è prevista una fase di UV Mapping.

A fronte di queste peculiarità, l'adozione di texture tradizionali, che richiedono una mappatura UV, non risulta un'opzione praticabile: la complessità delle mesh anatomiche, la topologia poco pulita unita all'impossibilità di creare una texture a priori, senza conoscere il modello e la sua potenziale UV Map, hanno guidato la scelta verso l'utilizzo di materiali "UV-independent", ovvero indipendenti dalla mappatura UV (tema approfondito nel paragrafo dedicato alla tri-planar projection).

Per realizzare questi materiali procedurali, è stato fondamentale sfruttare le potenzialità offerte dagli shader graphs di Unity. Gli shader graphs rappresentano uno strumento flessibile che consente di creare shader personalizzati attraverso un'interfaccia visuale, senza la necessità di scrivere codice shader. Attraverso questo sistema, è possibile

costruire un grafico che calcola l'aspetto del materiale in base a parametri come la posizione, l'orientamento e altre proprietà dei vertici o dei pixel.

Questo strumento è basato sull'idea di costruire shader attraverso un grafico nodale, ovvero una struttura di dati che consiste in nodi (o vertici) e connessioni (o bordi).

Ogni nodo dello Shader Graph rappresenta una specifica operazione o funzione che si può eseguire nel contesto dello shader, come l'addizione di due valori, il calcolo del dot product tra due vettori, o la generazione di un colore a partire da una texture. Questi nodi possono avere uno o più ingressi, chiamati socket, che rappresentano i valori di input per l'operazione, e una o più uscite, che rappresentano i risultati dell'operazione.

La creazione di uno shader attraverso lo Shader Graph avviene collegando i nodi tra di loro; il flusso di dati all'interno del grafico procede seguendo i collegamenti tra i nodi, venendo rimaneggiato fino a giungere al nodo di output. Questo modello di programmazione visuale offre una grande flessibilità, in quanto consente di esplorare facilmente diverse combinazioni di operazioni e di vedere immediatamente l'effetto che queste hanno sul materiale o sull'effetto visivo che si sta creando.

Uno dei punti di forza dello Shader Graph è la sua capacità di generare dinamicamente lo shader a partire dal grafico nodale. Questo significa che, ogni volta che si modifica il grafico, lo shader viene rigenerato per rispecchiare le modifiche apportate. Questo processo di generazione dello shader è completamente automatico e nascosto all'utente, il quale può concentrarsi sulla creazione del grafico senza preoccuparsi dei dettagli della scrittura dello shader.

Tuttavia, è importante notare che, a causa di questo processo automatico di generazione dello shader, il codice dello shader generato non è direttamente accessibile o modificabile dall'utente. Questo limita alcune opzioni avanzate di ottimizzazione o personalizzazione dello shader ed ha costituito una problematica per il settaggio dello stencil buffer nelle fasi finali del progetto.

3.1.3 Proiezione Tri-planare

Come evidenziato precedentemente, la scelta di una mappatura delle textures UV-independent si è rivelata necessaria per l'ottenimento di una resa realistica dei materiali.

È stata adottata quindi una proiezione tri-planare per la creazione degli shaders di tutte le mesh rappresentative delle strutture anatomiche prese in analisi da Endosight:

- Ossa della cassa toracica
- Pelle
- Fegato
- Milza
- Principali vasi sanguigni del fegato
- Massa tumorale (lesione)

Vediamo nel dettaglio di cosa si tratta:

La proiezione tri-planare è una tecnica di mappatura delle texture utilizzata per applicare una texture a una mesh 3D in modo tale da minimizzare la distorsione e l'allungamento che possono verificarsi con altre tecniche di mappatura, come la mappatura UV.

Questa tecnica risulta particolarmente efficace quando si lavora con mesh complesse o irregolari per le quali una mappatura UV potrebbe essere difficile o impossibile da realizzare in modo efficace.

Il concetto fondamentale della proiezione tri-planare è che la texture viene proiettata sulla mesh da tre direzioni ortogonali (solitamente lungo gli assi X, Y e Z del sistema di riferimento locale dell'oggetto). Ogni proiezione crea una versione "piatta" della texture sulla superficie della mesh, indipendentemente dalla sua forma o topologia.

Dopo aver effettuato le tre proiezioni, ogni pixel della mesh contiene tre valori di colore, ognuno derivante da ciascuna proiezione. Questi colori vengono poi combinati in un unico colore finale in base alla normale della superficie in quel punto. In pratica, ciò significa che la direzione in cui la superficie "guarda" determina quale delle tre proiezioni avrà più influenza sul colore finale.

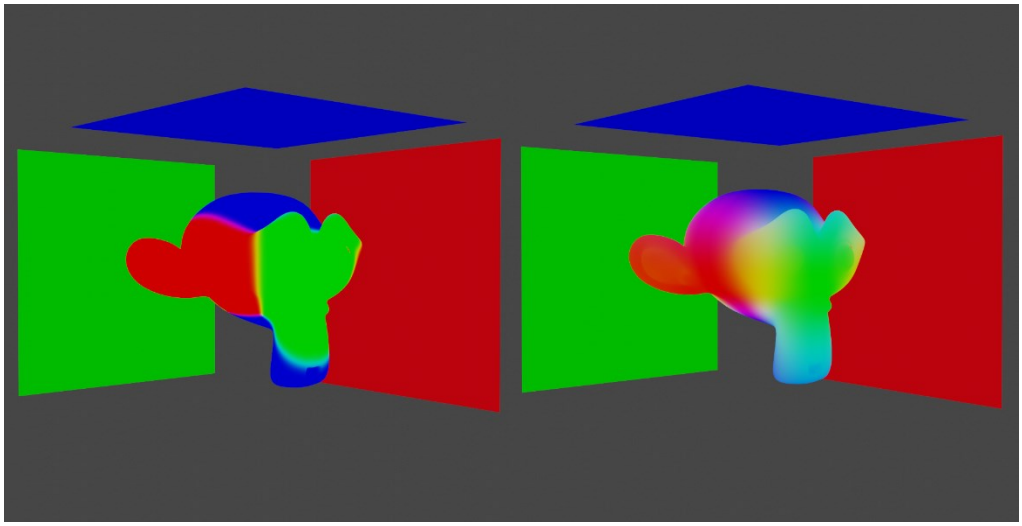

Figura 18: Rappresentazione grafica della proiezione tri-planare. Il gradiente tra le mappe proiettate è un parametro che può essere modificato e può essere reso più netto o più graduale.

Ad esempio, se la superficie sta "guardando" verso l'alto lungo l'asse Y, la proiezione Y avrà il maggior peso nel colore finale. Se la superficie sta "guardando" in una direzione che è una combinazione di Y e Z, i colori provenienti da queste due proiezioni saranno combinati in proporzione.

È importante notare che la proiezione tri-planare richiede più risorse di calcolo rispetto ad altre tecniche di mappatura delle texture, a causa del fatto che deve calcolare e combinare tre proiezioni separate per ogni pixel. Inoltre, poiché la stessa texture viene proiettata da tre direzioni diverse, possono verificarsi ripetizioni visibili o discontinuità nelle texture a meno che non vengano prese precauzioni per mitigarle.

3.1.4 Bones Shader Graph

Di seguito analizziamo lo shader graph del materiale “Bones”, applicato dinamicamente in Endosight al MeshRenderer del gameObject relativo alle ossa:

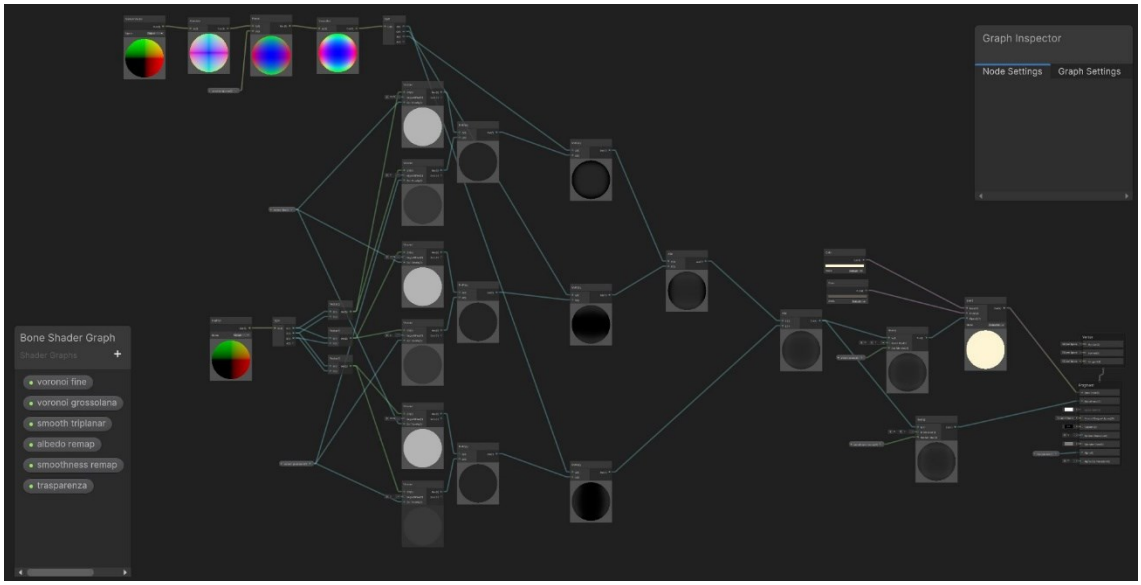

Figura 19: Bones Shader Graph nel suo insieme.

Come prima cosa è stato necessario generare i piani di proiezione delle texture planari tramite scomposizione in vector2 delle coordinate locali dell'oggetto (Figura 20).

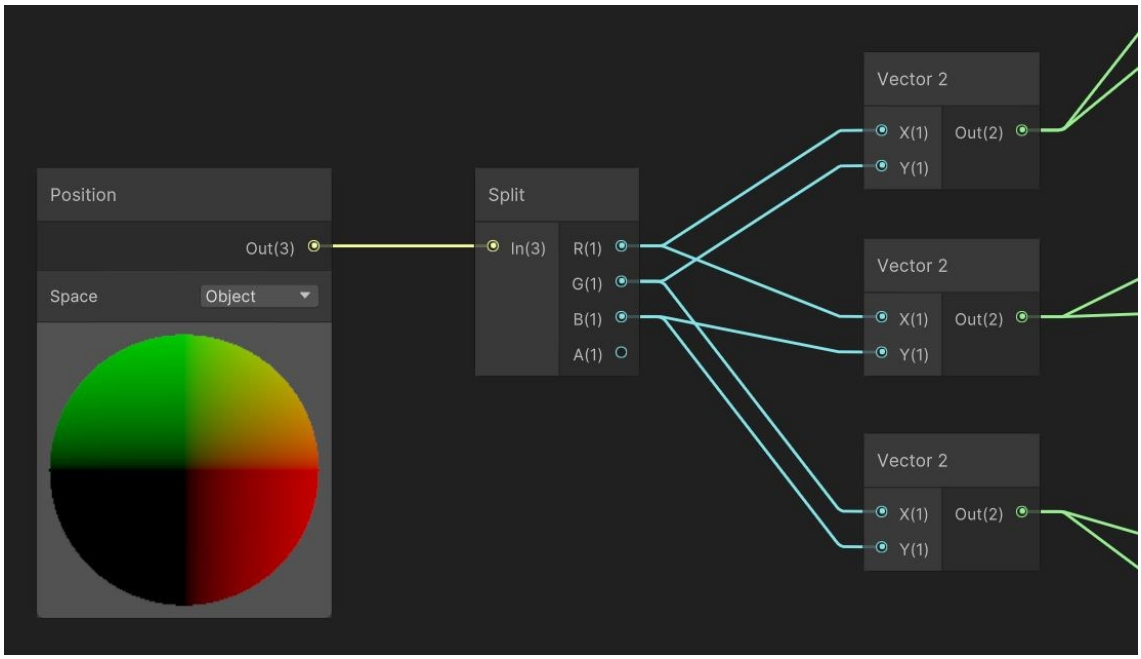

Figura 20: Bones Shader Graph, modellazione dei piani di proiezione

I vector2 così generati, sono serviti come coordinate di texture di due mappe di rumore di tipo “voronoi” moltiplicate tra loro (Figura 21).

Le mappe di rumore sono tendenzialmente dei valori rappresentabili con un singolo canale, ad esempio in scala di grigi da 0 a 255 in rappresentazione ad 8bit. Risulta quindi facile sfruttare una combinazione di mappe, a differenti granulosità, per ottenere una realistica variabilità della texture; in questo caso ne sono state utilizzate due.

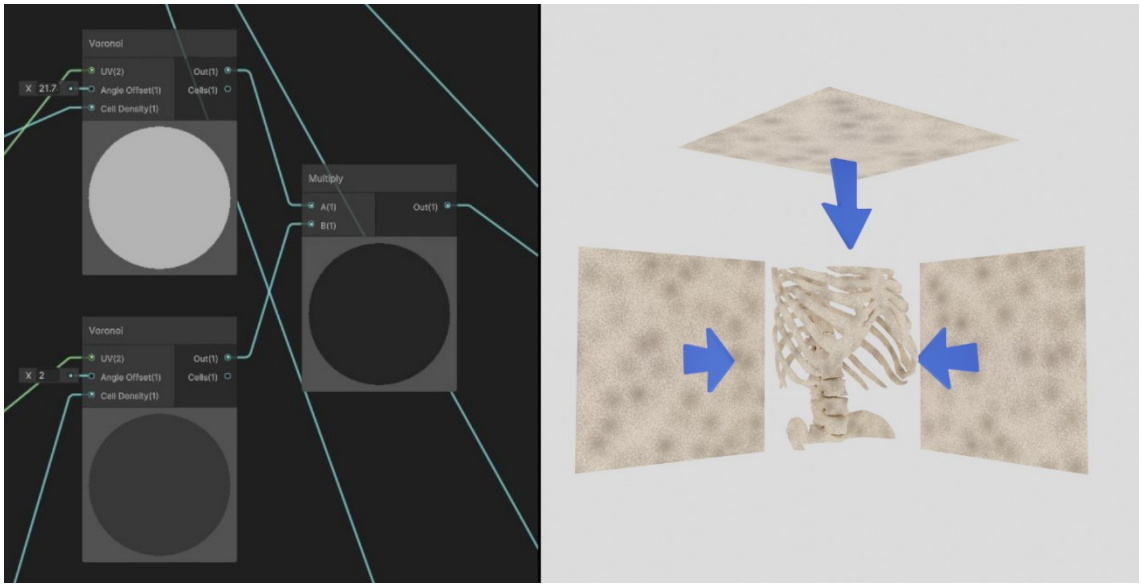

Figura 21: unione delle due mappe di rumore tramite operazione di moltiplicazione (sulla sinistra). Rappresentazione della proiezione tri-planare applicata a Bones Shader Graph (sulla destra).

Come spiegato prima, ogni punto della mesh, a questo punto del grafo, presenta tre colori differenti, derivanti dalla proiezione dei tre piani di texture; risulta quindi necessario inserire un metodo di scelta intelligente, per definire quale dei tre colori assegnare al pixel relativo allo specifico punto della mesh.

A tal fine è stato modellato un criterio decisionale basato sulla direzione delle normali della mesh. In Figura 22 si può notare l’applicazione della funzione “absolute”, che, similmente ad una funzione modulo, ribalta sul semiasse positivo le componenti negative delle normali dell’oggetto. In seguito, le tre componenti scalari del vettore che rappresenta le normali vengono elevate a potenza; questo passaggio permette di gestire il “fade” del criterio di selezione nelle zone di interfaccia tra le proiezioni (come mostrato

in Figura 22). Questo parametro è modificabile real time da inspector tramite la variabile pubblica “smooth triplanar”.

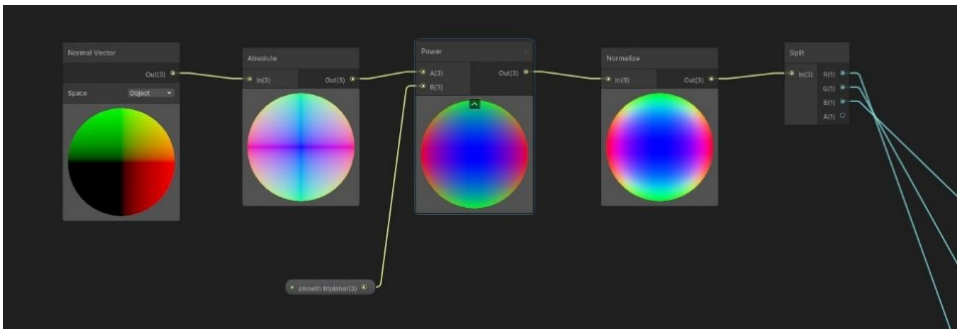

Figura 22: modellazione del criterio di scelta della proiezione, all’interno del Bones Shader Graph

Il criterio di scelta della proiezione interviene all’interno della logica dello shader graph grazie ad un nodo multiply, che moltiplica ogni piano di proiezione per la componente della normale ortogonale al piano stesso (Figura 23).

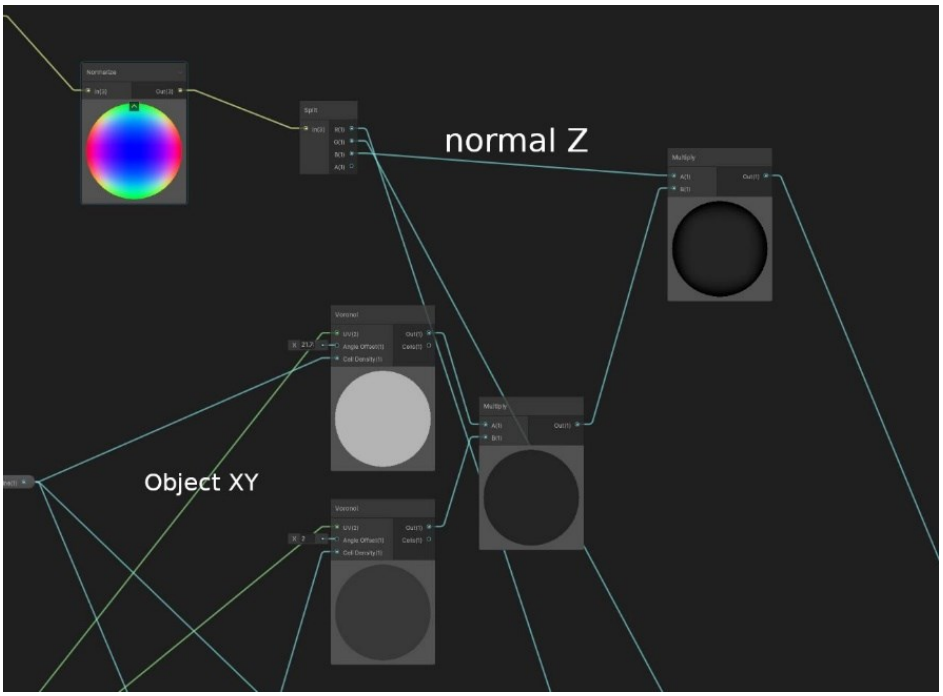

Figura 23: applicazione del criterio di scelta all’interno del Bones Shader Graph.

Successivamente i contributi delle tre proiezioni sono stati sommati tramite due nodi "add" consecutivi.

La texture ottenuta è stata poi data in pasto un nodo "remap", che mappa i valori in ingresso su un range differente, permettendo così di modificare il contrasto globale dell'immagine monocromatica (come evidenziato nelle Figure 24 e 25).

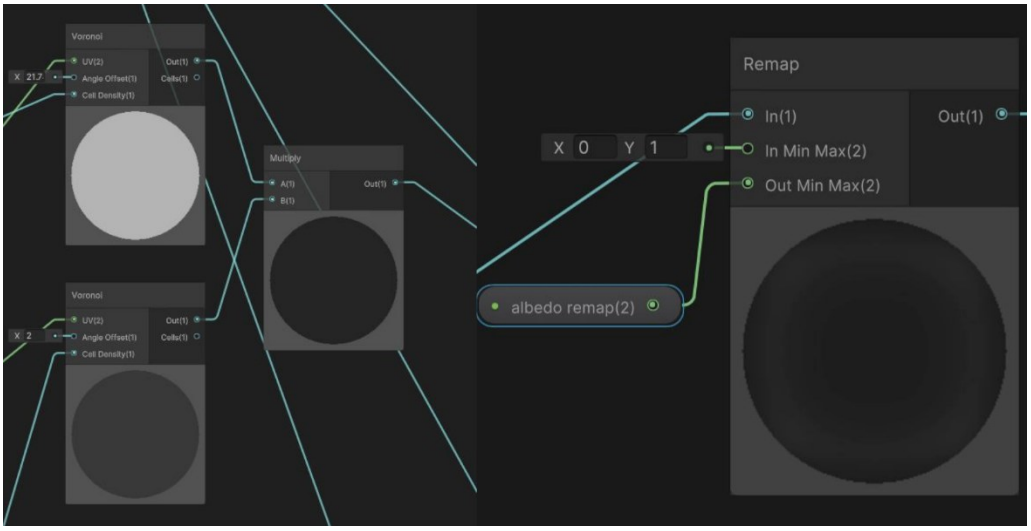

Figura 24: somma dei contributi lungo gli assi e rimappatura dei valori in scala di grigio

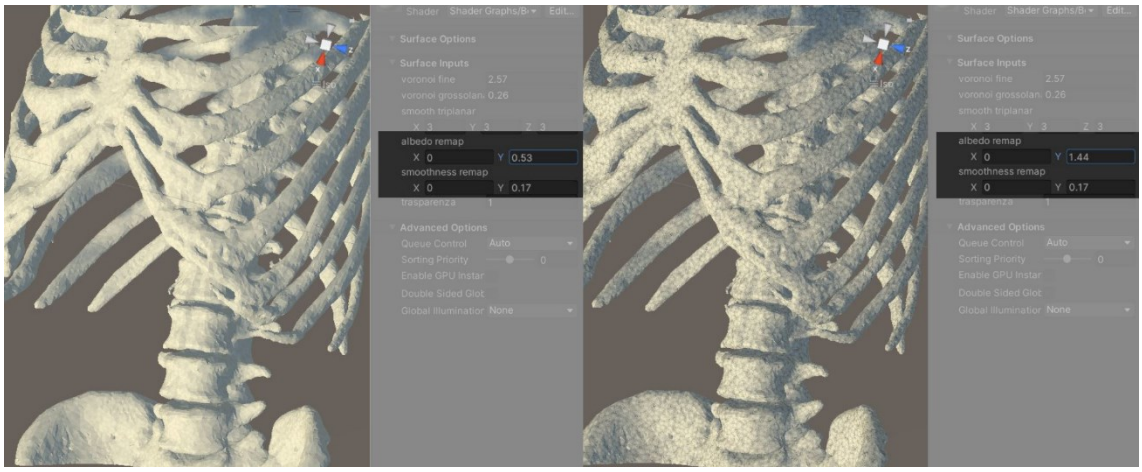

Figura 25: Rimappatura su differenti ranges della texture monocromatica.

Infine, la texture rimappata è stata utilizzata in maniera diretta per guidare il parametro smoothness del nodo fragment e un ulteriore nodo “RGBA Blend” per guidare il parametro “Base Color” (Figura 26).

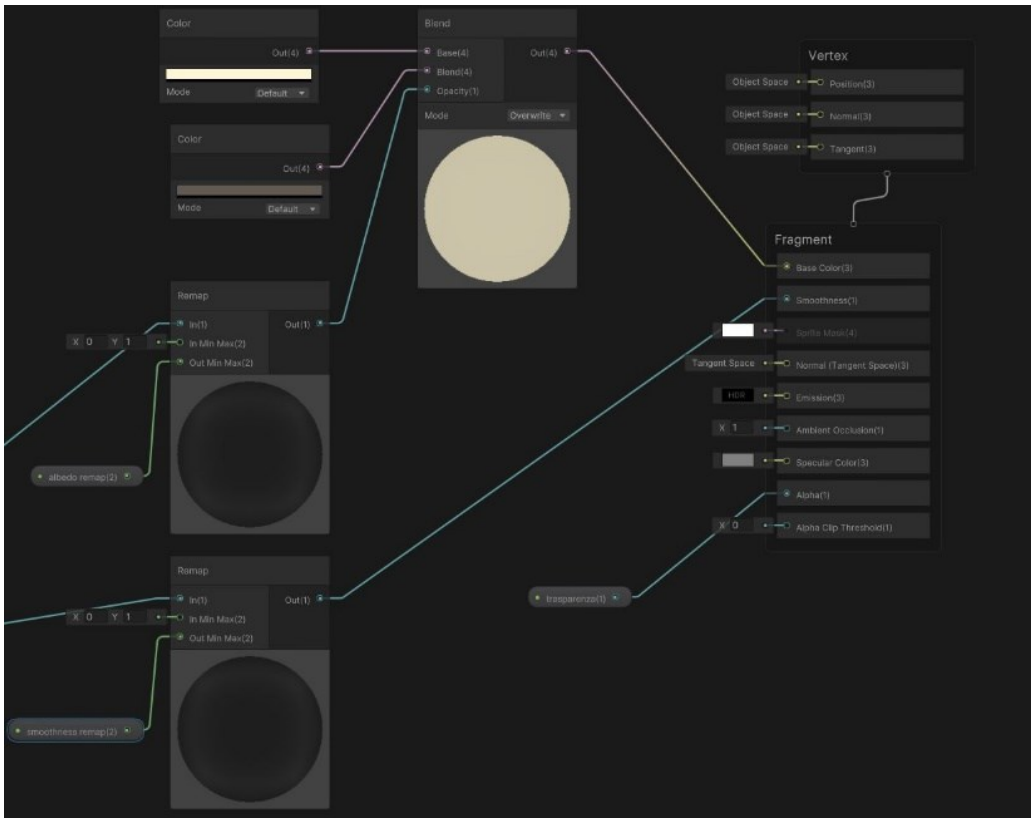

Figura 26: ingressi del nodo fragment, che gestisce il PBR material.

Gli shader dei materiali degli altri gameObject sono stati creati tramite shader graphs molto simili a questo preso in esempio.

3.2 VISUALIZZAZIONE DELLE MANI DELL'UTENTE SOPRA ALLE STRUTTURE ANATOMICHE

Come precedentemente evidenziato, la problematica principale, che si riscontra durante l'utilizzo di Endosight in realtà aumentata, risulta essere l'impossibilità di visualizzare le mani dell'utente al di sotto dei modelli 3D.

3.2.1 Soluzioni ipotizzate e successivamente scartate

Sono state ipotizzate e successivamente scartate due possibili soluzioni, prima di approdare all'opzione finale:

3.2.1.1 Soluzione Uno: Chroma Key

La prima soluzione immaginata è stata generare una maschera 2D tramite chroma key del video, che isolasse i pixel relativi alla mano del medico.

Tramite modifica della pipeline di render si sarebbero sovrascritti i pixel relativi alle mesh con i pixel del video sottostante; una modifica attuabile sarebbe stata intervenire sullo z buffer, assegnando una z pari a quella del layer video.

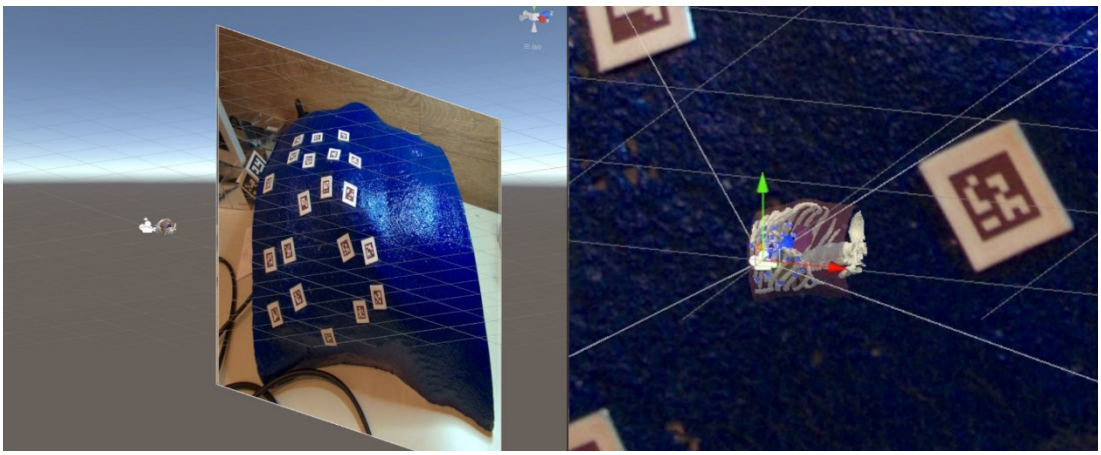

Figura 27: Scena di Unity: camera e piano immagine sono fissi all'interno dello spazio, mentre i gameObjects degli organi (posti tra la camera e il piano immagine) sono orientati in real time in funzione del calcolo della posa.

Questa soluzione è stata scartata in quanto poco attuabile nelle condizioni di lavoro in cui si trova ad operare Endosight:

1. Per poter mascherare le mani del medico si ha la necessità che tutto ciò che si trovi in asse con le mani sia di un colore molto differente da quello delle mani stesse. Questa variabile renderebbe il sistema Endosight meno "plug and play" e fortemente dipendente dalle condizioni ambientali in cui si ritrova ad operare.
2. Sarebbe necessario quantizzare il video utilizzando codec ad elevata profondità di bit (almeno 10bit), possibilmente senza sottocampionamento della crominanza (almeno 4:2:2) e ad elevata risoluzione. Questo comporterebbe uno stream di dati di notevole importanza.
3. Come evidenziato all'interno della sezione dedicata alle specifiche tecniche di Endosight, i sensori delle stereo camere sono molto densi e quindi presentano dei fotodiodi molto piccoli (2 μm x 2 μm). Questo comporta un *rapporto segnale rumore medio 42 dB* [9], non ottimale nell'ottica di campionamento di colore.

4. Un altro aspetto da tenere in conto sarebbe stato l'obbligo da parte della struttura sanitaria di munirsi di materiale chirurgico adatto al chroma key.

3.2.1.2 Soluzione Due: Hand Recognition

Avendo scartato la segmentazione della mano tramite chroma key, inizialmente, si è considerato l'impiego di algoritmi di riconoscimento della mano basati su machine learning per generare una maschera 2D della mano.

Questi algoritmi sfruttano frequentemente le reti neurali convoluzionali (CNNs) per identificare la posizione dei giunti delle mani all'interno di un'immagine, fornendo informazioni quali la posizione, l'orientamento e, in alcuni casi, la posizione delle singole dita.

[9] Specifiche tecniche consultabili al seguente link: https://hupuu.com/camera/1-1.8-sony-4k-cmos-imx334-en781-4k-8mp15fps-ex-sdi-1080p-hd-sdiahd-hd-sdi-analog-starlight?sku=CBB-OS2

Figura 28: esempi di hand recognition sviluppato tramite la libreria OpenCV di Python.

Tuttavia, la maggior parte degli algoritmi disponibili di hand detection sono implementati attraverso la libreria OpenCV per Python. Considerando che il sistema Endosight è stato realizzato in C++ e C#, l'integrazione di algoritmi basati su Python nel flusso di lavoro esistente, avrebbe rappresentato una sfida significativa.

Inoltre, pur essendo disponibili algoritmi basati su machine learning per il riconoscimento della mano, la loro applicazione nella creazione di una maschera completa della mano avrebbe presentato diverse difficoltà, tra cui la necessità di implementare tecniche di segmentazione semantica per estendere la maschera rilevata a tutta la mano e l'incremento della potenza computazionale richiesta per garantire il rendering real time (questi algoritmi sono molto pesanti in termini di complessità computazionale).

Inoltre, il contesto di utilizzo in Endosight è molto lontano dall'ideale: lo sfondo non è mai omogeneo e la posizione della mano del radiologo interventista nasconde spesso le dita durante l'intervento.

In considerazione di questi fattori, si è optato per un approccio alternativo. Questa scelta è stata motivata non solo dalle difficoltà tecniche sopra descritte, ma anche dall'opportunità didattica offerta dall'implementazione di un metodo basato sull'elaborazione delle immagini.

In particolare, si è scelto di sfruttare i dati di profondità forniti dalla camera stereoscopica per segmentare le mani. Questa soluzione ha presentato nuove sfide, che saranno discusse in dettaglio nelle sezioni seguenti.

3.2.2 Soluzione adottata: Depth Map

L'utilizzo di una depth map per estrarre informazioni sulla profondità della scena inquadrata è stato un elemento chiave nell'implementazione del sistema di realtà aumentata di Endosight.

Le mappe di profondità forniscono un modo efficace per rappresentare la distanza di ogni punto in una scena rispetto alla telecamera, consentendo la creazione di immagini tridimensionali da una serie di immagini bidimensionali.

Esistono diversi metodi per generare una depth map a partire da una singola immagine. Molti di questi approcci si basano su tecniche di apprendimento automatico e, in particolare, di apprendimento profondo.

Questi metodi, ad esempio, possono utilizzare reti neurali convoluzionali addestrate su un grande set di dati di immagini e le relative mappe di profondità, per imparare a prevedere la profondità a partire da una singola immagine.

Tuttavia, nonostante la sofisticatezza di questi metodi, essi presentano alcune sfide significative.

Prima di tutto, l'accuratezza di questi metodi dipende fortemente dalla qualità e dalla varietà del set di dati su cui la rete neurale è stata addestrata. Inoltre, queste tecniche tendono a essere computazionalmente intensive, richiedendo hardware potente e tendenzialmente sono poco sfruttabili per applicazioni real time.
Infine, i modelli basati su apprendimento profondo possono scontrarsi con scene che presentano pattern o strutture che non erano presenti nel set di dati di addestramento.

In contrasto con queste tecniche, l'approccio adottato in Endosight sfrutta la presenza di una stereo camera. Le stereo camere catturano due immagini leggermente sfasate della stessa scena, in modo simile a come vedono gli occhi umani. Attraverso algoritmi di stereo matching, si può calcolare la profondità della scena comparando le due immagini. Questo metodo offre diversi vantaggi: non richiede un ampio set di dati per l'addestramento, non è intensivo in termini computazionali come i metodi basati su apprendimento profondo e può gestire una varietà più ampia di scene senza incorrere in

problemi dovuti a pattern o strutture, che non sono stati presi in considerazione durante l'addestramento.

In ultima istanza, l'hardware di cui si compone il sistema Endosight presentava già un sistema di visione basato su due stereo camere, quindi la scelta è risultata praticamente immediata.

Una volta generata la depth map, sono stati eseguiti diversi passaggi di post-elaborazione per generare una mesh 3D che rappresentasse il contorno della mano. Questo processo ha incluso la triangolazione dei punti nella depth map e la pulizia della geometria generata per rimuovere eventuali artefatti indesiderati.

Una volta ottenuta la mesh, si sono presentate due possibili opzioni percorribili:

La prima era rendere visibile la mesh tramite uno shader, la mesh infatti, essendo descritta nel sistema di riferimento camera, si trova posizionata correttamente all'interno dello spazio 3D.

L'alternativa era modificare la pipeline di rendering attraverso alterazioni dello stencil buffer di Unity. In questo modo, i pixel relativi al piano su cui viene visualizzato il video proveniente dalla camera, sarebbero stati renderizzati sui pixel della mesh. Quest'ultima opzione ha offerto un maggiore controllo sul risultato finale e ha fornito un risultato visivamente migliore.

Nei paragrafi successivi analizzeremo più nel dettaglio tutti gli step che hanno portato alla creazione della mesh delle mani, partendo da un richiamo teorico riguardante la triangolazione.

3.2.3 Triangolazione: Fondamenti teorici

[Nota: Il seguente approfondimento teorico si basa sulle nozioni presenti all'interno del corso First Principle of Computer Vision, tenuto da Shree Nayar, T. C. Chang, docente di computer science presso la Columbia University e consultabile al seguente link: https://fpcv.cs.columbia.edu/; il quale rappresenta la fonte di testo e immagini presenti in questo sotto capitolo.]

La triangolazione è il principio fondamentale che sottende l'intero processo di generazione di una depth map da immagini stereo. Questa procedura può essere pensata come un parallelo del modo in cui gli esseri umani usano la visione binoculare per percepire la profondità. La triangolazione sfrutta le informazioni di due viste della stessa scena per inferire informazioni sulla profondità della scena stessa.

Immaginiamo di avere due fotocamere, una sinistra e una destra, che hanno catturato la stessa scena da due posizioni leggermente diverse.

Ogni fotocamera è calibrata e quindi conosciamo i suoi parametri interni (o_x *e* o_y).

La differenza posizionale tra le due telecamere è detta "base" e definisce un sistema di visione stereo semplice (Figura 29).

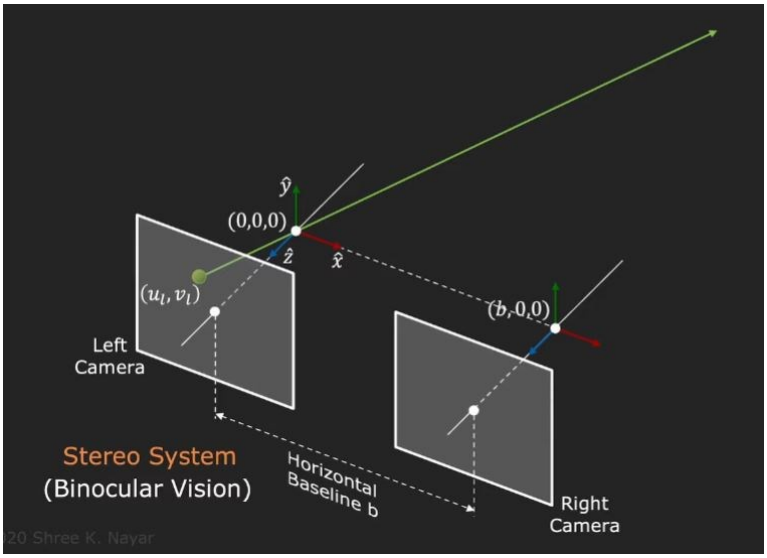

Figura 29: lo schema propone un sistema di visione stereo semplice.

Ora, consideriamo un pixel specifico nelle nostre immagini, denotato come (u_l, v_l) nel piano immagine di sinistra.

Non conosciamo le coordinate reali del punto, ma sappiamo che deve giacere su un raggio uscente dalla fotocamera di sinistra, definito dalle equazioni di proiezione prospettica definite di seguito:

$$u_l = f_x \frac{x}{z} + o_x$$

$$v_l = f_y \frac{y}{z} + o_y$$

$$v_r = f_y \frac{y}{z} + o_y$$

$$u_r = f_x \frac{x - b}{z} + o_x$$

Supponiamo ora di trovare il corrispondente di questo punto nell'immagine di destra, denotato come (u_r, v_r). Da qui possiamo emettere un altro raggio uscente dalla fotocamera destra.

Il punto di intersezione di questi due raggi è la posizione 3D del punto di scena che corrisponde ai nostri punti dell'immagine (Figura 30).

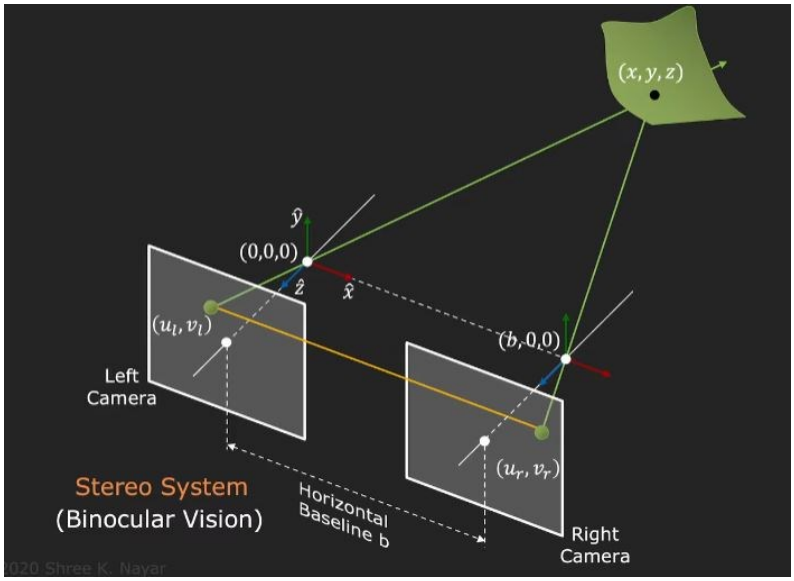

Figura 30: lo schema propone la risoluzione grafica del sistema di equazioni di proiezione prospettica

Per risolvere le coordinate 3D (x, y, z) di questo punto, possiamo risolvere il sistema di equazioni basato sulle equazioni di proiezione prospettica per entrambe le telecamere.

$$x = \frac{b(u_l - o_x)}{(u_l - u_r)}$$

$$y = \frac{b\, f_x\, (u_l - o_y)}{f_y\, (u_l - u_r)}$$

$$z = \frac{b\, f_x}{(u_l - u_r)}$$

Notiamo che nel denominatore di ciascuna di queste equazioni appare il termine $(u_l - u_r)$, che è la differenza tra le coordinate “u” del punto nell'immagine di sinistra e nell'immagine di destra.

Questa differenza è chiamata "disparità", ed è un'informazione chiave che ci permette di calcolare la profondità del punto nella scena.

È importante sottolineare che disparità e base sono linearmente proporzionali:

$$u_l - u_r = b \frac{f_x}{z}$$

Questo aspetto permette alcune riflessioni aggiuntive:

1. A z fissata, aumentare la baseline (b) determina un aumento della disparità
2. A baseline (b) fissata, la disparità aumenta con il diminuire della distanza z dal punto di osservazione

È proprio grazie a quest'ultima osservazione che è possibile determinare una mappa di profondità: i punti a disparità maggiore saranno quelli a z minore, mentre i punti più lontani avranno una disparità minore. Tutto questo è rappresentabile sul piano immagine attraverso una mappa a singolo canale chiamata disparity map, che esprime una relazione del tipo:

$$Z^2 \rightarrow Z[0, 255] \; per \; disparity \; a \; 8bit$$

Figura 31: Esempio di disparity map calcolata in riferimento all'immagine di sinistra

In questa breve analisi abbiamo però sorvolato su un importante punto: l'immagine di destra e l'immagine di sinistra sono due immagini diverse, acquisite da due sensori diversi; quindi, non esiste una correlazione diretta tra quelli che sono i pixel relativi allo stesso punto dello spazio tridimensionale impressionato dalle camere.

In altre parole, serve trovare un algoritmo che preso un pixel (u_l, v_l) sull'immagine di sinistra, trovi il suo corrispondente (u_r, v_r) sull'immagine di destra.

Questi algoritmi prendono il nome di "algoritmi di template matching" e sfruttano le stesse logiche implementate nei codificatori di sequenze video MPEG (in questi casi definiti "algoritmi di block matching").

Vediamo ora le logiche che sottendono questi algoritmi:

Sappiamo che tra immagine di destra e immagine di sinistra non esiste disparità lungo l'asse v, infatti:

$$v_l = v_r = f_y \frac{y}{x} + o_y$$

Quindi possiamo immaginare che i pixel corrispondenti ad uno stesso punto, giacciano sulla stessa linea orizzontale, come mostrato in Figura 32.

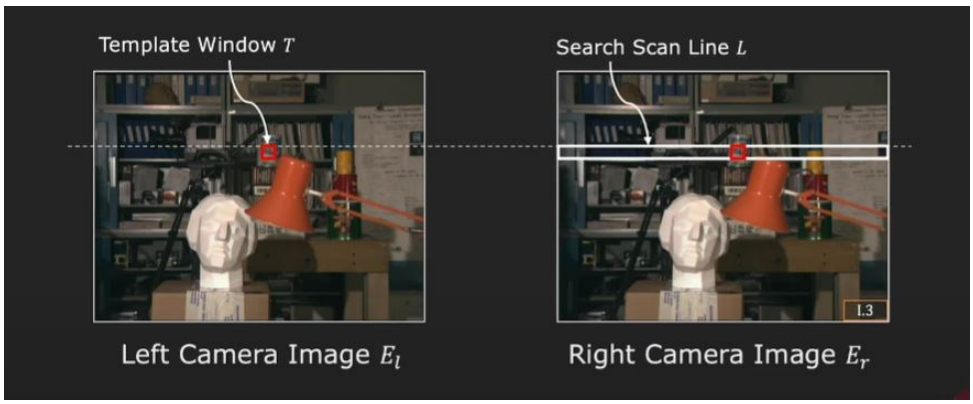

Figura 32: visualizzazione grafica dell'algoritmo di template matching. In rosso si può osservare il template di interesse, mentre il rettangolo bianco indica la zona di interesse nell'immagine di destra, all'interno della quale avviene la ricerca del match.

La ricerca del match tra i pixel dell'immagine di sinistra con quelli dell'immagine di destra avviene per blocchi di T x T pixel, chiamati templates.

L'algoritmo lavora iterativamente su tutti i templates del blocco di ricerca (evidenziato in bianco in Figura 32), in funzione della metrica di matching scelta.

Esistono diverse metriche possibili, ne riportiamo due:

Indichiamo con (k, l) le coordinate del centro del template, con L la dimensione orizzontale della finestra di ricerca sull'immagine di destra, con T la dimensione del template, e con $(\bar{k}, \bar{l})$ le coordinate del pixel centrale del template di matching, quindi:

1. SOMMA DELLE DIFFERENZE IN VALORE ASOLUTO (SAD)

$$\left(\bar{k}, \bar{l}\right) \in L \mid SAD\left(\bar{k}, \bar{l}\right) = Min(SAD(k, l))$$

con

$$SAD(k, l) = \sum_{(i,j) \in T} |E_l(i, j) - E_r(i + k, j + l)|$$

2. SOMMA DEL QUADRATO DELLE DIFFERENZE (SSD)

$$\left(\bar{k}, \bar{l}\right) \in L \mid SSD\left(\bar{k}, \bar{l}\right) = Min(SSD(k, l))$$

con

$$SSD(k, l) = \sum_{(i,j) \in T} (E_l(i, j) - E_r(i + k, j + l))^2$$

Avendo concluso l'approfondimento teorico dei concetti necessari a comprendere i capitoli successivi, analizzeremo di seguito l'implementazione pratica del calcolo della disparity map e della xyz_map.

I concetti teorici sopra descritti sottendono infatti le logiche implementate nella libreria dinamica Depth_Utility, il cui compito è quello di calcolare una mappa di profondità a partire da due immagini stereoscopiche.

3.2.4 Creazione della libreria "Depth_Utility" in C++

Analizziamo di seguito i vari step che hanno portato dalla semplice immagine stereoscopica all'ottenimento della mesh della mano.

Come per le diverse logiche modellate all'interno dell'intero sistema Endosight, abbiamo optato per la generazione di una libreria dinamica, che implementasse al suo interno la logica di generazione di una mappa di profondità, che abbiamo chiamato xyz_map (spiegheremo di seguito la sua definizione). Questa libreria ha preso il nome di "Depth_Utility", in quanto deputata alla generazione di una mappa di profondità.

La suddivisione delle logiche all'interno di sistemi informatici complessi, in diverse librerie dinamiche (.dll) è una pratica essenziale che fornisce numerosi benefici. Questa organizzazione del codice porta al **principio di singola responsabilità**, dove ogni modulo o libreria è responsabile per una singola parte del funzionamento del sistema.

"Il principio di singola responsabilità si applica alla programmazione orientata agli oggetti, ma può anche essere considerato un principio architetturale simile alla separazione dei concetti. Afferma che gli oggetti devono avere una sola responsabilità e un solo motivo per cambiare [10] *"*.

[10] Fonte: principi architetturali per gli applicativi informatici moderni, Microsoft.
Consultabile al seguente link: https://learn.microsoft.com/it-it/dotnet/architecture/modern-web-apps-azure/architectural-principles

In primo luogo, si tratta di promuovere la riutilizzabilità del codice: quando una specifica funzionalità è incapsulata all'interno di una libreria, può essere facilmente riutilizzata in diverse parti del sistema senza la necessità di duplicare il codice. Questo riduce non solo la quantità di codice da scrivere, ma limita anche la possibilità di errori e rende più semplice la manutenzione del codice nel lungo termine.

In secondo luogo, il codice all'interno delle librerie .dll è "cieco" rispetto a ciò che accade nel resto del sistema. Questo livello di incapsulamento garantisce che i dettagli dell'implementazione di una libreria siano nascosti al resto del sistema, fornendo un'interfaccia pulita e coerente.

Questo è particolarmente importante in un sistema complesso, in cui la comprensione di ogni dettaglio di implementazione potrebbe essere scoraggiante o semplicemente non pratica.

Infine, per squadre di sviluppo molto grandi, la suddivisione del codice in librerie distinte può favorire un più efficace lavoro di squadra e una gestione più efficiente del progetto. Ogni libreria può essere sviluppata, testata e mantenuta da un sotto-team o un individuo differente, in maniera tale che ciascuna sezione di sviluppo si concentri su un aspetto specifico del sistema. Questo può migliorare la produttività, la qualità del codice e la velocità di sviluppo.

3.2.4.1 Perché in C++

L'infrastruttura informatica di Endosight presenta due linguaggi di programmazione:

C++ e C#.

Le sezioni di infrastruttura implementate in C++ sono tendenzialmente quelle che si occupano dei tasks a basso livello e che necessitano di una elevata efficienza computazionale, come l'acquisizione, l'elaborazione delle immagini provenienti dalla stereocamera e il tracking degli ARUCO markers.

In C# sono invece sviluppati i layers relativi all'interfaccia utente: la visualizzazione

dell'ambiente di realtà aumentata avviene infatti mediante la finestra di "game" di Unity. Vengono quindi gestiti in C# (linguaggio nativo di Unity):

1. L'importazione nella scena dei file con estensione .obj relativi alle mesh degli organi interni
2. L'assegnazione dei materiali
3. La gestione della posa e della scala corretta degli stessi
4. L'invocazione dei metodi delle librerie C# costruite grazie a dei wrappers delle .dll sviluppate in C++. Questi metodi permettono di ottenere i dati elaborati lato C++ all'intero dell'ambiente C# di Unity.

Relativamente alla logica di calcolo della mappa di profondità è stato optato l'utilizzo della libreria OpenCV, nativa in C++. Questa .dll fornisce infatti dei metodi per il calcolo di una dipsarity map a partire da due immagini stereoscopiche e per la sua implementazione per il calcolo della xyz_map.

Esistono alcuni wrapper di OpenCV, come ad esempio Emgu CV, che permettono di fare chiamate alla libreria C++ direttamente all'interno dell'ambiente C#, tuttavia, si è scelto di sviluppare la libreria DepthUtility direttamente in C++, per ottenere il massimo delle prestazioni indiscriminatamente dal compito da svolgere e poter gestire al meglio la memoria.
In C++, infatti, l'allocazione e la de-allocazione della memoria è completamente gestita dal programmatore. Ciò permette di averne un controllo granulare, rendendo possibile ottimizzare la sua allocazione in base alle specifiche esigenze del programma. Questo può portare a una notevole efficienza e velocità, specialmente in operazioni computazionalmente onerose come l'elaborazione delle immagini.

3.2.4.2 Logica incapsulata nella libreria Depth_Utility

La libreria Depth_Utility, che è stata sviluppata in questo progetto, offre un'ampia gamma di funzionalità per calcolare la mappa di disparità (una rappresentazione della profondità) e la mappa xyz (una rappresentazione 3D dalle immagini stereoscopiche). Questa libreria è particolarmente utile perché fornisce funzioni per utilizzare diversi metodi per il calcolo della disparità, inclusi StereoBM (Block Matching), StereoSGBM

(Semi-Global Block Matching) e le loro controparti basate su CUDA, consentendo l'elaborazione accelerata dalla GPU (analizzeremo questi metodi nei paragrafi successivi).

Oltre a ciò, la libreria offre, attraverso la sua interfaccia, un set di parametri per il tuning di questi metodi, inclusi il numero di disparità, la dimensione del blocco TxT e i limiti di disparità (minimo e massimo). Questi parametri sono settabili da interfaccia di Unity, per migliorare la qualità della mappa di disparità prodotta e di conseguenza la xyz_map.

Inoltre, la libreria fornisce funzionalità per l'applicazione di operazioni di post-elaborazione sulla mappa di disparità. Queste includono l'operazione di chiusura morfologica, per rimuovere i piccoli buchi nella mappa di disparità, e l'applicazione di un filtro di smoothness per migliorare l'aspetto generale della mappa.

Infine, la libreria permette l'estrazione di una mappa di confidenza, che fornisce una misura di quanto si può avere fiducia in ciascun valore di disparità calcolato. Questa mappa di confidenza può essere utilizzata per ignorare i valori di disparità poco affidabili durante l'elaborazione successiva.

In sintesi, la libreria Depth_Utility fornisce una serie completa di strumenti per il calcolo della profondità da immagini stereoscopiche, permettendo agli sviluppatori di scegliere tra vari metodi di calcolo della disparità, ottimizzare i loro parametri, applicare operazioni di post-elaborazione, e ottenere una mappa di confidenza.

L'output principale della libreria è la xyz_map, che consiste in un vettore di punti tridimensionali che corrispondono alla triangolazione dei pixel delle immagini stereoscopiche; essa è costruita sulla disparity map, la quale, a differenza della precedente, non fornisce reali indicazioni sulla posizione dei punti nello spazio tridimensionale, ma associa dei valori di profondità (in z) in maniera relativa.

Il passaggio da disparity a xyz avviene grazie all'applicazione dei parametri delle camere, che abbiamo precedentemente indicato con o_x *e* o_y.

La mappa di disparità appare nel seguente modo:

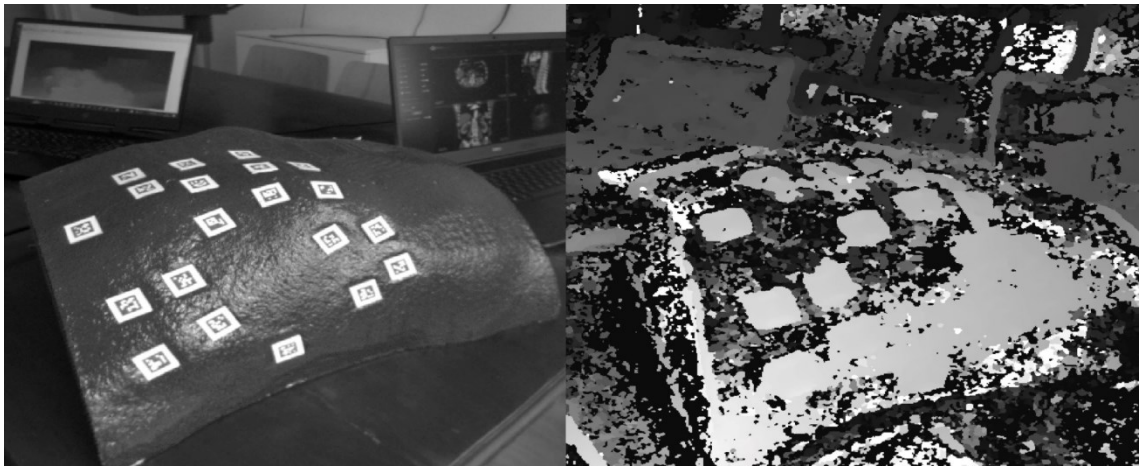

Figura 33: Mappa di disparità generata dai dati stereoscopici e riferita alla camera di sinistra

3.2.4.3 Algoritmi di stereo matching di OpenCV

È utile fare un breve excursus sugli algoritmi di stereo matching messi a disposizione da OpenCV per comprendere pregi e difetti di ognuno:

OpenCV offre diverse tecniche per l'esecuzione dello stereo matching, tra cui Block Matching (BM), Semi-Global Block Matching (SGBM) e Graph Cuts (GC). Questi algoritmi sono implementati come classi con un metodo “compute()” che prende come input un paio di immagini stereo e restituisce un'immagine di disparità. Queste tecniche si basano su principi diversi e hanno diversi requisiti di input e impatto sui risultati.

1. Block Matching (BM): Questo è l'approccio più semplice utilizzato per lo stereo matching. Funziona confrontando piccoli blocchi o finestre di pixel tra due immagini. Per ogni blocco nell'immagine sinistra, l'algoritmo cerca un blocco corrispondente sull'immagine destra lungo la stessa linea di scansione. L'output è la disparità, calcolata come la differenza tra le posizioni del blocco su entrambe le immagini. Nonostante la sua semplicità, l'algoritmo BM può avere problemi con le aree a texture poco riconoscibile, le superfici lucide e le regioni in cui l'illuminazione varia tra le due viste. Le dimensioni del blocco e il range di disparità sono tra gli input più importanti che influenzano i risultati di questo algoritmo.

2. Semi-Global Block Matching (SGBM): Questo è un miglioramento dell'algoritmo BM. Mentre BM considera solo le informazioni locali (cioè, all'interno di ciascun

blocco), SGBM prende in considerazione anche le informazioni globali sull'intera immagine. Minimizza una funzione di costo che tiene conto non solo della corrispondenza tra blocchi, ma anche della coerenza tra i valori di disparità dei blocchi vicini. Questo permette di ottenere risultati più precisi, specialmente nelle aree con texture poco riconoscibile. Tuttavia, l'algoritmo SGBM è computazionalmente più oneroso rispetto a BM. Anche qui, le dimensioni del blocco e il range di disparità sono input cruciali.

3. Graph Cuts (GC): Questo metodo rappresenta il problema dello stereo matching come un problema di ottimizzazione su un grafo. Ogni nodo del grafo rappresenta un pixel dell'immagine e i bordi del grafo rappresentano le relazioni di vicinanza tra i pixel. La funzione di costo da minimizzare tiene conto sia della corrispondenza tra i pixel (costo di dati) sia della coerenza tra i valori di disparità dei pixel vicini (costo di smoothing). La minimizzazione della funzione di costo viene eseguita tagliando il grafo in modo da dividere i nodi (i pixel) in due gruppi che corrispondono alle due viste. Questo algoritmo può produrre risultati molto accurati, ma è anche computazionalmente più costoso rispetto sia a BM che a SGBM.

Nell'implementazione di OpenCV, ognuno di questi algoritmi ha una serie di parametri che possono essere regolati per adattarsi alle specifiche esigenze di un progetto. Questi includono parametri come le dimensioni del blocco, il range di disparità, e vari parametri specifici dell'algoritmo che influenzano il costo dei dati e il costo di smoothing nel caso di SGBM e GC. Il tuning di questi parametri può avere un impatto significativo sulla qualità dei risultati di disparità prodotti da ciascun algoritmo.

3.2.4.4 Struttura e Interfaccia della Libreria

La libreria Depth_Utility.dll presenta la seguente struttura:

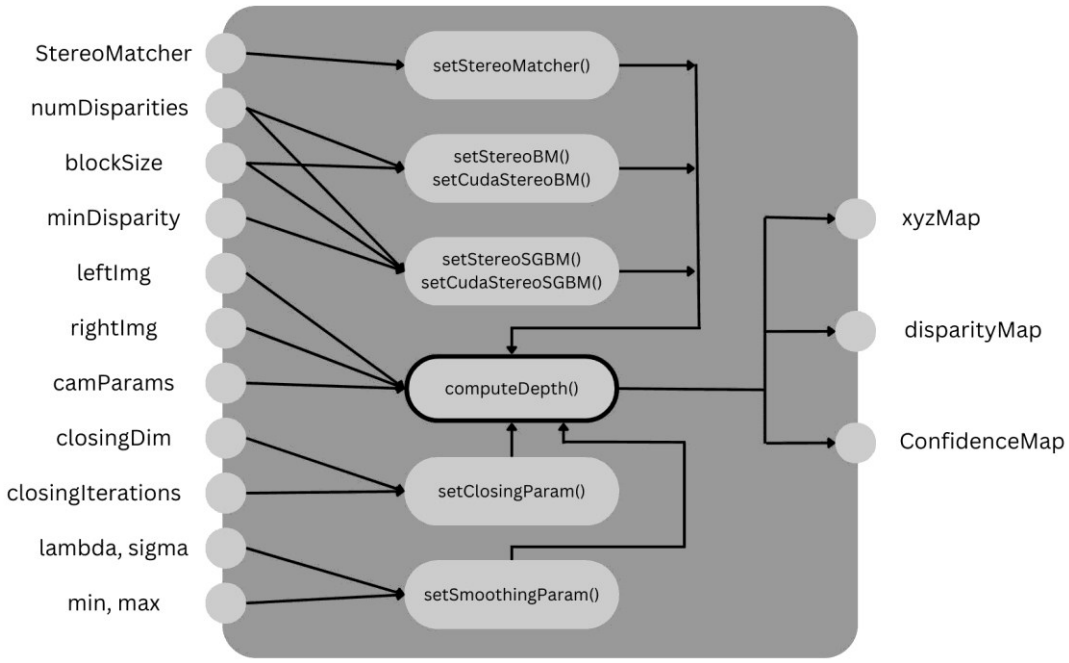

Figura 34 : schema relativo alla libreria Depth_Utility

Si possono osservare una serie di Input sulla sinistra, gli output sulla destra e la logica dei metodi implementati al suo interno.

Partiamo dall'analisi dei metodi per comprendere il funzionamento della libreria:

Dall'interfaccia della libreria si può accedere a otto metodi; sette servono a settare dall'esterno i parametri necessari al calcolo della disparity map e della xyz_map, questi parametri sono modellati come variabili di classe, sovrascritte dai metodi sopra citati. L'ottavo metodo, che presenta signature "*computeDepth()*" triggera, in funzione dei parametri settati precedentemente, il calcolo effettivo delle tre mappe riportate come output nel grafico (disparity, xyz, confidence).

1. *setStereoMatcher()*
   serve a passare all'interno della libreria un oggetto della classe stereoMatcher di OpenCV, nel caso lo si avesse creato esternamente alla libreria stessa.

2. *setStereoBM()*
   Questo metodo, se invocato, crea un oggetto `cv::stereoBM` che implementa l'algoritmo di Block Matching analizzato precedentemente e lo assegna alla variabile di classe omonima.

   Questo metodo necessita di due parametri, *[da documentazione ufficiale di OpenCV, consultabile al seguente link: https://docs.opencv.org/3.4/d9/dba/classcv_1_1StereoBM.html]*:
   - *[int] numDisparities:* l'intervallo di ricerca della disparità. Per ciascun pixel, l'algoritmo troverà la migliore disparità da 0 (disparità minima predefinita) a numDisparities. L'uso di numDisparities permette di impostare l'intervallo di disparità che l'algoritmo dovrebbe considerare durante la ricerca del corrispondente migliore per ciascun pixel, fornendo un mezzo per regolare la sensibilità dell'algoritmo alle variazioni di profondità nell'immagine. Valori alti di questo parametro permettono il calcolo di disparità anche per oggetti molto lontani (ovvero quelli a disparità alta).
   - *[int] blockSize:* La dimensione lineare dei blocchi confrontati dall'algoritmo, ovvero le dimensioni che nei paragrafi precedenti abbiamo indicato con T. La dimensione dovrebbe essere dispari, poiché il blocco è centrato sul pixel corrente. Una dimensione del blocco più grande implica una mappa di disparità più uniforme, sebbene meno precisa. Una dimensione del blocco più piccola fornisce una mappa di disparità più dettagliata, ma c'è una maggiore possibilità che l'algoritmo trovi una corrispondenza errata, il che si tramuta in una mappa "più rumorosa".

3. *setCudaStereoBM()*
   Questo metodo è la controparte di setStereoBM(). Prevede le stesse logiche, ma permette il calcolo della disparità tramite accelerazione grafica in GPU.

4. *setStereoSGBM()*
   Questo metodo, se invocato, crea un oggetto `cv::stereoSGBM` che implementa l'algoritmo di Semi-Global Block Matching analizzato precedentemente e lo assegna alla variabile di classe omonima.

   Questo metodo necessita di undici parametri, *[da documentazione ufficiale di OpenCV, consultabile al seguente link: https://docs.opencv.org/3.4/d2/d85/classcv_1_1StereoSGBM.html]*:
   - *[int] minDisparities:* Permette di settare anche il limite inferiore dell'intervallo di ricerca della disparità.
   - *[int] numDisparities*
   - *[int] blockSize*
   - *[int] P1:* Il primo parametro che controlla la fluidità della disparità. Si tratta di una penalità sul cambiamento della disparità di più o meno 1 tra pixel vicini. Aiuta a ridurre il rumore nella mappa di disparità prevenendo piccoli cambiamenti di disparità tra pixel vicini.
   - *[int] P2*: Il secondo parametro che controlla la fluidità della disparità. Più grandi sono i valori, più fluida è la disparità. P2 è la penalità sulla variazione di disparità di più di 1 tra pixel vicini. L'algoritmo richiede P2 > P1, per scoraggiare i grandi cambiamenti di disparità tra pixel vicini, il che potrebbe portare a una mappa di disparità discontinua.
   - *[int] disp12MaxDiff:* Questo parametro è utilizzato per eseguire un controllo di coerenza tra la mappa di disparità calcolata dall'immagine sinistra verso destra e quella calcolata dall'immagine destra verso sinistra. Se la differenza tra queste due mappe di disparità in un dato punto supera disp12MaxDiff, allora quel punto viene considerato non affidabile e la sua disparità viene impostata a -1.
   - *[int] preFilterCap:* Valore di troncamento per i pixel dell'immagine pre-filtrata. L'algoritmo calcola prima la derivata x in ogni pixel e ne tronca il valore nell'intervallo [-preFilterCap, preFilterCap]. I valori risultanti vengono passati alla funzione di costo dei pixel di Birchfield-Tomasi.
   - *[int] uniquenessRatio:* Margine in percentuale con cui il miglior valore (minimo) della funzione di costo calcolata dovrebbe "vincere" il secondo miglior valore per considerare corretta la corrispondenza trovata. Questo

parametro viene utilizzato per garantire che la disparità scelta per un dato pixel sia significativamente migliore (ossia abbia un costo inferiore) rispetto alla seconda migliore disparità. Questo aiuta a prevenire l'assegnazione di disparità errate.

- *[int] speckleWindowSize:* Dimensione massima delle regioni di disparità uniforme da considerare come macchie di rumore da invalidare.
- *[int] speckleRange:* Variazione massima di disparità all'interno di ogni componente connesso.
- *[int] mode:* Questo parametro determina quale variante dell'algoritmo StereoSGBM verrà utilizzata. `StereoSGBM::MODE_HH` è l'algoritmo a due passaggi a piena scala, che produce risultati migliori ma utilizza più memoria. Se questo parametro è impostato su false, viene utilizzata una versione più efficiente dal punto di vista della memoria, ma potenzialmente meno accurata.

5. *setCudaStereoSGBM()*

   Questo metodo è la controparte di setStereoSGBM(). Prevede le stesse logiche, ma permette il calcolo della disparità tramite accelerazione grafica in GPU.

6. *setClosingParam()*

   Questo metodo permette di abilitare l'utilizzo di algoritmi di chiusura morfologica sulla disparity map. Come vedremo in seguito, la disparity map è fortemente affetta da macchie e zone a disparità errata, riuscire a chiudere queste macchie comporterebbe un notevole miglioramento sul risultato finale. Nei paragrafi successivi sarà approfondito il tema della chiusura morfologica.

   Questo metodo necessita di due parametri:

   - *[int] closingDim*: dimensione lineare dell'operatore morfologico utilizzato. Nel nostro caso l'operatore morfologico è di forma quadrata. A dimensioni elevate corrisponde una chiusura più efficace, ma una maggiore perdita di dettaglio nell'immagine.

- *[int]* closingIterations: numero di volte in cui viene iterato l'algoritmo di chiusura morfologica sulla disparity map. Un numero elevato di iterazioni chiude "buchi" o "macchie" di dimensioni maggiori, ma rischia di denaturare la struttura della disparity map stessa.

7. *setSmoothingParam()*

   Questo metodo permette di abilitare l'utilizzo di algoritmi di filtering sulla disparity map. Come vedremo in seguito, la disparity map è fortemente affetta da macchie e zone a disparità errata, riuscire a chiudere queste macchie comporta un notevole miglioramento sul risultato finale.
   Nei paragrafi successivi sarà approfondito il tema del filtraggio della disparity map.

   Questo metodo necessita di quattro parametri:

   - *[int]* ***lambda***: Controlla l'importanza che viene data alla regolarizzazione durante il filtraggio, cioè l'importanza data alla diffusione dell'informazione di disparità tra i pixel vicini. Un valore di Lambda più alto darà più importanza alla regolarizzazione, cioè alla diffusione dell'informazione, rendendo l'immagine finale più liscia ma potenzialmente anche meno accurata se il valore è troppo elevato.

   - *[float]* **sigma**: Questo parametro controlla il raggio del filtro bilaterale, cioè quanto lontano il filtro cerca i pixel da considerare nel calcolo del valore del pixel corrente. Un valore di Sigma più alto significa che il filtro considererà i pixel più lontani, il che può contribuire a ridurre il rumore ma anche a sfocare i dettagli fini dell'immagine.

3.2.4.5 Operazione Morfologica di Chiusura

L'operazione morfologica di chiusura è una tecnica di elaborazione delle immagini che è particolarmente efficace nel rimuovere i piccoli buchi, senza alterare significativamente la dimensione globale dell'oggetto stesso. Inoltre, può aiutare a colmare piccole lacune tra parti disconnesse di un oggetto. rappresenta una sequenza di due operazioni fondamentali: dilatazione ed erosione.

Per implementare il Closing è stato sufficiente utilizzare i metodi messi a disposizione dalla libreria OpenCV; in particolare il metodo `cv::morphologyEx()` a cui sono stati passati i parametri mostrati in Figura 34.

L'output di questa funzione una disparity map che presenta una maggiore omogeneità :

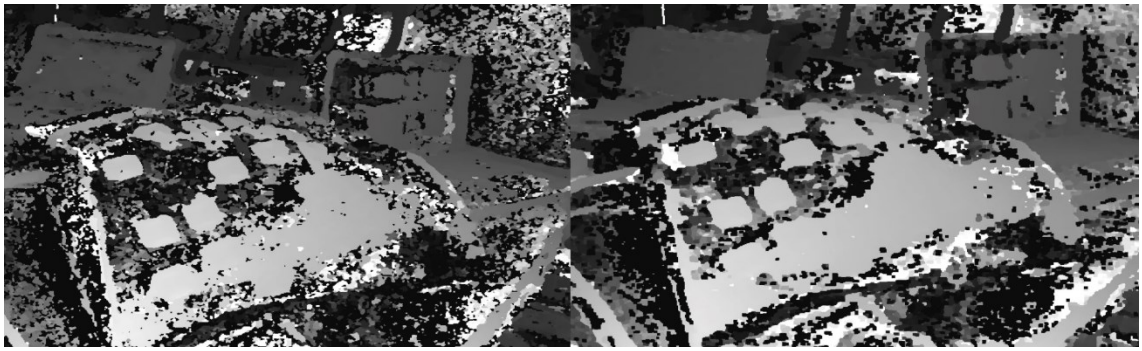

Figura 35: Immagine comparativa tra disparity map originale (sulla sinistra) e disparity map con closing applicato (sulla destra). Parametri di closing utilizzati: closingDim = 2x2px, closingIterations = 8.

3.2.4.6 Operazione di Smoothing sulla Disparity

Lo "smoothing" della disparity map è un processo di filtraggio che mira a ridurre il rumore e a migliorare la qualità generale della disparity map prodotta da un algoritmo di stereovisione. Questo processo di filtraggio è fondamentale perché le mappe di disparità generate direttamente dagli algoritmi di matching di stereovisione contengono spesso rumore ed errori, che possono derivare da varie fonti come l'inconsistenza dell'illuminazione tra le due immagini di destra e di sinistra, l'occlusione, e le texture ripetitive.

Nel contesto di OpenCV, lo smoothing della disparity map può essere ottenuto utilizzando il filtro di ponderazione minima locale, noto come filtro WLS.

Il filtro WLS è un tipo di filtro edge-preserving utilizzato per la riduzione del rumore nelle immagini. Questi filtri sono progettati per preservare i bordi nelle immagini durante il processo di riduzione del rumore.

Il filtro WLS funziona minimizzando una somma pesata dei quadrati delle differenze locali tra l'immagine originale e l'immagine filtrata. I pesi utilizzati in questa somma sono determinati da una funzione di distanza, che tende a dare un peso maggiore alle differenze tra pixel vicini che hanno un colore simile, e un peso minore alle differenze tra pixel vicini con colori diversi. Questo comportamento è quello che permette al filtro di preservare i bordi nell'immagine, dal momento che i bordi di un'immagine sono spesso definiti da cambiamenti bruschi nel colore.

OpenCV mette a disposizione una classe che modella il filtro WLS: `cv::Ptr<cv::ximgproc::DisparityWLSFilter>`.

La classe mette a disposizione due parametri utili a gestire l'impatto che il filtro ha sulla mappa in output:

1. **lambda**: Questo parametro è un moltiplicatore per la penalità di discontinuità. In altre parole, influenza quanto il filtro cerca di preservare i bordi nella disparity map. Un valore più alto di lambda incoraggia un risultato più liscio, riducendo la variazione di disparità tra i pixel vicini. Ad esempio, se lambda è impostato su un valore molto alto, il filtro lavorerà per minimizzare le differenze di disparità tra i pixel vicini, anche se ciò significa ignorare le differenze presenti nella disparity map originale. Al contrario, un valore più basso di lambda consentirà al filtro di preservare più dettagli a scapito di una maggiore rumore nella mappa finale.
2. **sigma**: Questo parametro è un moltiplicatore per la penalità basata sui colori. Regola la sensibilità del filtro alle differenze di colore tra pixel adiacenti. Se sigma è alto, le differenze di colore saranno più influenti nel determinare se due pixel dovrebbero avere una disparità simile. Ad esempio, se sigma è impostato su un valore molto alto, il filtro cercherà di preservare i bordi presenti nell'immagine originale, in cui i pixel su entrambi i lati del bordo hanno colori molto diversi. Se sigma è impostato su un valore basso, le differenze di colore saranno meno significative e il filtro darà più importanza alla continuità di disparità.

In Figura 36 si può osservare il risultato del filtro di smoothing applicato alla disparity map:

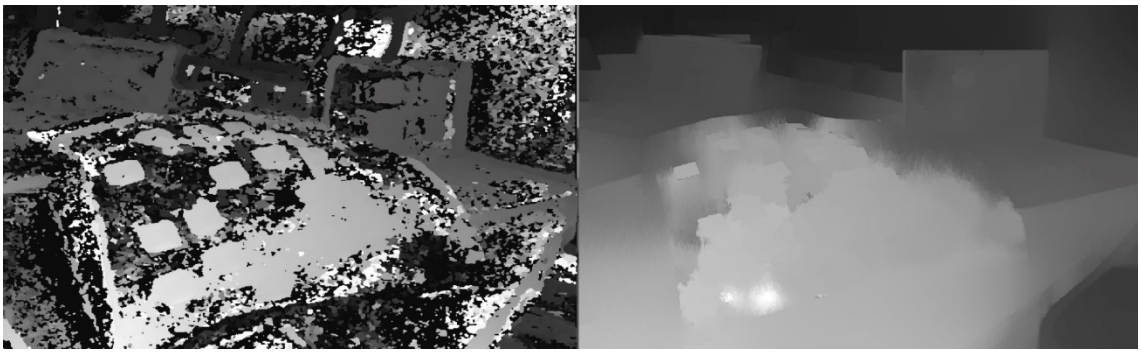

Figura 36: Disparity map filtrata (sulla destra) ottenuta applicando il filtro WLS sulla disparity map originale (sulla sinistra). I parametri applicati al filtro sono stati: lambda = 12000, sigma = 1.5

3.2.4.7 Da Disparity a xyz_map

Sempre all'interno della libreria dinamica Depth_Utility avviene la conversione da disparity map a xyz_map. Come precedentemente evidenziato la differenza sostanziale tra le due consiste nel fatto che la disparity map contiene soltanto informazioni di profondità relative, all'interno del dominio dell'immagine, mentre la xyz_map è un vettore di float che contiene le coordinate (x, y, z) dei punti dell'immagine, descritti nel sistema di riferimento camera.

Nel nostro caso la disparity map è stata costruita in riferimento alla camera di sinistra; quindi, l'origine del sistema di riferimento sarà il centro ottico di quest'ultima.

In termini di implementazione, la prima operazione è convertire la mappa di disparità in un formato a singolo canale con valori a virgola mobile (`CV_32FC1`). Questo viene fatto attraverso la chiamata al metodo `convertTo()`, che modifica il tipo di dati dell'immagine di disparità, inizialmente rappresentata come un'immagine di interi a 16 bit, in un'immagine di floating point a 32 bit.

Subito dopo, l'immagine di disparità viene scalata dividendo tutti i valori per 16.0. Questa operazione è necessaria perché l'immagine di disparità viene spesso calcolata in un formato fisso, con un fattore di scala per aumentare la precisione. La divisione riduce i valori al loro valore originale, semplificando le operazioni successive.

Per memorizzare i punti 3D risultanti dalla proiezione, viene preparata una matrice _xyzMap di tipo `CV_32FC3`, che rappresenta un'immagine a tre canali con valori float a 32 bit.

Una volta preparate le strutture dati, la mappa di disparità viene utilizzata per generare la mappa di punti 3D. Questo viene fatto mediante la funzione di OpenCV `cv::reprojectImageTo3D()`, che prende come input l'immagine di disparità e una matrice di proiezione Q.

Quest'ultima, definita precedentemente, contiene le informazioni necessarie per trasformare la disparità in profondità.

La matrice di proiezione Q è definita come segue:

$$\begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -\frac{1}{T_x} & \frac{c_x - c'_x}{T_x} \end{bmatrix}$$

- c_x e c_y sono i punti principali dell'immagine, che sono le coordinate del punto centrale dell'immagine. Questi valori sono utilizzati per centrare l'origine della mappa di disparità.
- f è la lunghezza focale della telecamera, espresso in termini di pixel. Questo valore è utilizzato per scalare i valori di disparità, tenendo conto delle caratteristiche fisiche della telecamera.
- Tx è la traslazione sull'asse x tra le due telecamere, che viene utilizzata per calcolare la profondità dei punti nella mappa 3D. Il segno negativo in $-\frac{1}{T_x}$ è dovuto al fatto che una disparità positiva corrisponde a una profondità positiva, ma la camera destra (da cui proviene la disparità) è traslata negativamente rispetto alla camera sinistra.

3.2.5 Triangolazione della nuvola di punti

[Nota: da questo paragrafo in avanti ci riferiremo con il termine "triangolazione" alla tecnica con la quale è possibile ottenere una mesh poligonale partendo da una nuvola di punti tridimensionali. Nei capitoli precedenti, con questo termine, ci si era riferiti al calcolo delle mappe di profondità partendo da delle immagini stereoscopiche.]

In questo sotto-capitolo verranno analizzati i passi che han portato alla realizzazione della mesh delle mani a partire dalla xyz_map.

Le logiche che vedremo in questo sotto-capitolo sono state tutte implementate lato C#; infatti, la libreria Depth_Utility, sviluppata in C++, viene utilizzata in questo ambiente tramite un wrapper.

Lo scambio di dati avviene tramite dei puntatori che puntano il primo indirizzo di memoria della variabile da scambiare. Il puntatore è infatti un costrutto che entrambi gli ambienti gestiscono con le stesse logiche.

Lato C# viene quindi inclusa la libreria Depth_Utility.cs (wrapper di quella C++) e vengono sfruttati i suoi metodi per ottenere la xyz_map, che in questo ambiente assume la forma di un vettore di float.

3.2.5.1 Filtraggio pre-triangolazione

All'interno della xyz_map possono comparire valori numerici non validi come dei NaN (Not a Number); la causa è imputabile a diversi fattori:

1. La presenza di regioni di occlusione (aree visibili in una delle due immagini stereoscopiche ma non nell'altra).
2. Errori nel calcolo della disparità (ad esempio a causa di rumore o di texture povere)
3. Pixel per i quali la corrispondenza non è stata trovata.
4. Il calcolo delle coordinate x, y, z è ascrivibile ad una divisione, in cui la disparità compare al denominatore. Se la disparità calcolata è zero o molto vicina allo zero, il risultato sarà infinito o non definito, che è solitamente rappresentato come NaN in molti sistemi di calcolo.

È importante filtrare questi valori prima di attuare la triangolazione dei punti al fine di evitare di incorrere in eccezioni durate la fase di triangolazione.

Non è possibile semplicemente eliminare i dati in questione, poiché è di vitale importanza mantenere la xyz_map come una struttura dati organizzata; si è deciso quindi di portare tutti i valori non finiti a zero.

3.2.5.2 Come si crea una mesh in Unity

Per generare una mesh 3D in Unity, il motore di rendering richiede una serie di dati specifici che definiscono la struttura geometrica e visiva dell'oggetto. Questi dati sono organizzati all'interno di un oggetto di tipo Mesh, il quale contiene fondamentalmente quattro elementi chiave:

1. **Vertici**: Un array di punti 3D (rappresentati come Vector3) che definisce i vertici della mesh. Ogni punto è definito dalle sue coordinate nello spazio tridimensionale.

2. **Triangoli**: Un array di interi che definisce gli indici dei vertici che compongono ciascun triangolo. Ogni triangolo è rappresentato da una tripletta di numeri che corrispondono agli indici dei vertici nell'array dei vertici. L'ordine di questi indici determina l'orientamento del triangolo (orario o antiorario), che a sua volta determina il verso in cui il triangolo sarà visibile.

3. **Normali**: Un array di vettori 3D che rappresenta le normali ai vertici. Le normali sono versori perpendicolari alla superficie della mesh nei punti corrispondenti ai vertici e sono utilizzate per calcolare l'illuminazione e l'ombreggiatura dell'oggetto.

4. **Coordinate UV**: Un array di coordinate 2D utilizzato per mappare le texture sulla superficie della mesh. Ogni coordinata UV corrisponde a un punto dell'immagine della texture che sarà mappata sul corrispondente vertice della mesh.

È importante sottolineare la relazione che esiste tra il vettore dei vertici e quello dei triangoli; essendo quest'ultimo un costrutto che semplicemente raggruppa i vertici del primo vettore in triplette, grazie all'indice che i vertici stessi occupano all'interno dell'array di vertici.

Vediamo un esempio:

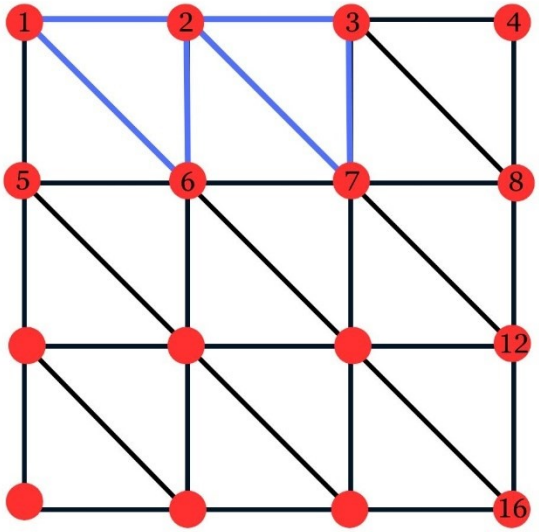

Figura 37

In riferimento alla Figura 37, immaginiamo di avere una griglia ordinata di vertici e di dover generare due triangoli (in blu in Figura).

In questo caso il vettore di vertici assumerà la seguente forma:

$$\text{Vetcor3[] vertices} = \begin{bmatrix} Vertex1 \\ Vertex2 \\ Vertex3 \\ Vertex4 \\ Vertex5 \\ Vertex6 \\ \vdots \\ Vertex16 \end{bmatrix} = \begin{bmatrix} (X1; & Y1; & Z1) \\ (X2; & Y2; & Z2) \\ (X3; & Y3; & Z3) \\ (X4; & Y4; & Z4) \\ (X5; & Y5; & Z5) \\ (X6; & Y6; & Z6) \\ & \vdots & \\ (X16; & Y16; & Z16) \end{bmatrix}$$

Il vettore dei triangoli assumerà di conseguenza la seguente forma:

$$\text{Int[] triangles} = \begin{bmatrix} 1 \\ 6 \\ 2 \\ 2 \\ 7 \\ 3 \\ \vdots \\ 12 \end{bmatrix} \begin{matrix} \\ \\ \text{Tris 1} \\ \\ \\ \text{Tris 2} \\ \\ \end{matrix}$$

L'ordine di inserimento degli indici all'interno della singola tripletta è un parametro che guida la generazione della normale associata al triangolo, seguendo la "regola della mano destra":

1. Inserimento in senso antiorario: normale uscente dal piano
2. Inserimento in senso orario: normale entrante nel piano

Per creare una mesh in Unity, si crea prima un oggetto Mesh; quindi, si popolano questi array con i dati appropriati e si passano come proprietà dell'oggetto mesh. Infine, si assegna l'oggetto Mesh a un componente MeshFilter o MeshRenderer su un GameObject.

In questo progetto sono stati ignorati il vettore delle normali e quello delle coordinate UV, in quanto essi rappresentano informazioni utili per lo shading e quindi non funzionali allo scopo finale della mesh generata. Ricordiamo ed anticipiamo il fatto che questa mesh non sarà renderizzata, ma servirà come driver di uno stencil buffer.

Vediamo nel paragrafo successivo come sono stati creati gli array dei vertici e dei triangoli.

3.2.5.3 Algoritmo di triangolazione

Inizialmente, l'array xyz_map viene convertito in un array di Vector3, di dimensioni pari ad un terzo dell'array originale; ricordiamo infatti che xyz_map, estratto tramite la libreria Depth_Utility, risulta come un array di float.

Come quello originale, questo nuovo array presenta i punti 3D della mesh, ma descritti attraverso il costrutto "Vector3" di Unity.

Successivamente, viene creato un array per contenere gli indici dei vertici che formano i triangoli della mesh. Non conoscendo a priori la triangolazione e dovendo inizializzare l'array ad una dimensione iniziale, questo array viene inizialmente configurato per contenere sei volte il numero dei vertici totali della xyz_map, poiché, nella peggiore delle ipotesi, ogni vertice può appartenere a sei triangoli diversi e quindi comparire sei volte all'interno dell'array dei triangoli.

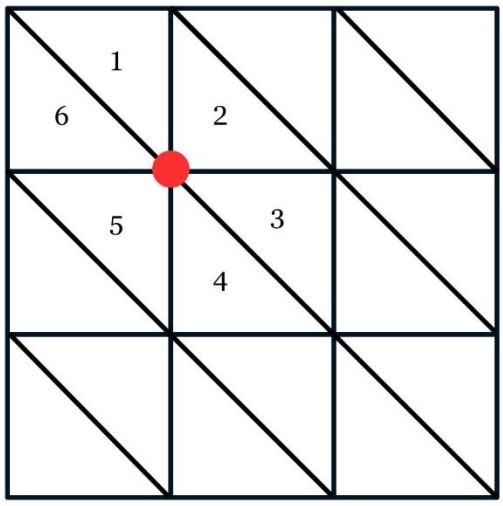

Figura 38: rappresentazione bidimensionale della triangolazione di una griglia di punti ordinata. Ogni vertice può appartenere ad un massimo di sei triangoli differenti

Successivamente, viene condotta un'iterazione su ogni quadrato della griglia. Per ciascun quadrato, sono stati identificati i due triangoli che lo compongono ("Upper" e "Lower") come mostrato in Figura 39.

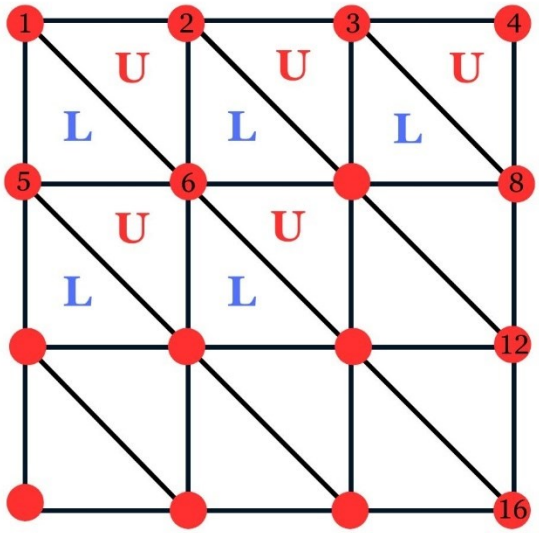

Figura 39:Suddivisione in triangoli Upper e Lower per ogni quadrato della griglia

Prima di inserire ogni tripletta all'interno del vettore di triangoli e quindi generare il poligono associato, l'algoritmo implementato passa per alcuni steps di pulizia:

1. Per ciascun triangolo, si verifica che tutti i vertici abbiano componente Z diversa da zero (ricordiamo infatti che i punti a valori non finiti erano stati posti a Z = 0 in fase di pre-triangolazione);
2. Viene imposto un volume di clipping, sottoinsieme proprio del volume descritto dalla nuvola di punti di xyz_map. Questo volume serve a scartare la triangolazione dei vertici non appartenenti a questo dominio. Questa soluzione è stata adottata per questioni di ottimizzazione e pulizia del risultato finale.
3. La triangolazione viene assoggettata ad una verifica sulla forma del triangolo stesso: I triangoli che presentano un'estensione lungo l'asse Z, maggiore di 2cm, non vengono generati (Figura 40).
   Questa tipologia di filtro permette di scartare la triangolazione di punti la cui profondità in Z è affetta da errori: è improbabile, infatti, che due punti adiacenti presentino delle discontinuità molto forti lungo l'asse Z, soprattutto se si pensa al profilo di una mano e a quanto densamente campionata essa risulta all'interno della xyz_map.

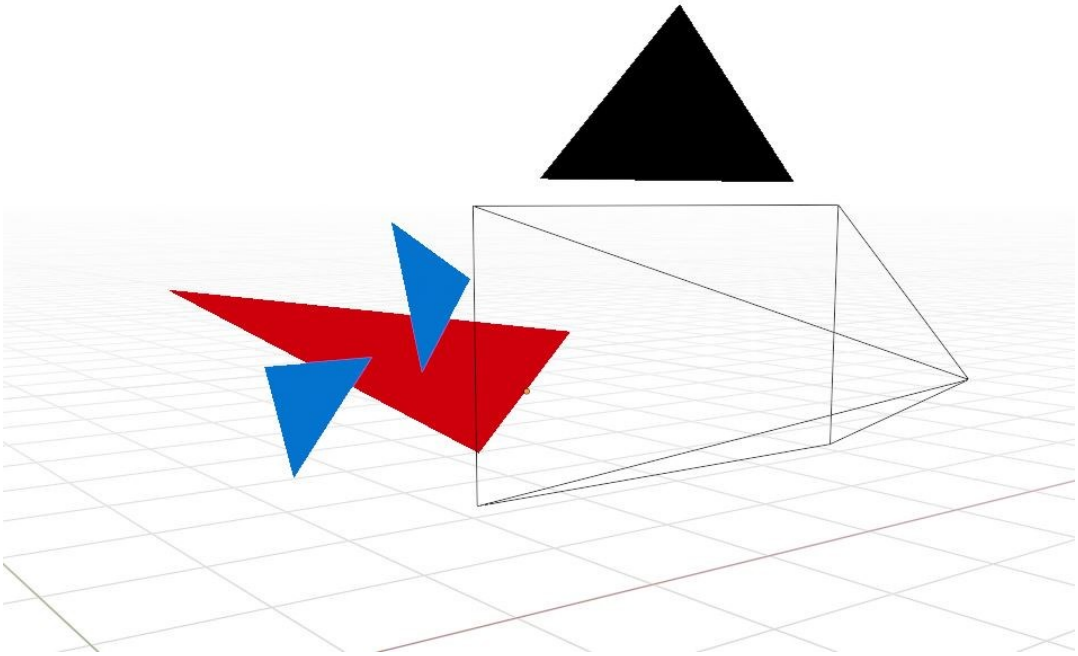

Figura 40: Il triangolo rosso viene scartato in fase d i generazione della mesh, in quanto troppo esteso lungo la direzione di vista della camera

È interessante analizzare maggiormente nel dettaglio l'implementazione di questo specifico filtro.

La triangolazione avviene solo se:

$$\frac{Max(Abs(Z_1 - Z_2), Abs(Z_2 - Z_3), Abs(Z_1 - Z_3))}{Z_{baricentro}} < 0.02m$$

È possibile osservare che l'estensione massima del triangolo lungo l'asse Z è normalizzato rispetto alla sua distanza dalla camera, al fine di rendere indipendente il filtro da quest'ultima variabile.

4. L'ultimo filtro riguarda l'implementazione di un " far clipping plane adattivo" (FCP), con normale lungo l'asse Z.

Questo filtro permette di evitare la triangolazione di tutti i punti non appartenenti alla mano, attraverso la gestione della posa real time del suddetto piano, in funzione della posizione del vertice della mesh "skin" più vicino alla camera.

La posizione del FCP viene calcolata frame per frame, calcolando la vicinanza dei vertici della mesh "skin" alla camera; l'iterazione sui vertici della mesh avviene con passi di 600 unità (un passo di 1 unità avrebbe comportato un costo computazionale troppo oneroso a fronte di un'approssimazione di poco migliore).

Vale la pena approfondire maggiormente l'implementazione dell'algoritmo che calcola la posa del FCP, in quanto all'interno dell'ambiente di Unity, le trasformazioni dei vertici e della camera sono descritti in due sistemi di riferimento differenti:
Inizialmente si descrive la camera nel SDR del gameObject "Skin", così da poter calcolare la distanza dei vertici da quest'ultima.
Una volta identificato il vertice che minimizza la distanza, si descrive la sua posizione nel sistema di riferimento della camera; in quanto la posa delle strutture anatomiche e della mesh della mano avviene proprio in questo sistema di riferimento.

3.2.6 Modifiche alla pipeline di render per la visualizzazione delle mani

3.2.6.1 Stencil buffer: definizione e confronto con lo Z-buffer.

Unity offre un'ampia varietà di tecniche di rendering per creare effetti visivi complessi. Tra questi, lo stencil buffer e il Z-buffer rivestono un ruolo significativo. Questo capitolo si propone di esplorare questi due strumenti, delineando la loro natura, il modo in cui vengono utilizzati e le differenze chiave tra di loro.

1. Lo **Stencil Buffer** è una componente del pipeline di rendering grafico che permette di controllare il processo di rendering a livello di pixel. Funziona in modo simile a una maschera, limitando l'area di rendering in base a specifiche condizioni programmabili. Per ciascun pixel, lo stencil buffer memorizza un

valore intero, il quale può essere modificato, confrontato e utilizzato per controllare se un pixel deve essere disegnato o meno.

Unity fornisce un'interfaccia per manipolare lo stencil buffer attraverso il linguaggio di programmazione shader. Gli sviluppatori possono specificare come modificare il valore dello stencil per ciascun pixel (incremento, decremento, sostituzione con un valore fisso, ecc.) e come confrontarlo con un valore di riferimento per decidere se renderizzare o meno il pixel.

2. lo **Z-buffer**, anche noto come depth buffer, è un'altra componente del pipeline di rendering che viene utilizzata per determinare la visibilità di un oggetto nella scena 3D. Per ogni pixel dell'immagine finale, lo Z-buffer memorizza una profondità, che rappresenta la distanza tra la camera e l'oggetto più vicino lungo la linea di vista. Quando più oggetti si sovrappongono nello stesso pixel, il valore di profondità di ciascuno viene confrontato con il valore corrente nello Z-buffer: se un oggetto è più vicino alla camera del corrente "vincitore", il suo valore di profondità viene scritto nel buffer e il suo colore viene scritto nell'immagine finale.

   Anche lo Z-buffer può essere manipolato tramite gli shader in Unity. Gli sviluppatori possono controllare se e come viene aggiornato il valore di profondità per ciascun pixel e se un pixel deve essere disegnato in base al confronto con il valore corrente dello Z-buffer.

Lo stencil buffer e il Z-buffer differiscono sia per quanto riguarda la programmazione, sia per quanto riguarda i risultati ottenuti.

Dal punto di vista della programmazione, lo stencil buffer offre un controllo più flessibile sul processo di rendering. Mentre lo Z-buffer permette solo di decidere se un pixel deve essere disegnato in base a un singolo criterio (la profondità), lo stencil buffer permette di specificare una varietà di operazioni e confronti. Inoltre, lo stencil buffer può essere utilizzato per creare effetti complessi che richiedono più passaggi di rendering, memorizzando informazioni tra un passaggio e l'altro.

In termini di risultati ottenuti, lo stencil buffer e lo Z-buffer sono utilizzati per raggiungere obiettivi diversi.

Lo Z-buffer è essenziale per ottenere una corretta occlusione degli oggetti 3D, assicurando che gli oggetti distanti siano nascosti da quelli vicini. Al contrario, lo stencil buffer è in grado di creare effetti che vanno oltre la mera occlusione, come il rendering selettivo di parti della scena, la creazione di effetti di portale o l'applicazione di post-processing ad aree specifiche dell'immagine.

3.2.6.2 Settaggio dello stencil buffer in Unity

Per la mesh della mano è stato utilizzato un materiale denominato 'mask'. Questo materiale fa uso di uno shader Unlit personalizzato, in cui i parametri 'renderType' e 'Queue' sono impostati come 'Opaque', mentre lo 'ZWrite' è disattivato. Questo permette alla mesh della mano di agire come una maschera, senza essere effettivamente renderizzata nell'immagine finale.

Per le altre mesh degli oggetti, lo shader è stato generato dinamicamente utilizzando lo shader graph di URP. Questo permette di avere un controllo completo sull'aspetto di ogni oggetto, senza la necessità di scrivere manualmente il codice dello shader.

Tra questi oggetti, 'Liver' e 'Skin' hanno un 'renderType' e una 'Queue' impostati come 'Transparent', in quanto devono permettere la gestione della trasparenza per poter visualizzare gli oggetti sottostanti.

Per sfruttare lo stencil buffer nel rendering selettivo, sono stati creati due livelli diversi, 'mask' e 'seeThrough'. Al livello 'mask' è associato l'oggetto mano, mentre a 'seeThrough' sono associati tutti gli altri oggetti. Questo permette di gestire separatamente il rendering della mano e degli altri oggetti.

Nell'asset del render di URP, è stato escluso il rendering di oggetti nei livelli 'mask' e 'seeThrough'. Sono state poi aggiunte due 'render features' per permettere il rendering personalizzato di questi oggetti.

La prima 'render feature', chiamata 'mask', filtra gli oggetti con 'Queue: Opaque' e 'layer: mask', quindi include solo la mano. Al momento dell'evento 'beforeRenderingTransparent', lo stencil buffer viene modificato per ogni pixel della mano, sostituendo il valore corrente con 1.

La seconda 'render feature', 'seeThrough', è responsabile del rendering di tutti gli altri oggetti. Questa feature filtra gli oggetti nel layer 'seeThrough' e li renderizza rispettando le regole dello stencil buffer. Questo significa che un pixel di un oggetto 'seeThrough' viene renderizzato solo se il corrispondente valore nello stencil buffer è 1, ovvero se si trova "sotto" la mano.

In questo modo, l'utilizzo dello stencil buffer permette di realizzare un effetto di "raggi X", dove la mano può essere utilizzata come una "lente" per visualizzare gli oggetti normalmente occlusi. La flessibilità dello stencil buffer consente di ottenere questo risultato con un controllo preciso e senza la necessità di manipolare manualmente la profondità o la trasparenza degli oggetti.

4 RISULTATI

In questo capitolo vengono esposti e commentati risultati ottenuti a seguito dell'implementazione delle soluzioni analizzante all'interno della sezione Materiali e Metodi.

4.1 CREAZIONE DI SHADERS REALISTICI

L'implementazione della URP e degli shader tri-planari ha permesso di ottenere dei materiali che si comportano fisicamente in maniera accurata con l'illuminazione, aggiungendo anche dettagli ed imperfezioni che contribuiscono alla resa realistica finale.

Nella figura mostrata di seguito viene mostrato il confronto tra i materiali, prima e dopo l'applicazione delle modifiche precedentemente approfondite.

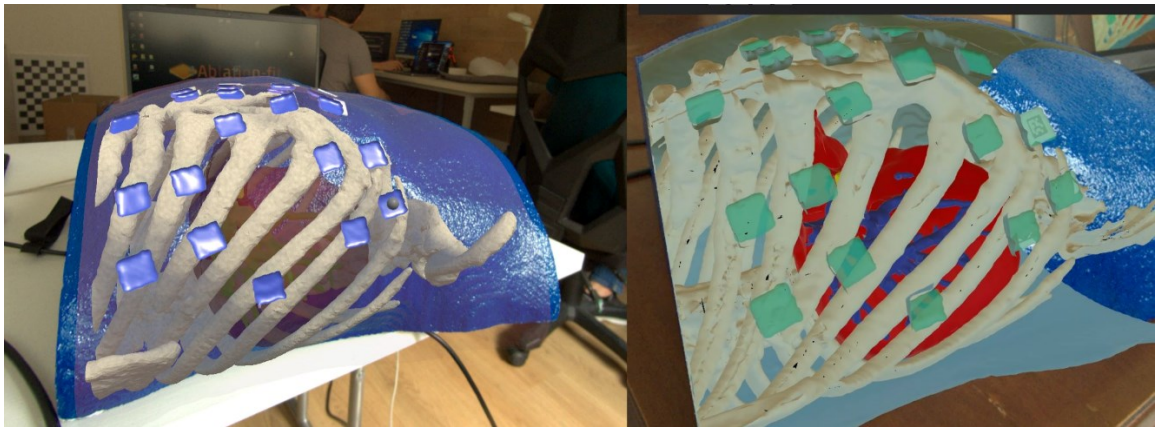

Figura 41: Confronto tra la resa visiva a seguito delle modifiche apportate (immagine di sinistra) e la resa visiva pre-esistente.

4.2 VISUALIZZAZIONE DELLE MANI DELL'UTENTE

L'analisi dei risultati ottenuti dall'implementazione delle tecniche di visualizzazione della mano può avvenire in due modi:

1. un'analisi qualitativa basata sui riscontri ottenuti da utenti che testano il sistema.
2. Un'analisi quantitativa sulla bontà della libreria deputata alla creazione della mappa di profondità e dei sistemi di filtraggio a valle delle chiamate alla libreria.

All'interno di questo documento vengono presentati i dati quantitativi ottenuti tramite testing della libreria, al variare dei suoi parametri in ingresso.

4.2.1 Metriche adottate: definizioni

Per poter quantificare i risultati, sono state identificate le seguenti tre metriche:

1. Precision.
2. Recall.
3. F-measure.

Queste metriche vengono utilizzate in una vasta gamma di applicazioni, tra cui il riconoscimento di pattern, il recupero di informazioni, la rilevazione di oggetti e, in generale, in qualsiasi task di classificazione in cui è importante valutare sia la capacità del modello di "colpire" gli elementi di interesse (precision), sia la sua capacità di non "lasciarsi sfuggire" elementi rilevanti (recall).

Utilizzando la seguente nomenclatura:

1. TP = true positives
2. FP = false positives
3. FN = false negatives
4. TN = true negatives

le precedenti metriche sono definite come segue[11]:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F - measure = 2\,\frac{Precision * Recall}{Precision + Recall}$$

Il grafico in Figura 42 aiuta a capire la precedente suddivisione:

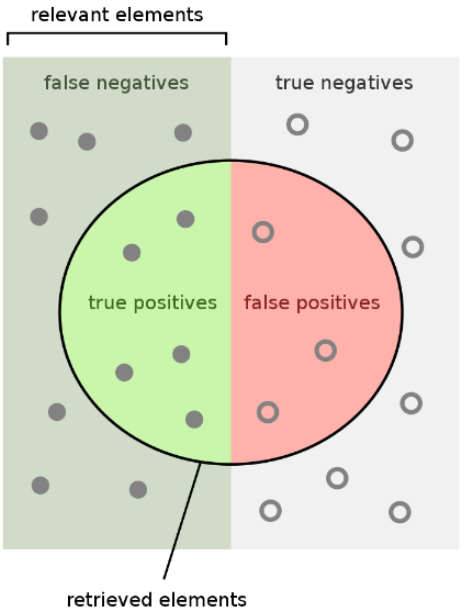

Figura 42: problema di classificazione generico[12].

[11] Fonte: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall
[12] Fonte: https://commons.wikimedia.org/wiki/User:Walber

I valori di Precision e Recall vengono solitamente plottati in un grafico che prende il nome di Recall-Precision Graph, che serve ad organizzare i dati in maniera tale da essere maggiormente comprensibili (esempio riportato in Figura 43)

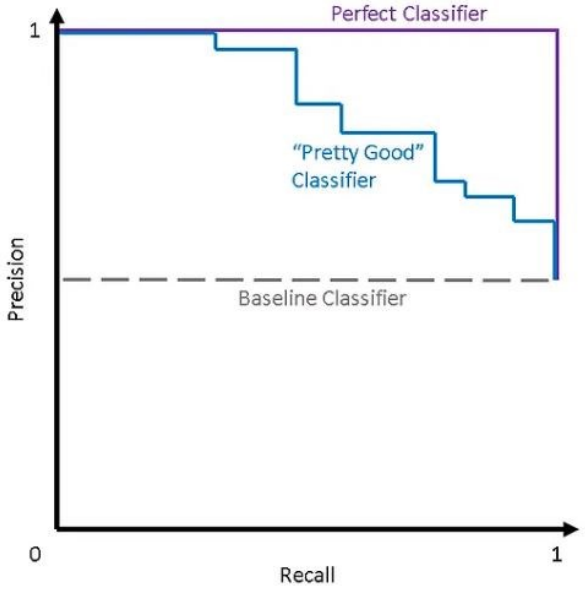

Figura 43: Recall-Precision Graph [13]

È interessante notare la forma del classificatore perfetto ideale (in viola in figura)

4.2.2 Metriche di valutazione declinate nel modello Depth_Utility.dll

Decliniamo le definizioni all'interno del nostro modello:

1. TP = vertici della mesh "mano", triangolati dall'algoritmo, realmente appartenenti alla sagoma della mano.
2. FP = vertici della mesh "mano", non appartenenti alla sagoma della mano.
3. TN = punti non appartenenti alla sagoma della mano, non triangolati dall'algoritmo.
4. FN = punti appartenenti alla sagoma della mano, non triangolati dall'algoritmo.

[13] Fonte della figura consultabile al seguente link: https://medium.com/@douglaspsteen/precision-recall-curves-d32e5b290248

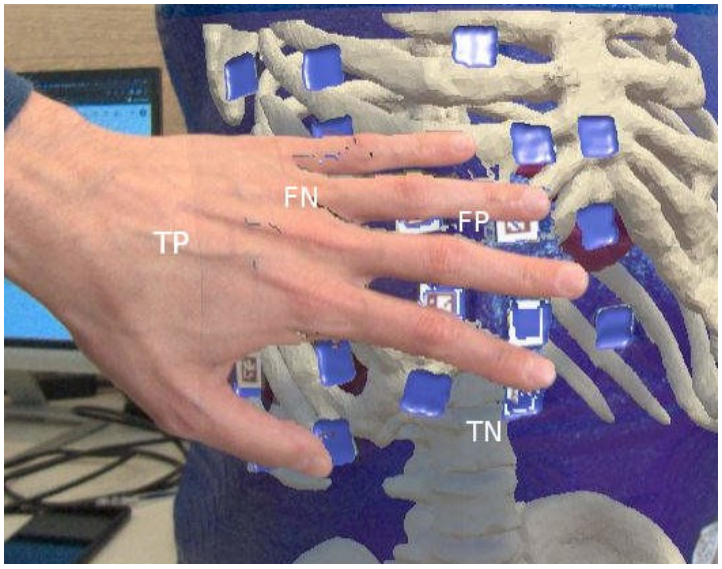

Figura 44: Visualizzazione in realtà aumentata con triangolazione della mano attiva

In Figura 44 si può distinguere la classificazione definita in precedenza.

I true positives risultano essere i pixel della mano correttamente visualizzati.

I false negatives sono i pixel presenti all'interno del contorno della mano, che però presentano il colore degli oggetti sintetici presenti al di sotto.

I false positives sono i pixel al di fuori del contorno della mano, in cui non vengono renderizzati gli oggetti sintetici.

I true negatives sono i pixel al di fuori del contorno della mano, in cui vengono correttamente renderizzati gli oggetti sintetici.

I test di Precision e Recall prevedono l'analisi di queste due grandezze, al variare di un parametro definito come *soglia*.

Nel nostro caso è stato scelto come parametro di soglia la profondità Z rispetto alla camera, come mostrato in Figura 45.

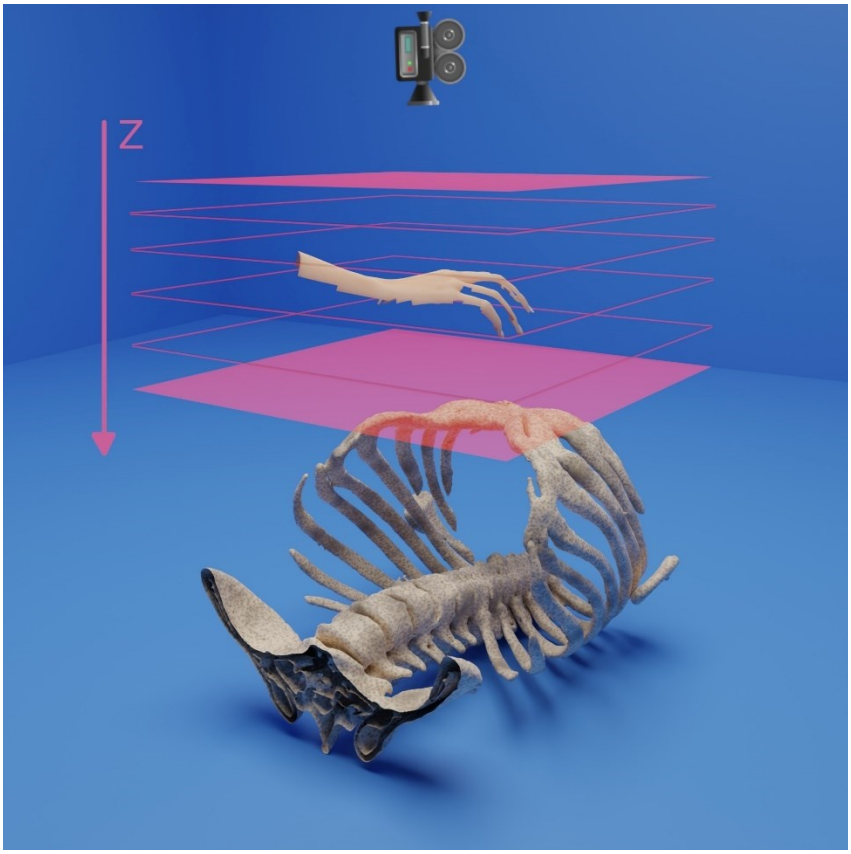

Figura 45: Rappresentazione grafica dei test di Precision and Recall applicati al sistema sviluppato

4.2.3 Condizioni sperimentali

I test sono stati effettuati su un campione di cento frames acquisiti dalla stereocamera, così da poter mediare i dati nel tempo ed ottenere un risultato indipendente dalla bontà della singola acquisizione.

L'acquisizione è avvenuta con le modalità presentate in figura:

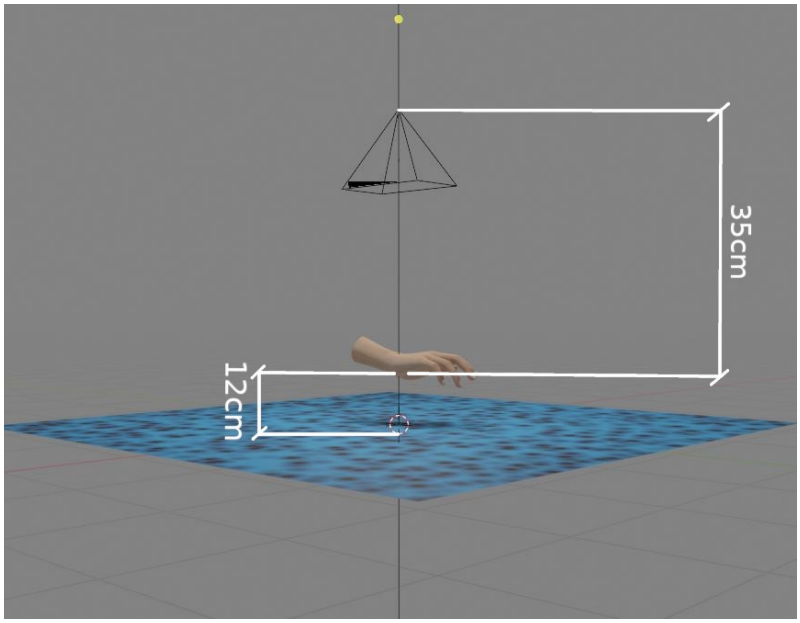

Figura 46: Condizioni sperimentali

La superficie posta sotto la mano serve ad introdurre, all'interno dell'analisi, le features relative ai true negatives. È quindi importante che presenti una texture ottimale per gli algoritmi di stereo matching e una superficie costante in Z all'interno del piano immagine.

In Figura 47 è presentato un singolo frame stereoscopico utilizzato per i test:

Figura 47: Frame di esempio utilizzato per i test di Precision and Recall

4.2.4 Test quantitativi

I grafici mostrati nei sottoparagrafi a seguire, mostrano l'andamento delle metriche Recall, Precision e F-measure al variare della soglia Z.

Ogni curva è stata ottenuta modificando un singolo parametro in ingresso al modello, lasciando gli altri parametri ai valori di default suggeriti da OpenCV.

Di seguito un elenco dei valori di default suggeriti da OpenCV:

Input algoritmi di stereo matching:

1. numDisparity = [int] 128
2. minDisparity = [int] 0
3. blockSize = [int] 21

input algoritmo di smoothing

1. lambda = [int] 12000
2. sigma = [float] 0.5
3. smoothMin = [int] -32000
4. smoothMax = [int] -30000

I diversi test possono essere classificati in due macrocategorie:

1. con smoothing disattivo (SD), finalizzato a comprendere la bontà dell'algoritmo di stereo matching
2. con smoothing attivo (SA), finalizzato a comprendere la bontà della classificazione dell'intero sistema implementato.

Le operazioni morfologiche di chiusura sono state scartate in fase di analisi in quanto il loro apporto alla classificazione non risultava rilevante.

4.2.4.1 Test 1: StereoBM_BinningFactor_SD

Il fattore di binning (BF) è semplicemente un fattore di scala della dimensione dell'immagine usata per il calcolo dello stereo matching.

Riportiamo di seguito alcuni esempi.

Le immagini stereoscopiche acquisite dal sistema Endosight vengono acquisite con una risoluzione pari a 1714 x 1229 pixel, quindi:

BF(1) => 1714 x 1229px

BF(2) => 857 x 615px

BF(3) => 571 x 410px

Ridimensionare le immagini in ingresso all'algoritmo comporta notevoli benefici sul costo computazionale e quindi sui tempi di calcolo.

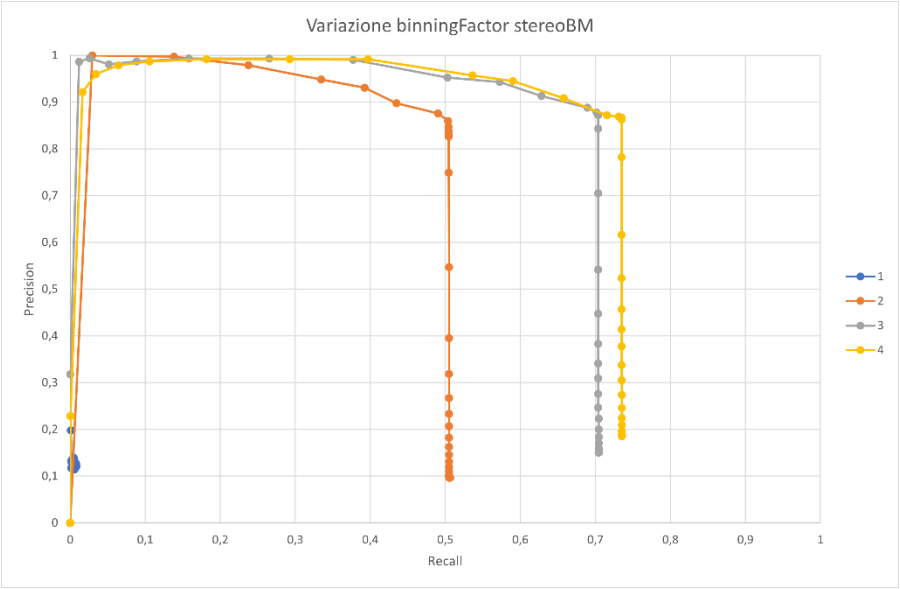

Grafico 1

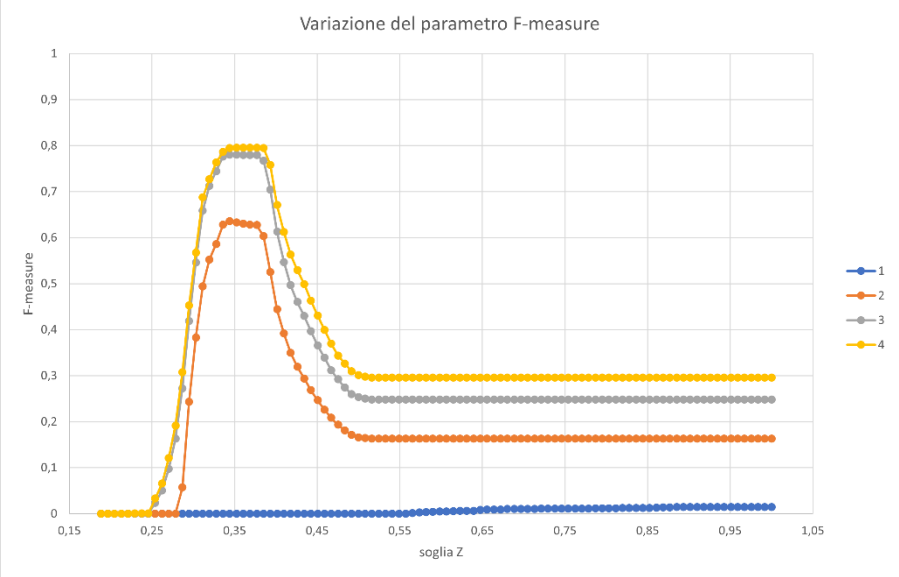

Grafico 2

I grafici mostrano un andamento simile a quello che in letteratura viene identificato come un classificatore ideale.

Possiamo osservare come un BF di 1 risulti in una classificazione completamente errata, mentre la bontà di classificazione aumenti all'aumentare del BF stesso, in maniera quasi controintuitiva: un'immagine di grandi dimensioni è maggiormente ricca di dettagli e quini in linea teorica dovrebbe generare dei risultati migliori.

Questa divergenza tra i risultati e ciò che ci si aspetterebbe può essere imputata a diversi fattori:

1. **Rumore e distorsioni**: Le immagini con risoluzione più alta tendono ad avere più rumore e distorsioni, che possono interferire con la corrispondenza dei punti tra le due immagini. Il binning riduce effettivamente il rumore e le distorsioni aggregando i pixel, contribuendo a migliorare l'accuratezza della mappa di disparità.

2. **Mancanza di texture**: Gli algoritmi di stereo matching si basano sulla presenza di texture nelle immagini per trovare corrispondenze. Se l'immagine in input è molto dettagliata ma contiene molte regioni uniformi, allora potrebbe essere difficile per l'algoritmo trovare corrispondenze affidabili.

3. **Potenza computazionale**: L'aumento della risoluzione delle immagini implica un notevole aumento del carico computazionale. Questo significa che l'algoritmo può non avere risorse sufficienti per processare accuratamente l'intera immagine.

4. **Ottimizzazione dell'algoritmo**: Alcuni algoritmi possono non essere ottimizzati per lavorare con immagini ad alta risoluzione. Potrebbero funzionare meglio con immagini di dimensioni moderate, dove c'è un buon equilibrio tra dettaglio e rumore.

Possiamo inoltre sottolineare come il parametro Precision mantenga dei valori alti per tutto il range di soglia, mentre la Recall mostri un massimo pari a 0.73 (relativo alla curva con BF = 4). Questo è indice del fatto che questo algoritmo non riconosce molti punti appartenenti alla mano.

4.2.4.2 Test 2: StereoBM_numDisparity_SD

I seguenti grafici sono stati ottenuti impostando un BF pari a 4 e variando il parametro numDisparity da 96 a 160 con passi di 16.

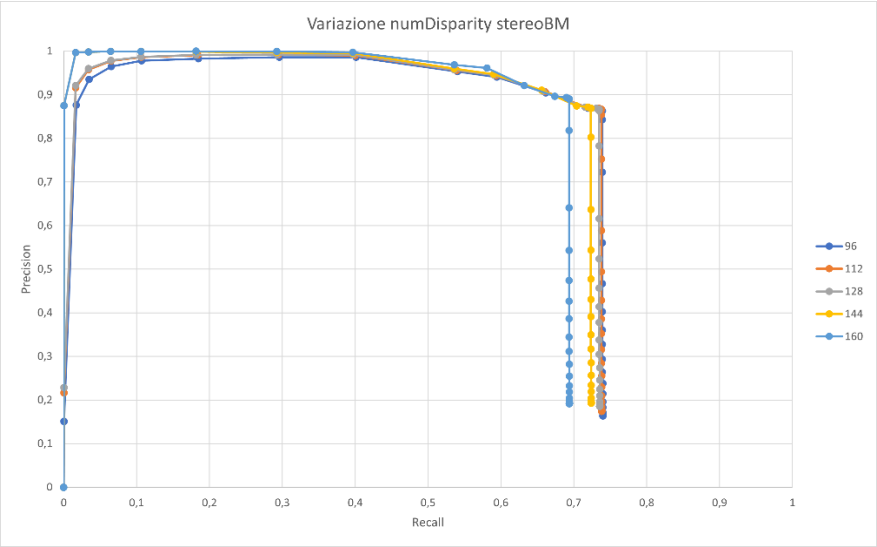

Grafico 3

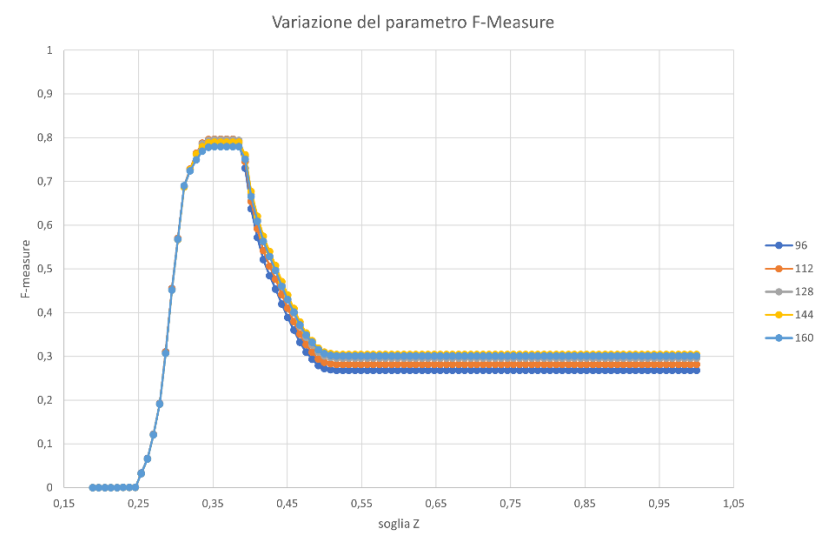

Grafico 4

Si può osservare come la variazione del parametro numDisparity generi dei risultati pressoché sovrapponibili con valori compresi tra 96 e 128.

Ricordiamo come questo parametro influenzi l'algoritmo di stereo matching:

1. **Impatto sulle performance**: Aumentare il valore di numDisparity comporta un aumento del costo computazionale dell'algoritmo, poiché implica più confronti per ciascun pixel. Di conseguenza, se numDisparity è alto, l'algoritmo richiederà più tempo per produrre la mappa di disparità.

2. **Impatto sulla qualità della mappa di disparità**: Il valore di numDisparity può influenzare la capacità dell'algoritmo di gestire oggetti a diverse distanze. Se il valore è troppo basso, l'algoritmo potrebbe non essere in grado di gestire correttamente gli oggetti che sono più lontani, poiché la disparità per questi oggetti potrebbe essere maggiore del valore impostato. Al contrario, se il valore è troppo alto, l'algoritmo potrebbe essere più soggetto a errori nelle regioni di bassa texture, poiché vi sono più possibilità di trovare corrispondenze errate.

4.2.4.3 Test 3: StereoBM_blockSize_SD

I seguenti grafici sono stati ottenuti impostando un BF pari a 4 e variando il parametro blockSize da 3 a 47 px.

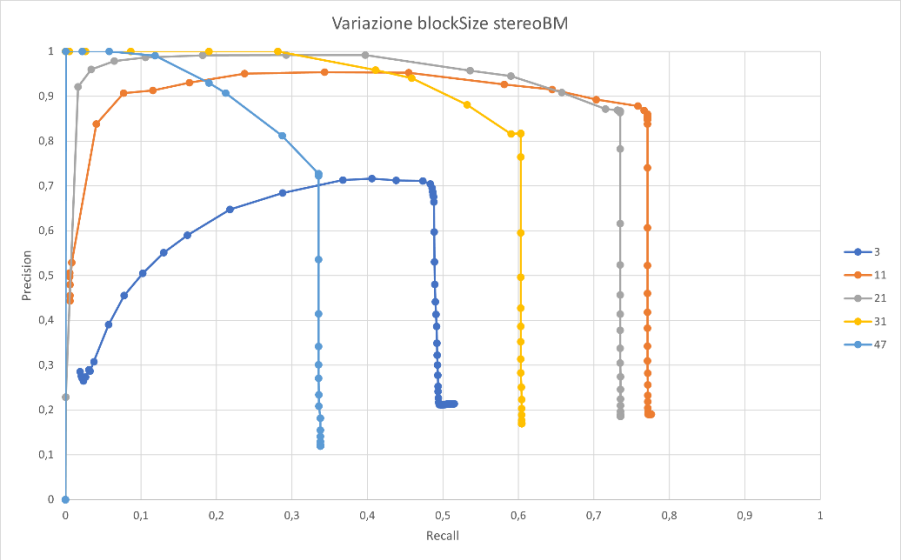

Grafico 5

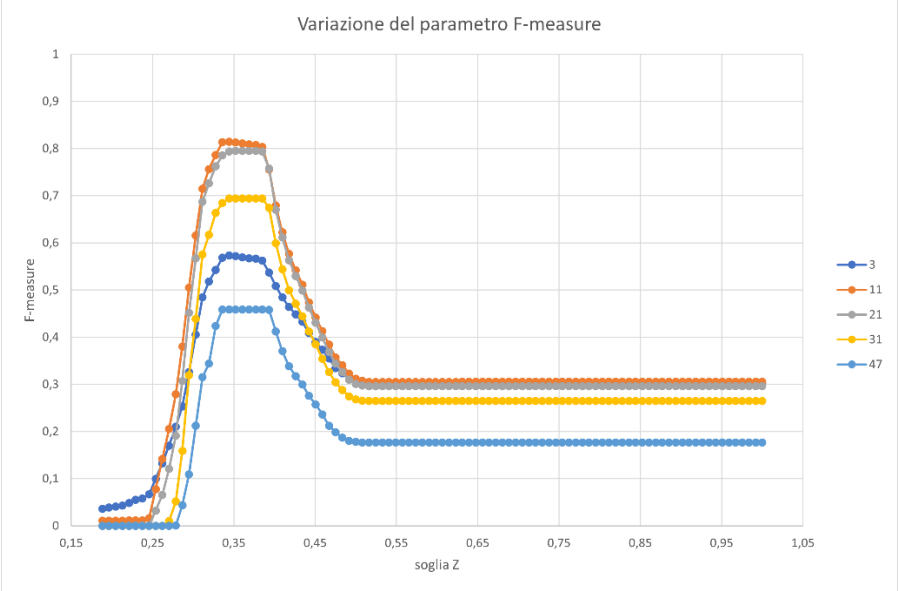

Grafico 6

Le variazioni del blockSize comportano una grande variazione sulla classificazione.

Possiamo infatti osservare come blockSize piccoli, ad esempio 3px, generino delle curve non ottimali, sia in termini di Precision che di Recall; lo stesso accade anche per blockSize troppo grandi (31, 47 px), anche se in questo caso la Precision presenta dei risultati migliori.

Il punto di ottimo viene raggiunto con una blockSize pari a circa 11 px.

Ricordiamo che il parametro blockSize inficia su:

1. **Qualità della mappa di disparità**: Il parametro blockSize gioca un ruolo cruciale nella gestione delle texture dell'immagine. Una dimensione del blocco più grande può ridurre l'impatto del rumore nell'immagine, poiché i pixel vengono mediati su un'area più grande, riducendo così la varianza. Tuttavia, l'uso di un blocco più grande può anche portare a una perdita di dettagli, specialmente nei contorni degli oggetti, dove la disparità cambia rapidamente. D'altro canto, una dimensione del blocco più piccola può preservare i dettagli dell'immagine, ma potrebbe essere più sensibile al rumore e potrebbe produrre mappe di disparità con una maggiore varianza.

2. **Performance**: L'uso di una dimensione del blocco più grande comporta un maggiore costo computazionale, poiché per ogni pixel, l'algoritmo deve confrontare un'area più ampia. Di conseguenza, un blockSize maggiore può portare a tempi di esecuzione più lunghi.

4.2.4.4 Test 4: StereoBM_BinningFactor_SA

In questa sezione analizziamo i risultati ottenuti imponendo i parametri di default mostrati precedentemente, variando il BF e applicando un algoritmo di smoothing alla mappa di disparità.

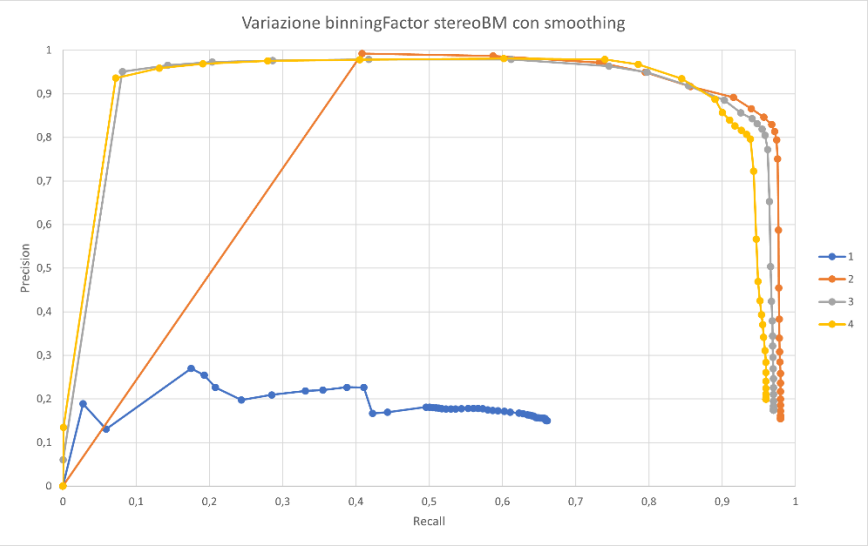

Grafico 7

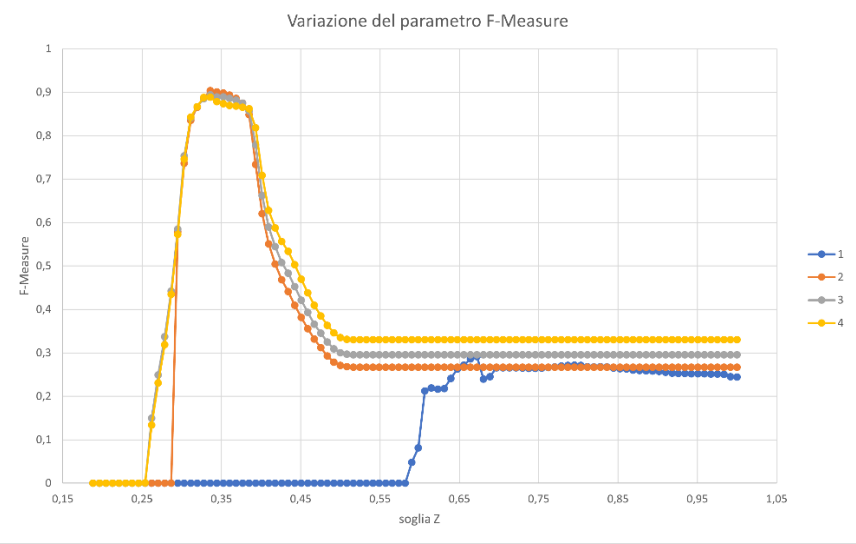

Grafico 8

Applicando lo smoothing si ottengono dei risultati migliori, soprattutto riguardo la Recall: osserviamo un valore massimo di 0.97, contro 0.74 senza smoothing.

Il valore F-Measure mostra dei valori massimi di 0.9, più alto di quello ottenuto senza smoothing (0.8).

Osserviamo inoltre una minore differenza relativa tra le curve al variare del BF.

È interessante anche notare che l'ottimo delle curve si ottiene con un BF di 2 e non di 4, come mostrato dalle curve con smoothing disattivo.

Una spiegazione a questo fenomeno potrebbe essere che lo smoothing interferisca in qualche modo con le informazioni utili che sono presenti nella mappa di disparità, quando si utilizza un fattore di binning più elevato. Ad esempio, se lo smoothing sta effettivamente "appiattendo" alcune delle caratteristiche importanti nella mappa di disparità, potrebbe effettivamente degradare la qualità dei risultati ottenuti con un fattore di binning più elevato.

4.2.4.5 Test 5: StereoBM_numDisparity_SA

Il test è stato effettuato applicando un BF pari a 4

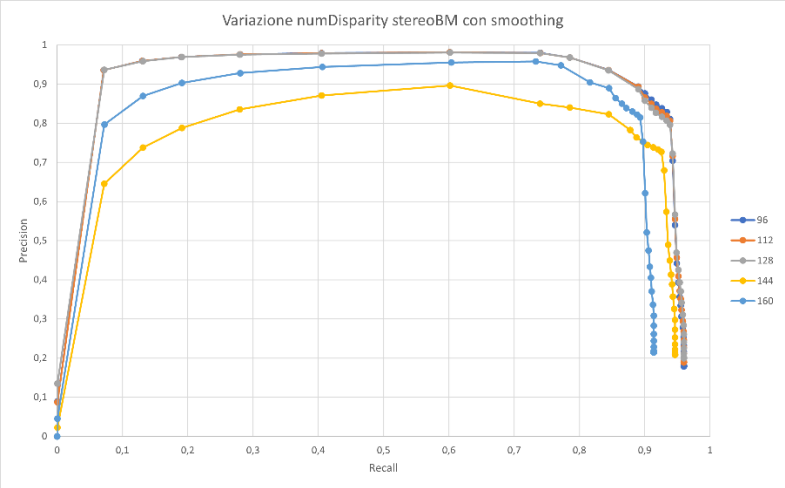

Grafico 9

Le considerazioni effettuate per il test SD valgono anche in questo caso.

Riportiamo un generale avvicinamento relativo tra le curve, in termini di somiglianza, a seguito dell'applicazione dello smoothing. numDisparity molto elevati continuano a fornire i risultati peggiori.

4.2.4.6 Test 6: StereoBM_blockSize_SA

I seguenti grafici sono stati ottenuti impostando un BF pari a 4 e variando il parametro blockSize da 3 a 47 px.

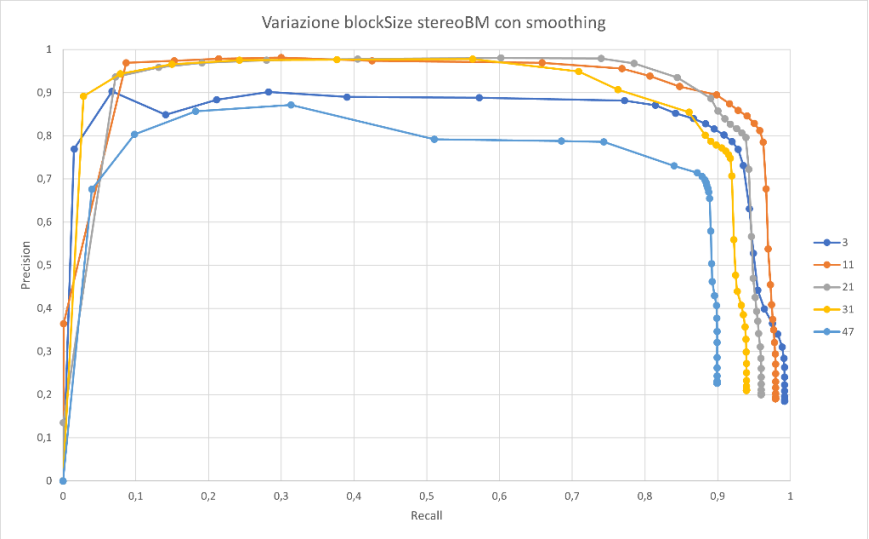

Grafico 10

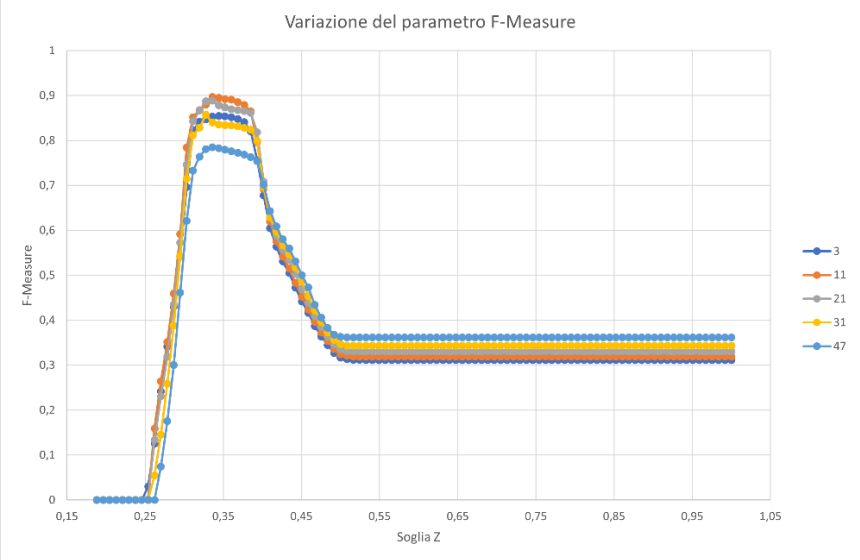

Grafico 11

Questo test mostra sostanzialmente un miglioramento generale delle curve, pur mantenendo il punto di ottimo con blockSize 11.

È importante sottolineare un miglioramento sostanziale della curva con blockSize pari a 3 rispetto alla versione SD, anche se rimane comunque sub-ottima e tendenzialmente inutile, in quanto richiede una complessità computazionale maggiore rispetto ad un valore di 11.

4.2.4.7 Test 7: StereoBM SD vs SA

In questo test mostriamo la differenza tra la classificazione con smoothing attivo e disattivo, ottenuto tramite stereoBM.

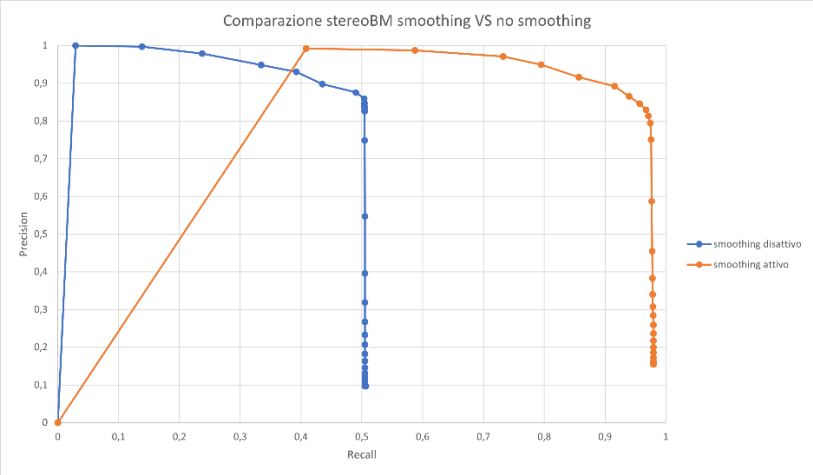

Grafico 12

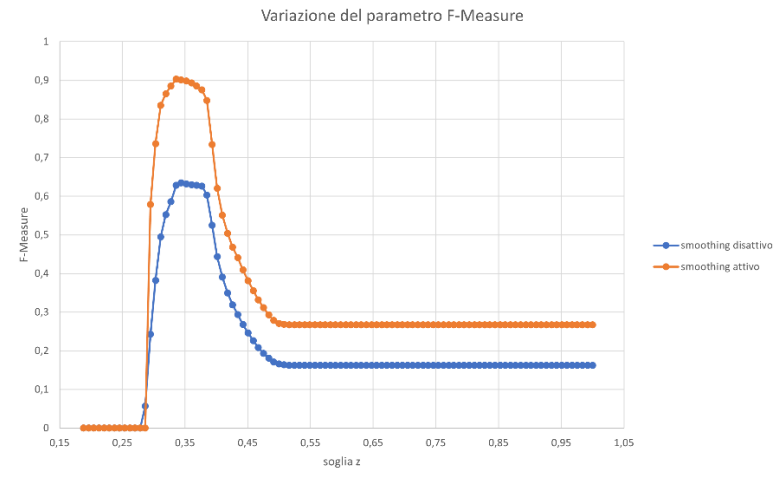

Grafico 13

I parametri utilizzati per questo test risultano i seguenti:

1. BF = 2
2. numDisparity = 96
3. blockSize = 11

Risulta ovvio come la classificazione subisca un notevole miglioramento a seguito dell'applicazione dello smoothing della disparity map, quasi raddoppiando il valore di Recall per ogni soglia.

Il grafico F-measure riconferma il notevole miglioramento del classificatore: il valore massimo passa da 0.63 a 0.9.

Bisogna però sottolineare che le condizioni di testing risultino leggermente a vantaggio della versione SA in quanto un BF di due risultava non ottimale per la versione SD.

4.2.4.8 Test 8: StereoSGBM_BinningFactor_SD

Da questo test in avanti si analizzerà la bontà del classificatore ottenuto tramite stereo SGBM, che ricordiamo essere un algoritmo computazionalmente più complesso e quindi più oneroso rispetto al semplice BM.

Per questo specifico test sono stati utilizzati gli stessi valori di binning utilizzati per i test dell'algoritmo stereo BM.

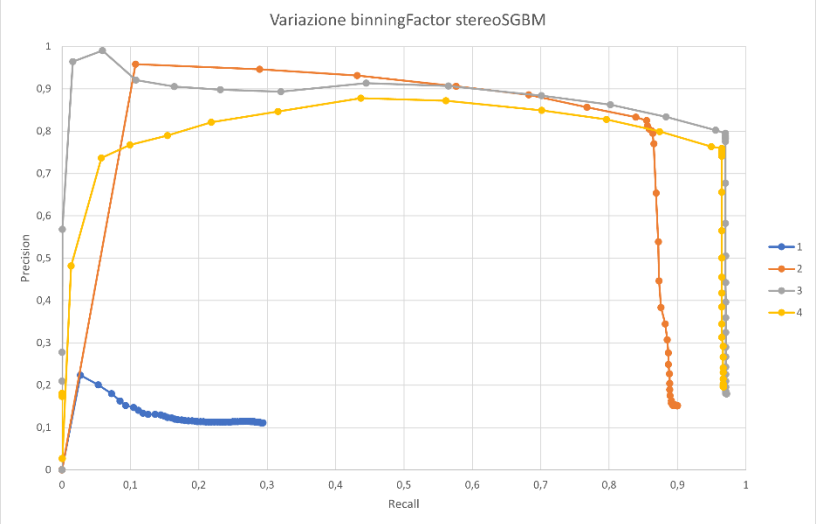

Grafico 14

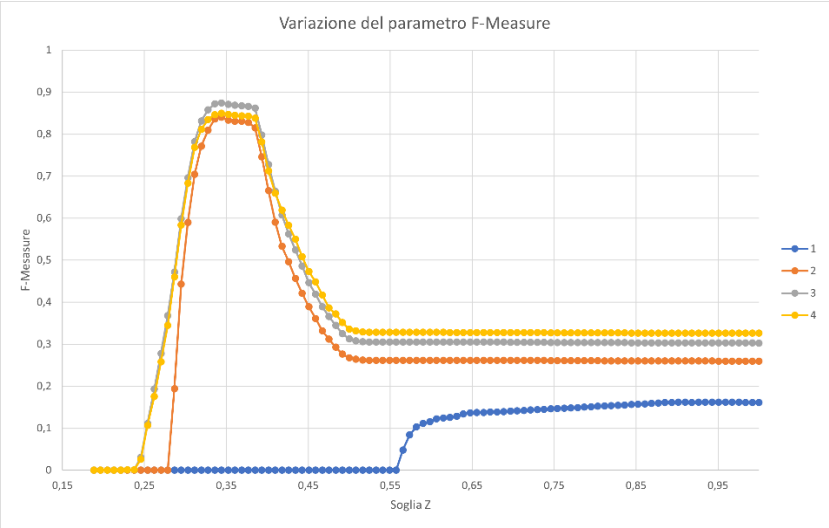

Grafico 15

Utilizzando stereo SGBM si può notare come la qualità generale del classificatore sia alta anche senza l'applicazione dello smoothing (F-Measure max pari a 0.88 con BF di 3).

Un BF di 1, anche in questo caso comporta dei risultati pessimi, probabilmente per le stesse ragioni presentate per il test con stereo BM.

Le curve presentano una sensibilità minore alle variazioni del BF, rispetto allo stereo BM, nonostante il punto di ottimo lo si ottenga con BF di 3.

In particolare, possiamo osservare una maggiore Recall a discapito di un lieve calo di precisione del classificatore.

4.2.4.9 Test 9: StereoSGBM_numDisparity_SD

I seguenti grafici sono stati ottenuti impostando un BF pari a 3 e variando il parametro numDisparity da 96 a 160 con passi di 16.

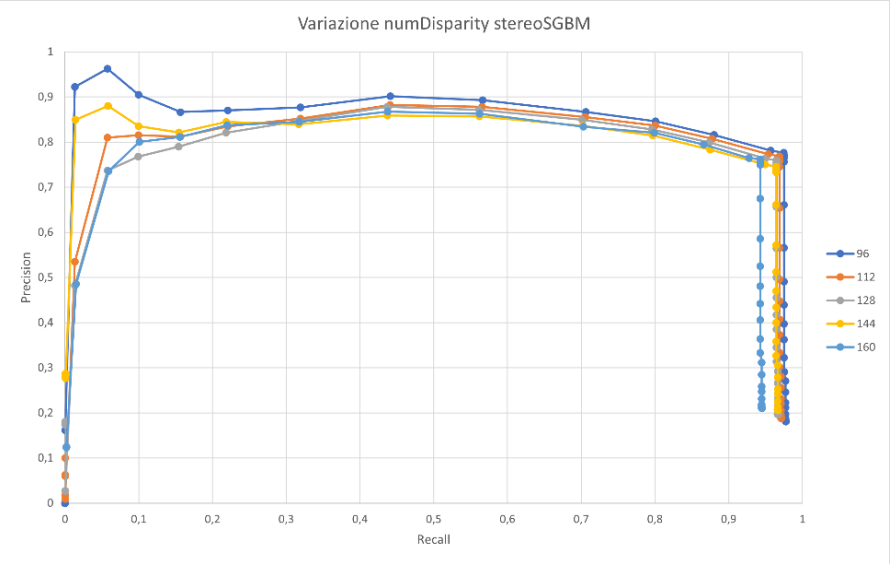

Grafico 16

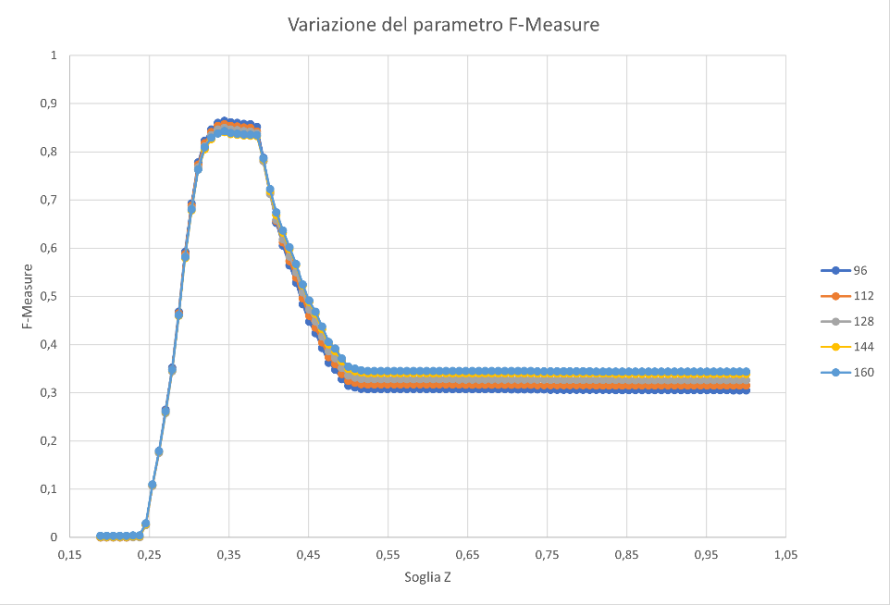

Grafico 17

Le curve risultano quasi del tutto sovrapponibili al variare del parametro numDisparity.

4.2.4.10 Test 10: StereoSGBM_blockSize_SD

I seguenti grafici sono stati ottenuti impostando un BF pari a 3 e variando il parametro blockSize da 3 a 47 px.

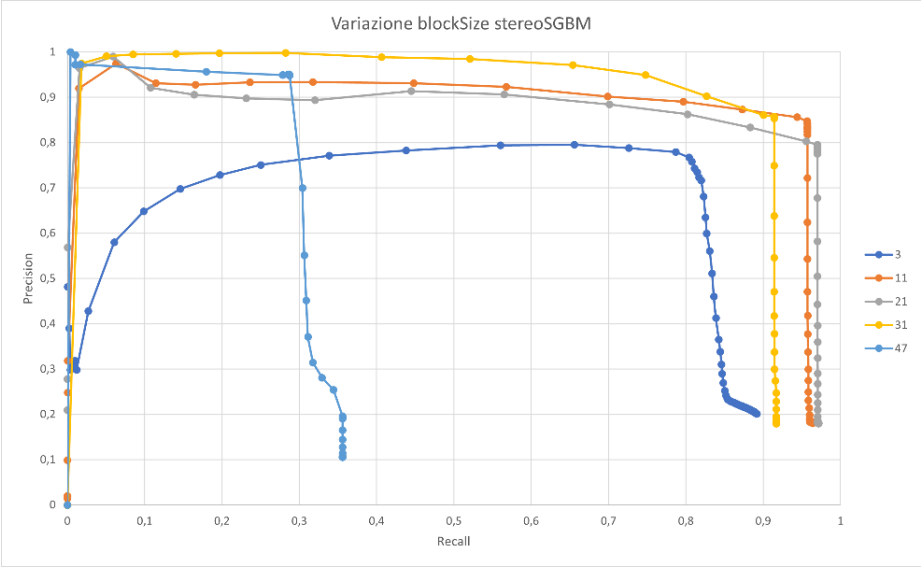

Grafico 18

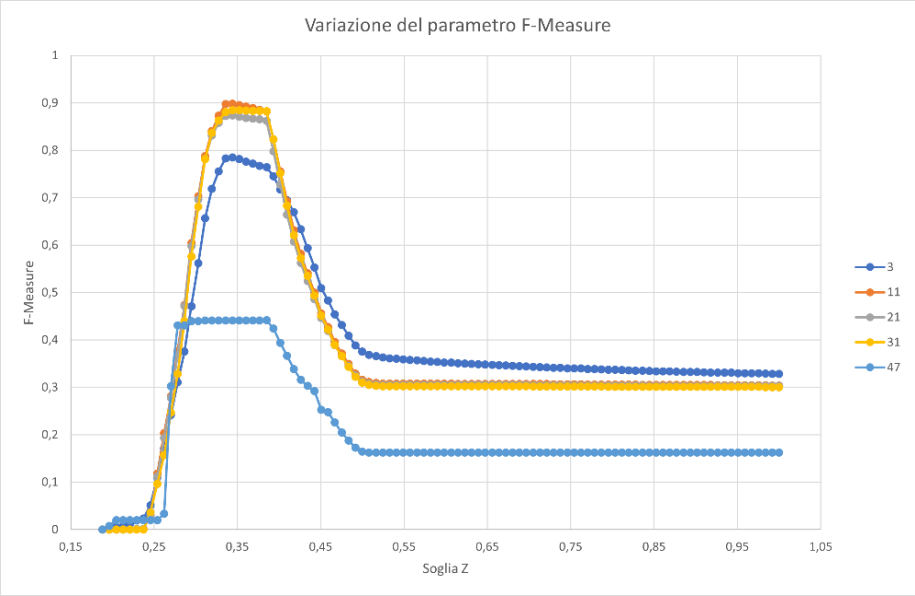

Grafico 19

Le stesse considerazioni riportate nel Test 3 valgono anche per questo test, con l'unica differenza che il classificatore mostra una Recall migliore per tutti i valori di soglia.

4.2.4.11 Test 11: StereoSGBM_BinningFactor_SA

In questa sezione analizziamo i risultati ottenuti imponendo i parametri di default mostrati precedentemente, variando il BF e applicando un algoritmo di smoothing alla mappa di disparità.

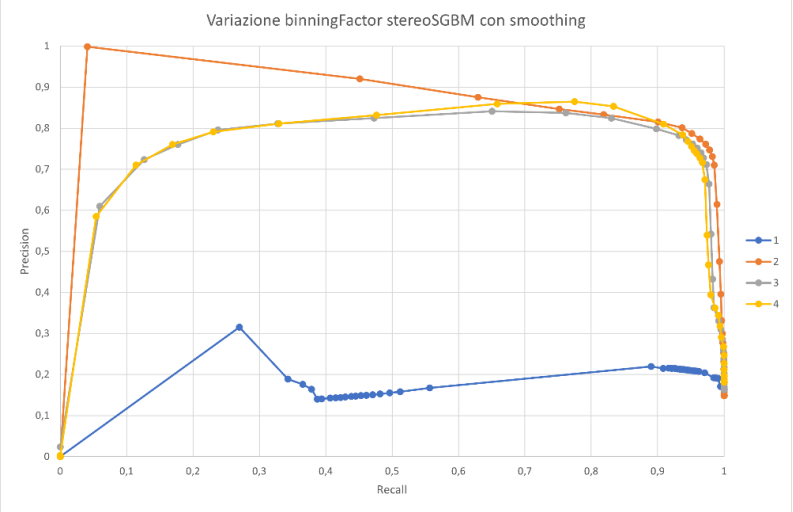

Grafico 20

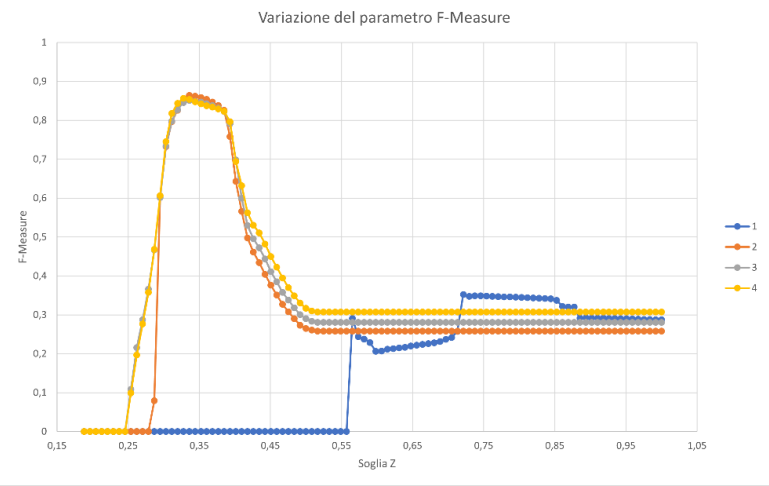

Grafico 21

Valgono le stesse considerazioni riportate per il Test 4, nonostante la bontà generale del classificatore risulti leggermente inferiore rispetto alla versione stereoBM

4.2.4.12 Test 12: StereoSGBM_numDisparity_SA

Il test è stato effettuato applicando un BF pari a 4

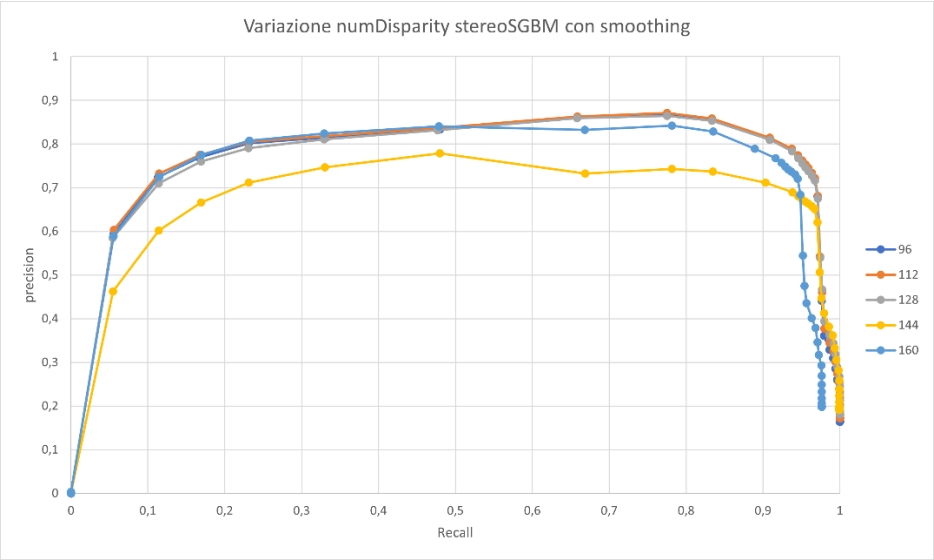

Grafico 22

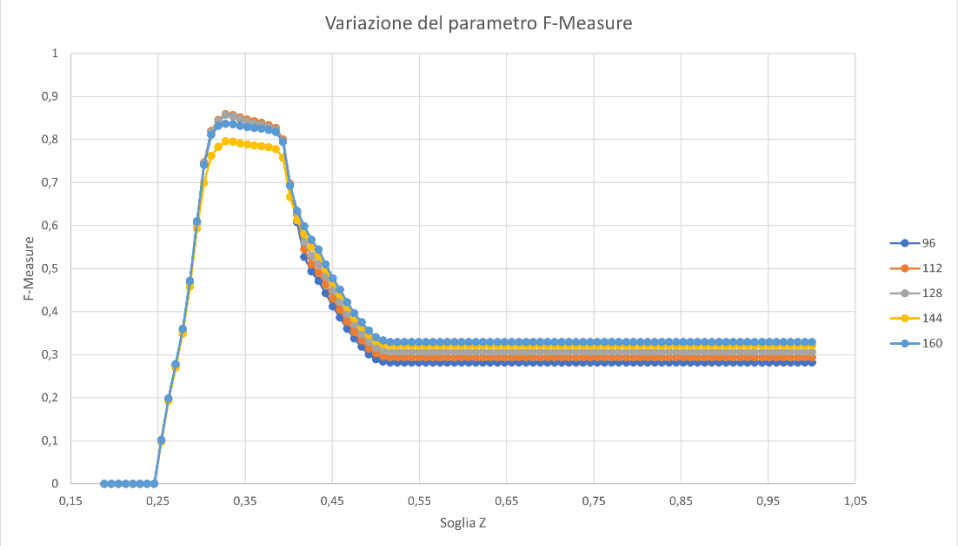

Grafico 23

Come nei precedenti test riguardanti la variazione di numDisparity, osserviamo un generale avvicinamento delle curve.

4.2.4.13 Test 13: StereoSGBM_blockSize_SA

I seguenti grafici sono stati ottenuti impostando un BF pari a 4 e variando il parametro blockSize da 3 a 47 px.

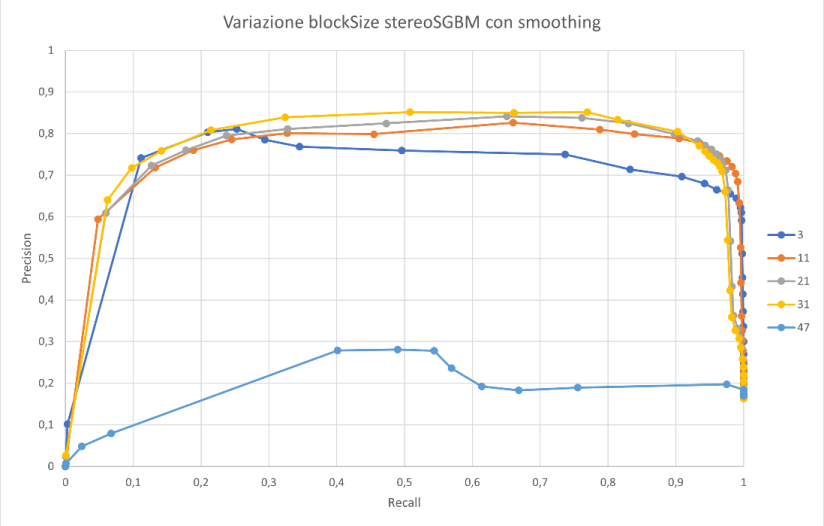

Grafico 24

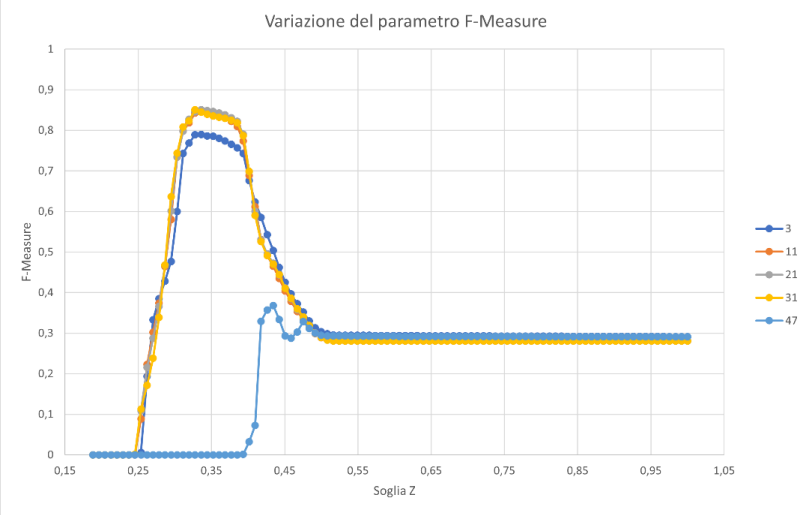

Grafico 25

4.2.4.14 Test 14: StereoSGBM SD vs SA

In questo test mostriamo la differenza tra la classificazione con smoothing attivo e disattivo, ottenuto tramite stereoSGBM.

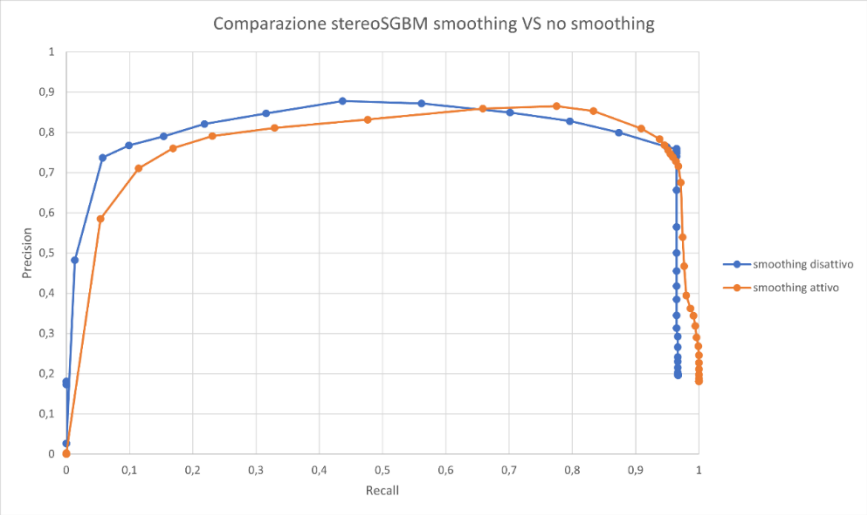

Grafico 26

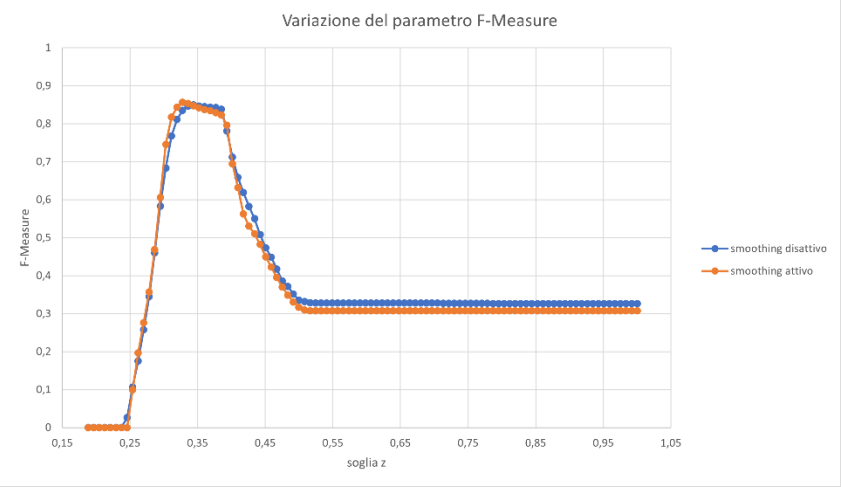

Grafico 27

A differenza dello stereo BM, i risultati ottenuti con smoothing disattivo sono quasi sovrapponibili con quelli ottenuti con smoothing attivo.

4.2.4.15 Analisi globale dei risultati

Alla luce di quanto evidenziato dai test riportati nei paragrafi precedenti, possiamo evidenziare una serie di comportamenti:

1. L'algoritmo di stereoBM genera dei buoni risultati soltanto se è attivato il filtro di smoothing, mentre lo stereoSGBM presenta buoni risultati anche senza filtro applicato.
2. Sia per stereoBM, che per stereoSGBM il filtro di smoothing elimina la variabilità dei risultati dipendente da diversi input in ingresso all'algoritmo (avvicinamento relativo delle curve); in altre parole i parametri di input degli algoritmi di stereo matching inficiano meno sul risultato a valle dello smoothing.
3. In generale, il parametro che sembra far variare maggiormente l'output è il binning factor: negli esempi riportati un binning factor di quattro risulta generare dei risultati migliori in termini di bontà del classificatore.

La scelta ottimale, avvalorata dai test sopra riportati risulta quindi essere:

1. stereoBM
2. smoothing attivo
3. BF = 4
4. blockSize = 11
5. numDisparity = 96

Essendo i risultati a valle dello smoothing molto simili tra loro, conviene optare per un'ottimizzazione riguardante il tempo di calcolo.

L'algoritmo stereoBM presenta una complessità dell'ordine di $O(dim)$, dove dim è la dimensione in pixel dell'immagine stereoscopica in ingresso.

StereoSGBM, d'altro canto, presenta una complessità pari a $O(dim^2)$, quindi, a parità di risultato, stereoBM risulta essere la scelta migliore.

Allo stesso modo, blockSize e numDisparity sono stati scelti in funzione dell'ottimizzazione dei tempi di calcolo.

L'introduzione degli algoritmi per il calcolo della mesh della mano comporta una complessità computazionale notevole. Il progetto è stato sviluppato su un calcolatore prestazionalmente inferiore rispetto alla macchina su cui è installata la versione commercializzata di Endosight. Attivando la triangolazione della mesh della mano, con i

settaggi ottimali indicati in precedenza, si assiste ad un notevole calo del frame rate.: l'applicazione perde le caratteristiche di real time, presentando un frame rate di circa 4FPS.

5 CONCLUSIONI

I risultati ottenuti rispettano ampiamente gli obiettivi ipotizzati inizialmente, sia in termini di qualità della resa visiva, sia in termini di miglioramento della sensazione di profondità e dell'operabilità fornita dal sistema Endosight.

La qualità grafica è importante per due motivazioni:

1. Una qualità grafica migliore è un fattore che può migliorare la user experience, nonché la percezione che ha l'utente del prodotto stesso; non va dimenticato, infatti, che Endosight è uno strumento commercializzato.
2. Il realismo dei materiali si ottiene specialmente tramite l'implementazione di un certo grado di variabilità e imperfezione delle superfici. Queste caratteristiche, rispetto ad una texture uniforme, aiutano il sistema di visione umano nel riconoscimento della forma e della profondità dell'oggetto.

La possibilità da parte dell'utente, di visualizzare la propria mano, durante le procedure di intervento, rappresenta un forte miglioramento dell'operabilità, traducendosi potenzialmente in interventi chirurgici maggiormente agevolati e quindi più sicuri.

5.1 CRITICITÀ DELLE LOGICHE IMPLEMENTATE

Al termine del progetto di tesi, il sistema risulta funzionante, ma ben lontano dal poter essere implementato all'interno della pipeline del software commercializzato:

Il classificatore della mano, nonostante la bontà dei grafici di Recall and Precision, risulta non pronto per la commercializzazione, poiché fortemente dipendente dalle condizioni esterne e affetto da imprecisioni dovute a diversi fattori fisiologici dell'utilizzo, come le occlusioni parziali e la sfocatura da moto.

Nella figura mostrata di seguito sono presenti due frame acquisiti durante la stessa sessione di utilizzo del sistema.

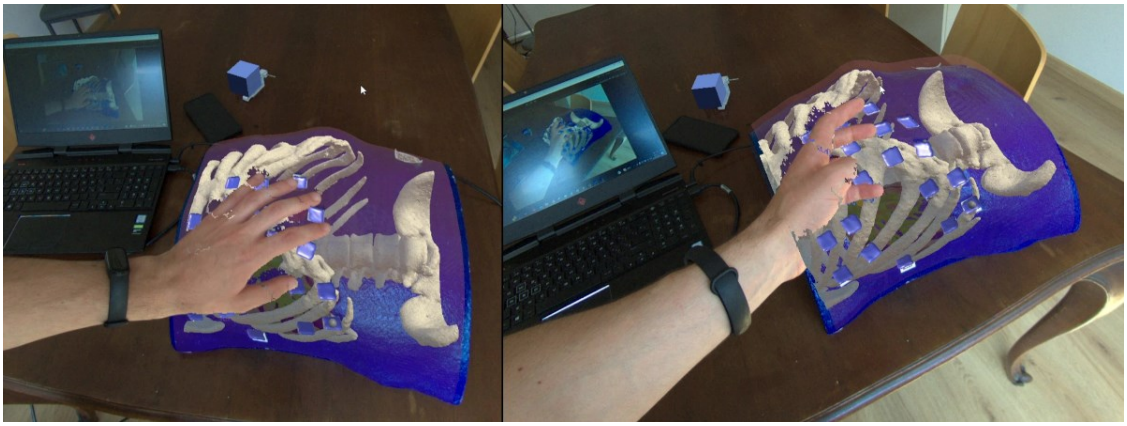

Figura 48: il frame di destra mostra una classificazione peggiore rispetto al frame di sinistra

Si può osservare in Figura 48 la bassa consistenza della classificazione durante l'utilizzo del sistema.

Come precedentemente accennato, si può imputare questo comportamento al mancato matching tra le immagini di destra e di sinistra, oppure ad un calcolo non corretto della disparità. Fattori come la sfocatura da moto influiscono molto su questo task.

Va inoltre sottolineato come il manichino di test utilizzato non sia congeniale ai fini del calcolo della disparità da immagini stereoscopiche: il materiale di cui è composto presenta delle componenti di luce speculare molto più marcate rispetto alla pelle umana.

La specularità/riflettività dei materiali è tendenzialmente da evitare in applicazioni basate su stereo matching, in quanto genera delle differenze patologiche tra le immagini acquisite dalla camera di sinistra rispetto a quelle della camera di destra.

Ci aspettiamo quindi dei risultati leggermente migliori nelle condizioni di utilizzo standard del sistema Endosight.

Il frame rate gioca un altro importante limite per il dominio applicativo di Endosight; la pipeline che permette la triangolazione della mesh della mano genera un calo medio del frame rate del 1125% (calcolato su laptop con IntelCore ® I7 9750H, Nvidia GTX 1660Super, 32GB RAM), assolutamente non adatto per task real time.

6 IPOTESI PER SVILUPPI FUTURI

Di seguito sono elencati alcune ipotesi di miglioramento del sistema:

1. Il sistema attuale non implementa accelerazione hardware su GPU (Graphics Processing Unit).
   La libreria Depth_Utility presenta la possibilità di gestire gli algoritmi di stereo matching attraverso computazione accelerata CUDA (utilizzabile con schede video Nvidia compatibili).
   Implementare questa funzionalità potrebbe snellire in maniera sostanziale i tempi di calcolo della mappa di disparità.

2. Per evitare risultati errati della disparity map, dovuti a incoerenze delle texture tra immagine di sinistra e immagine di destra, potrebbe essere utile implementare delle modifiche hardware al sistema Endosight: potrebbero essere installati dei filtri polarizzatori sulle lenti della stereocamera, finalizzati a minimizzare l'influenza delle riflessioni speculari delle texture inquadrate.

3. Una modifica radicale alle logiche di creazione della mesh della mano potrebbe comportare una classificazione migliore della mano: la mappa di disparità ottenuta a valle degli algoritmi di stereo matching potrebbe essere migliorata attraverso algoritmi di deep learning, addestrati per ricostruire una mappa di profondità nelle condizioni operative specifiche di Endosight.

4. Potrebbe essere implementata una logica di gestione dello stencil buffer attraverso l'interfaccia di Endosight: l'utente potrebbe decidere di renderizzare in primo piano le strutture anatomiche di suo interesse. Ad esempio, la lesione da trattare potrebbe essere riportata in primo piano in funzione della preferenza del radiologo interventista. La personalizzazione del software potrebbe rappresentare un punto chiave per la commercializzazione di Endosight.

7 BIBLIOGRAFIA

1) Milgram, P., Takemura, H., Utsumi, A., and Kishino, F. (1994). **Augmented reality: a class of displays on the reality-virtuality continuum**. Proc. SPIE 2351, 282–292. doi: 10.1117/12.197321.

2) Emmanuel Dubois, Laurence Nigay, Jocelyne Troccaz, Olivier Chavanon, Lionel Carrat, **Classification Space for Augmented Surgery, an Augmented Reality Case Study**, A. Sasse and C. Johnson (eds.), *Proceedings of Interact'99, IOS Press*, (1999), Edinburgh (UK), p. 353-359.

3) Ho-Gun Ha, Jaesung Hong, **Augmented Reality in Medicine**, *Hanyang Medical Reviews*, 2016;36:242-247.

4) Tobias Sielhorst, Christoph Bichlmeier, and Sandro Michael Heining, and Nassir Navab, **Depth Perception – A Major Issue in Medical AR: Evaluation Study by Twenty Surgeons**, R. Larsen, M. Nielsen, and J. Sporring (Eds.): MICCAI 2006, LNCS 4190, pp. 364–372, 2006. Springer-Verlag Berlin Heidelberg 2006.

5) Jonathan David Pfautz, **Depth perception in computer graphics**, *Technical Report Number 546, Computer Laboratory University of Cambridge*, 2002, UCAM-CL-TR-546 ISSN 1476-2986.

6) Catherine Diaz, Michael Walker, Danielle Albers Szafir, Daniel Szafir, **Designing for Depth Perceptions in Augmented Reality**, University of Colorado Boulder.

7) John Rhoades, Greg Turk, Andrew Bell, Andrei State, Ulrich Neumann and Amitabh Varshney, **Real-Time Procedural Textures**, Department of Computer Science University of North Carolina at Chapel Hill.

8) Rostam Affendi Hamzah, Haidi Ibrahim, Anwar Hasni Abu Hassan, **Stereo Matching Algorithm for 3D Surface Reconstruction Based on Triangulation Principle**, School of Electrical & Electronic Engineering Engineering Campus, Universiti Sains Malaysia, *2016 1st International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, Yogyakarta, Indonesia.

9) Fabrice Neyret, Raphael Heiss, Franck Senegas. **Realistic Rendering of an Organ Surface in Real Time for Laparoscopic Surgery Simulation**. *The Visual Computer*, Springer Verlag, 2002, 18 (3), pp.135–149. ff10.1007/s003710100118ff. ffinria-00537497f.

8 SITOGRAFIA

1) https://www.accuvein.com/
2) https://echopixeltech.com/
3) https://www.proximie.com
4) https://live.touchsurgery.com/
5) https://www.insideradiology.com.au/interventional-radiology/
6) https://docs.unity3d.com/Manual/render-pipelines.html
7) https://hupuu.com/camera/1-1.8-sony-4k-cmos-imx334-en781-4k-8mp15fps-ex-sdi-1080p-hd-sdiahd-hd-sdi-analog-starlight?sku=CBB-OS2
8) https://fpcv.cs.columbia.edu/
9) https://learn.microsoft.com/it-it/dotnet/architecture/modern-web-apps-azure/architectural-principles
10) https://docs.opencv.org/3.4/d9/dba/classcv_1_1StereoBM.html
11) https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall
12) https://commons.wikimedia.org/wiki/User:Walber
13) https://medium.com/@douglaspsteen/precision-recall-curves-d32e5b290248

Politecnico
di Torino