Alma Mater Studiorum
Università degli studi

Campus di Cesena
Scuola di Scienze

Modellazione di Mesh in Realtà Aumentata

presentata da
Joseph Giovanelli
relatore
Dr. Damiana Lazzaro

Corso di Laurea in
Ingegneria e Scienze Informatiche

Relazione finale in
Computer Graphics

Session II
Ottobre 2017

A un Astro che brilla ancora.

Introduzione

L'obiettivo del progetto consiste nella realizzazione di un motore grafico in realtà aumentata.
L'idea è quella di offrire una nuova interfaccia tridimensionale a tutti quei servizi che, nella realizzazione di una scena 3D, hanno difficoltà ad essere compiuti su uno schermo 2D. Per per far sì che queste operazioni vengano facilitate è necessario che si offra un'interazione in tutte e tre le dimensioni in contemporanea.
In particolare, in questa tesi si è scelto di sviluppare un'applicazione strettamente inerente alla computer graphics che permetta di modellare mesh 3D in realtà aumentata e poi, una volta create, dia l'opportunità di modificarle a proprio piacimento.
Diverse sono le casistiche in cui il monitor a due dimensioni ponga un limite invalicabile e costringa lo sviluppatore a effettuare più operazioni perditempo allo scopo di riuscire nel proprio obiettivo.
Consideriamo l'esempio della classica scena del film di King Kong in cui, mentre il gorilla sale il grattacielo, una telecamera gira intorno ad esso, creando una spirale che via via si ristringe verso l'alto.
Lo sviluppatore nell'inserire ciascuno dei punti, per i quali la curva interpolante dovrà passare, dovrà:

- posizionarsi su un piano con una certa profondità, fissando quindi la coordinata su un asse;
- selezionare il punto a schermo, fissando le altre due coordinate sugli assi rimanenti.

La semplice operazione di determinare un punto ha comportato all'esecuzione di più comandi per spostarci nello spazio, rendendo il tutto poco naturale e intuitivo.

Nasce quindi l'esigenza di navigare lo spazio in tutte le sue dimensioni, eliminando le classiche periferiche come monitor e mouse, che descrivono solo punti 2D. Infatti, non è possibile e pensabile esprimere in input una terza dimensione su uno schermo: la profondità non è dettata da alcun fattore.
Il software sviluppato in questa tesi è in realtà aumentata per far sì che l'utente possa creare gli oggetti di cui ha necessità proprio in relazione a ciò che lo ci circonda, in modo tale che egli abbia un feedback immediato di come il suo prodotto prende forma. L'interazione con gli oggetti vuole essere il più naturale possibile e, a tal fine, si è scelto di usare il tracciamento delle mani per comprendere che cosa l'utente voglia selezionare, spostare, ruotare, ingrandire o disegnare. Le applicazioni di un software di questo genere sono numerose; infatti, nella fase di ristrutturazione di un casa si faciliterebbe, grazie al suo forte impatto con la realtà, la creazione di mobili su misura; come, allo stesso tempo, nello studio dell'anatomia del corpo umano, si potrebbero comprendere meglio gli organi, se essi si potessero studiare, vedere e toccare proprio come se fossero sopra al nostro tavolo.
La simbiosi tra la piattaforma in realtà aumentata Vuforia e la periferica USB di tracciamento Leap Motion ha reso possibile lo sviluppo di questo software.
In questo testo verranno trattate solamente le scelte progettuali basilari e di maggior rilevanza senza scendere nel dettaglio tecnico di come molti aspetti sono stati effettivamente implementati; si tralasceranno molti dettagli di programmazione software per quanto riguarda i servizi multi utente e multi piattaforma, la connessione client-server tramite rete LAN e l'ottimizzazione e la serializzazione dei dati. Si vuole dare maggior rilievo, invece, all'aspetto del progetto inerente alla computer graphics e si scenderà più nello specifico nella trattazione del modulo di creazione delle mesh.
Nei capitoli successivi andremo ad esaminare nel dettaglio:

- le specifiche progettuali, le tecnologie scelte per adempirle e le motivazioni di tali scelte.In questo capitolo tratteremo il framework Unity, il motore grafico Blender, la periferica Leap Motion e la piattaforma Vuforia;

- la progettazione e l'implementazione dell'architettura di base, esplicitando le scelte progettuali di massima in modo tale da avere una inquadratura 360 dell'elaborato;
- la teoria matematica che ne sta alla base e comprende la creazione di curve e superfici e gli algoritmi di triangolazione studiati per la graficazione delle mesh;
- la progettazione e l'implementazione dettagliata del modulo inerente al tracciamento delle mani, l'interpolazione di punti tramite curve spline e la creazione della mesh in realtà aumentata.

Indice

Elenco delle figure

1

Le Tecnologie

In questo capitolo si forniranno, in primis, brevi cenni storici sull'evoluzione dell'informatica negli anni. Questa contestualizzazione è necessaria allo scopo di riuscire a capire meglio le esigenze degli utenti di oggi giorno dal punto di vista dell'interazione. In seguito, sarà offerta una trattazione sugli aspetti accennati nell'introduzione, come la realtà aumentata e il tracciamento delle mani. Solo successivamente andremo ad esaminare framework e piattaforme utilizzate nel progetto.

1.1 Una nuova Human Computer Interaction

L'interazione uomo-macchina è stata oggetto di studio sin dalla nascita dei primi computer, essendo di fatto il computer stesso creato e progettato per essere usato dall'uomo. L'HCI, human-computer interaction, è una disciplina a tutti gli effetti che copre aspetti di svariati campi che spaziano dall'informatica alla psicologia, dalle scienze cognitive all'ergonomia e il design. Il principale obiettivo di questa è l'usabilità, definita come misura con cui un prodotto può essere utilizzato dagli utenti per raggiungere specifici obiettivi con efficacia, efficienza e soddisfazione.

I primi calcolatori avevano ben poco di interattivo con gli utenti. La prima vera svolta è stata l'introduzione degli schermi grafici, i quali hanno portato alla creazione di dispositivi tali da faciltare l'interazione con essi. Molte di queste tecniche di interazione si devono alle ricerche e agli esperimenti effettuati dalla Xerox, che può essere considerata madre del dispositivo più usato in questo contesto: il mouse, inventato negli anni sessanta ma commercializzato solo intorno agli anni ottanta.

Il modo di interagire con i computer dal periodo che intercorre dall'era del mouse ad oggi è cambiato: i computer hanno cominciato a cambiare forma, diventando sempre più portatili, PC con touchpad integrato, fino a diventare tascabili, smartphone usabili tramite touchscreen.

La continua introduzione di nuovi dispositivi sempre più piccoli e interattivi nelle nostre case, uffici e auto ha portato a una loro pervasività e alla possibilità di creare nuovi servizi interattivi, che seguano l'utente nei suoi spostamenti e che si adattino alle sue esigenze. Inoltre, nel corso dell'ultimo decennio sono stati rilasciati diversi dispositivi innovativi ma, forse, più per svago che per altro.

Il Kinect è un accessorio che ha visto la luce nel giugno 2010 ed è stato pensato per l'utilizzo con la console Xbox 360 proprietaria Microsoft; la sua funzione è il riconoscimento del corpo umano, dalla testa ai piedi.

Google nel 2014 lancia un dispositivo, il cardboard, che permette all'utente, tramite l'uso di uno smartphone, di essere travolto da una realtà "non vera", meglio conosciuta con il nome di realtà virtuale.

Purtroppo, però, dagli ormai lontani anni settanta l'interazione negli ambienti di lavoro non ha fatto grandi progressi e gli strumenti che ancora oggi vengono usati sono gli stessi di una volta: le interfacce WIMP. Window Icon Menu Pointer è un acronimo che rappresenta tutti quei sistemi che utilizzano gli elementi finestra, icona, menu e puntatore e, avendo solo strumenti 2D, richiede il solo supporto delle classiche periferiche di input di un Personal Computer, come ad esempio il monitor, la tastiera, il mouse.

La comunicazione uomo-computer mira a essere simile, se non uguale, alla comunicazione tra esseri umani, all'interazione naturale, all'usabilità estrema che comporta un utilizzo immediato e spontaneo.

Il progetto sviluppato in questa tesi vuole proprio cercare di dare una dimostrazione di come l'uso di un calcolatore può essere migliorato e portato a un'interazione più naturale, senza perdere alcun tipo di usabilità. Come già anticipato, questa realtà è già presente, grazie a dispositivi quali il Kinect e i cardboard, ma solo in contesti di simulazione o videogame. Portare questo tipo di dispositivi di input/output in un contesto lavorativo e nella vita di ogni giorno offre, al futuro dell'informatica, la possibilità di cambiare drasticamente il modo di pensare quando si lavora al computer, durante l'acquisto online di un articolo o nella programmazione di un viaggio.

Qualsiasi attività che coinvolga l'uso del nostro laptop, tablet o smartphone potrà essere eseguita senza l'astrazione di un'interfaccia 2D. Qualcosa di simile è stato realizzato nell'ideazione dei Google Glasses, un programma di ricerca e sviluppo di Google Inc. con l'obiettivo di creare un paio di occhiali dotati di realtà aumentata. Quest'ultima, a differenza della realtà virtuale, ha l'obbiettivo di aumentare il contenuto informativo di tutto ciò che ci circonda, annotando sugli oggetti reali informazioni inerenti ad essi o creandone di nuovi. Un esempio di realtà aumentata può essere un'applicazione che,

entrando in un supermercato, riconosca determinati prodotti e informi l'utente dei relativi dettagli o che, guidando con dei google glasses, ci evidenzi la strada da percorrere così da non togliere lo sguardo dall'asfalto e non distrarci nel guardare il navigatore.
Nel nostro caso, a differenza di quelli precedenti, ci si è immaginati un mondo in cui non solo si possano vedere degli oggetti virtuali ma ci si possa anche interagire: il lavoro al computer non è più inteso come sedersi su una sedia e, tramite mouse e tastiera, manipolare un puntatore a schermo, ma entrare in una stanza e indossare un visore che permetta di personalizzare l'ambiente in cui si è, proprio come il desktop, e di interagire in maniera naturale con esso tramite l'uso delle mani. Comprare un articolo online come per esempio una scarpa permetterebbe di visionarla e osservarla nel dettaglio come se fosse proprio davanti a noi, appuntarsi una nota vorrebbe dire scrivere in modo naturale su un foglio e così via per il resto: prenotare un viaggio, fare una ricerca su google o sfogliare i post di un amico su Facebook.

1.2 Realtà aumentata e come realizzarla

La realtà aumentata non va assolutamente confusa con la realtà virtuale, di cui abbiamo parlato precedentemente. Quest'ultima è detta virtuale perchè crea un ambiente totalmente artificiale, costruito al computer, e lo rende credibile come se fossimo catapultati in un altro mondo.
I visori come il Google Cardboard riescono a dare questa esperienza grazie all'ausilio degli smartphone. L'immagine a 360 gradi di un panorama è visualizzata sullo schermo 2D. Essa viene ingrandita in modo tale da vedere solamente una porzione di essa, tanto quanto se ne vedrebbe se si stesse osservando il paesaggio con i proprio occhi, e sdoppiata in due immagini distinte, una che grafica maggiormente la parte sinistra e l'altra la parte destra. Inserendo lo smartphone con l'immagine così composta nel visore, equipaggiato di apposite lenti che la ingrandiscono davanti ai propri occhi, da' la sensazione a chi lo utilizza di trovarsi realmente immerso nello scenario.

Figura 1.1: CardBoard della Google Inc. che mostra la struttura sopra descritta

Inoltre, la porzione del panorama visualizzato viene cambiata in base ai sensori del-

lo smartphone, oscilloscopio e giroscopio, per farla variare in relazione ai movimenti della testa dell'utente.

La realtà aumentata, proprio per distinguerla da quella virtuale, viene anche detta realtà mediata, dove il mediatore è l'elaboratore. Quest'ultimo, tramite la fotocamera, riconosce un certo elemento e ne grafica immagini e testi intorno a sé, senza oscurarlo del tutto.

Molte applicazioni in realtà aumentata non necessitano di un visore ma si limitano a mostrare la realtà mediata sul display e l'interazione è effettuata tramite bottoni sul touchpad. Un esempio ce lo offre la rivista di scienza Focus. Oltre a sfogliarla, si può provare l'esperienza della realtà aumentata: viste attraverso lo smartphone le pagine di Focus si animano, prendono vita, regalando un'esperienza multimediale e in 3D, con animazioni, video, test e immagini immersive a 360°.

Figura 1.2: In mostra Focus A+, l'applicazione mobile della rivista scientifica

Per rendere il tutto più realistico viene sfruttato il visore e le tecniche precedentemente viste nella realtà virtuale. Applicando agli oggetti graficati una texture o un materiale è sorprendente quanto sia realistica la scena venutasi a creare. Nella figura 1.1 possiamo ora notare la camera che nel visore è presente ed è adetta a questo scopo.

1.3 Tracciamento delle mani tramite Leap Motion

In un progetto che promette la modellazione di mesh in modo innovativo e con un HCI (Human Computer Interaction) naturale non può non mancare un dispositivo fuori dagli schemi, nel nostro caso si chiama Leap. Fondato dall'azienda americana Leap Motion, il Leap è un dispositivo talmente piccolo che può tranquillamente essere integrato nei portatili di domani, nelle tastiere o nei monitor e nelle tv. Incaricato del tracciamento delle mani, riesce addirittura a distinguere quale dito si sta utilizzando, oppure se si ha in mano un altro oggetto, come una matita.
Il motivo che ha spinto gli sviluppatori a fondare l'azienda è stata la teoria che c'è alla base di questo testo: la convinzione che un tipo di input basato sulla semplicità e sulla naturalezza un giorno permetterà di sfruttare al massimo l'efficacia delle tecnologie.

Figura 1.3: Semplice funzionamento del Leap

Negli ultimi anni sono nati molti dispositivi simili a Leap Motion, basati sul riconoscimento dei gesti e l'interazione con il computer tramite i movimenti. Nella maggioranza dei casi questi dispositivi sono stati progettati per creare immersione nel mondo dei videogame e non su posti di lavoro. Dispositivi come Wiimote, Playstation Move o

Kinect non sono adattabili ad un uso quotidiano; uno dei punti di forza di questa startup è l'aver creato una periferica che necessita solamente di essa e delle proprie mani. Il Leap si collega al calcolatore con un attacco USB e diventa subito operativo.
La periferica usb presenta due telecamere e tre led infrarossi, che osservano un'area pari a una semisfera di diametro un metro, in grado di riconoscere fino a quattro mani contemporaneamente, con precisione millimetrica. Le dimensioni talmente ristrette di questo dispositivo gli permettono di essere montato su qualsiasi visore in modo tale da avere un'unico dispositivo extra-sensoriale.
La StartUp che sviluppa il Leap Service, il daemon in background che permette il rilevamento delle mani, non ha ancora rilasciato questo servizio per i sistemi operativi smartphone. Da quest'ultima limitazione si esclude l'uso di tanti Leap Motion, in modalità Head Mounted e si concretizza l'uso di un solo Leap in modalità Desktop (fIgure 1.3).

(a) Leap modalità Head Mounted **(b)** Visore con supporto Leap

Figura 1.4: Un esempio delle capacità del Leap Motion unito alla Realtà Aumentata

Leap Motion sin dalle origini mette a disposizione un SDK in diversi linguaggi di programmazione per lo sviluppo di applicazioni legate al Leap, offrendo pieno supporto alla communità degli sviluppatori. Le API fornite sono in continua evoluzione e usando linguaggi di livello come il C# integrato con il framework Unity vengono anche offerti numerosi strumenti di sviluppo.

1.4 Strumenti utilizzati e valutazioni

A lungo si è trattata l'importanza in questo progetto di tesi di aspetti come realtà aumentata e interazione uomo-macchina. Ulteriori specifiche del progetto sono:

- realizzare un applicativo multiutente, che dia la possibilità di vedere e modificare in real-time e in maniera condivisa gli oggetti; le modifiche sulle mesh possono essere attuate modificando in maniera cooperativa l'oggetto.
- realizzare un applicativo multipiattaforma in cui l'unico limite sia l'esigenza di una fotocamera incorporata nel dispositivo, in maniera da riconoscere il tavolo da lavoro e graficare su esso le mesh.

L'esigenza di realizzare un applicativo multiutente e multipiattaforma tale da unire numerose tecnologie, ha portato all'uso di un framework come Unity. Esso, oltre ad avere una community attiva che sviluppa plug-in ed estensioni per numerose tecnologie, ben si presta a creare un software Client-Server e platform-indipendent, dato che il suo principale uso è quello per la creazione di video-game.
Viene così definito il framework usato dai Client, smartphone con Sistema Operativo Android, e dal Server, un NoteBook con Sistema Operativo Windows 10. Unity come ambiente di sviluppo permette vari linguaggi di programmazione, quello scelto per tale progetto è stato il C#.

I principali argomenti in esame, ovvero modellazione di mesh e realtà aumentata, hanno portato all'uso di un motore grafico, per usare algoritmi di manipolazione su mesh, e di una piattaforma AR, per usare API di alto livello per graficare gli oggetti nella realtà.
Il motore grafico scelto è stato Blender, software libero e multipiattaforma estremamente diffuso e robusto per le numerose funzionalità che offre. Questa scelta è derivata, oltre dall'uso di questo software in precedenti corsi, dalla sua dettagliata documentazione e per la possibilità, essendo esso open-source, di aggiungere funzionalità ad esso,

tramite script in Python.
La piattaforma AR usata è Vuforia, la quale ha piena compatibilità con Unity, che garantisce le funzionalità di realtà aumentata su sistema operativo Android e Windows. Vuforia supporta la CardBoard mode e, in modo da garantire un'esperienza più coinvolgente, si è scelto di usare questa modalità con un dispositivo già in possesso, l'UNI-VR, CardBoard della UniEuro.

Nella prima fase di valutazione delle tecnologie da usare si è avuto un ripensamento sul framework Unity. Nonostante esso sia usato in maniera molto pervasiva nello sviluppo di video-game, è usuale importare modelli da altri motori grafici come Blender, questo per la sua mancanza di algoritmi efficienti per la manipolazione di mesh. UnrealEngine, invece, oltre ad offrire le stesse funzionalità di Unity, dispone di un motore grafico addirittura superiore a quello di Blender e ha pieno supporto da parte di Leap Motion per integrare il tracciamento delle mani.
La soluzione Unreal sarebbe stata probabilmente più efficiente rispetto a quella Unity+Blender ma, purtroppo, Unreal non supporta la piattaforma Vuforia e i plugin esistenti per la realtà aumentata (UnReal4AR | ARToolkit) sono a pagamento. L'università ETH Zürich di Zurigo ha sviluppato un plug-in per colmare questa lacuna, e integrare la realtà aumentata, ma si è prediletta l'affidabilità e la documentazione dettagliata di Vuforia, creando così l'architettura che vedremo in maniera più dettagliata nel prossimo paragrafo.

1.5 Vuforia

Le capacità di riconoscimento e di monitoraggio di Vuforia possono essere utilizzate su una varietà di immagini e oggetti.

- Gli Image Target sono immagini piatte, come le fotografie di un rullino, la carta d'imballaggio di un prodotto o un manifesto;
- VuMarks sono marcatori personalizzati che possono codificare una vasta gamma di formati. Essi supportano sia l'identificazione univoca che il monitoraggio per applicazioni AR;
- VuMark è il nuovo codice a barre: permette al produttore la libertà di un design personalizzato e adatto al brand, mentre codifica contemporaneamente i dati e agisce come un target AR. I disegni di VuMark sono completamente personalizzabili, in modo da avere un VuMark unico per ogni oggetto unico;
- I Multi-Targets vengono creati utilizzando più di un Target e possono essere disposti in forme geometriche regolari (ad es. Scatole) o in qualsiasi disposizione arbitraria;
- I Cylinder Target sono immagini avvolte su oggetti di forma quasi cilindrica (es. Bottiglie di bevande, tazze di caffè, lattine di soda);
- Il riconoscimento del testo consente di sviluppare applicazioni che riconoscono parole da un dizionario di 100.000 parole inglesi;
- Object Target: Vuforia può riconoscere e tracciare una vasta gamma di oggetti 3D. Il riconoscimento di oggetti consente di creare oggetti target mediante la scansione di oggetti fisici. Consente di creare applicazioni che riconoscano e traccino oggetti intricati.

Oltre alla Target Recognition, Vuforia fornisce una consapevolezza e comprensione dell'ambiente fisico dell'utente.
Smart Terrain è una tecnologia innovativa che può ricostruire l'ambiente fisico dell'utente come una mesh 3D. Consente agli sviluppatori di creare una nuova classe di giochi e di

esperienze realistiche di visualizzazione dei prodotti, in cui il contenuto del gioco può interagire con gli oggetti fisici e le superfici del mondo reale.
Una volta rilevato un bersaglio, la feature Extended Tracking consente il monitoraggio ad alto grado di persistenza.
Mentre il bersaglio esce dalla fotocamera, Vuforia utilizza altre informazioni dall'ambiente per dedurre la posizione dell'oggetto virtuale, monitorando visivamente l'ambiente. A questo scopo, Vuforia costruisce una mappa attorno all'obiettivo e assume che sia l'ambiente che l'obiettivo siano in gran parte statici.

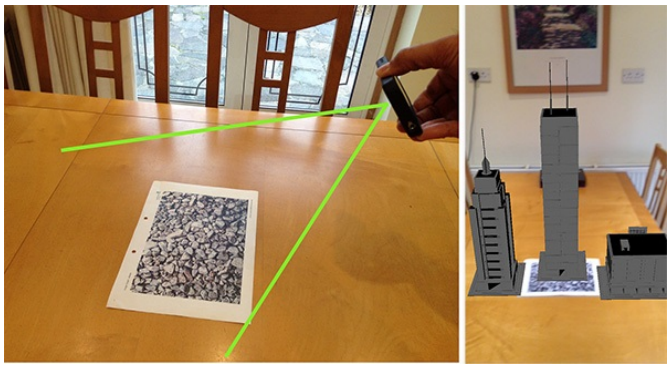


Figura 1.5: Vuforia con target inquadrato

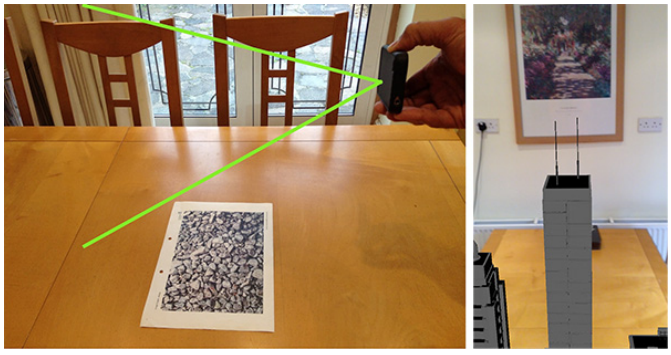


Figura 1.6: Vuforia con target non inquadrato

Il monitoraggio esteso può migliorare notevolmente due tipi di esperienza utente:

- Esperienze simili a giochi o giochi, con una grande quantità di contenuti dinamici che richiedono all'utente di puntare il dispositivo lontano dall'obiettivo in quanto l'utente debba seguire il contenuto virtuale;
- Visualizzazione di oggetti di grandi dimensioni come mobili, elettrodomestici, grandi arredi per la casa e anche modelli architettonici nella giusta scala e prospettiva.

Si utilizza questa funzionalità per facilitare la creazione di applicazioni più robuste.

2

Il progetto di interazione

Il software prodotto offre sostanzialmente due macro-feature: la creazione di mesh ottenuta per rivoluzione di curve spline definite a partire da punti nello spazio 3D rilevati mediante Leap Motion e l'interazione con esse. Ad un primo avvio viene reso disponibile, come mesh di prova, un cubo di un semplice materiale grigio con alto coefficiente di luce diffusiva, sul quale possono essere attuate diverse trasformazioni:

- le elementari, ovvero traslazione, rotazione e scalatura;
- altre più complesse come l'estrusione di facce e la modifica della geometria dei punti.

Questo capitolo vedrà nel dettaglio l'interazione con le mesh grazie al supporto del motore grafico Blender. Allo scopo di rendere più chiara e comprensibile questa sezione della relazione si andrà ad esaminare prima la progettazione del prototipo e successivamente la sua implementazione dettagliata, divisa per moduli.

2.1 Progettazione

Esaminati attentamente i servizi e i limiti delle tecnologie usate possiamo creare uno schema che descriva il comportamento del sistema realizzato.

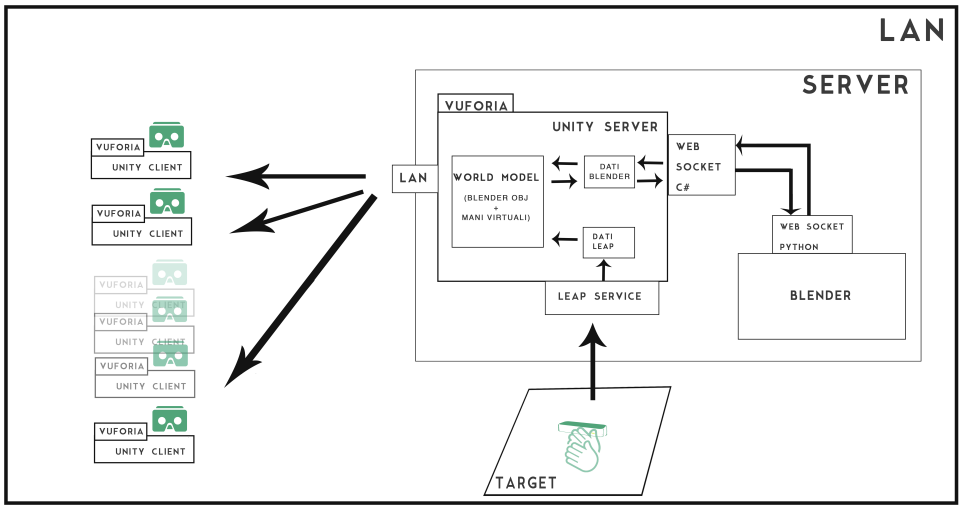


Figura 2.1: Disegno Progettuale

La prima cosa che si osserva dall'immagine sopra riportata è l'appartenenza di ogni elemento a una LAN. Infatti, per far sì che l'applicazione sia multiutente si è costruita un'architettura Client-Server in cui: il centro di calcolo (Server) preserva tutta la logica, la situazione del mondo e gestisce l'interazione con l'utente, mentre, i dispostivi smartphone (Client) ricevono solamente i dati (mesh presenti nel tavolo da lavoro) dal Server, tramite LAN.

Unico compito del Client è graficare gli oggetti presi in input e, per fare ciò, fa uso di Vuforia che gestisce la prospettiva, l'angolazione e tutto ciò che riguarda la visualizzazione degli oggetti in modo corretto a seconda della posizione dell'utente.

Vuforia visualizzerà le mesh quando, tramite algoritmi di object recognition, riconosce-

rà un'immagine chiamata target.
Anche il Server tiene conto della posizione di quest'ultimo, che sarà il vero e proprio piano da lavoro, infatti si considera come origine del mondo, punto di coordinate $(0, 0)$, il centro del target. Con l'obiettivo di far coincidere le coordinate del tracciamento delle mani con le coordinate del mondo si è collocato anche il Leap Motion nella medesima origine degli assi. Con tale disposizione di target e Leap Motion, l'utente riesce a interagire con le mesh mostrate, dato che esse vengono visualizzate all'interno del raggio di visione della periferica.
Nel momento in cui le mani entrano nell'area in questione, i dati di queste vengono mandati al Server tramite il Leap Service e tramite questi vengono creati i modelli virtuali delle mani adiacenti alle proprie, per avere un feedback dei punti rilevati. Le mani virtuali e le mesh formano il "World Model" presente nel Server, più nello specifico nello Unity Server, che riconosce, tramite le API del Leap, gli eventi che accadono nel mondo reale. Ogni volta che un evento viene scatenato, un'operazione deve essere attuata e viene inviato a Blender uno specifico comando per attuare la relativa reazione, il quale, alla fine di questa, invierà i dati modificati della mesh a Unity che provvederà ad aggiornare il World Model.

2.2 Implementazione

In questa sezione si andranno ad esaminare più nel particolare come si sono realizzati i vari moduli del progetto.

2.2.1 Blender e Unity Server

Per la comunicazione con il motore grafico si è usata una web socket e si è definito un protocollo che ottimizzasse il più possibile le performances; si è cercato, infatti, di mandare sempre solo le informazioni strettamente necessarie.
Sono possibili, a un comando impartito da Unity, due trasformazioni: sui vertici o sulla matrice del mondo.
La prima trasformazione avviene quando si entra in una modalità "Edit" della mesh e la si può modellare spostando vertici a proprio piacimento.
La matrice del mondo, invece, fa riferimento alle coordinate che in computer graphics vengono chiamate locali. Infatti, nei motori grafici, ogni componente vede come origine degli assi del proprio mondo il proprio centro e, solo successivamente, nella pipeline grafica, queste verranno mappate nelle coordinate della camera, con centro del mondo uniformato. Da questa definizione si può capire che ogni mesh possiede la propria matrice, relativa al proprio stato, e verrà modificata ogni qualvolta avviene un'operazione di trasformazione di posizione, orientamento o dimensione. Da quanto descritto sopra si evince che le due trasformazioni interessano dati completamente diversi ed indipendenti; quindi, si è provveduto allo scambio di questi dati solo quando essi venivano effettivamente modificati.
Per quanto concerne la parte di Blender, la web socket è stata costruita tramite un addon, scrivendo uno script Python. Si è risaliti, tramite la documentazione, alle strutture dati che esso offre per reperire informazioni sugli oggetti e come applicare le trasfor-

mazioni su di essi. In questo caso, per alleggerire il tutto, sarebbe meglio non usare Blender come ambiente di sviluppo grafico, ma come demone in background.
Nel prototipo, per questioni di comodità di debug, si è mantenuto Blender con l'add-on ma, in ottica di estendibilità, si è compilato Blender come modulo oggetto da poter usare all'interno di un proprio script.
Per quanto concerne la parte Unity Server si è costruita la socket tramite script C# agganciati a dei GameObject di Unity.
La classe che offre i servizi di comunicazione invia a Blender i comandi tramite un'enumerazione e scrive i dati ricevuti parsandoli e mettendoli in un Buffer. Il Buffer viene storicizzato per poi essere condiviso agli smartphone dal modulo che si occupa della parte Client-Server.

2.2.2 Unity Server e Unity Client

Al momento dell'avvio dell'applicativo in base al tipo di dispositivo, Server o Client, vengono instanziati o meno certi componenti. I moduli che riguardano la comunicazione con Blender, l'interfacciamento con il Leap Motion e la comunicazione dei dati ai Client vengono inizializzati solo sul Server.
Nel Server, il Buffer in singleton viene gestito da un modulo, il DataProvider, che riconosce, grazie al loop continuo di Unity, quando nuovi dati sono disponibili dalla socket. Ogni volta che esistono nuovi dati si devono aggiornare anche i Client e, tramite chiamate specifiche del Network Behaviour di Unity, si riesce a risalire allo stato del buffer sul Client e si riesce a capire se c'è la necessità di creare l'oggetto in questione o, se esiste già, di aggiornarlo.
Altra mole di dati che deve essere recapitata al Client sono i modelli virtuali delle mani; queste però non sono memorizzate nel Buffer dato che non sono oggetti provenienti da Blender. Come queste vengano costruite e come interagiscono con gli oggetti di Blender è compito di un altro modulo, che verrà visto nella sezione seguente, ma come

queste vengano inviate ai Client riguarda il gestore Client-Server.
I modelli, dopo esser creati dai dati del Leap, vengono compressi con un algoritmo noto in letteratura, il CLZF2, e poi serializzati in un vettore di byte, con l'ausilio di thread per migliorare le performance.
I dati vengono inviati ai Client, dove vengono deserializzati e decompressi per poi essere mostrati.
L'esigenza di creare delle proprie rappresentazioni delle mani e non usare quelle del Leap nasce dalla pesantezza di questi modelli e la conseguente lentezza di aggiornamento della visione del Client rispetto al Server.

2.2.3 Unity Server e Leap Motion

I dati resi disponibili dal Leap Motion tramite il Leap Service vengono usati essenzialmente in due modi: per riconoscere la collisione con un oggetto nel tavolo da lavoro e per rappresentare i modelli delle mani virtuali.
Per quanto riguarda il riconoscimento delle collisioni si è utilizzato il modulo del Leap chiamato Interaction Manager; il Leap Motion offre per Unity dei componenti per facilitare l'uso delle loro API in questo framework e la nuova versione dell'SDK ha visto ottimizzata in maniera molto intensiva questa sezione. Questo componente, istanziato solamente sul Server, permette di monitorare lo stato degli oggetti che hanno un Interaction Behaviour, ovvero possono interagire con le mani. A questo punto, tramite script C#, si sono implementati una serie di controlli per verificare quante mani stessero interagendo sulla mesh e in che modo. Attraverso il pattern Listener si è notificato il corrispettivo Handler dell'oggetto che, ricavata l'azione corrispondente, può inviare a Blender i dati.
Per sviluppare la propria rappresentazione delle mani si sono sfruttate le API del Leap che consentono di ricavare la distanza in millimetri da esso di tutte le parti della mano che vengono riconosciute. Al fine di alleggerire il modello si sono presi solo i dati ne-

cessari utili alla comprensione quali, il palmo della mano e, per ogni dito, la punta e la giuntura inferiore del metacarpo. Dopo aver costruito la propria struttura contenente i dati della mano e averla inviata ai Client, quest'ultimi rappresentano i dati tramite primitive estremamente leggere (cerchi e linee).

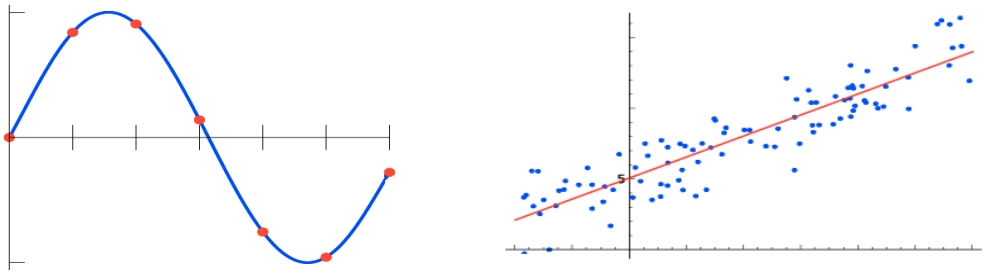
3

La teoria matematica delle curve

Questo capitolo provvederà a dare il necessario supporto matematico al lettore affinché si riesca a comprendere come vengono create le mesh.
L'obiettivo del progetto delle curve è realizzare un modulo che, tramite rivoluzione di una curva 3D sull'asse delle ordinate, componga un solido 3D. Questo modulo, per adempire a questo compito, riceve come input dall'utente i dati di un curva. I dati devono prima essere elaborati per poter essere usati, infatti la ricostruzione di dati discreti è un problema ben noto nell'informatica. A riguardo vengono studiati due approcci:

- L'interpolazione di dati sperimentali, questo nasce dall'esigenza di rappresentare in maniera continua un fenomeno reale di cui abbiamo solo una valutazione discreta;
- L'approssimazione dei dati sperimentali, si vuole costruire la curva che approssima i dati minimizzando la distanza tra la curva e i dati stessi.

Il primo caso viene studiato quando a partire da un set discreto di dati si vuole individuare una funzione che passi esattamente per questi valori; che ci permetta di poter costruire le valutazioni mancanti. Al contrario, il problema dell'approssimazione viene studiato quando i dati sono rindondanti e portano con sé una percentuale di errore.



(a) Interpolazione **(b)** Approssimazione

Figura 3.1: Interpolazione e approssimazione a confronto

Nel progetto in questione, piuttosto che un'interpolazione, si è scelto di attuare un'approssimazione dei dati discreti, data la loro natura: i punti descritti da una mano che disegna una curva sono affetti da errore.
Nonostante ciò tratteremo comunque il problema dell'interpolazione dato che, una volta appreso, ci servirà per comprendere meglio quello relativo all'approssimazione.
Una volta chiariti questi concetti per realizzare la curva si tratterà la tecnica con la quale viene effettuata la sua rivoluzione per creare la mesh.
A quest'ultima viene applicato uno degli svariati algoritmi di triangolazione presenti in letteratura, per far sì che essa venga visualizzata correttamente. In questa tesi se ne tratteranno alcuni a confronto.

3.1 L'interpolazione

In generale il problema dell'interpolazione consiste nel costruire una funzione che passi per tutti i punti assegnati.

3.1.1 Interpolazione polinomiale

Un possibile modo per risolvere questo problema consiste nell'individuare un polinomio di grado n che abbia quella determinata forma continua che cerchiamo. Quindi,

$$\begin{aligned} &\text{note le coppie} \quad (x_i, y_i) : i = 0, \dots n, \quad x_i \neq x_k, \quad i \neq k \\ &\text{dove} \quad y_i : i = 0, \dots n \quad \text{rappresentano le valutazioni di} \quad x_i \end{aligned}$$

L'interpolazione polinomiale si pone di trovare un polinomio P_n di grado n:

$$\begin{aligned} P_n &= a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \\ \text{tale che} \quad P_n(x_i) &= y_i : \quad i = 0, \dots n, \end{aligned}$$

Si ricava quindi il seguente sistema lineare :

$$\begin{cases} a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_n x_0^n = y_0 \\ a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_n x_1^n = y_1 \\ \dots \\ a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_n x_n^n = y_n \end{cases}$$

esprimibile in forma matriciale come $Aa = y$ dove:

$$A = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \qquad \text{matrice dei dati discreti} \quad (n+1) \times (n+1)$$

$$y = \begin{bmatrix} y_0 & y_1 & \cdots & y_n \end{bmatrix}^T \qquad \text{valutazioni delle} \quad x_i$$

$$a = \begin{bmatrix} a_0 & a_1 & \cdots & a_n \end{bmatrix}^T \qquad \text{vettore delle incognite}$$

Il sistema lineare ammette una, e una sola, soluazione se la matrice è quadrata e ha rango massimo.

La matrice A così costruita è detta matrice di Vandermonde, essa è quadrata perchè il numero delle variabili che abbiamo costruito è uguale al numero di condizioni a noi imposte dal problema e ha sempre rango massimo dato che $x_i \neq x_k, \quad i \neq k$.

La matrice di Vandermonde è una matrice molto mal condizionata.

Una matrice quadrata A si dice mal condizionata se piccole perturbazioni negli elementi di A, o piccole variazioni del vettore b, producono grandi variazioni nelle soluzioni x del sistema lineare $Ax = b$. Occorre, quindi, esprimere il polinomio in un'altra forma.

Nell'analisi numerica svariati sono i metodi di interpolazione polinomiale, tra i più noti Lagrange e Newton. Alcune considerazioni su questi tipi di polinomi:

- essi soffrono del cosiddetto fenomeno di Runge: al crescere del grado del polinomio la funzione si discosta ai bordi. Si ricorda che in un polinomio interpolante il grado cresce all'aumentare del numero dei punti dato che vengono concatenati n monomi;

- tramite alcune tecniche, come il polinomio di Chebychev, questo sgradevole fenomeno si può evitare, aumentando la convergenza ai bordi;

3.1.2 Interpolazione Spline

Sia $\Delta = \left\{ a \equiv x_0 < x_1 < \cdots < x_k < x_{k+1} \equiv b \right\}$

che genera $k+1$ intervalli così definiti: $I_i = \left[x_i, \quad x_{i+1} \right] \quad i = 0, \ldots k.$

Fissato $m < k$, si definisce spline polinomiale di ordine m (grado $m-1$) la funzione $s(x)$ che in ciascun intervallo I_i coincide con un polinomio $s_i(x)$ di ordine m e soddisfa nei punti x_i (nodi semplici) la seguente condizione:

$$\frac{d^j s_{i-1}(x_i)}{dx^j} = \frac{d^j s_i(x_i)}{dx^j} \quad i = 1, \ldots k \quad j = 0, \ldots m-2$$

Questa ci garantisce $s(x) \in C^{m-2}$, continuità fino alla $(m-2)$esima derivata.

$$s(x) = S_m(\Delta) = S_m \left\{ x_1, \ldots, x_k \right\}$$

La dimensione di $S_m(\Delta)$ è	$m+k =$	$(k+1)m-$	$(m-1)k$
$k+1$ intervalli di ordine m		$(k+1)m$	
a cui vanno tolte le $m-1$			$(m-1)k$
condizioni sui k punti di di raccordo			

3.1.2.1 Le Funzioni B-Spline

La base B-Spline genera lo spazio delle spline in modo stabile ed efficente. Si definisce $N_{i,m}(x)$ la funzione B-Spline di ordine m relativa a x_i tale per cui:

- $N_{i,m}(x) \neq 0 \quad x_i < x < x_{i+m}$ SUPPORTO COMPATTO
- $N_{i,m}(x) \geqslant 0 \quad x_i < x < x_{i+m}$ NON NEGATIVITA'
- $\int_{x_i}^{x_{i+m}} N_{i,m}(x)dx = \frac{x_{i+m}-x_i}{m}$ NORMALIZZAZIONE

In ciascun intervallo $I_j : j = i, \ldots i+m-1$ coincide un polinomio di ordine m $p(x) \in C^{m-2}$

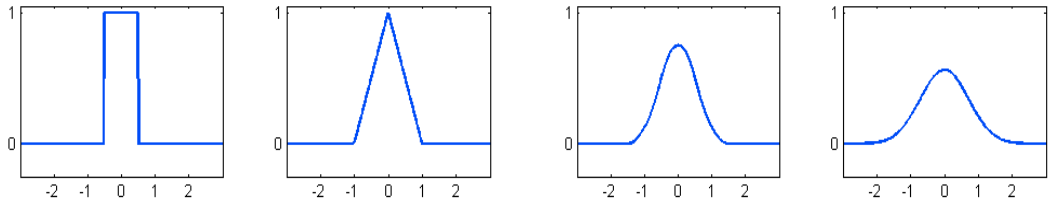


Figura 3.2: Rispettivamente $N_{i,1}, \quad N_{i,2}, \quad N_{i,3}, \quad N_{i,4}$

Le formule di Cox ci permettono di ricavare le B-Spline in maniera ricorsiva:

$$N_{i,1}(x) = \begin{cases} 1 & x_i < x < x_{i+1} \\ 0 & altrimenti \end{cases}$$

$$N_{i,h}(x) = \left(\frac{x - x_i}{x_{i+h-1} - x_i} N_{i,h-1}(x) + \frac{x_{i+h} - x}{x_{i+h} - x_{i+1}} N_{i+1,h-1}(x)\right) \quad h = 2, \ldots m$$

Per costruire $S_m(\Delta)$ che ha dimensione $m+k$ non ci bastano i k nodi veri, così aggiungiamo m nodi esterni a sinistra del primo estremo dell'intervallo e m nodi esterni a destra

dell'ultimo estremo dell'intervallo.

La nuova partizione viene chiamata partizione nodale estesa Δ^*.

$$\Delta^* = \left\{t_i\right\}_{-m+1}^{m+k} = \begin{cases} t_i < a \quad i = -m+1, \ldots 0 \quad \text{con } t_0 \equiv a \\ t_i \equiv x_i \quad i = 1, \ldots k \\ t_i > b \quad i = k+1, \ldots k+m \quad \text{con } t_{k+1} \equiv b \end{cases}$$

Possiamo definire $m + k$ B-Spline:

$$N_{i,h}(x) = \begin{cases} \frac{x-t_i}{t_{i+h-1}-t_i} N_{i,h-1}(x) + \frac{t_{i+h}-x}{t_{i+h}-t_{i+1}} N_{i+1,h-1}(x) & \textit{se} \quad t_i \neq t_{i+h} \\ 0 & \textit{altrimenti} \end{cases}$$

L'insieme $N_{i,m}(x) \quad i = m+1, \ldots k$ forma una base per $S_m(\Delta^*)$.

Quindi:

$$s(x) = \sum_{i=-m+1}^{k} a_i N_{i,m}(x)$$

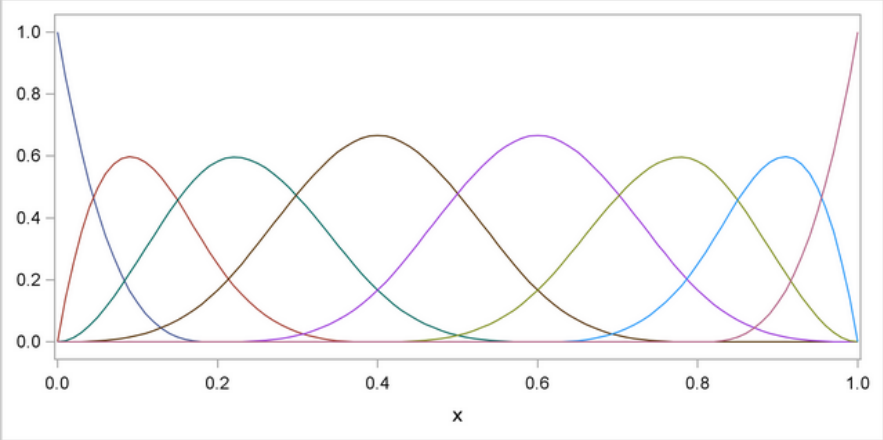


Figura 3.3: Esempio di come le $N_{i,m}$ generano lo spazio $S_m(\Delta^*), \quad m = 4$.

Esse applicate alle spline godono delle seguenti proprietà:

- se $x \in \begin{bmatrix} x_i & x_{i+1} \end{bmatrix}$
  - e non è un nodo ci sono m funzioni $\neq 0$
  - ed è un nodo ci sono $m-1$ funzioni $\neq 0$

- rappresentano una partizione dell'unità $\sum_{i=-m+1}^{k} N_{i,m}(x) = 1$

Quindi:

- per il supporto locale $s(x) = \sum_{j=l-m+1}^{l} a_j N_{j,m}(x) \quad \forall x \in \begin{bmatrix} -m+, \dots k \end{bmatrix}$ dato che le uniche $\neq 0$ sono da $l-m+1$ a l
- una $s(x)$ è combinazione convessa dei coeficienti a_j data la partizione dell'unità e quinidi

$$\begin{aligned} & \min \left\{a_j\right\} \leqslant s(x) \leqslant \max \left\{a_j\right\} && \forall j \in \begin{bmatrix} -m+1, \dots k \end{bmatrix} \\ \text{e quindi} \quad & \min \left\{a_j\right\} \leqslant s(x) \leqslant \max \left\{a_j\right\} && \forall j \in \begin{bmatrix} l-m+1, \dots l \end{bmatrix} \end{aligned}$$

- **Proprietà di variation diminishing**
  Il numero di variazioni di segno della spline è minore o uguale al numero di variazioni di segno $\left\{a_0, a_1, \dots a_k\right\}$

Per costruire una funzione $s(x) \in S_m(\Delta^*)$ dobbiamo studiare come posizionare i nodi veri $t_1, \dots t_k$ dell'insieme della spline interpolante $\Delta = \left\{a < t_1 < t_2 < \dots < b\right\}$.
Data la condizione $S(x_i) = y_i \quad i = 0, \dots N+1$
si ha che $N+2 = m+k \quad \Rightarrow \quad m = 4, \quad k = N+2-m = N-2$ nodi veri.

Il sistema risultante è:

$$\begin{cases} N_{1,4}(x_0)a_1 + N_{2,4}(x_0)a_2 + \cdots + N_{N+2,4}(x_0)a_{N+2} = y_0 \\ N_{1,4}(x_1)a_1 + N_{2,4}(x_1)a_2 + \cdots + N_{N+2,4}(x_1)a_{N+2} = y_1 \\ \vdots \\ N_{1,4}(x_{N+1})a_1 + N_{2,4}(x_{N+1})a_2 + \cdots + N_{N+2,4}(x_{N+1})a_{N+2} = y_{N+1} \end{cases}$$

Il sistema deve avere una, e una sola, soluzione.

Il Teorema di Schoemberg

Affinchè il rango sia massimo, per ciascuna funzione B-Spline deve esistere almeno un punto di interpolazione nel suo supporto.
Una scelta dei nodi che ci garantisce che il problema dell'interpolazione con le spline abbia una soluzione unica è la seguente:

$$t_0 = x_0 \quad t_1 = x_2 \quad \ldots \quad t_k = x_{N-1} \quad t_{k+1} = x_N$$

Un altro modo è il seguente:

$$t_0 = x_1 \quad t_2 = x_3 \quad \ldots \quad t_k = x_{N-2} \quad t_{k+1} = x_N$$

e le altre due condizioni vengono applicate agli estremi e prendono il nome di "condizioni ai bordi".

L'algoritmo di De Boor

Analogamente alle curve di Bezier, per la valutazione di una curva spline si considera l'algoritmo di De-Boor. Quest'ultimo ci permette di valutare una spline in tutta la sua parametrizzazione senza calcolare le funzioni base esplicitamente. Normalmente, ci si

dovrebbero ricavare le B-Spline che influiscono in quel determinato supporto ma tramite ricorsione, De-Boor, ci permette di evitare questo oneroso costo computazionale.
Se $x \in [r, r+1)$ si considerano solamente i h nodi attivi, cioè:

$$
\begin{aligned}
C(x) = \sum_{i=r-h+1}^{r} d_i N_{i,h}(x) = \\
&= \sum_{i=r-h+1}^{r} d_i \left[\tfrac{x-x_i}{x_{i+h-1}-x_i} N_{i,h-1}(x) + \tfrac{x_{i+h}-x}{x_{i+h}-x_{i+1}} N_{i+1,h-1}(x) \right] = \\
&= \sum_{i=r-h+1}^{r} d_i \left[\tfrac{x-x_i}{x_{i+h-1}-x_i} N_{i,h-1}(x) \right] + \sum_{i=r-h+1}^{r} d_i \left[\tfrac{x_{i+h}-x}{x_{i+h}-x_{i+1}} N_{i+1,h-1}(x) \right] = \\
&\text{le sommatorie devono essere ridefinite per le nuove funzioni di ordine } h-1 \\
&= \sum_{i=r-h+2}^{r} \left[\tfrac{x-x_i}{x_{i+h-1}-x_i} N_{i,h-1}(x) \right] + \sum_{i=r-h+1}^{r-1} d_i \left[\tfrac{x_{i+h}-x}{x_{i+h}-x_{i+1}} N_{i+1,h-1}(x) \right] = \\
&\text{e modificando i coefficienti e le funzioni base possono essere riunite} \\
&= \sum_{i=r-h+2}^{r} d_i \left[\tfrac{x-x_i}{x_{i+h-1}-x_i} N_{i,h-1}(x) \right] + \sum_{i=r-h+2}^{r} d_{i-1} \left[\tfrac{x_{i+h-1}-x}{x_{i+h-1}-x_i} N_{i,h-1}(x) \right] = \\
&= \sum_{i=r-h+2}^{r} \left[\tfrac{x-x_i}{x_{i+h-1}-x_i} d_i + \tfrac{x_{i+h-1}-x}{x_{i+h-1}-x_i} d_{i-1} \right] N_{i,h-1}(x)
\end{aligned}
$$

$$
\text{quindi } C(x) = \sum_{i=r-h+1}^{r} d_i N_{i,h}(x) = \sum_{i=r-h+2}^{r} d_i^{[1]} N_{i,h-1}(x)
$$

$$
\text{dove } d_i^{[1]} = \frac{x - x_i}{x_{i+h-1} - x_i} d_i + \frac{x_{i+h-1} - x}{x_{i+h-1} - x_i} d_{i-1}
$$

$$
\text{reiterando } C(x) = \sum_{i=r-h+1}^{r} d_i^{[1]} N_{i,h}(x) = \cdots = \sum_{i=r}^{r} d_i^{[h-1]} N_{i,1}(x) = d_i^{[h-1]}
$$

dato che $N_{i,1}(x) = 1$

L'idea è ridurre l'ordine delle B-spline per trovare i punti della curva. I passi dell'algoritmo ottenuti da quest'ultima formula possono essere rappresentati nella forma:

$$
\begin{array}{rccccccc}
d_{r-h+1} = & d^{[0]}_{r-h+1} & & & & & & \\
d_{r-h+2} = & d^{[0]}_{r-h+2} & d^{[1]}_{r-h+2} & & & & & \\
\vdots & & & d^{[2]}_{r-h+3} & & & & \\
\vdots & & & & \ddots & & & \\
d_{r-1} = & d^{[0]}_{r-1} & d^{[1]}_{r-1} & d^{[2]}_{r-1} & \cdots & d^{[h-2]}_{r-1} & & \\
d_r = & d^{[0]}_{r} & d^{[1]}_{r} & d^{[2]}_{r} & \cdots & d^{[h-2]}_{r} & d^{[h-1]}_{r} & = C(x)
\end{array}
$$

Si può notare come a livello grafico, a ogni passo, viene inserito un nodo, fino ad arrivare a molteplicità $h-1$, infatti all'aumentare della molteplicità diminuiscono le funzioni base $\neq 0$. A conferma di ciò, l'algoritmo viene anche chiamato Knot Insertion.

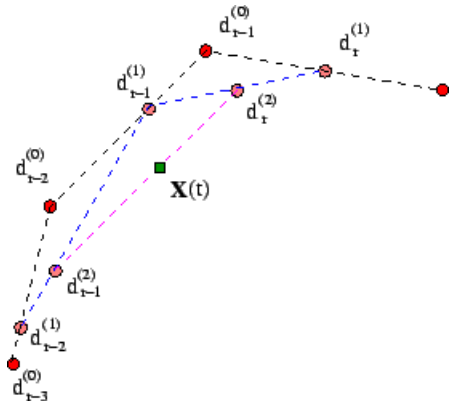


Figura 3.4: De Boor con $h = 4$

3.1.2.2 Curve e parametrizzazione

In matematica, le equazioni parametriche sono equazioni in cui le variabili (indipendenti e dipendenti) sono espresse a loro volta in funzione di uno o più parametri. Le equazioni parametriche sono comunemente utilizzate per esprimere le coordinate dei punti che costituiscono un oggetto geometrico come una curva o una superficie, in questo caso sono chiamate rappresentazioni parametriche o parametrizzazioni dell'oggetto. Ad esempio, le equazioni

$$x(t) = cos(t)$$
$$y(t) = sin(t)$$

formano una rappresentazione parametrica della circonferenza unitaria, dove t è il parametro, $t \in [0, 2\pi]$.
Una curva parametrica in $\mathbb{R}^2$ è un'applicazione:

	$C(t) : I \to \mathbb{R}^2$	
dove	$I = [a, b] \in \mathbb{R}$	
	$x(t), y(t)$	componenti parametriche
	t	parametro
	$C(t)$	parametrizzazione della curva
	$C(a), C(b)$	estremi della curva

Le rappresentazioni parametriche sono generalmente non uniche, in modo che le stesse possano essere espresse da una serie di parametrizzazioni diverse.
Per esempio sia data l'equazione parametrica della circonferenza $C(t) = (cos(2t), sin(2t)), t \in [0, 2\pi]$. La stessa circonferenza si ottiene anche con le funzioni coordinate $C1(t) = (sin(t), cos(t)), t \in [0, 2\pi]$. Le due rappresentazioni parametriche rappresentano il me-

desimo percorso ma valutandole per medesimi valori di t, rappresentano punti diversi sulla curva.
Consideriamo una curva piana in forma parametrica $C(t)$ descritta dall' equazione parametrica $C(t) = (C_x(t), C_y(t)), t \in [a, b]$.
Se una particella si muovesse nel tempo lungo la curva parametrica, ad ogni istante t, la posizione della particella sarebbe $(C_x(t), C_y(t))$.
Nel problema dell'interpolazione tramite spline visto fino a questo momento non si riesce a costruire una curva che non sia una funzione. Proprio per questo motivo verranno usate le curve parametriche in simbiosi con le spline, dove:

$$C(t) = \begin{cases} C_x(t) = \sum x_i N_{i,m}(t) \\ C_y(t) = \sum y_i N_{i,m}(t) \end{cases}$$

Le parametrizzazioni di una curva sono svariate ma in questo caso viene usata una semplice parametrizzazione uniforme:

```
uniformParameterization(float* linSpace, int rows) {
  float step = 1.0 / rows;
  for (int i = 0; i <rows; i++) {
    linSpace[i] = i*step;
  }
}
```

Essa prevede di creare per ogni intervallo della Spline lo stesso numero di nodi, indipendentemente dalla lunghezza dell'intervallo. Una parametrizzazione che, invece, tiene conto di questa particolarità è la parametrizzazione a lunghezza d'arco. Per questo motivo la parametrizzazione a lunghezza d'arco produce risultati migliori ma, in questo progetto di tesi, risulta sufficiente la parametrizzazione uniforme.

3.2 L'approssimazione

Una funzione interpolante passa per i punti dati, una curva approssimante minimizza la distanza tra la curva e i dati stessi.

Siano $f_n(x)$ la funzione che vuole minimizzare l'errore

$y = \{y_1 \quad \dots \quad y_m\}$ l'insieme delle ordinate dei punti $P_i(x_i, y_i)$

Tra i modi più usati per calcolare la distanza in problemi come questo citiamo:

- norma 2:

$$\|y - f_n\|_2 = \sqrt{\sum_{i=1}^{m} (y_i - f_n(x_i))^2}$$

- norma Chebychev:

$$\|y - f_n\|_\infty = \max_{i=1,\dots m} |y_i - f_n(x_i)|$$

Come mostra la figura sottostante minimizzare la defferenza potrebbe comportare all'indesiderata situazione in cui gli errori si eliminano a vicenda. La soluzione è minimizzare la somma dei quadrati di questi.

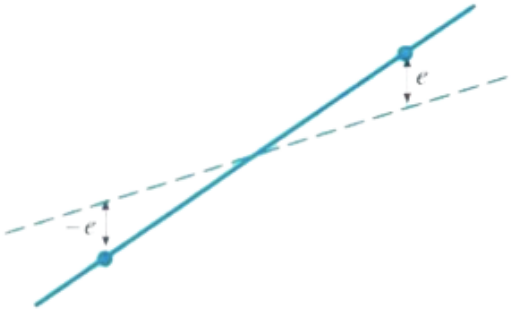


Figura 3.5: La figura mostra come una norma non idonea possa portare a degli errori.

3.2.1 L'approssimazione ai minimi quadrati

$$\begin{aligned}
\text{Assegnati i punti} \quad & P_i(x_i, y_i), \quad i = 1, \dots m \\
\text{e scelte le } n+1 \text{ funzioni base} \quad & \varphi_0, \varphi_1, \dots \varphi_n, \quad (n << m) \\
\text{si vuole costruire} \quad & f_n(x) = a_0\varphi_0 + a_1\varphi_1 + \dots + a_n\varphi_n \\
\text{tale che} \quad & \min_{(a_0,\dots a_n)} S = \min_{(a_0,\dots a_n)} \|y - f_n\|_2^2 = \sum_{i=1}^{m} (y_i - f_n(x_i))
\end{aligned}$$

Ossia determinare i coefficienti $a_i, \quad i = 0, \dots n$ in modo che lo scarto quadratico medio S sia minimo.
In forma matriciale:

$$\begin{aligned}
\text{Sia} \quad & y - f_n(x) = \sum_{i=1}^{m} y_i - \sum_{j=0}^{n} a_j\varphi_j(x_i) \\
\text{dove} \quad & \varphi_0(x_i) = N_{0,n}, \quad \varphi_1 = N_{1,n}(x_i), \dots \qquad \text{sono le funzioni B-Spline} \\
\text{allora} \quad & y - f_n(x) = y - Ha = y \qquad \text{con } H|_{i,j} = \varphi_j(x_i) = N_{j,n}(x_i)
\end{aligned}$$

Si vuole determinare il vettore a che renda minima la norma del vettore errore:

$$\min S = \min_a \|y - Ha\|_2^2$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} - \begin{bmatrix} N_{0,n}(x_1) & N_{1,n}(x_1) & \cdots & N_{n,n}(x_1) \\ N_{0,n}(x_2) & N_{1,n}(x_2) & \cdots & N_{n,n}(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ N_{0,n}(x_m) & N_{1,n}(x_m) & \cdots & N_{n,n}(x_m) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix}$$

Soluzione di un sistema lineare $Ha = y$

- normale se $m = n$ (si riconduce all' interpolazione)
- sovradeterminato se $m > n$ (quindi approsimazione)

3.2.1.1 Metodo delle equazioni normali

Calcoliamo le derivate parziali di S rispetto alle variabili e le poniamo uguali a zero.

$$S = \sum_{i=1}^{m} (y_i - (a_0\varphi_0 + a_1\varphi_1 + \cdots + a_n\varphi_n))^2$$

$$\begin{cases} \frac{dS}{da_0} = 0 \\ \frac{dS}{da_1} = 0 \\ \vdots \\ \frac{dS}{da_n} = 0 \end{cases}$$

Il sistema è detto sistema delle equazioni normali e ha una, e una sola, soluzione.
$f_n(x) = a_0\varphi_0 + a_1\varphi_1 + \cdots + a_n\varphi_n$ è detto funzione di regressione.
Il sistema lineare che ne deriva di ordine n+1 in forma matriciale è:

$$\begin{bmatrix} \sum_{i=1}^m \varphi_1(x_i)\varphi_1(x_i) & \sum_{i=1}^m \varphi_2(x_i)\varphi_1(x_i) & \cdots & \sum_{i=1}^m \varphi_n(x_i)\varphi_1(x_i) \\ \sum_{i=1}^m \varphi_1(x_i)\varphi_2(x_i) & \sum_{i=1}^m \varphi_2(x_i)\varphi_2(x_i) & \cdots & \sum_{i=1}^m \varphi_n(x_i)\varphi_2(x_i) \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^m \varphi_1(x_i)\varphi_n(x_i) & \sum_{i=1}^m \varphi_2(x_i)\varphi_n(x_i) & \cdots & \sum_{i=1}^m \varphi_n(x_i)\varphi_n(x_i) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^m y_i\varphi_1(x_i) \\ \sum_{i=1}^m y_i\varphi_2(x_i) \\ \vdots \\ \sum_{i=1}^m y_i\varphi_n(x_i) \end{bmatrix}$$

Dimostrazione

$$\text{Sia} \quad S = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} y_i - a_j \varphi_j(x_i) \right)^2$$

$$\text{allora} \quad \frac{d}{da_k} \sum_{i=1}^{m} \left(\sum_{j=1}^{n} y_i - a_j \varphi_j(x_i) \right)^2 = 0 \qquad k = 1, \ldots n$$

$$\sum_{i=1}^{m} 2 \left(\sum_{j=1}^{n} y_i - a_j \varphi_j(x_i) \right) \varphi_k(x_i) = 0$$

effettuando uno scambio di sommatorie:

$$2 \sum_{j=1}^{n} a_j \left(\sum_{i=1}^{m} y_i - \varphi_j(x_i) \right) \varphi_k(x_i) = 0$$

$$2 \sum_{i=1}^{m} y_i \varphi_k(x_i) - 2 \sum_{j=1}^{n} a_j \left(\sum_{i=1}^{m} \varphi_j(x_i) \varphi_k(x_i) \right) = 0$$

$$\sum_{j=1}^{n} \left(\sum_{i=1}^{m} y_i \varphi_k(x_i) \right) = \sum_{j=1}^{n} a_j \left(\sum_{i=1}^{m} \varphi_j(x_i) \varphi_k(x_i) \right)$$

Il costo computazionale del metodo delle equazioni normali per risolvere il problema dell'approssimazione ai minimi quadrati è dato dalla risoluzione di un sistema lineare. L'approssimazione di grado n richiede di risolvere un sistema lineare quadrato di dimensione $(n+1) \times (n+1)$. In generale possiamo riscrivere il sistema delle equazioni normali:

$$\min_a S = \|y - Ha\|_2^2 \Leftrightarrow H^T H a = H^T y$$

$$
\begin{array}{lll}
\text{Dimostrazione} & S(a) & = \|y - Ha\|_2^2 = \\
 & & = (y - Ha)^T (y - Ha) = \\
 & & = y^T y - y^T Ha - a^T H^T y + a^T H^T Ha = \\
 & & = y^T y - 2y^T Ha + a^T H^T Ha = \\
\text{ponendo} & \bigtriangledown S(a) & = 0 \\
 & \Rightarrow -2y^T H + 2H^T Ha & = 0 \\
\text{e quindi} & \Longrightarrow & H^T Ha = H^T y
\end{array}
$$

Se la matrice H ha rango massimo, $H^T H$ è non singolare, quindi il problema ha un'unica soluzione.
$2H^T H$ è la matrice Hessiana. $H^T H$ e' definita positiva se e solo se la matrice H ha rango massimo (quindi l'unica soluzione a è il minimo di S).

Teorema

Il problema LS (Least Square) ammette sempre soluzione.
La soluzione è unica se e solo se la matrice H ha rango massimo (le colonne di H sono linearmente indipendenti), se invece le colonne di H sono linearmente dipendenti allora il problema ha infinite soluzioni.

3.2.1.2 Metodi per la risoluzione del sistema

La soluzione del sistema lineare può essere eseguita con uno dei metodi diretti a proprio piacimento. Di seguito vediamo il metodo Cholesky che è uno dei metodi derivati da quello di Gauss per fatorizzazione LU; per tal motivo prima esamineremo quest'ultimo.

Fattorizzazione di Gauss

L'obiettivo dei metodi diretti è risolvere un sistema del tipo $Ax = b$.
Se fosse possibile decomporre $A = LU$, A non singolare L triangolare inferiore, U triangolare superiore:

$$LUx = b \quad \Leftrightarrow \quad \begin{cases} Ly = b \\ Ux = y \end{cases}$$

Questa esigenza nasce dal fatto che un sistema in cui la matrice è triangolare superiore o inferiore può essere risolto con rispettivamente la tecnica della sostituzione all'indietro o avanti.
Un esempio di forward substitution con matrice triangolare inferiore:

$$\begin{cases} a_{11}x_1 & & & & & & = b_1 \\ a_{21}x_1 & +a_{22}x_2 & & & & & = b_2 \\ \vdots & \vdots & \ddots & & & & = \vdots \\ a_{i1}x_1 & +a_{i2}x_2 & \cdots & a_{ij}x_j & & & = b_i \\ \vdots & \vdots & \vdots & \vdots & \ddots & & = \vdots \\ a_{n1}x_1 & +a_{n2}x_2 & \cdots & \cdots & \cdots & a_{nn}x_n & = b_n \end{cases}$$

$$\begin{cases} x_1 = \frac{b_1}{a_{11}} \\ x_i = \frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j}{a_{ii}} \end{cases}$$

Costo computazionale $\sum_{i=1}^{n} i \approx \frac{n(n+1)}{2} \implies O(\frac{n^2}{2})$

Si vogliono applicare $n-1$ operazioni elementari di Gauss $L_{n-1}L_{n-2}\dots L_2L_1$ alla matrice $[A|b]$ in modo tale da trasformarla in $[U|y]$; dove $L = [L_{n-1}L_{n-2}\dots L_2L_1]^{-1}$ triangolare inferiore e $U = [L_{n-1}L_{n-2}\dots L_2L_1]A$ triangolare superiore.

step 1	$L_1[A\|b]$
step 2	$L_2L_1[A\|b]$
$\vdots$	$\vdots$
step $n-1$	$L_{n-1}L_{n-2}\dots L_2L_1[A\|b]$

$$L_{n-1}L_{n-2}\dots L_2L_1[A|b] = [A_n|b_n] = [U|y]$$
$$[A|b] = [L_{n-1}L_{n-2}\dots L_2L_1]^{-1}[A_n|b_n] = L[U|y]$$

Alla fine non solo abbiamo ottenuto la fattorizzazione $A = LU$ della matrice ma abbiamo anche risolto il sistema $Ly = b$. Il metodo di Gauss consiste

- nella fattorizzazione LU della matrice $[A|b]$(che risolve anche $Ly = b$) e
- nella risoluzione con sostituzione all'indietro del sistema triangolare:

$$Ux = y$$

Un esempio:

$$A_1 = A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 4 & 2 & 1 \end{bmatrix} \qquad b_1 = b = \begin{bmatrix} 4 \\ 6 \\ 9 \end{bmatrix}$$

$$L_1 = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -4 & 0 & 1 \end{bmatrix} \quad A_2 = L_1 A_1 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & -2 & 0 \\ 0 & -2 & -3 \end{bmatrix} \quad b_2 = L_1 b_1 = \begin{bmatrix} 4 \\ 2 \\ -7 \end{bmatrix}$$

$$L_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \quad A_3 = L_2 A_2 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & -2 & 0 \\ 0 & 0 & -3 \end{bmatrix} \quad b_3 = L_2 b_2 = \begin{bmatrix} 4 \\ 4 \\ -9 \end{bmatrix}$$

$$L = [L_2 L_1]^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 4 & 1 & 1 \end{bmatrix} \quad U = A_3 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & -2 & 0 \\ 0 & 0 & -3 \end{bmatrix} \qquad y = b_3 = \begin{bmatrix} 4 \\ 2 \\ -9 \end{bmatrix}$$

Risolviamo quindi il sistema $Ux = y$

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & -2 & 0 \\ 0 & 0 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \\ -9 \end{bmatrix}$$

$x_3 = 3$

$x_2 = -1$

$x_1 = 2$

Le matrici L e U vengono costruite secondo il seguente schema:

$$L = \begin{bmatrix} 1 & 0 & \cdots & \cdots & \cdots & 0 \\ m_{21} & 1 & 0 & \cdots & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots & 0 \\ m_{i1} & m_{i2} & \ddots & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & 0 \\ m_{n1} & m_{n2} & \cdots & \cdots & m_{nn-1} & 1 \end{bmatrix} \quad U = \begin{bmatrix} a_{11} & a_{12} & \cdots & \cdots & \cdots & a_{1n} \\ 0 & a_{22} & a_{23} & \cdots & \cdots & a_{2n} \\ 0 & 0 & \ddots & \ddots & \vdots & \vdots \\ 0 & \ddots & 0 & a_{ij} & \cdots & a_{in} \\ 0 & \ddots & \ddots & 0 & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 0 & a_{nn} \end{bmatrix}$$

Gli elementi di U sulla diagonale principale si chiamano perni e devono essere diversi da zero affichè si abbia fattorizzazione LU.
Gli elementi di L vengono costruiti in maniera tale da azzerare il corrispettivo elemento in A.

Teorema di Cholesky

Se A è una matrice simmetrica definita positiva allora esiste ed è unica una matrice R triangolare inferiore, con elementi positivi sulla diagonale principale tale che:

$$A = RR^T$$

Calcolo della soluzione del sistema lineare $Ax = b$:

$$RR^T x = b \quad \Leftrightarrow \quad \begin{cases} Ry = b \\ R^T x = y \end{cases}$$

Partendo dal Teorema ricaviamo l'algoritmo:

$$A = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{n1} \\ a_{21} & a_{22} & \cdots & a_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} r_{11} & 0 & \cdots & 0 \\ r_{21} & r_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ r_{n1} & r_{n2} & \cdots & r_{nn} \end{bmatrix} \begin{bmatrix} r_{11} & r_{21} & \cdots & r_{n1} \\ 0 & r_{22} & \cdots & a_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

quindi per la prima colonna

$$a_{11} = r_{11}r_{11} \implies r_{11} = \sqrt{a_{11}}$$

$$a_{i1} = r_{i1}r_{11} \implies r_{i1} = \frac{a_{i1}}{a_{11}} \quad i = 2, \dots n$$

per la seconda colonna

$$a_{22} = r_{21}^2 + r_{22}^2 \implies r_{22} = \sqrt{a_{22} - r_{21}^2}$$

$$a_{j2} = r_{j1}r_{21} + r_{j2}r_{22} \implies r_{j2} = \frac{a_{j2} - r_{j1}r_{21}}{r_{22}} \quad j = 3, \dots n$$

per la j-esima colonna

$$a_{jj} = r_{j1}^2 + r_{j2}^2 + \cdots + r_{jj}^2 \implies r_{jj} = \sqrt{a_{jj} - r_{j1}^2 - r_{j2}^2 - \cdots - r_{jj-1}^2}$$

$$a_{ij} = r_{i1}r_{ji} + r_{i2}r_{j2} + \cdots + r_{ij}r_{jj}$$

$$\implies r_{ij} = \frac{a_{ij} - r_{i1}r_{ji} - r_{i2}r_{j2} - \cdots - r_{ij-1}r_{jj-1}}{r_{jj}} \quad i = j+1, \dots n$$

quindi

1: **function** Cholesky()
2: **for** $j = 1$ to n **do**
3: $r_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} r_{jk}^2}$
4: **for** $i = j + 1$ to n **do**
5: $r_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} r_{ik}r_{jk}}{r_{jj}}$
6: **end for**
7: **end for**
8: **end function**

3.3 Superficie spline generata per rivoluzione

Come spiegato nelle sezioni precedenti una curva spline è una curva parametrica della forma:

$$C(t) = \sum_i P_i N_i(t) \quad P_i \equiv \begin{pmatrix} x_i \\ y_i \end{pmatrix} \quad i = 0, \ldots n$$

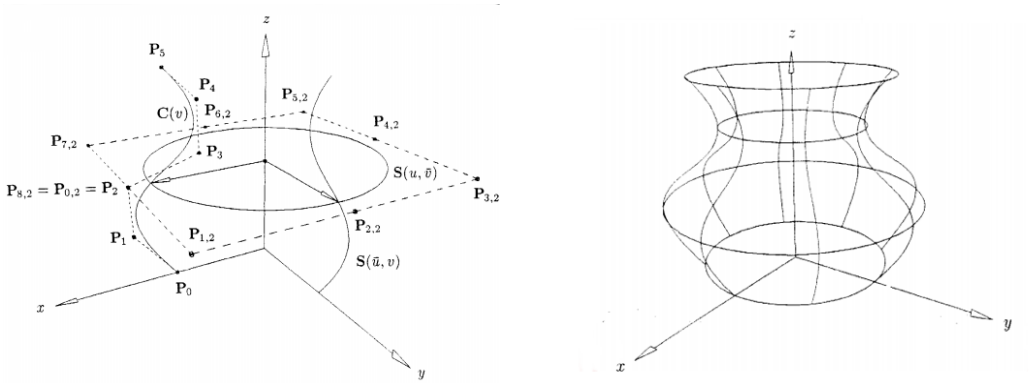
Una superficie spline è il luogo dei punti visitati da una curva spline (detta profilo), i cui vertici di controllo si muovono descrivendo un'altra curva spline detta traettoria. Quindi, ciascuno dei punti P_i è a sua volta una funzione parametrica.

$$C(t) = \sum_i P_i N_i(t)$$
$$\text{dove} \quad P_i = P_i(s) = \sum_j P_{ij} N_j(s)$$

Dunque la superficie spline, o di spin, si può scrivere come:

$$C(t,s) = \sum_i \sum_j P_{i,j} N_j(s) N_i(t)$$
$$\text{ma definendo} \quad N_{i,j}(t,s) = N_i(t) N_j(s)$$
$$\text{diventa} \quad C(t,s) = \sum_i \sum_j c_{i,j} N_{i,j}(t,s)$$

Se la curva traettoria è una circonferenza, allora la superficie è ottenuta per rivoluzione della curva profilo lungo la circonferenza.
Infatti, nel caso in cui l'asse di rotazione sia quello delle ordinate basterebbe disegnare la circonferenza nel piano XZ e parametrizzare i P_i. I punti nella superficie risultante vedranno la coordinata y pari a quella della spline profilo e le coordinate x e y pari a quelle della curva trettoria. Queste saranno rispettivamente il coseno e il seno della circonferenza.



(a) curva profilo **(b)** superficie risultante

Figura 3.6: Un esempio di rivoluzione intorno all'asse y

Di seguito uno speudo codice di esempio.

function SbyC(*Spline*, *splinePval*, *circlePval*, *Mesh*)

▷ Spline - array contenente la curva spline

▷ splinePval - numero punti di valutazione della spline

▷ circlePval - numero punti di valutazioni della circonferenza

▷ Mesh - array vuoto da riempire con il risultato

    for $i = 1$ to *splinePval* **do**
        for $j = 1$ to *circlePval* **do**
            $Mesh[i * circlePval + j].y = Spline[i].j$
            $angle = (2 * \phi / circlePval) * (circlePval * j)$
            $Mesh[i * circlePval + j].x = Spline[i].j * cos(angle)$
            $Mesh[i * circlePval + j].z = Spline[i].j * sen(angle)$
        end for
    end for
end function

3.4 La triangolarizzazione

Nella geometria computazionale, la triangolazione poligonale è la decomposizione di un'area poligonale (poligono semplice) P in un insieme di triangoli, cioè trovare un insieme di triangoli non intersecati tra loro che uniti formino P.
Le triangolazioni possono essere considerate come casi speciali di suddivisione di piani. Quando non ci sono fori o punti aggiuntivi, le triangolazioni formano un grafo planare esterno massimale.
Nella teoria dei grafi si definisce grafo planare un grafo che può essere raffigurato in un piano in modo che non si abbiano archi che si intersecano.
Un grafo è chiamato planare massimale se è planare, e se con l'aggiunta di un qualunque nuovo spigolo tra due vertici presenti fa cadere la planarità.
Un grafo è chiamato planare esterno se è immerso in un piano in modo che i vertici giacciono su una circonferenza e gli archi si trovano all'interno del corrispondente cerchio e non si intersecano. In maniera equivalente, c'è una faccia che in una opportuna raffigurazione include ogni vertice.

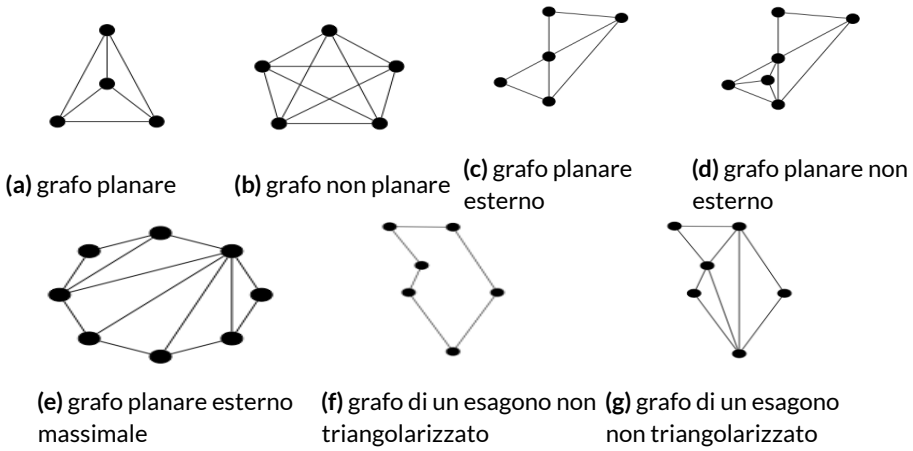


Figura 3.7: Esempi di grafi descritti precedentemente

La triangolarizzazione di una mesh 3D viene effettuata tramite una proiezione di essa su un piano XY, proprio come se essa venisse presa e stesa. A questo punto possono essere applicati uno dei seguenti algoritmi.

3.4.1 Triangolarizzazione a ventaglio

La triangolazione a ventaglio è un metodo semplice per ottenere una triangolazione poligonale. Si sceglie un vertice e si suddivide il poligono rilasciando diagonali di quel vertice verso gli altri. Non tutti i poligoni possono essere triangolarizzati in questo modo, ma il metodo è utilizzato in modo particolare per triangolare poligoni convessi.
Un poligono convesso è un poligono semplice (non autointerrotto) in cui nessun segmento di linea tra due punti al limite va mai fuori dal poligono. Allo stesso modo, è un semplice poligono il cui interno è un insieme convesso. In un poligono convesso, tutti gli angoli interni sono inferiori o uguali a 180 gradi, mentre in un poligono strettamente convesso tutti gli angoli interni sono strettamente inferiori a 180 gradi.
Nella geometria convessa, un insieme convesso è un sottoinsieme di uno spazio affine chiuso in combinazioni convesse. Più precisamente, in uno spazio euclideo, una regione convessa è una regione in cui, per ogni coppia di punti all'interno della regione, ogni punto del segmento di linea retta che unisce una coppia di punti è anche all'interno della regione.

(a) poligono concavo non triangolarizzabile con questa tecnica

(b) poligono convesso triangolarizzato con questa tecnica

Figura 3.8: Poligoni concavo e convesso a confronto

3.4.2 Triangolazione di Delaunay

Nella matematica e nella geometria computazionale, una triangolazione Delaunay, per un dato insieme P di punti discreti in un piano, è una triangolazione $D_T(P)$ tale che nessun punto in P è all'interno del circumcerchio di un qualsiasi triangolo in $D_T(P)$. Le triangolazioni di Delaunay massimizzano l'angolo minimo di tutti gli angoli dei triangoli nella triangolazione; tendono ad evitare i triangoli stretti. Tutti i triangoli sono ciclici; cioè ogni triangolo ha un cerchio circoscritto.
Ciò può essere dimostrato perché l'equazione generale di un cerchio con centro (a, b) e raggio r nel sistema di coordinate cartesiane è:

$$(x-a)^2 + (y-b)^2 = r^2$$

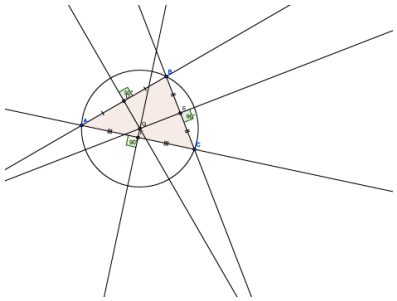


Figura 3.9: Creazione di un circumcerchio di un triangolo.

Un triangolo stretto è un triangolo con uno o due angoli estremamente acuti, quindi una forma lunga / sottile, che ha proprietà indesiderate durante alcuni processi di interpolazione o rasterizzazione. La triangolazione prende il nome da Boris Delaunay per il suo lavoro su questo tema dal 1934. Per una serie di punti sulla stessa linea non esiste una triangolazione Delaunay (la nozione di triangolazione è degenere in questo caso). Per quattro o più punti sullo stesso cerchio (ad esempio, i vertici di un rettangolo) la

triangolazione di Delaunay non è unica: ognuna delle due possibili triangolazioni che dividono il quadrangolo in due triangoli soddisfa la ”condizione di Delaunay”, ovvero il requisito che iI circumcerchio di tutti i triangoli abbiano interni vuoti. Considerando le sfere circoscritte, la nozione di triangolazione di Delaunay si estende a tre e dimensioni superiori. Le generalizzazioni sono possibili a metriche diverse dalla distanza Euclidea. Tuttavia, in questi casi, una triangolazione di Delaunay non è garantita o non sia unica.

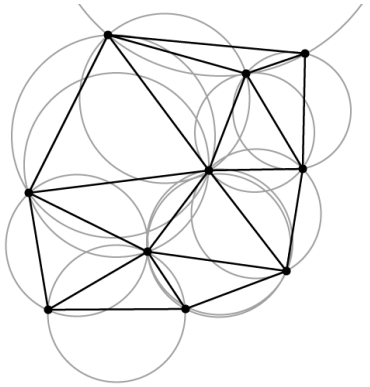


Figura 3.10: Esempio dell’algoritmo di Delaunay.

3.4.3 Triangolazione Greedy

La triangolazione Greedy è un metodo per calcolare una triangolazione poligonale o una triangolazione a punti, usando uno schema Greedy, che aggiunge i bordi uno alla volta alla soluzione in ordine rigoroso crescente per lunghezza, a condizione che un bordo non possa tagliare un bordo precedentemente inserito.

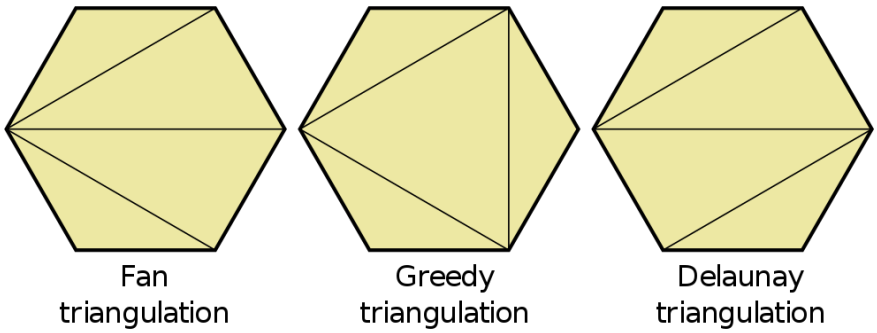


Figura 3.11: Esempio degli algoritmi di triangolarizzazione a confronto.

4

Il progetto delle curve

Per introdurre al meglio questo modulo vediamo nel dettaglio le funzionalità che offre e come. Entrati nella modalità di creazione mesh tramite curve, l'applicazione, una volta riconosciuta l'immagine target, grafica sul tavolo da lavoro un bottone.

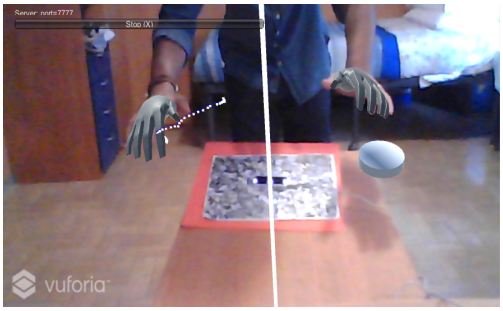
(a) Server non avviato

(b) Server avviato

Figura 4.1: Riconoscimento del target e disegno bottone

- Alla pressione di tale bottone viene graficato l'asse delle ordinate passante per il Leap. Da questo momento in poi l'applicativo tiene traccia di tutti i movimenti della mano destra e memorizza lo storico dei punti dell'indice.

(a) Riconoscimento mano destra

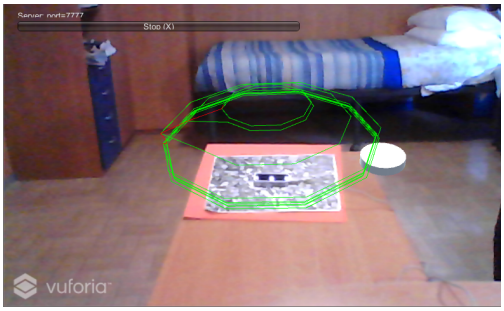


(b) Riconoscimento mano sinistra che preme il bottone

Figura 4.2: Tracciamento dell'indice destro

- Al momento del rilascio del bottone il software, dopo aver distrutto l'asse delle ordinate e i punti di interpolazione, esegue tre azioni fondamentali:
  - crea una spline tramite i metodi di approssimazione precedentemente visionati;
  - ruota la spline intorno all'asse delle ordinate precedentemente graficato;
  - triangolarizza la mesh ottenuta tramite due metodi di triangolarizzazione.

Figura 4.3: Creazione della spline al rilascio del bottone



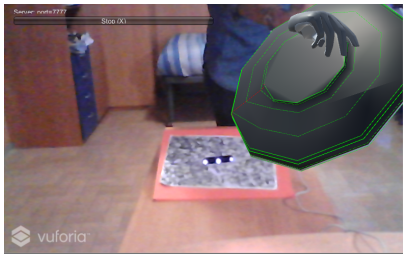
(a) Creazione Mesh

(b) Triangolarizzazione Mesh

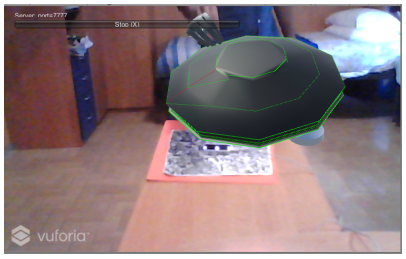
Figura 4.4: Step per la visualizzazione della Mesh

- Da questo momento in poi la mesh creata viene visualizzata sul tavolo da lavoro e può essere manipolata come si vuole.

(a) Scalatura della Mesh



(b) Rotazione della Mesh



(c) Traslazione della Mesh

Figura 4.5: Operazioni sulla Mesh

Per adempire alle seguenti specifiche si sono utilizzate diverse tecniche e tecnologie, di seguito vediamo:

- la macchina a stati finiti per il controllo del flusso, ovvero le azioni che deve compiere il programma a seguito di ogni specifico input;
- l'architettura del Leap Motion per il tracciamento delle mani;
- la GNU Scientific Library e il suo wrapping per Unity per l'uso di algoritmi numerici efficenti;
- l'implementazione della creazione della spline, la rivoluzione di essa e la triangolarizzazione per la visualizzazione delle mesh.

4.1 Finite State Machine

Una Finite State Machine (FSM), o macchina a stati finiti, è un modello matematico di calcolo. È una macchina astratta che, tra il numero finito di stati che può avere, in un dato momento, può essere esattamente in un solo stato. La FSM può cambiare da uno stato all'altro in risposta ad alcuni input esterni; questa modifica prende il nome di transizione. Una FSM è definita da un elenco dei suoi stati, il suo stato iniziale e le condizioni di attivazione per ogni transizione.
Il comportamento delle macchine a stati finiti può essere osservato in molti dispositivi della società moderna; quelli che eseguono una sequenza di azioni predeterminate in funzione di una sequenza di eventi. Alcuni esempi sono le macchinette automatiche, che dispensano i prodotti quando viene inserita la corretta somma di monete, gli ascensori, la cui sequenza di arresti è determinata dai piani richiesti, dai semafori, che cambiano la sequenza quando le automobili sono in attesa e le cassaforti che richiedono l'inserimento dei numeri di combinazione nell'ordine appropriato.
Un esempio di un semplice meccanismo che può essere modellato è un tornello. Utilizzato per controllare l'accesso alla metropolitana o nelle code dei parco divertimenti, è un cancello con tre bracci rotanti all'altezza della vita. Inizialmente le braccia sono bloccate, bloccando l'ingresso, impedendo ai passeggeri di passare. L'inserimento di una moneta o di un gettone in una fessura del tornello sblocca il meccanismo, consentendo a un singolo cliente di spingere. Dopo il passaggio del cliente, le braccia vengono bloccate nuovamente fino a quando non viene inserita un'altra moneta.
Considerato come una macchina a stati finiti, il tornello dispone di due stati possibili: bloccato e sbloccato. Ci sono due ingressi possibili che influenzano il suo stato: mettere una moneta nello slot (Coin) e spingere il braccio (Push). Nello stato bloccato, spingendo il braccio meccanico non si ha alcun effetto; non importa quante volte la pressione viene data, la macchina rimane nello stato bloccato. Mettere una moneta, sposta lo stato

della macchina da bloccato a sbloccato. Nello stato sbloccato, l'inserimento di monete non ha alcun effetto; vale a dire, l'aggiunta di monete supplementari non cambia lo stato. Tuttavia, un cliente che passa attraverso il braccio meccanico, fornendo una spinta come input, porta lo stato di nuovo a bloccato.

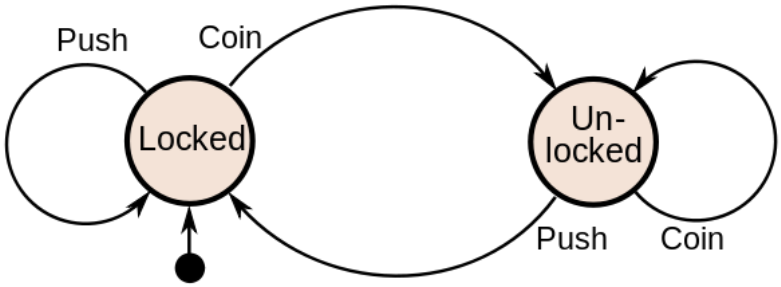
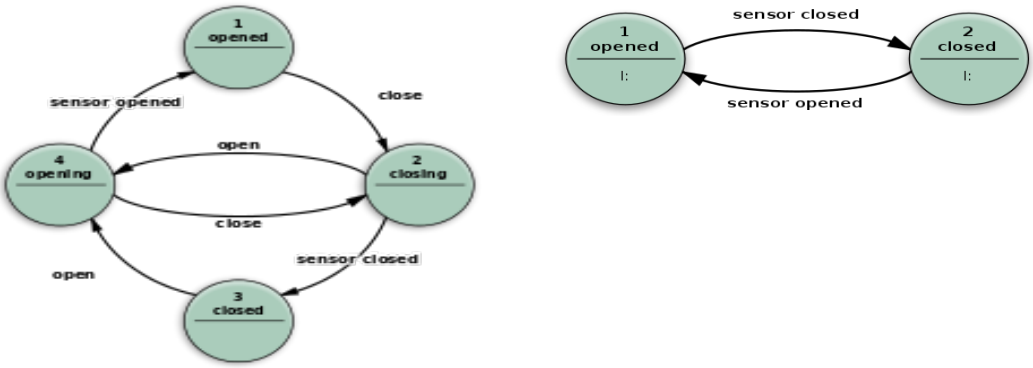


Figura 4.6: FSM di un tornello

Esistono due tipologie di FSM:

- La macchina di Moore: utilizza solo le azioni di input, cioè l'output dipende solo dallo stato. Il vantaggio del modello Moore è una semplificazione del comportamento. Considerate una porta ascensore. L'FSM riconosce due comandi: sensor_opened e sensor_closed, che innescano le modifiche dello stato. L'azione di input in stato opening fa iniziare un motore che apre la porta, l'azione di input in stato closing fa iniziare un motore nell'altra direzione che chiude la porta. Gli stati open e close arrestano il motore quando sono completamente aperti o chiusi. Queste segnalano la situazione: la porta è aperta o la porta è chiusa;
- La macchina di Mealey: utilizza anche le azioni di input, cioè l'output dipende dall'ingresso e dallo stato. L'uso di una Mealy FSM porta spesso a una riduzione del numero di stati. L'esempio in figura mostra un Mealy FSM che implementa lo stesso comportamento come nell'esempio di Moore. Ci sono due azioni di ingresso: avviare il motore per chiudere la porta se arriva sensor_closed e avviare il motore nell'altra direzione per aprire la porta se arriva sensor_opened. Gli stati intermedi opening e closing non vengono mostrati.



(a) Macchina di Moore **(b)** Macchine di Mealey

Figura 4.7: Un esempio delle FSM a confronto

La macchina a stati finiti che si è sviluppata per questo progetto è una macchina di Mealey che comprende due stati:

- NONE;
- HOLDSTAY.

La macchina parte con stato NONE, il che comporta una inattività da parte della macchina. Essa aspetta un'evento di input, la pressione del bottone, affinchè possa passare a HOLDSTAY. Quando il bottone viene premuto un booleano viene settato a true e si può avviare la procedura di transizione myHoldOn(). Lo stato passa a HOLDSTAY dove, in ogni frame, nella procedura myHoldStay(), viene effettuato il tracciamento delle mani. Quando avverrà il rilascio del bottone la callback incaricata setterà il booleano a false e di conseguenza verrà avviata la procedura myHoldOut(). Questa si occuperà di creare la spline, la mesh e visualizzarla a seguito della triangolarizzazione.

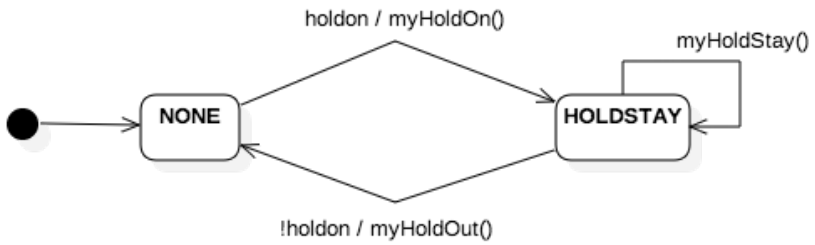


Figura 4.8: FSM del progetto

```
switch(state)
{
    case states.NONE:
       if (CurvesManager.hold)
       {
           myHoldon();
           CurvesManager.state = states.HOLDSTAY;
       }
       break;
    case states.HOLDSTAY:
       myHoldstay();
       if (!CurvesManager.hold)
       {
           myHoldout();
           CurvesManager.state = states.NONE;
        }
        break;
}
```

4.2 Architettura Leap Motion

Il software Leap Motion funge da servizio (in Windows) o daemon (su Mac e Linux). Il software si collega al dispositivo Leap Motion Controller tramite il bus USB. Le applicazioni abilitate accedono al servizio Leap Motion per ricevere i dati di monitoraggio del movimento. Orion (Leap Motion SDK) offre due varietà di API per ottenere i dati: un'interfaccia nativa e un'interfaccia WebSocket. Queste API consentono di creare applicazioni abilitate al Leap in più linguaggi di programmazione. L'interfaccia nativa è una libreria dinamica che è possibile utilizzare per creare nuove applicazioni desktop/-mobile. L'interfaccia WebSocket e la libreria client di JavaScript consentono di creare applicazioni Web.

4.2.1 Accesso tramite libreria dinamica

L'interfaccia di applicazione nativa viene fornita tramite caricamento di una libreria dinamica. Questa libreria si collega al servizio Leap Motion e fornisce i dati di monitoraggio all'applicazione. Ci si può collegare direttamente alla libreria direttamente dalle applicazioni C++ e Objective C oppure tramite uno dei binding languages forniti per Java, C# e Python. In letteratura ci si riferisce a un binding languages da un linguaggio di programmazione a una libreria o un servizio di sistema operativo per intendere un'interfaccia di programmazione (API) che fornisce il glue code per utilizzare tale libreria o servizio in un determinato linguaggio di programmazione. Intendiamo per glue code il codice sorgente che serve solo ad adattare diverse parti del codice che altrimenti sarebbero incompatibili; un esempio ne è la Java Native Interface (JNI), quando si mappano gli oggetti di un database utilizzando l'object-relational mapping. L'architettura in figura 4.4 mostra come una applicazione che utilizzi Leap Motion possa utilizzare le sue librerie dinamiche per connettersi al demone in esecuzione e ricevere informazioni:

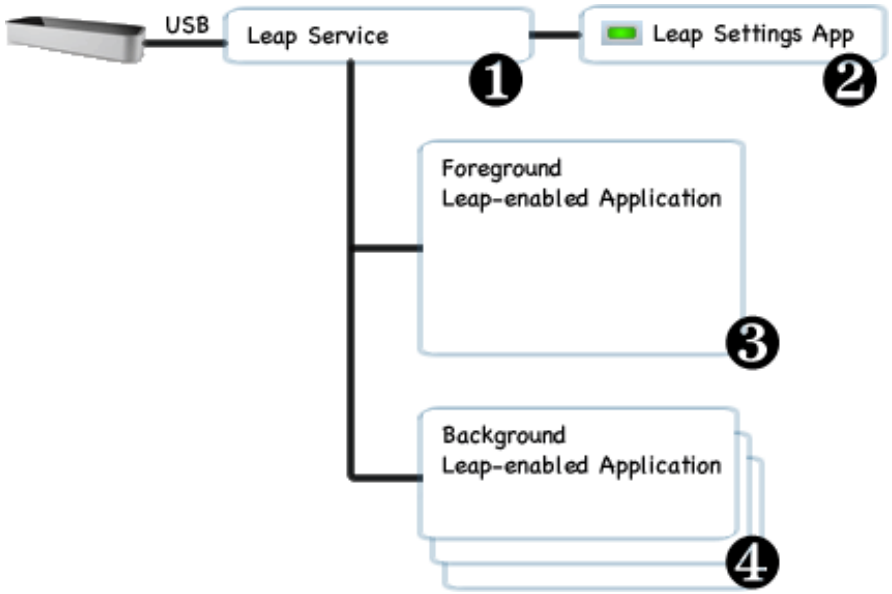


Figura 4.9: Architettura del Leap con libreria dinamica

1. Il servizio Leap Motion riceve dati dal controllore sul bus USB. Processa queste informazioni e le invia alle applicazioni abilitate Leap. Per impostazione predefinita, il servizio invia solo i dati di monitoraggio all'applicazione in primo piano.

2. L'applicazione Leap Motion viene eseguita separatamente dal servizio e consente all'utente di configurare l'installazione. L'applicazione è un applet del Pannello di controllo su Windows e un'applicazione sulla Barra del Menu su Mac OS X.

3. L'applicazione in primo piano Leap-enabled riceve i dati di tracking del movimento dal servizio. Un'applicazione Leap-enabled può connettersi al servizio Leap Motion utilizzando la libreria nativa direttamente (C ++ e Objective-C) o tramite un binding language tra i linguaggi di programmazione disponibili (Java, C# e Python).

4. Quando un'applicazione abilitata a Leap perde lil diritto di esecuzione da parte del sistema operativo, il servizio Leap Motion smette di inviare dati su di esso. Le applicazioni destinate a lavorare in background possono richiedere l'autorizzazione a ricevere i dati anche in background.

4.2.2 Accesso tramite Web Socket

Il servizio Leap Motion esegue un server WebSocket nel dominio localhost alla porta 6437. L'interfaccia WebSocket fornisce i dati di monitoraggio sotto forma di messaggi JSON. È disponibile una libreria client JavaScript che risponde ai messaggi JSON e presenta i dati di monitoraggio come oggetti JavaScript normali.
Come prima mostriamo l'architettura:

1. Il servizio Leap Motion fornisce un server WebSocket che ascolta su http://127.0.0.1:6437.
2. Il pannello di controllo Leap Motion consente agli utenti finali di abilitare o disattivare il server WebSocket.
3. Il server invia dati di monitoraggio sotto forma di messaggi JSON. Un'applicazione può inviare i messaggi di configurazione al server.
4. La libreria JavaScript del client leap.js dovrebbe essere utilizzata nelle applicazioni web. La libreria stabilisce la connessione al server e risponde ai messaggi JSON. L'API presentata dalla libreria JavaScript è simile in filosofia e struttura all'API nativa.

Questa interfaccia è destinata principalmente alle applicazioni web, ma può essere utilizzata da qualsiasi applicazione che possa stabilire una connessione WebSocket. Il server è conforme a RFC6455.

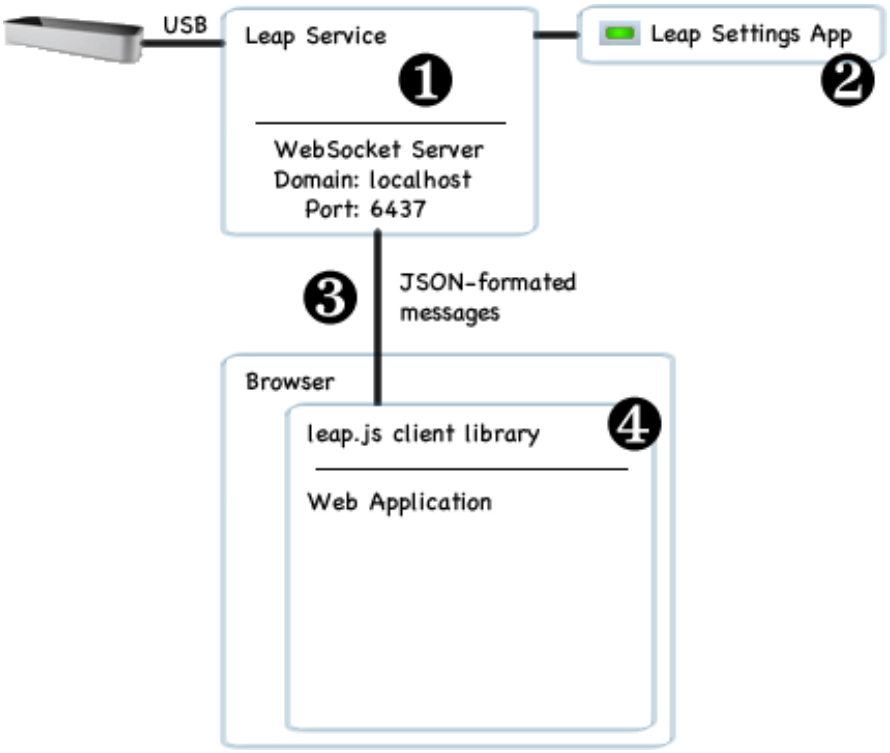


Figura 4.10: Architettura del Leap con WebSocket

4.2.3 API per il tracciamento

Le API per il Leap Motion presentano i dati di monitoraggio come una serie di snapshot chiamati Frame. Ogni Frame di dati contiene le posizioni misurate su ciascuna entità rilevata in quello snapshot. Ogni oggetto Frame contiene un'istantaneo snapshot della scena registrata dal controllore Leap Motion. Le mani, le dita e gli strumenti sono le entità fisiche fondamentali tracciate dal sistema.

E' possibile ottenere un oggetto Frame che contiene i dati di tracciamento da un oggetto Controller. È possibile ottenere un frame ogni volta che l'applicazione è pronta per elaborarla usando il metodo frame () della classe Controller. La classe Frame definisce diverse funzioni che consentono l'accesso ai dati nel fotogramma. Ad esempio, il seguente codice illustra come ottenere gli oggetti di base tracciati dal sistema Leap:

```
Controller controller = new Controller();
if(controller.isConnected())
{
  Frame frame = controller.frame();
  HandList hands = frame.hands();
  PointableList pointables = frame.pointables();
  FingerList fingers = frame.fingers();
  ToolList tools = frame.tools();
}
```

Il polling dell'oggetto Controller per i Frame è la strategia più semplice e spesso migliore quando l'applicazione ha una frequenza di frame naturale. Basta chiamare la funzione frame() sul Controller quando l'applicazione è pronta per elaborare un frame di dati.

Quando si utilizza il polling, è possibile che si otterrà lo stesso frame due volte in una riga (se la frequenza del frame applicativo supera il tasso di frame Leap) o salta un frame (se la frequenza di frame Leap supera la velocità del frame applicativo). In molti casi, i frame mancanti o duplicati non sono importanti. Ad esempio, quando si sposta un oggetto sullo schermo in risposta al movimento della mano, il movimento è comunque fluido, anche senza i frame mancanti.

Come si può notare dalla macchina a stati precedentemente spiegata, nello stato HOLSTAY incaricato del tracciamento delle mani effettua la procedura myHoldStay(). Unity gestisce i GameObject come script con due funzioni base: Start() e Update(). In questo modo quando la FSM è su HOLDSTAY si potrà effettuare il tracciamento tramite polling:

```
void myHoldstay()
{
    processFrame(controller.getFrame());
}

void processFrame(Frame frame)
{
   if (frame != null)
   {
       foreach (Hand hand in frame.Hands)
       {
            if (hand.IsRight)
            {
                setTip(hand.GetIndex().TipPosition.ToVector3());
            }
        }
    }
}
```

4.3 GNU Scientific Library

La GNU Scientific Library (GSL) è una raccolta di routine per l'elaborazione numerica. Le routine sono state scritte da zero in C e presentano una moderna Application Program Interface (API) per i programmatori C, consentendo ai wrapper di essere scritti per lingue di altissimo livello. Il codice sorgente è distribuito sotto la GNU General Public License.
La libreria copre un'ampia gamma di argomenti nel calcolo numerico. Le routine sono disponibili per le seguenti aree:

Complex Numbers	Roots of Polynomials	Special Functions
Vectors and Matrices	Permutations	Combinations
Sorting	BLAS Support	Linear Algebra
CBLAS Library	Fast Fourier Transforms	Eigensystems
Random Numbers	Quadrature	Random Distributions
Quasi-Random Sequences	Histograms	Statistics
Monte Carlo Integration	N-Tuples	Differential Equations
Simulated Annealing	Numerical Differentiation	Interpolation
Series Acceleration	Chebyshev Approximations	Root-Finding
Discrete Hankel Transforms	Least-Squares Fitting	Minimization
IEEE Floating-Point	Physical Constants	Basis Splines
Wavelets	Sparse BLAS Support	Sparse Linear Algebra

Figura 4.11: Aree coperte dalla libreria.

L'uso di queste routine è descritto nel manuale. Ogni capitolo fornisce definizioni dettagliate delle funzioni, seguite da programmi di esempio e riferimenti agli articoli sui quali sono basati gli algoritmi.

4.3.1 Algebra Lineare

La libreria fornisce operazioni di algebra lineare che operano direttamente sugli oggetti gsl_vector e gsl_matrix. Queste routine utilizzano gli algoritmi standard da Matrix Computations di Golub & Van Loan con le chiamate BLAS di livello-1 e livello-2 per l'efficienza. I BLAS (Basic Subprograms Linear Algebra) sono routine che forniscono elementi costruttivi standard per l'esecuzione di operazioni di base e di matrice. Il Livello 1 BLAS svolge operazioni scalari, vettoriali e vettoriali vettoriali, il livello 2 BLAS esegue operazioni di matrice-vettore e la BLAS Livello 3 svolge operazioni matrici-matrici. Poiché i BLAS sono efficienti, portatili e ampiamente disponibili, sono comunemente utilizzati per lo sviluppo di software di algebra lineare di alta qualità. Di seguito alcuni comandi usati che meritano più attenzione:

```
gsl_matrix * A
```

La struttura gsl_matrix contiene sei componenti, le due dimensioni della matrice, una dimensione fisica, un puntatore a memoria, i dati, un puntatore al blocco di proprietà del blocco di matrice, se presente, e un flag di proprietà. La struttura gsl_matrix è molto semplice:

```
typedef struct
{
  size_t size1;
  size_t size2;
  size_t tda;
  double * data;
  gsl_block * block;
  int owner;
} gsl_matrix;
```

La dimensione fisica determina il layout della memoria e può differire dalla dimensione della matrice per consentire l'utilizzo di sottomatrici. Le matrici sono memorizzate in ordine di riga, il che significa che ogni riga di elementi forma un blocco contiguo nella memoria. Questo è lo standard C-language ordering di array bidimensionali. Le funzioni di allocazione della memoria a una matrice seguono lo stile di malloc e free. Eseguono anche lo stesso controllo degli errori. Se non è disponibile memoria sufficiente per creare una matrice, le funzioni richiamano il gestore di errori GSL (con numero di errore GSL_ENOMEM) oltre a restituire un puntatore nullo.

```
gsl_matrix * gsl_matrix_alloc(size_t n1, size_t n2)
```

Questa funzione crea una matrice di n1 righe e n2 colonne, restituendo un puntatore a una struttura appena inizializzata. Un nuovo blocco viene assegnato agli elementi della matrice. Il blocco è posseduto dalla matrice e sarà deallocato quando la matrice viene deallocata. L'input zero per n1 o n2 è valido e restituisce un risultato non nullo.

```
int gsl_matrix_transpose_memcpy(gsl_matrix * dest, const
   gsl_matrix * src)
```

Questa funzione rende la matrice dest la trasposta della matrice src copiando gli elementi di src in dest. Questa funzione funziona per tutte le matrici a condizione che le dimensioni della matrice dest corrispondano alle dimensioni trasposte della matrice src.

```
int gsl_linalg_cholesky_decomp(gsl_matrix * A)
```

Questa funzione fattorizza la matrice quadrata A simmetrica e positiva definita nella decomposizione Cholesky $A = LL^T$. In input vengono utilizzati i valori delle parti diagonale e inferiore triangolare della matrice A (la parte triangolare superiore viene ignorata). In output le parti diagonale e inferiore triangolare della matrice di ingresso A contengono la matrice L, mentre la parte triangolare superiore non è modificata. Se la matrice non è positiva, la decomposizione non riesce, restituendo il codice di errore GSL_EDOM.

```
int gsl_linalg_cholesky_solve(const gsl_matrix * cholesky, const
    gsl_vector * b, gsl_vector * x)
```

Questa funzione risolve il sistema $Ax = b$ utilizzando la decomposizione Cholesky di A ottenuta dalla funzione precedentemente spiegata.

4.3.2 Wrapper GSL per Unity

In Unity, normalmente si usano gli script per creare funzionalità, ma si può anche includere codice creato al di fuori di Unity sotto forma di un plugin. Ci sono due tipi di plugin che possono utilizzare in Unity: plugin gestiti e plugin nativi.

I plug-in gestiti sono gestiti da assembly .NET creati con strumenti come Visual Studio o MonoDevelop. Contengono solo il codice .NET, il che significa che non possono accedere a funzionalità non supportate dalle librerie .NET. Tuttavia, il codice gestito è accessibile agli strumenti standard .NET che Unity utilizza per compilare gli script. Infatti, c'è poca differenza nell'utilizzo tra il codice plugin gestito e il codice script di Unity, ad eccezione del fatto che i plugin vengono compilati fuori da Unity e quindi la sorgente potrebbe non essere disponibile.

I plugin nativi sono librerie di codice nativo specifiche per la piattaforma. Possono accedere a funzioni quali chiamate OS e librerie di codice di terze parti che altrimenti non sarebbero disponibili per Unity. Tuttavia, queste librerie non sono accedibili dagli strumenti di Unity nel modo in cui lo sono le librerie gestite. Ad esempio, se si dimentica di aggiungere un file di plugin gestito al progetto, riceverai messaggi standard di errore del compilatore. Se si fa lo stesso con un plugin nativo, si vedrà solo un rapporto di errore quando si tenta di eseguire il progetto.
Di solito, gli script vengono conservati in un progetto come file di origine e compilati da Unity ogni volta che la fonte cambia. Tuttavia, è anche possibile compilare uno script per una libreria collegata dinamicamente (DLL) utilizzando un compilatore esterno. La DLL risultante può quindi essere aggiunta al progetto e le classi che contiene possono essere collegate a oggetti come gli script normali.
Generalmente, in Unity, è molto più facile lavorare con script rispetto che con le DLL.

Tuttavia, è possibile avere accesso a un codice di terze parti fornito sotto forma di DLL. Durante lo sviluppo del proprio codice, potrebbe essere possibile utilizzare compilatori non supportati da Unity (F#, per esempio) compilando il codice in una DLL e aggiungendolo al progetto Unity.
Unity ha un ampio supporto per i plugin nativi, che sono librerie di codice nativo scritte in C, C ++, Objective-C, ecc. I plugin consentono al codice gestito (scritto in Javascript o C#) di chiamare funzioni da queste librerie. Questa funzione consente di integrare Unity con le librerie middleware o codice esistente in C / C ++.
Per utilizzare un plugin nativo bisogna innanzitutto scrivere le funzioni in un linguaggio C-like per accedere a tutte le funzioni necessarie e compilarle in una libreria. In Unity, è inoltre necessario creare uno script C# che chiama le funzioni nella libreria nativa.
Il plugin nativo dovrebbe fornire una semplice interfaccia C che lo script C# poi espone ad altri script utente. Bisogna quindi creare un plug-in gestito per accedere a uno nativo con la stessa interffaccia.
Vediamo, infatti, il nostro esempio per il wrapping di GSL (GNU Scientific Library) e la creazione delle spline.

Libreria gestita in C#

```
public class MyCsharpLibrary
{
   [DllImport("MyCppLibrary")]
   public static extern void addPoint(float x, float y, float z,
      int pos);

   [DllImport("MyCppLibrary")]
   public static extern void uniformParameterization(float[]
      linSpace, int rows);

   [DllImport("MyCppLibrary")]
   public static extern void work(int size, int pval);

   [DllImport("MyCppLibrary")]
   public static extern float returnXSpline(int pos);

   [DllImport("MyCppLibrary")]
   public static extern float returnYSpline(int pos);

   [DllImport("MyCppLibrary")]
   public static extern float returnZSpline(int pos);
}
```

Libreria nativa in C++

```
#include <stdio.h>
#include <gsl/gsl_sf_bessel.h>
//Per le matrici e i vettori
#include <gsl/gsl_blas.h>
//Per Cholensky
#include <gsl/gsl_linalg.h>
//Per la valutazione del polinomio nella least square
   approximation
#include <gsl/gsl_poly.h>

extern "C" {
  TESTFUNCDLL_API void addPoint(float x, float y, float z, int
     pos);

  TESTFUNCDLL_API void uniformParameterization(float* linSpace,
     int rows);

  TESTFUNCDLL_API void work(int size, int pval);

  TESTFUNCDLL_API float returnXSpline(int pos);

  TESTFUNCDLL_API float returnYSpline(int pos);

  TESTFUNCDLL_API float returnZSpline(int pos);
}
```

4.4 Spline, rivoluzione e triangolarizzazione

Una volta avuti i punti dal Leap e aver capito come collegare una libreria c++ possiamo:

- scrivere il codice in C++ per la creazione delle spline in modo da usufruire degli algoritmi numerici della GSL;
- scrivere in C# l'algoritmo di rivoluzione già visto nel capitolo precedente;
- implementare gli algoritmi di triangolarizzazione.

Ovvero, la nostra funziona myHoldOut() avrà un aspetto del tipo:

```
void myHoldout()
{
      int splinePval = 10;
      int meshPval = 10;

      CurvesUtilities.calculateSpline(splinePval);

      CurvesUtilities.calculateMesh(meshPval);

      CurvesUtilities.triangulateMesh(splinePval, meshPval);

 }
```

4.4.1 Creazione Spline

Per creare il sistema delle equazioni normali fissiamo il numero di righe pari al numero dei punti tracciati e le colonne pari alla metà. Creiamo una partizione nodale tramite una parametrizzazione uniforme in base al numero di righe e una partizione nodale estesa tramite parametrizzazione uniforme in base al numero di colonne. Siano, quindi m, l'ordine della spline pari a 4, e k i nodi veri, pari al numero di righe / 2; allora:

```
uniformParameterization(rowsPartition, rows);
extendedNodalPartition(colsPartition, cols);

buildLSMatrix(rowsPartition, colsPartition, A, rows, cols, m);
```

dove uniformParameterization è una normale parametrizzazione uniforme e extended-NodalPartition:

```
//Nodi fittizi a sinistra coincidenti con l'estremo sinistro
   dell'intervallo = 0.
for (int i = 0; i < m; i++) {
  nodes[i] = 0;
}
//Nodi veri equispaziati
float* trueNodes = new float[k + 2]{};
uniformParameterization(trueNodes, k + 2);
for (i = m; i < m + k; i++) {
  nodes[i] = trueNodes[cont];
  cont++;
}
//Dimensione iniziale della partizione nodale estesa
int size = 2 * m + k;
```

```
//Nodi fittizi a destra coincidenti con l'estremo destro
   dell'intervallo = 1.
for (int i = m + k; i < size; i++) {
  nodes[i] = 1;
}
```

Mentre buildLSMatrix calcola, in base all'intervallo internodale, la funzione B-Spline e restituisce in A la matrice:

```
for (int k = 0; k < rows; k++) {
  l = localizeInternodalInterval(t[k], colsPartition, cols);
  buildBspline(l, nodes, t[k], v, cols, m);
  for (int j = 0; j < cols; j++) {
    A[k][j] = v[j];
  }
}
```

dove grazie alle formule di Cox la funzione buildBSpline calcola le funzioni base diverse da 0:

```
for (i = 0; i < m - 1; i++) {
  for (j = l - i; j <= l; j++) {
    v[j - 1] = nodes[i + j + 1] - point * v[j] / (point -
       nodes[j] + nodes[i + j + 1] - point);
    tmp = point - nodes[j] * v[j] / (point - nodes[j] + nodes[i
       + j + 1] - point);
  }
  v[l] = tmp;
}
```

Successivamente si scompone la matrice$A$ in H e la sua trasposta in modo tale da attuare l'algoritmo di Cholesky per la risoluzione del sistema lineare. Il tutto viene fatto tramite le chiamate a funzione alla GSL.

Ottenuti i coeficienti non rimane che valutare la curva con l'algoritmo di DeBoor:

```
//per ogni valore nella partizione nodale
for (tg = 0; k < pval - 1; tg += tstep) {
       //si localizza l'intervallo
       l = localizeInternodalInterval(tg, nodes, size + m);
       //si inizializzano i coeficienti in base al sistema appena
          risolto
       for (int i = 0; i < size; i++) {
         cx[cont] = alphax[i];
         cy[cont] = alphay[i];
         cz[cont] = alphaz[i];
                  cont++;
       }
       //tramite l'algoritmo di deBoor si calcola all'n-esimo
          passo il coeficiente
       for (int j = 1; j < m; j++) {
         for (int i = l - m + j + 1; i <= l; i++) {
           ti = i;
           tmj = i + m - j;
           den = nodes[tmj] - nodes[ti];

           cx[i] = (cx[i] * (tg - nodes[ti]) + cx[i - 1] *
              (nodes[tmj] - tg)) / den;
           cy[i] = (cy[i] * (tg - nodes[ti]) + cy[i - 1] *
```

```
            (nodes[tmj] - tg)) / den;
        cz[i] = (cz[i] * (tg - nodes[ti]) + cz[i - 1] *
            (nodes[tmj] - tg)) / den;
      }
    }
    //La valutazione in deBoor è nel coefficiente l-esimo e
       non nello 0
    sx[k] = cx[l];
    sy[k] = cy[l];
    sz[k] = cz[l];
    k++;
  }
  //Sistemiamo l'ultimo punto
  sx[pval - 1] = sx[pval - 2];
  sy[pval - 1] = sy[pval - 2];
  sz[pval - 1] = sz[pval - 2];
```

4.4.2 Rivoluzione e Triangolarizzazione

A seguito della creazione della Spline, la rivoluzione di essa intorno all'asse delle ordinate risulta identica allo pseudocodice fornito nel capitolo 3. Una rivoluzione di questo tipo, salvata in un vettore unidimensionale, presenta interessanti caratteristiche.
Dato che per ogni punto della spline, generiamo una circonferenza con punti di valutazione pari a *meshPval* e abbiamo *splinePval* punti, il vettore avrà dimensioni $meshPval \cdot splinePval$, dove:

- ogni punto di ogni circonferenza, internamente, avrà indice $0, 1, \ldots, j, \ldots meshPval$;
- ogni punto originale della Spline $0, 1, \ldots, i, \ldots, splinePval - 1$ nella mesh avrà indice $0 \cdot meshPval, 1 \cdot meshPval, \ldots, i \cdot meshPval, \ldots, (splinePval - 1) \cdot meshPval$
- quindi un punto j di una data circonferenza i avrà indice $i \cdot meshPval + j$.

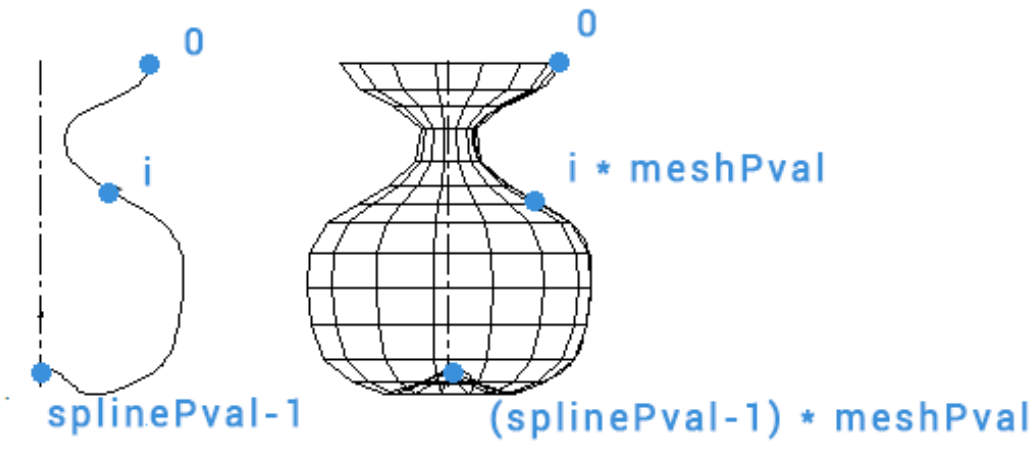
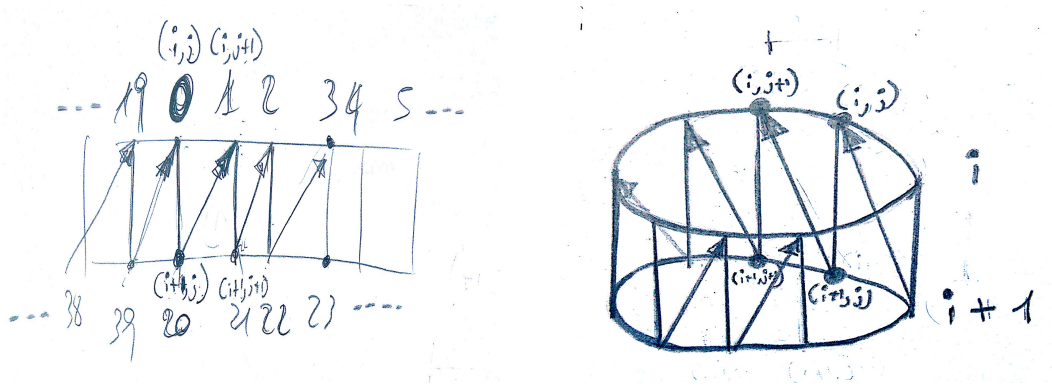


Figura 4.12: Mesh risultatante dalla rivoluzione.

Il nostro problema della triangolazione si limita, quindi:

- per la facce laterali, formare due facce triangolari per ogni quadrato descritto dai vertici (i,j) $(i,j+1)$ $(i+1,j)$ $(i+1,j+1)$
- triangolare le facce delle due basi tramite un algoritmo molto semplice data la loro geometria, quale l'algoritmo a ventaglio;

Con la mesh così composta possiamo, quindi, rappresentare ogni quadrato di ogni piano da dividere nel seguente modo:



(a) Proiezione ortogonale **(b)** Proiezione prospettica

Figura 4.13: Disegno di un piano della nostra SbC

Importante è creare i triangoli in modo tale che la normale alla superficie punti verso l'esterno. In geometria, una normale è una linea o un vettore che è perpendicolare ad un dato oggetto. Ad esempio, nel caso bidimensionale, la normale ad una curva in un determinato punto è la linea perpendicolare alla linea tangente alla curva del punto. Nel caso tridimensionale, una superficie normale, o semplicemente normale, ad una superficie in un punto P è un vettore che è perpendicolare al piano tangente a quella superficie in P. La parola "normale" è anche usata come aggettivo: la linea normale a un piano, la componente normale di una forza, il vettore normale, ecc. Il concetto di normalità viene generalizzato nell'ortogonalità. Per un poligono convesso (come

un triangolo), una superficie normale può essere calcolata come il prodotto trasversale vettoriale di due spigoli (non paralleli) del poligono.
Per un piano dato da $ax + ba + cz + d = 0$, il vettore (a, b, c) è la normale. Rispettando la figura 4.8 triangolarizziamo nel seguente modo:

```
for (int i = 0; i < splinePval - 1; i++)
{
  for (int j = 0; j < meshPval; j++)
  {
    faces.AddLast((meshPval * i + j + 1) == meshPval * (i + 1) ?
       meshPval * i : (meshPval * i + j + 1));
    faces.AddLast(meshPval * i + meshPval + j);
    faces.AddLast((meshPval * i + meshPval + j + 1) == meshPval
       * (i + 2) ? meshPval * (i + 1) : (meshPval * i + meshPval
       + j + 1));

    faces.AddLast(meshPval * i + j);
    faces.AddLast(meshPval * i + meshPval + j);
    faces.AddLast((meshPval * i + j + 1) == meshPval * (i + 1) ?
       meshPval * i : (meshPval * i + j + 1));
  }
}
```

Così facendo formiamo le facce:
$(i, j+1) \quad (i+1, j) \quad (i+1, j+1)$
$(i, j) \quad (i+1, j) \quad (i, j+1)$

Per quanto concerne invece la triangolazione delle facce delle basi si è usata una classe dello stesso Unity in grado di triangolare in modo efficace figure semplici come le circonferenze. In questo caso i vertici sono tutti inscritti in una circonferenza e l'algoritmo usato da Unity, l'algoritmo a ventaglio, è il più idoneo in termini di trade-off di efficenza/costo. La classe adoperata, Triangulator, lavora solamente con piano di 2D e quindi vengono mappati i nostri punti 3D escludendo la coordinata y per il calcolo della normale. Successivamente questi vertici vengono messi in un insieme che, dato in pasto al Triangulator, restituisce un insieme di facce triangolari.

```
Vector2[] vertices2D = new Vector2[ meshPval];

for (int i = 0; i < splinePval; i++)
{
  for (int j = 0; j < meshPval; j++)
  {
    verts.AddLast(new Vector3(myMeshPoints[i * meshPval + j].x,
       myMeshPoints[i * meshPval + j].y, myMeshPoints[i *
       meshPval + j].z));
    if (i == 0)
    {
      vertices2D[j] = new Vector2(myMeshPoints[i * meshPval +
         j].x, myMeshPoints[i * meshPval + j].z);
    }
  }
}

Triangulator tr = new Triangulator(vertices2D);
int[] indices = tr.Triangulate();
```

```
//il verso con cui vengono considerati i vertici della seconda
    faccia (l'inferiore) sono inversi dalla prima (la superiore)
    per far puntare il vettore normale nel verso opposto (in
    basso), cosicchè essa punti all'esterno della mesh, e quindi
    sia visibile.
for (int i = 0; i < 2; i++)
{
   for (int j = 0; j < indices.Length; j++)
   {
      faces.AddLast(indices[i == 0 ? j : (indices.Length - 1 - j)]
         + (i * (splinePval - 1) * meshPval));
   }
}
```

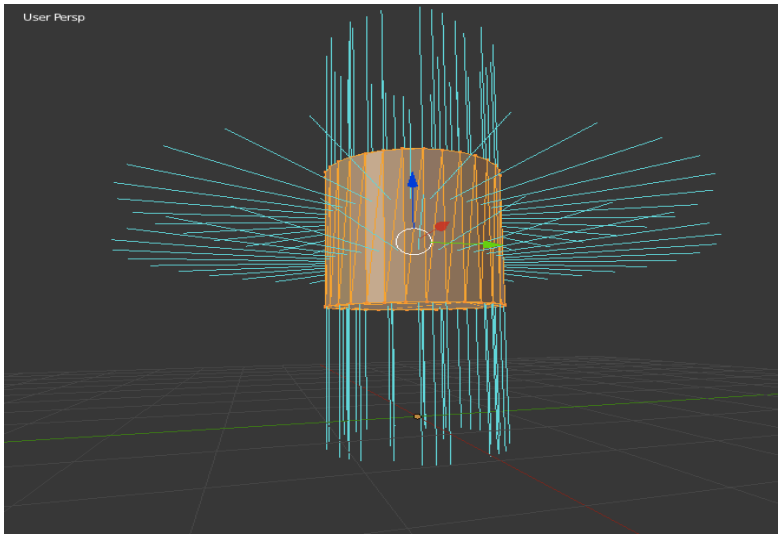


Figura 4.14: Normali alle facce triangolari di un cilindro

4.5 Esempi di modellazione

In questa sezione vengono visualizzati altri esempi di modelli virtuali immersi nella realtà.

In particolare vengono forniti gli screen effettuati da uno smartphone in modalità VR. Partendo da opportune curve 3D descritte tramite la posizione delle dita sono stati modellati rispettivamente un cappello a forma di clindro, un cappelo di paglia, un imbuto, un'anfora, un piatto e una vaso da fiori. Nelle ultime due immagini si può notare come la mesh sia di colore diverso, ciò è dovuto all'interazione dell'utente con essa. In questo caso caso si visualizzano anche il modello delle mani virtuali.

Figura 4.15: Cappello a forma di cilindro.

Figura 4.16: Cappello di paglia.

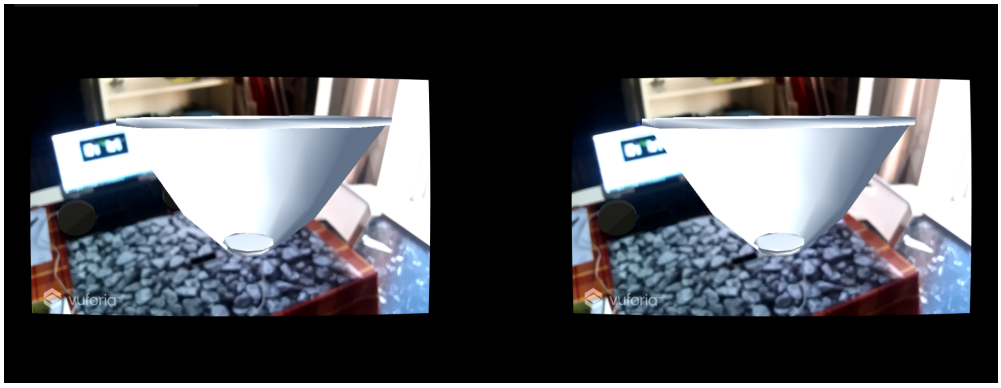


Figura 4.17: Imbuto.

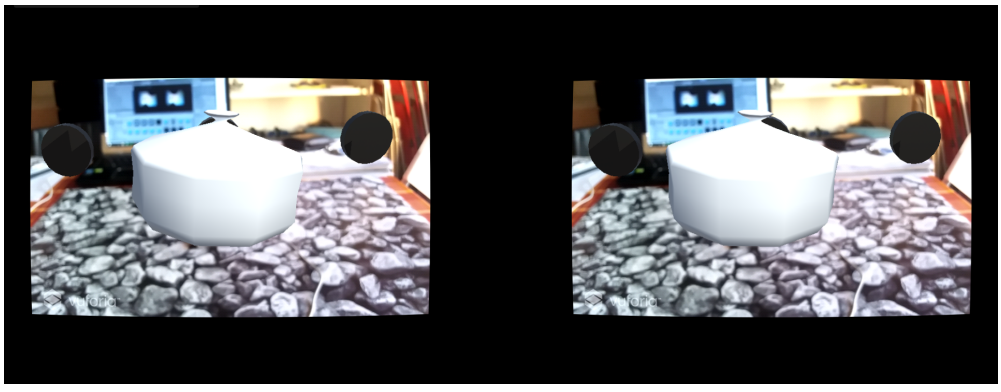


Figura 4.18: Anfora.

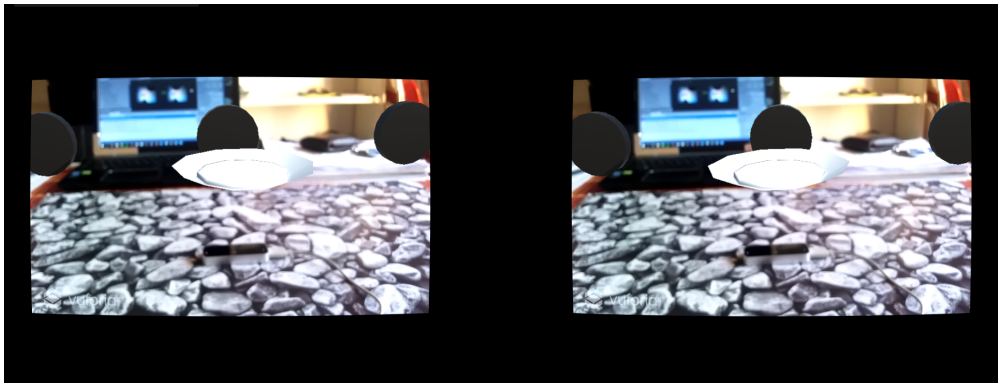


Figura 4.19: Piatto.

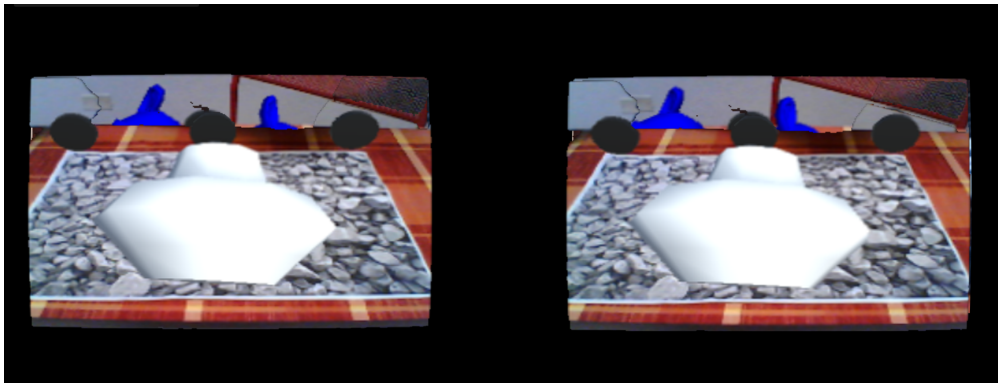


Figura 4.20: Vaso a fiori.

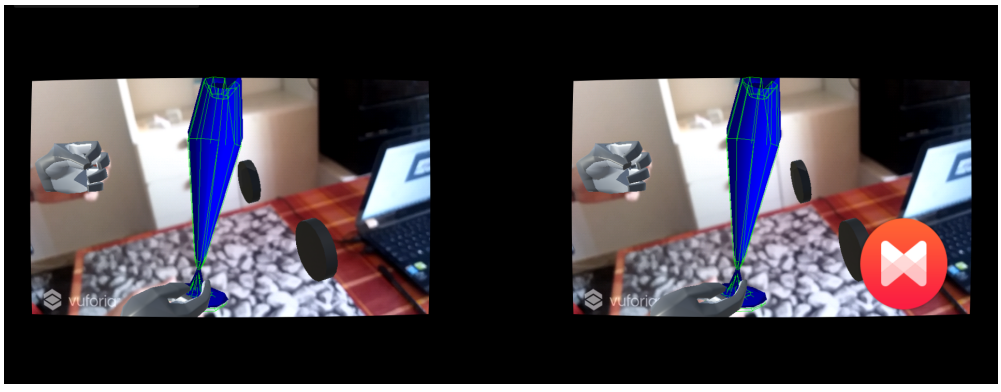


Figura 4.21: Calice.

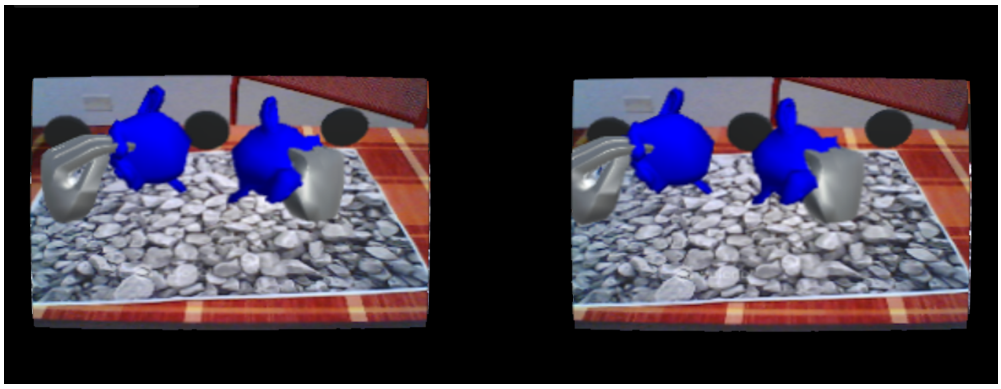


Figura 4.22: Doppia interazione.

5

Conclusioni e sviluppi futuri

Dopo aver esposto le caratteristiche del progetto sviluppato, si può concludere che esso soddisfa i requisiti iniziali.
L'esperienza della prova finale ha dato modo di approfondire le tecnologie e le modalità di sviluppo di applicazioni in realtà aumentata che, difficilmente avremmo potuto vedere in maniera così dettagliata in altri corsi.
Tecnologie del genere, data la loro vicinanza con la realtà, hanno una grande potenzialità in numerose branche scientifiche; esse possono essere utili per simulazioni ancora più dettagliate, metodi di apprendimento più efficaci e potrebbero far parte di un sistema di pervasive computing per migliorare e incrementare le informazioni su quello che ci circonda.
Tecnologie del genere danno un supporto non indifferente nell'integrare l'informatica nella vita di ogni giorno, con lo scopo di migliorare la qualità di quello che si sta facendo.

Proprio per questi motivi lo sviluppo di questo progetto è stato stimolante.
Oltretutto, tra le numerose applicazioni si è visto lo sviluppo di un software che permettesse di portare la modellazione di mesh nella realtà e, questo, ha permesso di lavorare con il mondo della computer graphics, altra branca altrettanto affascinante.
Oltre alla realtà aumentata, il progetto, mi ha posto davanti ad altre tecnologie, come il tracciamento delle mani con il dispositivo Leap Motion, ed è stato molto soddisfacente integrare più tecnologie per rendere il software il più "coinvolgente" possibile.
Nell'unire il Leap con il resto del progetto si sono, poi, trovati degli ostacoli da superare e, nel farlo, si sono potuti apprezzare l'importanza degli algoritmi di serializzazione e compressione, e la loro influenza nelle performance. Si è potuto sperimentare quanto, in un'applicazione Client-Server, il numero di Client connessi, la banda disponibile e la mole di dati influisca nelle latenze e quindi nella resa del software.
In conclusione, il forte interesse nei confronti dell'argomento e le conoscenze acquisite durante il percorso, mi fanno ritenere soddisfatto dell'esecuzione dell'intero progetto.
Possibili sviluppi futuri potrebbero riguardare:

- l'aumento delle features sotto il punto di vista del progetto dell'interazione, integrazione maggiore del motore grafico Blender per azioni più complicate sulle Mesh, quali estrusione e modifica della geometria dei punti;
- l'aumento delle features sotto il punto di vista del progetto delle curve, implementazione curve NURBS e relativa modifica dei punti;
- l'adattamento del software ad altre branche come quella medica, come supporto allo studio, architetttonica, come supporto alla progettazione, e aerospaziale, come supporto alla prototipazione.

Ringraziamenti

Il mio ringraziamento più grande va a mia madre che c'è sempre stata nonostante non potesse capire le mie difficoltà, grazie per l'amore, la fiducia, i caldi abbracci, il supporto e la pazienza.
Non da meno ringrazio mio padre che ha sempre creduto in me, mi ha supportato con tutto se stesso e ha sempre pregato per me.
Ringrazio mio fratello che, come sempre, c'è stato e mi ha aiutato nel lavoro di tesi quando ero più giù di morale, non credo che senza il suo incoraggiamento e aiuto avrei potuto completare il mio lavoro.
Colgo l'occasione per ringraziare Alba e Blue per aver portato un sorriso a casa nostra.
Ringrazio mio zio Massimo, mia zia Lorena, mia nonna Virginia, mia zia Lidia, mia cugina Silvia, suo marito Massimo e Mattia per il loro affetto.
Allo stesso modo ringrazio Elena, Patrizio, Giovanni, Giacomo e Teresa.
Vorrei anche cogliere l'occasione per ringraziare tutti i miei amici, quelli che avevo già e quelli che ho conosciuto in questi tre anni di lavoro, in particolare:
Sara, Federica, Alessandra, Camilla e Martina le amiche di una vita che non hanno bisogno di parole;
Edoardo C., Edoardo T., Eugenio, Nicola, Christian e Alessandro i momenti con loro non hanno prezzo, ci sono sempre stati e li considero come fratelli;
Claudio, Michele ed Elena con cui ho stretto subito tanto e ho un rapporto bellissimo;
Mariele per la sua amicizia;
Jeremias, un amico che, nonostante la lontananza, continuerò a non dimenticare;

Giulia, Andrea, Marcello, Tommaso e Lorenzo compagni di università che hanno affrontato con me scoglio dopo scoglio;
Marco con cui mi sono sempre confrontato e aiutato durante il percorso, compreso questo lavoro; Matteo oltre a essere un grande amico, è stato un ottimo compagno di progetti;
Grazie a chi ha vissuto con me in studentato come Daniele, Mirko, Claudia e Maria Luisa; a chi ho conosciuto grazie a loro, Alessandra e Leriana; ai compagni di calcetto Davide, Diego, Luca, Valerio, Simone e Giuseppe; a chi ha portato un po' di pazzia in via ix febbraio, Andrea.
In particolare i momenti vissuti nell'appartamento in via ix febbraio sono stati unici. Vorrei ringraziare a chi mi ha aperto le porte in secondo anno e mi hanno fatto sentire a casa, Emiliana, Gloria e Andrea e a chi è stato sempre più di un coinquilino, Francesco P. e Francesco A.
Un ringraziamento a Giuseppe, compagno di progetti, di calcetto, coinquilino e amico. Spero di non aver dimenticato nessuno.

Un ringraziamento lo devo anche ai miei professori per le capacità e l'esperienza che hanno sempre messo a mia disposizione, in particolare la Prof.essa Damiana Lazzaro che mi ha fatto appassionare ancor più alla matematica ed è sempre stata presente per qualsiasi chiarimento e al Prof. Alessandro Ricci che con il suo carisma mi ha fatto appassionare alla sua materia e mi ha fatto esplorare la realtà aumentata.

Un ringraziamento particolare va' a mia nonna Rosa che l'avrei voluta presente più di ogni altra cosa.
E, finalmente, per ultima, ma non per importanza, vorrei ringraziare la mia Ilaria per il suo amore, la sua pazienza e la sua fiducia in me. Grazie per aver reso ogni momento difficile, un momento migliore. Questa laurea senza di lei non ci sarebbe stata.

Bibliografia

[1] Stephen Cawood, Mark Fiala *Augmented reality : a practical guide*, 2007.

[2] Oliver Bimber, Ramesh Raskar *Spatial augmented reality : merging real and virtual worlds*, 2005.

[3] Patrick Felicia *Unity 5 from zero to proficiency (intermediate)*, 2016.

[4] Patrick Felicia *Unity 5 from zero to proficiency (advanced)*, 2016.

[5] Les Piegl, Wayne Tiller *The Nurbs Book*, 1997.

Sitografia

[6] *Blender Wiki | Developer*,
https://wiki.blender.org/index.php/Dev:Contents

[7] *Blender Websocket*,
https://github.com/KoltesDigital/websocket-server-for-blender

[8] *Leap Motion | Community*,
https://community.leapmotion.com

[9] *Leap Motion | Developer*,
https://developer.leapmotion.com/documentation/csharp

[10] *Leap Motion | Unity*,
https://developer.leapmotion.com/unity#116

[11] *Leap Motion | Modules*,
https://developer.leapmotion.com/documentation/csharp

[12] *Stack Overflow*,
https://github.com/leapmotion/UnityModules/wiki

[13] *Unreal | Features*,
https://www.unrealengine.com/features

[14] *Unreal ETH Zurich| Augmented Reality*,
https://github.com/adynathos/AugmentedUnreality

[15] *Unreal4AR*,
http://www.unreal4ar.com

[16] *Unity | Documentation*,
https://docs.unity3d.com/Manual

[17] *Unity | Community*,
https://forum.unity3d.com

[18] *Unity | Triangulator*,
http://wiki.unity3d.com/index.php?title=Triangulator

[19] *Vuforia Developer Portal | Support*,
https://developer.vuforia.com/supportl

[20] *LaTeX: a document preparation system*,
https://en.wikibooks.org/wiki/LaTeX