Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Informatica

Leap Aided Modelling (LAM) in Blender

Tesi di Laurea in Computer Graphics

Relatore:
Prof.ssa
Serena Morigi

Presentata da:
Luca Rinaldi

Correlatore:
Dott.
Flavio Bertini

Sessione I
Anno Accademico 2015/2016

Introduzione

L'evoluzione tecnologica e l'utilizzo crescente della computer grafica in diversi settori stanno suscitando l'interesse di sempre più persone verso il mondo della modellazione 3D. I software di modellazione, tuttavia, si presentano spesso inadeguati all'utilizzo da parte di utenti senza esperienza, soprattutto a causa dei comandi di navigazione e modellazione poco intuitivi. Dal punto di vista dell'interazione uomo-computer, questi software devono infatti affrontare un grande ostacolo: il rapporto tra dispositivi di input 2D (come il mouse) e la manipolazione di una scena 3D. Il progetto presentato in questa tesi è un addon per Blender che consente di utilizzare il dispositivo Leap Motion come ausilio alla modellazione di superfici in computer grafica. L'obiettivo di questa tesi è stato quello di progettare e realizzare un'interfaccia user-friendly tra Leap e Blender, in modo da potere utilizzare i sensori del primo per facilitare ed estendere i comandi di navigazione e modellazione del secondo. L'addon realizzato per Blender implementa il concetto di *LAM* (Leap Aided Modelling: modellazione assistita da Leap), consentendo quindi di estendere le feature di Blender riguardanti la selezione, lo spostamento e la modifica degli oggetti in scena, la manipolazione della vista utente e la modellazione di curve e superfici Non Uniform Rational B-Splines (NURBS). Queste estensioni sono state create per rendere più veloci e semplici le operazioni altrimenti guidate esclusivamente da mouse e tastiera. Il presente documento di tesi è strutturato nel seguente modo: nel Capitolo 1 vengono illustrati gli obiettivi dell'addon, mostrando le motivazioni che hanno portato alla sua progettazione; nel Capitolo 2 viene descritto il dispositivo Leap

Motion e vengono elencate le funzionalità offerte agli sviluppatori grazie alle sue API, aprendo una breve parentesi anche sui dispositivi simili presenti sul mercato; il Capitolo 3 contiene una presentazione del software open-source di modellazione 3D Blender, una panoramica sui software di LAM esistenti e la descrizione delle feature di navigazione implementate all'interno dell'addon; nel Capitolo 4 vengono illustrate in forma teorica le curve e le superfici NURBS, mostrando in particolare le tecniche di Surface by Curves come l'estrusione e lo skinning; nel Capitolo 5 vengono descritte le feature di modellazione realizzate per il progetto, come l'implementazione delle tecniche di Surface by Curves illustrate nel capitolo precedente; per concludere, il Capitolo 6 mostra la validazione finale dell'addon prodotto, analizzando pregi e difetti di ogni feature implementata e mostrando un rendering di una scena 3D realizzata utilizzando prevalentemente LAM.

Indice

Elenco delle figure

Capitolo 1

Scenario e obiettivi

1.1 Scenario

La computer grafica 3D ha assunto sempre più importanza negli ultimi anni e viene utilizzata in moltissimi settori anche diversi tra di loro, come il cinema, i videogiochi, l'ingegneria, l'arte e la medicina. Oltre ad essere utilizzata a scopi professionali, con il passare del tempo sempre più persone (anche senza competenze tecniche specifiche) sono interessate a realizzare un prodotto utilizzando la computer grafica 3D. Questo interesse crescente è frutto soprattutto dei seguenti motivi:

- Il mondo del cinema, negli ultimi anni, ha fatto un uso sempre più frequente degli effetti speciali realizzati grazie alla computer grafica. Il cambiamento forse più evidente è stato il passaggio dai cartoni animati bidimensionali a quelli 3D (Pixar, Dreamworks, etc.), realizzati completamente in computer grafica. Gli spettatori e la critica cinematografica si sono ormai adeguati ad un utilizzo ottimale di queste tecniche.

- I videogiochi fatturano circa 90 miliardi di euro, più di musica e cinema messi insieme. Per questo motivo il mercato videoludico è in rapida espansione, così come il numero di videogiocatori. Sempre più sviluppatori, anche e soprattutto alle prime armi, cercano di fare suc-

cesso creando un proprio videogioco da vendere al pubblico. Lo standard grafico dei videogiochi si avvicina sempre di più al fotorealismo in real-time e ultimamente le aziende videoludiche si concentrano sull'immersività offerta ai videogiocatori, ad esempio sviluppando visori per realtà virtuale.

- Le stampanti 3D, nell'ultimo periodo, hanno suscitato l'interesse di molte persone, sia per scopi professionali che per scopi personali. Il mercato sta iniziando ad offrire stampanti 3D economiche per uso domestico, a prezzi sempre più bassi, che offrono la possibilità di stampare modelli 3D scaricati online. Tuttavia, gli utenti interessati a creare e stampare un proprio modello devono imparare le basi della modellazione 3D.

Grazie ai numerosi software di modellazione 3D disponibili è possibile produrre immagini ed animazioni a partire da modelli geometrici tridimensionali. Queste immagini possono rappresentare scene ed oggetti reali (fotorealismo) o meno e vengono realizzate seguendo i seguenti passaggi:

Modellazione La modellazione descrive il processo per definire la forma degli oggetti. Un modello 3D viene spesso definito su un sistema di riferimento cartesiano tridimensionale attraverso mesh poligonali e curve e superfici NURBS. Una mesh poligonale è un insieme di vertici, lati e facce (triangoli o quadrilateri) che definisce la forma di un oggetto poliedrico.

Layout e animazione Prima di trasformare gli oggetti in immagini, questi devono essere inseriti una scena. Questo processo definisce la posizione, la dimensione e la rotazione degli oggetti nello spazio tridimensionale, specificando anche le relazioni che possono avere tra di loro. Si può anche associare una animazione ad un oggetto definendo le sue trasformazioni nel tempo.

Rendering Il processo di rendering si occupa di convertire il modello in una immagine (o in un insieme di frame, per quanto riguarda le animazioni)

proiettando i modelli tridimensionali in una superficie 2D. In questa fase viene definito il colore di ogni pixel dell'immagine finale in base alle proprietà dell'oggetto e all'illuminazione della scena (shading).

1.2 Obiettivi

I processi di modellazione e layout, nei più comuni software di modellazione 3D, richiedono che l'utente interagisca con oggetti e scene tridimensionali utilizzando dispositivi di input 2D (mouse, touchpad, etc.). Alcune funzionalità di questi programmi risultano essere limitate da questo vincolo (ad esempio lo spostamento di un oggetto in profondità o la navigazione in scena) ed esistono comunque dei metodi non molto intuitivi per raggiungere l'obiettivo. Mentre gli utenti abituati all'utilizzo dei software di modellazione riescono a superare questi limiti grazie all'esperienza, gli utenti alle prime armi potrebbero trovarsi in difficoltà e i tempi di apprendimento possono essere abbastanza lunghi.

Il progetto presentato in questa tesi ha l'obiettivo di ridurre i vincoli di interazione tra l'utente e la scena 3D, cercando di rimuovere i limiti appena descritti nel software open-source di modellazione 3D Blender (descritto nel Paragrafo 3.1). Per fare questo si utilizza il dispositivo di input Leap Motion (descritto nel prossimo capitolo), in grado di rilevare la posizione delle mani sopra di esso e di accettare quindi un input 3D. Si vuole dunque migliorare l'interazione tra utente e computer introducendo comandi user-friendly, facendo assistere a Leap la normale modellazione 3D di Blender, definendo così il concetto di *LAM* . Tuttavia, l'utilizzo di Leap come unico dispositivo di input, a lungo andare, potrebbe richiedere troppo sforzo fisico e risultare stressante per l'utente, con il rischio di ottenere il risultato opposto a quello voluto. L'obiettivo principale non è stato quindi quello di utilizzare esclusivamente Leap come dispositivo di input per la modellazione, ma quello di renderlo un dispositivo aggiuntivo e cooperante a mouse e tastiera.

In particolare, per permettere a Leap Motion e Blender di interfacciarsi

tra di loro, è stato implementato un addon per Blender che integra diverse funzioni appartenenti a due insiemi: feature di navigazione e feature di modellazione. Le feature di navigazione, descritte in dettaglio nel Capitolo 3, comprendono tutte quelle funzioni relative allo spostamento e alla modifica degli oggetti in scena, inclusa la manipolazione della vista utente (camera). Queste funzionalità, già presenti in Blender, sono state migliorate e rese più semplici grazie all'utilizzo di Leap. Le feature di modellazione, descritte nei Capitoli 4 e 5, comprendono invece delle tecniche di creazione di curve NURBS (Non Uniform Rational B-Splines) e di superfici attraverso *Surface by Curves* (SbyC) con Leap Motion. A differenza delle feature di navigazione, quelle di modellazione sono state aggiunte in Blender, in quanto non presenti nella versione nativa del software.

Capitolo 2

Il dispositivo Leap Motion

2.1 Introduzione a Leap Motion

Leap Motion è una startup americana fondata nel 2010 con l'obiettivo di sviluppare tecnologie per la rilevazione del movimento nell'interazione tra uomo e computer. Il motivo che spinse Michael Buckwald e David Holz a fondare l'azienda fu la convinzione che un tipo di input basato sulla semplicità e sulla naturalezza, avrebbe permesso di sfruttare al massimo una tecnologia così potente come quella dei nostri giorni. Per questa ragione, nel 2012, crearono il *Leap*[1], uno strumento in grado di rilevare la posizione delle mani e delle dita per consentire all'utente di interagire direttamente con il software attraverso il solo movimento delle mani.

Il Leap è un dispositivo che va posizionato sul tavolo di lavoro (in commercio esistono anche notebook e tastiere con il Leap integrato) e che grazie alle sue due telecamere monocromatiche e ai suoi 3 led infrarossi, riesce a rilevare la posizione delle mani e delle dita (o di strumenti di puntamento, come una matita) in un'area semisferica capovolta che va dal basso verso l'alto con un'ampiezza di circa 150 gradi e con un raggio che va approssimativamente dai 25 ai 600 millimetri in altezza.

[1]Il dispositivo "Leap" può essere anche chiamato "Leap Motion", come il nome dell'azienda in cui è stato inventato.

Figura 2.1: Leap Motion

Leap Motion mette a disposizione SDK in vari linguaggi di programmazione per lo sviluppo di applicazioni Leap-oriented. A Febbraio del 2016, Leap Motion ha rilasciato Orion, un aggiornamento di Leap focalizzato sui visori di realtà virtuale (il Leap può essere attaccato davanti al visore), in grado di migliorare notevolmente le prestazioni del dispositivo per il riconoscimento in tempo reale. Le applicazioni per Leap Motion vengono rilasciate su un App Store ufficiale [2] che contiene più di 200 applicazioni per diversi sistemi operativi, sia gratuite che a pagamento. Leap viene anche utilizzato da chirurghi e ricercatori nel campo della medicina (rif. [2]), dai progettisti di automobili (rif. [3]) e dai musicisti (rif. [4]).

2.2 Dispositivi simili

Negli ultimi anni sono nati molti dispositivi simili a Leap Motion, basati sul riconoscimento dei gesti e sul rilevamento della posizione delle mani e di altre parti del corpo. Nella maggioranza dei casi questi dispositivi sono stati progettati per creare immersione nel mondo dei videogiochi, per poi essere sfruttati anche in altri settori.

[2]https://apps.leapmotion.com/

(a) Wiimote (b) Playstation Move

(c) HTC Vive controller

Figura 2.2: Controller sensibili al movimento

Wiimote, Playstation Move e HTC Vive controllers

Wiimote (Wii Remote), Playstation Move e HTC Vive, in figura 2.2, sono tre diversi controller sensibili al movimento, ideati rispettivamente da Nintendo, Sony e HTC per le piattaforme Nintendo Wii, Playstation 3 e PC. Questi controller hanno l'obiettivo di rilevare in tempo reale la posizione delle mani (ma non delle singole dita) per interagire direttamente con il videogioco. La differenza principale tra Leap Motion e questi dispositivi è che questi ultimi hanno bisogno che l'utente li tenga in mano per tutta la durata del gioco, mentre per utilizzare Leap non è necessario indossare o

tenere in mano nulla. Per il riconoscimento dello spostamento delle mani, i controller contengono accelerometri e giroscopi, utili soprattutto per rilevare in modo accurato la rotazione del polso e la velocità di spostamento. Tutti questi dispositivi utilizzano strumenti aggiuntivi per calcolare con precisione la posizione delle mani nello spazio: Wiimote utilizza una *Wii Sensor bar* da applicare sopra al televisore, Playstation Move utilizza una telecamera stereoscopica (*Playstation Camera*) e HTC Vive usa dei sensori chiamati *Lighthouse*.

Kinect

Figura 2.3: Microsoft Kinect One

Il Kinect (o il suo successore, il Kinect One, in figura 2.3), è un dispositivo creato da Microsoft per il riconoscimento del corpo umano, ideato per le console di gioco Xbox 360 e Xbox One (rif. [5]). Grazie ad una telecamera RGB, una telecamera ad infrarossi e ad un array di microfoni, Kinect è capace di riconoscere i comandi vocali e il movimento di ogni parte del corpo (testa, braccia, mani, gambe e busto), fino ad un massimo di quattro giocatori. Le alte potenzialità di questo dispositivo lo hanno reso famoso anche fuori dal contesto videoludico, infatti viene spesso utilizzato in campi scientifici e di ricerca.

HoloLens

Hololens (figura 2.4 e rif. [6]) è un visore di realtà aumentata attualmente in sviluppo da Microsoft e presentato all'Electronic Entertainment Expo

Figura 2.4: Microsoft HoloLens

(E3) 2015. Il visore integra sensori avanzati, un display ottico 3D ad alta definizione ed un sistema di scansione spaziale dei suoni, permettendo di interagire con esso mediante lo sguardo, la voce o i gesti delle mani. I sensori al suo interno permettono inoltre di effettuare head-tracking, catturare video e catturare suoni. Inoltre Hololens contiene una Holographic Processing Unit (HPU), un coprocessore che gestisce attività come la mappatura del territorio (con riconoscimento di ostacoli e superfici), il riconoscimento dei gesti ed il riconoscimento vocale.

2.3 Architettura di Leap Motion

Il software di Leap Motion è in continua esecuzione su un demone e si connette al dispositivo (Controller) attraverso il bus USB. Le applicazioni che utilizzano Leap interrogano questo demone per ricevere le informazioni di tracking aggiornate frame per frame, facendo uso delle apposite API fornite dai creatori del dispositivo. L'SDK di Leap Motion offre API per JavaScript, Unity, C#, C++, Java, Unreal, Objective-C e Python. L'architettura in figura 2.5 mostra come una applicazione che utilizzi Leap Motion possa utilizzare le sue librerie dinamiche per connettersi al demone in esecuzione e ricevere informazioni sui dati di tracking:

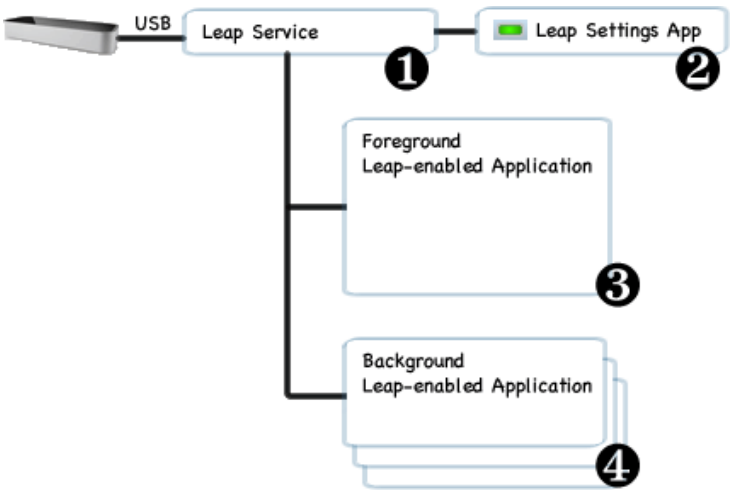


Figura 2.5: Architettura di Leap Motion

1. Il demone di Leap Motion riceve le informazioni da Leap attraverso il bus USB, le processa e le spedisce alle applicazioni che fanno uso delle API. Di default i dati vengono spediti solo alle applicazioni foreground, ma le applicazioni in background possono comunque fare la richiesta per la ricezione dati (essa potrà, eventualmente, essere negata dall'utente).

2. L'utente può modificare le impostazioni di Leap Motion (sensibilità, performance, privacy, etc.) attraverso un pannello di controllo.

3. Una applicazione in foreground che utilizza le API di Leap Motion può ricevere i dati di tracking del Leap facendone richiesta al suo demone.

4. Quando una applicazione va in background o comunque perde il focus dell'utente, il demone di Leap Motion smette di inviarle i dati. Per ricevere comunque i dati in qualsiasi modalità, l'applicazione deve prima richiedere i permessi all'utente.

2.4 Dati di tracking rilevati dal dispositivo

Il Leap Motion è capace di rilevare le mani, le dita e oggetti di puntamento (viene rilevato come oggetto di puntamento qualsiasi oggetto simile ad una

penna o ad una matita) nel suo raggio di azione. Esso aggiorna le applicazioni connesse con insiemi multipli di dati di tracking, suddivisi per frame (ogni frame contiene una lista di oggetti relativi alle entità rilevate, come mani o dita).

Sistema di coordinate

Leap Motion utilizza un sistema di coordinate cartesiane dove l'origine è posizionata esattamente sopra e nel centro del dispositivo e gli assi sono disposti come in figura 2.6. L'asse x è parallelo al dispositivo con valori crescenti verso destra, l'asse y è ortogonale al dispositivo con valori crescenti verso l'alto, mentre l'asse z rappresenta la profondità tra l'utente e lo schermo, con valori crescenti verso l'utente.

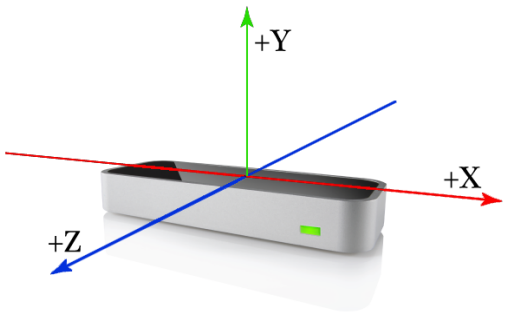


Figura 2.6: Sistema di coordinate di Leap Motion

Leap Motion misura le distanze in millimetri, il tempo in microsecondi, la velocità in mm/s e gli angoli in radianti.

Riconoscimento delle mani

Il riconoscimento della mano in Leap Motion tiene conto della sua identità, della sua posizione e la lista delle dita associate ad essa.

Il software di Leap Motion utilizza modelli interni delle mani umane per dare una stima della posizione delle parti della mano anche quando alcune di queste non vengono rilevati dai sensori del dispositivo (in base anche ai dati

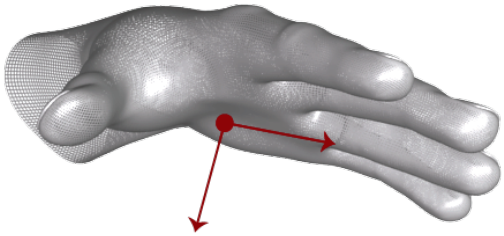


Figura 2.7: Vettori normale e direzione di una mano

dei frame precedenti). Una applicazione che fa uso delle API di Leap Motion può prendere in input l'insieme degli oggetti relativi alle mani rilevati da Leap in un particolare frame. Tra le proprietà di ognuno di questi oggetti ci sono la posizione di ogni mano, la sua direzione e la sua normale (figura 2.7). Inoltre gli oggetti in questione forniscono anche informazioni sulla chiusura della mano (se chiusa a pugno o completamente aperta), sul tipo di mano (destra o sinistra) e sulle cinque dita collegate ad essa. Leap Motion può rilevare anche più di due mani alla volta (ad esempio con due persone) ma si deve fare attenzione a non sovrapporle, perché una mano sotto all'altra può oscurare quella sopra.

Riconoscimento delle dita

Figura 2.8: Posizione e direzione delle dita di una mano

Il Leap Motion offre informazioni su ogni dito della mano. In particolare una applicazione che fa uso delle API di Leap Motion può ricevere informazioni sulla posizione e sulla direzione di ogni dito (figura 2.8) in coordinate di Leap Motion. Anche in questo caso il software provvede a stimare la posizio-

ne delle dita (o delle parti di dito) non rilevate direttamente dal dispositivo, in modo che l'applicazione possa comunque proseguire. La stima sulla posizione delle parti di dito mancanti viene fatta utilizzando le caratteristiche anatomiche di ogni dito, in base alla posizione delle sue ossa (metacarpo, falange, falangina e falangetta).

Ogni oggetto corrispondente ad un dito viene identificato in base alla mano di appartenenza e al suo tipo (pollice, indice, medio, anulare o mignolo). Ognuno di questi oggetti descrive anche la posizione e l'orientamento di ogni osso del dito (quattro ossa per dito).

Immagini grezze

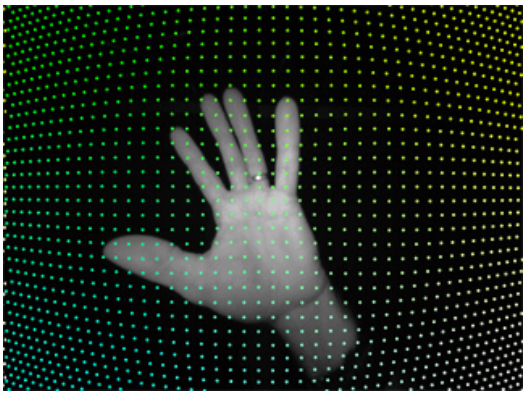


Figura 2.9: Immagine grezza ripresa da Leap Motion

Le ultime API aggiornate di Leap Motion, oltre ai dati di tracking visti nei paragrafi precedenti, offrono anche la possibilità di recuperare le immagini grezze monocromatiche a scala di grigi catturate dalle due telecamere stereoscopiche ad infrarossi (le applicazioni che intendono leggere ed elaborare queste immagini devono richiedere il consenso all'utente). Ad ogni frame l'applicazione con i determinati permessi può richiedere a Leap Motion le due immagini (telecamera destra e telecamera sinistra) ed utilizzarle ad esempio per il riconoscimento di marker catarifrangenti per la realtà aumentata o per il rendering dell'immagine catturata.

Le due immagini sono fornite come matrici di valori di luminosità (0-255) corrispondenti ai relativi pixel. Le immagini sono distorte ma grazie alle API di Leap Motion è possibile correggerle attraverso gli appositi dati di calibrazione (figura 2.9). Le due videocamere sono posizionate sullo stesso asse e distano tra di loro 40mm, dunque analizzando le due immagini è possibile calcolare la profondità di ogni pixel.

2.5 Gestures riconosciute

Oltre al normale rilevamento di mani e dita, Leap Motion offre all'utente la possibilità di interagire con le applicazioni attraverso delle gesture predefinite. Le gesture sono pattern di movimento che indicano l'intenzione dell'utente di eseguire una determinata azione. Esse possono durare diversi secondi e dunque le API di Leap Motion forniscono gli aggiornamenti relativi alle gesture ad ogni frame, per tutta la loro durata. Per riconoscere le gesture, l'applicazione che ne fa uso deve prima abilitare il loro riconoscimento.

Circle

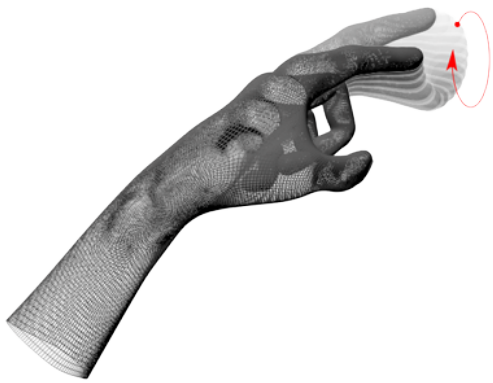


Figura 2.10: Gesture: Circle

La gesture *Circle* viene riconosciuta come ogni cerchio in aria (in senso orario o antiorario) fatto da un dito o da un oggetto (figura 2.10). Essendo

una gesture continua nel tempo, Leap Motion aggiorna ogni frame con i progressi del cerchio finché il dito smette di ruotare o rallenta troppo. Leap Motion fornisce il numero di giri e la normale al cerchio "disegnato" in aria, con la quale è possibile vedere se il dito è ruotato in senso orario o antiorario.

Swipe

Figura 2.11: Gesture: Swipe orizzontale

Leap Motion riconosce come gesture *Swipe* il movimento lineare di un dito in qualsiasi direzione (in figura 2.11 quella orizzontale). Anche in questo caso la gesture è continua nel tempo, dunque all'interno di ogni frame sono presenti i dati aggiornati fino a quel punto. La gesture termina quando il dito cambia direzione o il movimento diventa troppo lento. In ogni frame sono presenti i dati relativi alla direzione, alla velocità e alla lunghezza della gesture.

Key taps

La gesture *Key tap* viene riconosciuta quando un dito si muove velocemente verso il basso (figura 2.12), come se si stesse premendo un pulsante immaginario. In questo caso la gesture è discreta ed è quindi contenuta nel

Figura 2.12: Gesture: Key Tap

solo frame in cui viene rilevata. I dati utili relativi a questa gesture sono la la direzione e la posizione del dito nello spazio 3D.

Screen taps

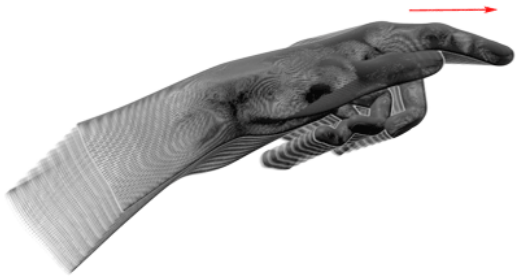


Figura 2.13: Gesture: Screen Tap

Leap Motion riconosce la gesture *Screen tap* quando un dito si muove in avanti e poi torna rapidamente indietro (figura 2.13), come se dovesse toccare uno schermo invisibile. Anche in questo caso la gesture è discreta e i dati inclusi nel frame includono la direzione e la posizione del dito nello spazio 3D.

Capitolo 3

Leap Aided Modelling in Blender

3.1 Blender: software di modellazione 3D

Blender è un famoso software open-source e multipiattaforma di modellazione 3D. Esso permette la modellazione, il rigging, l'animazione, il compositing e il rendering di immagini tridimensionali. Consente inoltre di creare applicazioni/giochi 3D ed utilizzare strumenti per la simulazioni di fluidi, rivestimenti e particelle. Blender permette di gestire una grande varietà di primitive geometriche, come le mesh poligonali e le curve di Bézier/NURBS.

Nel 1995 il programma è stato sviluppato dallo studio di animazione NeoGeo come applicazione interna. Quando nel 1998 NeoGeo fu acquistata da un'altra compagnia, Ton Roosendaal (il creatore di Blender) e Frank van Beek fondarono l'azienda NaN (Not a Number Technologies) per concentrarsi nello sviluppo approfondito di Blender, inizialmente distribuito come shareware. Nel 2002 l'azienda fallì e Roosendaal aprì una campagna di finanziamento per rendere open-source il software, la quale andò a buon fine raggiungendo più di 100.000 € di donazioni.

Script in Python

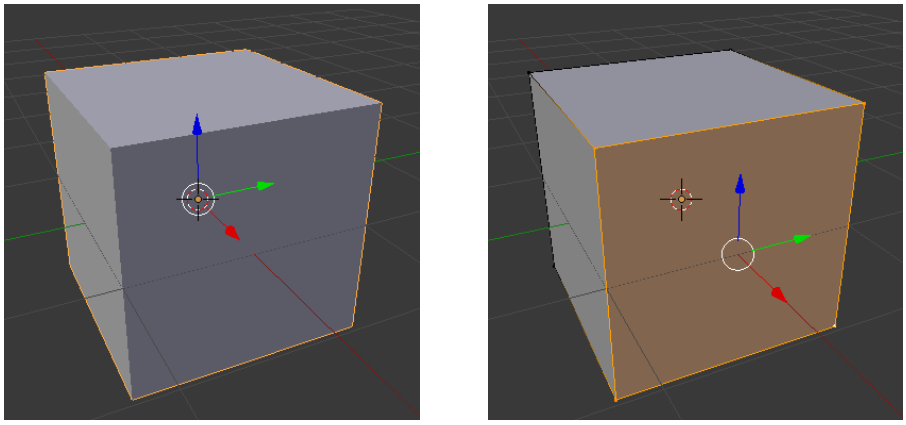
Una delle più importanti caratteristiche di Blender è la possibilità di eseguire script e addon in Python per automatizzare e controllare numerosi aspetti del programma e della scena, mettendo a disposizione del programmatore delle API ben documentate. Il programma, infatti, dispone di un interprete Python interno totalmente funzionante che permette agli utenti di aggiungere o modificare le funzionalità di Blender. Python è un linguaggio di programmazione interpretato e orientato agli oggetti che fa uso di moduli, eccezioni e gestione dinamica dei tipi. Questo linguaggio è stato espressamente progettato per essere usato come un linguaggio di estensione o di scripting per quelle applicazioni che richiedono un'interfaccia programmabile, come Blender.

Blender offre la possibilità di scrivere ed eseguire gli script in Python attraverso un editor di testo o una console dei comandi interni al programma. Inoltre è possibile anche eseguire script di Python su file esterni scritti in precedenza, decidendo se farli eseguire all'avvio di Blender o come addon da attivare a parte. Le API di Python offerte da Blender permettono di:

- Modificare ogni dato che si può modificare dall'interfaccia grafica;
- Modificare le preferenze dell'utente e il funzionamento dei pulsanti della tastiera;
- Creare nuovi strumenti (interattivi o meno) e nuovi elementi dell'interfaccia grafica (pannelli, menù, etc.);
- Creare nuovi motori di rendering per Blender.

Modalità di Blender

Le modalità di Blender sono una funzione object-oriented di livello globale, dunque il programma si può trovare in una sola modalità alla volta. Le modalità attivabili dipendono dal tipo di oggetto selezionato e ciascuna di esse permette di modificare determinati aspetti degli oggetti selezionati.



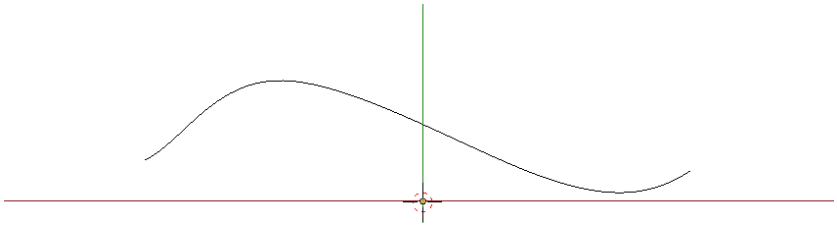
(a) Object Mode (b) Edit Mode

Figura 3.1: Modalità di Blender: visualizzazione mesh

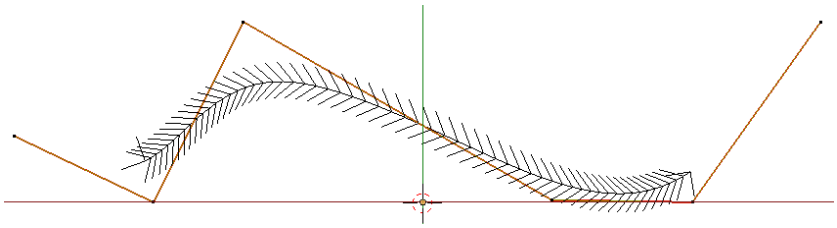
Le due modalità principali sono:

Object Mode Questa è la modalità predefinita di Blender ed è disponibile per tutti gli oggetti. Consente di modificare l'oggetto inteso come un unico blocco. Sono quindi consentite le operazioni di spostamento, rotazione e scala di un oggetto. In figura 3.1(a) si può vedere una mesh poligonale cubica selezionata in modalità Object Mode, mentre in figura 3.2(a) si può vedere una curva NURBS selezionata sempre nella stessa modalità.

Edit Mode Questa è una modalità disponibile per tutti gli oggetti renderizzabili, come le mesh poligonali, le curve e le superfici. Permette di definire la forma di un oggetto andando a modificare le parti che lo compongono (i vertici/lati/facce per le mesh, i punti di controllo per le curve e le superfici, etc.). Dopo aver selezionato un oggetto in Object Mode è possibile passare alla modalità Edit Mode (e viceversa) premendo il tasto *Tab* sulla tastiera. In figura 3.1(b) è illustrata la mesh poligonale cubica in modalità Edit Mode, dove è possibile selezionare uno o più vertici (in questo caso è stata selezionata una faccia) e modificarli. In figura 3.2(b) si può vedere la visualizzazione in Edit Mode della stessa curva NURBS del punto precedente ed in questo caso è



(a) Object Mode



(b) Edit Mode

Figura 3.2: Modalità di Blender: visualizzazione curva NURBS

possibile spostare i punti di controllo (non visibili in Object Mode) per modificarne l'aspetto.

Esistono anche altre modalità secondarie, attivabili solo con oggetti di tipo mesh: *Sculpt Mode* (strumenti per la scultura tridimensionale), *Vertex Paint Mode* (consente di definire i colori dei vertici), *Texture Paint Mode* (consente di dipingere la texture direttamente sul modello), *Weight Paint Mode* (consente di definire il peso di gruppi di vertici), *Particle Mode* (consente di modificare sistemi di particelle) e *Pose Mode* (consente di mettere in posa una *armature*).

3.2 Progetti esistenti di LAM

Per capire meglio le funzionalità di Leap Motion e Blender e per valutare i loro possibili metodi di interazione, è stato fatto fatto uno studio sui software di modellazione già esistenti di Leap Aided Modelling, valutandone i relativi pregi e difetti.

Freeform

Figura 3.3: Freeform (Sculpting)

Freeform (chiamata anche Sculpting, rif. [11]) è una applicazione ufficiale presente sull'App Store di Leap Motion che permette di sfruttare i sensori del dispositivo per lo sculpting di una sfera 3D. Attraverso dei comodi menù circolari è possibile selezionare il materiale, il colore, lo sfondo, la velocità di rotazione della sfera e gli strumenti di modifica. I progetti possono essere salvati come file .ply, .stl o .obj.

Questa applicazione, pur discostandosi dall'obiettivo principale del progetto di questa tesi, mostra in modo abbastanza chiaro le potenzialità di Leap Motion come strumento alla modellazione 3D.

BlenderBQ

BlenderBQ (rif. [12]) è una interfaccia open-source tra Leap Motion e Blender che consente di utilizzare le principali funzioni del software di modellazione attraverso il solo utilizzo di Leap e dei comandi vocali. Questo progetto è molto interessante perché si pone l'obiettivo di sostituire completamente mouse e tastiera per dare spazio alla modellazione "a mano libera". Azioni consentite:

- Spostare un oggetto in scena;
- Scalare un oggetto in scena;

- Selezionare la prospettiva della vista (solo comandi vocali);
- Entrare nella modalità “pottery”: in questa modalità è possibile aggiungere o riuovere materia dall’oggetto con il dito indice mentre questo continua a girare (simile a Freeform);
- Entrare nella modalità “paint”: in questa modalità è possibile selezionare la tonalità del colore dell’oggetto muovendo la mano.

Leap Blender

Leap Blender (rif [13]) è un addon open-source per Blender su Windows che consente di utilizzare Leap Motion per facilitare lo spostamento dell’oggetto in scena. Azioni consentite:

- Ruotare la vista verticalmente o orizzontalmente con la mano aperta;
- Muovere lo spazio di lavoro con due pugni chiusi;
- Fare uno zoom sul centro della scena con la mano aperta;
- Muovere l’oggetto selezionato con un pugno chiuso;
- Ruotare l’oggetto selezionato verticalmente e orizzontalmente con due mani aperte.

Hand Tracking 3D Blender

Hand Tracking 3D (rif. [14]) è un altro addon di Blender che permette di utilizzare Leap Motion per lo spostamento e l’orientamento di un oggetto nello spazio. Utilizza diverse modalità di controllo (selezionabili con la tastiera) e semplifica in questo modo le gesture necessarie per ogni azione. Azioni consentite:

- Spostare l’oggetto selezionato in scena con il dito indice;
- Ruotare l’oggetto selezionato sul posto con la mano aperta;

- Traslare e ruotare contemporaneamente l'oggetto selezionato con la mano aperta;
- Ruotare i gomiti in un'armatura "MakeHuman" attraverso l'utilizzo della gesture Circle;
- Modificare la posizione delle mani del modello"MakeHuman" rispecchiando quelle dell'utente.

Valutazione finale dei sistemi esistenti

I quattro software di LAM descritti nei paragrafi precedenti presentano diversi pregi e difetti che possono essere considerati nella progettazione di un nuovo addon di interazione tra Leap Motion e Blender. In particolare Freeform e BlenderBQ, nel limite delle operazioni consentite, riescono a sostituire completamente l'utilizzo di mouse e tastiera, concentrandosi dunque sul solo utilizzo di Leap Motion. Questo può essere certamente interessante dal punto di vista didattico e sperimentale, ma nella pratica, a lungo andare, risulta essere faticoso e frustrante: mouse e tastiera sono stati creati per ridurre al minimo lo sforzo fisico richiesto per l'interazione con il sistema (le mani sono appoggiate sulle periferiche), mentre Leap Motion richiede movimenti fisicamente più impegnativi. Dunque la soluzione migliore sarebbe una cooperazione tra i dispositivi di input già presenti e Leap Motion, in modo che quest'ultimo venga utilizzato per estendere l'interazione e non per sostituirla.

Leap Blender e Hand Tracking 3D offrono spunti molto interessanti per la progettazione di un nuovo addon. Le feature implementate da entrambi gli strumenti (come lo spostamento di un oggetto o della vista utente) rappresentano il punto base dal quale partire per lo sviluppo. La differenza importante tra questi due strumenti si trova nel metodo di selezione di queste feature: mentre Leap Blender utilizza molte gesture con un'unica modalità di controllo (per ogni feature di Blender implementata esiste la relativa gesture in Leap), Hand Tracking 3D gestisce meno gesture con più modalità di

controllo (per ogni feature in Blender implementata esiste la relativa modalità selezionabile da tastiera). La soluzione ideale sarebbe trovare una via di mezzo tra queste due situazioni: per aumentare l'apprendibilità dell'utente si deve cercare di avere un minor numero possibile di gesture, senza però rendere troppo frequente l'utilizzo della tastiera.

3.3 Strumenti utilizzati per l'implementazione

Per implementare e testare l'addon di integrazione di Leap Motion in Blender sono stati utilizzati i seguenti strumenti hardware e software:

- Sistema operativo: Linux Ubuntu 14.04 x86_64
- Leap Motion (modello LM-010)
- Blender 2.74
- Python 3.4:
  - Leap Motion SDK 2.3.1 (rif. [7])
  - API di Blender per addon
  - Modulo Scipy (rif. [21])

Prima di iniziare lo sviluppo è stato opportuno seguire un tutorial su come estendere Blender (rif. [8]), imparando le basi del modulo bpy (API di Blender per Python). Successivamente è stato anche seguito un tutorial su come utilizzare le API fornite dall'SDK di Leap Motion per Python (rif. [7]).

Dopo aver imparato le basi per sviluppare indipendentemente sui due strumenti, è stato creato un addon di prova per Blender che utilizzasse l'SDK di Leap Motion. In questa fase è stata notata una incompatibilità tra le librerie offerte da Leap Motion e una versione di Python superiore alla 3.0 (richiesta da Blender 2.74). Per risolvere questo problema è stato dunque

generato un wrapper tra l'SDK di Leap Motion e Python 3.4 con SWIG (rif. [9]).

Blender non è thread-safe, dunque non è stato possibile utilizzare come addon un normale script di Python in background che stesse in ascolto su Leap Motion e che chiamasse una procedura per ogni singolo frame. Per risolvere il problema si utilizzano i *Modal Operators* di Blender (rif. [10]) con l'aiuto di un Timer. In questo modo è possibile interrogare periodicamente Leap Motion per ottenere ed analizzare ogni suo singolo frame. Si può attivare l'addon come qualsiasi altro operatore di Blender: aprendo il menu di ricerca con la barra spaziatrice e selezionando l'opzione *Leap Motion*. Per disattivare l'addon premere il tasto *ESC* sulla tastiera.

3.4 Feature di navigazione implementate

In questo paragrafo sono descritte le feature implementate per lo spostamento e la modifica degli oggetti in Blender.

3.4.1 Mapping delle coordinate e spostamento di un oggetto

Il problema inizialmente riscontrato è stato quello di fare un mapping tra le coordinate 3D di Leap Motion (figura 2.6) e quelle di Blender, per far corrispondere ad esempio il movimento di un dito a quello di un oggetto sulla scena. Per risolvere questo problema sono stati analizzati i codici delle applicazioni open-source già esistenti ed è stato seguito il relativo tutorial presente nella documentazione di Leap Motion (rif. [15]). In particolare si utilizza una classe offerta da Leap SDK chiamata InteractionBox (rif. [16]), che permette di normalizzare le coordinate delle mani all'interno di una "scatola" immaginaria costruita sopra al Leap Motion (figura 3.4). Tenendo dunque conto delle differenze tra il sistema di riferimento di Leap Motion e quello di Blender, sono riuscito a mappare la posizione del dito indice con la posizione

dell'oggetto attivo sulla scena di Blender, riuscendo quindi a spostarlo sulle tre dimensioni della scena.

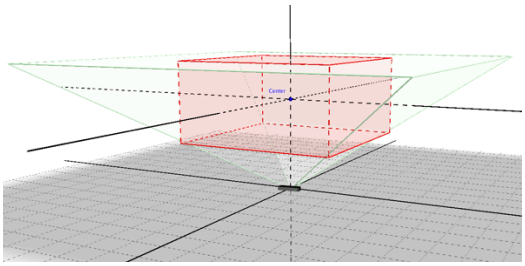


Figura 3.4: Interaction Box

Questa soluzione parziale ha fatto nascere tre diversi problemi:

1. L'oggetto si spostava solo in base alla posizione del dito in un determinato frame, non tenendo dunque conto dei movimenti precedenti. A causa di questo, ogni volta che il dito usciva e rientrava dal campo di rilevamento, l'oggetto veniva spostato direttamente alla posizione mappata, cancellando tutti gli spostamenti precedenti.

2. Lo spostamento dell'oggetto non teneva conto della vista utente. Ogni spostamento dell'oggetto veniva fatto in base alla prospettiva "Top" di Blender, creando ovviamente abbastanza confusione con rotazioni della vista differenti.

3. L'oggetto si spostava troppo lentamente rispetto alla velocità di spostamento del dito.

Per risolvere il punto 1 si calcola il vettore V_t (normalizzato e mappato) di spostamento del dito tra il frame attuale e quello precedente, facendo quindi la differenza tra i due punti normalizzati. In questo modo, sommando il vettore così ottenuto con la coordinate dell'oggetto in scena, si riescono a mappare gli spostamenti senza basarsi esclusivamente sulla posizione assoluta del dito. Chiamando D_t la posizione normalizzata del dito al tempo t si ottiene:

$$V_t = (D_t - D_{t-1})$$

Dunque, per trovare la posizione P_t dell'oggetto in scena al tempo t, si ha che:

$$P_t = P_{t-1} + V_t \tag{3.1}$$

Per risolvere il problema del punto 2 è necessario tenere conto del valore di Q_t, ovvero il quaternione di rotazione della vista utente al tempo t. Questo valore viene utilizzato per ruotare il vettore di spostamento del dito in base alla prospettiva dell'utente. Inoltre, per risolvere il punto 3, si moltiplica il vettore così ottenuto per una costante scalare k (≈ 40), data dal rapporto tra la velocità di spostamento di Blender e quella del dito nella realtà. In questo modo l'equazione (3.1) diventa:

$$P_t = P_{t-1} + k(Q_t * V_t) \tag{3.2}$$

dove $Q_t * V_t$ è la rotazione del vettore V_t in base al quaternione Q_t.

3.4.2 Modalità e selezione di un oggetto

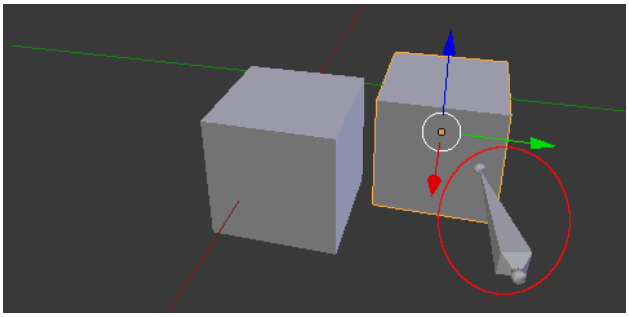


Figura 3.5: Modalità di selezione oggetto

Oltre allo spostamento dell'oggetto sulle tre dimensioni, un'altra importante feature implementata è stata quella della selezione degli oggetti in scena. Per trovare una soluzione a questo problema è stato opportuno prima dare una risposta a queste domande:

- Come capire quando l'utente vuole selezionare l'oggetto e non spostarlo?

- Cosa mostrare su schermo per dare un feedback all'utente sulla posizione virtuale del dito?
- In quale momento l'oggetto può considerarsi selezionato?

Per rispondere alla prima domanda, tenendo conto del discorso sul rapporto gesture/modalità fatto nel Paragrafo 3.2, sono state inserite tre modalità diverse e si può utilizzare il tasto Shift per passare da una modalità a quella successiva. In questa lista sono presenti le varie feature implementate (descritte nei paragrafi successivi) divise per modalità:

Spostamento Modalità di default per lo spostamento di oggetti in Blender.

- Object Mode: In questa modalità è possibile spostare o ruotare l'oggetto selezionato in precedenza, verticalmente o orizzontalmente. Per spostare l'oggetto si usa il dito indice della mano destra e per ruotarlo si usa la mano destra chiusa.
- Edit Mode: Dopo aver selezionato uno o più vertici (anche segmenti o facce di una mesh) in questa modalità è possibile spostarli o ruotarli. Per spostare un insieme di vertici si usa il dito indice della mano destra, per ruotarlo si usa la mano destra a pugno chiuso.

Selezione Modalità per la selezione di oggetti in Blender.

- Solo Object Mode: Questa modalità consente di selezionare un oggetto sulla scena. Il puntatore appare e scompare automaticamente al cambio di modalità e per muoverlo si usa il dito indice della mano destra.

Modifica Modalità per la modifica degli oggetti in Blender.

- Object Mode: In questa modalità è possibile scalare l'oggetto selezionato in precedenza. Per scalare un oggetto si muove in profondità il dito indice della mano destra.

- Edit Mode: Questa modalità consente di scalare vertici,segmenti o facce di una mesh. Dopo averli selezionati si può applicare lo scaling muovendo il dito indice della mano destra in profondità.

Per rispondere alla seconda domanda si è scelto di far selezionare un oggetto all'utente attraverso un "puntatore" capace di muoversi nelle tre dimensioni di Blender (a differenza del puntatore del mouse che non si può muovere in profondità). Per fare questo è stato creato un oggetto in Blender simile ad un dito (oggetto "Bone") che viene automaticamente importato nella scena attuale quando l'utente passa alla modalità *Selezione*. Anche in questo caso, esattamente come nello spostamento di un oggetto, il puntatore si muove in base al mapping tra la posizione del dito indice dell'utente e le coordinate di Blender, tenendo conto della vista attuale. Per fare indicare il puntatore sempre in direzione della scena si applica una rotazione all'oggetto puntatore uguale a quella della vista utente.

Come risposta all'ultima domanda è stato deciso che una mesh viene selezionata quando il puntatore la interseca. Per fare questo si utilizza la funzione *object.closest_point_on_mesh* di Blender che, dati in input un punto P_i (punta del puntatore) e una mesh, restituisce in output il punto P_o della mesh più vicino a P_i e la normale N della faccia nella quale P_o è contenuto. Da questi dati si riesce a calcolare V:

$$V = (P_o - P_i) \cdot N$$

dunque il punto P_i è all'interno della mesh quando $V < 0$. In figura 3.5 è presente un esempio di selezione di un cubo (il puntatore è quello all'interno del cerchio rosso).

3.4.3 Rotazione di un oggetto

Nella modalità *Spostamento* (Object Mode) è possibile ruotare l'oggetto selezionato in precedenza muovendo il pugno chiuso destro in verticale o in orizzontale. Per fare questo si considera lo spostamento normalizzato della

mano da un frame a quello precedente (tenendo come punto di riferimento il palmo della mano), sugli assi X e Y di Leap. Impostando la prospettiva *Top* di Blender (quella in cui le coordinate di Leap corrispondono a quelle di Blender su schermo), si assegna ad un movimento sull'asse X Leap una rotazione attorno all'asse Y Blender e ad un movimento sull'asse Y Leap una rotazione inversa attorno all'asse X Blender.

Per applicare la rotazione all'oggetto in relazione a qualsiasi altra prospettiva di vista, deve essere tenuto in considerazione il quaternione di rotazione Q_v (ovvero la rotazione della vista attuale rispetto alla prospettiva *Top*). Chiamando Q_o il quaternione di rotazione sugli assi X e Y dell'oggetto, la rotazione finale R è data da:

$$R = Q_v * Q_o * Q_v^t$$

ovvero si applica una rotazione all'oggetto in base alla vista attuale, lo si ruota di Q_o e si riapplica una rotazione inversa a quella della vista attuale. Per ruotare l'oggetto di un quaternione di rotazione R, lo si converte prima in una matrice di rotazione e lo si aggiunge alla matrice di trasformazione dell'oggetto data da Blender.

3.4.4 Scaling di un oggetto e feature in Edit Mode

In modalità *Modifica* (Object Mode) è possibile scalare l'oggetto selezionato muovendo il dito indice della mano destra in profondità (verso lo schermo o nel senso opposto). Per fare questo si aggiunge semplicemente lo spostamento normalizzato del dito sull'asse Z (calcolato tra il frame attuale e quello precedente) alla proprietà *scale* dell'oggetto, moltiplicandolo per una costante.

Dopo aver implementato lo spostamento, la rotazione e lo scaling di un oggetto nella modalità Object Mode di Blender, l'obiettivo successivo è stato quello di portare queste feature anche nella modalità Edit Mode. In questa modalità non si lavora con le mesh poligonali intese come oggetti, ma con i loro vertici, i loro lati (edges) e le loro facce. Per riuscire a modificare le

proprietà di questi elementi è stato importato il modulo *bmesh* che interpreta lati e facce come insiemi di vertici. Utilizzando le stesse modalità per Object Mode viste nei paragrafi precedenti, si calcola la posizione, il fattore di scala e la rotazione da far assumere all'insieme di vertici selezionati. In particolare si utilizza la funzione *bmesh.ops.scale* per applicare uno scaling ai vertici passati in input e la funzione *bmesh.ops.rotate* per modificare la loro rotazione.

3.4.5 Navigazione in scena

Una funzionalità che deve essere sempre disponibile, indipendentemente dalla modalità in uso, è quella della rotazione e dello spostamento della vista utente (camera). Modificare la vista della scena, infatti, è una delle azioni più frequenti e deve essere possibile farla anche contemporaneamente alle altre azioni. Per questo motivo è stato deciso di affidare questo compito alla mano sinistra o ad una struttura esterna (attraverso l'utilizzo di marker), in modo che in entrambi i casi sia possibile modificare la vista e contemporaneamente utilizzare la mano destra (o il mouse) per interagire con gli oggetti in scena. Si può passare dall'utilizzo della mano sinistra alla struttura esterna (o viceversa) premendo il tasto P sulla tastiera.

Inoltre viene la possibilità di fare lo zoom sulla scena utilizzando la gesture *Circle* con l'indice della mano sinistra. Lo zoom viene applicato in base al senso di rotazione del dito.

Navigazione con la mano sinistra

Per ruotare la vista sugli assi X e Y si può muovere il pugno sinistro in verticale o in orizzontale. Per spostare la vista e per ruotarla sull'asse Z si usa la mano sinistra aperta. Per implementare questa meccanica si mappa la posizione del pugno (palmo della mano) per mappare gli assi di spostamento Leap X e Y rispettivamente con gli assi di rotazione Blender Y e X.

Per la rotazione sull'asse Z sono state prese in considerazione le normali al palmo della mano aperta che si usano per calcolare la rotazione tra il frame precedente e quello attuale. Chiamando N_i la normale al palmo della mano

al frame i e chiamando il vettore A il prodotto vettoriale tra N_{i-1} e N_i, allora il quaternione di rotazione Q è dato da:

$$Q = (1 + N_{i-1} \cdot N_i, A.x, A.y, A.z)$$

Normalizzando il quaternione Q si ottiene il quaternione di rotazione da applicare all'asse Z.

Navigazione con struttura esterna

Modificare la vista utente utilizzando la mano sinistra non è sempre la scelta migliore: in certi casi, ad esempio, si potrebbe preferire di avere un controllo più preciso ed intuitivo sulla scena, in modo che per raggiungere una particolare prospettiva non ci sia bisogno di tante piccole rotazioni e traslazioni consecutive. Per migliorare l'esperienza dell'utente e per sperimentare le funzionalità di Leap Motion nel campo della realtà aumentata, è stato scelto di utilizzare una struttura esterna come supporto alla rotazione della vista.

La struttura è stata progettata in modo da essere posizionata sopra al Leap Motion e alle mani dell'utente, senza però ostacolare gli altri movimenti delle mani o l'utilizzo della tastiera. La struttura può essere ruotata o spostata in tutte le direzioni e si interfaccia con il Leap Motion attraverso tre marker catarifrangenti di dimensione diversa posizionati a triangolo sotto di essa.

Per la lettura dei marker si possono elaborare le due immagini grezze (destra e sinistra) catturate dalle videocamere stereoscopiche ad infrarossi (Paragrafo 2.4), seguendo i seguenti passi per ogni frame (rif. [18]):

1. Controllare che entrambe le immagini siano state prese correttamente attraverso la proprietà *is_valid*.

2. Esaminare la matrice relativa ai pixel dell'immagine sinistra[1], applicando una soglia di luminosità all'immagine (> 250).

[1]La matrice ha una dimensione di 640x240 pixel.

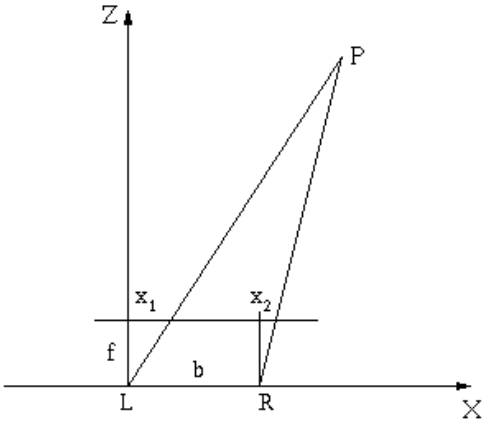


Figura 3.6: Triangolazione dei raggi

3. Per ogni pixel luminoso individuato, applicare ricorsivamente la soglia a tutti i pixel adiacenti nelle quattro direzioni, ottenendo la dimensione dell'area luminosa sull'immagine. Se la dimensione è accettabile (ovvero se è compresa tra valori definiti in base alla dimensione dei marker) allora memorizzare il punto centrale di quest'area come media dei punti rilevati e proseguire ad esaminare i pixel restanti.

4. Se sono state trovate almeno tre aree luminose corrispondenti ai marker, applicare lo stesso procedimento all'immagine destra.

5. Se il numero di aree luminose trovate nell'immagine destra è uguale a quello dell'immagine sinistra, ordinare le coppie <dimensione area, pixel centrale> per dimensione dell'area, in modo da ottenere una corrispondenza tra i marker rilevati nelle due immagini.

6. Per ogni marker presente nelle immagini calcolare il suo pixel centrale nell'immagine sinistra e quello nell'immagine destra. Utilizzare la funzione *rectify* (offerta dalle API di Leap Motion) su entrambi i punti per ottenere i coefficienti angolari verticali e orizzontali (M_L^H, M_L^V, M_R^H e M_R^V) dei raggi che partono dal centro della fotocamere e ar-

rivano ai punti sull'immagine, nel sistema di riferimento delle relative videocamere.

7. Applicare la triangolazione sui coefficienti angolari prendendo come riferimento la figura 3.6, dove:

   - L ed R sono le videocamere stereoscopiche ad infrarossi posizionate sullo stesso asse X, mentre f è la lunghezza focale di entrambe le videocamere (nel caso di Leap Motion $f = 1mm$);
   - La base delle videocamere è perpendicolare ai loro assi ottici e b è la distanza tra i loro centri (in Leap Motion $b = 40mm$);
   - XY è il piano parallelo alle immagini riprese dalle due videocamere, mentre YZ è il piano dove risiedono gli assi ottici. L'origine O è posizionata al centro della videocamera destra (L);
   - P è il punto nello spazio 3D corrispondente ai pixel rilevati sulle immagini. L'obiettivo è calcolare le coordinate del punto P.

   Allora si può applicare la triangolazione nel seguente modo. Sapendo che:

$$M^H = x/z$$

$$M^V = y/z$$

   e che:

$$R - 20 = L + 20,$$

$$Z * M_R^H - 20 = Z * M_L^H + 20$$

   Si possono dunque ottenere le coordinate (X,Y,Z) del punto P come:

$$Z = 40/(M_R^H - M_L^H),$$

$$X = Z * M_R^H - 20 = Z * M_L^H + 20,$$

$$Y = Z * M_L^V = Z * M_R^V$$

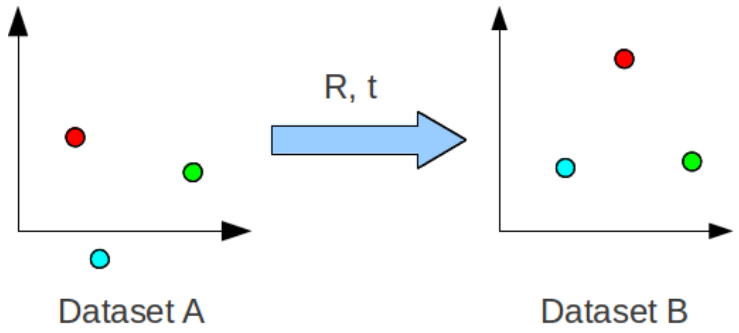


Figura 3.7: Rotazione e traslazione dei marker da un frame a quello successivo

Dopo il calcolo dei punti nello spazio 3D relativi alla posizione dei marker, si deve calcolare la loro rotazione R e loro traslazione t rispetto al frame precedente (figura 3.7). Chiamando P_i la lista degli N punti nel frame i, si cerca dunque di soddisfare la seguente equazione:

$$\forall j.\ P_i[j] = R * P_{i-1}[j] + t$$

con $j = 1..N$. Ovviamente i punti da un frame all'altro potrebbero subire più trasformazioni rispetto ad una semplice traslazione e ad una rotazione, dunque in ogni frame si deve minimizzare il valore $R * P_{i-1}[j] + t - P_i[j]$ per ogni j.

Per fare questo (rif. [19]) si seguono i seguenti passi:

1. Trovare il baricentro B_i di P_i e il baricentro B_{i-1} di P_{i-1}, dove:

$$B_i = \frac{\sum_{j=1}^{N} P_i[j]}{N}$$

2. Trovare la rotazione ottimale R utilizzando l'algoritmo *Singular Value Decomposition*(SVD) fornito dal modulo *numpy.linalg* di Python. SVD prende in input una matrice E e restituisce le tre matrici U, S, V tali che $E = USV^T$. Per trovare la rotazione ottimale si deve prima rimuovere la traslazione di entrambi gli insiemi di punti traslando i loro baricentri

nell'origine (figura 3.8). Successivamente si costruisce la matrice delle covarianze H come:

$$H = \sum_{j=1}^{N} (P_{i-1}[j] - B_{i-1})(P_i[j] - B_i)^T$$

applicando SVD ad H, ottenendo le matrici U, S, V, si calcola la matrice di rotazione ottimale R come:

$$R = VU^T$$

3. Dopo aver trovato i baricentri e la rotazione ottimale, calcolare la traslazione t come:

$$t = -R \times B_{i-1} + B_i$$

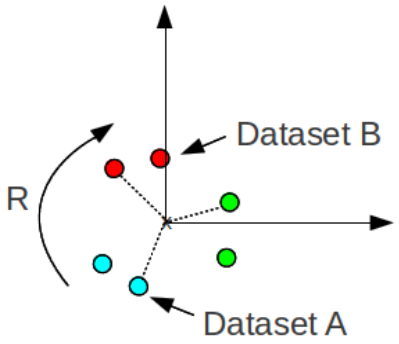


Figura 3.8: Rotazione ottimale dei punti

Alla fine di questa procedura si prendono i valori di R e t e si aggiungono alla rotazione e alla traslazione della vista. Eseguendo questa operazione ad ogni frame non ci sono cambiamenti quando la struttura esterna è immobile o quando le mani oscurano i marker sovrastanti, mentre negli altri casi lo spostamento della struttura esterna corrisponde ad uno spostamento equivalente della vista di Blender.

Per testare e sperimentare il funzionamento della struttura esterna, è stato realizzato un prototipo utilizzando materiali trovati in casa:

- Una lampada da tavolo allungabile e flessibile;

Figura 3.9: Prototipo della struttura esterna

- Tre strisce catarifrangenti prese dai giubbotti catarifrangenti salvavita;
- Un foglietto di carta colorato per mascherare le parti riflettenti della lampada.

Utilizzando questi materiali è stata costruita la struttura in foto 3.9 e in figura 3.10 sono presenti i pixel dei tre marker visti dalle videocamere destra e sinistra. Questo prototipo risulta essere abbastanza funzionante, seppur con qualche problema di precisione dovuto alla luminosità esterna[2] o a movimenti troppo veloci, ma non consente di eseguire operazioni complesse, come ad esempio la rotazione di 90 o più gradi sugli assi X e Y. I problemi di precisione derivano soprattutto dal numero di marker: più il numero di marker è alto (devono comunque essere abbastanza distanti tra di loro), più il calcolo della trasformazione è preciso. Inoltre anche la qualità e la forma dei marker influiscono sull'accuratezza del risultato: i marker catarifrangenti

[2]Il Leap Motion regola la propria sensibilità in base alla luminosità dell'ambiente esterno. I marker troppo vicini al dispositivo potrebbero essere rilevati come fonti di luce.

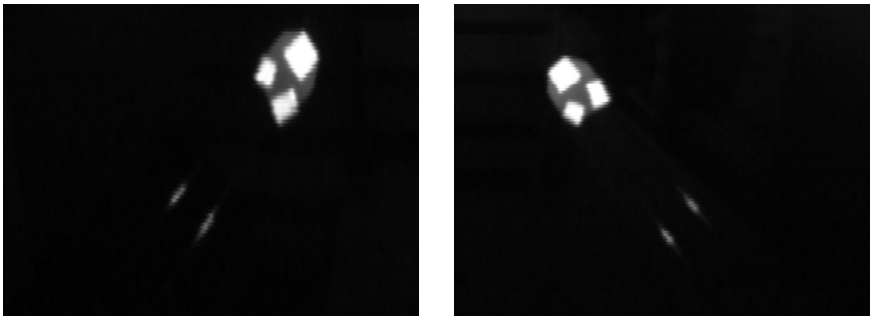


Figura 3.10: Marker visti dalle videocamere

professionali, spesso utilizzati per il motion capture, sono di forma sferica e permettono quindi di essere rilevati con poche difficoltà da ogni angolazione. Il problema della rotazione potrebbe invece essere risolto con una apposita struttura ad anello completamente mobile.

Capitolo 4

Teoria sulle curve e sulle superfici NURBS

4.1 NURBS

Le *NURBS* (Non Uniform Rational B-Splines, rif. [20]) sono una classe di curve geometriche spesso utilizzate in computer grafica per la rappresentazione di curve e superfici complesse. Grazie alla accuratezza e alla flessibilità della modellazione delle curve NURBS, esse vengono utilizzate in moltissimi processi (illustrazioni, animazioni, fabbricazione, etc.) e sono presenti nella maggior parte dei software di modellazione grafica, tra cui Blender. Le curve e le superfici NURBS possono rappresentare dunque in modo accurato sia oggetti geometrici standard (linee, cerchi, sfere, etc.) che oggetti complessi (automobili, case, corpi umani, etc.), utilizzando in entrambi i casi una quantità di informazioni sensibilmente inferiore a quella utilizzata per la rappresentazione degli stessi oggetti attraverso mesh poligonali.

Oltre all'implementazione delle interazioni tra Leap Motion e Blender discusse nel capitolo precedente, è stata dedicata una buona parte del progetto di tesi al disegno e alla modellazione di curve e superfici NURBS in Blender, utilizzando esclusivamente il dispositivo Leap Motion. In questo capitolo vengono descritte brevemente le basi e le potenzialità teoriche delle curve e

delle superfici NURBS, mentre quello successivo contiene i dettagli sulla loro gestione all'interno dell'addon realizzato.

4.1.1 Curve NURBS

Una curva NURBS è definita da tre caratteristiche:

Grado Il grado K è un numero intero positivo diverso da zero che definisce la libertà della forma della curva. Un curva NURBS di grado ad 1 equivale ad una curva polyline, i cerchi NURBS hanno grado 2 e la maggior parte delle curve NURBS più libere ha grado 3 o 5. L'*ordine* di una curva NURBS è un intero positivo pari ad $K + 1$ e definisce il numero di punti di controllo vicini che influenzano un singolo punto sulla curva.

Punti di controllo I punti di controllo P sono una lista di N punti, dove N è pari o superiore all'ordine della curva NURBS. Il modo più semplice per modificare la forma di una curva NURBS è modificare la posizione dei suoi punti di controllo. I punti di controllo *non* fanno parte della curva ma servono per manipolarla (in Blender possono essere modificati solo nella modalità Edit Mode). Ad ogni punto di controllo P_i è associato un peso w_i, un numero positivo diverso da zero che definisce la sua capacità di attrarre la curva. Quando i punti di controllo di una curva hanno tutti lo stesso peso (di solito 1), la curva viene denominata *non razionale*, altrimenti è detta *razionale* (la lettera R in NURBS indica che la curva *può* essere razionale).

Vettore nodale Il vettore nodale V di una curva NURBS è una sequenza non decrescente di $K + N + 1$ valori scalari positivi (nodi) che determinano come e dove i punti di controllo influenzano la curva. In particolare il vettore nodale divide lo spazio parametrico in intervalli che possono essere modificati da un determinato numero di punti di controllo vicini (in base a K). Il numero di volte in cui un nodo si

ripete è detto *molteplicità* del nodo e non può essere maggiore di K. La molteplicità influisce sulla continuità della curva e può forzare la creazione di angoli in curve che altrimenti sarebbero smooth. Molti software di modellazione, tra cui Blender, non permettono agli utenti di visualizzare o modificare i vettori nodali delle curve NURBS, perché questi sono utili esclusivamente per i calcoli interni.

Le curve e le superfici NURBS sono utili per le seguenti ragioni:

- Sono invarianti per trasformazioni affini: le operazioni come le rotazioni e le traslazioni possono essere applicate sia alla curva che ai suoi punti di controllo per ottenere lo stesso risultato.
- Offrono un'unica regola matematica sia per le forme analitiche che per quelle libere.
- Garantiscono la flessibilità per la modellazione di un grande numero di forme.
- Se messe a confronto con altri metodi di modellazione (es. mesh), le NURBS permettono di ridurre lo spazio di memorizzazione degli oggetti.
- Possono essere valutate abbastanza velocemente utilizzando i giusti algoritmi.

Una curva NURBS viene quindi definita come:

$$s(t) = \frac{\sum_{i=1}^{N} P_i w_i N_{i,K}(t)}{\sum_{i=1}^{N} w_i N_{i,K}(t)} \tag{4.1}$$

dove N è il numero di punti di controllo della curva, $0 \leq t \leq 1$ e $\forall i, k$. $N_{i,k}(t)$ è dato da:

$$N_{i,k}(t) = \frac{t - V_i}{V_{i+k} - V_i} N_{i,k-1}(t) + \frac{V_{i+k+1} - t}{V_{i+k+1} - V_{i+1}} N_{i+1,k-1}(t)$$

$$N_{i,0}(t) = \begin{cases} 1, \text{ se } V_i \leq t < V_{i+1} \\ 0, \text{ altrimenti} \end{cases}$$

Knot insertion

Possono essere applicate diverse trasformazioni ad una curva NURBS. Ad esempio se una curva di grado K ha N punti di controllo, si può definire la stessa curva con lo stesso grado e $N + 1$ punti di controllo. Per fare questo si tiene in considerazione la lunghezza M del vettore nodale V:

$$M = K + N + 1 \tag{4.2}$$

La tecnica di *knot insertion* aggiunge un nodo al vettore nodale V (incrementando dunque M) senza modificare né il grado K della curva NURBS né la sua forma. In questo modo, dunque, per rispettare l'equazione (4.2), il numero di punti di controllo N deve essere incrementato a sua volta, aggiungendo un nuovo punto di controllo alla curva senza modificarne l'aspetto. Un nodo può essere aggiunto più volte, fino a che esso non raggiunga la sua molteplicità massima (K).

4.1.2 Superfici NURBS

Una superficie NURBS è il risultato di un prodotto tensoriale di due curve NURBS. Per essa valgono le stesse proprietà delle curve NURBS, con la differenza che utilizza una griglia bidimensionale di punti di controllo invece che un semplice vettore. In questo modo la forma parametrica viene definita su due diversi valori: u e v. Assumendo che la superficie NURBS sia il prodotto tensoriale delle due curve NURBS di ordine rispettivamente K e H e con numero di punti di controllo rispettivamente N e M, allora tenendo come riferimento la definizione 4.1 per le curve NURBS, le superfici NURBS

sono definite come:

$$S(u,v) = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M} P_{i,j} w_{i,j} N_{i,K}(u) N_{j,H}(v)}{\sum_{i=1}^{N}\sum_{j=1}^{M} w_{i,j} N_{i,K}(u) N_{j,H}(v)} \tag{4.3}$$

4.2 Surface by Curves (SbyC)

Spesso le superfici NURBS vengono costruite attraverso una interpolazione o una approssimazione di una o più curve NURBS. Esistono diverse tecniche per la generazione di superfici a partire da curve (SbyC, *Surface by Curves*) , in base al risultato che si vuole ottenere. In questo paragrafo illustro brevemente le tecniche principali di SbyC e nel Capitolo 5 parlo della loro implementazione in Blender utilizzando il dispositivo Leap Motion.

Superfici di estrusione

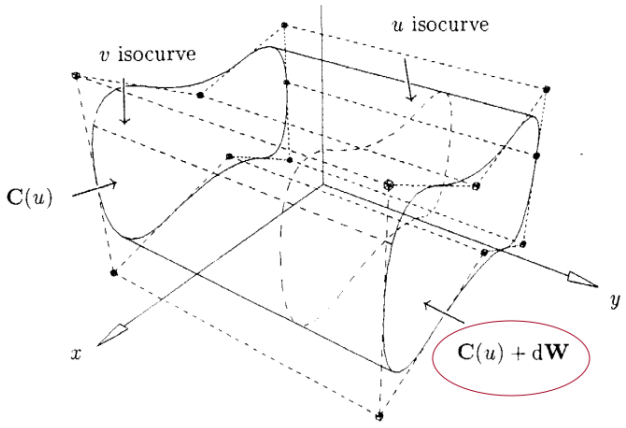
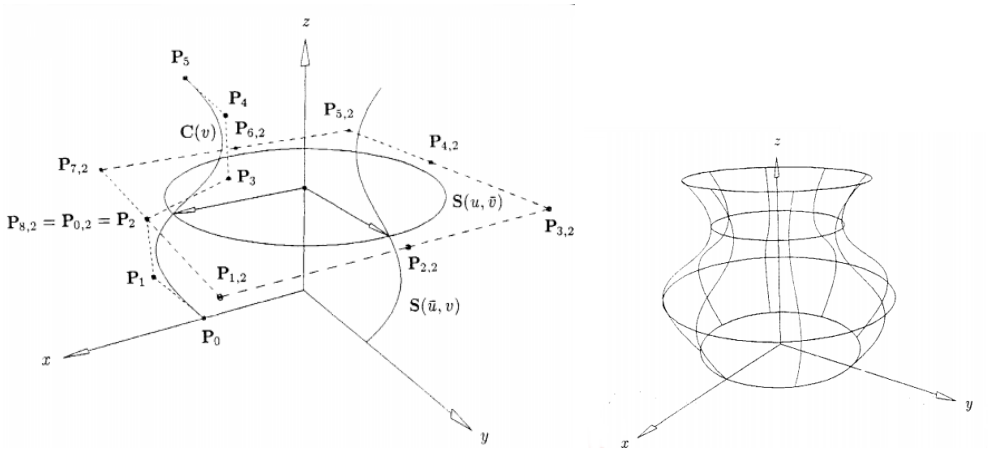


Figura 4.1: Esempio di estrusione

L'*estrusione* è una tecnica automatica di SbyC che consente di partire da una *curva spline profilo* $C(t)$ e di trascinarla per una certa distanza D in direzione W (vettore unitario), così da generare la superficie $S(u,v)$ (figura 4.1). Per un dato $\underline{u}$, la superficie di estrusione $S(\underline{u},v)$ è un segmento retto



(a) Curva profilo (b) Superficie di rotazione

Figura 4.2: Esempio di rotazione

da $C(\underline{u})$ a $C(\underline{u}) + dW$. Per un dato $\underline{v}$, invece, la superficie di estrusione è data da:

$$S(u, \underline{v}) = C(u) + \underline{v}dW = \sum_{i=1}^{N}(P_i + \underline{v}dW)N_{i,K}(u) \tag{4.4}$$

Tenendo dunque conto dell'equazione (4.4), la superficie di estrusione a partire dalla curva $C(t)$ di ordine K con N punti di controllo, per una distanza d in direzione W è data da:

$$S(u, v) = \sum_{i=1}^{N}\sum_{j=1}^{2} P_{i,j}N_{i,K}(u)N_{j,2}(v) \tag{4.5}$$

con $P_{i,1} = P_i$ e $P_{i,2} = P_i + dW$. Se la curva spline $C(u)$ è razionale con pesi w_i, allora anche $S(u, v)$ è razionale con pesi $w_{i,1} = w_{i,2} = w_i$.

Superfici di rotazione

Una superficie di *spin* (anche detta superficie di *rotazione*) di una curva spline $C(t)$ è la sua rotazione attorno ad un asse r (figura 4.2). Assumendo che $C(t)$ abbia vettore nodale V, giaccia sul piano XY e che debba essere ruotata attorno all'asse Z, allora fissando un determinato $\underline{u}$, $S(\underline{u}, v)$ diventa la curva ruotata di un certo angolo rispetto a Z. Tenendo invece fisso un certo

$\underline{v}$, $S(u, \underline{v})$ è la circonferenza che giace su un piano perpendicolare all'asse Z, ad una fissata quota $\underline{z}$ e con centro sull'asse Z.

La superficie di rotazione deve essere costruita interpolando la curva NURBS di rotazione con nove punti di controllo sul seguente vettore nodale:

$$V = \{0, 0, 0, \frac{1}{4}, \frac{1}{4}, \frac{1}{2}, \frac{1}{2}, \frac{3}{4}, \frac{3}{4}, 1, 1, 1\}$$

e con i pesi

$$W = \{1, \frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 1\}$$

Dunque la superficie NURBS di spin è:

$$S(u, v) = \frac{\sum_{i=1}^{9} \sum_{j=1}^{N} P_{i,j} w_{i,j} N_{i,3}(u) N_{j,K}(v)}{\sum_{i=1}^{9} \sum_{j=1}^{N} w_{i,j} N_{i,3}(u) N_{j,K}(v)} \tag{4.6}$$

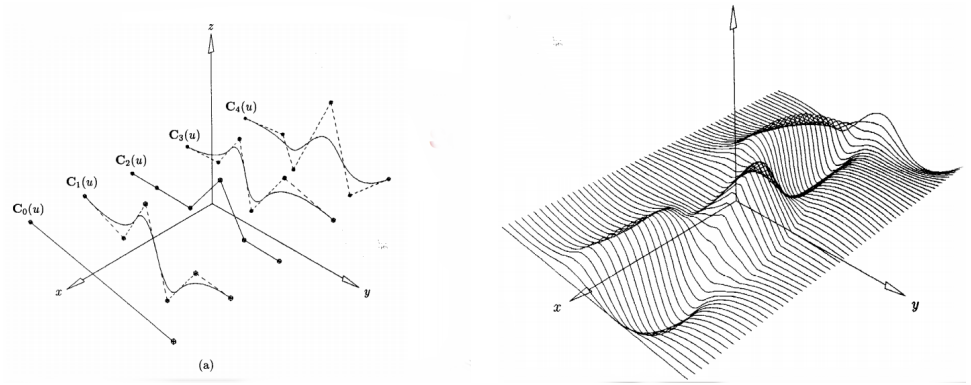
dove i punti di controllo $P_{i,j}$ si ottengono per rotazione dei punti di controllo della curva originale:

$$P_{i,j} = \begin{cases} P_j, \text{ se } i = 1 \\ P_j \text{ ruotato di 45 gradi se } i = 2, .., 9 \end{cases} \tag{4.7}$$

Superfici di skinning

L'operazione SbyC di *skinning* consiste nella generazione di una superficie per interpolazione di H curve spline ($\{C_i(t)\}$ con $i = 0, .., H$) nella direzione v della superficie da costruire (figura 4.3). Supponendo che tutte le curve abbiano lo stesso vettore nodale V, lo stesso ordine K e dunque lo stesso numero di punti di controllo N, allora l'interpolazione di una griglia di punti di controllo della superficie risultante viene fatta determinando una parametrizzazione media in u ed una in v. Una superficie di skinning è quindi data da H curve sezionali così definite:

$$C_i^w(v) = \sum_{j=1}^{N} Q_{i,j}^w N_{j,K}(v) \tag{4.8}$$



(a) Curve profilo (b) Superficie di skinning

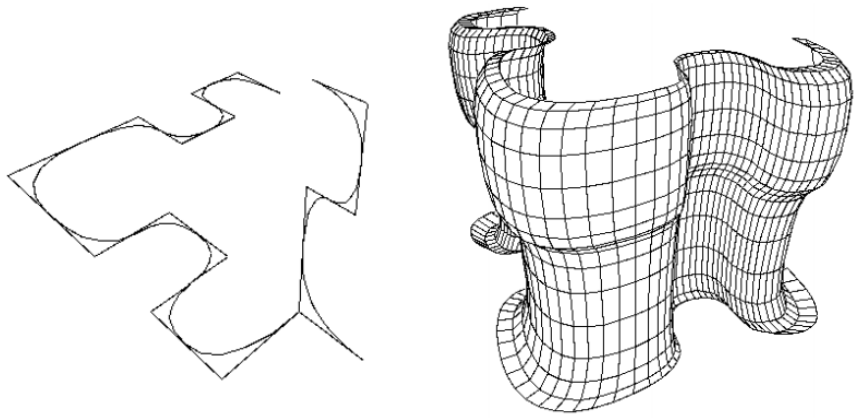
Figura 4.3: Esempio di skinning

per $i = 0, .., H$. Allora per la direzione u si sceglie un ordine K', un insieme di parametri e una partizione nodale in modo da costruire N curve di interpolazione dei punti di controllo $Q^w_{i,j}$ delle sezioni curve, ottendo così i punti di controllo $P^w_{i,j}$ della superficie di skinning finale.

Superfici di swung

Lo *swinging* è la creazione di una superficie a partire da una curva profilo $P(u)$ sul piano XZ e da una curva traiettoria $T(v)$ sul piano XY intorno all'origine, in modo da replicare la curva profilo lungo il percorso delineato dalla curva traiettoria (figura 4.4). In questo modo viene generata una superficie di swung attorno all'asse Z tale che tutte le sue sezioni u abbiano la forma della curva traiettoria $T(v)$ scalata uniformemente nelle direzioni X e Y. Tutte le sezioni v devono invece avere la forma della curva profilo $P(u)$ scalata nella direzione X. Data la proprietà di invarianza per trasformazioni affini, l'operazione di swinging può essere applicata direttamente ai punti di controllo di $P(u)$ e $T(v)$. I punti di controllo della superficie di swung $S(u, v)$ sono dati da:

$$S_{i,j} = (\alpha P^x_i T^x_j, \alpha P^x_i T^y_j, P^z_i) \tag{4.9}$$



(a) Curve traiettoria e profilo (b) Superficie di swinging

Figura 4.4: Esempio di swinging

con $w_{i,j} = w_i w_j$. Una superficie di swung può essere vista come una superficie di rotazione che interpola una traiettoria diversa da un semplice cerchio. In egual modo si può dire che se si considera una superficie di swung con una curva traiettoria che sia un cerchio unitario e $\alpha = 1$, si ottiene una superficie di rotazione dalla curva profilo.

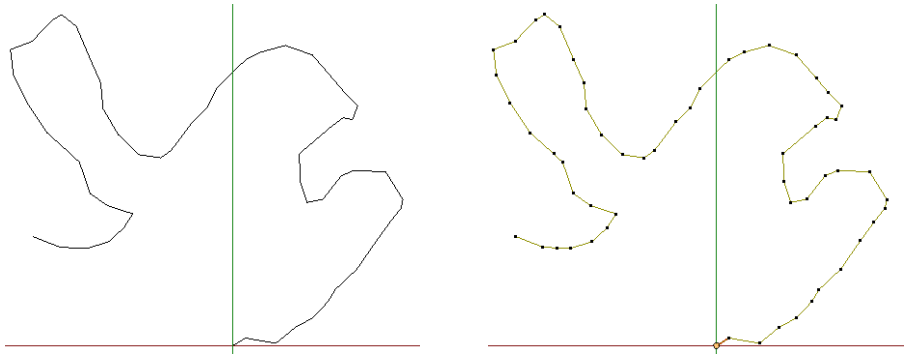
Capitolo 5

Modellazione SbyC in Blender

5.1 Disegno di curve con Leap Motion

Una parte importante dell'addon di LAM realizzato è stata quella relativa al disegno e alla modellazione delle curve e superfici NURBS in Blender utilizzando Leap Motion. Le potenzialità di questo dispositivo, come è stato descritto nei capitoli precedenti, permettono di tracciare con una buona precisione lo spostamento del dito frame per frame. Tenendo conto di questa caratteristica è stata aggiunta una nuova modalità alle tre viste in precedenza (*Spostamento*, *Selezione* e *Modifica*) per permettere all'utente di disegnare una curva direttamente in Blender utilizzando i sensori di Leap Motion. La modalità *Disegno* consente di disegnare una curva polyline (sequenza di segmenti) "a mano libera" con il dito indice, sfruttando le primitive offerte dalle API di Blender e da quelle di Leap Motion. Per garantire un tratto più preciso e per dare un migliore feedback visivo all'utente, si limita il disegno della curva alle due dimensioni (piano XY), senza quindi considerare lo spostamento in profondità del dito.

Si può entrare nella modalità *Disegno* dalla modalità *Spostamento* in Object Mode, premendo il tasto *D* sulla tastiera. Dopo la pressione del tasto si attiva automaticamente la prospettiva *Top* di Blender e si può iniziare a disegnare la curva a mano libera (il tratto parte dall'origine) spostando il dito



(a) Object Mode (b) Edit Mode

Figura 5.1: Curva disegnata a mano

indice della mano destra nello spazio sopra al Leap Motion. Lo spostamento del dito viene rilevato solo in altezza e in larghezza e viene accettato solo se viene eseguito lentamente (gli spostamenti improvvisi vengono ignorati). Si può terminare il disegno della curva ed uscire dalla modalità *Disegno* premendo di nuovo il tasto *D*. La figura 5.1 mostra una curva polyline disegnata con Leap Motion, sia in Object Mode che in Edit Mode. Per implementare il disegno della curva all'interno dell'addon subito dopo la pressione del tasto D, si seguono i seguenti passi:

1. Si crea un nuovo oggetto Blender di tipo *CURVE 2D* e lo aggiungo alla scena nella posizione dell'origine;

2. Si inizializza la curva come *POLYLINE* e si inserisce il suo primo punto sempre sull'origine;

3. Si cambia la rotazione della vista di Blender azzerando il suo quaternione di rotazione (prospettiva *Top*);

4. Per ogni vettore di spostamento normalizzato del dito indice destro, se la sua lunghezza è compresa in un certo intervallo, si aggiunge alla curva un nuovo punto dato dalla somma dell'ultimo punto aggiunto e il vettore di spostamento rilevato. La lunghezza del vettore spostamento non deve essere né troppo corta (si possono rilevare micro-spostamenti

anche quando il dito è fermo) né troppo lunga (sia per evitare errori di riconoscimento che per evitare di registrare spostamenti accidentali).

Tuttavia l'obiettivo non è quello di disegnare una semplice curva polyline, ma è quello di costruire una curva NURBS a partire da quella disegnata. La curva NURBS deve quindi *approssimare* o *interpolare*, in base a diversi parametri, la curva polyline disegnata dall'utente. Nel prossimo paragrafo viene spiegato in dettaglio questo procedimento.

5.1.1 Approssimazione e interpolazione

Blender non permette di trasformare una curva polyline in una curva NURBS in modo corretto: l'unico modo per applicare una conversione è cambiare manualmente il *tipo* di curva, da POLYLINE a NURBS (ogni punto sulla curva polyline diventa un punto di controllo della curva NURBS). Questa potrebbe essere una soluzione parziale se si considerano polyline con punti molto vicini tra di loro e si voglia costruire una curva NURBS interpolante. Tuttavia, nella maggioranza dei casi, la curva polyline disegnata manualmente non rappresenta in modo esatto la curva NURBS che l'utente voleva ottenere: il disegno a mano può essere impreciso e dei minimi errori di riconoscimento potrebbero causare delle piccole scalettature indesiderate (in molti casi si desidera ottenere una curva *smooth*).

Per risolvere questo problema è stato importato in Blender il modulo di Python chiamato *Scipy* e viene utilizzato in particolare il sottomodulo *scipy.interpolate* (rif. [21]). Questo modulo contiene una funzione chiamata *splprep()* che prendendo in input i punti di una curva, il grado e un fattore di smooth s, restituisce in output una tupla contenente il vettore nodale, i punti di controllo e il grado della curva NURBS approssimata. Le curve NURBS in Blender, come è già stato detto nel Capitolo 4, non vengono gestite attraverso il loro vettore nodale ma solo attraverso la posizione dei loro punti di controllo. Per implementare l'interpolazione (o l'approssimazione) di una curva polyline si seguono i seguenti passi:

1. Si chiama la funzione *splprep()* sui punti della curva polyline e sui parametri decisi dall'utente, così da ottenere i punti di controllo della curva NURBS;

2. Si crea un nuovo oggetto Blender di tipo *CURVE 2D* e lo si aggiunge alla scena nella posizione dell'origine;

3. Si inizializza la curva come *NURBS* e si aggiungono i punti di controllo ottenuti dal punto 1;

4. Si assegnano alla curva NURBS appena creata i parametri aggiuntivi scelti dall'utente (ordine, curva ciclica, etc.).

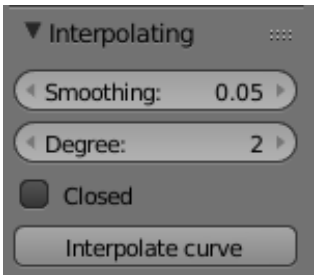
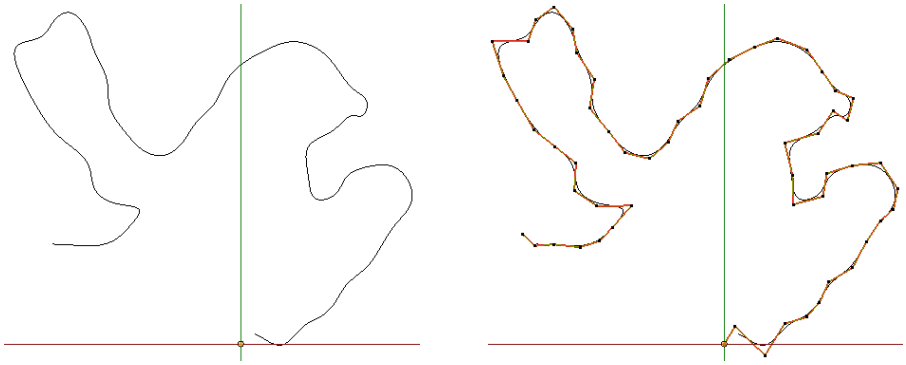


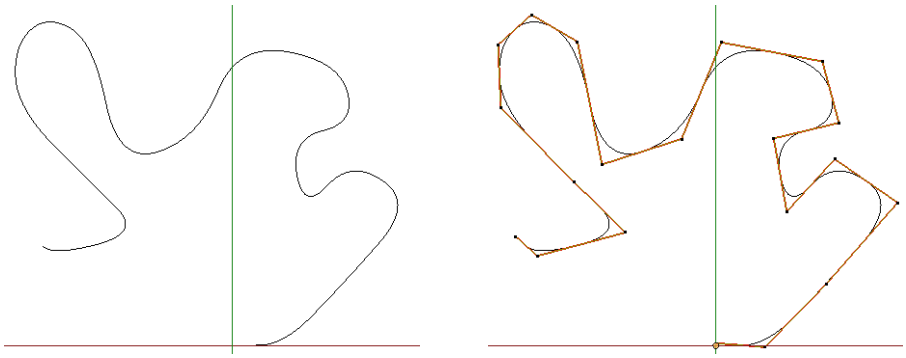
Figura 5.2: Pannello: opzioni di interpolazione/approssimazione

Viene data la possibilità di modificare i valori di interpolazione grazie ad un apposito pannello (figura 5.2) che compare tra gli strumenti di Blender quando viene selezionata una qualsiasi curva polyline 2D. All'incrementare del valore s di smooth la curva NURBS risulta più approssimata e meno fedele all'originale, mentre con un valore di s uguale a 0 viene creata una curva NURBS in modo che interpoli i punti della polyline. Inoltre all'interno del pannello è presente l'opzione *closed* che, se attivata, consente di chiudere la curva NURBS creata (Blender interpreta la chiusura come parametro aggiuntivo, non modifica i punti di controllo della curva). In figura 5.3 viene mostrato il risultato della curva in figura 5.1 dopo il processo di interpolazione, mentre in figura 5.4 è presente la stessa curva dopo il processo di approssimazione.



(a) Object Mode (b) Edit Mode

Figura 5.3: Curva NURBS interpolata



(a) Object Mode (b) Edit Mode

Figura 5.4: Curva NURBS approssimata

L'addon offre quindi due metodi diversi per creare una curva NURBS approssimata:

- In modalità *Disegno*, dopo aver disegnato la curva polyline, si può premere il tasto *D* sulla tastiera per terminare il disegno ed ottenere automaticamente una curva NURBS approssimante. I parametri di approssimazione si possono cambiare dal pannello di interpolazione.

- Selezionando una qualsiasi curva polyline 2D si può premere il pulsante *Interpolate Curve* nel pannello di interpolazione, dopo aver cambiato i parametri in base alle proprie preferenze.

5.2 Implementazione di Surface by Curves

La gestione delle curve e delle superfici NURBS di Blender non è completa o intuitiva come dovrebbe essere. In particolare, prendendo in considerazione le tecniche di Surface by Curves viste nel Paragrafo 4.2, Blender non offre la possibilità di convertire gli oggetti di tipo *CURVE* (curve NURBS) in oggetti di tipo *SURFACE* (superfici NURBS). Per ottenere gli stessi risultati (o risultati simili), ad esempio, si dovrebbe procedere in questo modo:

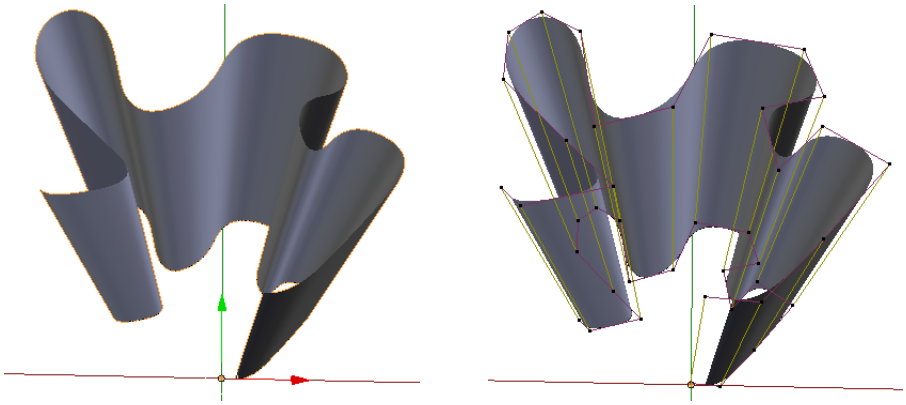
Estrusione/Skinning In Blender non è possibile applicare una estrusione o uno skinning ad una curva NURBS per creare una superficie. Tuttavia in Blender esiste un oggetto di tipo *SURFACE* di dimensione $N \times 1$ che può essere considerato una curva NURBS anche se in realtà è una superficie. Questo oggetto può subire le trasformazioni di estrusione (tasto E della tastiera) e skinning (duplicando la superficie e premendo il tasto F) in modalità Edit Mode.

Spin Non si possono creare superfici NURBS di rotazione in Blender, nemmeno utilizzando un oggetto di tipo *SURFACE*. Un metodo molto usato per ottenere un risultato simile è quello di creare la curva NURBS di rotazione, convertirla in mesh ed applicare lo spin sulla mesh così creata (questa soluzione consente però di ottenere come risultato una mesh poligonale e non una superficie NURBS).

Per questo motivo sono state implementate nell'addon le tecniche di SbyC viste in precedenza, in modo che Leap Motion possa essere utilizzato in ogni operazione del processo: dalla creazione della curva polyline fino alla trasformazione in superficie NURBS.

5.2.1 Estrusione

Considerando l'equazione (4.5), è stato creato un metodo per ottenere una superficie NURBS di estrusione dalla curva NURBS approssimata ottenuta dal procedimento nel Paragrafo 5.1.1. I parametri principali dell'estrusione



(a) Object Mode (b) Edit Mode

Figura 5.5: Superficie NURBS creata per estrusione

sono la lunghezza L e la suddivisione S (numero di repliche della poligonale di controllo della curva originale, di default $S = 2$ ed $S \geq 2$). Considerando che la curva NURBS 2D disegnata con Leap Motion giace sul piano XY, l'estrusione sull'asse Z (figura 5.5) viene implementata in questo modo:

1. Si crea un nuovo oggetto Blender di tipo *SURFACE 3D* e lo si aggiunge in scena nella stessa posizione della curva NURBS (rif. [17]);

2. Si inizializza la superficie come *NURBS* ed si aggiungono S poligonali di controllo dove:

   - La prima poligonale è uguale a quella della curva NURBS originale;
   - Ogni altra poligonale è uguale a quella precedente translata di $\frac{L}{S-1}$ sull'asse Z;

3. Si uniscono le poligonali con la funzione *make_segment()* contenuta nelle API di Blender (equivalente del tasto F sulla tastiera).

Anche in questo caso l'utente può scegliere se estrudere una curva NURBS (anche 3D) manualmente oppure utilizzare Leap Motion per estrudere la curva appena disegnata. Esistono dunque due metodi differenti per estrudere una curva:

- In modalità *Disegno*, dopo aver disegnato la curva polyline, si può premere il tasto *E* sulla tastiera per approssimare la curva disegnata (con i parametri del pannello di interpolazione) ed iniziare l'estrusione su questa nuova curva NURBS. Dopo aver premuto il tasto[1], alzando e abbassando il dito indice destro sopra al Leap Motion si cambia l'altezza della superficie (asse Z). Per terminare l'estrusione si deve premere di nuovo il tasto *E* (questo comporta l'uscita dalla modalità *Disegno*).

- Dopo aver selezionato una curva NURBS (anche 3D) compare tra gli strumenti di Blender il pannello di estrusione (figura 5.6). Dopo aver scelto la lunghezza dell'estrusione sull'asse Z e la suddivisione, si può estrudere la curva premendo sul tasto *Extrude curve*.

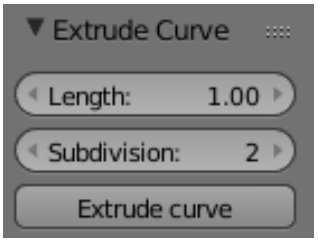


Figura 5.6: Pannello: opzioni di estrusione

5.2.2 Spin (Rotazione)

Per implementare in Blender le superfici NURBS di rotazione, sono state analizzate le equazioni 4.6 e 4.7, confrontandole con le funzioni offerte dalle API di Blender. Considerando che la curva NURBS disegnata dall'utente giace sul piano XY, è stata aggiunta all'addon la possibilità di creare superfici NURBS di spin ruotando di 360 gradi le curve NURBS 2D e tenendo X come asse di riferimento.

[1]Dopo la pressione del tasto *E* la vista utente ruota automaticamente di 45 gradi sull'asse X per facilitare l'estrusione.

Un aspetto di cui si è dovuto tenere conto è la differenza che esiste in Blender tra *oggetto* (visualizzabili in Object Mode) e *dati dell'oggetto* (visualizzabili in Edit Mode). In particolare, per quanto riguarda le curve, la posizione e la rotazione dell'oggetto non influisce sulla reale posizione e rotazione dei suoi punti di controllo. I punti di controllo della curva usano un sistema di riferimento locale che ha come origine la posizione dell'oggetto curva. Dunque prima di utilizzare lo spinning sulla curva, si devono applicare le dovute trasformazioni di rotazione e traslazione ai suoi punti di controllo, in modo da farli corrispondere a quelli visualizzati in Object Mode.

In questo caso l'utente può ottenere superfici di rotazione diverse applicando trasformazioni alla curva NURBS 2D, oppure cambiando il parametro S di suddivisione ($S \geq 4$). Lo spinning è stato implementato in questo modo:

1. Si ruotano e si spostano i punti di controllo della curva NURBS 2D in base alla rotazione (solo rispetto all'asse Z perché la curva è sul piano XY) e la traslazione dell'oggetto curva;

2. Si crea un nuovo oggetto Blender di tipo *SURFACE 3D* e lo si aggiunge in scena nella stessa posizione della curva NURBS;

3. Si inizializza la superficie come *NURBS* e si aggiungono S poligonali di controllo dove:

   - La prima poligonale è uguale a quella della curva NURBS originale;
   - Ogni altra poligonale è uguale a quella precedente ruotata di $\frac{2\pi}{S}$ rispetto all'asse X;

4. Si uniscono le poligonali con la funzione *make_segment()*;

5. Si rende ciclica la superficie per unire la prima e l'ultima poligonale di controllo.

Per creare la superficie di rotazione esistono due metodi diversi:

- Dopo aver selezionato una curva NURBS 2D appare tra gli strumenti il pannello di spin (figura 5.7) in cui è possibile impostare il parametro di suddivisione S. Una volta che la superficie è stata spostata e ruotata in base alle preferenze dell'utente, è possibile generare la superficie di rotazione premendo il tasto *Spin curve* nell'apposito pannello.

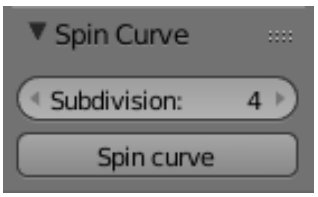


Figura 5.7: Pannello: opzioni di spin

- Utilizzando Leap Motion in modalità *Disegno*, dopo aver terminato il disegno della curva polyline e aver selezionato i relativi parametri di interpolazione, si può eseguire con il dito indice sinistro la gesture *Circle* (il senso di rotazione è indifferente). In questo modo la curva disegnata viene automaticamente approssimata in una curva NURBS 2D che viene utilizzata a sua volta per creare la relativa superficie di rotazione (dopo questa azione si esce automaticamente dalla modalità *Disegno*).

Un esempio di spinning potrebbe essere fatto sulla curva in figura 5.1. Dopo aver eseguito la gesture *Circle* con il dito indice della mano sinistra si è ottenuto il risultato in figura 5.8, ovvero la rotazione derivata dalla curva approssimata in figura 5.4.

5.2.3 Skinning

Lo skinning, a differenza delle tecniche precedenti, richiede la presenza di più curve profilo NURBS con lo stesso numero di punti di controllo, posizionate lungo una determinata direzione. Data la formula 4.8, è stato implementato lo skinning di N curve NURBS 2D ($N \geq 2$) lungo la direzione dell'asse Z in questo modo:

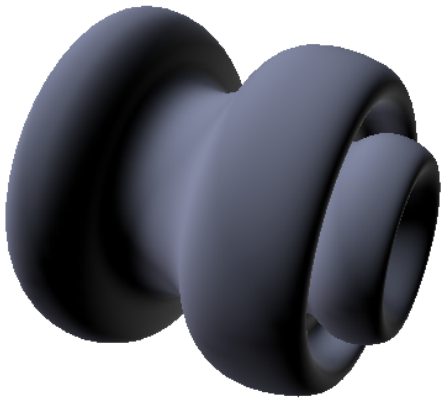


Figura 5.8: Superficie NURBS creata per rotazione

1. Si spostano i punti di controllo di ogni curva NURBS 2D in base alla traslazione del relativo oggetto curva;

2. Si ordinano le N curve NURBS in modo crescente per posizione relativa all'asse Z;

3. Si crea un nuovo oggetto Blender di tipo *SURFACE 3D* e lo si aggiunge in scena nella stessa posizione della prima curva NURBS;

4. Si inizializza la superficie come *NURBS* e si aggiungono N poligonali di controllo corrispondenti alle curve traslate e ordinate;

5. Si uniscono le poligonali con la funzione *make_segment()*.

Dopo aver selezionato un numero arbitrario di curve NURBS 2D parallele al piano XY e ordinate in base a Z, l'utente può genere una superficie di skinning cliccando il pulsante *Skin curves* nell'apposito pannello (figura 5.9) che compare tra gli strumenti. Tuttavia questa feature ha un grande limite: tutte le curve devono avere lo stesso numero di punti di controllo. Questo vincolo non permette di applicare direttamente lo skinning ad un eventuale insieme di curve disegnate direttamente con il Leap Motion, perché le curve NURBS che interpolano o approssimano le polyline hanno generalmente un numero diverso di punti di controllo.

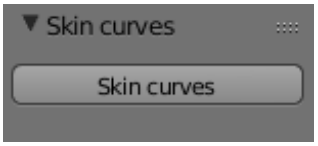


Figura 5.9: Pannello: skinning

Per risolvere questo problema si utilizza la tecnica di *knot insertion* vista nel Paragrafo 4.1.1. Supponendo di avere N curve di grado K, ognuna con il proprio vettore nodale V_i $(i = 1, .., N)$, si definisce V^u l'unione dei vettori nodali:

$$V^u = \bigcup_{i=1}^{N} V_i \tag{5.1}$$

Dopo aver trovato V^u si possono definire i V_i^D come:

$$\forall i.\ V_i^D = V^u - V_i \tag{5.2}$$

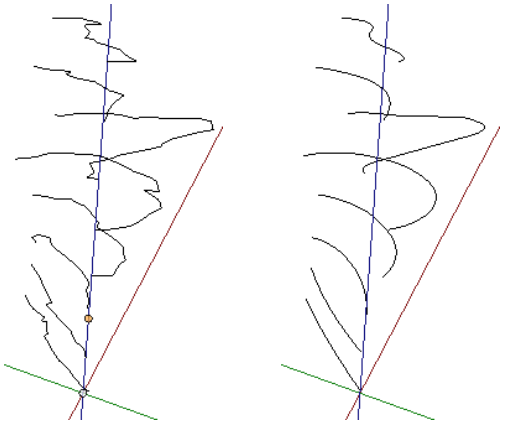
ovvero come la differenza tra l'unione dei vettori nodali di tutte le curve e i nodi presenti all'interno del vettore della curva i. Quello che si vuole ottenere è che tutte le N curve aggiungano nodi (attraverso knot insertion) al loro vettore nodale affinché risulti uguale a V^u. In questo modo, per rispettare l'equazione (4.2), tutte le curve avranno anche lo stesso numero di punti di controllo senza però modificare la loro forma.

Purtroppo, come detto in precedenza, Blender non permette di leggere o modificare i vettori nodali delle curve NURBS. Dunque non è possibile applicare il procedimento appena descritto ad un insieme di curve NURBS costruite nel modo classico. Tuttavia, utilizzando Leap Motion, è possibile disegnare le curve profilo in sequenza e memorizzare i loro vettori nodali[2]. Di conseguenza, per creare la superficie di skinning sulle curve NURBS disegnate con Leap Motion, si seguono i seguenti passi:

1. Si calcola V^u (equazione (5.1)) dai vettori nodali memorizzati;

2. Per ogni curva profilo i si calcola V_i^D (equazione (5.2));

[2]I vettori nodali sono restituiti dalla funzione *splprep()* durante l'approssimazione.



(a) Polyline (b) NURBS

Figura 5.10: Curve profilo per skinning

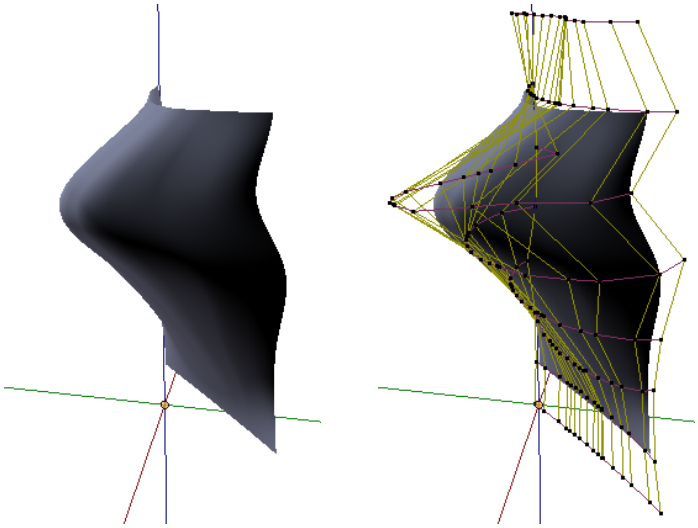
3. Si applica knot insertion sulla curva i per ogni nodo contenuto in V_i^D, utilizzando la funzione *insert()* nel sottomodulo *scipy.interpolate* ;

4. Si crea un nuovo oggetto Blender di tipo *SURFACE 3D* e lo si aggiunge in scena nella stessa posizione della prima curva profilo;

5. Si inizializza la superficie come *NURBS* e si aggiungono N poligonali di controllo corrispondenti alle nuove curve profilo create;

6. Si uniscono le poligonali con la funzione *make_segment()*.

Per generare una superficie di skinning utilizzando il Leap Motion si può entrare in modalità *Disegno* e disegnare la prima curva profilo con il dito indice destro. Si può passare alla curva profilo successiva (le curve sono equidistanti tra di loro) premendo il tasto J sulla tastiera. Dopo aver disegnato abbastanza curve (figure 5.10), premere due volte di seguito il tasto J per applicare lo skinning alle curve NURBS approssimanti delle curve polyline appena disegnate (figure 5.11). Dopo aver applicato lo skinning si esce automaticamente dalla modalità *Disegno*.



(a) Object Mode (b) Edit Mode

Figura 5.11: Superficie NURBS creata per skinning

5.2.4 Swinging

Lo swinging richiede il disegno di due curve NURBS 2D ortogonali tra di loro: una curva profilo P (con N_P punti di controllo) posizionata sul piano XZ e una curva traiettoria T (con N_T punti di controllo) sul piano XY intorno all'origine. I punti di controllo della superficie di swung devono rispettare l'equazione (4.9) vista nel Paragrafo 4.2. Come si può intuire analizzando questa equazione, i punti di controllo di P e T devono essere *diversi da zero*: ad esempio, assumendo per assurdo che un certo punto P_k abbia le coordinate $(0, 0, 3)$, allora si avrebbe:

$$\forall j.\ S_{k,j} = (0 \cdot \alpha T_j^x, 0 \cdot \alpha T_j^y, 3) = (0, 0, 3)$$

dunque la superficie risultante avrebbe N_T punti coincidenti in un'unica posizione e risulterebbe deformata rispetto alle aspettative dell'utente.

Considerando che il disegno della curva con il Leap Motion parte dall'origine (e dunque avrebbe almeno un punto di controllo con valori molto vicini allo zero), si traslano i punti di controllo delle curve durante la costruzione

(a) Polyline (b) NURBS

Figura 5.12: Curve traiettoria e profilo per swinging

della griglia di punti di controllo della superficie:

$$S_{i,j} = (\alpha(P_i^x + 1)(T_j^x + 1), \alpha(P_i^x + 1)(T_j^y + 1), P_i^z) \tag{5.3}$$

in questo modo il centro di riferimento per la curva T diventa $(-1, -1, 0)$ e quello per la curva P diventa $(-1, 0, 0)$. Per generare una superficie di swung si seguono i seguenti passi:

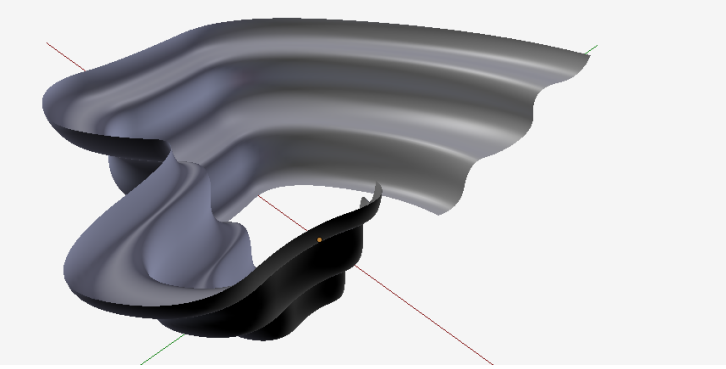
1. Si crea un nuovo oggetto Blender di tipo *SURFACE 3D* e lo si aggiunge in scena nella stessa posizione della curva traiettoria;

2. Si inizializza la superficie come *NURBS* e si aggiungono N_P poligonali di controllo, ognuna con N_T punti calcolati in base all'equazione (5.3);

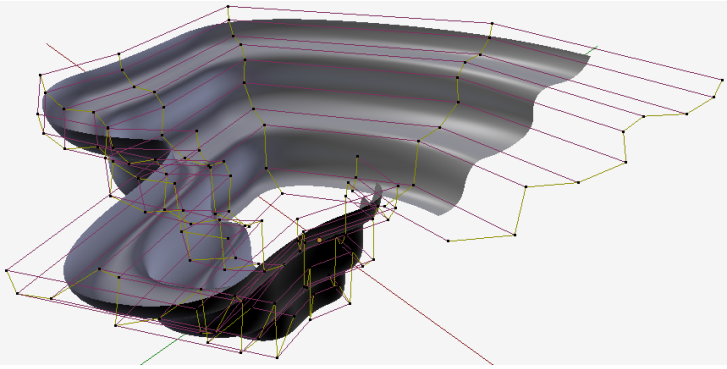
3. Si uniscono le poligonali con la funzione *make_segment()*.

Usando Leap Motion ed entrando in modalità *Disegno*, si può iniziare la creazione di una superficie di swung disegnando prima la curva traiettoria. Dopo aver terminato il disegno della curva traiettoria, premendo il tasto K sulla tastiera[3] si può iniziare a disegnare la curva profilo. Premendo ancora il tasto K si applica lo swinging alle curve NURBS approssimate (figure 5.12) per ottenere la superficie NURBS finale (figure 5.13) e si esce automaticamente dalla modalità *Disegno*.

[3]Dopo la pressione del tasto K la vista utente ruota automaticamente di 90 gradi sull'asse X per facilitare il disegno della curva profilo sul piano XZ.



(a) Object Mode



(b) Edit Mode

Figura 5.13: Superficie NURBS creata per swinging

Capitolo 6

Validazione del sistema

6.1 Semplicità e feedback

Per validare l'addon realizzato sono stati tenuti in considerazione due fattori principali: la semplicità di utilizzo e il feedback ricevuto. In questo capitolo si valuta dunque ogni feature implementata con Leap Motion in base a queste due metriche, per poi dare una validazione globale sull'implementazione di LAM in Blender[1].

La *semplicità* di utilizzo si riferisce alla comodità dell'utente (espressa in termini di tempo) di utilizzare l'addon rispetto alle feature native di Blender. Questa metrica cerca quindi di rispondere alle seguenti domande:

- Per eseguire una determinata azione è più semplice utilizzare la feature dell'addon o quella nativa in Blender?
- Utilizzando Leap Motion è possibile eseguire le stesse azioni che si possono fare utilizzando mouse e tastiera?
- Quale soluzione consente all'utente di raggiungere il risultato voluto nel minor tempo possibile?

[1]La validazione descritta nei paragrafi successivi è frutto di considerazioni personali basate sulla prova pratica dell'addon. Non sono stati effettuati test coinvolgendo altri utenti.

Si valutano dunque le feature in termini di *azioni*, considerando il modo più veloce per raggiungere un determinato obiettivo.

Il *feedback* ricevuto si riferisce alla correttezza e precisione delle soluzioni proposte dall'addon rispetto a quelle native di Blender. Questa metrica cerca di rispondere alle seguenti domande:

- Qual è lo sforzo fisico dell'utente nell'eseguire una determinata azione?
- Quanto è intuitivo eseguire una determinata azione utilizzando Leap Motion?
- Quale soluzione permette all'utente di raggiungere il risultato voluto con la maggiore precisione possibile?

Si fa quindi una analisi delle feature implementate in termini di precisione, confrontandole con le stesse operazioni effettuate in modo nativo.

6.1.1 Validazione degli strumenti di navigazione

In questo paragrafo vengono analizzate le feature di navigazione (spostamento oggetti e camera) implementate all'interno dell'addon e viste nel Capitolo 3, confrontandole una ad una con le relative funzionalità native di Blender.

Spostare oggetti

Spostare oggetti è una delle feature più usate in Blender e quindi deve essere eseguita nel modo più semplice possibile. Supponendo di partire dalla modalità *Spostamento* dell'addon, si possono fare le seguenti valutazioni:

- Spostare un oggetto utilizzando il metodo nativo di Blender (mouse) può essere fatto semplicemente tenendo premuto su di esso e trascinandolo sulla scena (due azioni). Leap Motion consente di spostare un oggetto con una sola azione: lo spostamento del dito indice.

- Quando un oggetto deve essere spostato in profondità (a causa dei movimenti su due dimensioni del puntatore del mouse) sono richieste più azioni. In questo caso, infatti, l'utente è costretto a ruotare la vista e spostare l'oggetto, oppure intervenire ripetutamente sugli assi di spostamento dell'oggetto. Con Leap Motion lo spostamento in profondità può essere fatto in una sola azione.

Questa feature risulta essere *semplice* sia nativamente che utilizzando l'addon. La soluzione di Leap Motion, tuttavia, risulta essere leggermente più veloce rispetto all'utilizzo del mouse.

La *precisione* dello spostamento dell'oggetto in aree ristrette si è rivelata migliore utilizzando il metodo nativo rispetto a quello implementato. Inoltre lo sforzo fisico fatto spostando oggetti con Leap Motion può diventare frustrante se ripetuto nel tempo. Spostare l'oggetto utilizzando il dito risulta essere comunque abbastanza intuitivo. Per questo la soluzione migliore sarebbe una integrazione tra i due metodi, utilizzando ad esempio quello nativo per gli spostamenti precisi e quello con Leap Motion per gli spostamenti in profondità.

Ruotare oggetti

La rotazione di un oggetto si effettua più raramente rispetto al semplice spostamento, ma è comunque una operazione comune. Per questo deve essere eseguita in un modo semplice e intuitivo. Supponendo sempre di partire dalla modalità *Spostamento* si può ruotare un oggetto con Leap Motion muovendo il pugno chiuso della mano destra in verticale e in orizzontale (una azione). Per ruotare un oggetto in modo nativo si deve invece cliccare su un asse di rotazione dell'oggetto e trascinarlo fino a raggiungere la rotazione desiderata (due azioni). Tuttavia l'addon non offre la possibilità di ruotare l'oggetto sull'asse Z e per rotazioni ampie sono richiesti più movimenti del pugno chiuso (aprendo la mano tra uno spostamento e l'altro). In entrambi i casi, dunque, la rotazione risulta essere una operazione abbastanza *semplice*, garantendo comunque una velocità leggermente superiore utilizzando Leap Motion per

le rotazioni semplici. Per questa feature il *feedback* dell'utente risulta essere abbastanza equilibrato: la precisione tra il metodo nativo e quello implementato è simile (per le piccole rotazioni è comunque necessario utilizzare il mouse o, ancora meglio, modificare manualmente i valori parametrici) e lo sforzo fisico fatto utilizzando Leap Motion non non è frustrante se non vengono ruotati troppi oggetti di seguito. Ruotare l'oggetto utilizzando il pugno chiuso è abbastanza intuitivo perché sembra che l'oggetto abbia delle "maniglie" da afferrare per essere ruotato. Anche in questo caso, comunque, la soluzione migliore sarebbe una integrazione tra i due metodi.

Selezionare oggetti

La selezione degli oggetti è la feature principale di Blender e può essere fatta con una sola azione (click del mouse) in Blender. Anche utilizzando il Leap Motion, in modalità *Selezione*, è possibile selezionare un oggetto con una sola azione (spostando il dito indice per muovere il cursore sopra all'oggetto scelto). Nativamente non è possibile selezionare un oggetto totalmente oscurato da un altro con una sola azione, mentre l'addon consente di selezionare anche in profondità. In questo caso la selezione nativa risulta più *semplice* di quella implementata, ma si dimostra più lenta nei casi di oggetti totalmente oscurati. La *precisione* del metodo implementato, per via dell'utilizzo di un singolo dito, risulta essere abbastanza buona e consente di selezionare anche piccoli oggetti. Selezionare oggetti con Leap Motion risulta però meno intuitivo rispetto alla classica selezione con il puntatore del mouse, anche perché quest'ultima azione viene utilizzata in quasi ogni altro software.

Scalare oggetti

Lo scaling uniforme[2] degli oggetti è una operazione eseguita più raramente rispetto alle precedenti. Dopo aver selezionato l'oggetto, con il metodo nativo

[2]Lo scaling non uniforme è presente nativamente in Blender ma non è stato implementato nell'addon.

si può effettuare uno scaling in tre azioni (click sul pulsante *Scale*, spostamento del puntatore e click del mouse per confermare), mentre utilizzando il Leap Motion, supponendo di essere in modalità *Modifica*, basta una sola azione (muovere il dito indice di profondità, senza necessità di conferma). Mettendo dunque a confronto le operazioni di scaling uniforme, quella nativa risulta essere leggermente più lenta e complessa di quella implementata con Leap Motion.

Lo scaling con Leap Motion è abbastanza *preciso* e può sostituire efficacemente l'utilizzo del mouse. Scalare l'oggetto selezionato utilizzando il dito indice si è rivelata dunque una scelta comoda ed intuitiva. Inoltre l'utilizzo di un solo dito (come nei casi dello spostamento e della selezione) consente a Leap Motion di rilevare più precisamente la sua posizione e quindi di ottenere buoni risultati.

Modificare la vista

La vista utente può essere ruotata e spostata in ogni istante ed è una operazione abbastanza frequente. In particolare, senza utilizzare l'addon, la rotazione dello scenario richiede due azioni (click di due tasti del mouse e spostamento del puntatore) e la traslazione ne richiede tre (tasto *MAIUSC* sulla tastiera, click di due tasti del mouse e spostamento del puntatore). Utilizzando Leap Motion esistono due metodi per modificare la vista:

Mano sinistra Per ruotare e traslare la vista utente con la mano sinistra serve una sola azione (spostamento della mano chiusa o aperta). In questo caso però la rotazione e lo spostamento della vista risultano essere operazioni più *semplici* con i metodi nativi rispetto all'utilizzo di Leap Motion.

Sia la traslazione che la rotazione con la mano sinistra si sono rivelati discretamente intuitivi (similmente alla rotazione di un oggetto), pur ottenendo scarsi risultati dal punto di vista della *precisione*. Spesso un cambio improvviso di gesture (tra mano aperta e mano chiusa) può causare traslazioni o rotazioni indesiderate.

Struttura esterna Utilizzando una struttura esterna con i marker è possibile traslare e ruotare la vista anche contemporaneamente e per questo entrambe le operazioni vengono considerate come una sola azione. Tenendo come riferimento una struttura fisica, questo metodo risulta essere anche più *semplice* ed intuitivo rispetto agli altri, perché si ha un modello reale come rappresentazione di quello virtuale. Inoltre la *precisione* con la struttura esterna, se realizzata adeguatamente, è migliore rispetto all'utilizzo della mano sinistra e lo sforzo fisico è praticamente nullo.

In entrambi i casi, Leap Motion risulta essere leggermente più veloce dei metodi nativi ed offre la possibilità di modificare la vista utente contemporaneamente alle altre operazioni (ad esempio allo spostamento di un oggetto), mentre nativamente questo non risulta possibile.

6.1.2 Validazione della modellazione 3D

In questo paragrafo vengono validate le tecniche di modellazione su curve e superfici NURBS viste nei Capitoli 4 e 5. In particolare, per dimostrare l'effettiva usabilità dell'implementazione di queste tecniche, vengono mostrati degli esempi di modelli creati utilizzando esclusivamente il Leap Motion, per essere poi aggiunti in una scena raffigurante un tavolo all'aperto. Il rendering finale di questa scena è mostrato all'interno del Paragrafo 6.2.

Disegnare una curva polyline

Blender offre nativamente la possibilità di disegnare una curva polyline, seppur in modo diverso da quanto è stato implementato con l'ausilio di Leap Motion. Per disegnare una curva polyline nativamente:

1. Premere su *Draw* nella scheda *Grease Pencil* del pannello degli strumenti;
2. Tenere premuto il tasto sinistro del mouse e spostare il puntatore per disegnare la curva;

3. Premere su *Convert - Polygon Curve* nella scheda *Grease Pencil*;
4. Premere su *Erase* nella stessa scheda;
5. Tenere premuto il tasto sinistro del mouse e spostare il puntatore per rimuovere la curva disegnata.

Dunque, nativamente, il disegno di una curva polyline a mano libera viene eseguito in circa sette azioni. Con Leap Motion l'operazione può essere eseguita in tre azioni (tasto *D* sulla tastiera, disegno della curva con il dito indice e tasto *D* per confermare). In entrambi i casi il disegno della curva è abbastanza *semplice*, ma il procedimento risulta essere notevolmente più veloce utilizzando Leap Motion rispetto al metodo nativo. Dopo aver terminato il disegno della curva con Leap Motion, inoltre, si crea automaticamente anche la curva NURBS approssimante.

La *precisione* ottenuta dal disegno eseguito con il mouse non è raggiungibile dal disegno della curva con Leap Motion, infatti in alcuni frame lo spostamento potrebbe essere erroneamente rilevato troppo alto (o troppo basso), portando così ad incongruenze di continuità (questi errori vengono comunque corretti durante la creazione della curva NURBS approssimante). Lo sforzo fisico nel disegno della curva con Leap Motion è ovviamente maggiore rispetto al disegno con il mouse, ma questo non influisce negativamente per curve non troppo lunghe. Disegnare la curva risulta essere intuitivo sia utilizzando il metodo nativo che quello implementato, mentre il procedimento completo della creazione della curva si è rivelato essere molto più intuitivo nel secondo caso.

Creare una superficie di estrusione

Come detto in precedenza, Blender non permette nativamente di creare superfici NURBS di estrusione partendo da curve NURBS o polyline. Si può comunque creare una superficie di estrusione partendo da una "superficie NURBS piana" in circa cinque azioni dopo aver creato la curva (tasto *Tab* per passare in Edit Mode, tasto *A* per selezionare tutti i vertici, tasto

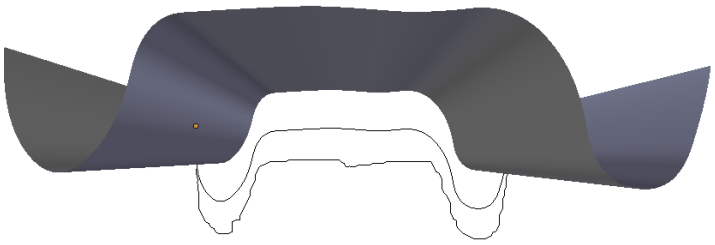


Figura 6.1: Superficie di estrusione: Vassoio

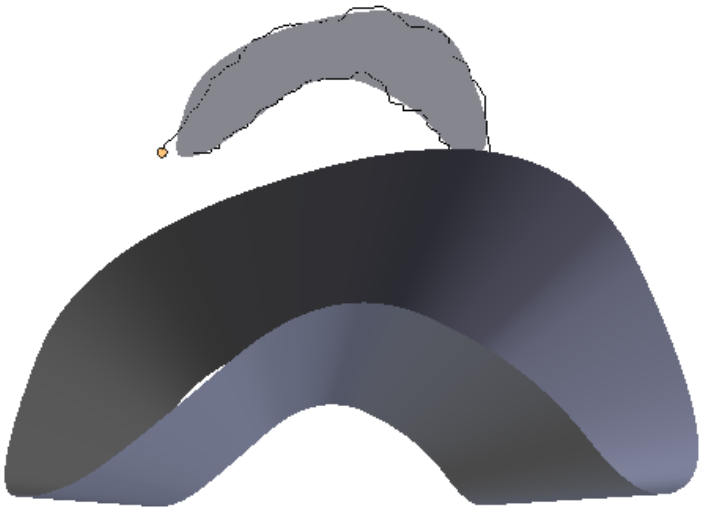


Figura 6.2: Superficie di estrusione: Schienale della sedia

E per applicare l'estrusione, spostamento del cursore e tasto di conferma). Con l'addon, invece, è possibile applicare una estrusione parametrica sulle curve NURBS dal pannello di estrusione in circa tre azioni (modifica dei due parametri e conferma). Partendo da una curva polyline ed utilizzando Leap Motion si può creare una superficie di estrusione in tre azioni (tasto *E* per applicare l'estrusione, spostamento del dito indice e tasto *E* per confermare). In questo caso, dunque, Leap Motion risulta essere l'alternativa più *semplice* e più veloce rispetto alla soluzione nativa di Blender.

I due metodi di estrusione hanno lo stesso livello di *precisione*, ma quello implementato risulta essere il più intuitivo (con un minimo sforzo fisico). Inoltre solo le soluzioni presenti nell'addon offrono la possibilità di applicare l'estrusione a curve polyline e NURBS.

Per verificare l'usabilità della estrusione sono state modellate due superfici con Leap Motion rappresentanti un vassoio per reggere le bibite, in figura 6.1, e lo schienale di una sedia (da una curva NURBS chiusa), in figura 6.2. La creazione di questa superficie è stata molto semplice e si è ottenuto il risultato voluto in pochi secondi.

Creare una superficie di rotazione

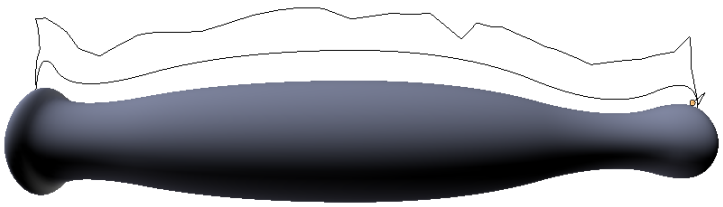


Figura 6.3: Superficie di rotazione: Gamba del tavolo

Utilizzando le funzioni native di Blender non è possibile creare superfici NURBS di rotazione a partire da una curva. In alternativa è possibile creare una mesh poligonale di rotazione applicando lo spin ad una conversione in mesh della curva, ottenendo ovviamente un risultato diverso rispetto ad

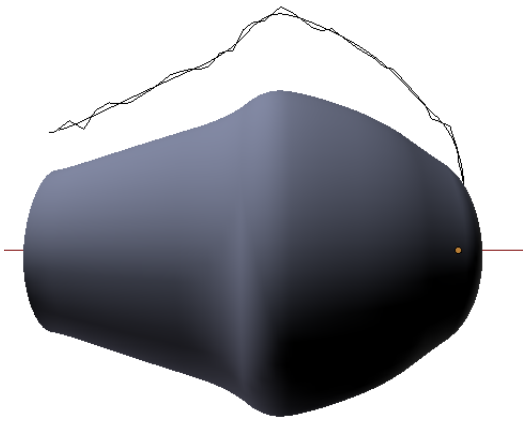


Figura 6.4: Superficie di rotazione: Caraffa

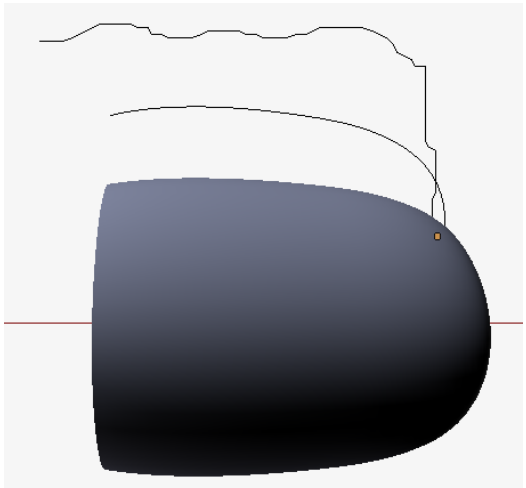


Figura 6.5: Superficie di rotazione: Bicchiere

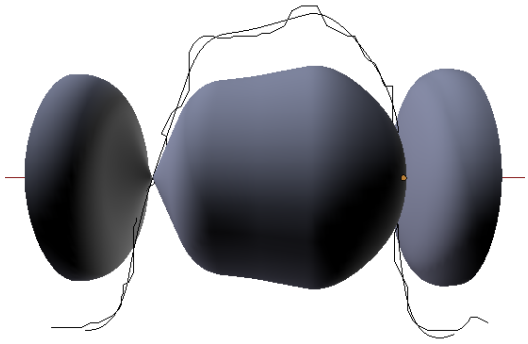


Figura 6.6: Superficie di rotazione: Caramella

una superficie NURBS di rotazione. Questa operazione comprende un grande numero di azioni (conversione della curva a mesh poligonale, inserimento dei parametri di rotazione, spinning, join dei vertici di confine, etc.) e difficilmente può essere paragonata alla creazione di una superficie NURBS di rotazione (sia da pannello che tramite gesture) di Leap Motion. Quest'ultima risulta infatti più *semplice*, *intuitiva* e veloce, pur limitandosi a rotazioni di 360 gradi sull'asse X.

Per testare il funzionamento di questa feature sono stati creati diversi oggetti per il rendering finale: le gambe per il tavolo in figura 6.3, una caraffa in figura 6.4, delle caramelle in figura 6.6 e dei bicchieri in figura 6.5. Tutti questi oggetti sono stati creati utilizzando Leap Motion e il processo ha richiesto pochi secondi per oggetto.

Creare una superficie di skinning

Lo skinning è una operazione complessa che richiede una discreta conoscenza nell'uso di Blender. Nativamente è possibile costruire una superficie di skinning solo utilizzando come curve profilo le "superfici NURBS piane", esattamente come nel caso dell'estrusione. Per farlo sono richieste circa $2C_i \cdot (N-1)+2$ azioni, dove N è il numero di curve profilo e C_i è il numero di azioni per modellare la curva i. Partendo dalla prima curva in modalità

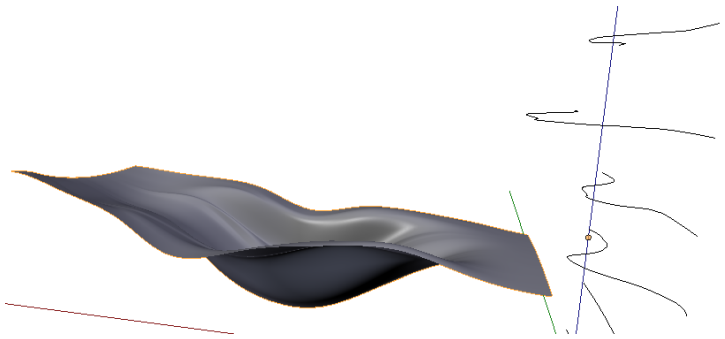


Figura 6.7: Superficie di skinning: Ciotola per caramelle

Edit Mode, per applicare lo skinning nativamente si deve ripetere il seguente procedimento $N - 1$ volte:

1. Duplicare la curva precedente;
2. Spostare il duplicato nella giusta posizione;
3. Modellare la curva (C_i azioni);

Inoltre servono altre due azioni per selezionare tutti i vertici e costruire la superficie (tasto F).

Utilizzando l'addon e Leap Motion, a differenza del metodo nativo, è possibile creare superfici di skinning partendo da curve NURBS o polyline. La costruzione della superficie utilizzando il pannello di skinning richiede circa lo stesso numero di azioni del caso nativo, mentre nella soluzione con Leap Motion è possibile costruire la superficie con circa $2(N - 1) + 2$ azioni (tasto J per ogni curva, disegno di ogni curva e due volte il tasto J per costruire la superficie). Inoltre solo utilizzando il disegno delle curve con Leap Motion si possono creare superfici di skinning su curve NURBS con un numero diverso di punti di controllo. Considerando la *semplicità* di utilizzo, lo skinning risulta essere più facile e veloce utilizzando il dispositivo Leap Motion e disegnando le curve a mano, mentre il metodo nativo e il pannello degli strumenti si sono rivelati entrambi abbastanza complessi da usare.

La *precisione* ottenuta dallo skinning nativo è più alta rispetto al metodo implementato (solo se si considera il disegno delle curve), mentre quest'ulti-

mo è sicuramente più intuitivo. Lo sforzo fisico effettuato utilizzando Leap Motion può incidere negativamente sulla valutazione con un alto numero di curve profilo.

Come esempio di superficie di skinning è stata costruita con Leap Motion una ciotola per caramelle, in figura 6.7. Il procedimento è stato abbastanza semplice e ha richiesto qualche minuto per essere portato a termine. Tuttavia, come si può osservare dal risultato ottenuto, la superficie risulta essere imprecisa e questo può essere uno svantaggio per certi tipi di oggetti. L'imprecisione è dovuta soprattutto alla difficoltà di duplicare la curva profilo disegnata precedentemente (replicare la curva a mano libera è praticamente impossibile). Questo potrebbe essere risolto aggiustando i punti di controllo della superficie dopo la sua creazione, oppure modificando le curve NURBS approssimate da quelle disegnate e generare successivamente la superficie di skinning utilizzando l'apposito pannello.

Creare una superficie di swung

Figura 6.8: Superficie di swung: Maniglia della caraffa

In Blender, nativamente, non si possono creare superfici NURBS di swung da due curve NURBS ortogonali. Si può ottenere un risultato simile allo swinging estrudendo la curva profilo lungo una curva traiettoria (sweeping), utilizzando il parametro *Bevel* nell'oggetto curva di Blender. Utilizzando

questo metodo non si crea una superficie NURBS modellabile in Edit Mode, ma si ottiene un risultato visivo molto simile. In particolare la superficie ottenuta non è altro che l'oggetto della curva traiettoria collegato all'oggetto della curva profilo, dunque modificando la curva profilo si modifica visivamente anche la superficie. Questa operazione è comunque troppo differente dallo swinging per poter fare un paragone corretto tra la funzione nativa e quella implementata.

Utilizzando Leap Motion è possibile creare una superficie di swung, partendo dalla curva traiettoria, in circa tre azioni (tasto K, disegno della curva profilo e tasto K per confermare). Questo procedimento risulta quindi essere *semplice* e veloce, restituendo come risultato una superficie NURBS di swung derivata dalla curva traiettoria e dalla curva profilo (ma indipendente da esse). La *precisione* dello swinging con Leap Motion dipende dalla precisione dei due disegni (sicuramente meno precisi rispetto alla modellazione della curva con il mouse), mentre lo sforzo fisico del disegno di due curve è accettabile. La creazione della superficie è guidata ed intuitiva: la prospettiva della vista cambia in base alla curva disegnata e bastano due disegni per costruire la superficie.

Per verificare la semplicità di utilizzo e il feedback ricevuto dalla funzione implementata nell'addon, è stata realizzata una superficie di swung con Leap Motion rappresentante la maniglia della caffettiera costruita in precedenza. Il processo per generare questa superficie (in figura 6.8) ha richiesto circa un minuto ed è stato abbastanza semplice ed intuitivo.

6.1.3 Rendering di validazione

Per mostrare la validità della modellazione di curve e superfici NURBS 3D con Leap Motion, è stato realizzato il rendering in figura 6.9, che rappresenta un prato con un tavolo, delle caramelle, dei bicchieri e una caraffa. In questo semplice rendering sono state inserite tutte le superfici NURBS viste nei paragrafi precedenti e le uniche mesh poligonale presenti sono il ripiano del tavolo e il pavimento. La scena raffigurata è stata costruita utilizzando anche

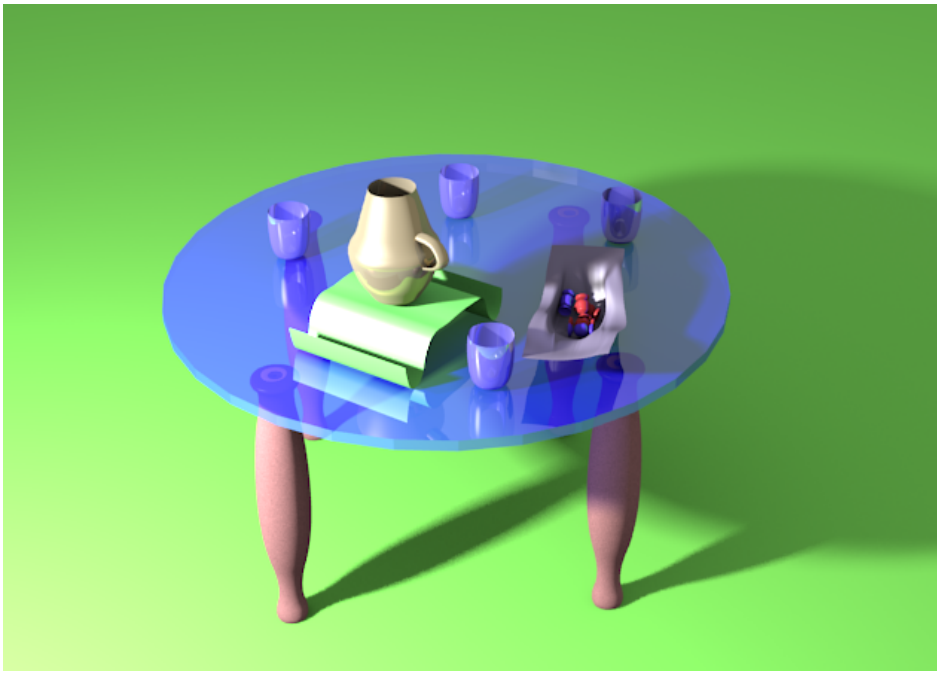


Figura 6.9: Rendering finale: scena all'aperto

le feature di navigazione con Leap Motion offerte dall'addon, soprattutto quelle dello spostamento, della rotazione e dello scaling degli oggetti.

Successivamente è stata realizzata un'altra scena con gli stessi oggetti inseriti in una stanza con specchi e sedie, in figura 6.10. Per costruire le sedie sono state utilizzate la superficie di rotazione in figura 6.3 per le gambe e quella di estrusione in figura 6.2 per lo schienale.

6.2 Validazione globale

Dopo aver validato ogni feature sia dal punto di vista della semplicità di utilizzo che da quello del feedback ricevuto, si può passare a fare una analisi riassuntiva dell'addon realizzato, in modo da valutarne in modo globale i pregi e i difetti. Nella maggior parte delle funzioni analizzate si è visto che il tempo necessario per raggiungere un determinato obiettivo utilizzando Leap Motion è sensibilmente inferiore rispetto alla modalità nativa di Blender.

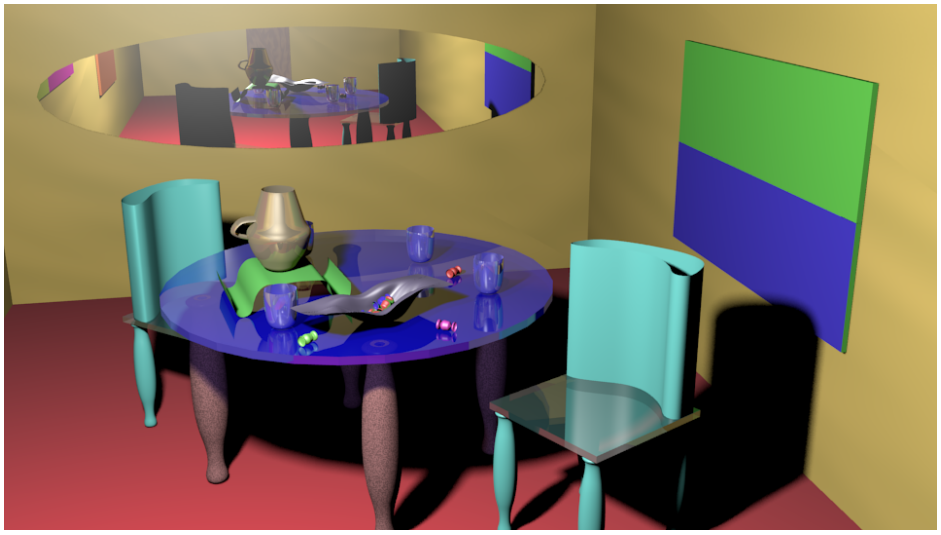


Figura 6.10: Rendering finale: scena in una stanza

Questa non è comunque una motivazione sufficiente per utilizzare le funzioni dell'addon al posto di quelle native, infatti alcune operazioni con Leap Motion risultano meno precise e/o meno complete rispetto a quelle classiche. Tuttavia esistono anche funzionalità implementate nell'addon che Blender nativamente non consente di fare, come ad esempio quelle relative alla modellazione di curve e superfici NURBS. La soluzione migliore dovrebbe essere una *integrazione* tra i due sistemi, in modo che ognuno venga usato in base alle esigenze dell'utente, senza che una opzione escluda l'altra. Ad esempio:

- Per spostare un oggetto in profondità si potrebbe utilizzare prima il dito indice con Leap Motion e poi, nel caso non sia abbastanza preciso, aggiustarlo con il mouse o con i parametri numerici dal pannello di Blender.

- Per ruotare velocemente la vista sarebbe meglio utilizzare la struttura esterna con i marker, mentre per cambiare prospettiva (*Top*, *Front*, etc.) si potrebbero utilizzare i comandi classici di Blender.

- I metodi SbyC nell'addon consentono di generare curve e superfici NURBS (dunque superfici facilmente modellabili) velocemente e in-

tuitivamente. Nel caso si vogliano creare superfici più precise, si potrebbero modificare direttamente le superfici NURBS manipolando le loro griglie di punti di controllo, altrimenti si potrebbero utilizzare gli appositi pannelli (implementati sempre nell'addon) su curve modellate con i metodi nativi di Blender. Supponendo invece che si vogliano delle superfici più complesse (ad esempio superfici di rotazione su più assi), la scelta ricadrebbe sui metodi classici di Blender discussi in precedenza (nel caso esistano), per creare però superfici di natura diversa, come le mesh poligonali.

Le funzionalità implementate *estendono* quelle native di Blender, senza quindi avere l'obiettivo di sostituirle del tutto. Inoltre la distribuzione delle feature sulle due mani (o sulla struttura esterna) consente di eseguire più azioni contemporaneamente, come ruotare la vista utente mentre si sposta o si modifica un oggetto. Si possono eseguire contemporaneamente anche azioni "ibride", come lo spostamento dello scenario con la mano destra e la selezione di un oggetto con il mouse (mano sinistra).

Capitolo 7

Conclusioni e sviluppi futuri

Il progetto discusso in questa tesi soddisfa i requisiti iniziali presentati nel Capitolo 1 e consente di utilizzare in modo efficace Leap Motion come strumento di supporto alla modellazione 3D di Blender. Lo sviluppo di questo progetto ha permesso di approfondire le mie conoscenze riguardanti la computer grafica e la programmazione in Python, in particolare nell'ambito della analisi delle immagini (Capitolo 3) e in quello della creazione di superfici e curve NURBS da curve polyline (Capitoli 4 e 5). Validando il prodotto ottenuto in base alla semplicità di utilizzo e al feedback ricevuto, esso si è rivelato essere veloce e semplice da utilizzare, anche se alcune operazioni risultano avere una precisione inferiore rispetto a quelle native di Blender (Capitolo 6). Per questo motivo, in accordo con le considerazioni iniziali, Leap Motion dovrebbe essere visto come uno strumento di supporto e di estensione alle feature già offerte da Blender, senza dunque avere l'obiettivo di sostituirle.

L'addon realizzato può essere considerato come uno scheletro dal quale partire per lo sviluppo di progetti futuri, sempre focalizzati sul concetto di LAM. Un eventuale progetto futuro potrebbe ampliare l'addon implementando, ad esempio, uno o più tra seguenti punti:

- Estensione e miglioramento delle funzionalità già implementate, concentrandosi sulla velocità, sulla semplicità e sulla precisione. Ad esem-

pio:

  - Implementare tecniche per la selezione di vertici, lati e facce;
  - Migliorare il disegno di curve polyline, consentendo di creare anche curve NURBS 3D in modo intuitivo;
  - Dare maggiore libertà nell'esecuzione delle tecniche di Surface by Curves, ad esempio permettendo all'utente di scegliere su quale asse estrudere una curva o di quanti gradi eseguire uno spin;
  - Creare una struttura esterna con più marker (possibilmente sferici) e una mobilità completa, migliorando il calcolo di rotazioni/-traslazioni della vista (Paragrafo 3.4.5).

- Aggiunta di feature riguardanti le mesh anche per altre modalità di Blender, diverse da Object Mode ed Edit Mode (Paragrafo 3.1). Ad esempio:
  - Dare la possibilità all'utente di modellare una mesh in Sculpt Mode, prendendo ispirazione dai progetti visti nel Paragrafo 3.2;
  - Consentire all'utente di colorare direttamente le texture di una mesh in Texture Paint Mode;
  - Permettere all'utente di modificare una armatura nella modalità Pose Mode, mappando le dita delle mani con gli oggetti *Bone* (in modo simile a quanto già implementato nel software Hand Tracking 3D Blender).

- Inserimento dei comandi vocali per sostituire i comandi dati da tastiera all'interno dell'addon (cambio di modalità, SbyC, etc.). Un esempio di questa estensione potrebbe essere BlenderBQ (Paragrafo 3.2).

- Implementazione di altre tecniche di Surface By Curves (swinging 3D, superfici tubolari, etc.).

- Integrazione di Leap Motion per lo sviluppo di animazioni.

- Leap Aided Modelling - Virtual Reality (LAM-VR): estensione del progetto anche per la realtà virtuale. Utilizzando *Orion* (rif. [22]) e un visore di realtà virtuale (Oculus Rift, HTC Vive o simili), si potrebbe creare un tavolo di lavoro virtuale per la modellazione 3D in Blender tramite Leap Motion (Orion garantisce una maggiore precisione e un migliore feedback rispetto al classico SDK).

Bibliografia e Sitografia

[1] Leap Motion: http://www.leapmotion.com/

[2] Leap in medicina: http://www.tedcas.com/en/node/1562

[3] Leap in automotive: https://www.engadget.com/2015/01/05/mercedes-benz-f015-luxury-in-motion/

[4] Leap in musica: http://www.theverge.com/2016/3/28/11316906/ableton-connection-kit-lego-mindstorms

[5] Microsoft Kinect: http://www.xbox.com/it-IT/xbox-one/accessories/kinect-for-xbox-one

[6] Microsoft HoloLens: https://www.microsoft.com/microsoft-hololens

[7] Leap Motion SDK documentation: https://developer.leapmotion.com/documentation/python/index.html

[8] Blender API documentation: https://www.blender.org/api/blender_python_api_current/

[9] SWIG: http://www.swig.org/

[10] Blender's Modal Operators: https://wiki.blender.org/index.php/Dev:Py/Scripts/Cookbook/Code_snippets/Interface

[11] Applicazione Freeform: https://apps.leapmotion.com/apps/sculpting/

[12] BlenderBQ: https://github.com/BlenderBQ/BBQ

[13] Leap Blender: https://github.com/gellweiler/leap-blender

[14] Hand Tracking 3D: https://slsi.dfki.de/software-and-resources/hand-tracking-for-3d-editing/

[15] Coordinate mapping: https://developer.leapmotion.com/documentation/python/devguide/Leap_Coordinate_Mapping.html

[16] Interaction Box: https://developer.leapmotion.com/documentation/python/api/Leap.InteractionBox.html

[17] Blender NURBS surfaces: http://blenderscripting.blogspot.it/2014/04/scripting-nurbs-surfaces.html

[18] Retroreflective marker tracking: https://community.leapmotion.com/t/retroreflective-marker-tracking-script-for-unity/1596

[19] Rotazione dei marker: http://nghiaho.com/?page_id=671

[20] Piegl L. e Tiller W., *The NURBS Book*, Berlino, Springer-Verlag, 1995

[21] Scipy interpolate: http://docs.scipy.org/doc/scipy/reference/interpolate.html

[22] Orion: https://developer.leapmotion.com/orion

Indice analitico