Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Informatica

Leap Aided Modelling & Animation

Tesi di Laurea in Computer Graphics

Relatore:
Prof.ssa
Serena Morigi

Correlatore:
Dott.
Flavio Bertini

Presentata da:
Stefano Beccaletto

Sessione III
Anno Accademico 2016/2017

All men dream, but not equally.
Those who dream by night in the dusty recesses of their minds,
wake in the day to find that it was vanity,
but the dreamers of the day are dangerous men,
for they may act on their dreams with open eyes,
to make them possible...

T.E. Lawrence

Introduzione

Sull'onda delle nuove tecnologie messe a disposizione sul mercato la computer grafica è diventata parte integrante di molti ambiti, professionali e non professionali, nonché oggetto di consumo come i videogiochi, montaggio di filmati, fotoritocco ed industria cinematografica. A pari passo con l'evoluzione tecnologica, anche i software di modellazione hanno subito rinnovamenti e maggior supporto, riuscendo a raggiungere una fascia di pubblico molto più ampia. Tuttavia, le competenze necessarie per il loro utilizzo vanno ben al di là di quelle in possesso all'utente medio e richiedono un'elevata formazione ed esperienza. Questo è dovuto principalmente all'interfaccia degli stessi software che risulta essere poco intuitiva e non *user friendly*, oltre che alle dinamiche laboriose che portano alla realizzazione di modelli tridimensionali, scenari ed animazioni. Inoltre, dal punto di vista dell'interazione uomo-computer, i software di modellazione devono affrontare un grande ostacolo, dettato dall'utilizzo di dispositivi di input bidimensionali (tastiera, mouse) per lavorare all'interno di un ambiente tridimensionale. Il lavoro presentato in questa tesi consiste nell'estensione di un addon per Blender che permette di utilizzare il dispositivo Leap Motion come strumento di supporto per la modifica e la modellazione di oggetti tridimensionali. Questo progetto ha l'obiettivo di accrescere l'interfaccia di Blender attraverso il Leap Motion, sfruttando i sensori e lo spazio tridimensionale d'azione del dispositivo, per facilitare ed estendere le funzionalità di navigazione, modellazione, modifica e animazione del software. Viene ripreso quindi il concetto di Leap Aided Modelling (Modellazione Assistita da Leap) affiancato al concetto di Animation

(Animazione), mirando ad ampliare le funzioni di Blender nella navigazione in scena, nella modifica e nella scultura di modelli tridimensionali, e nell'animazione attraverso il disegno di curve percorso NURBS (Non Uniform rational B-Spline) per descrivere il moto di uno o più modelli 3D all'interno della scena. Pertanto le funzioni implementate hanno l'obiettivo di velocizzare ed alleggerire le dinamiche operazionali governate tipicamente da mouse e tastiera, fornendo un supporto ulteriore al software di modellazione Blender preso in esame.

La struttura di questa tesi è organizzata nel seguente modo: nel Capitolo 1 vengono presentati gli obiettivi che ci si è posti all'inizio per il successivo sviluppo delle feautures implementate; nel Capitolo 2 viene introdotto il dispositivo Leap Motion dando delucidazioni in merito alle potenzialità che offre e agli strumenti che sono messi a disposizione degli sviluppatori, con una breve panoramica sui dispositivi simili presenti; nel Capitolo 3 viene presentato il software di modellazione Blender 3D, facendo luce sulle principali modalità presenti, con alcuni esempi sui sistemi di interazione già sviluppati tra Leap Motion e Blender; nel Capitolo 4 vengono elencati gli strumenti per lo sviluppo e le funzioni di navigazione, modellazione e modifica implementate; nel Capitolo 5 vengono esposte le basi teoriche sulle curve NURBS e sul moto lungo un percorso, descrivendo con attenzione le metodologie per definire diversi tipi di orientamento di un oggetto; nel Capitolo 6 vengono presentate le funzioni implementate nell'addon per il disegno di curve e la successiva animazione lungo di esse di uno o più modelli, seguendo le basi teoriche poste nel capitolo precedente; nel Capitolo 7 si presenta una valutazione globale per ciascuna delle feautures inserite nell'addon valutandone i pregi ed i difetti, ed infine una descrizione sul processo di creazione di un paesaggio per i test eseguiti in fase di animazione per più oggetti all'interno della scena.

Indice

Elenco delle figure

Capitolo 1

Scenario ed Obiettivi

1.1 Scenario

All'inizio degli anni Sessanta del secolo scorso, negli Stati Uniti iniziò a circolare il termine *Computer Graphics* per designare le prime applicazioni volte a riprodurre oggetti e corpi in movimento tramite un calcolatore elettronico. La Computer Grafica si diramò rapidamente a molti settori, ma solo alla fine degli anni Ottanta si potè definire tale, grazie allo sviluppo di pacchetti grafici impiegati nell'industria cinematografica e dei videogiochi che potevano beneficiare dei sempre più potenti calcolatori di uso personale.
Oggi la *Computer Graphics*, integrata alle varie tecnologie multimediali disponibili (audio, video, immagini animate, testi interattivi ecc.), è diventata uno strumento fondamentale per ogni applicazione computerizzata, ampliando la cerchia delle persone che ne usufruiscono per scopi personali ed alle grosse industrie (Pixar, Walt Disney, Dreamworks, ecc.) per dare vita alle forme più avanzate di sperimentazione artistica digitale. Per il suo sviluppo è stato senz'altro determinante, oltre alla crescente potenza di calcolo, anche lo sviluppo di innovative tecnologie per la visualizzazione grafica e parallelamente alla creazione di molteplici software di modellazione 3D. Qualsiasi tipo di modello o animazione 3D si voglia riprodurre ha bisogno di un'elaborazione grafica in Blender, Maya, Cinema4D, 3DS Max o di qualsivoglia software

grafico in commercio. Grazie all'evoluzione degli strumenti e dei software messi a disposizione, si è in grado di muovere l'occhio di uno spettatore all'interno di ambienti virtuali differenti, creare o muovere modelli all'interno di una scena a partire da trasformazioni e primitive geometriche tridimensionali. Schematicamente, il metodo di produzione di immagini o animazioni è composto da tre elementi: la creazione di modelli geometrici, la descrizione della scena che fa da cornice agli oggetti, ed il "motore di rendering" che si fa carico dei calcoli per la realizzazione del tutto.

Modellazione: La modellazione descrive il processo nel quale si definisce l'aspetto dell'oggetto o degli oggetti che si vogliono creare. I modelli tridimensionali possono essere delineati da un sistema di riferimento cartesiano tridimensionale tramite l'utilizzo di *patch* basate su spline NURBS, che approssimano curve continue alle tre dimensioni, e attraverso mesh poligonali generate da un insieme di vertici, lati e facce che ne esplicitano in maniera grezza la forma.

Scena e Animazione: Gli oggetti appena creati devono essere inseriti in una scena prima del processo di rendering. In questa fase si può costruire l'ambiente virtuale a partire da primitive geometriche, con la possibilità di scomporle o comporle assieme per formare oggetti complessi, definendone la rotazione e la dimensione nello spazio tridimensionale. Si possono aggiungere anche legami di parentela e relazioni tra oggetti, specificando le sue trasformazioni nel tempo per dar vita ad una animazione.

Rendering: Il rendering identifica il processo di conversione di un'immagine dove, a partire da modelli tridimensionali vengono prodotti frammenti (*pixels*) che ne descrivono la geometria in 2D. Durante la fase vengono aggiunte informazioni sul colore di ogni frammento in base alla tipologia di illuminazione, alle proprietà specificate per l'oggetto, e alle posizioni relative agli altri oggetti presenti in scena. Nel caso di animazioni

vengono renderizzate molteplici immagini, chiamate *Frames*, per poi essere assemblate in sequenza.

1.2 Obiettivi

I differenti software di modellazione 3D prevedono l'utilizzo dei comuni dispositivi di input disponibili, quali mouse, tastiera, touchpad o tavolette grafiche. Le operazioni di modellazione o animazione vengono sempre eseguite attraverso l'utilizzo di questi strumenti che, in alcuni casi, risultano essere dei vincoli notevoli nell'eseguire operazioni che intuitivamente si possono considerare semplici (ad esempio la navigazione libera della scena o la descrizione della traiettoria di un oggetto nell'ambiente 3D). Per lo sviluppo del progetto di tesi si è fatto riferimento al software di modellazione Blender 3D, un programma *open-source* multipiattaforma che fornisce tutti gli strumenti di base per la modellazione, animazione e rendering di immagini tridimensionali.
L'obiettivo è quello di ridurre i vincoli di interazione che si possono incontrare nella modellazione e animazione della scena 3D cercando si sopperire alle limitazioni poste dai comuni dispositivi di input. E' stato quindi utilizzato il dispostitivo di input Leap Motion (Capitolo 2) che offre un supporto di riconoscimento per le mani e le dita, se poste sopra di esso, fornendo la possibilità di gestire delle coordinate ed un input 3D. Si vuole quindi fornire un supporto aggiuntivo a quelle che sono le operazioni di modellazione e animazione, affiancando il Leap Motion ai comuni dispositivi di input 2D. Si definisce quindi la nozione di *LAM* (Leap Aided Modelling) attraverso l'implementazione di un addon [6] per Blender che inserisce ulteriori funzionalità a quelle già presenti. In particolar modo vengono inseriti tre differenti strumenti che comprendono la navigazione in scena, l'animazione di un oggetto lungo un percorso e la modellazione.

Movimento della Scena

Una delle funzionalità principali più utilizzate in una scena 3D è la modifica, attraverso la rotazione e lo spostamento, della vista utente. Precedentemente la navigazione all'interno della scena avveniva attraveso la mano sinistra sfruttano il movimento del pugno per lo spostamento lungo gli assi X, Y e la mano aperta per la rotazione e spostamento lungo l'asse Z.
Modificare la vista della scena risultava poco intuitivo ed é stato quindi deciso di rivoluzionare le pose di riferimento.

- L'obiettivo è quello di rendere più dinamico e veloce lo spostamento sfruttando tre dita della mano, una per ogni asse.

Selezione e Modifica Vertici

Una delle funzioni fondamentali molto usate durante la fase di modellazione è la possibilita di agire direttamente sui vertici che definiscono la struttura di un oggetto. Dunque uno degli obiettivi che ci si è posti è la possibilità di selezionare e modificare, a proprio piacimento, i vertici che compongono una mesh.

Animazione lungo percorso

In Blender un modo differente per spostare oggetti nello spazio 3D è costringerli a seguire un percorso descritto da una curva. L'oggetto seguirà il percorso per un certo numero di frame, che rappresentano il tempo per percorrere l'intera lunghezza della curva. Si vuole quindi offrire la possibilità di disegnare una curva di tipo *NURBS* che rappresenti il moto di un oggetto presente nella scena, inserendo diverse modalità di orientamento.
Gli obiettivi sono:

- **Disegno:** Abilitare il disegno di una curva percorso nella scena 3D ed associare l'oggetto selezionato al percorso

- **Velocità:** Associare la velocità nel disegno della curva con il Leap con la velocità di spostamento dell'oggetto lungo il percorso
- **Orientamento:** Dare la possibilità di definire l'orientamento dell'oggetto lungo il percorso.

Modellazione Sculpt

La modalita Sculpt permette di utilizzare il mouse come uno scalpello per modellare una mesh, in particolar modo permette di incidere o mettere in rilievo dei vertici utilizzando diversi pennelli. Le funzioni comuni a quasi tutti i pennelli a disposizione sono quelle di *add*, dove gli effetti hanno un'azione additiva e permettendo di creare dei rilievi, e *subtract* che rappresenta l'effetto opposto ed ha un'azione sottrattiva dando la possibilità di creare delle depressioni sulla superficie di mesh selezionata. Però, l'azione di modifica sulla mesh di ogni pennello è dipendente dalla vista utente, non agisce quindi nell'intero ambiente 3D poiché vincolata ad una vista 2D.
L'obiettivo è quello di sfruttare gli strumenti di sculpting attraverso il Leap Motion ed espanderne la funzionalità:

- **Modellazione:** Creare un'interazione con Leap che permetta di sfruttare le funzioni di *add* e *subtract* dei vari pennelli.
- **Spazio 3D:** Si deve definire il raggio d'azione per controllare la direzione di modifica dei vertici nella scena 3D, come soluzione al vincolo della vista utente.

Capitolo 2

Leap Motion Controller

2.1 Il dispositivo Leap Motion

La Leap Motion è una società di sviluppo tecnologico specializzata nel fabbricare tecnologie di rilevazione del movimento nell'interazione fra uomo e computer. Fu fondata nel 2010 da David Holz e Michael Buckwald sotto il nome di *OcuSpec*, tutto ebbe inizio per cercare di arginare la frustrazione provata con mouse e tastiera nel creare modelli 3D. Dopo un lento inizio, operando in maniera del tutto invisibile al pubblico, nel marzo del 2012 la Leap Motion annuncia lo sviluppo del loro primo prodotto, il *Leap* [1]. Questo strumento è in grado di rilevare le mani e le dita, con una precisione di 0,01 millimetri, determinandone la posizione ed offrendo una rivoluzionaria interazione con il software fornito grazie al semplice movimento di queste. Il Leap Motion è un dispositivo USB di dimensioni ridotte progettato per essere posto su di un tavolo o scrivania rivolto verso l'alto. E' dotato di 2 telecamere monocromatiche e 3 LED infrarossi che si occupano di rilevare la posizione delle mani e delle dita, o oggetti simili come penne e strumenti di puntamento, in un'area approssimativamente a forma di semisfera capovolta dal basso verso l'alto di circa un metro, con un raggio che va dai 25 ai 500 millimetri ed un'ampiezza all'incirca di 150 gradi. I LED generano una luce IR pattern-less, mentre le 2 telecamere generano quasi 200 frames di dati che

vengono analizzati dal software sintetizzando i dati di posizione 3D attraverso il confronto con i frames 2D generati dalle 2 telecamere. La Leap Motion

Figura 2.1: Leap Motion: unità ottica di tracciamento per le mani

sin dagli inizi ha messo a disposizione un SDK, in diversi linguaggi di programmazione, per lo sviluppo di applicazioni legate al Leap, offrendo pieno supporto alla comunità degli sviluppatori. I differenti software sviluppati vengono rilasciati attraverso un App Store ufficiale che contiene un gran numero di applicazioni, gratuite e a pagamento, che abbracciano diversi sistemi operativi. Il Leap risultando duttile viene usato in diversi campi come quello artistico [2], medico [3] e riabilitativo contro il mutismo [4] ed in robotica [5]. Inoltre nel Febbraio 2016 è stato rilasciato un software specifico, chiamato Orion, costruito esclusivamente per la realtà virtuale offrendo la possibilità di installare il Leap davanti al visore aumentando considerevolmente le prestazioni del dispositivo e la possibilità di interazione con esso.

2.2 Dispositivi simili

Nel corso dell'ultimo decennio sono stati rilasciati diversi dispositivi simili al Leap Motion per il riconoscimento motorio di mani e corpo. I precursori

del Leap ed i successivi imitatori abbracciano una fascia di pubblico strettamente correlata al mondo dei videogiochi su console e su PC, fornendo solo a posteriori ed in alcuni casi, il supporto necessario per essere sfruttati in settori differenti da quello nativo.

Microsoft Kinect

Il Kinect è un accessorio che ha visto la luce nel giugno 2010 ed è stato pensato per l'utilizzo con Xbox 360 e successivamente rivisitato ed aggiornato per usufruirne anche con Xbox One (Figura 2.2). Inizialmente conosciuto con il nome di Project Natal, esso si focalizza sul riconoscimento del corpo umano, dalla testa ai piedi, offrendo possibilità diverse nell'utilizzo dei videogiochi. Kinect usufruisce di una telecamera ad infrarossi ed un'altra RGB per captare il movimento del corpo, inoltre è dotato di un array di microfoni che vengono utilizzati dal sistema per calibrare l'ambiente circostante e garantiscono il riconoscimento di alcuni comandi vocali preimpostati. La periferica permette di utilizzare la console senza l'uso del joypad, riuscendo a seguire il movimento anche di 4 persone. Le potenzialità del dispositivo e la grande richiesta in differenti campi hanno spinto Microsoft a distribuire driver ed SDK per il suo pieno sfruttamento.

Figura 2.2: Microsoft Kinect: periferica per il riconoscimento del corpo

Microsoft HoloLens

HoloLens è una periferica, sviluppata da Microsft, per la realtà aumentata ed appartiene alla categoria degli *smartglasses* (Figura 2.3). Il dispositivo ha acquisito popolarità per essere il computer precursore che ha portato all'utilizzo di Windows Holographic, una piattaforma all'interno del sistema operativo Windows 10. HoloLens è composto da un sistema di misurazioni interno (accellerometro, giroscopio, magnetometro), una videocamera, un display ad alta risoluzione, un array di quattro microfoni e un sensore di luce ambientale che, associati alla grossa capacità di calcolo del processore interno HPU, permettono di interagire attraverso la voce, lo sguardo e movimenti delle mani analizzando l'ambiente circostante con la ricostruzione una mappa interna che permette di delinearne i tratti mettendo in risalto oggetti presenti e ostacoli.

Figura 2.3: Microsoft HoloLens: visore per la realtà aumentata

Playstation Move, Wiimote

Playstation Move e Wiimote sono delle periferiche sensibili al movimento svilluppate per lo scenario videoludico. Sono svilluppati rispettivamente da Sony e Nintendo per le rispettive console, con l'intenzione di crescere la partecipazione e l'esperienza del videogiocatore rendendolo parte attiva du-

rante le fasi di gioco. A differenza di Leap Motion entrambi i dispositivi sono pensati per essere tenuti in mano e sono dotati di giroscopi ed accelerometri che permettono di tracciare la velocità e la posizione delle mani in un dato momento. Per il loro corretto funzionamento hanno bisogno di un dispositivo esterno che permette di processare con precisione i movimenti eseguiti, ovvero, nel caso di Playstation Move è necessario disporre di una telecamera (Playstation Eye), come anche Wiimote sfrutta una telecamera ad infrarossi chiamata Wii Sensor bar.

(a) Ninetendo Wiimote (b) Playstation Move

Figura 2.4: Periferiche sensibili al movimento

2.3 Architettura del Sistema

La struttura del software Leap Motion prevede l'esecuzione costante in background di un servizio (su piattaforme Windows) o di un demone (su Linux e Mac OSX). Il canale di comunicazione viene stabilito attraverso un bus USB tra il controller ed il software in esecuzione. Le applicazioni che sfruttano il Leap interrogano il software ad intervalli regolari (frames) per ricevere le informazioni sul movimento effettuato dalle mani.

L'SDK di Leap Motion fornisce due metodi differenti per il tracciamento dei dati: un'interfaccia WebSocket ed un'interfaccia Nativa (Figura 2.5). Le API messe a disposizione coprono un'ampia cerchia di linguaggi di programmazione, che permettono di creare applicazioni Leap-oriented. Nei linguaggi di programmazione sono inclusi: C#, JavaScript, Unity, Unreal, C++, Java, Objective-C e Python.

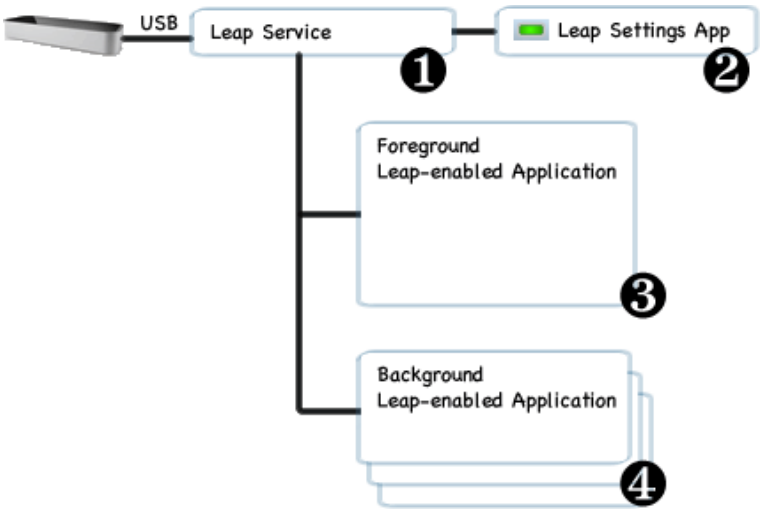


Figura 2.5: Architettura del Leap Motion (Interfaccia Nativa)

Interfaccia Nativa

L'applicazione dell'intertfaccia Nativa è offerta tramite il caricamento dinamico della libreria di supporto. Si può creare il link di comunicazione con qualsiasi dei linguaggi di programmazione sopra elencati, provvedendo alla connessione ed il flusso di dati tra il servizio di Leap Motion e l'applicazione. I passi fondamentali vengono esplicitati di seguito e sono illustrati in Figura 2.2.

1. Il servizio di Leap Motion riceve i dati dal dispositivo attraverso il bus USB, i quali vengono prima processati e poi spediti alle API interne alle applicazioni. Di default i dati di tracking vengono spediti alle applicazioni foreground. Tuttavia, nel caso fosse necessario, i dati possono

essere richiesti e passati anche alle apllicazioni in background. Questa richiesta deve essere specificata direttamente.

2. Attraverso un'applicazione che fornisce un pannello di controllo, separato dal dal servizio di Leap Motion, si può modificare e configurare il proprio dispositivo in molteplici aspetti (area di interazione, privacy, sensibilità, etc.).

3. Le applicazioni in foreground ricevono i dati di tracciamento dal servizio di Leap Motion riuscendo a connettersi attraverso l'utilizzo della libreria messa a disposizione (API di Leap Motion).

4. Nel momento in cui una applicazione perde il focus del sistema operativo, ovvero va in background, il servizio di Leap Motion interrombe lo scambio di dati con essa. Ma, come accennato nel primo punto, le applicazioni che desiderano lavorare in background possono ottenere il permesso di ricevere i dati solo se vengono concessi i permessi.

2.4 Sistema di tracciamento del dispositivo

Il sistema di Leap Motion riconosce e traccia il movimento di mani, dita ed oggetti simili o vicini alla forma di un dito (penne, bacchette ed oggetti di puntamento). Il dispositivo opera ad un altra precisione offrendo la possibilità di tracciare il numero di frame, la posizione, il movimento e le gesture effettuate sopra di esso. Durante l'operazione di tracciamento vengono memorizzati i dati rilevati in un insieme di *frames*. Ciascun frame contiene una lista di tutte le entità di tracciamento rilevate in quel preciso istante, l'oggetto frame si può definire il punto di partenza o la radice alla base del modello di raccolta dati del Leap Motion.

Sistema di coordinate

Il sistema di Leap Motion utilizza un sistema di coordinate cartesiane destrorso nel quale l'origine è posizionata esattamente al centro del dispositivo. L'asse x e l'asse z si estendono lungo il piano orizzontale parallelo al dispositivo, dove l'asse x è parallela al lato lungo del dispositivo con valori positivi verso destra. L'asse y è perpendicolare al dispositivo con valori positivi crescenti verso l'alto, mentre l'asse z ha valori di profondità crescente verso la direzione dell'utente.
Le unità di misura utilizzate dalle API del dispositivo sono le seguenti: la distanza è espressa in millimetri, il tempo è espresso in microsecondi, il tempo viene espresso in millimitri al secondo, ed infine gli angoli sono espressi in radianti. Uno schema riassuntivo del sistema di coordinate adottato dal Leap Motion è visibile in Figura 2.6.

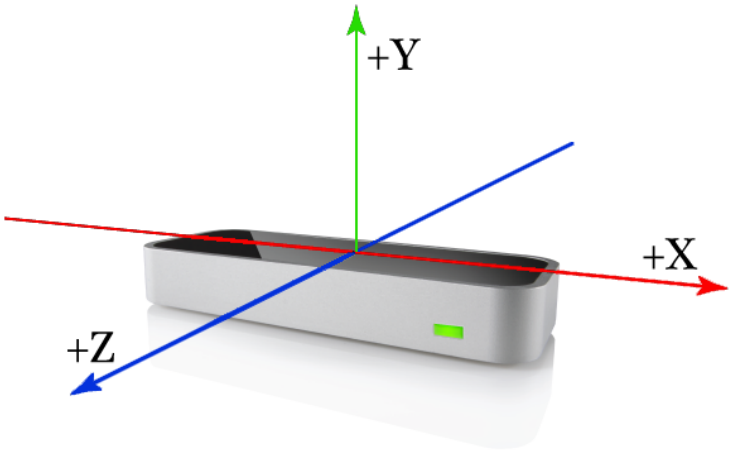


Figura 2.6: Sistema di coordinate destrorso di Leap Motion

Riconoscimento delle mani

Il modello della mano fornisce informazioni sulle caratteristiche, la posizione e l'identità della mano rilevata da Leap Motion, nonché una lista del numero delle dita associate alla mano. Il software fa uso di un modello inter-

no dell'anatomia delle mani umane per sfruttare un sistema di tracciamento predittivo che permetta di ottenere una stima della posizione delle parti della mano, anche se queste non dovessero essere rilevate o chiaramente visibili dal dispositivo. Il software utilizza le parti visibili della mano, il modello interno, e i dati memorizzati nei precedenti frames per calcolare approssimativamente la posizione delle parti che in quell'istante non sono chiaramente visibili. L'insieme delle parti rilevate dal dispositivo forniscono informazioni fondamentali sull'orientamento della mano, la sua normale al palmo, la sua direzione (Figura 2.7).
Leap Motion opera anche una distinzione tra le due mani, se destra o sinistra, ed il loro allungamento, quindi se in posizione chiusa o aperta. Il sistema infine non risulta essere vincolato all'utilizzo per utente singolo, ma può essere sfruttato da più persone ponendo attenzione a non ostacolare il rilevamento delle altre mani.

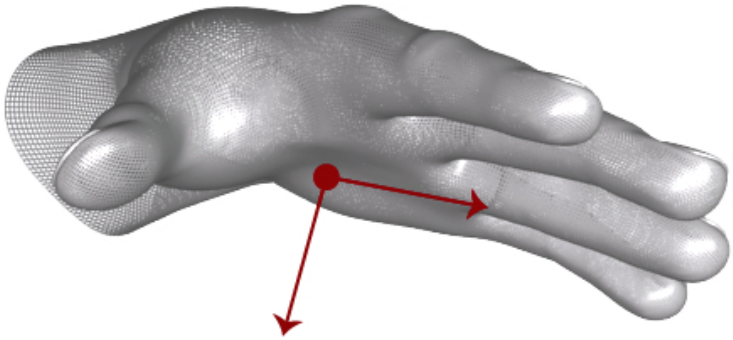
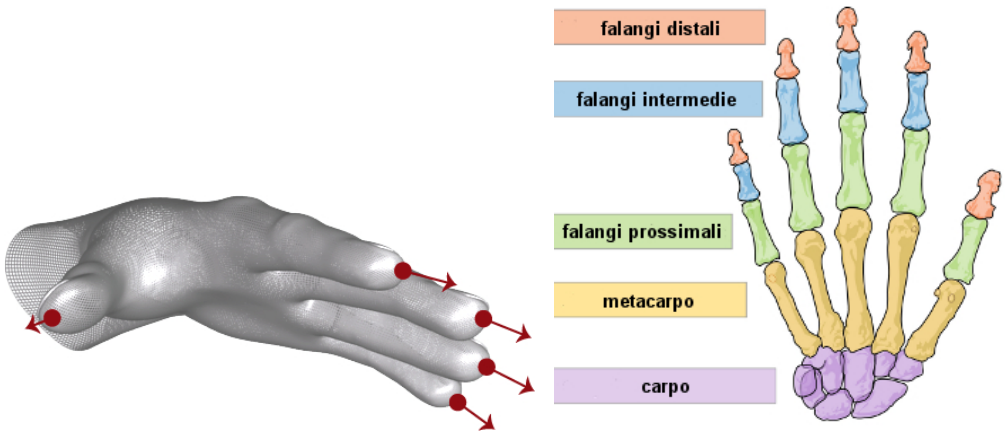


Figura 2.7: La normale al palmo e la direzione definiscono l'orientamento della mano

Riconoscimento delle dita

Il dispositivo Leap Motion fornisce informazioni per ogni dito della mano, con la possibilità di raccogliere dettagli sulla tipologia del dito, la sua direzione, la sua posizione, la distanza percorsa o il suo tempo di visibilità ad ogni frame Figura 2.8(a). Come nel caso della mano il software adopera,

sfruttando il modello interno, una stima sulla posizione delle dita non rilevate attraverso il dispositivo. Per far ciò vengono sfruttate le caratteristiche anatomiche di ciascun dito, correlate alla sua disposizione ossea (metacarpo, falangi prossimali, falangi intermedie, falangi distali) Figura 2.8(b).



(a) Direzione e posizione delle dita (b) Struttura ossea della mano

Figura 2.8: Riconoscimento delle dita di una mano

Le dita vengono identificate direttamente dal loro nome (pollice, indice, medio, anulare, mignolo), facendo completa distinzione sulla mano di appartenenza.

Immagini grezze

Oltre alle informazioni di tracciamento descritte in precedenza, le API di Leap Motion forniscono la possibilità di prelevare immagini grezze monocromatiche sfruttando le due telecamere stereoscopiche interne al dispositivo. Per usufruire delle immagini rilevate all'interno di una applicazione, ad ogni frame è necessario effettuare la richiesta ad una o entrambe le telecamere presenti. Però, come per le applicazioni in background, devono essere concessi espressamente i permessi all'applicazione per poterle utilizzare.
La raccolta di immagini grezze è stata inserita recentemente all'interno delle API di Leap Motion per favorire lo sviluppo di applicazioni orientate alla

realtà aumentata, con possibilità di sfruttare marker appartenenti ad una struttura esterna [6]. Le immagini inizialmente si presentano in maniera distorta, ma grazie agli strumenti messi a disposizione è possibile correggerle e calibrarle correttamente. La Figura 2.9 mostra la visuallizzazione delle mani dalle telecamere del dispositivo Leap Motion.

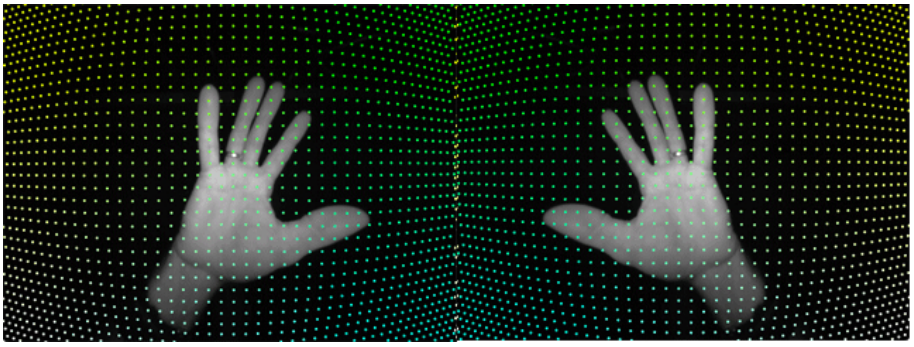


Figura 2.9: Immagine grezza delle mani

2.5 Gestures

Il software di Leap Motion fornisce degli strumenti per riconoscere alcuni movimenti predefiniti della mano e delle dita, tali movimenti vengono chiamati *gestures*. Il riconoscimento delle *gestures* viene osservato per ogni singolo dito, ed il software provvede a riportare ed elaborare i dati rilevati in un singolo frame attraverso lo stesso modus operandi dei dati di tracciamento per mani, dita ed oggetti di puntamento.
Le *gestures* riconosciute dal software del dispositivo sono:

Swipe

Il movimento lineare di un qualsiasi dito all'interno di uno spazio nel raggio di Leap Motion, viene riconosciuto come gesture *Swipe* (Figura 2.10). La

gesture deve essere continua nel tempo, e durante il disegno si memorizzano i dati di tracciamento per ogni frame dall'inizio alla fine del movimento. L'aggiornamento dei dati termina quando il dito cambia direzione oppure c'è un brusco cambio di velocità.

Figura 2.10: Gesture Swipe riconosciuta da Leap Motion

Circle

Il dispositivio è in grado di riconoscere il movimento di un dito che disegna un cerchio nello spazio e lo associa ad una gesture chiamata *Circle*. Si può sfruttare qualsiasi dito o oggetto di puntamento. Perché il disegno del cerchio possa essere riconosciuto, è necessario che il movimento sia continuo e può essere effettuato in senso orario o antiorario (Figura 2.11). Una volta che la gesture è iniziata si aggiornano i dati di tracciamento, ed il progresso relativo al disegno del cerchio. La gesture termina nel momento in cui il dito rallenta eccessivamente oppure si ferma.

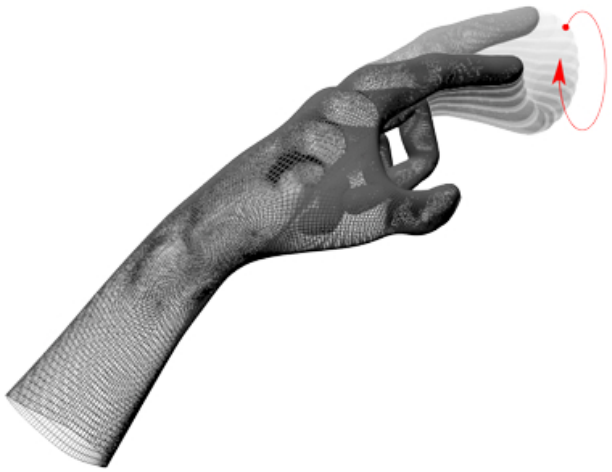


Figura 2.11: Gesture Circle riconosciuta da Leap Motion

Key taps

Il software permette di rilevare il movimento di un dito che punta verso il basso (Figura 2.12). Il movimento della gesture si esegue come se si premesse un pulsante o un tasto di pianoforte. A differenza delle gesture precedenti, il rilevamento è discreto e viene considerato il singolo frame rilevato in cui è stata eseguita l'azione.

Figura 2.12: Gesture Key taps riconosciuta da Leap Motion

Screen taps

La gesture *Screen taps* viene riconosciuta da Leap Motion quando un qualsiasi dito effettua un movimento avanti e indietro velocemente, come se si dovesse toccare uno schermo touchscreen (Figura 2.13). La gesture, come la precedente, è discreta ed il rilevamento è correlato al singolo frame.

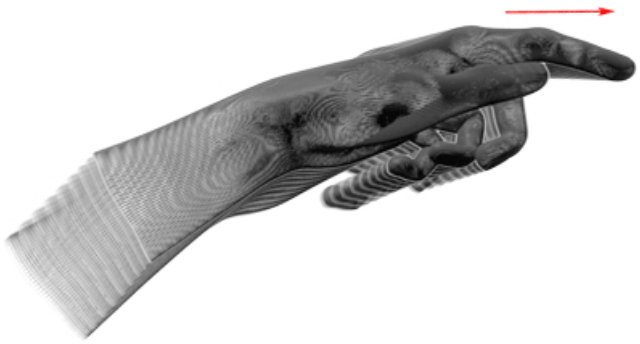


Figura 2.13: Gesture Screen taps riconosciuta da Leap Motion

Capitolo 3

Leap Motion e Blender

3.1 Introduzione a Blender

Blender è un software *open-source* per la modellazione 3D gratuito e completamente aperto ad ogni modifica. E' un software multipiattaforma che permette la modellazione, l'animazione, il rigging, compositing e rendering di immagini tridimensionali. Inoltre offre funzionalità per simulazioni di fluidi, di rivestimenti, di particelle, mappature UV, per la creazione di applicazioni o giochi 3D, nonché di animazioni 3D.
In origine, il programma fu sviluppato come applicazione interna dallo studio di animazione NeoGeo che divenne velocemente il più grande studio di animazione in Olanda. Successivamente nel 1995 lo strumento di modellazione utilizzato da NeoGeo venne reinventato e riscritto e destinato ad essere quello che oggi è il software 3D *open-source* più conosciuto, Blender. Nel 1998 Ton Roosendaal, il creatore di Blender, fondò la società Not a Number Technologies (NaN) per procedere con uno sviluppo approfondito e alla distribuzione del programma che inizialmente venne etichettato come *shareware*. I risultati non ottimali dell'azienda portarono alla bancarotta nel 2002 e Roosendaal nel giugno dello stesso anno iniziò una campagna di raccolta fondi per rendere *open-source* il programma. Nel settembre dello stesso anno si ottennero più di 100.000 ottenendo la soglia minima per il rilascio del codice sorgente.

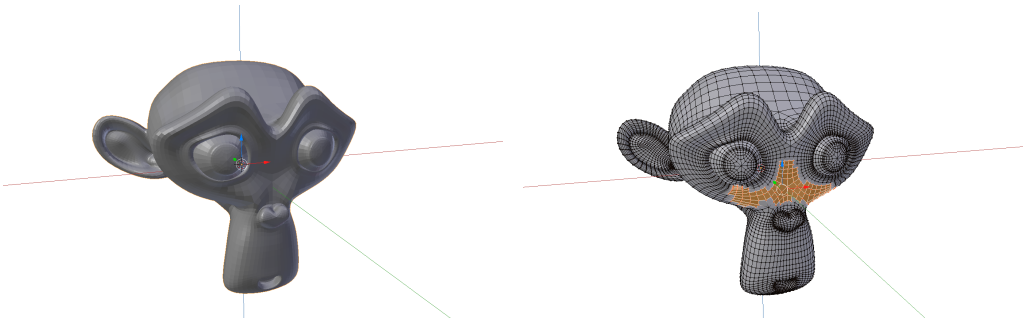
Ora Blender è un progetto *open-source* molto attivo ed è guidato dalla Blender Foundation.

Scripting

Blender possiede una caratteristica molto potente, dispone di un interprete Python interno pienamente equipaggiato. Questo offre una grossa opportunità ad utenti e sviluppatori di creare o aggiungere plugin e addon attraverso degli script scritti in Python. Quest'ultimo è un linguaggio di programmazione che si può definire "pseudocompilato", risultando ottimo per molte e varie tipologie di applicazioni, dal web, al networking, fino ad arrivare alla grafica 3D. Python risulta essere un linguaggio molto interattivo, che fonda le sue radici sulla programmazione ad oggetti e comprende moduli, eccezioni, gestione dinamica dei tipi, tipi dinamici di alto livello e classi, riuscendo ad abbinare una notevole potenza con una sintassi molto chiara. Come detto precedentemente Python è stato concepito per essere duttile in differenti scenari, in particolar modo risulta molto utile se ustato per lo sviluppo di estensioni per software che hanno un'interfaccia programmabile. Proprio per questo motivo Blender ne fa uso.
Quando Blender è diventato *open-source* molti dei nuovi sviluppatori hanno supportato la nascita della nuova fondazione rendendosi parte attiva nello sviluppo e crescita del programma. Con la modifica alla UI (Interfaccia Utente) e l'inserimento di nuove funzioni, le API di Python sono la parte di Blender che ha usufruito maggiormente dello sviluppo, offrendo un supporto alla comunità di utenti e sviluppatori grazie alla documentazione online. In aggiunta, Blender fornisce un editor di testo e una console di comandi interni per dare la possibilià di scrivere ed eseguire gli script in Python mentre si usa il programma; oppure è possibile eseguire gli stessi tramite file esterni sfruttandoli come addon da attivare all'avvio di Blender.

3.2 Modalità di Blender

E' assodato che Blender sia un programma difficile da imparare e meno intuitivo, a discapito dei diretti rivali (Maya, 3DSmax, Lightwave, ecc.). La maggior parte delle funzioni e modalità presenti possono essere richiamate attraverso scorciatoie da tastiera, grazie ad una mappatura dei tasti con le numerose funzioni messe a disposizione. Blender presenta differenti modalità che possono essere attivate solo singolarmente e non in contemporanea. Ogni modalità presenta delle funzioni differenti che agiscono su un oggetto in maniera differente ed arbitraria, ciascuna delle quali attivabile in correlazione al tipo di oggetto selezionato, con un vincolo delle azioni eseguibili su di esso. Le due modalità principali sono:



(a) Object Mode (b) Edit Mode

Figura 3.1: Modalità di Blender con la mesh Suzanne

3.2.1 Object Mode

E' la modalità di default di Blender ed è disponibile per qualsiasi oggetto presente nella scena. Le azioni e funzioni che mette a disposizione sono l'insieme delle trasformazioni standard applicabili ad un oggetto: sono consentite operazioni di spostamento, rotazione e scala. Un esempio della modalità appena descritta si può osservare in Figura 3.1(a) che mostra la mesh Suzanne, mascotte emblematica del programma.

3.2.2 Edit Mode

E' possibile passare dalla modalità Object Mode di un oggetto selezionato alla modalità Edit Mode tramite la pressione su tastiera del tasto *Tab*. L'azione è reversibile e da la possibilità di cambiare velocemente da una modalità all'altra. Le funzioni messe a disposizione in Edit Mode permettono di agire direttamente sulla struttura interna dell'oggetto, con la possibilità, prendendo in esempio delle mesh poligonali, di modificare vertici, lati, facce. Nella Figura 3.1(b) è possibile vedere come si presenta la modalità e come si possano selezionare le parti strutturali dell'oggetto.

Sono disponibili anche ulteriori modalità che possono essere richiamate. Esse sono attivabili principalmente con oggetti di tipo mesh e permettono di utilizzare manipolatori differenti:

Vertex Painting: Mette a disposizione una via più comoda e facile per la colorazione di un oggetto. Si può manipolare il colore dei singoli vertici di una mesh, piuttosto che la texture. Quando un vertice viene colorato, il colore di quest'ultimo si modifica in accordo alle proprietà del pennello con cui si è agito.

Texture Painting: La modalità permette di editare una UV texture oppure un immagine in un modo intuitivo e semplice attraverso il 3D View Editor o UV/image editor. Una UV texture è un'immagine utilizzata per colorare la superficie della mesh, e si ottiene attraverso uno o più UV maps.

Weight Paint Mode: Principalmente viene utilizzata per il rigging di mesh per definire il peso di un insieme di vertici correlato all'influenza rispetto ad un oggetto *bone* stabilito.

Particle Mode: Consente di modificare sistemi di particelle (capelli, vestiti, ecc.) variando direttamente i *key-points* ed il loro percorso.

Pose Mode: Consente di definire la posa di un *armature* modificando, con trasformazioni simili alla modalità Object Mode, il comportamento di oggetti *bones*.

Un'altra modalità di cui si farà uso nel prosecuio della tesi è la modalità Sculpt Mode descritta nel pragrafo a seguire.

3.2.3 Sculpt Mode

In questa modalità è possibile modellare diffrenti mesh, ad alto contenuto di vertici, attraverso l'utilizzo del mouse come se fosse uno scalpello. Ovvero letteralmente scolpire lungo la superficie dell'intera mesh poligonale.
Sculpt Mode è molto simile alla modalità Edit Mode perché entrambe hanno l'obbiettivo di alterare la forma di un oggetto, la differenza principale sta nel flusso di lavoro che hanno: la modalità scultura lavora anch'essa con vertici, lati e facce di una mesh, ma anziché concentrarsi sui singoli elementi strutturali agisce su una porzione di area dell'oggetto attraverso un pennello. In Figura 3.2 si può osservare il processo di lavorazione che illustra come partire da una mesh con pochi poligoni si possa definire e delineare un volto umano. Per la scultura sono messi a disposizione dell'utenti diversi strumenti, suddivisi per categorie, ai quali si fa riferimento. Gli insiemi principali sono denominati *Brush*, *Texture*, *Symmetry*, *Stroke*, *Curve*.

Brush: Menù messo a dispodizione da Blender per poter selezionare la tipologia di pennello da utilizzare sulla mesh ad altra risoluzione. Esisteno diversi tipi di pennelli (Figura 3.3), per ciascuno di essi è possibile definire il raggio d'azione, la forza e se agisce in maniera additiva o sottrattiva.

Texture: Offre la possibilità di caricare una texture da immagine esterna per poterla utilizzare come pattern principale per il pennello. Prima di poterla utilizzare è necessario però definire il Materiale della mesh nel menù apposito.

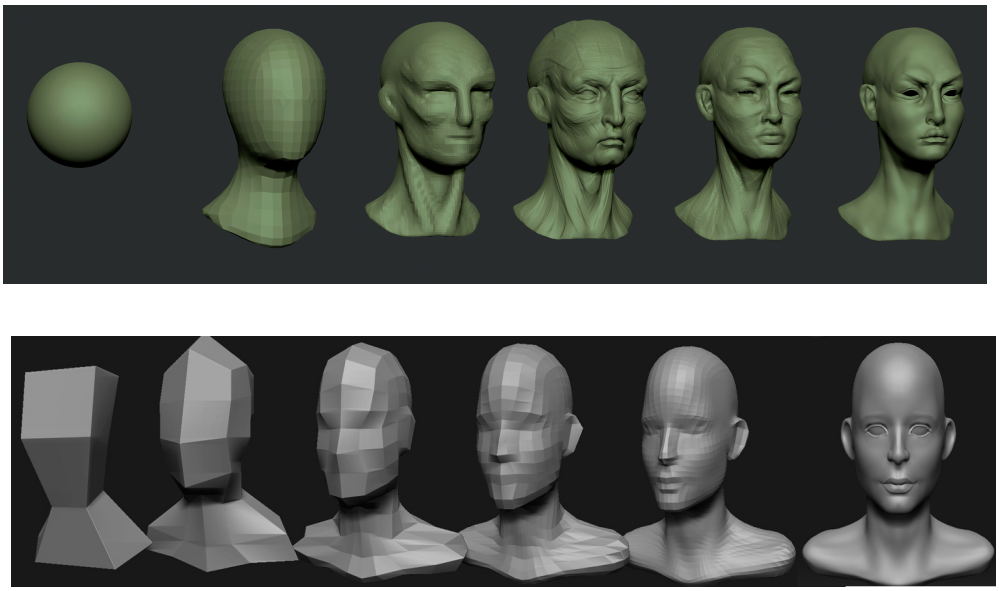


Figura 3.2: Workflow del processo di sculpting di due mesh

Simmetry: Durante la deformazione della mesh con il pennello è possibile applicare le modifiche in maniera simmetrica ad un determinato asse. Questo da la possibilità di concentrarsi solo su di una porzione di mesh, ponendo maggiore attenzione ai dettagli. Un esempio potrebbe essere la creazione si un volto dove alcuni tratti somatici sono speculari.

Stroke: E' possibile modificare il metodo di applicazione della pennellata seguendo alcuni criteri. I criteri messi a disposizione sono differenti, come la gestione attraverso il disegno di una curva, impostando una spaziatura tra un'applicazione ed un'altra, oppure modificando il *jitter* ovvero la sua frequenza di azione.

Curve: Permette di controllare la diminuzione di forza del pennello attraverso differenti tipologie di curve preimpostate. La diminuzione è configurata dal centro del pennello fino ad arrivare al bordo, mappando i valori a partire dall'inizio della curva per tutta la sua lunghezza.

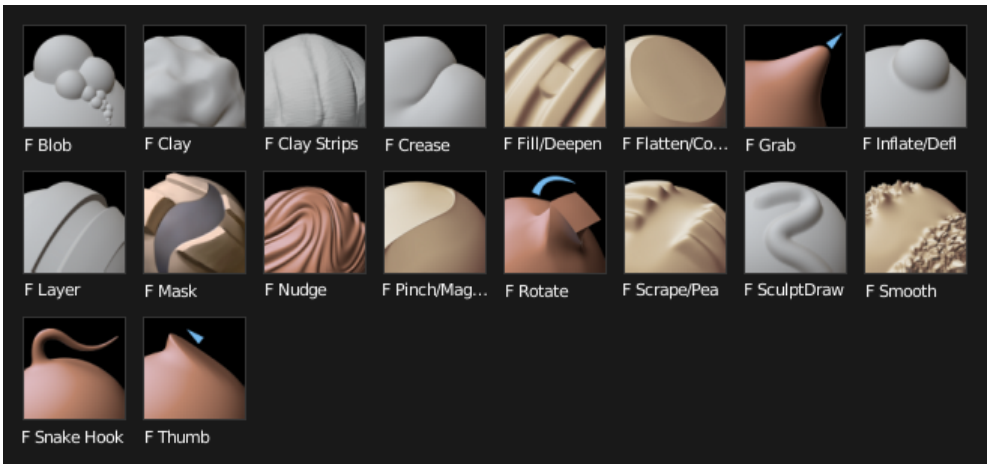


Figura 3.3: Pennelli presenti nella modalità Sculpt

3.3 Constraints e Modifiers in Blender

Blender offre un insieme di strumenti per la modifica strutturale e comportamentale di un oggetto all'interno di una scena 3D. Questi strumenti vengono rispettivamente chiamati *Modifiers* e *Constraints* e forniscono una grande versatilità e flessibilità durante la fase di creazione e successiva animazione di un oggetto.

3.3.1 Constraints

I Constraints (Vincoli) sono degli ottimi strumenti per controllare il comportamento di un oggetto. Questi vincoli hanno come concetto base l'associazione dei dati di un oggetto al comportamento di un altro oggetto (Figura 3.4). Essi risultano molto utili durante il processo di animazione, nella fase di *rigging* possono essere applicati alle ossa di un'armatura per definire legami articolari e pose, possono governare il movimento degli occhi che seguono il volo di un aereo e possono sincronizzare la rotazione delle ruote di una macchina. I Constraints possono essere anche combinati assieme per creare animazioni sofisticate e più complesse, che richiederebbero molto tempo per essere realizzate.

Questo li rende uno degli strumenti più potenti messi a disposizione da Blender. A disposizione ci sono quattro tipologie di Constraints:

Motion Tracking: Si occupano della gestione del movimento di una camera o di un oggetto, fornendo all'oggetto attivo la rotazione e la posizione in un arco di tempo data una clip già registrata. Vengono utilizzati per l'editing video, ad esempio per inserire una mesh all'interno di un video già registrato.

Transform: Vengono utilizzati per associare tutte le trasformazioni compiute nella scena 3D da un oggetto, ad uno o più oggetti. Si possono impostare preferenze che modificano i valori delle traformazioni mantenendo i vincoli specificati.

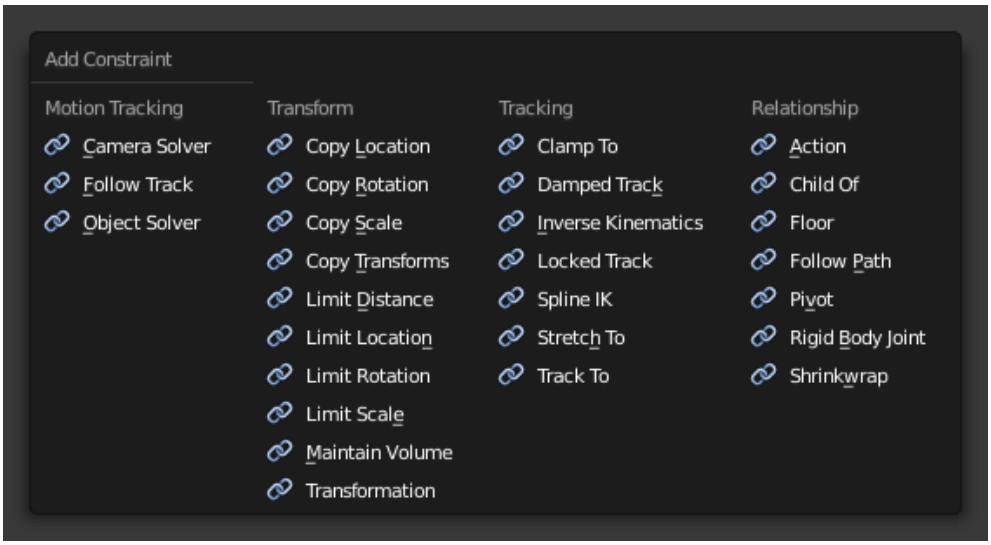


Figura 3.4: Constraints messi a disposizione da Blender

Tracking: Questi Constraints si occupano di tracciare e seguire il movimento di un oggetto in correlazione ad un altro. Si utilizzano frequentemente per gestire, con oggetti armature, il comportamento di ossa e delle catene che possono formare, risultando molto utili con la *Inver-*

se Kinematics (Cinematica Inversa). Sono utili anche per osservare il movimento di un oggetto attraverso una telecamera.

Relationship: Hanno lo scopo di esplicitare delle relazioni fra oggetti differenti. Possono essere utilizzati per creare piani asintotici con la posizione di una mesh come può essere un pavimento, muovere in relazione alla lunghezza di una curva un oggetto in un arco di tempo, oppure impostare gradi di parentela fra un oggetto ed un altro.

3.3.2 Modifiers

I Modifiers (Modificatori) sono delle operazioni automatiche che agiscono su di un oggetto in maniera non distruttiva (Figura 3.5). Essi forniscono degli strumenti che apportano diversi effetti ad un oggetto, aiutando in modo considerevole operazioni che richiederebbero molto tempo per essere eseguite. Si ha quindi la possilità di combinarli ed attivarli in maniera del tutto dinamica e totalmente personalizzabile, per poi applicare le traformazioni apportate in un secondo momento. Il loro lavoro si focalizza nel modificare come un oggetto viene visualizzato e successivamente renderizzato, senza intaccarne la geometria che può essere modificata direttamente. I modificatori possono essere utilizzati con diversi tipi di oggetto come mesh, curve o superfici e l'elenco di quelli attivabili e disponibili differisce a seconda del tipo di oggetto attivo.
Sono presenti quattro tipologie di Modifiers:

Modify: Sono dei modificatori che non influenzano in maniera diretta la forma dell'oggetto a cui vengono applicati. Essi agiscono sull'oggetto attraverso altri tipi di dati, come ad esempio i suoi vertici.

Generate: Questa è la categoria dei modificatori costruttivi che modificano direttamente l'apparenza dell'oggetto, oppure si occupano automaticamente dell'aggiunta di nuova geometria, quindi di nuovi poligoni, alla mesh a cui vengono applicati.

Deform: I modificatori di tipo Deform modificano l'aspetto generale dell'oggetto a cui vengono applicati, il tutto avviene senza l'aggiunta di nuovi poligoni senza la modifica della sua geometria. Possono errsere utilizzati per tutte le mesh, le curve o le superfici ed anche per l'inserimento e modifica di un testo.

Simulate: Questo è il gruppo per la simulazione della fisica degli oggetti. Definiscono il comportamento dell'oggetto in relazione allo strumento che dovrebbe rappresentare. Offrono la possibilità di simulare il comportamento di fluidi, vestiti, collisioni, fumo, sistemi particellari, ecc.

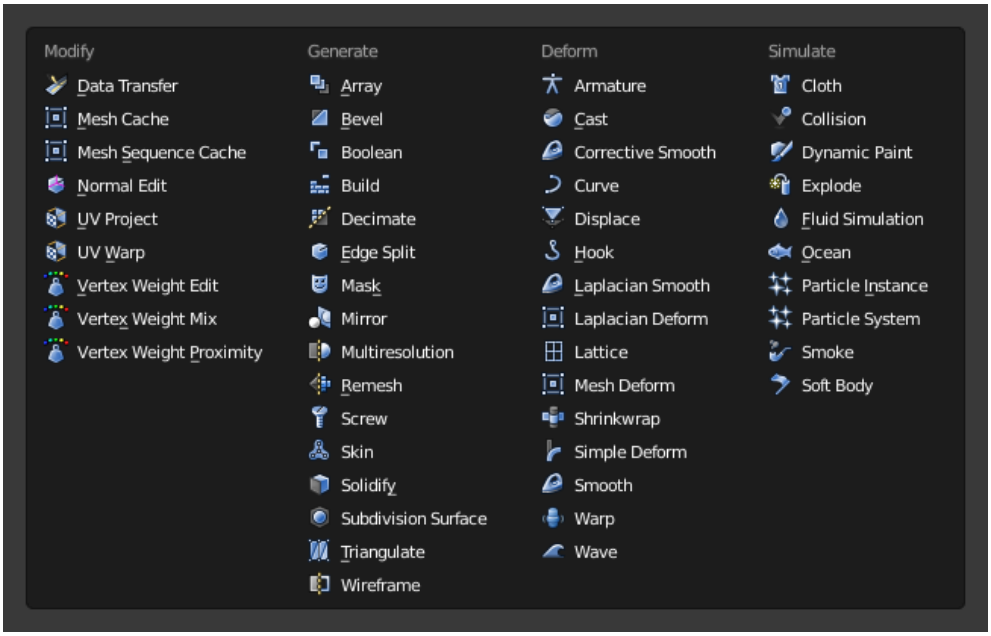


Figura 3.5: Modifiers messi a disposizione da Blender

.

3.4 Stato dell'Arte

La versatilità di Leap Motion ha accresciuto lo sviluppo di progetti volti a sfruttare il dispositivo all'interno di ambienti per la modellazione. Questo mi ha dato la possibilità di visionare i progetti ed il lavoro effettuato per la realizzazione degli strumenti messi a disposizione. Di seguito vengono elencati quelli a cui si è fatto riferimento.

Leap Modal Controller

Il Leap Modal Controller [17] è un addon per Blender pensato per spostare la posizione e l'orientamento di un oggetto all'interno della scena 3D, utilizzando gesture riconosciute, oppure il semplice movimento della mano nello spazio. Oltre a fornire un'interazione con tutti gli oggetti, anche di tipo *bones*, l'addon fornisce un'interazione con le armature *MakeHuman*. L'addon necessita del supporto della tastiera, attraverso la quale è possibile avviare la funzione desiderata. Le possibilità che offre sono:

- Con la mano aperta effettuare rotazioni e traslazioni di un oggetto in scena
- Spostare un oggetto con il movimento del dito indice
- Con le mani cambiare la posizionedelle mani del modello *MakeHuman*
- Effettuare rotazioni dei gomiti del modello *MakeHuman* utilizzando la gesture Circle

Freeform

Leap Motion attraverso l'App Store ufficiale ha rilasciato Freeform [20], che permette di sfruttare il dispositivo per modellare e deformare l'aspetto di una sfera, offrendo la possibilità di cimentarsi con lo sculpting. L'applicazione attualmente è disponibile solo su piattaforme Windows e fornisce vari strumenti di interazione.

L'utilizzo del mouse è necessario solo per l'avvio dell'applicazione, tutto il resto serà gestito tramite il dispositivo grazie all'inserimento di comodi menù circolari che danno la possibilità di selezionare colore, sfondo, materiali e i vari strumenti messi a disposizione. Inoltre una volta terminata la modellazione si ha la possibilità di esportare il modello nei più comuni formati riconosciunti dai vari programmi di modellazione. Le potenzialità di Leap Motion e dell'applicazione stessa sono servite da stimolo ulteriore per lo sviluppo di un'interazione con lo strumento di modellazione fornito da Blender.

Figura 3.6: Freeform per Leap Motion

Blender BQ

Blender BQ [18] è un addon che fornisce diverse funzionalità, con l'obiettivo di sostituire completamente l'utilizzo di mouse e tastiera con il Leap Motion. Per l'avvio delle funzioni implementate si affida all'utilizzo di comandi vocali predefiniti, precedentemente memorizzati in un dizionario. Con questa aggiunta Blender BQ introduce una nuova meccanica operazionale

nell'utilizzo del software di modellazione. Gli strumenti messi a disposizione sono:

- Effettuare spostamenti di un oggetto in scena
- Scalare le dimensioni di un oggetto in scena
- Con il comando vocale *pottery* entrare nell'omonima modalità per poter aggiungere o sottrarre materia all'oggetto selezionato. Viene anche riconosciuta la gesture *swipe* per far ruotare l'oggetto mentre viene aggiunta materia.
- Con il comando vocale *paint* si può entrare nell'omonima modalità ed è possibile selezionare il colore dell'oggetto con il movimento della mano.
- Con diversi comandi vocali è possibile selezionare la prospettiva controllando la vista utente.

Leap Motion Amimated Hand

E' un semplice addon pensato per sfruttare il Leap Motion all'interno del Blender Game Engine [19]. Sono stati costruiti dei modelli grezzi di entrambe le mani, con palmo e dita che sono oggetti separati tra di loro e quindi oggetti che possono essere animati indipendentemente. Esso si occupa quindi di animare i modelli creati a seconda del movimento della mano e delle dita stesse, creando e visualizzando su schermo una buona replica del comportamento dato dalle mani.

Capitolo 4

Leap Aided Modelling

L'obiettivo che ci si è posti all'inizio nello sviluppo dell'addon è stato quello di fornire un'interazione tra il dispositivo Leap Motion e le modalità offerte da Blender. Si è provato quindi ad arginare i limiti bidimensionali posti da mouse e tastiera in fase di modellazione attraverso il Leap Motion, mappando le coordinate a tre dimensioni all'interno della scena tridimensionale di Blender. In questo capitolo, verranno illustrate le feautures di navigazione e modelling implementate: la gestione del movimento in scena per la modifica della vista utente, l'editing e la selezione dei vertici di *mesh poligonali* descrivendone la struttura, ed infine il supporto creato per la modalità scultura. Quest'ultima offre la possibilità di manipolare manualmente i vertici di una mesh, con il semplice movimento del mouse si ha la possibilità, tirando o spingendo, di modificare la superficie di una mesh in un modo molto simile alla lavorazione e modellazione dell'argilla o della plastilina.

4.1 Strumenti per lo sviluppo

Per lo sviluppo e l'implementazione dell'addon con Leap Motion è stata utilizzata la seguente configurazione:

- Linux Mint 18 Cinnamon 64-bit
- Blender 2.78

- Leap Motion (LM-010)
- Python 3.5
  - API Blender [11]
  - Leap Motion API/SDK 2.3.1 [10]
  - Modulo Numpy/Scipy [13]

Inizialmente prima di procedere con l'implementazione dell'addon si è preso confidenza con gli strumenti che Blender fornisce, focalizzandosi con attenzione al procedimento di creazione di animazioni e sculture con il programma. Per far ciò si è fatto riferimento alla documentazione messa a disposizione da Blender e si sono seguiti dei tutorial per apprendere le basi della scultura e della gestione di un'animazione. Successivamente ci si è documentati attraverso le API di Blender [11] per apprendere il funzionamento del modulo bpy per Python implementato in precedenza [6]. Si è seguito un tutorial per apprendere l'utilizzo delle API messe a disposizione da Leap Motion per Python [10] e, dopo aver preso confidenza con entrambi gli strumenti, è iniziato lo sviluppo dell'addon. Per il corretto funzionamento Blender richiede una versione di Python superiore alla 3.0, creando un'incompatibilità con le librerie messe a disposizione da Leap Motion. Il problema è stato risolto grazie all'utilizzo di SWIG [12] il quale si occupa di generare un wrapper che funge da ponte tra le librerie di Leap Motion e Python 3.5 (Figura 4.1).
Come riportato in [6] Blender non risulta essere thread-safe. Viene quindi utilizzato un *Modal Operator* di Blender [15] con l'aggiunta di un timer che permette di interrograre periodicamente, quindi per ogni frame, il Leap Motion dando la possibilità di analizzarlo ed estrarne i dati. L'attivazione dell'addon può essere fatta in qualsiasi momento, alla pressione della barra spaziatrice comparirà una barra di ricerca che permette l'attivazione dell'operatore sotto il nome di Leap Motion Extension. Per disattivare ed uscire dall'addon basta la pressione del tasto *ESC* da tastiera.

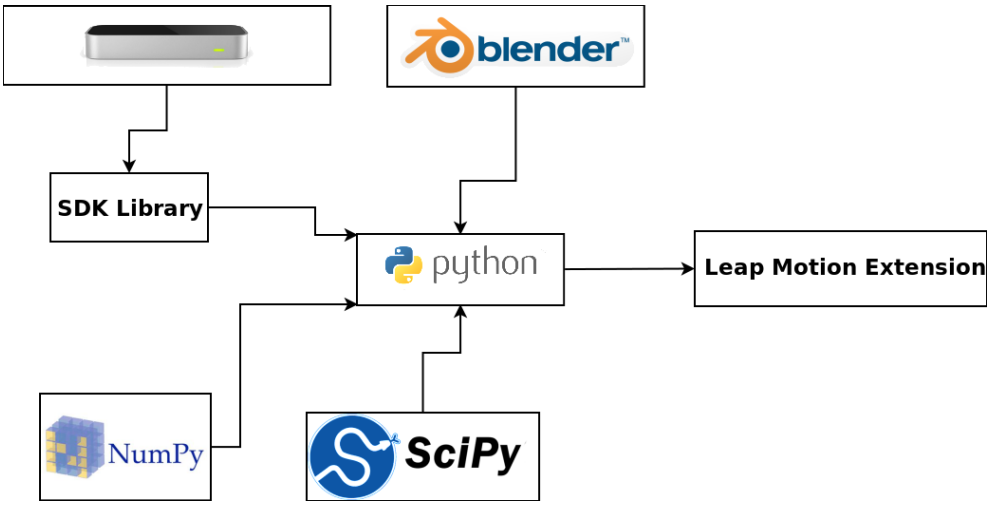


Figura 4.1: Schema riassuntivo dell'implementazione

4.2 Funzioni di Navigazione

In questa sezione verranno descritte le funzioni di implementate per lo spostamento e la modifica della vista utente.

4.2.1 Mappatura delle coordinate

Per permettere a Blender di riconoscere gli input esterni provenienti da Leap Motion è necessario eseguire una mappatura tra le coordinate 3D del dispositivo (Figura 2.6) e quelle all'interno della scena 3D di Blender. Questo passaggio è fondamentale ai fini del corretto funzionamento dell'addon, sia per quanto riguarda il movimento della scena, sia per il corretto tracciamento del movimento di un oggetto, sia per la possibilità di disegno di percorsi che l'oggetto dovrà seguire e anche per stabilire le coordinate di azione del pennello nella fase di sculpting. Nel procedere con l'implementazione si è fatto riferimento alle API di Leap Motion, in particolar modo alle tipologie di mapping fornite [16]. Per aver maggior padronanza con gli strumenti a disposizione, ci si è documentati sulle dinamiche di comportamento di una classe fornita da Leap Motion chiamata Interaction Box e precedentemente utilizzata [6]. Lo scopo di questa classe è delimitare l'ampiezza della semisfera

di tracciamento, all'interno di una "scatola interattiva" (Interaction Box) con spazi ben definiti per migliorare il tracciamento delle mani ed ovviare errori di riconoscimento in punti critici nel campo d'azione del dispositivo (Figura 4.2). La classe offre un metodo che normalizza le coordinate di Leap Motion per poi riuscire a mapparle successivamente all'interno della applicazione.

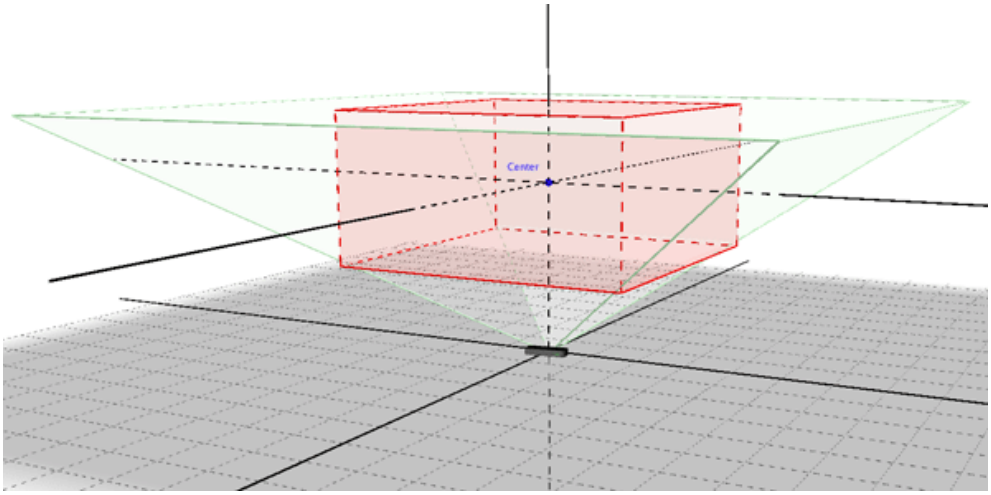


Figura 4.2: Area di tracciamento fornita da Interaction Box

4.2.2 Gestione del movimento

Il movimento della vista utente e della scena di conseguenza, è una delle azioni maggiormente eseguite all'interno di Blender e di qualsiasi altro programma di modellazione. Si deve quindi avere la possibilità di effettuare spostamenti, traslazioni e rotazioni all'interno della scena 3D in maniera del tutto libera ed arbitraria. La scelta è stata quella di mantenere la mano sinistra come gestore principale del movimento, ottimizzandone la sensibilità e l'usabilità all'interno dell'ambiente 3D. Questo offre la possibilità di lavorare con entrambe le mani sfruttando esclusivamente il Leap Motion, oppure affidandosi ad un qualche altro strumento di input 2D utilizzandolo con la mano destra. Di default, all'avvio dell'addon, il movimento tramite la mano sinistra è disattivato per ovviare spiacevoli cambi di scena non volontari. E'

possibile attivarlo tramite la pressione del tasto P da tastiera e se premuto nuovamente provvede alla disattivazione della funzione.
Inoltre, come precedentemente implementato [6], è possibile affidare il compito del movimento ad una struttura esterna oppure eseguire lo zoom nella scena attraverso la gesture *Circle* con l'indice della mano sinistra. Lo zoom in avviene eseguendo una rotazione in senso orario, lo zoom out con una rotazione in senso antiorario. La navigazione in scena, per renderla facile ed intuitiva, si governa attraverso una posa di riconoscimento data dall'estensione delle tre dita della mano sinistra: *pollice*, *indice*, *medio*. Per tracciare e riportare il movimento lungo l'asse x, y e z si fa riferimento alla posizione del palmo in un dato istante di tempo (Frame), avendo la possibilità di muoversi lungo le tre direzioni a suo piacimento.
Il vettore che governa lo spostamento S_t si calcola come la differenza tra il frame attuale appena registrato e quello precedente, eseguendo quindi una sottrazione tra i vettori normalizzati all'interno dell'Interaction Box. Nel dettaglio, se si chiama H_t la posizione della mano al frame t il vettore di spostamento S_t è dato da:

$$S_t = (H_t - H_{t-1})$$

Oltre al movimento canonico lungo gli assi, è possibile eseguire e controllare una rotazione "tilt" lungo gli assi X e Z. E' stato quindi preso in considerazione la normale al palmo della mano N_t ad ogni istante di tempo t, calcolando il quaternione di rotazione Q_t tra il frame attuale e quello precedente. Più precisamente, data la normale al palmo N_t all'istante di tempo t ed ottenuto il prodotto vettoriale V tra N_{t-1} e N_t, il quaternione di rotazione Q_t viene calcolato come:

$$Q_t = (1 + N_t * N_{t-1}, V.x, V.y, V.z)$$

Successivamente il quaternione Q_t viene normalizzato per ottenere la rotazione da applicare agli assi X e Z. A fronte di ottenere una maggiore precisione durante gli spostamenti, sono state inserite delle soglie minime e di diversa

grandezza per ogni asse, per sopperire ai piccoli movimenti involontari della mano che farebbero muovere la camera in maniera indesiderata. In aggiunta si è inserita la possibilità di controllare il movimento della vista vincolandolo ad un singolo asse. Il tutto può essere attivato attraverso la pressione del tasto *Windows key*, su tastiera, più la lettera corrispondente all'asse che si desidera muovere (X, Y, Z).
La scelta di utilizzare *pollice*, *indice*,*medio* come posa, nonostante quest'ultime non vengano considerate per i calcoli relativi allo spostamento, è stata dettata dalla maggiore precisione che si ottiene nel riconoscimento parte del sensore Leap. Inoltre può essere considerata una posa molto intuitiva a cui far riferimento per lo spostamento lungo i tre assi.

4.3 Editing di Mesh Poligonali

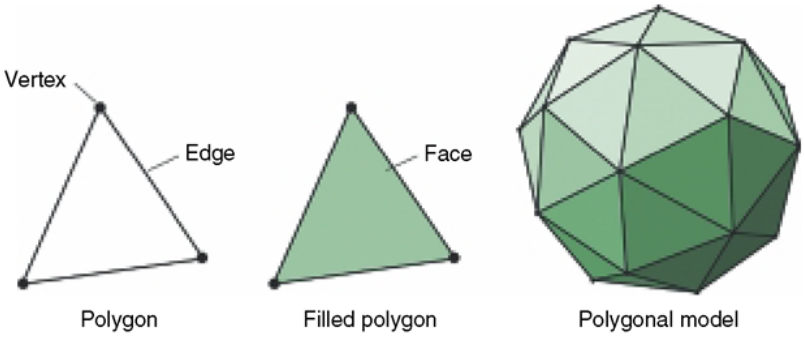


Figura 4.3: Struttura di una mesh triangolare

In Blender ed in vari programmi di grafica 3D la rappresentazione di un oggetto all'interno della scena si utilizzano le *mesh poligonali*. La superfice di un oggetto 3D è rappresentata da un insieme di poligoni nello spazio che ne determinano la forma, le *mesh poligonali* si possono definire come un'insieme di *vertices* (vertici), *edges* (lati), *faces* (facce) connessi tra loro in maniera che:

- ogni *lato* è condiviso da al più due *facce* adiacenti.

- ogni *lato* è elemento di connessione tra due vertici.
- la *faccia* è l'elemento planare dato da un insieme di punti nello spazio racchiuso tra almeno tre lati.
- ogni *vertice* è condiviso da almeno due lati.

Inoltre viene introdotto il vettore *normale* ad una *faccia* che permette di determinarne il suo orientamento, utile per il calcolo di visibilità e ombreggiatura.

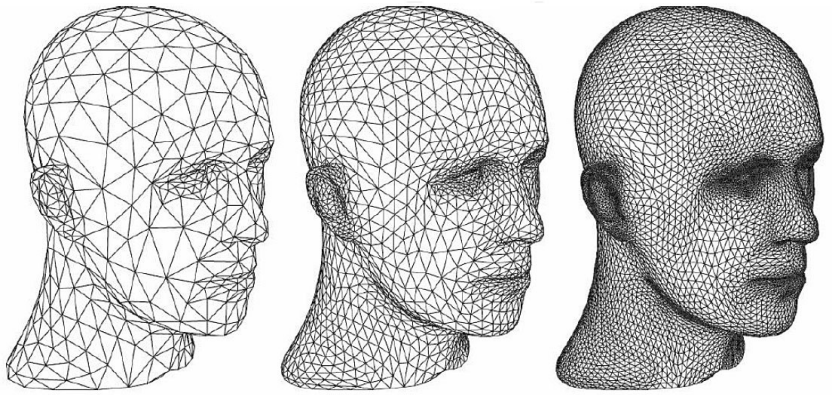


Figura 4.4: Mesh poligonali con differenti suddivisioni e numero di poligoni

Analizzando il tutto dal punto di vista geometrico, si sa che per tre punti non allineati passa un solo piano e, attraverso gli stessi punti, è possibile costruire un triangolo. Si ottiene quindi che per un qualsiasi poligono formato da un numero di punti superiore a tre, non è detto che esso stia su di un unico piano. Per questo motivo ogni poligono considerato viene suddiviso e memorizzato come un insieme di più triangoli (Figura 4.3), dato che questi riescono a garantire la planarità della figura che devono rappresentare.
Perciò la forma di ogni solido viene memorizzata attraverso i poligoni che ne delimitano le facce e quindi di conseguenza come un insieme di triangoli adiacenti. A tal proposito, si intuisce facilmente che, maggiore è il numero di triangoli che compongono la struttura di una mesh e più accurata sarà la

rappresentazione dell'oggetto 3D risultante (Figura 4.4).
Quindi si può definire una mesh triangolare come:

- Sia M una *mainfold* (Varietà geometrica) a due dimensioni di un'arbitraria topologia inclusa in R^3.

- Una mesh triangolare $m = (V, E, T)$ è una terna rappresentante una planarità a tratti approssimata di M. La terna è composta da:

  Vertices:

  $V = \{1 \cdots N\}$

  Edges:

  $E = \{(i, j) \in V \times V : X_j \in N(X_i)\}$ tra due vertici.

  Faces:

  $T = \{(i, j, k) \in V \times V \times V : (i, j), (i, k), (k, j) \in E\}$ tra due lati.

Le mesh descrivono quindi la sola topologia dell'oggetto e non contengono nessuna informazione sulle proprietà geometriche. Per definirle si devono specificare le coordinate all'interno dello spazio tridimensionale dei vertici in modo tale che: $X_i \in R^3$ per ogni $i \in V$. L'orientamento di ogni faccia invece, come detto in precedenza, è stabilito dalla lista delle normali.

4.3.1 Modifica e selezione Vertici

Oltre alla gestione del movimento della vista utente, un'altra funzione implementata è stata quella della selezione e successiva modifica dei vertici di una mesh attraverso l'utilizzo del *proportional editing*. Seguendo le precedenti impostazioni [6], si utilizzano quattro differenti modalità, ciascuna delle quali include diverse possibilità. Premendo il tasto *Shift* sul lato destro della tastiera, si può passare da una modalità all'altra, di seguito sono riportate quelle disponibili:

Spostamento

- Object Mode: E' possibile spostare o ruotare l'oggetto selezionato all'interno della scena. Per lo spostamento viene utilizzato il dito indice della mano destra mentre per ruotarlo si utilizza la mano destra chiusa.

- Edit Mode: Dopo aver selezionato uno o più vertici, facce o segmenti si possono applicare traslazioni o rotazioni. Per lo spostamento si utilizza sempre il dito indice della mano destra e per la rotazione la mano destra chiusa.

Sculpting

E' possibile entrare direttamente nella modalità di sculpting per procedere alla scultura e deformazione dell'oggetto selezionato.

Selezione

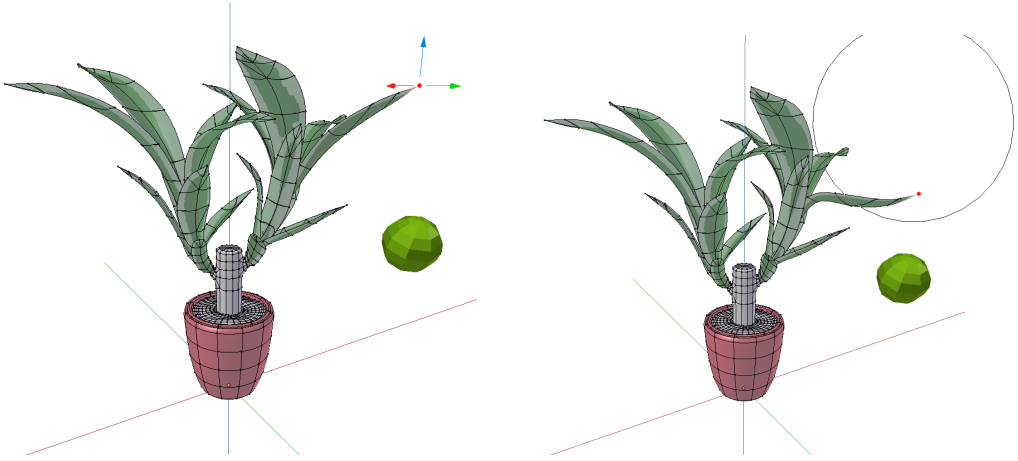
- Object Mode: Grazie ad un puntatore che viene creato nel momento in cui la modalità si attiva, si ha la possibilità di selezionare un oggetto all'interno della scena. Successivamente quando c'è un cambiamento di modalità il puntatore viene distrutto.

Modifica

- Object Mode: Dopo aver selezionato un oggeto nella scena, si ha la possibilità di scalare l'oggetto. Per effettuare l'operazione si utilizza il dito indice della mano destra muovendolo in profondita.

- Edit Mode: Si può nuovamente scalare un oggetto dopo averlo selezionato, oppure si può utilizzare un puntatore sferico per selezionare i vertici di una mesh per poi traslarli utilizzando il *proportional editing*. Si può scalare l'oggetto dopo aver selezionato tutti i vertici con il dito indice della mano destra muovendolo in profondità, con due

dita della mano destra è possibile muovere il puntatore per selezionare/deselezionare i vertici. Il raggio di selezione è proporzionale alla grandezza del puntatore, per aumentarla/diminuirla si usa la mano destra aperta muovendola in profondità. La selezione/deselezione è alternabile alla pressione del tasto *;* da tastiera, dopo aver selezionato i vertici, con il dito indice si attiva automaticamente il *proportional editing* ed è possibile spostarli (Figura 4.5).



(a) Vertice selezionato (pallino rosso) (b) Spostamento con proportional editing

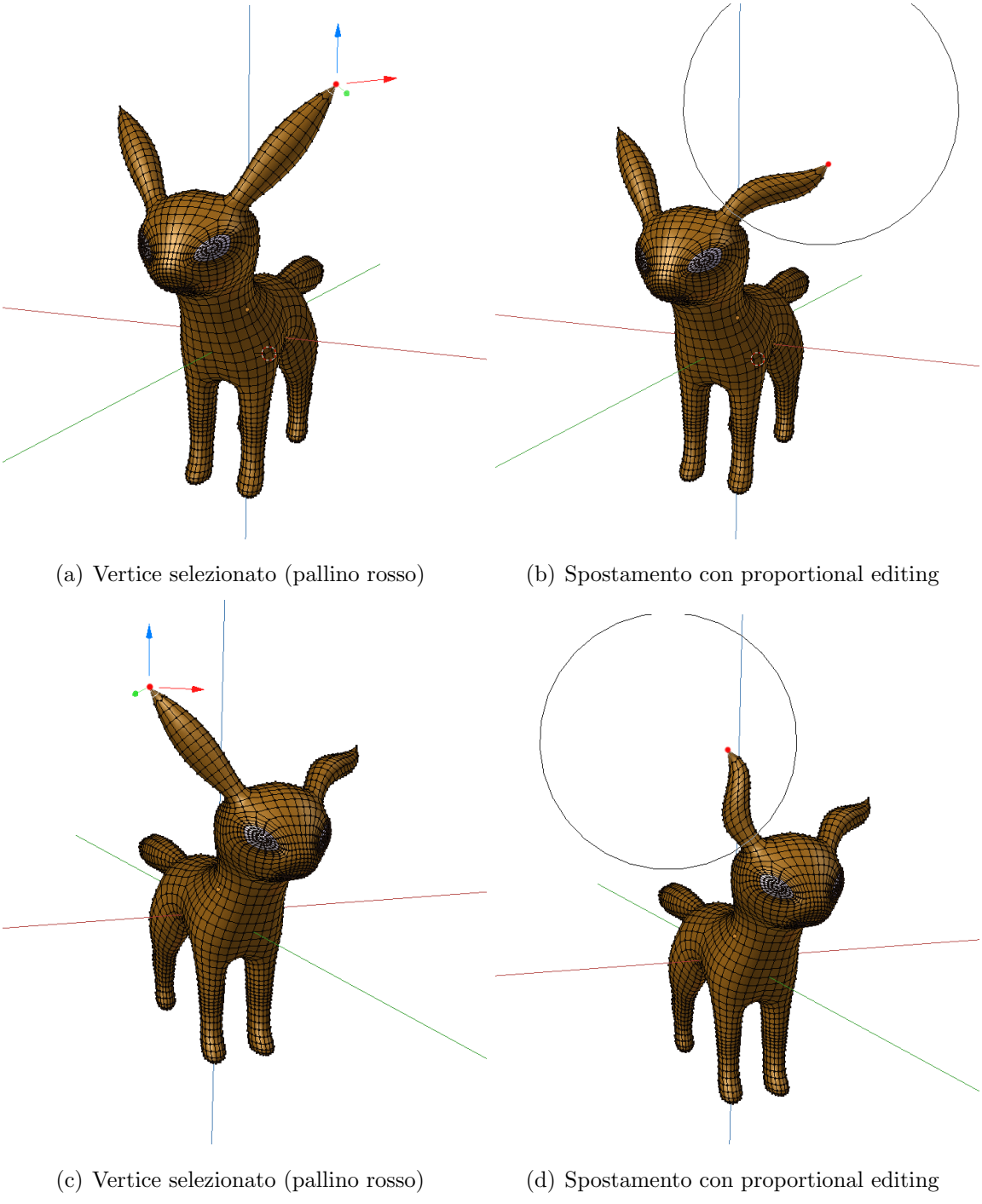
Figura 4.5: Selezione di un vertice con il puntatore e deformazione con proportional editing di una pianta.

Per capire meglio come funziona la modalità di modifica in Edit Mode è importante fare chiarezza sulle funzionalità messe a disposizione dal *proportional editing*. Quando si lavora con geometrie dense con molteplici suddivisioni, può diventare molto difficile apportare modifiche e regolazioni ai vertici della mesh. Se si vuole evitare di creare deformazioni indesiderate di singoli vertici è possibile utilizzare lo strumento di modifica proporzionale. Questo strumento risulta essere molto efficace per la deformazione di vertici, a partire da quelli selezionati, secondo un determinato raggio di azione. Esso agisce in un modo molto simile ad una calamita per deformare perfettamente la superficie del modello, senza creare pieghe o salti indesiderati, modificando

tutti i vertici all'interno del raggio impostato (Figura 4.6). Inoltre, è possibile selezionare il profilo della curva utilizzata per la deformazione (*Falloff*) tra sette impostazioni differenti (Root, Sphere, Smooth, Constant, Random, Linear, Sharp).
Detto ciò una volta entrati nella modalità *Modifica* e dopo essere passati alla modalità edit mode di blender viene creato un "puntatore" di tipo sfera che ha la possibilità di traslare e ruotare all'interno della scena tridimensionale, sopperendo al vincolo di profondità del mouse. Il movimento dell'oggetto puntatore viene governato dal movimento e dalla direzione del dito indice, tenendo conto della mappatura delle coordinate effettuata in precedenza. Però per attivarne lo spostamento è necessario mostrare al sensore due dita della mano destra, ad esempio indice e medio, altrimenti il puntatore non verrà mosso. Per effettuare la selezione dei vertici viene utilizzata una funzione *object.closest_point_on_mesh* di Blender che dato in input un punto P_i (vertice della mesh attiva in Edit Mode) e una mesh (il puntatore sfera) restituisce come output il punto P_0 del puntatore più vicino al vertice P_i insieme alla sua normale N della faccia che contiene P_0.
Quindi si calcola V come:

$$V = (P_0 - P_i) \cdot N$$

si ha quindi che il vertice P_i preso in considerazione è all'interno del puntatore se $V < 0$. Dopo aver selezionato i vertici con il puntatore è possibile deformarli con il *proportional editing* con l'inidce della mano destra, ed utilizzando due dita della mano sinistra con un movimento verticale è possibile aumentare/diminuire il raggio d'azione. Per poter poi selezionare il *Falloff* che si desidera utilizzare è possibile premere la combinazione di tasti *Shift + O* per passare sequenzialmente da una impostazione all'altra. Infine viene offerta la possibilità di attivazione o disattivazione della funzione *mirror* della modifica proporzionale semplicemente mostrando al Leap Motion la mano sinistra aperta e procedendo con il dito indice della mano destra con la modifica dei vertici.



(a) Vertice selezionato (pallino rosso)

(b) Spostamento con proportional editing

(c) Vertice selezionato (pallino rosso)

(d) Spostamento con proportional editing

Figura 4.6: Selezione di vertici per la deformazione con proportional editing delle orecchie di un cane.

4.4 Sculpting con Leap Motion

La modellazione ed editing di mesh poligonali, può essere effettuato anche attraverso la modalità *Sculpt Mode*. La loro deformazione può essere eseguita attraverso gli strumenti messi a disposizione dallo strumento di sculpting e descritti in precedenza nel paragrafo 3.2.3. Per modellare all'interno della modalità scultura si possono percorrere due strade differenti, si può utilizzare il *Multiresolution Modifiers* oppure la *Dynamic Topology*. Entrambe risultano essere delle buone soluzioni ed il loro utilizzo dipende dal risultato che si desidera ottenere.

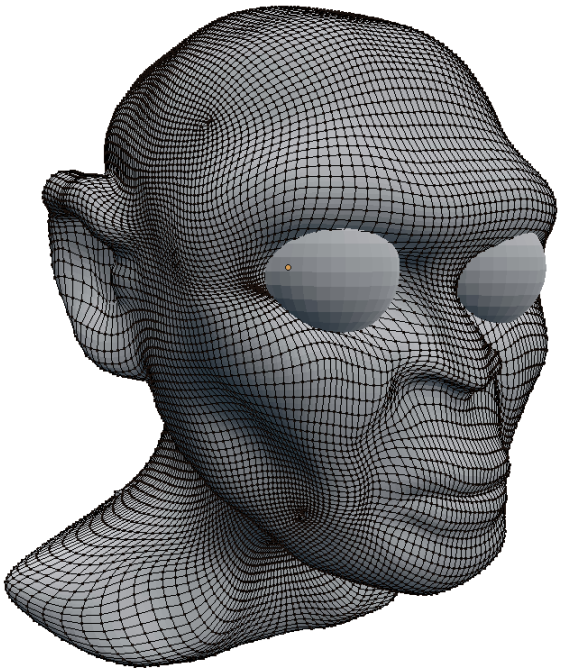


Figura 4.7: Sculpting con *Multiresolution Modifiers* in Edit Mode

4.4.1 Multiresolution Modifiers

Sculpt Mode da la possibilità di manipolare una porzione di area di un oggetto modificando i sui vertici, per questo motivo necessita di una mesh ad alta densità di vertici. Se si pensa ad un cubo ed ai suoi 8 vertici, risulta

difficile modellarne la forma, si ha bisogno infatti di mesh molto più dettagliate che possono avere 100 o anche 1000 vertici. Di conseguenza, si deve aggiungere al cubo preso in esame un *Multiresolution Modifiers* che lavora in modo molto simile al *Subdivision Surface Modifier* apportando delle suddivisioni alla mesh, ma in aggiunta permette di editare il livello di suddivisioni in modalità scultura.
Maggiore è il numero di suddivisioni e molti più dettagli verranno aggiunti al cubo, che apparirà come una sfera, e darà la possibilità di scolpire la sua superficie attraverso i vari pennelli presenti (Figura 4.7).
Ovviamente più alto è il numero di suddivisioni, maggiore sarà anche il carico legato al processore del computer in uso.

Figura 4.8: Sculpting con *Dynamic Topology* in Edit Mode

4.4.2 Dynamic Topology

A partire dalla versione 2.6 Blender ha introdotto una nuova feauture chiamata *DynaTopo* che significa Dynamic Topology. Questa modalità non necessità l'utilizzo del *Multiresolution Modifiers* (Figura 4.8). Nonostante questo offre ugualmente un buon rendimento e un minor stress del processore in fase di computazione, dato che non aggiunge vertici lungo l'intera superficie della mesh, ma focalizza l'aggiunta solo nelle aree di mesh scolpite con il pennello. Nel dettaglio, ogni bordo della mesh all'interno dell'area coperta dal pennello che supera una certa grandezza viene diviso a metà. La grandezza con cui ogni bordo viene suddiviso è controllata dall'impostazione *Detail Size*. Oltre all'impostazione di suddivisione, può essere abilitata l'opzionde di collasso dei bordi ed una opzione di smoothing e shading.
L'aspetto negativo di questa modalità è che dopo la modellazione la mesh risulta non avere una buona topologia, a causa dei vertici inseriti dinamicamente ed in maniera casuale. Per risolvere questo problema è necessario usare una tecnica chiamata *Re-topology*.

Interazione con Leap Motion

L'interazione della modalità *Sculpt Mode* con il dispositivo Leap Motion non ha portato ad ottenere i risultati sperati. A causa della scarsa documentazione offerta da Blender per l'operatore di sculpting il lavoro svolto risulta incompleto e non del tutto funzionante. Ugualmente di seguito verranno descritte le operazioni eseguite ed i problemi riscontrati durante lo sviluppo di questa parte dell'addon. Per accedere alla modalità di scultura, dopo aver caricato l'addon, tramite la pressione del tasto *Shift* da tastiera si deve selezionare la modalità interna *Sculpting*. Una volta selezionatà all'interno di Blender si passerà automaticamente alla modalità *Sculpt Mode* e verrà inserito nella posizione dell'origine un puntatore di tipo sfera uguale a quello presente nella modalità *Modifica* per la selezione di vertici della mesh (Figura 4.9). E' possibile di spostare e ruotare il puntatore attraveso il movimento e

la direzione del dito indice secondo la mappatura di coordinate descritta nella sezione 4.2. L'applicazione dell'incisione (*stroke*) del pennello attivo in quel

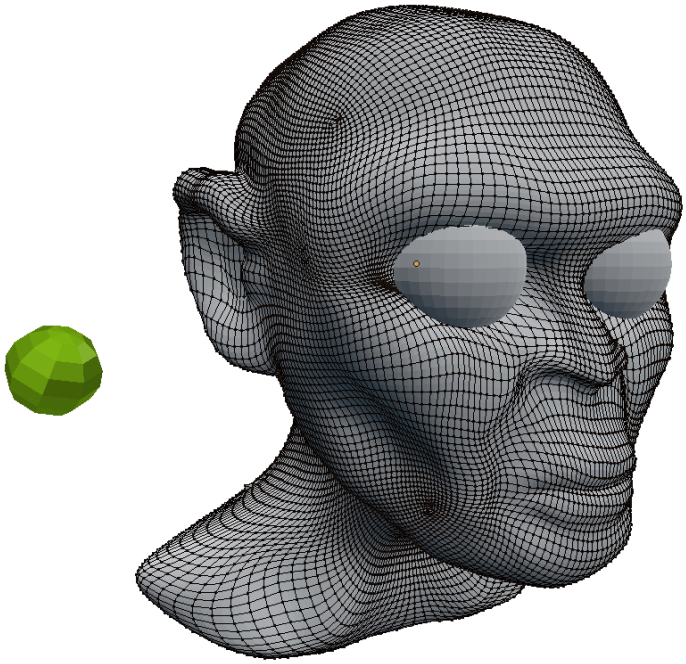


Figura 4.9: Sculpt Mode in Blender con il puntatore per Leap Motion

momento, è governata dal posizionamento del puntatore sull'area della mesh che si desidera scolpire, mentre il raggio d'azione può essere aumentato o diminuito con il movimento verticale di due dita della mano sinistra. Ovviamente rimangono attivi i tasti di scelta rapida dei vari pennelli. Queste sono le funzioni attualmente implementate, ma il principale problema riscontrato è l'applicazione dello *stroke*, quindi dell'incisione del pennello sulla mesh.
Il problema è quindi direttamente correlato al comando in Python che determina l'incisione chiamato `brush_stroke`. Quest'ultimo, secondo la documentazione di Blender, prende in imput una proprietà di gruppo denominata `OperatorStrokeElement` ed è così strutturata:

```
is_start  Type: boolean, default False
```

```
location  Type: float array of 3 items in [-inf, inf],
                default (0.0, 0.0, 0.0)

mouse     Type: float array of 2 items in [-inf, inf],
                default (0.0, 0.0)

pen_flip  Type: boolean, default False

pressure  Type: float in [0, 1], default 0.0

size      Type: float in [0, inf], default 0.0

time      Type: float in [0, inf], default 0.0
```

Nell'implementazione si è correttamente creato il gruppo completandolo con i dati richiesti in input, seguendo le indicazioni reperite in un forum [24] o da addons [25] [26] [18] che già ne usufruivano. Purtroppo nonostante questo l'incisione non avviene, ma non da neanche segni di errore nella lettura del comando datogli in input, rendendo difficile la possibilità di comprendere ed eventualmente correggere l'istruzione.
Un'ultima precisazione riguarda gli addons trovati [25] [26] [18], tutti utilizzano e sono stati implementati con una versione di Blender più datata rispetto a quella utilizzata per lo sviluppo della tesi, ed è possibile che le dinamiche di utilizzo di `brush_stroke` siano differenti. A causa della mancata disponibilità di un'ulteriore macchina su cui lavorare, non è stato possibile testare l'effettivo funzionamento degli addons sopra citati con versioni più vecchie di Blender. Perciò le uniche prove effettuate sono state legate al computer che si è utilizzato per lo sviluppo, attraverso il quale però ci si è imbattuti in errori di incompatibilità dovuti alle diverse versioni di Python utilizzate dalle varie versioni di Blender.

Capitolo 5

Teoria sulle curve NURBS e Animazione

In questo capitolo vengono trattate le basi teoriche per la realizzazione di curve NURBS e del controllo del movimento di un punto lungo una curva. Verranno così definite le basi teoriche per l'implementazione della funzionalità di animazione realizzate all'interno dell'addon che verranno presentate nel capitolo successivo. Parte del lavoro di tesi infatti è stato volto alla creazione di percorsi per generare animazioni di oggetti o camere utilizzando il dispositivo Leap Motion, con l'obiettivo di facilitare operazioni di questo tipo.

5.1 Curve 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 parte del progetto di tesi al disegno e alla modellazione di curve NURBS in Blender, utilizzando esclusivamente il dispositivo Leap Motion. Di seguito verranno presentate le basi teoriche delle curve NURBS, indicandone caratteristiche e proprietà.

Una NURBS è formata da un'equazione parametrica di grado n in funzione di un parametro u che varia nell'intervallo [0,1] ed è definita dalle seguenti caratteristiche:

Il 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 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.

I 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).

I Nodi: 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 rappresentare con pochi parametri superfici di alta qualità.

- Permettono un alto controllo di forma, modificando anche solo un punto di controllo la modifica si propaga all'interno gradualmente.

Una curva NURBS con vettore nodale V e grado K viene quindi definita dalla seguente formula:

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

dove N rappresenta il numero di punti di controllo $P_i \in R^3$ della curva nell'intervallo parametrico $0 \leq u \leq 1$ e le funzioni base $N_{i,k}(u)$ sono definite dalla seguente formula ricorrente:

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

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

Inserimento di un Nodo

Questa operazione consente di aumentare i gradi di libertà di una curva senza modificarne la forma. 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 inserendo un nodo nel vettore nodale V.
La tecnica di inserimento di un nodo aggiunge un nodo al vettore nodale V senza modificare né il grado K della curva NURBS né la sua forma. In questo modo, dunque, 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).

5.2 Controllo del moto lungo una curva

La tecnica standard per animare degli oggetti (mesh, camere) in una scena, richiede che l'animatore imposti delle posizioni chiave (*key points*) in

differenti punti lungo l'animazione, organizzandoli in modo da ottenere il moto degli oggetti automaticamente interpolando i *key points* attraverso una curva *smooth* (i.e. NURBS, Bézier,..).
Il problema del movimento degli oggetti in animazione risulta essere molto più complesso della semplice interpolazione tra i punti lungo la curva perché, in aggiunta alla posizione degli oggetti il loro orientamento deve essere interpolato tra le varie posizioni chiave. Il principale problema dell'utilizzare la costruzione a tratti delle curve B-Splines per interpolare tra i vari *key-frames* in una sequenza di animazione, è che quest'ultimi tipicamente non sono spaziati ad intervalli di tempo costanti. Dunque ogni segmento della curva, parametrizzata da 0 a 1, avrà una parametrizzazione differente rispetto al valore complessivo del tempo, e la continuità tangente generata dalla curva verrà persa. Questo può apparire durante l'animazione come un cambio repentino di velocità di un oggetto perché la traiettorià mantiene la sua continuità G^1 ma non la continuità C^1. Di conseguenza questo risulta inaccettabile in un'animazione realistica, perchè implica infinite accellerazioni nei punti di giuntura, cosa non fisicamente fattibile, dunque sbagliata. Un requisito comune, sopratutto nei sistemi per il movimento di una camera, è avere alti gradi di continuità. Tuttavia la continuità della velocità (continuità C^1) risulta essere abbastanza per un movimento uniforme convincente, però improvvisi cambiamenti di velocità potrebbero far apparire l'animazione scattosa in certe circostanze.
Pertanto è desiderabile avere un meccanismo attraverso il quale la continuità C^2 può essere ottenuta durante la costruzione a tratti della curva. Quindi il disegno di una curva NURBS all'interno dello spazio tridimensionale, è solo il primo passo per la creazione di un moto di un oggetto [22] e sono necessarie ulteriori operazioni per poter ottenere il risultato voluto. Come prima operazione si deve stabilire una relazione tra il variare del parametro u e la distanza percorsa lungo la curva, quindi definire un metodo che permetta di avanzare lungo il percorso ad incrementi regolari. Una volta definita questa relazione sarà possibile muovere un oggetto a velocità costan-

te e potranno essere introdotti dei metodi per incrementare o diminuire la sua velocità. Per poter imporre dei cambiamenti di velocità, all'oggetto in questione deve essere garantita la possibilità di muoversi a velocità costante. La velocità costante è assicurata dalla parametrizzazione con lunghezza ad arco (*arc length*) della funzione di interpolazione della curva.

5.2.1 Calcolo con Arc Length

Dai punti campionati nello spazio, costruire l'approssimante parametrico $P(u)$ può essere considerato come rappresentare tre funzioni. Questo perchè la posizione di un punto è espressa in tre dimensioni ed il movimento di un oggetto nello spazio tridimensionale è espresso dalla sua posizione che è data dalla terna (x, y, z). Dunque per poterlo mouvere lungo la curva disegnata è necessario interpolare la sua posizione nello spazio. L'oggetto definisce di conseguenza uno *spazio curva*, ovvero una funzione parametrizzata, valutata su un punto nello spazio tridimensionale, come una funzione del valore parametrico. Se per esempio si prende il caso di una funzione polinomiale cubica, la sua equazione parametrica nello spazio tridimensionale è:

$$\begin{aligned} P(u) &= (x(u), y(u), z(u)) \\ x(u) &= a_x u^3 + b_x u^2 + c_x u + d_x \\ y(u) &= a_y u^3 + b_y u^2 + c_y u + d_y \\ z(u) &= a_z u^3 + b_z u^2 + c_z u + d_z \end{aligned} \tag{5.1}$$

Data l'equazione 5.1, ci si accorge che con la medesima è possibile descrivere una curva NURBS di grado $K = 3$ e di ordine $O = 4$. Una volta specificata la curva, un oggetto si può muovere lungo di essa al variare della variabile parametrica u, calcolando di conseguenza le coordinate (x, y, z) del corrispondente punto lungo la curva NURBS. Questo tipo di informazione è importante, essa ci indica la distanza coperta dall'oggetto nello spazio definito dalla curva, purtroppo però non si hanno ancora informazioni su come la distanza venga percorsa in un intervallo di tempo (velocità). Per fare ciò

si deve introdurre una funzione *distanza-tempo* che metta in relazione i due valori. Questa funzione viene chiamata *parametrizzazione arc lenght*, dove per *arc length* si intende la distanza lungo la curva.

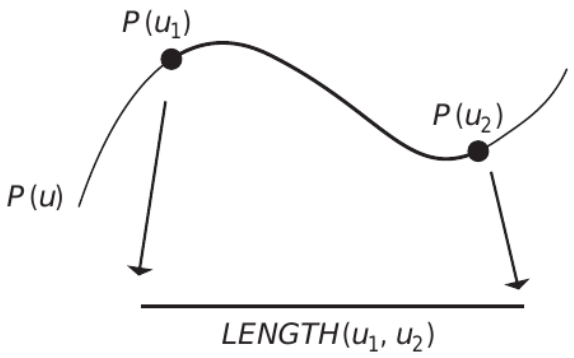


Figura 5.1: Arc Length

Il calcolo analitico della *parametrizzazione arc lenght* è dato da:

$$S = \int_0^L \parallel \frac{dP}{du} \parallel du \tag{5.2}$$

Con $u \in [0, 1]$, dove L è la lunghezza della curva e:

$$\frac{dP}{du} = \left(\frac{dx(u)}{du}, \frac{dy(u)}{du}, \frac{dz(u)}{du} \right)$$

La parametrizzazione permette il movimento lungo la curva NURBS ad una velocità costante, provvedendo a valutare la curva ad intervalli *arc length* uguali. Inoltre offre la possibilità di gestire l'accelerazione e decelerazione dell'oggetto lungo la curva NURBS grazie al controllo della distanza percorsa in un dato intervallo di tempo. Poiché il valore della variabile parametrizzata u non è uguale all'*arc length*, è difficile selezionare i valori di u per i quali l'oggetto si muove lungo la curva alla velocità desiderata. Infatti generalmente la relazione tra la variabile parametrica e *arc length* risulta essere non lineare, è quindi necessario stabilirne una.

Supponendo di definire la funzione $LENGTH(u_a, u_b)$ come la lunghezza della curva nello spazio tridimensionale descritto dal punto $P(u_a)$ al punto $P(u_b)$ (Figura 5.1) si presentano due problemi da risolvere:

- Dati entrambi i parametri u_a e u_b trovare $LENGTH(u_a, u_b)$.
- Data l'*arc length* s ed il parametro u_a trovare u_b tale che $LENGTH(u_a, u_b) = s$. Ovvero risolvere la seguente equazione $s - LENGTH(u_a, u_b) = 0$.

Il primo punto può essere risolto quando la computazione *arc length* s viene calcolata come una funzione della variabile u, e si ha che $s = S(u)$ con s_i che denota l'*arc length* ad uno specifico valore u_i. Quindi se l'inversa, $u = S^{-1}(s) = U(s)$, può essere stimata o calcolata si può procedere alla effettiva parametrizzazione *arc length* che è data da $P(U(s))$. Il secondo punto invece si può risolvere specificando, in differenti modi, la distanza che l'oggetto dovrebbe percorrere lungo la curva ad ogni istante di tempo.
A tal proposito, in generale, nessuno dei due problemi menzionati precedentemente ha approcci di tipo analitico per trovare una soluzione, sono richieste pertanto tecniche che prevedono soluzioni numeriche. Una di queste, che ha dato le basi per la gestione del movimento all'interno dell'addon sviluppato, è la stima dell'*arc length* attraverso la "differenza in avanti" (*forward differencing*).

5.2.2 Stima dell'arc length con Forward Differencing

Il metodo più facile ed intuitivo per stabilire una corrispondenza tra la variabile parametrica e l'*arc length* è di campionare la curva in una moltitudine di valori parametrici, ciascuno dei quali produrrà un punto lungo la curva. I punti che sono stati campionati permetteranno di approssimare l'*arc length* attraverso la somma analitica sulla *arc length* della distanza tra tutti i punti consecutivi. Ad esempio data una curva NURBS $P(u)$ l'approssimazione della posizione lungo la curva per il variare di u compreso tra [0,1], discretizzato con passo costante $\triangle t = 0.5$ può essere definita da una funzione

$G[i]$ calcolata come:

$$\begin{aligned}
G[0] &= 0.0 \\
G[1] &= \text{la distanza tra } P(0.0) \text{ e } P(0.05) \\
G[2] &= G[1] \text{ più la distanza tra } P(0.05) \text{ e } P(0.10) \\
G[3] &= G[2] \text{ più la distanza tra P}(0.10) \text{ e } P(0.15) \\
&\dots \\
G[20] &= G[19] \text{ più la distanza tra } P(0.95) \text{ e } P(1.0)
\end{aligned}$$

Questa funzione si chiama *chord length* e approssima la curva con una poligonale approssimazione della arc-length e, per $\triangle t > 0$:

$$\begin{aligned}
t_i &= i \triangle t \\
P_i &= P(t_i) \\
S &= \sum_i \parallel P_{i+1} - P_i \parallel_2
\end{aligned}$$

Con $\| \cdot \|$ norma 2, ovvero la distanza tra i due punti sulla curva.
Sia $s(u_i)$ la lunghezza approssimata della curva da $u = 0.0$ a $u = u_i$ il calcolo della *chord length* è dato:

$$s(u_i) = s(u_{i-1}) + \parallel P_i - P_{i-1} \parallel_2$$

Un esempio di questa funzione è visibile in Figura 5.2 che mostra la mappatura dei valori V per il valore parametrico u ed il corrispondente valore della funzione G.
Per conoscere la distanza percorsa s lungo la curva dall'inizio di quest'ultima, se ci si trova al valore u dell'intervallo parametrico, si può utilizzare la tabella in Figura 5.2 nel seguente modo: l'*arc length* s può essere interpolata linearmente grazie alle approssimazioni G, calcolate precedentemente, utilizzando la differenza tra il valore parametrico dato v ed i valori $G[i]$ e $G[i+1]$ che definiscono gli estremi che lo comprendono.

$$s = G[i] + \frac{v - V[i]}{V[i+1] - V[i]}(G[i+1] - G[i]) \tag{5.3}$$

Parameter, Arc Length Pairs		
Index	**Parametric Value (*V*)**	**Arc Length (*G*)**
0	0.00	0.000
1	0.05	0.080
2	0.10	0.150
3	0.15	0.230
4	0.20	0.320
5	0.25	0.400
6	0.30	0.500
7	0.35	0.600
8	0.40	0.720
9	0.45	0.800
10	0.50	0.860
11	0.55	0.900
12	0.60	0.920
13	0.65	0.932
14	0.70	0.944
15	0.75	0.959
16	0.80	0.972
17	0.85	0.984
18	0.90	0.994
19	0.95	0.998
20	1.00	1.000

Figura 5.2: Tabella di mappatura dei valori

L'equazione 5.3 se applicata due volte e sottratta per le rispettive distanze ottenute, risolve il problema del primo punto descritto nel paragrafo precedente. Nella situazione opposta, quando si deve calcolare il valore del parametro u data l'*arc length* si può eseguire lo stesso procedimento.
Ad esempio, se si dovesse stimare la posizione di un punto con $G = 0.75$ unità di *arc length* dall'inizio della curva, il valore più vicino sarebbe $G[8] = 0.72$ che corrisponde al valore $V[8] = 0.40$. In questo caso il valore che si desidera calcolare si trova a tre ottavi tra i valori in tabella 0.72 e 0.80, quindi il vaolre parametrico viene stimato dal seguente calcolo:

$$u = 0.40 + \frac{3}{8}(0.45 - 0.40) = 0.41875$$

La soluzione infine dell'equazione $LENGTH(u_a, u_b) = s$ presentata nel precedente paragrafo può essere trovata sfruttando i valori della Figura 5.2 per stimare l'*arc length* associata al valore di u_a, sommarlo al valore dato di s, e

successivamente sfruttare i valori noti per stimare il valore parametrico della lunghezza risultante. Il vantaggio di questo approccio è che risulta facile ed intuitivo da implementare ed è veloce in fase di computazione. Purtroppo però questa tipologia di stima di valori è soggetta inevitabilmente ad errori dovuti al calcolo.
Se si introducessero metodi migliori per l'interpolazione si aiuterebbe a ridurre gli errori di stima del valore parametrico in fase di calcolo. Ad esempio anziché un'interpolazione lineare si protrebbe alzare il grado nella fase di computazione della variabile parametrica, il tutto ovviamente a sfavore del tempo di computazione che subirebbe evidenti rallentamenti. Pertanto si deve adoperare una scelta in termini di velocità/accuratezza dell'animazione.

5.2.3 Controllo della velocità

Dopo aver effettuato la parametrizzazione della curva con *arc length* è possibile controllare la velocità alla quale la curva viene attraversata.
Come accennato precedentemente l'avanzare ad intervalli regolari di *arc length* lungo di essa avrà come risultato il movimento dell'oggetto a velocità costante avendo più controllo sulla velocità. Si introduce una funzione che associa valori parametrici all'*arc length*, generando in tal maniera un controllo per l'attraversamento della curva. Una delle funzioni più comuni che si occupa di questo viene chiamata attraversamento *ease-in/ease-out*.
Esso produce un movimento di accellerazione di un oggetto dal punto di partenza, raggiunge una velocità massima, e decelera in prossimità dell'ultimo punto per poi fermarsi. Il parametro di input per la funzione di controllo della velocità è il tempo t ed il suo output è l'*arc length* s, quindi la distanza. La velocità costante si ottiene valutando a valori equamente spaziati di s dove s è una funzione lineare di t: ovvero una volta parametrizzata la curva con *arc length*, la parametrizzazione viene normalizzata in modo tale che la variabile parametrica possa variare tra 0 e 1 nel tracciamento dello spazio curva. La velocità lungo la curva può essere controllata variando i valori di *arc length* in un modo differente rispetto ad una funzione lineare di t, con la

mappatura del tempo con l'*arc length* che risulta essere indipendente dalla forma della curva considerata. Per esempio se si considera il parametro t compreso tra [0,1] la curva parametrizzata con *arc length* e normalizzata, la funzione *ease-in/ease-out* può essere implementata come $s = ease(t)$ dove alla variazione uniforme del tempo t tra [0,1], s inizierà a 0, aumenterà lentamente il valore raggiungendo la velocità nei valori intermedi, per poi decelerare lentamente in prossimità di 1 (Figura 5.3).

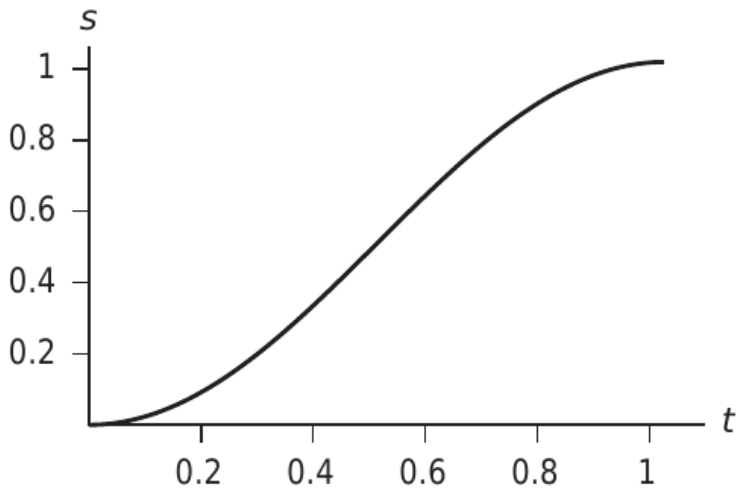


Figura 5.3: Esempio di una funzione $ease(t)$

La variabile s è quindi utilizzata come parametro interpolante in qualsiasi funzione che produca valori di spaziatura. Il controllo della velocità può essere quindi espresso in differenti maniere, con l'obiettivo comune di produrre una funzione distanza tempo $S(t)$ che per ogni istante t restituisce la distanza s percorsa lungo la curva. Lo spazio curva definisce dove andare, mentre la sua funzione distanza tempo definisce il quando, quindi ad un certo tempo t, $S(t)$ rappresenterà la distanza percorsa lungo la curva al tempo t. Detto ciò per trovare la posizione di un oggetto in un dato istante, se si considera la Figura 5.2 , si può ricavare il valore parametrico $u = U(s)$ corrispondente a quella *arc length*. Di conseguenza si valuta u per ricavare il punto lungo lo spazio definito dalla curva come $p = P(u)$, quindi si ha che $p = P(U(S(t)))$.

5.2.4 Orientamento lungo un percorso

Il movimento di un oggetto lungo una curva è una delle tecniche più popolari e facili da utilizzare. Tuttavia se si pensa al movimento di un oggetto o di una camera si deve tener conto anche dell'orientamento nel processo di attraversamento lungo la curva. Tipicamente viene definito un sistema di coordinate *(u, v, w)* per l'oggetto che deve essere animato, con origine il punto $P(s)$ lungo il percorso. I vettori che lo definiscono sono ortogonali tra loro con l'asse w che rappresenta la direzione verso quale l'oggetto è rivolto (direzione di vista) l'asse v che rappresenta l'UP vector e l'asse u che definisce il vettore perpendicolare ai due. Considerando un sistema di coordinate destrorso, l'asse u punterà alla sinistra dell'oggetto (Figura 5.4).

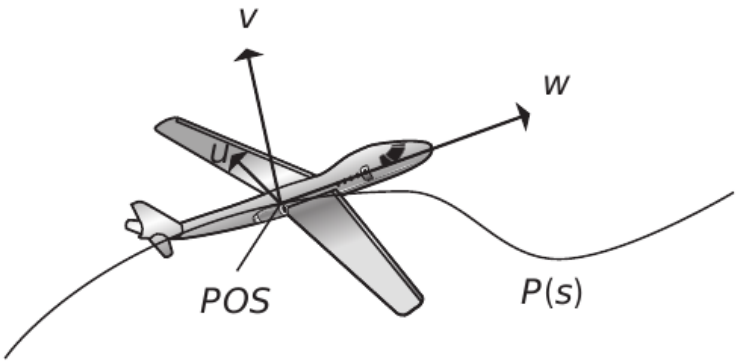


Figura 5.4: Sistema di coordinate destrorso per l'orientamento

Esistono vari metodi per gestire l'orientamento lungo una curva, ovviamente la scelta dipende dal risultato che si desidera ottenere. Di seguito vengono presentate alcuni metodi per specificare l'orientamento dell'oggetto che si desidera animare.

Frenet Frame

Questo metodo da la possibilità di definire l'orientamento di un oggetto o di una camera facendo riferimento direttamente alle proprietà della curva considerata. Il sistema di coordinate (u,v,w) è determinato dalla curvatura

e dalla tangente della curva, ed in ciascun punto $P(s)$ l'orientamento cambia per l'intera lunghezza. I vettori che lo definiscono sono normalizzati ed ortogonali tra loro, con w nella direzione della derivata prima $P^{'}(s)$, v il vettore ortogonale a w e al piano definito dalla tangente e dalla curvatura (derivata seconda $P^{''}(s)$) e u a completare la terna. Quindi dato un valore parametrico s il Frenet frame è possibile calcolarlo come:

$$\begin{aligned} w &= P^{'}(s) \\ u &= P^{'}(s) \times P^{''}(s) \\ v &= u \times w \end{aligned} \tag{5.4}$$

Il Frenet frame offre e sfrutta le informazioni ottenute direttamente dalla curva, questo però può portare a dei problemi lungo i segmenti che risultano avere curvatura nulla ($P^{''}(s) = 0$) perché in quel punto il Frenet frame non è definito. Uno dei metodi per ovviare a questo problema è considerare la differenza angolare tra i due vettori agli estremi che definiscono il tratto di curva con curvatura nulla. L'angolo ϕ viene calcolato come l'arcoseno del prodotto scalare dei due vettori, quindi $\phi = acos(v_1 \cdot v_2)$, e può essere interpolato linearmente lungo il tratto con curvatura nulla (Figura 5.5).

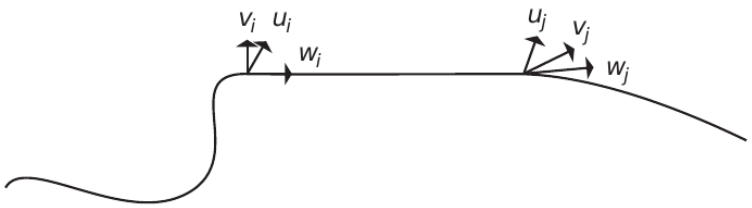


Figura 5.5: Frenet Frame con tratto di curva con curvatura nulla

Center Of Interest

Se si suppone di voler definire l'orientamento di una camera, il metodo più semplice per farlo è impostare un centro di interesse (COI) in un punto fissato all'interno della scena tridimensionale. In questo caso il COI è direttamente

determinato dal vettore di vista ($w = COI - POS$), dove con POS si indica il punto $P(s)$.
Il vettore UP v in questo caso può essere visto come la direzione positiva sull'asse y mentre il vettore u è calcolato come ($u = w \times y - axis$). La terna risultante è:

$$\begin{aligned} w &= COI - POS \\ u &= w \times y - axis \\ v &= u \times w \end{aligned} \tag{5.5}$$

Questo metodo risulta molto efficace nei percorsi che sorvolano una determinata area (COI), con la camera che si muove lungo il percorso ed è direzionata sul centro di interesse.

Path Following

Supponiamo sempre di voler definire l'orientamento della camera lungo il percorso seguendo un punto lungo la curva con l'inquadratura (*look ahead*), il vettore di vista w può essere definito attraverso l'utilizzo di un delta parametrico $\triangle s$. Data la posizione della camera definita dal vettore posizione $P(s)$, si ottiene di conseguenza il COI grazie al calcolo $P(s) + \triangle s$. Alla fine della curva quando il valore $P(s) + \triangle s$ è oltre alla lunghezza del percorso parametrizzato, il vettore di vista w viene interpolato come il vettore tangente in quel punto.

$$\begin{aligned} w &= (P(s) + \triangle s) - P(s) \\ u &= w \times y - axis \\ v &= u \times w \end{aligned} \tag{5.6}$$

Supponiamo ora di voler muovere la camera lungo un percorso e di voler tracciare il movimento di un oggetto che si muove lungo un percorso differente. La posizione della camera viene descritta sempre dal punto $P(s)$, mentre il COI è specificato come $C(s)$ su un percorso differente. Deve però

essere specificato l'UP vector considerando anche l'altro percorso a cui si fa riferimento, e viene calcolato come $U(s) - P(s)$. La terna risultante è:

$$\begin{aligned} w &= C(s) - P(s) \\ u &= w \times (U(s) - P(s)) \\ v &= u \times w \end{aligned} \tag{5.7}$$

Capitolo 6

Leap Aided Animation

Il nucleo delle funzioni implementate nell'addon è stato quello del disegno di curve NURBS con il dispositivo Leap Motion nello spazio tridimensionale. Con l'intenzione di definire percorsi di animazione che possano includere differenti tipi di orientamento, per gli oggetti all'interno della scena di Blender. Ci si è quindi dedicati all'implementazione delle tecniche di animazione viste nel capitolo precedente, utilizzando i *constraints* introdotti nel Capitolo 3; per poi sperimentarle all'interno della scena, mettendo in risalto le funzioni implementate.

6.1 Curve percorso con Leap Motion

Le potenzialità del dispositivo Leap Motion, come esplicitato in precedenza, danno la possibilità di tracciare con molta precisione il movimento del dito ad ogni frame. Tenendo conto di ciò si è implementata una nuova modalità, in aggiunta a quelle già presenti (*Spostamento, Sculpting, Selezione, Modifica*), che permette di disegnare all'interno della scena 3D di Blender una curva che verrà utilizzata come percorso di animazione di un oggetto. La modalità di *Disegno Percorso* da la possibilità di disegnare nello spazio tridimensionale una curva *polyline* utilizzando il dito indice della mano destra, sfruttando le API messe a disposizione da Blender e Leap Motion. Una

volta avviato l'addon in Blender è possibile entrare nella modalità *Disegno Percorso* dalla modalità *Spostamento* in Object Mode attraverso la pressione del tasto *;* da tastiera. Dopo la pressione del tasto si può iniziare il disegno della curva percorso a mano libera sfruttando il dito indice della mano destra rimanendo all'interno dello spazio riconosciuto dal dispositivo Leap Motion (*Interaction Box*). L'origine del tratto di disegno della curva è legato alla posizione del *3D Cursor*, si può quindi decidere da dove iniziare il disegno modificando la posizione di quest'ultimo all'interno della scena. I movimenti e gli spostamenti del dito indice vengono registrati ad ogni frame e permettono di vedere su schermo il disegno della sequenza di segmenti che formano la curva *polyline*. Dopo la pressione del tasto *;* l'implementazione del disegno della curva *polyline* avviene secondo i seguenti passi fondamentali:

1. Dopo aver posizionato il *3D Cursor* all'interno della scena avviene la creazione di un nuovo oggetto *curve 3D*

2. La curva è inizializzata come *polyline* ed il primo punto è nella posizione del *3D Cursor*.

3. Ogni vettore di spostamento rilevato viene normalizzato e viene aggiunto un nuovo punto dato dalla somma dell'ultimo vettore aggiunto e il vettore di spostamento corrente.

Perché la curva possa essere usata come percorso è necessario ricavare dalla *polyline* appena disegnata una curva di tipo NURBS. Per far ciò è necessario quindi approssimare o interpolare la curva *polyline* secondo certi parametri. La scelta è stata quella di utilizzare esclusivamente l'approssimazione, e nella sezione che segue verrà descritto il procedimento utilizzato.

6.1.1 Approssimazione

Come appurato in [6] Blender non permette una corretta trasformazione della curva *polyline* in una curva NURBS, obbligando a gestire manualmente il passaggio di conversione tra una tipologia e l'altra. Si è quindi deciso di

adoperare un'approssimazione poiché tendenzialmente il disegno della curva *polyline* non rispecchia perfettamente la curva NURBS che si voleva disegnare, questo perché possono essere effettuati involontariamente microspostamenti con il dito che generano scalettature indesiderate. Per porre rimedio a questo problema è stato utilizzato il modulo *Scipy* [13] di Python che contiene al suo interno il sottomodulo *scipy.interpolate*, del quale si è sfruttato la funzione interna *splprep()*. Questa funzione prende in input i punti di una curva, il grado, ed un valore di smooth e restituisce in output una tupla che contiene il vettore nodale, i punti di controllo ed il grado della curva *NURBS* generata dall'approssimazione. Dopo il disegno della curva *polyline* l'approssimazione avviene alla pressione del tasto *;* e segue i seguenti passi:

1. Viene chiamata la funzione *splprep()* per i punti della curva *polyline* disegnata precedentemente.

2. All'interno della scena nella posizione del *3D Cursor* c'è la creazione di un nuovo oggetto *curve 3D*

3. La curva viene inizializzata come curva *NURBS* e vengono aggiunti i punti di controllo restituiti dalla funzione *splprep()*.

4. Vengono assegnati alla curva NURBS appena creata i parametri impostati, se non modificati di default smooth $s = 3$ e grado della curva $g = 3$.

Per impostare i valori di smooth ed il grado e selezionare le impostazioni di animazione ed orientamento è stato aggiunto a Blender un pannello laterale *Animation Tools* (Figura 6.5). Come si può dedurre intuitivamente, con l'incrementare del valore di *smooth* la curva NURBS ottenuta risulta essere meno segmentata e più approssimata, mettendo in risalto la differenza rispetto all'originale (Figura 6.1).

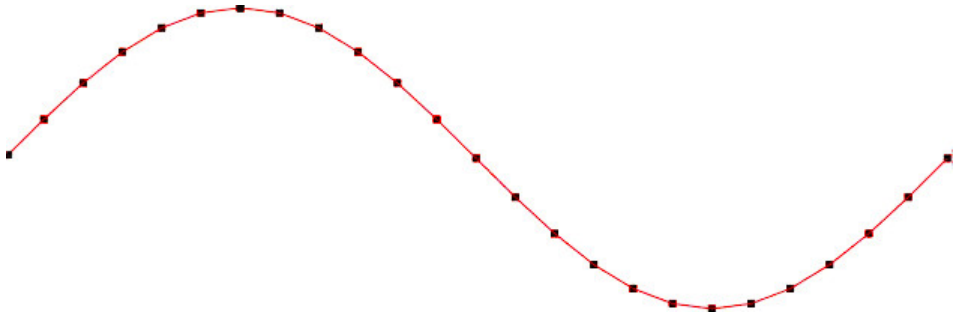


Figura 6.1: Curva NURBS approssimata in rosso ed i suoi punti in nero.

6.2 Constraints utilizzati

Di seguito viene spiegato il funzionamento dei *constraints* utilizzati in fase di animazione, facendo chiarezza sulle funzioni che mettono a disposizione.

Track To Constraint

Questa tipologia di vincolo applica rotazioni all'oggetto che ne fa uso, in modo tale che punti verso un'asse "To" definita, in direzione di un altro oggetto che viene specificato come *target*. Inoltre viene specificata un'asse "Up" mantenuta permanentemente allineata con l'asse globale Z di default (Figura 6.2). Non è possibile selezionare lo stesso asse per il "To" e "Up" poiché il vincolo specificato non funzionerebbe più a dovere.

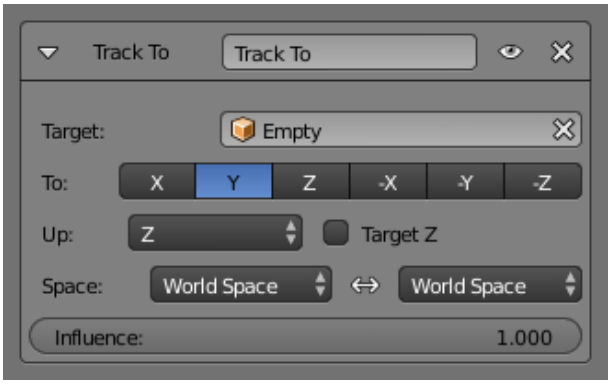


Figura 6.2: Pannello di controllo Track To Constraint

Copy Transforms Constraint

Questo vincolo è molto basilare. Semplicemente costringere l'oggetto che ne fa uso ad avere le stesse traformazioni (rotazioni, traslazioni, ecc.) dell'oggetto specificato come *target* (Figura 6.3).

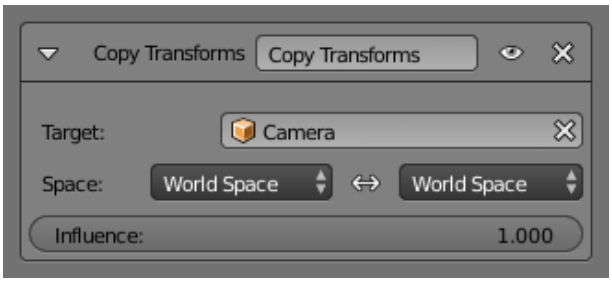


Figura 6.3: Pannello di controllo Copy Transforms Constraint

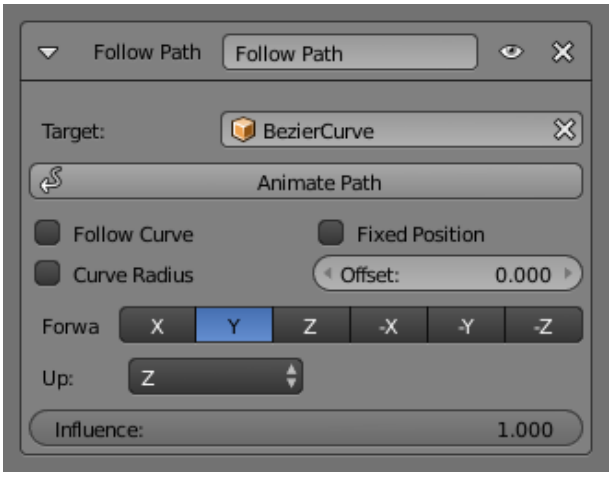


Figura 6.4: Pannello di controllo Follow Path Constraint

Follow Path Constraint

Questo constraint mette in relazione un oggetto con una curva e fa si che l'oggetto si muova lungo la curva (Figura 6.4). Se l'oggetto che lo utilizza ha attiva l'opzione *Follow Curve* verrà vincolato ad essa e la seguirà come se fosse un percorso. L'oggetto che ne fa uso viene valutato sempre in coordinate

globali ed il suo movimento lungo la curva percorso può essere controllato in 2 modi differenti:

- Il modo più semplice è definire il numero di frame di valutazione della curva, ed ogni frame è associato ad un valore di offset (di default, frame iniziale: 1 [= offset di 0)], durata: 100)

- L'altro metodo piu preciso è definire l'*Evaluation Time* della curva attraverso il Graph Editor con l'utilizzo delle *F-Curves*.

6.3 Animazione lungo un percorso

Il disegno della curva *polyline* e l'approssimazione alla curva NURBS, sono il primo passo per la definizione di un percorso sul quale muovere uno o più oggetti all'interno della scena. Ovviamente prima di procedere con l'approssimazione, si deve obbligatoriamente selezionare l'oggetto o gli oggetti, dato che il processo di approssimazione e animazione sono sequenziali dopo la pressione del tasto *;* da tastiera. La sequenza di azioni, che concretizza l'animazione lungo un percorso, è sintetizzata nei seguenti passaggi:

1. Si seleziona la posizione del *3D Cursor* all'interno della scena.

2. Si seleziona l'oggetto o gli oggetti all'interno della scena che si desiderano far muovere.

3. Dopo la selezione alla pressione del tasto *;* inizia il disegno della curva percorso.

4. Attraverso il pannello Animation Tools si personalizzano le impostazioni.

5. Si preme nuovamente il tasto *;* da tastiera per confermare e realizzare il percorso animamato.

Prima di procedere con la descrizione delle dinamiche che portano all'animazione è necessario dare delucidazioni in merito alle opzioni attivabili e selezionabili all'interno del pannello *Animation Tools* (Figura 6.5). Il pannello è suddiviso in tre categorie, ciascuna delle quali offre opzioni differenti:

Approximation: Come detto in precedenza, in questa sezione è possibile impostare i valori di approssimazione (*Smooth, Grado*) della curva disegnata.

Animation: Si ha la possibilità di definire il frame di partenza dell'animazione e la sua durata espressa in secondi.

Object Orientation: In questa sezione si può selezionare il tipo di orientamento dell'oggetto dal menù a tendina, può essere abilitata la durata dell'animazione secondo il tempo di disegno (*Leap Seconds*), si può abilitare l'opzione di velocità costante, si può mantenere in memoria i dati elaborati dal Leap Motion (*Keep Animation Data*) per un successivo utilizzo, ed infine settare la distanza tra un oggetto e l'altro (*Distance*). Quest'ultima disponibile solo per l'opzione di orientamento *Follow Object*.

Utilizzando il Leap Motion per il disegno della curva, la prima domanda che che ci si è fatta è stata: Quali tipo di informazioni potevano essere utili per animare degli oggetti? In risposta a questo, si è pensato di raccogliere le informazioni fornite dal dispositivo relative alla velocità e alla direzione del dito indice nel mentre la curva viene disegnata. Grazie alle API offerte e messe a supporto del dispositivo, il reperimento dei dati è stata cosa semplice, a questo punto si doveva capire come poter trasportare le informazioni ottenute all'interno di Blender. Perciò, il passo successivo è stato quello di acquisire informazioni sulle dinamiche di animazione di Blender, ovvero sulla gestione del tempo di animazione ed in particolar modo su come sia possibile fornire al programma informazioni relative alla velocità e all'orientamento di uno o più oggetti in un arco di tempo.
La maggior parte dei programmi di animazine, Blender incluso, visualizzano

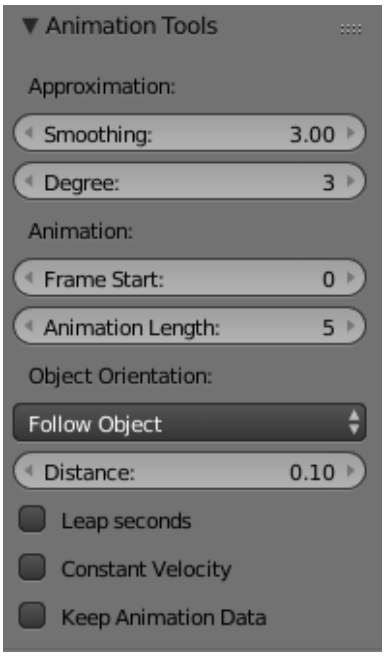


Figura 6.5: Animation Tools: opzioni di Approssimazione, Animazione e Orientamento oggetto

e gestiscono il passaggio del tempo attraverso una barra chiamata *Timeline*. Quest'ultima è graduata in fotogrammi (*frame*) e la sua porzione attiva determina la lunghezza della sequenza temporale, ovvero l'effettiva durata dell'animazione (Figura 6.6). Il numero di fotogrammi rappresenta un intervallo di tempo, ogni fotogramma rappresenta invece un'istante di tempo e la sequenza di più fotogrammi rappresenta un'animazione. Quest'ultima è governata da una *frequenza di fotogrammi* rapportati alla misura del tempo in secondi. Tipicamente lo standard video esprime la *frequenza di fotogrammi* in FPS (*Frame per Seconds*) ed è uguale a 30, ovvero 1 secondo di animazione equivale a 30 fotogrammi. Quindi, come gestire il comportamento di un oggetto in un dato istante di tempo (*frame*)? Per rispondere a questa domanda è necessario introdurre un concetto fondamentale utilizzato da Blender e da tutti i programmi di animazione, ovvero il concetto di fotogramma chiave (*keyframe*) Figura 6.6.
Questi fotogrammi chiave vengono utilizzati per specificare una qualsiasi tipo

di azione, transizione, movimento o rotazione in un preciso istante di tempo. E' possibile quindi decidere il comportamento di un oggetto (velocità, rotazione, posizione, ecc.) inserendo un *keyframe* in un preciso momento e non necessariamente ad ogni istante durante il corso dell'animazione.
La domanda che sorge spontanea è: perché non ad ogni istante? Perché Blender si occupa di calcolare le trasformazioni negli altri fotogrammi non etichettati come *keyframe*. Supponiano ad esempio che un'animazione abbia una durata di 120 fotogrammi, ad un oggetto, nel fotogramma numero 1, potrà essere assegnata una specifica lista di trasformazioni impostando il fotogramma corrente come *keyframe*. L'animatore quindi potrà inserire ulteriori trasformazioni e creare un nuovo *keyframe* per l'oggetto al frame 120. Dopo queste operazioni Blender sarà in grado di creare una transizione omogenea dal frame 1 al frame 120, adattando proporzionalmente le varie trasformazioni nel passaggio tra i vari fotogrammi (processo di interpolazione)[1].

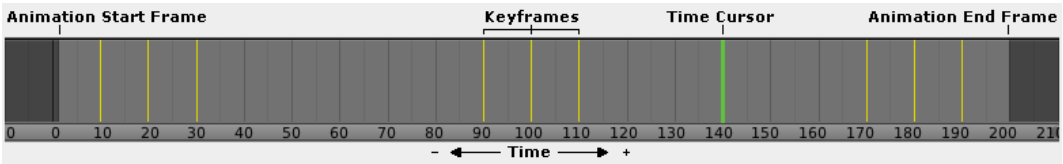


Figura 6.6: Timeline di Blender per la gestione del tempo di un'animazione

A questo punto, una volta chiarite le dinamiche della gestione del tempo all'interno di Blender, si dovevano risolvere i seguenti problemi:

1. Specificare la durata in fotogrammi della curva percorso disegnata.

2. Definire in quale istante e sopratutto quanti *keyframes* inserire.

3. Trovare un modo per impostare la velocità costante e la velocità rilevata durante il disegno della curva.

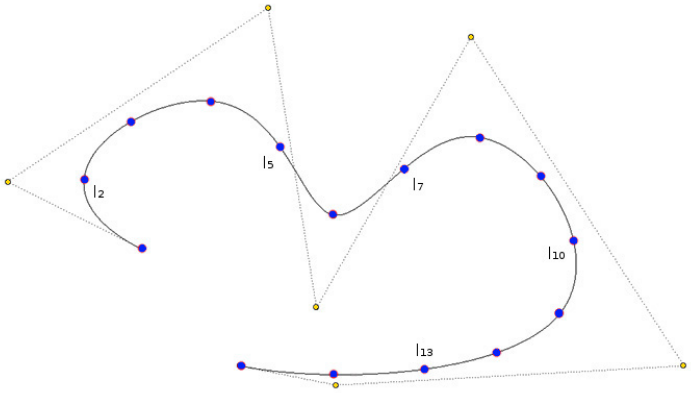
[1]L'interpolazione in Blender viene gestita attraverso le *F-Curves*, modificabili tramite il *Graph Editor*

4. Definire la direzione e le varie trasformazioni da applicare all'oggetto o agli oggetti nel tempo.

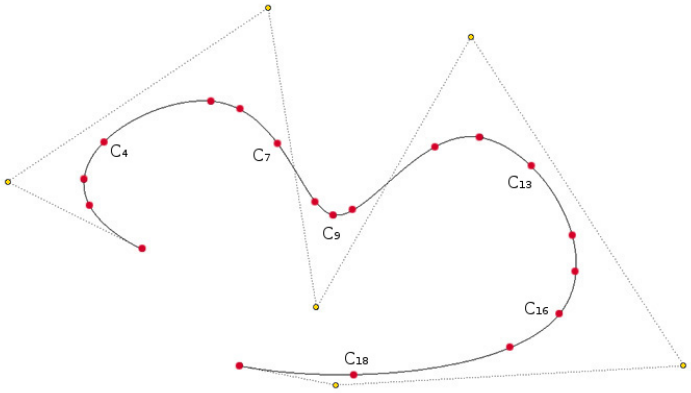
Per risolvere il primo punto è bastato editare il parametro *path duration* interno alla curva disegnata per riuscire ad impostare la durata di animazione del percorso.
Il secondo punto per essere risolto doveva tener conto dei dati più rilevanti raccolti con il Leap Motion, era quindi necessario effettuare un campionamento dei dati lungo la curva che fornisse indicazioni sulla posizione di inserimento dei *keyframes* e sul numero di *keyframes* da inserire. Inizialmente il campionamento, ed il successivo inserimento dei *keyframes*, era ad intervalli regolari che venivano stabiliti a priori (Figura 6.7a). Questo approccio però risultava non essere preciso e creava dei problemi nella successiva valutazione delle traformazioni lungo la curva. Pertanto si è deciso di sfruttare le porprietà della curva stessa analizzando il grado di curvatura C_i nel punto P_i come:

$$C_i = P_i''$$

Per il calcolo della derivata seconda si è utilizzato la funzione *splev* [14] interna al sottomodulo *scipy.interpolate*, che prende in input un array di punti, la curva approssimata in precedenza, ed il valore della derivata. Essa restituisce in output una sequenza di valori che rappresentano la valutazione della funzione della curva nei punti dati in input. Così facendo si ha la possibilità di aumentare il campionamento nei tratti con maggior curvatura, dove è richiesta una maggiore precisione, e diminuirlo invece nei tratti più rettilinei ed omogenei (Figura 6.7b). Più precisamente, una volta ottenuti tutti i valori di curvatura si calcola il picco di massima curvatura della curva e, a partire da questa, vengono ricavate tre fasce differenti che vanno a rappresentare rispettivamente un intervallo di curvatura *Massimo, Medio, Minimo* che determineranno il numero di campionamenti effettuati. Ad esempio, se il grado di curvatura $C_{max} = 300$ le altre due fasce saranno $C_{mid} = 200$ e $C_{min} = 100$ con il numero di campionamenti che varia a seconda della fascia



(a) Intervalli regolari



(b) Grado di Curvatura

Figura 6.7: Campionamento della curva nei tratti con maggior curvatura (pallini rossi) e ad intervalli regolari (pallini blu).

in cui ricade la curvatura C_i nel punto P_i.
Per ogni grado di curvatura C_i:

- se $C_{mid} < C_i \leq C_{max}$ tutti i dati ricavati nel punto P_i vengono immediatamente memorizzati.

- se $C_{min} < C_i \leq C_{mid}$ vengono memorizzati i dati nel punto P_i solo se C_i è il terzo[2] valore analizzato in questo intervallo.

- se $C_i \leq C_{min}$ vengono memorizzati i dati nel punto P_i solo se C_i è il quarto[2] valore analizzato in questo intervallo.

Per risolvere il terzo punto, impostare la velocità di percorrenza, equivale a specificare a Blender lo spostamento effettuato dall'oggetto o dagli oggetti in un dato *frame*. Sulle basi teoriche descritte nella sezione 5.2, il movimento lungo la curva può essere definito al variare del parametro u compreso nell'intervallo [0,1].
Quindi per poter gestire lo spostamento in Blender si deve avere la possibilità di definire il valore parametrico u della curva. Questo può essere effettuato grazie al *Follow Path Constraint*, in particolar modo attraverso l'opzione *Fixed Position* che offre la possibilità di muovere l'oggetto impostando un valore di *offset* compreso tra [0,1], in cui i valori estremi rappresentano l'inizio e la fine del tratto di curva.

Velocità costante

Per impostare la velocità costante per l'attraversamento della curva, è bastato inserire due *keyframes* rispettivamente all'inizio e alla fine della curva con i valori estremi dell'intervallo di *offset*. Così facendo Blender è in grado di calcolare lo spostamento in maniera omogenea lungo tutta la lunghezza della curva.

Velocità variabile

Per settare i cambiamenti di velocità effettuati durante il disegno della curva una volta ottenuta la velocità in un punto P_i tramite il Leap Motion, secondo il campionamento della curvatura C_i, essendo la velocità espressa in millimetri al secondo serviva un metodo che la rapportasse correttamente ad un valore di offset. Si è deciso quindi di calcolare lo spazio percorso in coordinate Leap Motion, per poi rapportarlo, tramite una proporzione, allo

[2]Viene utilizzato un contatore interno che si resetta ogni qual volta si memorizza un valore, in modo tale da riavviare il conteggio.

spazio percorso in termini di *offset*. Quindi oltre alla velocità, direzione e posizione si è tenuto in memoria dal Leap Motion il tempo di disegno t_i per ogni punto P_i campionato. Siccome il tempo registrato in ogni punto è conteggiato a partire dall'inizio del disegno della curva, per poter calcolare correttamente lo spostamento in un tratto di curva dal punto P_{i-1} a P_i si deve calcolare il tempo T_i relativo a quel intervallo come:

$$T_i = t_i - t_{i-1}$$

Successivamente si calcola lo spostamento S_i tra il punto P_i e P_{i-1} e lo spostamento totale S in coordinate di Leap Motion come:

$$S_i = T_i \cdot v_i$$
$$S = (S_i + S_{i+1} \dots S_N)$$

Dove v_i rappresenta la velocità rilevata nel punto P_i.
Una volta ottenuto lo spostamento per ogni punto campionato e lo spostamento totale, viene eseguita la proporzione per calcolare il corretto valore di *offset* O_i come:

$$L_i = \frac{S_i}{S}$$
$$O_i = L_i + O_{i-1}$$

Con L_i che rappresenta la lunghezza espressa in *offset* tra il punto P_i e P_{i-1}. A questo punto la velocità nel punto P_i può essere espressa attraverso lo spostamento O_i, vengono quindi inseriti i vari *keyframes* secondo i campionamenti della curvatura C_i con il corrispondente valore O_i, inserendo sempre i due *keyframes* all'inizio e alla fine della curva con gli estremi dell'intervallo di *offset* ovvero 0 e 1.

Riassumendo, vengono eseguite le seguenti operazioni di base:

1. Si memorizza il tempo ed i vettori di posizione, velocità, direzione ad ogni frame e si normalizzano.

2. Viene calcolato il numero di frame per la durata del percorso moltiplicando i secondi s impostati per 30 frame. (*1s = 30 frame*)

3. Viene calcolata la curvatura in ogni punto della curva e ne viene memorizzata la posizione. Secondo l'intervallo di curvatura *Massimo, Medio, Minimo* si definisce il numero di campionamenti dei vettori precedenti.

4. Si calcola l'offset O_i nel punto P_i secondo il campionamento effettuato in precedenza.

Una volta eseguiti questi calcoli, in base alla selezione del tipo di orientamento vengono eseguite operazioni differenti. Quindi per risolvere il quarto punto, sulle basi teoriche esposte nel paragrafo 5.2.4 sono state implementate quattro modalità differenti che vanno a coprire le tipologie di orientamento illustrate in precedenza, denominate *Frenet Frame, Follow Object, Object Following Path, Center of Interest.*
Per definire l'orientamento e la direzione degli oggetti sono stati utilizzati i *quaternioni* per ovviare al problema del *gimbal lock*. Questi sono rappresentati da una quaterna (w, x, y, z) con (x, y, z) che rappresentano gli assi di rotazione, ed il quarto valore w che indica di quanto l'oggetto è ruotato. Quest'ultimo infatti può essere visto come il "peso" della direzione originale, quindi maggiore sarà il valore di w "più peso" avrà la direzione originale dell'oggetto e meno sarà l'effetto della trasformazione a lui applicata.

6.3.1 Frenet Frame

Questo tipo di modalità da la possibilità di definire l'orientamento di un oggetto grazie alle proprietà della curva disegnata (Figura 6.8). Dopo aver selezionato l'oggetto che si desidera far muovere viene calcolato secondo l'equazione (5.4) il sistema di coordinate (u,v,w) determinato dalla curvatura e dalla tangente della curva, che cambia orientamento lungo l'intera lunghezza. I vettori che lo definiscono sono ortogonali tra loro con w che rappresenta la direzione di vista v che rappresenta l'UP vector e u il vettore ortogonale

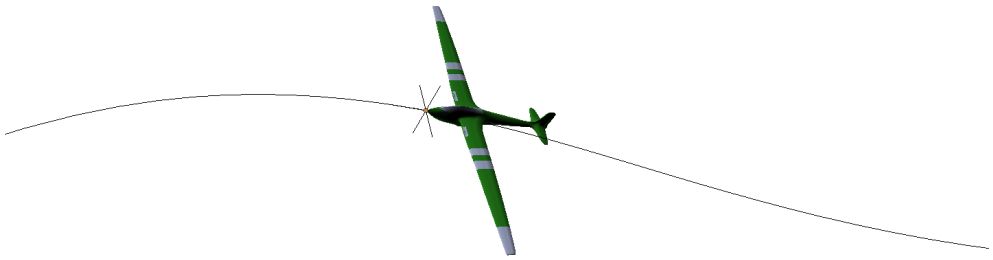


Figura 6.8: Frenet Frame: Orientamento dell'oggetto sulla curva.

a w e al piano definito dalla curvatura e dalla tangente. Successivamente vengono effettuate le seguenti operazioni:

- Viene creato un oggetto *empty* nella posizione dell'origine e impostata la rotazione in QUATERNION.
- Si aggiunge il *Follow Path Constraint* all'oggetto *empty* impostando come target la curva, con l'asse UP settata su l'asse y e si abilita l'opzione *Fixed Position*.
- Si aggiungono i *keyframes* con offset 0 e 1 all'inizo e alla fine.
- Se la velocità costante non è abilitata, per ogni posizione relativa al campionamento effettuato in precedenza viene inserito un *keyframe* con il corretto valore di offset.
- Per ogni posizione relativa al campionamento effettuato in precedenza viene inserito un *keyframe* con il quaternione di rotazione dato dalla terna (u,v,w) in quel punto.

- Vengono copiate le dimensioni dell'oggetto e assegnate all'*empty*. Poi si aggiunge il *Copy Transforms Constraint* all'oggetto selezionato, con target l'*empty*.

Per un corretto orientamento è necessario che all'oggetto selezionato vengano applicate alcune trasformazioni in coordinate locali. L'oggetto deve puntare verso l'asse z negativa e avere l'asse y positiva come up vector.

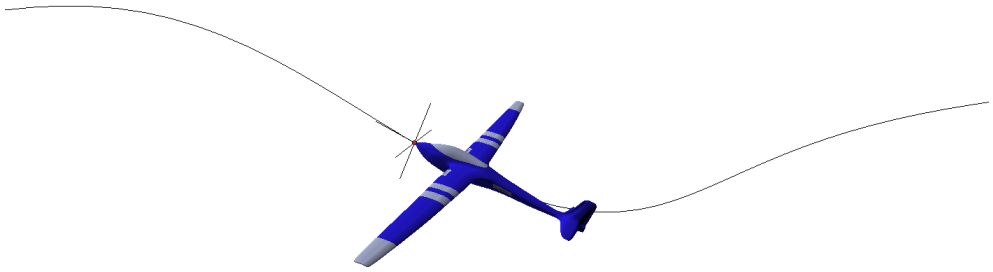


Figura 6.9: Object Following Path: Orientamento dell'oggetto che segue la curva disegnata.

6.3.2 Object Following Path

Questo orientamento è un'alternativa all'implementazione del *Frenet Frame* e si ottiene lo stesso risultato (Figura 6.9). La differenza principale sta nel sistema di coordinate (u,v,w) che, anziché essere calcolato sfruttando le proprietà interne della curva, viene rappresentato dal vettore direzione campionato durante il disegno della curva. In tal maniera si sfruttano direttamente le API di Leap Motion che provvedono a fornire con precisione informazioni utili per l'implementazione. Il procedimento di configurazione

ed impostazione risulta essere identico a quello del *Frenet Frame* descritto in precedenza.

6.3.3 Follow Object

In questa modalità è possibile selezionare due oggetti e muoverli lungo la stessa la curva ad opportuna distanza l'uno dall'altro. Un esempio è il movimento di un oggetto seguito da una camera che lo inquadra (Figura 6.10). All'oggetto seguito dalla telecamera vengono applicate le rotazioni previste per il *Frenet Frame*, secondo l'equazione 5.4, con alcune modifiche al valore di *offset* nei diversi *Keyframes*. Il secondo oggetto si occupa di tracciare il movimento del primo oggetto ed anche lui vengono apportate alcune modifiche al valore di *offset* nei diversi *Keyframes*. Queste modifiche dipendono dal valore della distanza impostato tramite il pannello *Animation tools* (Figura 6.5), che rappresenta lo spostamento nell'intervallo di percorrenza [0,1]. Di seguito sono spiegate nel dettaglio le operazioni eseguite per gli oggetti, quello seguito e la camera che segue.

Oggetto seguito:

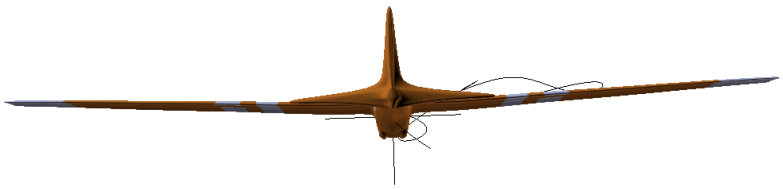
- Viene creato un oggetto *empty* nella posizione dell'origine e impostata la rotazione in QUATERNION.
- Si aggiunge il *Follow Path Constraint* all'oggetto *empty* impostando come target la curva, con l'asse UP settata su l'asse y e si abilita l'opzione *Fixed Position*.
- Si aggiungono i *keyframes* con offset uguale alla metà della distanza impostata all'inizo e 1 alla fine.
- Se la velocità costante non è abilitata, per ogni posizione relativa al campionamento effettuato in precedenza viene inserito un *keyframe* con il valore di offset sommato a metà della distanza settata.

- Per ogni posizione relativa al campionamento effettuato in precedenza viene inserito un *keyframe* con il quaternione di rotazione dato dalla terna (u,v,w) in quel punto.
- Vengono copiate le dimensioni dell'oggetto e assegnate all'*empty*. Poi si aggiunge il *Copy Transforms Constraint* all'oggetto seguito, con target l'*empty*.

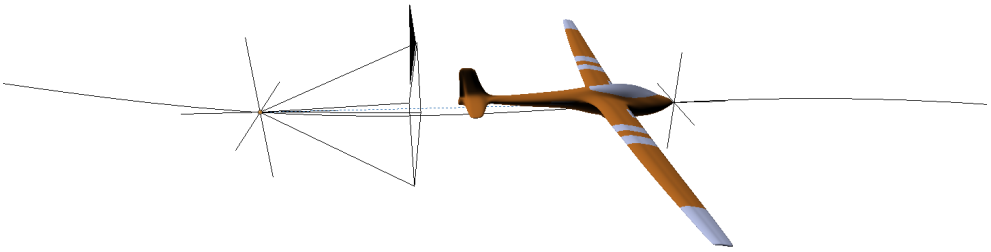
Come detto in precedenza è necessario che all'oggetto seguito vengano applicate le trasformazioni in coordinate locali.
L'oggetto dovrà puntare verso l'asse z negativa e avere l'asse y positiva come up vector.

Oggetto che segue:

- Viene creato un oggetto *empty* nella posizione dell'origine.
- Si aggiunge il *Follow Path Constraint* all'oggetto *empty* impostando come target la curva, con l'asse UP settata su l'asse y e si abilita l'opzione *Fixed Position*.
- Si aggiungono i *keyframes* con offset 0 all'inizo e alla fine 1 meno la metà del valore distanza impostato.
- Se la velocità costante non è abilitata, per ogni posizione relativa al campionamento effettuato in precedenza viene inserito un *keyframe* con il valore di offset sottratto a metà della distanza impostata.
- Viene aggiunto all'oggetto *empty* il *Track To Constraint*, che rappresenterebbe la terna (u,v,w) data dall'equazione 5.6, viene assegnato come target l'*empty* dell'altro oggetto impostando il tracciamento sull'asse $-Z$ e UP sull'asse Y
- Vengono copiate le dimensioni dell'oggetto e assegnate all'*empty*. Poi si aggiunge il *Copy Transforms Constraint* all'oggetto che segue, con target l'*empty*.



(a) Camera View



(b) View 3D

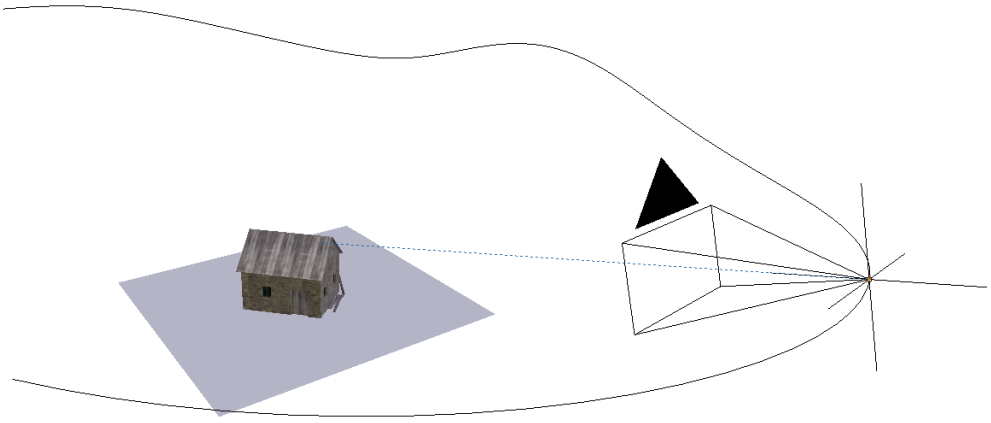
Figura 6.10: Follow Object: Orientamento oggetto Frenet Frame e camera con *Track to*

6.3.4 Center of Interest

Selezionando due oggetti, per esempio una camera e una mesh, si ha la possibilità di focalizzare l'inquadratura di una camera verso un centro di interesse, la mesh o un qualsiasi altro oggetto, mentre la camera si sposta lungo la curva disegnata (Figura 6.11). Il centro di interesse (COI) è direttamente determinato dal vettore di vista, mentre il vettore UP in questo caso può essere visto come la direzione positiva sull'asse y. Di seguito sono spiegate le operazioni eseguite:

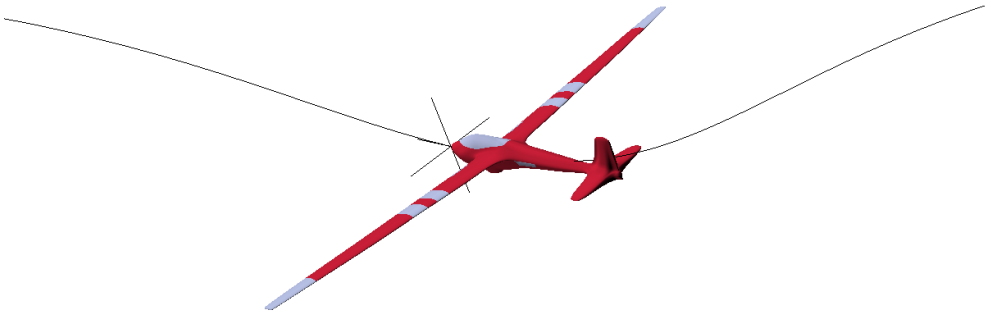
- Viene creato un oggetto *empty* nella posizione dell'origine associato alla camera.
- Si aggiunge il *Follow Path Constraint* all'oggetto *empty* impostando come target la curva, con l'asse UP settata su l'asse y e si abilita l'opzione *Fixed Position*.
- Si aggiungono i *keyframes* con offset 0 all'inizo e 1 alla fine.
- Se la velocità costante non è abilitata, per ogni posizione relativa al campionamento effettuato in precedenza viene inserito un *keyframe* con il corretto valore di offset.
- Viene creato un altro oggetto *empty* nella posizione dell'origine associato al centro di interesse e vengono copiate le dimensioni, rotazioni e posizione dell'oggetto e assegnate all'*empty*.
- Viene aggiunto all'oggetto *empty* della camera il *Track To Constraint*, che rappresenterebbe la terna (u,v,w) data dall'equazione 5.5, viene assegnato come target l'*empty* del centro di interesse impostando il tracciamento sull'asse $-Z$ e UP sull'asse Y
- Poi si aggiunge il *Copy Transforms Constraint* ad entrambi gli oggetti, con target i rispettivi *empty*.

(a) Camera View

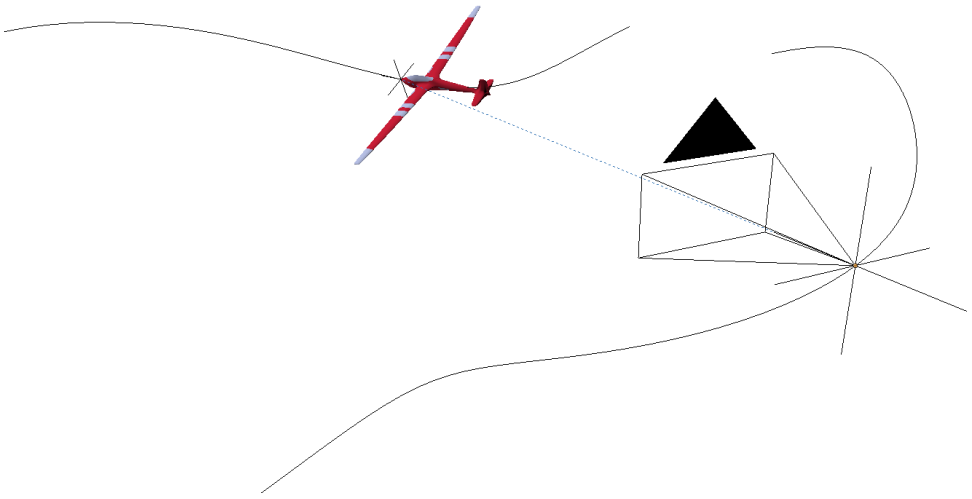


(b) View 3D

Figura 6.11: Center of Interest: Oggetto tracciato dalla camera con *Track to*



(a) Camera View



(b) View 3D

Figura 6.12: Following Object along Curve: Orientamento oggetto Frenet Frame tracciato dalla camera con *Track to*

6.3.5 Following Object along Curve

Questa modalità non è direttamente selezionabile attraverso il pannello *Animation Tools*, ma può essere ottenuta con la combinazione delle altre già presenti. L'obietttivo è quello di tracciare il movimento di un oggetto che si sposta lungo un percorso tramite una telecamera che si sposta lungo un percorso differente, come esposto nel paragrafo 5.2.4 attraverso l'equazione 5.7. Per far ciò supponiamo di aver già realizzato un percorso in cui un oggetto sfrutta il *Frenet Frame* per muoversi lungo la curva. A questo punto basta avviare nuovamente il disegno di un'altra curva percorso e selezionare la modalità *Center of Interest* selezionando come input la camera e l'oggetto che si desidera tracciare. Il risultato è visibile in Figura 6.12.

6.3.6 Gestione dell'animazione

Dopo aver messo in luce le dinamiche con cui si è arrivati a generare una curva animata percorsa da uno o più oggetti, di seguito vengono fatti alcuni chiarimenti in merito alle possibilità offerte. La procedura descritta in cinque passi all'inizio di questa sezione permette di ultimare le operazioni di animazione ed orietamento descritte in precedenza. Quindi, una volta confermato con il tasto *;* da tastiera, viene realizzato il percorso animato, ed attraverso la *timeline* (Figura 6.6), si possono visionare i *keyframes* inseriti in precedenza analizzando frame per frame il movimento degli oggetti o dell'oggetto.
In Figura 6.13 viene illustrato la barra di controllo della *timeline* per la riproduzione e la gestione dell'animazione in Blender.
Quest'ultimo è suddiviso in 5 caselle:

1. E' un pannello per il controllo e la riproduzione dell'animazione in un intervallo di frame stabilito. Molto utile per analizzare e riprodurre solo un tratto di animazione impostando un *Preview range*.

2. E' il pannello di controllo dei frame, le prime due caselle specificano il frame di inizio e di fine dell'animazione, mentre l'ultima specifica l'attuale frame selezionato con il *Time Cursor* (Figura 6.6).

3. I bottoni presenti al suo interno permettono di controllare il *Time cursor*. Ovvero si può avanzare/indietreggiare frame per frame, oppure riprodurre in sequenza tutti i frame che compongono l'animazione, lasciando la possibilità di decidere se la riproduzione è sequenzialmente in avanti (Start to End) oppure all'indietro (End to Start).

4. Questo pannello serve per sincronizzare l'animazione in caso di rallentamenti eccessivi durante la sua riproduzione, con la possibilità di impostare il *Frame Rate* attuale quindi i suoi FPS (Frame per Seconds).

5. E' un pannello per la gestione dei *keyframes* all'interno della scena. Attraverso questo è possibile editare quelli già presenti, oppure inserirne di nuovi grazie alla funzione di *Auto Keyframe*. Quest'ultima, attraverso il pulsante rosso di registrazione, aggiungera o sostituirà i *keyframes* presenti per l'oggetto attivo quando viene eseguita una trasformazione qualsiasi in scena.

Figura 6.13: Barra di controllo della Timeline di Blender

Se si è soddisfatti del percorso e dell'animazione creata, come spiegato poc'anzi, attraverso il pannelo in Figura 6.13 può provvedere a visualizzare in scena l'animazione. E se si volesse modificare l'orientamento, o gli oggetti da animare, o la velocità, mantenendo lo stesso percorso disegnato? Il tutto è possibile se l'opzione *Keep Animation Data*, nel pannello *Animation tools* (Figura 6.5), è stata attivata. Quindi, per poter effettuare le modifiche elencate sopra si devono eseguire queste operazioni in sequenza:

1. Si selezionano l'oggetto o gli oggetti[3] che si desidera far muovere.

[3]Se l'oggetto o gli oggetti sono differenti rispetto a quelli precedenti e si vuole rimuoverli dal percorso, manualmente si deve rimuovere il *Follow Path Constraint* dall'empty associato o eliminare quest'ultimo

2. Si seleziona la curva precedentemente disegnata.

3. Si apportano le modifiche volute attraverso il pannello *Animation tools*.

4. Si preme il tasto *;* da tastiera per confermare ed apportare le modifiche.

Capitolo 7

Validazione delle funzioni implementate

In questo capitolo, verrà fatta un'analisi di valutazione delle funzioni implementate, con l'obiettivo di mettere in risalto gli aspetti positivi e negativi che si sono incontrati durante il loro utilizzo. Successivamente invece verranno descritti i passaggi chiave che hanno portato alla realizzazione della scena che fa da cornice all'animazione creata.

7.1 Semplicità e feedback

Per la valutazione delle funzioni implementate con Leap Motion, seguendo l'impostazione già utilizzata in precedenza [6], si sono tenuti in considerazione due fattori principali: la *semplicità* nell'utilizzo ed il *feedback* ricevuto. Con *semplicità* di utilizzo si intende la comodità con cui si eseguono le operazioni, ed in particolar modo il tempo impiegato per finalizzarle.
Il tutto viene messo a confronto con le tempistiche che si avrebbero se si utilizzassero le funzioni native di Blender. L'obiettivo quindi è quello di dare una risposta alle seguenti domande:

- Nell'eseguire un'azione è più semplice utilizzare la funzione dell'addon o quella nativa di Blender?

- Attraverso il Leap Motion è possibile eseguire le medesime azioni che si possono fare con mouse e tastiera?
- Qual'è la soluzione che permette di raggiungere il risultato voluto nel minor tempo possibile?

Si tiene conto quindi, nella valutazione finale, del numero di azioni che si devono eseguire per raggiungere l'obiettivo prefissato.
Per *feedback* ricevuto invece si valuta il livello di precisione e correttezza delle delle funzioni all'interno dell'addon, a confronto con quelle già interne a Blender. Si cercherà quindi di rispondere alle seguenti domande:

- Qual'è lo sforzo fisico che si deve fare per eseguire un'azione?
- Quanto risulta essere intuitivo eseguire un'azione con Leap Motion?
- Qual'è la soluzione che con la maggior precisione permette di raggiungere il risultato voluto?

Tenendo conto di questi due criteri, vengono analizzate le funzioni implementate mettendole a confronto con l'iter nativo previsto da Blender.

7.1.1 Validazione strumenti di navigazione

In questo paragrafo vengono analizzate le funzioni di navigazione nella scena implementate nell'addon e precedentemente descritte nella sezione 4.2, mettendole singolarmente a confronto con quelle presenti di default in Blender.

Modifica della vista utente

La vista utente viene mossa molto frequentemente, venendo ruotata o spostata ad ogni istante. Utilizzando le funzioni interne a Blender per ruotare la scena sono richieste due azioni tipicamente: il click di due tasti del mouse ed il successivo spostamento. Per traslare la vista sono necessarie invece tre azioni: premere il tasto *MAIUSC* sulla tastiera, click con i due tasti del

mouse e lo spostamento. Invece il movimento della vista utente usando il dispositivo Leap Motion può essere effettuato grazie alla mano sinistra.

Mano Sinistra: La rotazione e ed il movimento della vista utente sono governati dalla mano sinistra attraverso una posa di riconoscimento iniziale data da *indice, pollice, medio*, è richiesta quindi una sola azione per eseguire gli spostamenti e le rotazioni. Tuttavia, in questo caso, risulta essere leggermente più *semplice* gestire rotazione e spostamento con le funzioni interne a discapito dell'utilizzo del Leap Motion.
Questo è dovuto alla poca *precisione*, che viene a mancare quando ci si imbatte in alcuni movimenti, ovvero se il sensore non rileva per un istante o in modo scorretto la posa di riconoscimento iniziale può causare traslazioni o rotazioni indesiderate. Proprio in supporto a questo era stato aggiunto, alla pressione del tasto P da tastiera, la possibilità di attivare/disattivare il movimento, impedendo il verificarsi di queste rotazioni o traslazioni indesiderate, sopratutto nella fase di rimozione della mano dal raggio d'azione del sensore.

7.1.2 Validazione editing mesh poligonali

In questo paragrafo vengono analizzate le funzioni, implementate nell'addon, di selezione e modifica di una mesh all'interno della scena. In riferimento alla descrizione fatta nella sezione 4.3, le funzioni verranno messe singolarmente a confronto con quelle presenti di default in Blender.

Selezione e Modifica di Vertici

La selezione o deselezione di un vertice è una delle funzioni principali di Blender che possono essere eseguito con una sola azione: click del mouse. Il problema sorge nel momento in cui si devono selezionare più vertici, per fare questo ci sono due soluzioni: si può procedere con il singolo click per ognuno di quelli interessati quindi molteplici azioni per la selezione, oppure si possono sfruttare i selettori circolari o rettangolari, tramite la pressione di

un tasto per attivarli ed un'altro per definirne il raggio di selezione. Utilizzando il Leap Motion è possibile selezionare uno o più vertici attraverso una sola azione grazie allo spostamento di due dita della mano destra. I vertici selezionati muovendo le dita, sono vincolati alla grandezza del puntatore utilizzato, quest'ultimo modificabile con una azione tramite il movimento della mano aperta.
Inizialmente Blender non offre la possibilità di selezionare dei vertici totalmente oscurati da altri con una sola azione, mentre sfruttando il Leap Motion si ha la possibilità di selezionare anche in profondità. In questo caso però la soluzione nativa risulta essere leggermente più *semplice* nella selezione di singoli vertici, ma più lenta nella selezione di molteplici vertici. La *precisione* invece risulta essere abbastanza buona, e consente di selezionare qualsiasi vertice della mesh anche singolarmente.

Per la modifica dei vertici, attraverso *proportional editing*, con le funzioni native di Blender sono richieste tre azioni: attivazione del *proportional editing*, click del mouse e spostamento dei vertici selezionati. Con il Leap Motion è possibile spostare i vertici con una azione attraverso il semplice movimento del dito indice. Per variare il raggio di azione del *proportional editing* in Blender si usa una azione: la rotazione della rotellina del mouse. Anche con Leap Motion c'è bisogno di una singola azione per ingrandire o diminuire il raggio, lo spostamento di due dita della mano sinistra verticalmente. Confrontando le azioni, la funzione implementata risulta essere più *semplice* rispetto alla funzione nativa in Blender.
La *precisione* è abbastanza buona e consente qualsiasi spostamento, ma quella offerta dalle funzioni native risulta essere leggermente maggiore.
Si potrebbe sfruttare la combinazione dei due per apportare le modifiche scelte, il Leap Motion per la libertà di movimento nello spazio tridimensionale ed il mouse per quelle minime correzzioni di precisione.

7.1.3 Validazione delgli strumenti di animazione

Di seguito verranno valutate ed analizzate le funzioni implementate nell'addon riguardanti il disegno della curva percorso per il moto di uno o più oggetti all'interno della scena, in riferimento alla descrizione effettuata nel corso dei Capitoli 5 e 6. Ciascuna opzione e modalità di orientamento viene messa a confronto con quelle messe a disposizone di default da Blender, valutandone l'effettiva efficienza ed usabilità. Verranno mostrate inoltre delle immagini rappresentanti alcuni frame delle animazioni create all'interno di una scena. Il processo di modellazione volto alla creazione di quest'ultima verrà descritto nella sezione 7.2.

Disegno di una curva percorso

Blender offre nativamente la possibilità di disegnare una curva polyline, ma non una curva NURBS. Il desegno della polyline è strutturato in maniera differente rispetto a quanto implementato e segue i seguenti passi:

1. Nella sceda *Grease Pencil* dal pannello degli strumenti premere *Draw*.
2. Mantenere la pressione sul tasto sinistro del mouse per spostare il puntatore e disegnare la curva.
3. Premere su *Convert - Polygon Curve* nella scheda di *Grease Pencil*.
4. All'interno della stessa scheda premere *Erase*.
5. Mantenere la pressione sul tasto sinistro del mouse per spostare il puntatore e cancellare la curva disegnata.

Una curva polyline però non può essere utilizzata come percorso, occorre quindi creare una curva NURBS. Per poter creare una curva NURBS ed usarla come percorso in Blender si devono seguire i seguenti passi:

1. Si aggiunge in scena un oggetto di tipo *Path*.[1]

[1]NUBRS molto semplice da editare ed ideale per la creazione di percorsi.

2. Si entra in *Edit Mode*, si inizia ad estrudere i punti di controllo della curva per modificarne la lunghezza.

3. Si modifica la posizione dei punti di controllo per ottenere la forma desiderata.

Dunque, analizzando il numero di azioni, nativamente il disegno della polyline a mano libera richiederebbe circa sette azioni. La creazione della curva NURBS invece richiede un minimo di tre azioni, ma può arrivare fino ad un numero indefinito di N azioni a seconda delle modifiche effettuate nel punto due e nel punto tre.
Con Leap Motion si può eseguire questa operazione in tre azioni: pressione del tasto *;* sulla tastiera, si disegna il percorso con il dito indice, si preme nuovamente il tasto *;* per confermare. I metodi forniti da Blender per il disegno della curva polyline e la creazione della curva NURBS risultano essere abbastanza semplici, però risultà ancora più *semplice* ed intuitivo l'utilizzo del Leap Motion. Inoltre il procedimento sfruttando il Leap Motion é notevolmente più veloce rispetto al metodo nativo sia per una curva che per l'altra.
Per quanto riguarda la *precisione* nel disegno della curva, è nettamente superiore quella del mouse rispetto a quella con il dispositivo. Infatti, può succedere che in alcuni frame lo spostamento possa essere erroneamente rilevato portando a dei salti di continuità nel disegno della polyline, questi però vengono corretti in fase di creazione della curva NURBS approssimante. Durante il disegno della curva con Leap Motion lo sforzo fisico è sicuramente maggiore rispetto al movimento del mouse, ma non influisce negativamente sul suo utilizzo, pertanto viene ribadito che il disegno della curva percorso risulta essere semplice ed intuitivo se fatto con il dispositivo.

Creazione del moto e dell'Animazione

In Blender per creare il moto di uno o più oggetti lungo un percorso per una successiva animazione in maniera nativa, sono necessari più azioni che

possono essere racchiuse in due categorie:

Impostazione del percorso: Questo passaggio prevede l'impostazione di durata del percorso, selezione di uno o più oggetti e l'inserimento del *Follow Path Constraint* con il settaggio delle impostazioni.
Quindi le azioni previste sono:

1. Si seleziona la curva percorso e tramite il pannello laterale, nelle proprietà dell'oggetto curva alla sezione *Path Animation*, si imposta il numero di frame di durata del percorso.

2. Si seleziona l'oggetto o gli oggetti da muovere singolarmente e per ognuno di essi si aggiunge il *Follow Path Constraint* e si imposta la curva come target, si abilita l'opzione *Follow Curve*, *Fixed Position* ed infine si preme il pulsante *Animate Path*.

Questa procedura come specificato nel secondo punto deve essere eseguita per ogni oggetto singolarmente all'interno della scena. Quindi se indichiamo con O il numero degli oggetti, il numero di azioni totale T_p per l'impostazione del percorso da animare è $T_p = 2 \cdot O$.

Impostazione delle trasformazioni: Questo prevede l'inserimento nei diversi frame di tutte le informazioni relative allo spostamento (velocità) e all'orientamento di uno o più oggetti. Le modifiche vanno applicate singolarmente per ogni oggetto che si desidera far muovere:

1. Attraverso la *Timeline* di Blender si muove il *Time Cursor* per selezionare il frame attivo sul quale memorizzare le trasformazioni.

2. Dopo aver selezionato l'oggetto si deve impostarne lo spostamento. Questo avviene tenendo premuto il tasto sinistro del mouse muovendolo lungo il percorso, oppure si può modificare direttamente il valore di *offset* dal pannello del *Follow Path Constraint*.

3. Con il mouse ci si posiziona sul valore di *offset* impostato nel pannello del *Follow Path Constraint*, click con il tasto destro del mouse e si seleziona la voce *Insert Keyframes* per impostare il fotogramma chiave.

4. Con l'oggetto selezionato, attraverso la barra laterale nella sezione *Transform* si seleziona il tipo di rotazione da considerare (*QUATERNION* o *EULER*). Si modifica la quaterna o la terna con i valori di rotazione che si vogliono impostare.

5. Con il mouse ci si posiziona su ogni valore di rotazione impostato nella sezione *Transform*, click con il tasto destro del mouse e si seleziona la voce *Insert Keyframes* per impostare il fotogramma chiave.

La procedura descritta precedentemente mette in luce come la procedura di inserimento di un singolo *keyframe*, per i relativi spostamenti e trasformazioni, richieda diversi passaggi. La procedura infatti richiede di essere ripetuta più volte a seconda del numero di *keyframes* che si vogliono inserire. Quindi, se K è il numero di *keyframes* da inserire, il numero di azioni totali T_t per l'impostazione delle trasformazioni di un oggetto è $T_t = 5 \cdot K$. Di conseguenza il numero totale di azioni T delle due categorie è $T = T_p + T_t$.

La funzione di moto lungo un percorso implementata in Blender con il dispositivo Leap Motion risulta senza dubbio essere molto più veloce e *semplice* rispetto alla procedura nativa di Blender. La feauture garantisce un grosso risparmio in termini di tempo, dato che processa automaticamente l'inserimento dei *keyframes* in base alle impostazioni di orientamento selezionate. Il numero di azioni richieste infatti è limitato a cinque, un valore nettamente inferiore a T, e prevede: la selezione dell'oggetto o degli oggetti, la pressione del tasto *;* sulla tastiera, il disegno del percorso con il dito indice, selezione con il mouse delle impostazioni desiderate nel pannello *Animation Tools*, si preme nuovamente il tasto *;* per confermare.
A conferma della maggior velocità di esecuzione delle operazioni con Leap Motion, c'è anche il fatto che si ha la possibilità di animare in contemporanea

due oggetti secondo un determinato orientamento. In aggiunta, c'è da sottolineare il fatto che, nel conteggio delle azioni da parte delle funzioni native sono stati omessi volontariamente la creazioni di oggetti *empty*, utili per un maggior controllo delle trasformazioni, e l'utilizzo ed inserimento del *Copy Transform Constraint* e del *Track To Constraint* per certi tipi di orientamento. In termini di *precisione* si può far riferimento alle valutazioni espresse per la curva percorso, mentre la valutazione della *precisione* dei diversi metodi di orientamento viene discussa singolarmente in seguito con qualche esempio.

Center Of Interest

In Figura 7.1 è possibile vedere una sequenza di frame rappresentanti questa modalità. I render delle immagini sono stati fatti ad una distanza di 50 frame l'uno dall'altro per dare l'idea del movimento in maniera più netta. Come si può notare dalle immagini la camera che si muove lungo la curva percorso disegnata, ha come centro di interesse il paesaggio montuoso creato e con l'avanzare del tempo quest'ultimo rimane invariato. In termini di *precisione* l'opzione *Center of Interest* soddisfa pienamente le aspettative.

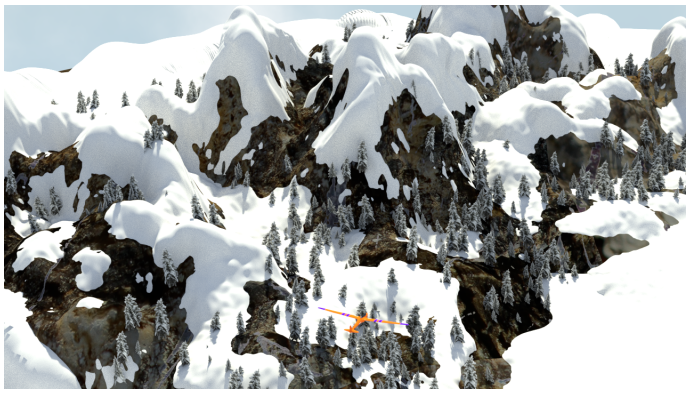
Follow Object e Frenet Frame

Osservando la Figura 7.2 è possibile vedere una sequenza di frame rappresentanti il moto lungo un percorso di una camera che segue un aereo con orientamento *Frenet Frame*. Anche qui i render delle immagini sono stati fatti ad una distanza di 50 frame l'uno dall'altro per dare l'idea del movimento in maniera più netta. Osservando le immagini si può vedere come la camera inquadri sempre l'aereo che si muove lungo la curva percorso disegnata, quest'ultimo con orientamento *Frenet Frame* e con l'avanzare del tempo cambia anche la sua posizione. In termini di *precisione* l'opzione *Follow Object* soddisfa le aspettative, in particolar modo il movimento a seguire della camera risulta essere molto buono e anche l'orientamento *Frenet Frame* dell'aereo che segue la curva. Tuttavia per una reale simulazione del

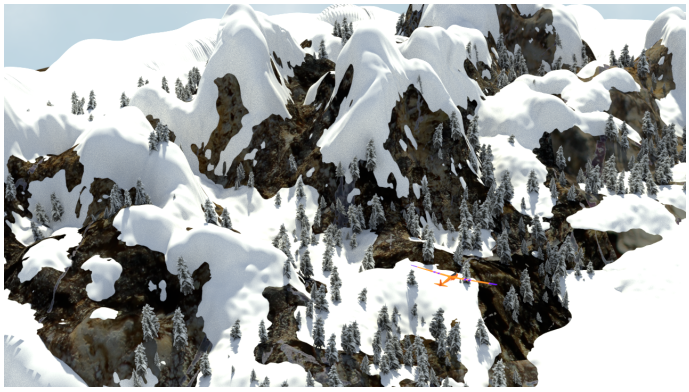
moto dell'aereo sarebbero necessari uteriori accorgimenti nella rotazione, ad esempio l'inclinazione laterale che suggerisce il campio di rotta.

Following Object along Curve

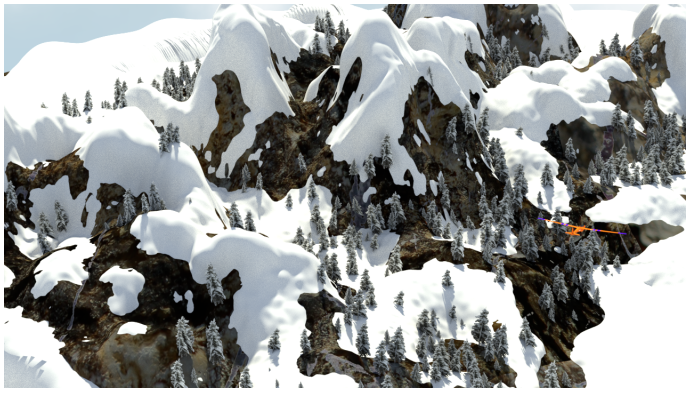
La Figura 7.3 illustra una sequenza di frame rappresentanti l'orientamento *Following Object along Curve* che prevede l'inquadratura da parte di una camera che si muove lungo un percorso, di un altro oggetto che si muove su un percorso differente. Come per le precedenti immagini i render sono stati fatti ad una distanza di 50 frame l'uno dall'altro per dare l'idea del movimento in maniera più netta. Le immagini mostrano come la camera che si muove lungo la sua curva percorso abbia l'inquadratura fissa sull'aereo che si muove anch'esso lungo un'altra curva percorso con orientamento *Frenet Frame*, e l'avanzare del tempo mantiene il rapporto tra i due. In termini di *precisione* per il moto dell'aereo valgono le considerazioni fatte precedentemente per il *Frenet Frame*, mentre il moto della camera con focus sull'aereo valido per l'opzione *Following Object along Curve* risulta essere preciso soddisfando pienamente le aspettative.



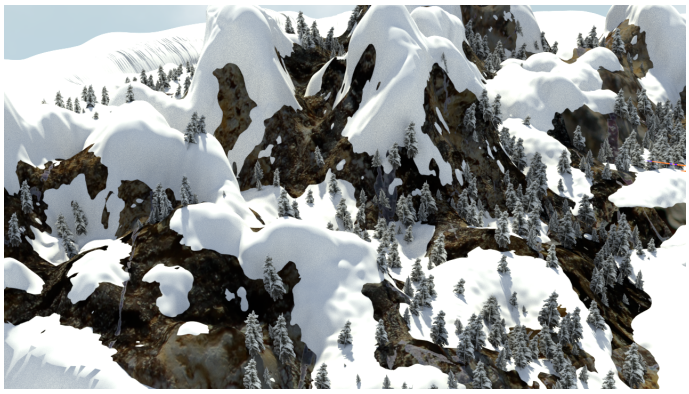
(a) Center of Interest al frame 0



(b) Center of Interest al frame 50

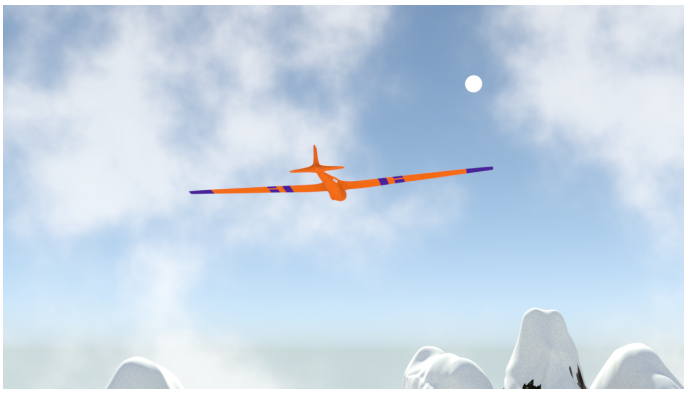


(c) Center of Interest al frame 100



(d) Center of Interest al frame 150

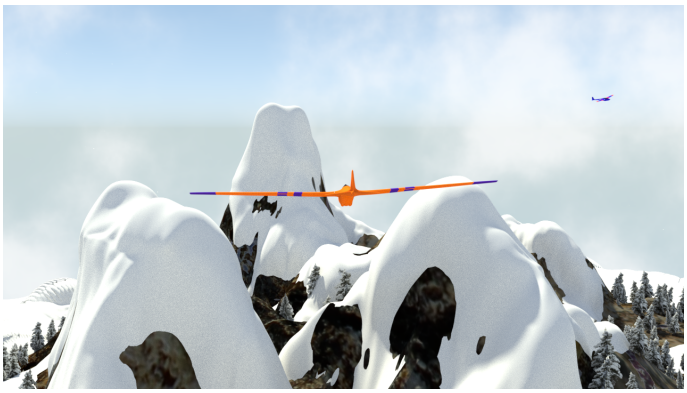
Figura 7.1: Orientamento *Center of Interest*.



(a) Follow Object al frame 0

(b) Follow Object al frame 50

(c) Follow Object al frame 100



(d) Follow Object al frame 150

Figura 7.2: Orientamento *Follow Object* e *Frenet Frame*

(a) Following Object along Curve al frame 0

(b) Following Object along Curve al frame 50

(c) Following Object along Curve al frame 100

(d) Following Object along Curve al frame 150

Figura 7.3: Orientamento *Following Object along Curve*

7.2 Creazione della Scena

Oltre alla programmazione e allo sviluppo delle funzioni offerte, parte del lavoro è stato dedicato all'approfondimento delle dinamiche di modellazione e gli strumenti messi a disposizione da Blender, con lo scopo di realizzare una scena che potesse validare le funzioni implementate.

Modifiers Utilizzati

Di seguito viene spiegato il funzionamento dei *modifiers* utilizzati in fase di modellazione della scena, elencando le loro caratteristiche.

Subdivision Surface Modifier

Questo modificatore viende utilizzato per dividere le facce che compongono una mesh in porzioni più piccole, facendo apparire la mesh più arrotondata e *smooth*. Questo da la possibilità di lavorare con mesh complesse a partire da un basso contenuto di vertici e facce. Inoltre è possibile modellare mesh ad alta risoluzione senza salvare o mantenere in memoria necessariamente una grande quantità di dati. E' possibile selezionare il grado di suddivisioni, nella vista 3D ed in fase di render, attraverso il pannello di controllo del modificatore (Figura 7.4).

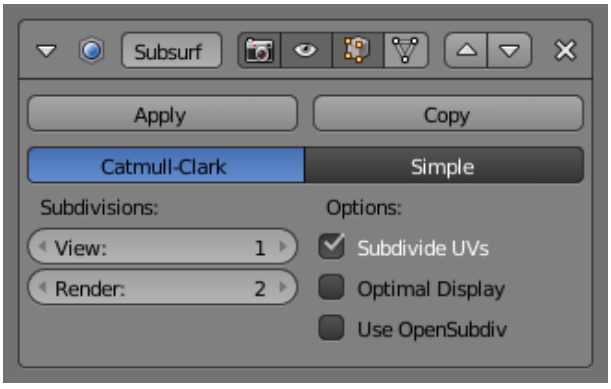


Figura 7.4: Pannello di controllo Subdivision Surface Modifier

Displace Modifier

Il Displace Modifier si occupa di dislocare i vertici di una mesh a partire dall'intensità di una texture datagli in input (Figura 7.5). La dislocazione dei vertici può essere effettuata lungo una particolare asse, lungo la normale dei vertici stessi, oppure può essere sfruttata la componente RGB della texture per dislocare i vertici nelle direzioni locali X, Y e Z in maniera simultanea (Vector Displacement).

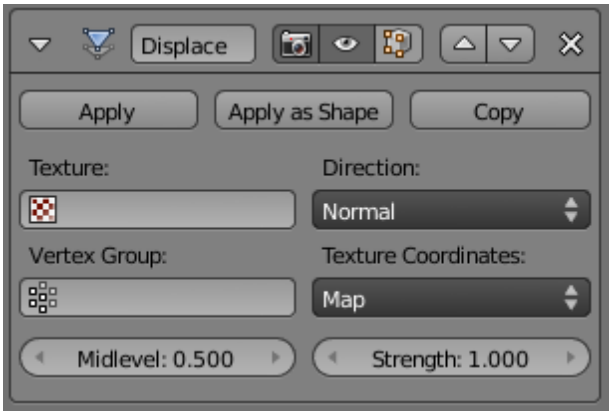


Figura 7.5: Pannello di controllo Displace Modifier

Particle System Modifier

Questo modificatore permette di attivare, nell'oggeto che ne fa uso, la gestione di sistemi particellari. Esso può essere utilizzato per realizzare differenti scenari per la gestione di un oggetto che si desidera generare o muovere casualmente più volte. Risulta infatti essere perfetto per la creazione di foreste, erba, neve, nuvole ecc.

Modellazione della scena

Una delle parti più e coinvolgenti del lavoro svolto in questa tesi è stata la fase di modellazione, nella quale si è dato sfogo alla fantasia e ci si è messi alla prova. Gli strumenti creati durante lo sviluppo dell'addon, come

introdotto in precedenza, sono stati pensati per animazioni di oggetti e camere lungo percorsi definiti da curve. Tenendo conto di ciò era necessario creare una scena abbastanza realistica che potesse includere e far risaltare nel migliore dei modi queste funzioni. Si è scelto quindi di modellare la rappresentazione di un paesaggio montano che facesse da cornice al movimento di alcuni aerei [23] all'interno della scena (Figura 7.6). Di seguito verranno

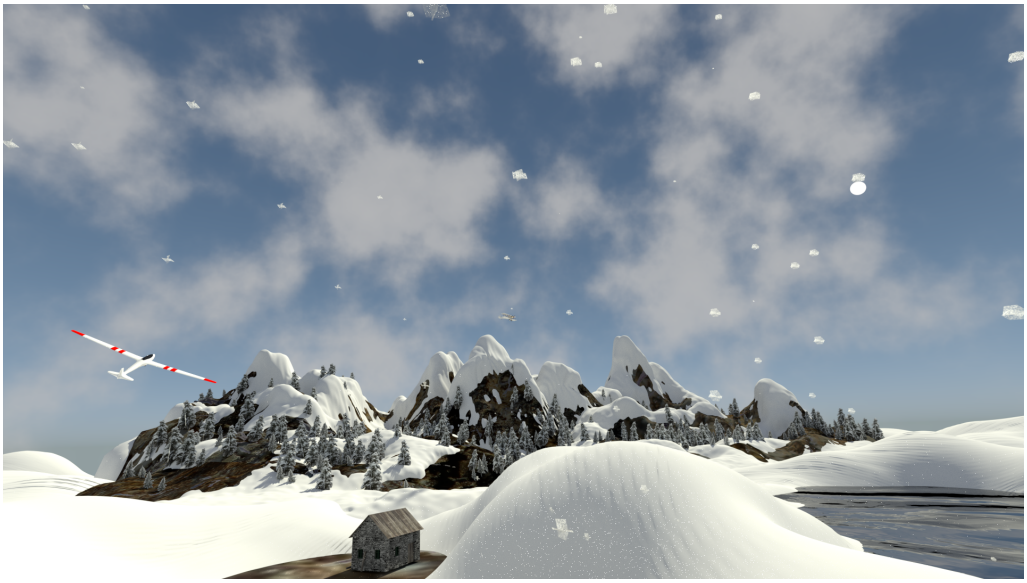


Figura 7.6: Render della scena realizzata

spiegati i passi chiave per la realizzazione dello scenario. Per una maggiore precisione nella creazione di alcune parti della scena, si è fatto riferimento ad alcuni dei numerosi tutorial presenti online, ed il motore di rendering che si è scelto di utilizzare è il Cycles Render basato sul ray-tracing che offre ottime performance nella gestione dei materiali e degli effetti attraverso il *Node Editor*, oltre che la possibilità di sfruttare direttamente la GPU in fase di rendering. Inizialmente si è partiti con il creare le montagne attraverso la mesh *landscape* fornita direttamente da Blender.
Una volta creata compare un pannello nella barra degli strumenti attraverso il quale è possibile decidere la grandezza, il numero di suddivisioni presenti di default, la base di partenza ed il seme casuale delle creste. Con la modifica di

questi valori si può vedere dinamicamente come la mesh *landscape* si modelli a seconda dei parametri impostati, ed una volta trovata la configurazione che si gradisce basta premere il tasto invio da tastiera per confermare le modifiche. Il passo successivo è stata la creazione dei pini innevati, il tutto grazie all'utilizzo di un paio di immagini di due diverse tipologie di pini. Sono state importate all'interno di Blender come piano immagine, sfruttando l'opzione apposita, per poi essere poste perpendicolarmente al piano formato dagli assi X ed Y della scena 3D. Per dare il senso di tridimensionalità sono stati duplicati gli oggetti nella stessa posizione per poi essere ruotati di 90° lungo l'asse Z in modo tale che i piani immagine si incrociassero a vicenda per poi unirli in un unico oggetto ed aggiungerli ad un *Gruppo* (Figura 7.7). Una

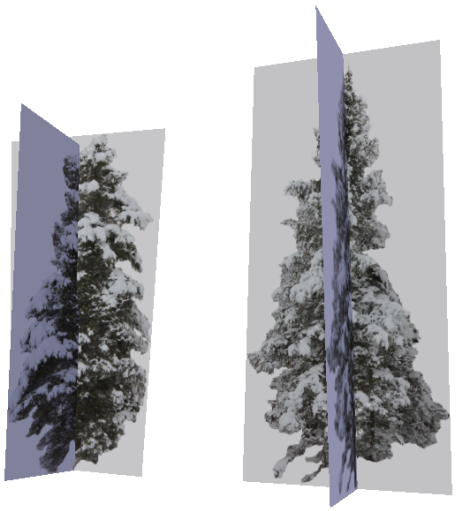


Figura 7.7: Realizzazione dei Pini innevati

volta creati gli alberi, per porli alla base delle montagne create, dopo aver selezionato la mesh *landscape* si è utilizzata la modalità *Weigth Painting* che permette di colorare ed assegnare dei "pesi" lungo la superficie della mesh attraverso un pennello che evidenzia le zone e la densità con cui saranno presenti i pini precedentemente creati. Si è quindi utilizzato il *Particle System Modifier* di tipo *Hair*, applicato alla mesh *landscape*, per il posizionamento

dei pini secondo la colorazione effettuata in precedenza dando in input al modificatore il *Gruppo*. Dopo aver posizionato la vegetazione si è passati alla creazione della neve lungo la superficie coperta dalle montagne.
Per far ciò la prima operazione è stata la duplicazione della mesh *landscape* nella stessa posizione, per poi spostarla leggermente più in basso lungo l'asse Z per far si che non si sovrapponga a quella già presente, aggiungendogli in ultima un *Subdivision Surface Modifier*. Si è passati quindi in *Sculpt Mode* selezionando la mesh duplicata e si è iniziato a scolpire con il pennello *SculptDraw* in modo additivo per far crescere la densità della mesh, oltre la mesh *landscape*, nelle zone in cui si voleva visualizzare ed ottenere l'effetto neve. In aggiunta con l'uso di altri due pennelli (*Pinch, Smooth*) si sono rispettivamente appuntite le creste della montagna e arrotondate le zone con maggior avvallamento di neve. Infine per creare in alcune zone l'effetto di posizionamento casuale della neve lungo la catena montuosa si è utilizzato il *Displace Modifier* dando in inpunt la texture *clouds* predefinita ed interna a Blender. Gli altri due oggetti creati sono stati la casa ed il piccolo laghetto, che non hanno richiesto grosse operazioni di modellazione. La casa è stata formata a partire da un cubo al quale si sono applicate alcune estrusioni e modifiche a singoli vertici per ottenere la forma desiderata, il laghetto è formato semplicemente da due piani posti uno sopra l'altro utilizzati rispettivamente per il fondale e per la superfice di acqua. In ultima si sono creati dei fiocchi di neve ed un *Gruppo* con lo stesso metodo per la creazione dei pini, poi si è aggiunto alla scena un piano parallelo ad essa con il *Particle System Modifier* di tipo *Emitter* attivo dandogli in input il *Gruppo*. Come ultima cosa sono stati impostati i valori in relazione alla gravità, alla generazione e alla caduta casuale dei fiocchi di neve.
Dopo aver creato e modellato la scena si è aggiunto uno sfondo di tipo *Sky texture* e si è utilizzato il *Node Editor* per aggiungere materiali e vari effetti agli oggetti presenti. In particolare si è fatto uso di tecniche di Bump Mapping, Noise Texture, Shader Mix e Moltiplicatori per ottenere una rappresentazione il più realistica possibile.

Conclusioni

Il progetto presentato in questa tesi soddisfa i requisiti posti inizialmente ed illustrati nel Capitolo 1, unica eccezione è la modalità di sculpting per la quale non si sono riusciti ad arginare i limiti esposti nella Sezione 4.4. Nonostante questo l'addon permette una buona un'interazione tra il software di modellazione Blender ed il Leap Motion come strumento di supporto. Il lavoro svolto nel progetto ha permesso di approfondire ed arricchire le conoscenze in merito alla Computer Grafica, grazie al software Blender e all'utilizzo delle modalità messe a disposizione, ed inoltre della programmazione con il linguaggio Python. Più precisamente si sono acquisite maggiori competenze nelle dinamiche che coinvolgono la fase di modellazione (Capitolo 4) e la fase di creazione di curve percorso NURBS per il moto e l'animazione di oggetti (Capitolo 5 e 6). La validazione dell'addon ottenuta secondo la semplicità di utilizzo ed il feedback ricevuto, si è dimostrata veloce, semplice ed intuitiva, mettendo però in risalto, in alcuni casi, una maggior precisione nell'eseguire le operazioni nativamente con il software Blender (Capitolo 7). A tal proposito le considerazioni che si avanzano, è che Leap Motion dovrebbe essere considerato uno strumento di supporto aggiuntivo per le funzioni già offerte da Blender, escludendo quindi la possibilità di sostituirlo a quelle native.
Pertanto l'addon realizzato può essere considerato come una solida base per l'estensione delle funzionalità di Blender, e può portare ad ulteriori sviluppi futuri per il raffinamento del sistema, focalizzandosi sempre sul concetto di Leap Aided Modelling.
Alcuni sviluppi futuri per il miglioramento e la crescita dell'addon sono

elencati di seguito:

- Ampliamento delle funzioni implementate, con l'obiettivo di migliorarne la precisione e la semplicità di utilizzo. Ad esempio:
  - Oltre alla selezione di vertici, implementare tecniche per la selezione di lati e facce.
  - Migliorare il disegno di curve percorso inserendo un controllo di calibrazione per eliminare gli spostamenti non desiderati durante il disegno.
  - Risolvere il problema legato alla modalità Sculpt Mode (Sezione 4.4), in modo da poter modellare una mesh, non solo con un dito ma con più dita della mano.
- In fase di animazione, dopo il disegno del percorso, dare la possibilità di ridefinire in precisi istanti sulla curva nuove rotazioni di orientamento per gli oggetti. Ad esempio con la mano aperta simulare il movimento di virata di un aereo.
- Aggiunta di funzioni per il trattamento di mesh in altre modalità di Blender differenti da Edit Mode ed Object Mode (Sezione 3.2). Ad esempio:
  - Permettere di colorare in maniera diretta le texture che definiscono il colore di una mesh in Texture Paint Mode.
  - Consentire all'utente di controllare e modificare un'armatura in modalità Pose Mode, attraverso una mappatura delle dita delle mani con oggetti *Bone* (prendendo spunto da quanto già effettuato nel software Leap Modal Controller).
- Leap Aided Modelling - Virtual reality: estensione dell'addon per la realtà virtuale. Utilizzando *Orion*, API di Leap Motion per VR, in

unione ad uno dei visori di realtà virtuale (Oculus Rift, HTC Vive o simili), si potrebbe creare un tavolo da lavoro virtuale per la modellazione 3D in Blender tramite Leap Motion.

Bibliografia

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

[2] Leap per artisti: http://drivenpixels.com/ethereal

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

[4] Leap per mutismo: http://www.motionsavvy.com/

[5] Leap in robotica: http://www.mirrortraininginc.com/

[6] Luca Rinaldi. *Leap Aided Modelling*. Università degli studi di Bologna, 2016.

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

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

[9] Playstation Move: https://www.playstation.com/en-us/explore/accessories/playstation-move/

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

[11] Blender API: https://www.blender.org/api/blender_python_api_2_78a_release/

[12] SWIG: https://www.swig.org/

[13] Modulo Scipy/Numpy: https://www.scipy.org/

[14] Modulo Interpolate Splev: https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.splev.html

[15] Blender Modal Operator: https://wiki.blender.org/index.php/Dev:Py/Scripts/Cookbook/Code_snippets/Interface

[16] Coordinate Mapping: https://developer.leapmotion.com/documentation/python/devguide/Leap_Coordinate_Mapping.html

[17] Leap Modal Controller: https://slsi.dfki.de/software-and-resources/hand-tracking-for-3d-editing/

[18] Blender BQ: https://github.com/BlenderBQ/BBQ/

[19] Leap Motion Animated Hands: http://www.magben.de/?h1=3d&h2=leap_motion_hand

[20] Leap Motion Sculpting: https://apps.leapmotion.com/apps/sculpting/windows

[21] Piegl Les e Tiller Wayne, *the NURBS Book*, Berlino, Spinger-Verlag,1995

[22] Rick Parent, *Computer Animation: Algorithms & Techniques*, Ohio State University, 2012

[23] Modelli 3D: http://www.tf3dm.com/

[24] Coordinate per Sculpting: https://blenderartists.org/forum/showthread.php?322779-How-do-I-pass-coordinates-to-the-sculpt-brush-operator

[25] Stroke Grease Pencil: https://bitbucket.org/ohsnapitsjoel/sculpt_stroke_on_grease_pencil

[26] StarkPy Sculpt: https://bitbucket.org/gxian/stark/src

Ringraziamenti

Desidero ricordare tutti coloro che mi hanno aiutato con la tesi attraverso suggerimenti, critiche ed osservazioni, a loro va la mia gratitudine.
Ringrazio anzitutto la professoressa Morigi Serena, Relatore, ed il Dottore Bertini Flavio, Correlatore: senza il loro supporto e la loro guida sapiente questa tesi non esisterebbe.
Un ringraziamento va ai colleghi di università, ai miei compagni di squadra e a tutte le persone che ho avuto modo di conoscere nel tempo. In particolar modo ringrazio gli amici che mi hanno incoraggiato, sopportato ed aiutato in questi anni spendendo parte del proprio tempo per e con me.
Vorrei infine ringraziare la mia famiglia ed in particolar modo Mamma e Papà, a cui dedico questo lavoro. Senza di voi nulla di tutto ciò sarebbe stato possibile.