POLITECNICO DI TORINO

Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Confronto tra Tecniche di Posa di Modelli 3D per la Creazione di Storyboard in Realtà Estesa

Relatori

Prof. Federico MANURI

Prof. Andrea SANNA

Candidato

Francesco ANZOINO

Anno Accademico 2024/2025

Abstract

Nell'ambito dell'animazione, che sia essa tradizionale 2D o digitale 3D, con il termine "Posing" ci si riferisce al task che consiste nella creazione e definizione delle pose per i diversi personaggi da animare. Definire le pose chiave di un personaggio è fondamentale per visualizzare la storia a partire dallo script e come i personaggi vi si inseriscono e interagiscono. Tale pratica diventa cruciale nella creazione dello storyboard, dell'animatic e nel successivo processo di animazione. Software per fare posing noti in commercio (oltre alla modellazione e animazione) sono Blender, Autodesk Maya o Presto i quali, tramite le funzioni base o add-on appositi, permettono di creare un'armatura (rig), associarla ad una mesh e, manipolando i vari controller dell'armatura, deformare la mesh per definirne la posa. Ispirandosi a tali sistemi di controllo, lo scopo di questo elaborato di tesi è investigare soluzioni alternative per il posing in realtà estesa, proponendo un sistema intuitivo che consenta di mettere in posa tramite cinematica diretta e inversa all'interno di un applicativo Unity, con l'obiettivo di creare storyboard. L'attenzione si concentra sul valutare quale dei vari approcci proposti risulti essere più intuitivo per l'utente. Stabilito un formato di riferimento per le armature dei modelli di partenza, il sistema è in grado di generare automaticamente le catene cinematiche necessarie per il controllo dell'armatura in cinematica diretta o inversa di modelli umanoidi e non, come animali o creature di fantasia. Il sistema realizzato consente all'utente di mettere in posa un personaggio utilizzando liberamente la cinematica diretta, inversa o una combinazione di entrambe. Permette di applicare trasformazioni come traslazione, rotazione e scalamento, oltre a salvare, azzerare e ripristinare le pose create in qualsiasi momento a piacimento. Il sistema è stato ideato e sviluppato come un plugin per Unity, integrabile in un qualsiasi progetto. Fornisce un meccanismo automatico per importare modelli nella scena da una cartella, creando e applicando loro le catene cinematiche. Un'interfaccia utente, progettata appositamente per la realtà estesa, permette di gestire le diverse funzionalità offerte dal plugin. Grazie a questo sistema, è stato possibile svolgere dei test per confrontare i due metodi di posa previsti e un terzo basato sul "Play & Stop" di animazioni predefinite. L'obiettivo è quello di valutare quale tra i tre risulti più efficace per gli utenti in applicazioni di realtà estesa.

Acknowledgements

Table of Contents

List of Figures VII

Acronyms IX

1 Introduzione 1
- 1.1 Contesto generale . . . 1
- 1.2 Personaggi Virtuali: Cosa Mettere in Posa . . . 2
- 1.3 Rigging e Cinematica: Controllare un Personaggio Virtuale . . . 2
- 1.4 Posing Task . . . 3
  - 1.4.1 Finalità . . . 3
- 1.5 Storyboard: Utilità e Tipologie . . . 4
- 1.6 Dominio Applicativo: Realtà Estesa . . . 4
- 1.7 Obiettivo e Sviluppo . . . 5
  - 1.7.1 Contributo Principale . . . 5
  - 1.7.2 Risultati Attesi . . . 5
  - 1.7.3 Struttura della Tesi . . . 6

2 Stato dell'Arte 7
- 2.1 Dalla Modellazione all'Animazione . . . 7
  - 2.1.1 Mesh: Una Superficie malleabile . . . 7
  - 2.1.2 Rig: Un'Armatura Invisibile . . . 8
  - 2.1.3 Step del processo di Rigging . . . 8
- 2.2 Cinematica Computazionale . . . 9
  - 2.2.1 Cinematica Diretta . . . 9
  - 2.2.2 Cinematica Inversa . . . 10
- 2.3 Realtà Virtuale: Immersione . . . 12
  - 2.3.1 Device Impiegati . . . 12
  - 2.3.2 Svantaggi della Realtà Virtuale . . . 13
- 2.4 Posing in Realtà Virtuale . . . 13
  - 2.4.1 Benefici della VR per il Posing . . . 13
  - 2.4.2 Esempi di applicazioni . . . 13

2.4.3 Applicazioni con supporto AI 15
2.5 Realtà Aumentata: Integrazione 15
2.5.1 Device Impiegati 16
2.5.2 Svantaggi della Realtà Aumentata 16
2.6 Posing in Realtà Aumentata 16
2.7 Realtà Mista: Estensione 17
2.7.1 Device impiegati 17
2.7.2 Svantaggi della Realtà Mista 18
2.8 Posing in Realtà Mista 18
2.8.1 Posing non in Realtà Estesa 19

3 Sistema di Posing Realizzato 20
3.1 Software Utilizzati 20
3.1.1 Blender 20
3.1.2 Unity 21
3.1.3 Visual Studio 22
3.2 Risorse Impiegate 23
3.2.1 Modelli 3D 23
3.2.2 Pacchetti Unity 23
3.2.3 Visori per la Realtà Estesa 24
3.3 Progettazione del Sistema di Posing 24
3.3.1 Condizioni iniziali 25
3.3.2 Funzionalità principali 25
3.3.3 Controllers e Scripts 26
3.3.4 Comunicazione tra sottosistemi 26
3.3.5 Modalità di interazione con il sistema 27
3.4 Pre-elaborazione in Blender 28
3.4.1 Condizioni al Rigging 28
3.4.2 Generazione XML del Modello 29
3.5 Sistema di Posing basato su Tecniche di Cinematica 30
3.5.1 Classi per le Catene Cinematiche 30
3.5.2 Generatore dei Modelli Manipolabili 31
3.5.3 Descrittore del Modello 3D 33
3.5.4 Controller Centrale del Sistema 35
3.6 Sottosistema delle Trasformazioni 42
3.6.1 Gestore della Posizione 43
3.6.2 Gestore della Rotazione 44
3.6.3 Gestore della Scala 45
3.7 Sistema di Posing basato su Animazioni Predefinite 46
3.7.1 Sistema di Controllo delle Animazioni 47

4 Valutazione Sperimentale e Risultati 49
4.1 Progettazione dei test . . . 49
4.1.1 Obiettivi . . . 50
4.1.2 Metriche scelte . . . 50
4.2 Conduzione del Test . . . 51
4.2.1 Campione scelto . . . 51
4.2.2 Tutorial . . . 51
4.2.3 Use Case . . . 51
4.2.4 Questionario finale . . . 52
4.2.5 Analisi dei risultati . . . 53
4.2.6 Ampliamento dei test . . . 54

5 Conclusione 56
5.1 Sviluppi Futuri . . . 57
5.1.1 Sviluppi legati alla gestione delle pose . . . 57
5.1.2 Sviluppi legati all'impiego dell'AI . . . 57
5.1.3 Sviluppi legati ai modelli . . . 59
5.1.4 Sviluppi legati alla tecnica di posing . . . 59
5.1.5 Sviluppi legati alle tecnologie . . . 60
5.1.6 Sviluppi legati alla collaborazione . . . 60

A Codice Sorgente 61
A.1 XML Generator . . . 61
A.2 KinematicChains . . . 63
A.3 FindBoneSequence . . . 67
A.4 SetUpCharacterSettings . . . 71
A.5 BoneHandleMapping . . . 73

Bibliography 75

List of Figures

3.1 Gerarchia delle ossa in Blender con nomenclatura 29
3.2 Handle per FK e IK in Unity 34
3.3 Pannello di controllo del Sistema Centrale con Microsoft MRTK . . 36
3.4 Pannello di controllo del Sistema Centrale con Meta SDK 37
3.5 Sezione del Menù per la Scelta della Cnematica 38
3.6 Posing tramite Cinematica Diretta 40
3.7 Posing tramite Cinematica Inversa 41
3.8 Sezione del Menù per il Controllo degli Handle 41
3.9 Sezione del Menù per la Gestione della Posa 42
3.10 Sezione del Menù delle Trasformazioni 43
3.11 Gizmo per le Traslazioni 44
3.12 Gimbal per le Rotazioni 45
3.13 Slider per le Dimensioni 46
3.14 Personaggio Animato che Cammina 48
3.15 Pannello di Controllo del Sistema di Posing basato su Animazioni . 48

Acronyms

IK
: Inverse Kinematic

FK
: Forward Kinematic

VR
: Virtual Reality

AR
: Augmented Reality

MR
: Mixed Reality

XR
: Extended Reality

MVC
: Model - View - Controller

UI
: User Interface

AI
: Artificial Intelligence

Chapter 1

Introduzione

Il capitolo introduttivo di questa tesi ha lo scopo di fornire una panoramica generale sul contesto di riferimento e sugli elementi tecnici analizzati, evidenziandone caratteristiche e rilevanza nel campo della realtà estesa. Vengono inoltre presentati l'obiettivo principale dello studio, le metodologie adottate per lo sviluppo del sistema proposto, i risultati attesi e la struttura della tesi, con una sintesi dei contenuti trattati nei capitoli successivi.

1.1 Contesto generale

L'animazione 3D e la produzione cinematografica animata rappresentano un settore complesso e articolato, caratterizzato da una pipeline produttiva ben definita e ricca di tecnicismi. Ogni fase del processo, dalla progettazione alla realizzazione finale, è fondamentale per garantire la creazione di personaggi virtuali animati e ottenere un prodotto in linea con le aspettative artistiche e tecniche. Questa pipeline trova applicazione in diversi ambiti, tra cui cinema, gaming e realtà virtuale, condividendo elementi e metodologie comuni. Uno degli aspetti essenziali di questa pipeline è il posing, ovvero il processo di creazione e definizione delle pose chiave dei personaggi. Questa pratica è fondamentale non solo per la narrazione visiva, ma anche per la successiva animazione, comprendendo fasi cruciali come la realizzazione dello storyboard e dell'animatic. Strumenti software come Blender [1], Autodesk Maya [2] e Presto [3] offrono funzionalità avanzate per il posing, consentendo la creazione di armature (rig), la loro associazione a una mesh e la manipolazione attraverso controller per ottenere pose precise. L'evoluzione di queste tecnologie e il loro impiego in ambienti di realtà estesa rappresentano una nuova frontiera, che apre la strada a modalità di interazione più intuitive e immersive nel processo di posing e animazione. Pertanto, forniamo una definizione generale delle principali componenti con le quali si interagirà nel corso di questo elaborato di tesi.

1.2 Personaggi Virtuali: Cosa Mettere in Posa

Un personaggio virtuale non è altro che una rappresentazione digitale di un determinato soggetto, che sia esso umano, animale o di fantasia. Generalizzando, possiamo applicare tale definizione anche alla trasposizione di un oggetto reale in digitale, definendolo esclusivamente come un oggetto virtuale; nel nostro caso, trattando delle pose, ci riferiremo principalmente a personaggi virtuali. L'impiego di una rappresentazione digitale di un soggetto reale può avere molteplici scopi: dalla semplice integrazione in un ambiente virtuale tridimensionale, all'utilizzo in ambiti disparati quali film d'animazione, videogiochi, simulazioni (e.g. in medicina o in ambito ingegneristico) o in applicativi in realtà estesa. La vera e propria rappresentazione di tale personaggio è incarnata da un modello 3D, il quale è composto a sua volta da diversi elementi. Per il nostro caso specifico definiamo:

- **Mesh:** Una superficie modellabile e deformabile che rappresenta la forma visibile del personaggio.
- **Rig:** Un'armatura invisibile associata al modello 3D per permettere il movimento dello stesso e la deformazione della mesh.

Tali elementi sono fondamentali per la definizione di un modello 3D da mettere in posa e animare, in quanto senza la Mesh non esisterebbe una superficie visibile del personaggio; senza il Rig non potremmo muovere naturalmente la mesh per creare l'animazione.

1.3 Rigging e Cinematica: Controllare un Personaggio Virtuale

Definiti chi saranno i soggetti su cui agire, è necessario fare una panoramica delle modalità tramite il quale mettere in posa dei personaggi virtuali. Il processo che consiste nella creazione di un'apposita struttura scheletrica (rig) con vincoli precisi e associazione ad una mesh da deformare, prende il nome di **Rigging**. Le modalità di gestione e controllo dei movimenti di un rig, dopo l'associazione alla mesh, prendono il nome di **Cinematica**; in particolare, esistono due principali tecniche cinematica per la messa in posa di un personaggio virtuale, ognuna con vantaggi e svantaggi distinti:

- **Cinematica Diretta (FK)**: Si basa sul controllo gerarchico delle ossa, dove il movimento di un osso a monte influenza i movimenti delle ossa successive nella catena. È utile per movimenti prevedibili ma richiede maggiore lavoro per movimenti complessi.

- **Cinematica Inversa (IK)**: Permette di controllare una catena di ossa manipolandone direttamente l'estremità; le posizioni delle ossa intermedie vengono calcolate automaticamente. Tale soluzione facilita la realizzazione di movimenti naturali e fluidi.
- **Combinazione di FK e IK**: Uso contemporaneo delle due tipologie di cinematica, a seconda delle esigenze e delle preferenze dell'animatore.

1.4 Posing Task

A partire da dei modelli 3D così definiti e liberamente controllabili, è possibile infine realizzare il processo che porta alla definizione di una posa. Con il termine **Posing**, infatti, ci si riferisce al processo di manipolazione e deformazione di un personaggio tramite manipolazione delle articolazioni del suo scheletro, al fine da definire pose significative, espressive e caratterizzanti. Fare posing è utile, se non fondamentale, in diversi contesti quali cinema e animazione, videogiochi, illustrazioni, concept art e, nel nostro caso specifico, per fare storyboarding 3D.

1.4.1 Finalità

Fare posing in maniera completa e corretta ha diverse finalità e utilità su tutta la pipeline di lavoro. Di seguito vengono elencate le finalità e i risvolti per cui effettuare posing correttamente è utile:

- **Caratterizzazione dei personaggi e espressività:** Attraverso la definizione di certe pose chiave, queste possono trasmettere le informazioni sullo stato d'animo, le intenzioni e la personalità del personaggio, caratterizzandolo e delineandone un profilo.
- **Interazione realistica dei personaggi con l'ambiente:** Definire determinate pose è utile per descrivere come un certo personaggio interagisce, si integra e si rapporta all'ambiente che lo circonda, il tutto in maniera credibile.
- **Supporto alle animazioni e allo storyboard:** Una delle finalità essenziali del posing è proprio quella di essere un passaggio fondamentale della creazione delle animazioni e della stesura dello storyboard. L'insieme delle pose fondamentali sono essenziali quando si definiscono i momenti più significativi in un'animazione, includendo posa iniziale, finale e di climax. Ancora più importante per il nostro caso, fare posing permette di definire le pose utili ai fini di creazione di storybaord quando si effettua storyboarding 3D.

In merito a quest'ultima applicazione, si può evidenziare cosa sia uno storyboard e l'utilità dello storyboarding in una produzione animata.

1.5 Storyboard: Utilità e Tipologie

Con il termine **Storyboard** si indica una rappresentazione grafica della storia che si vuole narrare, costituito e suddiviso in scene chiave che permettono di descrivere lo sviluppo narrativo della storia. Da ciò, fare **Storyboarding** vuol dire proprio creare una sequenza di immagini (che siano esse schizzi disegnati a mano, frame catturati digitalmente o illustrazioni) con, per ognuna di esse, una serie di informazioni e annotazioni che descrivono dialoghi, movimenti di camera, tipologia di obiettivi da utilizzare e altri aspetti tecnici per la scena da riprendere [4]. Avere a disposizione, prima delle riprese, di uno storyboard completo e ben sviluppato e fondamentale per diverse ragioni, tra cui:

- **Divisione della storia:** Lo storyboard aiuta a creare una struttura della storia con passaggi logici e comprensibili.
- **Visualizzazione anticipata:** Lo storyboard è parte del previs, per cui tramite la pre visualizzazione di cosa riprendere con la camera, permette di valutare in anticipo inquadrature, movimenti e interazioni tra personaggi.
- **Ottimizzazione delle riprese:** Tramite lo storyboard è fin da subito chiaro la serie di animazioni da realizzare, evitando quindi animazioni superflue o spreco di tempo e risorse.
- **Allineamento tra i tecnici di settore:** Lo storyboard è un punto di riferimento a cui i membri dello staff tecnico si riferiscono per seguire il flusso di riprese e essere allineati.

Lo storyboard può essere realizzato in 2D, sia manualmente che in digitale, oppure in 3D all'interno di un ambiente in realtà estesa, utilizzando strumenti specifici.

1.6 Dominio Applicativo: Realtà Estesa

Il dominio applicativo in cui si pone questo elaborato di tesi è proprio di introdurre un sistema di posing all'interno di un applicazione in realtà estesa; si fornisce quindi una definizione partendo dal definire le tecnologie che la compongono e in cui si diversifica. In particolare:

- **Realtà Virtuale (VR)**: Immersione in un ambiente completamente digitale, interagendo tramite dispositivi come visori aappositi.
- **Realtà Aumentata (AR)**: Sovrapposizione di elementi digitali al mondo fisico, visibili tramite dispositivi come smartphone o occhiali intelligenti.

- **Realtà Mista (MR)**: Integrazione tra elementi digitali e reali, permettendo l'interazione in tempo reale tra entrambi.

L'insieme di queste tipologie di tecnologie, atte a combinare il mondo fisico e quello digitale, ricadono sotto un'unica categoria di impiego, definita come **Realtà Estesa (XR)**.

1.7 Obiettivo e Sviluppo

Dopo aver presentato una panoramica generale di tutti gli elementi con cui agire, si espone il task principale che questa tesi si propone di assolvere. L'elaborato di tesi ha come obiettivo quello di esplorare una soluzione alternativa a quella attualmente presente in un applicativo per fare storyboarding in realtà estesa. Lo scopo principale è semplificare e migliorare il processo di posing di personaggi virtuali, con il fine di dare piena libertà ad un animatore di controllare e definire le pose che i personaggi devono assumere.

1.7.1 Contributo Principale

Il contributo che questo elaborato di tesi si auspica è di fornire un sistema integrato intuitivo e semplice da usare, che permetta una completa libertà sulla manipolazione flessibile dei modelli da mettere in posa. Il sistema realizzato permetterebbe quindi di effettuare posing mediante cinematica diretta, inversa o una combinazione delle due (a seconda delle preferenze dell'animatore e della complessità della posa). Sarebbe possibile inoltre interagire con il personaggio virtuale applicando le trasformazioni base che un qualsiasi software di modellazione mette a disposizione, quali traslazione, rotazione e scalamento. Infine, con un sistema locale di memorizzazione, è possibile salvare una posa definita localmente, resettare la posa alle condizioni iniziali e poter salvare una lista di pose preimpostate da poter utilizzare all'occorrenza e al bisogno.

1.7.2 Risultati Attesi

Allo stato attuale l'applicazione prevede un sistema di messa in posa basato sullo "Start & Stop" di un'animazione predefinita su un personaggio virtuale. Con l'introduzione di un sistema di controllo descritto nella sezione 1.7.1, ci si aspetta un netto miglioramento nello svolgimento delle operazioni da effettuare, avendo pieno controllo sulla creazione della posa desiderata mediante IK o FK, velocizzando l'intero processo (anche grazie all'archivio di pose predefinite sempre accessibile) e definendo con maggior precisione i dettagli delle pose più complesse.

1.7.3 Struttura della Tesi

La tesi si struttura in 5 capitoli, i quali tratteranno i seguenti argomenti:

- **Capitolo 1:** Introduzione al contesto di lavoro e degli strumenti trattati, fornendone definizione e una panoramica generale.
- **Capitolo 2:** Presentazione dello stato dell'arte in cui vengono scandagliate le soluzioni attualmente presenti in letteratura e utili ai fini dell'elaborato di tesi, in relazione alle alle tecniche di cinematica impiegate, dominio applicativo (XR) e tecnologie sfruttate.
- **Capitolo 3:** Esposizione tecnica della soluzione proposta, mostrando nel dettaglio le diverse componenti del sistema realizzato.
- **Capitolo 4:** Raccolta e discussione dei risultati ottenuti tramite le sperimentazioni, verificando quanto ottenuto dalla sperimentazione rispetto alle metriche scelte.
- **Capitolo 5:** Conclusioni del lavoro di tesi e possibili migliorie e innovazioni applicabili al sistema fino ad ora definito.

Chapter 2

Stato dell'Arte

Il secondo capitolo si propone di approfondire gli aspetti tecnici necessari per affrontare il task di posing e animazione dei personaggi virtuali, con particolare attenzione agli ambienti di realtà estesa dove fare posing. Verranno ripresi concetti legati alla modellazione 3D, al processo di rigging e alle principali tecniche cinematiche utilizzate in animazione; nello specifico la cinematica diretta (FK) e la cinematica inversa (IK). In particolare, si descriveranno nel dettaglio concetti chiave, come la mesh, i rigs e le tecniche di rigging. Il capitolo esplorerà inoltre le soluzioni esistenti in letteratura per la gestione della cinematica nei personaggi virtuali, analizzando le metodologie applicate in contesti come l'animazione, i videogiochi e le simulazioni interattive. In aggiunta, si farà un focus sulle tecnologie immersive legate alla realtà estesa (XR), evidenziando le principali differenze, i vantaggi e gli svantaggi di VR, AR e MR. Saranno coinvolte le tecnologie più diffuse per l'interazione con tali ambienti immersivi e come queste possano integrarsi con le tecniche di rigging e cinematica per migliorare l'esperienza interattiva e la realizzazione di animazioni in contesti virtuali. Il principale obiettivo del capitolo, ripercorrendo e approfondendo gli argomenti citati in precedenza, è quello di fornire una panoramica delle soluzioni e delle applicazioni presenti in letteratura per ciascuna tecnica di posing, con riferimenti alle specifiche ambientazioni immersive e alle diverse tecnologie impiegate.

2.1 Dalla Modellazione all'Animazione

2.1.1 Mesh: Una Superficie malleabile

La **Mesh** è la superficie visibile di un modello 3D definita una rete di poligoni (principalmente triangoli o quadrilateri) che ne caratterizza la geometria. Tali poligoni devono coprire l'intera superficie dell'oggetto e sono caratterizzati da

vertici, spigoli e facce. Ulteriori aspetti importanti sono la **Topologia**, ovvero la distribuzione dei poligoni nella mesh (influente nella deformazione della stessa quando viene applicato un rig) e le **Texture** e **Materiali**. Questi ultimi sono, rispettivamente, delle immagini 2D applicate sulla mesh per creare effetti visivi (e.g. colori, riflessi, ombre) e tutte le informazioni relative a come la mesh interagisce con la luce (realizzando un modello, ad esempio, lucido o opaco a seconda della preferenza). Infine, una suddivisione più fine dei poligoni (ovvero un numero maggiore di poligoni piccoli rispetto a pochi e grandi) aumenta il livello di dettaglio della mesh e la sua "morbidezza".

2.1.2 Rig: Un'Armatura Invisibile

Il **Rig** è una struttura più o meno complessa che agisce come un'armatura interna alla mesh e invisibile, con lo scopo di deformarla secondo determinati criteri. Le componenti principali dell'armatura sono le **Bones** (Ossa) e le **Joints** (Articolazioni) le quali ricalcano la funzione dello scheletro negli esseri viventi e ne simulano il comportamento. Nello specifico, le ossa sono associate a delle parti specifiche della mesh per poi, ruotandole o spostandole, alterarne la posizione. Le articolazioni, invece, connettono le ossa tra loro e definiscono come le varie sezioni della mesh si muovono l'una rispetto all'altra

2.1.3 Step del processo di Rigging

Elementi chiave nel processo di rigging, oltre alla creazione di un sistema di ossa e articolazioni tarato sulla mesh che gli si vuole associare, sono anche altri componenti da dover tenere in conto per poter controllare al meglio il tutto. In dettaglio, il processo nel quale avviene l'associazione di una porzione di mesh ad un osso viene detto **Skinning** e nel fare ciò è necessario definire con quale misura un determinato osso influenza i vertici ad esso associati. Per cui un osso può con più o meno influenza e peso controllare i vertici assegnatigli, e la definizione del grado di influenza con cui l'osso agisce viene detto **Weight Painting**. Ne consegue che, più un vertice è influenzato, maggiore sarà il peso dell'osso su di esso. Inoltre, al rig sono aggiunti degli elementi detti **Controller** per semplificare il processo di messa in posa e animazione. L'uso dei controller evita agli animatori di dover manipolare direttamente ossa e articolazioni, semplificando così l'animazione e il controllo di movimenti complessi. Infine, è possibile applicare una serie di regole aggiuntive dette **Contraints** (Vincoli) per descrivere come le ossa interagiscono, ad esempio quali sono i limiti sulle rotazioni o come le ossa tra loro si influenzano, per cui al muoversi di un determinato osso, le altre ad esso vincolate seguono il movimento

2.2 Cinematica Computazionale

Con il termine **Cinematica Computazionale** ci si riferisce ad una branca della meccanica applicata dedita allo studio e simulazione dei movimenti dei sistemi meccanici utilizzando metodi computazionali e algoritmi matematici per definire posizione, orientamento, velocità e accelerazione nel tempo [5]. L'obiettivo principale è risolvere problemi di cinematica applicati a sistemi complessi, comprendenti diversi gradi di libertà e vincoli applicati alle componenti del sistema. Tale disciplina trova applicazione in vari ambiti, dalla robotica alla simulazione fisica, fino alla progettazione di sistemi meccanici. L'ambito di interesse specifico in questo elaborato di tesi è l'animazione 3D, per cui vengono esplorate le principali metodologie per descrivere, calcolare e definire il movimento di sistemi articolati associati a modelli virtuali. In particolare, vengono approfondite le due metodologie principali: la Cinematica Diretta e la Cinematica Inversa.

2.2.1 Cinematica Diretta

La **Forward Kinematic (FK)**, o Cinematica Diretta, è una tecnica di animazione che permette di controllare una catena ossea dell'armatura agendo direttamente su ogni osso, modificandone manualmente la rotazione o la posizione in modo gerarchico; il movimento applicato a un osso padre si propaga automaticamente alle ossa figlie della catena [6]. Ad esempio, per animare il movimento di un braccio, modificando la rotazione dell'articolazione della spalla, anche le ossa dell'avambraccio, del polso e della mano si sposteranno di conseguenza, rispettando la gerarchia ossea. Questo approccio garantisce un controllo dettagliato sulla posa e sul movimento, permettendo di ottenere movimenti naturali e coerenti; la gestione di movimenti articolati può risultare difficoltosa. Dal punto di vista computazionale, il calcolo della Cinematica Diretta si basa su trasformazioni matriciali che descrivono la posizione e l'orientamento di ogni osso rispetto al suo predecessore. Le principali tecniche di calcolo utilizzate per la gestione della FK sono le seguenti:

- **Matrici di Trasformazione e Denavit-Hartenberg:** La Cinematica Diretta sfrutta le matrici di trasformazione (solitamente matrici di rotazione e traslazione) per determinare la posizione assoluta di ogni osso della catena. Il metodo Denavit-Hartenberg descrive la relazione tra le giunture di un sistema articolato, permettendo una rappresentazione efficiente della cinematica di modelli articolati. Queste matrici di trasformazione permettono di calcolare la posizione di ogni segmento della catena ossea in base ai parametri angolari delle articolazioni, con la propagazione del movimento dalla radice della catena fino alla sua estremità [7].

- **Impiego dei Quaternioni per la Rotazione:** I Quaternioni sono una rappresentazione alternativa alle Matrici di Rotazione e alle Coordinate di Eulero. I Quaternioni sono privi del problema del **Gimbal Lock**[1], che può verificarsi nelle matrici di rotazione tradizionali, e offrono un vantaggio significativo quando si devono interpolare rotazioni in animazione 3D. I Quaternioni sono utilizzati per descrivere rotazioni in uno spazio tridimensionale e spesso sono preferiti real-time, poiché più efficienti a livello computazionale rispetto alle matrici di rotazione e permettono una gestione stabile delle rotazioni [8].

In termini di propagazione dell'informazione, è possibile tracciare un'analogia tra la Cinematica Diretta e il concetto di **Forward Propagation** nel Machine Learning [9]. In entrambi i casi, l'informazione viene propagata in avanti lungo una struttura gerarchica; nella Cinematica Diretta, infatti, il movimento viene trasmesso a partire dalla radice della catena ossee fino all'estremità. In questo caso, la posizione finale dell'ultimo osso è il risultato delle trasformazioni cumulative applicate lungo la catena ossea. La Cinematica Diretta è ideale per controllare movimenti prevedibili e ben definiti, consentendo di realizzare animazioni sotto il controllo totale dell'animatore. Tuttavia, la necessità di gestire manualmente ogni singolo osso può rendere il processo più laborioso, specialmente per movimenti complessi.

2.2.2 Cinematica Inversa

L'**Inverse Kinematic (IK)**, o Cinematica Inversa, è una tecnica di animazione che permette di controllare una catena ossea dell'armatura agendo solo sugli estremi, mentre il sistema calcola automaticamente posizione e rotazione delle ossa intermedie per ottenere il movimento desiderato [10]. Tale tipologia di cinematica consente all'animatore di muovere esclusivamente l'estremo della catena desiderata e lasciare che le posizioni delle ossa di tutta la catena vengano definite in relazione ad essa mediante un algoritmo, permettendo all'intera struttura ossea di adattarsi dinamicamente per mantenere la coerenza del movimento. Ad esempio, per animare il movimento di una gamba, basterebbe spostare il piede nella posizione desiderata e il sistema calcolerebbe automaticamente le rotazioni delle ossa della caviglia, polpaccio e coscia. Le posizioni così calcolate sono tuttavia influenzate sia dalla gerarchia ossea vigente, sia dai vincoli imposti sulle rotazioni. Dal punto di vista computazionale, il calcolo della Cinematica Inversa sfrutta svariati algoritmi risolutivi; diversi sono i metodi numerici adottati, di seguito elencati come riportati

[1]Perdita di un grado di libertà nella rotazione e creando una situazione in cui non è possibile ruotare liberamente in tutte le direzioni quando assi di rotazione si allineano

in [11]. Le principali tecniche di calcolo utilizzate per la gestione dell'IK sono le seguenti:

- **Jacobiano e Pseudoinversa:** Questo metodo sfrutta la matrice Jacobiana, che definisce la relazione tra le variazioni delle giunture e il movimento dell'end-effector (l'estremità della catena cinematica). La cinematica inversa viene risolta attraverso la pseudoinversa del Jacobiano, una tecnica che consente di ottenere una soluzione approssimata ottimale, molto utile in sistemi con vincoli complessi. Nonostante garantisce un'elevata precisione, questo metodo può risultare oneroso a livello di calcoli, specialmente in sistemi con un alto numero di gradi di libertà [12].
- **Cyclic Coordinate Descent (CCD):** Questo metodo iterativo regola progressivamente ogni giuntura della catena, iniziando dall'estremità e procedendo verso la base. Ad ogni iterazione, ciascun osso viene ruotato in modo da ridurre la distanza tra l'end-effector (l'estremità della catena cinematica) e la posizione target. Questo approccio è semplice ed efficiente dal punto di vista computazionale, ma può generare movimenti meno naturali rispetto ad altre tecniche [13].
- **Forward And Backward Reaching Inverse Kinematics (FABRIK):** Questo metodo iterativo utilizza una propagazione bidirezionale lungo la catena ossea per risolvere il problema della cinematica inversa. Si distinguono in due fasi: nella fase di **forward reaching**, l'end-effector (l'estremità della catena cinematica) viene spostato verso la posizione target, mentre nella fase di **backward reaching**, la base della catena viene riportata alla sua posizione originale, aggiornando progressivamente le giunture intermedie. Questo approccio consente di ottenere movimenti più fluidi e realistici rispetto ad altre tecniche [14].

In termini di propagazione dell'informazione, è possibile tracciare un'analogia tra la Cinematica Inversa e il concetto di **Backward Propagation** nel Machine Learning [15]. In entrambi i casi, l'informazione viene trasmessa all'indietro lungo una struttura gerarchica; nella Cinematica Inversa, infatti, il processo parte dall'end-effector, la cui posizione è nota, e risale la catena ossea per determinare la configurazione delle giunture necessaria a raggiungere tale posizione. La Cinematica Inversa è particolarmente utile per simulare movimenti realistici e reattivi, rendendo il processo di animazione più rapido ed efficiente. Tuttavia, presenta alcune limitazioni: la perdita di controllo sulle ossa intermedie può generare movimenti innaturali o instabili, specialmente in modelli con vincoli articolari rigidi. Inoltre, la complessità computazionale cresce con l'aumento del numero di articolazioni, rendendo necessario l'uso di tecniche di ottimizzazione per garantire prestazioni in tempo reale.

2.3 Realtà Virtuale: Immersione

La **Realtà Virtuale** (**VR**) è una tecnologia che permette l'interazione con un ambiente completamente digitalizzato e generato tramite software, in cui l'utente è completamente immerso tramite l'utilizzo di appositi sistemi. L'interazione con l'ambiente virtuale avviene tramite dispositivi aggiuntivi come controller, guanti, sistemi di tracciamento corporeo o sensori di movimento. In questo modo, la VR permette di simulare esperienze sensoriali e interattive, dove il mondo reale viene sostituito da un ambiente digitale che può replicare situazioni reali o, in alcuni casi, creare scenari impossibili da realizzare nel mondo fisico. Tra le applicazioni più diffuse troviamo giochi immersivi, simulazioni per l'addestramento professionale, esperienze educative e utilizzi in ambito medico e psicologico.

2.3.1 Device Impiegati

Interagire con un ambiente virtuale simulato richiede l'uso di hardware avanzato per l'immersione e, molto spesso, non sempre accessibile a causa dei costi elevati. I dispositivi principali sono i **Visori VR** per l'accesso alla vista del mondo virtuale e appositi **Controller** per l'interazione con lo stesso. Nello specifico:

- **Visore VR:** Il Visore VR è il dispositivo che permette all'utente di entrare nell'ambiente virtuale. Il visore è dotato di schermo ad alta risoluzione per ogni occhio ed è montato su un apposito headset provvisto di cuffie per l'audio ambientale, sensori di tracciamento dei movimenti della testa e sistema di regolazione per la vestibilità. Tra i visori in commercio ricordiamo esempi come le serie HTC Vive, Meta Quest, l'Apple Vision Pro, Samsung Gear, Playstation VR e Razer OSVR.

- **Controller VR:** I Controller VR sono dispositivi che consentono l'interazione tra utente e ambiente virtuale, permettendo di manipolare oggetti, selezionare menu e compiere azioni personalizzate in base all'applicazione. I modelli più moderni sono dotati di **tracking posizionale** che utilizza sensori ottici a infrarossi per monitorare con precisione i movimenti delle mani nello spazio, migliorando così il realismo e l'immersività dell'esperienza. Oltre ai classici controller a mano (dispositivi simili a joystick che tracciano posizione e rotazione), esistono altre tipologie di controller avanzati come **Guanti VR** per il tracking dettagliato delle dita con feedback aptico e **Controller di movimento** come bande indossabili che rilevano la posizione dell'intero corpo o di alcune sue parti specifiche.

2.3.2 Svantaggi della Realtà Virtuale

Ci sono diversi fattori negativi che affliggono la VR. Tra questi, la necessità di l'hardware specifico e relativamente costoso, che può limitarne l'accessibilità. Inoltre, le limitazioni fisiche dell'ambiente circostante e i possibili disagi come l'affaticamento visivo, la nausea o il motion sickness [2] sono problematiche tutt'ora presenti.

2.4 Posing in Realtà Virtuale

L'impiego della realtà virtuale per ottimizzare le pipeline lavorative nel campo dell'animazione è un'idea già esplorata in letteratura. Esistono diversi impieghi della VR per realizzare ambienti collaborativi, dove artisti e animatori possono immergersi e svolgere le proprie attività, sfruttando i mezzi messi a disposizione dai sistemi VR. Tali ambienti offrono notevoli benefici, contribuendo al miglioramento delle varie pipeline lavorative.

2.4.1 Benefici della VR per il Posing

I risultati favorevoli all'adozione di strumenti in VR sono evidenziati nello studio [16], che analizza come rendere più "smart" gli strumenti preesistenti, utilizzando tecniche di ottimizzazione per sfruttare le capacità della VR e migliorare la creazione di contenuti 3D. Lo studio si divide in due sezioni principali: la prima riguarda la valutazione dei benefici derivanti dall'applicazione delle tecniche di VR a animazioni complesse, mentre la seconda si concentra sul mostrare come lavorare in VR possa portare vantaggi tramite strumenti di interazione "smart", rispetto a ambienti tradizionali come Unity. Lo studio dimostra che, in generale, l'impiego della VR per i due esperimenti ha prodotto risultati più accurati e produttivi. Ottimi risultati sono stati riscontrati quindi per animazioni complesse e interazioni tramite queste tipologie di interfacce avanzate; controllare l'accuratezza nelle tecniche di animazione in VR è quindi fondamentale.

2.4.2 Esempi di applicazioni

Definiti i benefici derivanti dall'impiego della VR, sono presenti esempi chiari in letteratura di software progettati specificamente per supportare gli animatori, fornendo una serie di strumenti per eseguire diverse attività; un esempio è riportato in [17]. Viene proposto un tool che consente, grazie a un design accessibile e a

[2] Disagio fisico causato dalla discordanza sensoriale tra ciò che si vede e ciò che il corpo percepisce.

strumenti appositi, la creazione di storyboard e previs con tanto di gestione di tutti i tecnicismi legati ad essi, il tutto in maniera collaborativa tra la troupe e i 3D artists. La soluzione del paper prevede l'adozione di un ambiente in VR, dove ciascun partecipante può maneggiare gli elementi con appositi controller. Per offrire un sistema utile e funzionale per gli scopi descritti, sono state prese delle precise scelte di design. Pertanto, il sistema deve essere: intuitivo, rapido e garantire continuità tra le varie fasi di lavoro. Le modalità previste sono:

- Storyboard Mode: consente di creare una scena statica, scattare fotogrammi statici e annotare appunti relativi ai fotogrammi ottenuti, definendo i movimenti di camera;
- Previsualisation Mode: permette di animare le scene statiche definite precedentemente, creando e modificando i movimenti della camera lungo binari virtuali, tracciati tramite i controller VR.
- Technical Mode: consente di controllare gli aspetti tecnici, come i dispositivi di ripresa (camera rigs) e l'illuminazione (light rigs), per ottimizzare la scena e testare l'effetto visivo.

Prendendo nello specifico la modalità Storyboard, il paper propone di animare mediante FK scheletri con vincoli. In particolare, tramite i controller a 6DoF, è possibile selezionare liberamente le ossa dello scheletro dei personaggi di interesse e manipolarli a piacimento, rispettando i vincoli imposti dalla catena cinematica della sezione che si sta manipolando. Tramite dei classici controller sono possibili le classiche operazioni di selezione, manipolazione degli oggetti, navigazione nella scena. I risultati ottenuti nelle sperimentazioni hanno dimostrato i benefici che un tool di questo tipo può fornire, sia per i principianti che per gli utenti esperti. Gli utenti che hanno testato il tool sono riusciti ad ottenere performance soddisfacenti con pochi minuti di training sull'utilizzo del sistema. Per cui, gli studi presenti in letteratura suggeriscono che fare posing in un ambiente VR sia sicuramente funzionale per gli utenti, sia esperti che principianti, soprattutto se si adottano tecniche di controllo già familiari negli applicativi tradizionali. Ricreare un sistema di controllo tramite IK e FK risulta, dunque, vincente.

Un altro esempio in letteratura combina un ambiente VR con la possibilità di ottenere la posa da applicare a un modello tramite il tracciamento di uno sketch disegnato. Questo è il caso presentato in [18], dove l'idea principale consiste nel trasferire le metodologie tradizionali di posing dal 2D al 3D, consentendo agli animatori di disegnare sketch direttamente in un ambiente virtuale e lasciare che questi stabiliscano la posa da applicare al modello 3D. I requisiti per adottare questo approccio includono la necessità di disporre di un modello già riggato e skinnato, eliminando così la necessità per l'animatore di prepararlo manualmente. Una volta definiti gli sketch disegnati nel'ambiente virtuale, il sistema allinea

automaticamente il modello ad essi, minimizzando le distanze tra i punti chiave; add-on e SDK apposite forniscono strumenti per disegnare liberamente in un ambiente virtuale. Per il mapping tra il personaggio virtuale e lo sketch fornito in input, il problema viene formalizzato come un'ottimizzazione matematica, garantendo un posizionamento accurato del modello in base alle linee guida disegnate dall'utente.

2.4.3 Applicazioni con supporto AI

Uno dei possibili approcci al posing contempla il Deep Learning come metodo risolutivo, consentendo di estrarre automaticamente la posa di un modello 3D a partire da diverse fonti, come video online. Un esempio di questo approccio è illustrato in [19], dove si analizza il problema della creazione di animazioni 3D e l'uso della VR. L'applicativo descritto nel paper prende il nome di "*VideoPoseVR*" e tale soluzione sfrutta il Deep Learning per la ricostruzione del movimento in 3D a partire da video online, consentendo di etichettare i movimenti (captioning) e salvarli in un dataset. Gli utenti che si interfacciano con il sistema possono quindi importare dei video, cercare i movimenti nel dataset, modificarne la Motion Timeline [3] e combinare diverse sequenze di moti da più video per creare nuove animazioni in VR. Nel complesso, *VideoPoseVR* è stato valutato positivamente sia per le sue funzionalità che per l'esperienza utente. I partecipanti ai test lo hanno trovato intuitivo e facile da apprendere, apprezzando la possibilità di utilizzare video online per animare in VR. Tuttavia, sono state riscontrate alcune problematiche nella fase di estrazione del movimento dai video, in particolare quando: le persone nei video sono troppo piccole, rendendo difficile una corretta analisi del movimento; il movimento è parzialmente occluso o non perfettamente visibile, compromettendo la precisione della ricostruzione. Nonostante queste limitazioni, il sistema dimostra un grande potenziale nel semplificare e velocizzare il processo di animazione in VR.

2.5 Realtà Aumentata: Integrazione

La **Realtà Aumentata** (**AR**) è una tecnologia che permette di sovrapporre elementi creati digitalmente interattivi al mondo reale, dando la possibilità all'utente di mantenere contezza dell'ambiente fisico in cui si trova. Gli elementi con i quali interagisce possono includere ad esempio modelli 3D, interfacce grafiche interattive, informazioni testuali e suoni. La AR mantiene un legame diretto con il mondo fisico, estendendo le possibilità di interazione ma senza sostituirlo e lasciando che l'utente non si isoli. Ciò la rende particolarmente adatta per applicazioni pratiche, come la formazione professionale, la simulazione industriale e il supporto medico.

[3] Rappresentazione temporale per la gestione delle animazioni 3D nel tempo.

2.5.1 Device Impiegati

Per sperimentare la Realtà Aumentata è possibile utilizzare diverse tipologie di dispositivi, più o meno costosi e con diverse prestazioni e funzionalità. Vengono riportati alcuni esempi:

- **Smartphone e Tablet:** La soluzione più accessibile e diffusa per AR. Attraverso la fotocamera, lo schermo e i sensori di movimento integrati, questi dispositivi sovrappongono immagini digitali al mondo reale.
- **Smart Glasses (Occhiali AR):** Sono dispositivi più avanzati e costosi. Ricordano degli occhiali indossabili ma sono dotati di **lenti trasparenti** o **Display Heads-Up (HUD)** che proiettano gli elementi virtuali nel campo visivo. Essendo dei dispositi indossabili, l'interazione che l'utente ha risulta essere più naturale e immediata. Esempi di dispositivi di questo genere in commercio sono i Microsoft HoloLens, i Magic Leap e i Google Glass Enterprise.

2.5.2 Svantaggi della Realtà Aumentata

Anche la Realtà Aumenta non è esente da limitazioni. Questa tecnologia è profondente dipendente dalle condizioni dell'ambiente circostante, in particolare dalle condizioni di luce, presenza di superfici riconoscibili e dalla qualità del tracking spaziale. Non trovandoci in un ambiente completamente digitale, la manipolazione degli oggetti virtuali, essendo meno controllata, è meno accurata. Infine, sebbene i dispositivi come smartphone e tablet sono accessibili, i dispositivi più avanzati hanno costi elevati

2.6 Posing in Realtà Aumentata

Anche la realtà aumentata può essere impiegata per il posing, adattandosi ai dispositivi specifici di questa tecnologia. Esempi di applicativi in letteratura che sfruttano l'AR per il posing sono [20], il cui obiettivo è superare le limitazioni imposte da dispositivi tradizionali come mouse e tastiera, offrendo una maggiore percezione della tridimensionalità dei modelli da animare grazie alla loro proiezione in un ambiente AR. Il funzionamento di tale sistema è il seguente: il personaggio da animare viene proiettato nello spazio reale scelto dall'utente. Attraverso un cursore 3D, controllato dalla fotocamera del dispositivo, è possibile selezionare gli handle dello scheletro del personaggio. Una volta selezionato un handle, il suo movimento viene gestito tramite lo spostamento del dispositivo stesso. In questo modo, il cursore associato all'handle si muove in sincronia con la fotocamera del dispositivo, offrendo un controllo a 6DoF dello scheletro. Il movimento degli handle è vincolato dai limiti delle articolazioni che rappresentano. La cinematica

impiegata per il posing è la cinematica inversa. Infine, è possibile salvare le posizioni dei controller in IK per ottenere diverse pose, che possono essere utilizzate per generare animazioni 3D. Sebbene questa soluzione risulti interessante, presenta alcune limitazioni. Ad esempio, quando si lavora con modelli di dimensioni ridotte ("table scale"), può risultare difficile gestire il posing in modo preciso, poiché gli handle più piccoli richiedono un'elevata vicinanza dell'utente al modello per una manipolazione accurata.

2.7 Realtà Mista: Estensione

La **Realtà Mista** (**MR**) combina gli elementi della Realtà Virtuale e della Realtà Aumentata, in modo tale che gli oggetti digitali possano interagire e rispondere dinamicamente all'ambiente fisico. Il punto di forza della MR è la capacità di integrare elementi virtuali con quelli fisici in tempo reale, creando un'esperienza più immersiva e interattiva. Questa tecnologia consente di manipolare oggetti digitali senza l'uso di controller tradizionali, sfruttando il riconoscimento avanzato delle mani e il tracciamento spaziale. Le applicazioni della Realtà Mista si concentrano principalmente su ambiti professionali e avanzati, offrendo un alto grado di realismo nelle interazioni e una migliore fusione tra il mondo reale e quello digitale.

2.7.1 Device impiegati

L'interazione con un ambiente di Realtà Mista richiede dispositivi specifici per garantire una perfetta integrazione tra digitale e reale. I principali strumenti utilizzati includono i Visori MR e i Sistemi di Tracciamento delle Mani.

- **Visori MR:** I visori MR sono dispositivi che permettono di visualizzare e interagire con oggetti virtuali nel mondo reale. Questi integrano sensori di profondità e telecamere per mappare l'ambiente circostante e posizionare con precisione gli oggetti digitali nello spazio. Visori in commercio sono, ad esempio, Microsoft HoloLens, Meta Questo Pro o HTC VIVE Focus.

- **Sistemi di Tracciamento delle Mani:** Sono sistemi che permettono il riconoscimento costante della posizione delle mani. Tramite questi sistemi di tracciamento è possibile manipolare gli oggetti digitali direttamente con le mani, senza bisogno di controller fisici. Questi sistemi sono spesso integrati nei visori VR o esterni, tipo tecnologie come Leap Motion Controller, che utilizza sensori a infrarossi per rilevare con precisione i movimenti delle dita e delle mani.

2.7.2 Svantaggi della Realtà Mista

Nonostante i suoi vantaggi, la realtà mista presenta alcuni svantaggi. I dispositivi necessari sono costosi e richiedono prestazioni di calcolo elevate, rendendoli meno accessibili rispetto alle soluzioni VR e AR. Inoltre, essendo una tecnologia relativamente nuova, è ancora poco diffusa sul mercato rispetto ai sistemi VR o AR. Infine, l'accuratezza nel riconoscimento delle mani e la precisione del posizionamento degli oggetti virtuali possono variare in base alle condizioni ambientali, compromettendo talvolta l'esperienza dell'utente.

2.8 Posing in Realtà Mista

Anche nella realtà mista sono stati sviluppati sistemi per il task di posing, e in letteratura si trovano diversi esempi significativi. Uno dei più rilevanti è l'applicativo presentato in [21], che permette di svolgere attività di posing, gestione delle sequenze aniamte e network playback. Questo consente a più artisti e direttori di collaborare insieme e rivedere i contenuti prodotti direttamente in realtà mista. "textitPoseMMR" è stato progettato specificamente per il posing collaborativo, offrendo la possibilità a diversi utenti di fare posing e animare in 3D, sia che si trovino nello stesso luogo o a distanza. In un ambiente AR, gli utenti possono interagire contemporaneamente nel mondo reale e in quello virtuale, con confini spaziali tracciati che definiscono l'area di lavoro e facilitano una collaborazione più naturale. Il sistema è compatibile con dispositivi commerciali, e grazie alla possibilità di fare editing in MR, ogni utente può seguire in tempo reale i cambiamenti apportati dagli altri. Il sistema include diverse funzionalità, come l'Immersive Posing, che permette di scegliere tra due approcci principali per il posing: FK e IK. L'Animation Editing consente di creare keyframe dopo il posing, registrare i movimenti in clip video e rivedere il contenuto creato. La funzionalità di Networking Animation Pose permette l'editing collaborativo delle animazioni in tempo reale, e il Live Collaborative Animation Editing permette di facilitare la collaborazione simultanea tra più utenti. I risultati dimostrano che consentire il controllo da parte di più manipolatori rimuove inefficienze e distrazioni, permettendo agli artisti di concentrarsi sul loro lavoro. È stato suggerito che un modo per passare tra IK e FK sarebbe molto utile. I risultati ottenuti e il sistema sviluppato sono una fonte di ispirazione per replicare alcuni dei meccanismi descritti anche in questo lavoro di tesi.

Il paper [22] propone un sistema ibrido che combina dispositivi tipici della VR e dell'AR per creare un'interfaccia che supporta interazioni tangibili sia in 2D che in 3D. L'idea è quella di integrare le capacità di tracking 3D e gli input spaziali dei controller VR con le funzionalità di touchscreen, feedback tattili e input 2D degli smartphone. In questa configurazione, il controller VR è impiegato per fornire input spaziali, mentre lo smartphone viene utilizzato per gli input 2D e per l'output

multimediale. I controller VR offrono una serie di vantaggi grazie ai loro input fisici come trigger, pulsanti, touchpad e joystick; Queste caratteristiche li rendono strumenti versatili per la manipolazione di oggetti virtuali. L'uso del touchscreen degli smartphone è ormai un'interfaccia familiare e intuitiva per la maggior parte degli utenti, offrendo alta risoluzione dello schermo e riconoscimento multi-touch, risultando quindi efficiente per le interazioni 2D. I test del sistema riportano che il livello di usabilità è buono; tuttavia, alcuni aspetti risultano problematici, come la struttura ingombrante data dall'uso combinato di smartphone e controller.

2.8.1 Posing non in Realtà Estesa

Nella letteratura scientifica sono presenti diversi applicativi che, invece di sfruttare la Realtà Estesa, impiegano altre tipologie di dispositivi per il posing. Considerare queste soluzioni risulta rilevante, poiché possono costituire una fonte di ispirazione per futuri sviluppi o per un'eventuale integrazione all'interno di un ambiente di Realtà Estesa. A differenza dei tradizionali metodi di interazione con i software di modellazione, che si basano su mouse e tastiera, in letteratura esistono esempi in cui si è esplorata l'integrazione di dispositivi alternativi. Un caso interessante è descritto in [23], dove il fulcro del progetto è l'impiego di un'interfaccia gestuale basata sul tracciamento delle mani. Il paper si ispira a "*Character Motion Control Interface with Hand Manipulation Inspired by Puppet Mechanism*" [24], che tuttavia presentava problematiche legate alla scarsa accuratezza nel rilevamento delle articolazioni delle dita. Nel dettaglio del nostro caso, il sistema utilizza un Leap Motion Controller [4] per consentire il drag del modello (e delle sue parti) all'interno dell'ambiente 3D, permettendo così di eseguire il posing. A seconda del modello impiegato, diverse gesture della mano controllano specifiche parti del corpo, con alcune limitazioni imposte dai vincoli di rotazione del polso dell'utente di cui si registrano i movimenti per il controllo del modello. Lo studio ha testato sia la cinematica diretta (poi scartata a causa della sua complessità d'uso) sia la cinematica inversa, risultata più efficace per il controllo del modello. Tra le osservazioni emerse, si evidenzia che un Leap Motion Controller consente di rilevare numerose gesture e di mapparle sul modello selezionato. Tuttavia, per una selezione più precisa delle articolazioni da controllare, è necessario integrare questo dispositivo con un sistema di interazione più generico, come un mouse.

[4] Dispositivo che permette di rilevare i movimenti delle mani e delle dita

Chapter 3

Sistema di Posing Realizzato

Il terzo capitolo si propone di fornire una panoramica degli aspetti tecnici riguardanti il sistema di messa in posa realizzato, con particolare attenzione al design e alla composizione del sistema stesso. Partendo dai software impiegati e dalle risorse coinvolte, si esaminerà la logica alla base della comunicazione tra i vari moduli che compongono il sistema. Verrà quindi descritta nel dettaglio la filiera di creazione di un personaggio virtuale pronto per essere messo in posa, partendo dalle fasi iniziali del rigging e della modellazione, e passando per la creazione del personaggio manipolabile in ambienti di Realtà Estesa. In questa sezione, saranno inoltre esposte le funzionalità delle diverse features che il sistema mette a disposizione dell'utilizzatore. Ogni componente introdotta verrà descritta nel dettaglio, evidenziando il suo ruolo e il funzionamento all'interno del sistema. Il capitolo si concluderà con un focus sul funzionamento del sistema di posing precedentemente in uso (basato sulle animazioni dei personaggi virtuali), in quanto con quest'ultimo verrà effettuato il confronto finale sull'utilizzo.

3.1 Software Utilizzati

Per la realizzazione del progetto sono stati impiegati diversi software specifici del settore della computer grafica. In particolare, ci si riferisce a software per la modellazione 3D dei personaggi, Game Engine per la realizzazione dell'applicativo vero e proprio e un ambiente di sviluppo per lo scripting degli eventi e degli elementi virtuali.

3.1.1 Blender

Blender [1] è un software open source di grafica 3D multipiattaforma, sviluppato da Blender Foundation. Gli scopi di tale software sono molteplici, principalmente la

produzione di contenuti virtuali dalla modellazione di oggetti virtuali alla produzione animata. Le sue funzionalità sono maggiormente sfruttate da 3D artists, designer e animatori. Tra la vasta gamma di operazioni che Blender permette di eseguire troviamo:

- **Modellazione 3D:** Una delle feature principali che Blender mette a disposizione è proprio quella di poter creare e modificare modelli tridimensionali grazie a strumenti di modellazione poligonale, oltre a realizzare sculpting [1], texturing [2] e creazione di materials. [3]

- **Animazione:** Il software include ottimi strumenti per l'animazione, consentendo di creare azioni e movimenti realistici per oggetti e personaggi animati e inanimati, grazie al supporto delle animazioni scheletriche, morphing [4], strumenti per effettuare rigging e simulazioni fisiche.

- **Scripting:** Infine, grazie al supporto Python di cui dispone, è possibile ampliare le funzioni di Blender fornendogli script e plugin e permettendo la customizzazione lato utente delle funzioni del software.

L'utilizzo di Blender in questo elaborato di tesi (soprattutto per le funzioni di rigging e scripting) è stato fondamentale in quanto, imponendo determinate condizioni in fase di modellazione, è possibile sfruttare le informazioni utili ricavabili dalle armature dei personaggi virtuali ed elaborarle nell'ambiente virtuale, il tutto mediante uno script Python personalizzato.

3.1.2 Unity

Unity[25] è un motore di gioco (Game Engine) multipiattaforma sviluppato da **Unity Technologies** utilizzato per la creazione di contenuti interattivi tra cui videogiochi, simulazioni virtuali e applicazioni in realtà virtuale, aumentata ed estesa. Per assolvere a questi scopi, Unity offre un ambiente di sviluppo integrato e versatile nell'utilizzo, permettendo di creare e distribuire applicazioni per piattaforme quali PC, console, dispositivi mobile e web. Unity integra un motore di rendering 2D e 3D con diverse pipeline grafiche; nello specifico sono Universal

[1] Tecnica di modellazione tridimensionale avanzata che simula il processo di scolpire un oggetto fisico, permettendo di aggiungere dettagli complessi alla superficie.

[2] Processo di applicazione di immagini o mappe su una superficie tridimensionale, al fine di aggiungere dettagli visivi e texture fisiche.

[3] Rappresentazione digitale delle proprietà fisiche di una superficie 3D, definendone le caratteristiche relative all'interazione con la luce, come la riflessione e la trasmissione.

[4] Tecnica di animazione 3D che consente la trasformazione graduale di una forma in un'altra.

Render Pipeline (URP) e High Definition Render Pipeline (HDRP), con il supporto di sistemi di creazione di shader [5] (Shader Graph) e calcolo dell'illuminazione statica e dinamica. Tramite Unity è possibile inoltre simulare fenomeni fisici grazie ai suoi due motori fisici quali PhysX (NVIDIA) e Havok Physics, oltre a supportare le funzioni di animazione e rigging. Grazie agli strumenti di animazione offerti da Unity, è stato possibile sviluppare il sistema descritto in questa tesi, oltre alla possibilità di associare ai personaggi virtuali della logica specifica. Per fare ciò, Unity utilizza come linguaggio di programmazione principale **C#**, tramite il quale è possibile realizzare script [6] per definire e pilotare i comportamenti che i **Game Objects**[7] devono assumere. Infine, questo motore grafico mette a disposizione strumenti per lo sviluppo di applicativi multiplayer in cui sincronizzare più enti che collaborano in rete nello stesso ambiente virtuale.

3.1.3 Visual Studio

Visual Studio[26] è un ambiente di sviluppo (IDE, Integrated Development Environment) sviluppato da **Microsoft**, progettato per supportare diversi linguaggi di programmazione e la creazione di applicazioni per varie piattaforme, tra cui desktop, web, mobile e cloud. L'IDE, inoltre, ha integrazioni con diversi strumenti di sviluppo, diventando pratico e utile in diverse situazioni per le fasi di scripting e debugging [8]. Visual Studio vanta IntelliSense, un editor di codice avanzato e completo, comprendendo un sistema di suggerimenti automatici che aiuta nella scrittura del codice. Infine, integra strumenti di Version Control [9] ed è estendibile con plugin[10] o strumenti di terze parti. Visual Studio è stato scelto come IDE per la realizzazione del progetto di tesi per la sua compatibilità e integrazione diretta con Unity, permettendo di realizzare script C# e Python, oltre ad avere strumenti e moduli appositi per Unity e a supporto degli sviluppatori.

[5] Programma che definisce il comportamento visivo di un oggetto in un ambiente 3D

[6] Sequenza di comandi e istruzioni scritte in un linguaggio di programmazione che automatizza e definisce determinati comportamenti da esibire.

[7] Entità all'interno di un motore di gioco che rappresentano oggetti nel mondo virtuale con diverse proprietà associate

[8] Processo di analisi, individuazione e correzione di errori nel codice di un programma

[9] Sstema che gestisce e traccia le modifiche a un insieme di file nel tempo.

[10] Moduli aggiuntivi che estendono le funzionalità di un software principale.

3.2 Risorse Impiegate

Per la realizzazione del sistema di posing, partendo dai software di settore, sono stati necessari anche componenti aggiuntivi e hardware specifico per il corretto funzionamento dell'applicativo. In particolare, sono stati utilizzati asset [11] di varie tipologie, estensioni e tecnologie specifiche compatibili con l'applicativo sviluppato.

3.2.1 Modelli 3D

I **Modelli 3D** costituiscono la base su cui costruire l'armatura necessaria, che il sistema di posing consente di manipolare e mettere in posa. Questi **Asset** sono stati forniti gratuitamente dalla piattaforma online **Sketchfab** [27], che ha permesso il download di modelli utili per la realizzazione del sistema e per il testing da parte degli utenti. Alcuni di questi modelli sono statici (utilizzati durante la fase di testing), mentre i principali sono dotati di un'armatura e, in alcuni casi, animazioni predefinite. I formati dei file dei modelli 3D utilizzati sono **FBX**, **OBJ** e **glTF**, tutti compatibili con Blender. Necessitando di modelli riggati e un set di animazioni per ogni modello da mettere in posa, le animazioni mancanti sono state ottenute dalla piattaforma online **Mixamo**[28], che fornisce pacchetti di animazioni già pronte e adattabili a modelli riggati. Le animazioni mancanti sono state realizzate tramite Blender.

3.2.2 Pacchetti Unity

Definire e controllare i comportamenti degli oggetti coinvolti nel sistema, verificare il suo funzionamento e effettuare il testing, è stato necessario ricorrere a funzioni e librerie adatte allo scopo. Per ottenere quest componenti si è ricorso a specifici **packages** e **plugin** per Unity, nello specifico:

- **Final IK[29]:** Package avanzato per Unity che fornisce soluzioni per realizzare sistemi controllabili mediante IK. Le sue componenti e funzioni permettono di definire catene cinematiche complesse fornendo una gerarchia di oggetti, come può essere una sequenza di giunti di un braccio robotico o la catena ossea di un braccio. Il pacchetto offre il controllo su una serie di variabili legate alle catene cinematiche, come i pesi di ogni giunto sulla catena o l'inserimento di vincoli di rotazione di vario genere. Fornisce inoltre la possibilità di interconnettere differenti catene cinematiche, creandone di sempre più articolare e complesse.

[11] Risorsa di varia natura utilizzabili all'interno di un progetto, come modelli 3D, texture o script.

Final IK è ampiamente utilizzato per animazioni dinamiche dove i personaggi devono interagire con oggetti o ambienti in tempo reale.

- **Quick Outline[30]:** Package che consente di creare effetti di contorno (Outline) sui modelli 3D, utilizzato per evidenziare oggetti specifici all'interno di una scena. Particolarmente utile in applicazioni interattive per rendere gli oggetti interagibili più evidenti e facilmente riconoscibili dall'utente.
- **Microsoft MRTK[31]:** Il Mixed Reality Toolkit è un framework open source sviluppato da Microsoft per la creazione di applicazioni di Realtà Aumentata e Realtà Mista in Unity. È pensato principalmente per dispositivi come HoloLens e Windows Mixed Reality, ma supporta anche altre piattaforme come OpenXR e visori Meta/Oculus. Offre strumenti per gesti, hand tracking e interfacce utente immersive. Ai fini di questo elaborato di tesi, dopo un iniziale utilizzo, è stato deprecato in favore di package utili a realizzare applicativi specifici per sistemi Meta.
- **Meta SDK[32]:** Package fornito da Meta[33] per Unity atto a fornire gli elementi per lo sviluppo di applicazioni in Realtà Estesa su dispositivi come Meta Quest 2, Quest 3, e Quest Pro. Include strumenti per la gestione di input da controller, hand e head tracking, e ottimizzazione delle prestazioni per esperienze immersive in applicativi specifici per sistemi Meta. Infine, il package fornisce inoltre tutti gli elementi visuali secondo le guideline di Meta.

3.2.3 Visori per la Realtà Estesa

Oltre ai software, è necessario disporre di hardware specifico per la Realtà Estesa, in grado di supportare e realizzare applicazioni del settore. Dopo un iniziale deploy del sistema su **Oculus Rift**, il dispositivo finale scelto per l'implementazione e destinazione d'uso del sistema sviluppato in questa tesi è il **Meta Quest Pro**. Questo dispositivo, realizzato da Meta, combina funzioni di VR e AR, offrendo esperienze immersive avanzate, sia per giochi che per applicazioni professionali, come simulazioni o design 3D. Nel nostro caso, si integra perfettamente con l'applicativo dedicato alla creazione di storyboard in XR.

3.3 Progettazione del Sistema di Posing

Dopo aver mostrato gli strumenti e le risorse impiegate, si effettua una panoramica del sistema, descrivendone la progettazione e il funzionamento, le funzionalità principali e le features che integra, il core della sua struttura, i moduli che lo compongono e come questi comunicano tra loro e permettono di assolvere allo scopo per il cui sistema è stato creato.

3.3.1 Condizioni iniziali

Per funzionare correttamente, il sistema richiede specifiche condizioni iniziali durante la creazione dell'armatura del personaggio virtuale. Tali condizioni devono essere realizzate nel software di modellazione 3D e rigging e necessariamente soddisfatte, pena la possibilità di creare le giuste catene cinematiche al passaggio nell'ambiente virtuale.

3.3.2 Funzionalità principali

Partendo dallo stato dell'arte e mettendosi nell'ottica del suo impiego per la creazione di storyboarding, sono state individuate le seguenti funzionalità principali a supporto del lavoro di un creatore digitale:

- **Visualizzazione delle informazioni principali del personaggio selezionato**
- **Posing del personaggio mediante Cinematica Diretta**
- **Posing del personaggio mediante Cinematica Inversa**
- **Attivazione e Disattivazione delle Catene Cinematiche e Outline di segnalazione dei personaggi manipolabili**
- **Visualizzazione delle informazioni relative ai giunti disponibili in FK e le catene cinematiche manipolabili in IK**
- **Dimensionamento dei giunti e end effector selezionabili**
- **Dimensionamento del Gimbal[12] di rotazione dei giunti in FK**
- **Applicazione delle trasformazioni base quali traslazione, rotazione e scalamento di un personaggio**
- **Salvataggio di posa temporanea, rollback all'ultima posa salvata e reset alla posa di partenza**

[12] **Sistema di supporto meccanico che consente ad un oggetto di ruotare liberamente su più assi**

3.3.3 Controllers e Scripts

Il comportamento degli oggetti in scena in Unity e la logica di funzionamento del sistema sono definiti da una serie di script C# configurati sia come contenitori di classi utili al sistema, sia come controller associati a ciascuna parte del sistema e ai rispettivi GameObject. Inoltre, alla base della creazione delle catene cinematiche in Unity, è stato creato un apposito script Python da eseguire in Blender. Sono stati realizzati un totale di **sette** script fondamentali (di cui sei in C# per Unity e uno in Python per Blender) per il funzionamento dell'intero sistema, **uno** per il controllo della scena Unity, **cinque** impiegati per il controllo della scena e della demo da somministrare fase di testing e **uno** dedicato al porting[13] del sistema di controllo della posa esistente.

3.3.4 Comunicazione tra sottosistemi

Il sistema così definito è un sistema complesso nella sua interezza, in quanto è costituito da una serie di componenti fondamentali per la creazione delle catene cinematiche manipolabili un Unity e da altri moduli che svolgono le diverse funzioni descritte in precedenza. Il sistema è stato progettato seguendo i principi del **Pattern MVC**[14], in cui i descrittori dei modelli 3D rientrano nei **Model**, i controllori del sistema, dei suoi moduli e il controller del creatore dei modelli 3D rientrano nei **Controller** e i Canvas UI e GameObject relativi all'interazione con lo user rientrano nelle **View**. Nello specifico, sono stati definiti le seguenti classi di funzionamento e controllori:

- **Classi utili a definire le catene cinematiche in Unity a partire dai modelli in Blender.**
- **Controllore relativo al singolo GameObject associato al personaggio.**
- **Controllore della generazione dei GameObject in Unity a partire dai modelli Blender.**
- **Controllore generale del sistema di cinematica e controllo del personaggio**
- **Controllore del sottosistema delle trasformazioni base applicabili al personaggio.**

[13]Processo di adattamento di un software da una piattaforma a un'altra, garantendo che le funzionalità originali siano preservate.

[14]Pattern architetturale utilizzato nello sviluppo di software, che separa l'applicazione in tre componenti principali: Model, View e Controller.

- **Controllore del Gimbal per i giunti in FK.**

La logica di funzionamento dell'intero sistema è semplice nei passaggi ma articolata nella comunicazione e connessione tra le varie parti. Il sistema prende in input un file FBX contenente un modello 3D e un file XML che ne descrive la struttura dell'armatura, modulo di generazione dei modelli in Unity procede a creare un GameObject per ogni FBX. Tale controller, a partire da un GameObject generale con specifiche componenti e funzionalità, crea tanti GameObject per per ogni modello, componendone la struttura e attivando il relativo GameObject che ne descrive le feature e le variabili che contraddistinguono ogni modello. All'attivazione, il descrittore, tramite l'utilizzo di classi e utili a definire le catene cinematiche e componenti dai pacchetti provenienti da Final IK, procede a creare i giunti di rotazione per la FK e gli end effector per la IK e ad associarli all'armatura del modello, ognuno relativo ad ogni singolo giunto e end effector dell'armatura stessa. Ogni GameObject rappresentante un personaggio diventa così un'entità interattiva e manipolabile; in contemporanea, è stato realizzato un GameObject che rappresenta il controller centrale del sistema di posing. A questo è associato uno script dedicato alla gestione degli altri moduli del sistema (sottosistema delle trasformazioni e Gimbal per le rotazioni con relativi controller) e alle interazioni con loro, il controllo della modalità di cinematica scelta, il dimensionamento degli handle interagibili (IK, FK e Gimbal) e la gestione delle pose create. Per l'interazione con il controller centrale del sistema è stato realizzato un pannello di controllo interagibile (configurato come un Canvas UI) che permette la selezione delle funzioni desiderate. I moduli con i quali il sistema centrale comunica sono i controllori del sistema delle trasformazioni base e del Gimbal per le rotazioni. Il primo raccoglie tutti i comportamenti associati ai componenti interagibili per il controllo della traslazione, rotazione e scala e li applica al personaggio scelto, passato al "sottosistema tramite il controller centrale. Il secondo permette di ottenere dal controller centrale quale giunto deve essere manipolato, tramite un Gimbal apposito, permette di controllarne le rotazioni.

3.3.5 Modalità di interazione con il sistema

Essendo pensato per un ambiente in Realtà Estesa, le modalità di interazione con gli elementi del sistema sono molteplici e sfruttano diverse possibilità che sistemi di XR, come i prodotti Meta, mettono a disposizione degli utenti. Le principali modalità di interazione con gli oggetti virtuali sono le seguenti:

- **Controller:** Questa è la modalità standard di interazione negli ambienti VR: un utente possono utilizzare i classici controller sia avvicinandoli direttamente agli oggetti, sia puntandoli tramite un raggio proiettato dal controller stesso.

- **Hand Tracking:** Grazie ai moderni sistemi di riconoscimento delle mani e gesture detection integrati nei dispositivi AR e XR, è possibile interagire direttamente con gli oggetti virtuali avvicinando la mano o utilizzando un raggio proiettato dalla mano stessa.

Vediamo quindi nello specifico il funzionamento delle singole componenti del sistema, partendo dalle condizioni di modellazione fino alle ino alle funzionalità di gestione della posa.

3.4 Pre-elaborazione in Blender

Il primo passo per il corretto funzionamento del sistema così definito, per permettere la creazione delle catene di controllo in Unity coerenti con quelle cinematiche definite sull'armatura in fase di rigging, è quello di applicare specifiche condizioni sull'armatura stessa del personaggio. Una volta impostate tali condizioni, è quindi necessario accompagnare l'esportazione del modello in formato FBX anche un file XML associato che descrive le caratteristiche dell'armatura.

3.4.1 Condizioni al Rigging

In fase di rigging di un modello vengono definite le catene cinematiche sull'armatura, come queste sono composte, come si comportano e quale impatto hanno sulla mesh associata al personaggio. Per garantire il corretto riconoscimento del ruolo di ogni osso delle catene da trasferire in Unity (e per implementare sistemi di controllo in Forward Kinematis o fare porting utilizzando le componenti di Final IK per l'Inverse Kinematic), le condizioni da rispettare consistono nell'utilizzo di una nomenclatura specifica relativa al ruolo di ogni osso. Imporre questo passaggio in fase di modellazione consente di automatizzare il processo di creazione delle catene di controllo in Unity. In questo modo, non è necessario eseguire operazioni aggiuntive nel motore di gioco ma semplicemente segnalare al generatore dei modelli manipolabili le cartelle contenenti gli FBX e XML dei personaggi di cui fare posing in Unity. In particolare:

- **Nome Osso + " [ROOT]":** Identifica il nodo padre della catena cinematica e permette di segnalare quando una catena inizia.
- **Nome Osso + " [BONE]":** Identifica un semplice osso che fa della catena cinematica permette di segnalare quale osso fa parte della catena cinematica di cui [ROOT] è il nodo padre.
- **Nome Osso + " [LEAF]":** Identifica l'ultimo nodo della catena cinematica e permette di segnalare quando la catena termina.

- **Nome Osso + " [END EFFECTOR]":** Identifica quale osso della catena è impiegato come End Effector da impiegare come controllore dell'intera catena in IK.
- **Nome Osso + " [CHAIN]":** In caso di catene complesse e innestate, permette di identificare una nuova catena innestata di cui [ROOT] è il nodo padre.

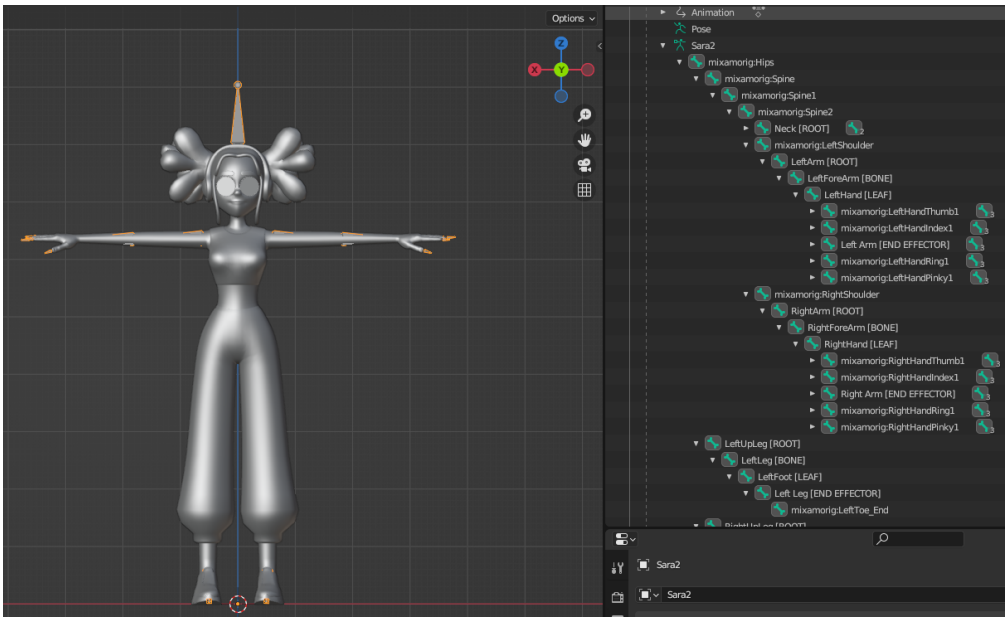

Figure 3.1: Struttura gerarchica delle ossa di uno scheletro in Blender con nomenclatura predefinita secondo condizione impostata

Utilizzare una tale nomenclatura servirà agli script in Unity per identificare nella gerarchia del GameObject che rappresenta lo scheletro del personaggio quali GameObjects costituiscono una catena e la sua struttura, avendo contezza del ruolo di ogni osso in essa. In caso di catene complesse, lo script permette l'identificazione dei diversi rami che insistono sullo stesso nodo e creare differenti sistemi di controllo per ogni catena innestata. Una volta applicata tale nomenclatura, si procede ad esportare il modello in formato FBX.

3.4.2 Generazione XML del Modello

Ad accompagnare la versione FBX da consegnare a Unity, il sistema necessita di accedere alle informazioni sull'armatura del modello descritte mediante un file formato XML. Le informazioni delle ossa sono riportate in ordine sequenziale e non

gerarchico. Attraverso la lettura di tale file è possibile ottenere diverse informazioni in merito alle ossa come posizione, rotazione, scala e possibili limiti applicati alle rotazioni delle ossa. Grazie alla completezza delle informazioni contenute nell'XML si è scelto di utilizzare questo formato per identificare, nel GameObject dell'FBX, quali elementi della gerarchia dello scheletro sono rilevanti per la definizione di una catena e per determinare se applicare specifici componenti in base al ruolo di ciascun elemento. Lo script Python eseguibile in Blender è riportato nella sezione *Appendix A.1.* Sia l'XML che l'FXB devono avere lo stesso nome quando inseriti nelle apposite cartelle per la lettura in Unity, previa impossibilità di associare l'FBX al rispettivo file descrittivo.

3.5 Sistema di Posing basato su Tecniche di Cinematica

Definite le condizioni necessarie per la corretta lettura del modello quando si effettua il porting da FBX a GameObject (previa impossibilità di creare corrette catene cinematiche e sistemi di manipolazione dell'armatura), approfondiamo il funzionamento delle componenti del sistema centrale. In particolare, il core del sistema consiste nella possibilità di prendere la cartella contenente gli FBX e XML, creare un GameObject per ogni modello (seguendo una specifica struttura), associargli un descrittore e raccogliere tutti i GameObject. Il descrittore avrà il compito di creare le catene cinematiche e di controllo dell'armatura del singolo personaggio e da interfaccia tra il sistema centrale e il modello, passando tutte le informazioni necessarie per la manipolazione del personaggio. Il sistema centrale, a sua volta, una volta selezionato il personaggio da mettere in posa, ricava le informazioni dal descrittore associato e permette il controllo mediante cinematica diretta e inversa, assumendosi la responsabilità principale nella manipolazione dell'armatura secondo la cinematica scelta. Inoltre, il sistema centrale gestisce il controllo degli handle (GameObject associati all'armatura) e il loro dimensionamento. Infine, consente di salvare le informazioni circa una posa realzzata, effettuare rollback all'ultima posa salvata o ripristinare la posa iniziale del personaggio. Al sistema centrale è connesso il sottosistema per applicare le trasformazioni base (traslazione, rotazione e scala) e il Gimbal responsabile delle rotazioni dei giunti quando si effettua FK. Ora si analizza nel dettaglio il comportamento di ciascuna componente e il ruolo che ricopre.

3.5.1 Classi per le Catene Cinematiche

IIl primo script fondamentale per il sistema centrale e per la gestione della creazione dei GameObject associati ai modelli è quello contenente una serie di classi utili alla

rappresentazione delle catene cinematiche nella scena. Lo script `KinematicChains` raccoglie tre classi serializzabili:

- `BoneChain`: Classe che modella una catena cinematica. Come attributi ha `List<Transforms> bones`, contenente le informazioni rappresentanti dei giunti che compongono una catena di interesse e `List<string> boneNames`, contenente i nomi dei giunti della catena. Provvista di Costruttore, metodi `Get()` e `Set()`, metodi di aggiunta alle liste un nuovo elemento, metodi per il conteggio degli elementi presenti nelle liste.

- `HandlesChain`: Classe che modella una catena di handles. Come attributi ha `List<Transforms> handles`, contenente le informazioni rappresentanti degli handles da associare ad una catena cinematica e `List<string> handleNames`, contenente i nomi degli handles. Provvista di Costruttore, metodi `Get()` e `Set()`, metodi di aggiunta alle liste un nuovo elemento, metodi per il conteggio degli elementi presenti nelle liste.

- `BonesTransform`: Classe che modella le posizioni e le rotazioni dei giunti di una catena cinematica. Come attributi ha `List<Vector3> positions`, contenente le informazioni sulle posizioni dei giunti della catena e `List<Quaternion> rotations`, le informazioni sulle rotazioni dei giunti della catena. Provvista di Costruttore, metodi `Get()` e `Set()`, metodi di aggiunta alle liste un nuovo elemento, metodi per il conteggio degli elementi presenti nelle liste.

Queste classi saranno impiegate dai successivi script per modellare e tracciare, via codice, le informazioni relative ai GameObject presenti nella scena. In particolare, esse consentiranno di gestire i giunti delle catene cinematiche, gli handles utilizzati per la manipolazione e l'insieme di posizioni e rotazioni dei giunti che definiscono la posa assunta da un personaggio. Possibile consultare il codice relativo alle classi custom create nella sezione *Appendix A.2*.

3.5.2 Generatore dei Modelli Manipolabili

Altro componente fondamentale del sistema centrale è il GameObject responsabile della creazione dei modelli. Questo contiene lo script responsabile di tale operazione, chiamato `CharacterCreationManager`, il quale necessita i seguenti input:

- Percorso della cartella contenente gli FBX dei modelli da manipolare contenuta nella directory "Resources".

- Percorso della cartella contenente gli XML dei modelli da manipolare contenuta nella directory "Resources".

- GameObject nel quale raccogliere tutti i modelli generati in Unity e farli diventare tutti figli parietari dello stesso padre.
- Prefab che costituisce la base da dove istanziare e caratterizzare il nuovo GameObject associato al modello da manipolare. Tale prefab prende il nome di CharacterCretorPack.

Lo scopo principale dello script è, partendo dal CharacterCretorPack, istanziare tanti GameObject per quanti sono i modelli presenti nell'apposita directory. Gli elementi presenti nel CharacterCretorPack sono i seguenti GameObject:

- **CharacterDescriptor:** GameObject che ha come unico componente lo script che raccoglie i dati relativi al singolo modello e permette la creazione delle catene cinematiche per IK e catene di controllo per FK
- **Body:** GameObject che rappresenta il modello vero e prorpio, al cui interno sono presenti ulteriori due elementi, ovvero il GameObject *IKTargets* che conterrà gli handle interagibili per fare cinematica inversa e il GameObject *InteractableComponent*, responsabile di contenere i componenti Meta per l'interazione con il modello in realtà estesa.
- **KeyPoints:** GameObejct che raccoglie ulteriori tre elementi relativi ai punti cardine di ogni modello, ovvero il punto più basso, più alto e il punto centrale del modello.

Sfruttando tutti elementi presenti nel CharacterCretorPack, lo script provvede quindi a scorrere i modello all'interno della cartella apposita e, per ognuno di essi, associare le componenti di cui è fatto l'FBX agli elementi presenti nel Prefab. In particolare, prende la *Mesh* e l'*Armatura* dall'FBX e li rende figli di un nuovo CharacterCretorPack (di fatto portando gli elementi essenziali di un modello in Unity nel prefab), aggiungendo nuovi componenti come un apposito Collider[15] alla mesh (basato sulla stessa) e l'Outline da attivare quando si vuol notificare che si può interagire con un certo personaggio. I renstanti elementi presenti nel Prefab verranno poi modificati e inizializzati dal descrittore del personaggio. Ultima azione compiuta da CharacterCreationManager è quella di leggere, a partire dal nome dell'FBX che sta considerando, il relativo file XML dall'apposita cartella procedere a definire le ossa nello scheletro che costituiscono catene utili. A svolgere questo compito è la funzione `FindBoneSequence`, la quale ha come parametri:

- `XmlNode rootNode:` Elemento del documento XML identificato come nodo padre

[15]Componente utilizzato in Computer Grafica per rilevare le collisioni tra oggetti all'interno di un ambiente 3D.

- `GameObject characterFBXSkeleton:` GameObject dell'FBX il quale contiene al suo interno la gerarchia di ossa e giunti che compongono l'armatura del personaggio
- `CharacterManager characterManager:` Descrittore del GameObject relativo ad un personaggio e unico componente di CharacterDescriptor.

La funzione FindBoneSequence ha come scopo principale quello di instanziare tanti oggetti BoneChain della classe relativa per quante sono le catene cinematiche che riconosce dal file XML. Per riconoscere quali sono le ossa utili che compongono possibili catene sfrutta i i loro nomi; se nel nome è presente "[ROOT]", "[BONE]", "[LEAF]", "[END EFFECTOR]" o "[CHAIN]" identifica quelle ossa come utili e le inserisce all'interno delle liste di Transforms e strings tramite il costruttore della classe BoneChain. L'inserizione in lista, rispetto alla gerarchia che riconosce secondo i legami esplicitati precedentemente. Successivamente, inserisce all'interno del descrittore ogni BoneChain che riconosce come completa (arrivati a leggere l'osso con il nome [END EFFECTOR]) e il riferimento a ogni End Effector che riconosce come tale. Grazie a questa funzione è possibile anche identificare le catene cinematiche innestate e più complesse. La logica e il codice che compone le funzioni citate è riportato nella sezione *Appendix A.3*

3.5.3 Descrittore del Modello 3D

Il descrittore di un personaggio in scena è un GameObject, contenuto nel personaggio stesso, al quale è associato come unico componente lo script `CharacterManager`, il quale ha due scopi principali nel funzionamento. Il primo scopo è fornire un'interfaccia che permetta al sistema centrale di posing di accedere a tutte le informazioni necessarie per manipolare il personaggio. Tra queste rientrano, ad esempio, la dimensione degli handle e la posa salvata. Il secondo è prendere le `BonesChain` e i riferimenti agli *End Effector*,impostati dal CharacterCreationManager, e procedere a definire tutti gli elementi necessari per la manipolazione delle catene cinematiche in FK e IK, oltre a impostare correttamente i GameObject relativi a punti cardine del modello. Nello specifico, gli elementi che permettono il posing sono:

- **Handle FK:** GameObject composto da una sfera interagibile e un nome visibile su un Canvas, utilizzato per la selezione un giunto da manipolare in FK.
- **Target IK:** GameObject composto da una sfera interagibile e un nome visibile su un Canvas, utilizzato per manipolare una catena cinematica in IK.
- **FABRIK Component:** Component del Package *Final IK* che, associato ad un nodo padre di una catena e segnalandogli le ossa ne fanno parte, tramite

un risolutore, permette di calcolare automaticamente i legami tra le ossa, associandole tra loro e dando accesso alla manipolazione completa della catena in IK tramite End Effector (da segnalare anch'esso).

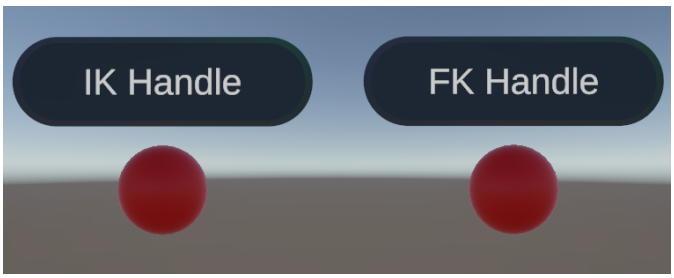

Figure 3.2: Da sinistra: Handle per IK e Handle per FK impiegati in Unity per la costruzione dei sistemi di controllo della cinematica.

A partire da questi elementi base, lo script permette di costruire il sistema necessario per poter controllare le catene sia in cinematica diretta che inversa. Come ultima azione, la classe `CharacterManager` procede a salvare la posa di partenza del modello e inizializzare i campi relativi all'ultima posa salvata e realizzata. Queste informazioni (posizioni, rotazioni e dimensioni dei handle FK e target IK) vengono archiviate all'interno di specifici attributi della classe:

- Per la Cinematica Diretta (FK):
  `List<BonesTransform startBoneTransforms`,
  `List<BonesTransform> savedBoneTransforms`,
  `List<BonesTransform> lastBoneTransforms`.

- Per la Cinematica Inversa (IK):
  `List<Vector3> startIKTargetPositions`,
  `List<Vector3> savedIKTargetPositions`,
  `List<Vector3> lastIKTargetPositions`.

Controllo della Cinematica Diretta

Per il controllo della cinematica diretta lo script esegue `CreateHandleChain (BonesChain boneChains)`, la quale viene chiamata tante volte quante sono le `BonesChain` fornite dal descrittore del modello, accettandole come parametro. Per ogni `BonesChain` segnalata, la funzione istanzia la classe `HandlesChain` e riempire le liste che la compongono. Dopo aver ricavato le ossa e i relativi nomi da una `BonesChain`, esegue la funzione `CreateHandle(Transform bone, string boneName)`, grazie alla quale istanzia un GameObject handle a partire dal prefab

FK Handle che viene collocato nella stessa gerarchia e allo stesso livello del GameObject che rappresenta il giunto nell'armatura del personaggio, acquisendone il nome. La funzione procede ad attivarne il componente Outline, impostare il testo del Canvas con il nome dell'osso. Infine, i riferimenti a ciascun handle creato vengono aggiunti alla lista `List<Transforms> handles` mentre i rispettivi nomi vengono archiviati in `List<string> handleNames`. In questo modo, per ogni `BonesChain` viene associato una `HandlesChain`, creando una corrispondenza implicita tra i GameObject delle ossa e degli handle.

Controllo della Cinematica Inversa

Per il controllo della cinematica diretta lo script esegue `CreateFABRIKChain (BonesChain boneChains, Transform endEffector, string targetName)`, la quale viene chiamata tante volte quante sono le `BonesChain` fornite dal descrittore del modello; accetta come parametri una `BonesChain`, l'End Effector della stessa e il relativo nome. Per ogni `BonesChain`, la funzione acquisisce i riferimenti al primo e all'ultimo osso escludendo l'End Effector, dove al primo viene aggiunto il componente FABRIK, assegnandogli i riferimenti a tutte le ossa della `BonesChain`, in modo da poter dare al *solver* l'insieme dei GameObject da mettere in relazione per creare una catena cinematica in Unity. Contemporaneamente, viene eseguita la funzione `CreateTarget(Transform endEffector, string targetName)`, la quale procede ad istanziare un GameObject *target* a partire dal Prefab *IK Handle* e a collocarlo all'interno della collezione di oggetti *IKTargets* specifica del modello. La funzione procede ad attivarne il componente Outline, impostare il testo del Canvas con il nome dell'End Effector dato in input come parametro. Infine, i riferimenti a ciascuna nuova catena cinematica Unity creata viene inserita in nella lista `List<FABRIK> IKFABRIKChains`, attributo della classe descrittore `CharacterManager`. I riferimenti a tutti i targets delle catene cinematiche vengono inseriti nella lista `List<Transform> IKTargets`, altro attributo di `CharacterManager`.

3.5.4 Controller Centrale del Sistema

Altro componente fondamentale del sistema è il GameObject a cui è associato lo script del Controller centrale del Sistema, chiamato `KinematicSystemController`. Questo script, insieme al Generatore dei modelli, costituisce il core dell'intero sistema, in quanto è il responsabile della scelta della modalità di cinematica da impiegare e ne coordina i sistemi di controllo, raccogliendo in se tutte le funzioni associate ai comportamenti da far assumere agli oggetti in scena quando si interagisce con loro.Inoltre, nel `KinematicSystemController` convergono tutti i moduli che costituiscono le funzioni che il sistema mette a disposizione dell'utente,

permettendo l'accesso a tutte le funzionalità, come il dimensionamento degli handle, le trasformazioni di base e la gestione e salvataggio delle pose. Se non è selezionato nessun personaggio, il Controller centrale rimane in attesa che, all'interazione con un modello manipolabile (tramite il suo GameObject InteractableComponent), il suo descrittore comunichi al controller tutte le informazioni necessarie per la sua gestione e manipolazione.

User Interface del Sistema

L'**Interfaccia Utente** (UI) realizzata consiste principalmente in un Canvas, sempre visibile e interagibile nell'ambiente. Inizialmente sviluppata utilizzando i componenti forniti dal Package MRTK, è stata successivamente sottoposta a un processo di porting. Durante questa fase, sono stati modificati sia i componenti che la costituiscono (con i relativi adattamenti delle funzioni associate agli elementi visivi interagibili) sia la disposizione dei comandi e degli elementi sul Canvas. Inoltre, sono stati impiegati elementi grafici proprietari e specifici di Meta, uniformando lo stile dell'interfaccia alle relative linee guida.

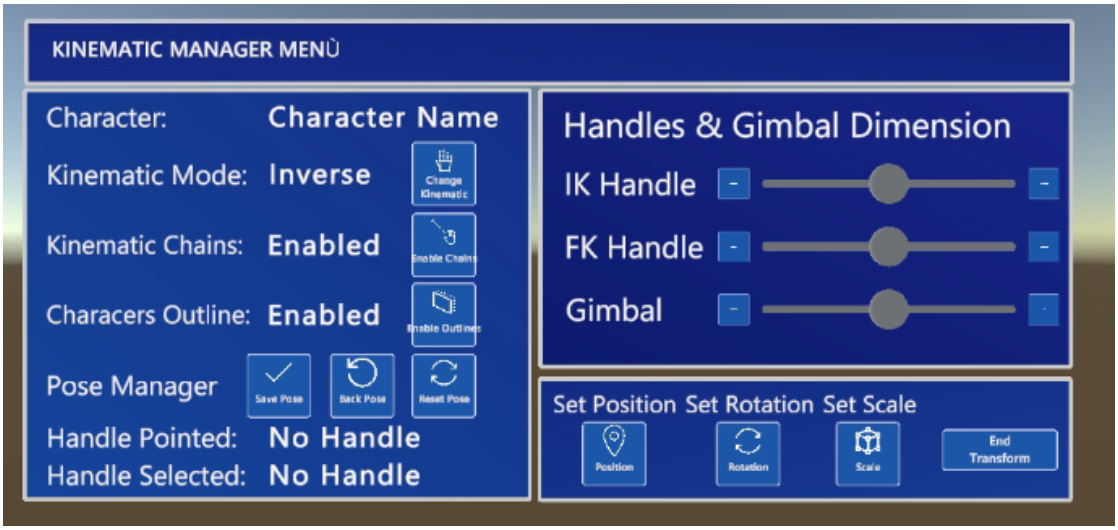

Figure 3.3: Pannello di controllo del Sistema di posing basato su Cinematica Diretta e Inversa con le diverse funzioni selezionabili, realizzato con i componenti Microsoft MRTK

L'utente può interagire con il Canvas utilizzando controller, mani o il raggio generato da essi. È possibile spostarlo e ruotarlo attorno al suo asse verticale semplicemente interagendo con i suoi contorni. Il Canvas è strutturato in un menu di selezione suddiviso in diverse sezioni, ognuna dedicata a una specifica funzionalità del sistema, tutte accessibili all'interno di una vista scrollabile. Le modalità di

interazione con gli elementi grafici restano le stesse utilizzate per il Canvas, quindi tramite controller, mani e raggio da quest'ultimi. Un elemento sempre disponibile, indipendentemente dalla selezione di un personaggio da posizionare, è il *Toggle Button* per l'attivazione del **Passthrough**[16]. Questo pulsante è collocato nell'angolo in alto a destra dell'interfaccia ed è attivabile in qualsiasi momento. Nelle sezioni seguenti verrà descritto nel dettaglio il ruolo di ciascuna funzionalità offerta dal sistema.

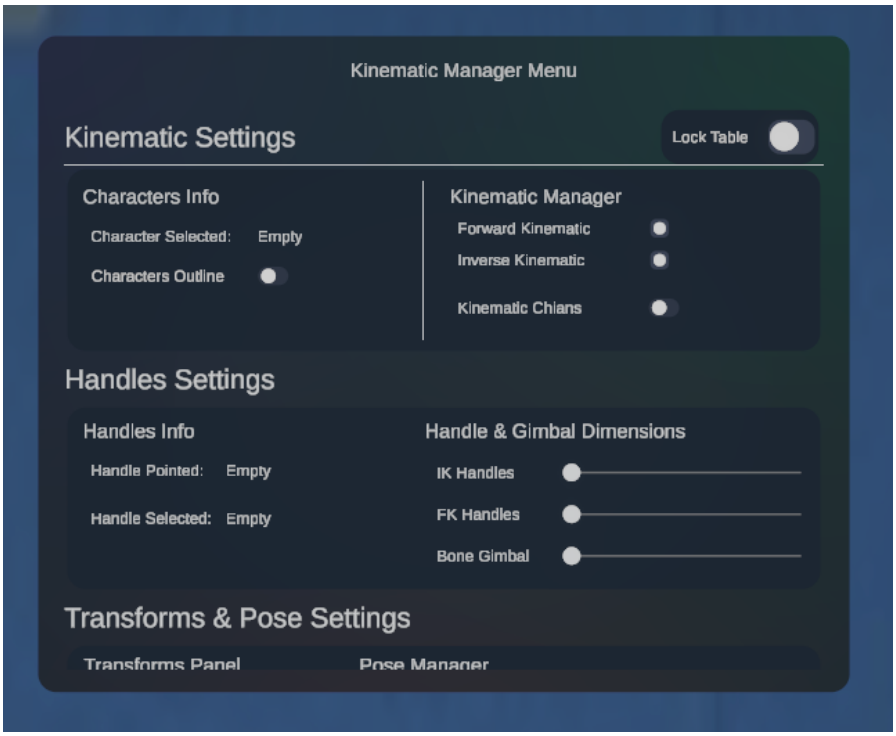

Figure 3.4: Pannello di controllo del Sistema di posing basato su Cinematica Diretta e Inversa con le diverse funzioni selezionabili, realizzato con i componenti Meta SDK

Associazione Descrittore - Sistema Centrale

Ogni modello include un *GameObject* che consente e risponde alle interazioni in Realtà Estesa tramite controller e mani. Quando selezionato o preso, l'*InteractableComponent* del personaggio esegue la funzione `SetCharacterToPose()`

[16]Tecnologia impiegata nei visori di realtà virtuale e realtà mista che consente all'utente di vedere il mondo reale attraverso le telecamere integrate nel visore

dello script descrittore, che assegna all'attributo `characterDescriptor` del `KinematicSystemController` l'intero descrittore del personaggio. Contestualmente, quando un nuovo personaggio viene assegnato tramite il descrittore, il controller centrale esegue nel suo metodo `Update()` un blocco di codice che trasferisce i valori del descrittore agli attributi del controller stesso, chiamando la funzione `SetUpCharacterKinematicManager()`. Inoltre, vengono invocate due ulteriori funzioni con lo scopo di trasferire i valori degli attributi del descrittore alla UI per il dimensionamento degli handle e al sistema delle Trasformazioni per applicare le modifiche al personaggio. Queste funzioni sono, rispettivamente `SetFromGimbalHandleSliderValue()` e `SetUpCharacterTransformManager()`. Tutte le funzioni che riguardano il passaggio di informazioni tra il descrittore e il controller del sistema centrale sono riportate nella sezione *Appendix A.4*. Una volta acquisite tutte le informazioni necessarie, il sistema procede a effettuare un mapping ufficiale tra gli elementi presenti nelle liste `BoneChains` e `HandlesChain` del modello, al fine di permettere una ricerca più facile e veloce del giusto osso da ruotare quando viene selezionato l'handle a esso associato. La funzion responsabile del mapping è consultabile nella sezione *Appendix* **??** Una volta completate le dovute impostazioni (che si ripetono ogni volta che viene selezionato un nuovo personaggio), il sistema centrale consente all'utente di selezionare, tramite l'interfaccia apposita, la modalità di cinematica da utilizzare, dando accesso a tutte le altre funzionalità offerte dal sistema. Vediamo ora nello specifico come sia possibile effettuare il posing per le due modalità di cinematica.

Figure 3.5: Sezione del pannello di controllo che mostra le informazioni del personaggio, la scelta della modalità di cinematica e l'attivazione o disattivazione dell'outline e delle catene cinematiche

Posing in FK

Quando l'utente seleziona l'opzione "Forward Kinematic" nell'interfaccia, il sistema disattiva la collezione dei targets per l'IK e attiva tutti gli handle delle

varie `HandlesChain`. In questo modo, sul modello diventano visibili le sfere corrispondenti alle ossa, accompagnate dai relativi nomi, consentendo all'utente di riconoscere facilmente quale osso andrà a ruotare selezionando l'handle, senza la necessità di ricorrere all'interfaccia. Interagendo con un handle, quest scompare e viene sostituito dal sistema dal GameObject che gestisce la rotazione dell'osso relativo a quel giunto. Questo è il **Gimbal**, un GameObject composto da tre anelli, ciascuno relativo a uno degli assi X, Y e Z. Ogni anello è colorato in base dell'asse attorno al quale ruota: **Rosso** per l'asse X, **Verde** per l'asse Y e **Blu** per l'asse Z. La struttura e la composizione ricalca i classici Gizmo[17] utilizzati nei software grafici. L'utente, avvicinando il controller o le mani, può interagire con gli anelli, scegliendo l'asse intorno al quale desidera ruotare l'osso. Quando il controller si avvicina a uno degli anelli, questo viene evidenziato dall'attivazione del relativo Outline. Il sistema centrale, alla selezione dell'handle, comunica con il controller del Gimbal, associato a quest'ultimo tramite un componente apposito che applica lo script `GimbalController`, informandolo su quale osso è stato selezionato. Questo avviene grazie al mapping tra giunti e ossa, effettuato al momento della selezione del personaggio.Il controller del Gimbal prende quindi in carico la rotazione dell'osso, lasciando che l'osso ruoti automaticamente in risposta alla rotazione impostata tramite l'interazione con il Gimbal. Parallelamente, il controller centrale si occupa della gestione dei target per l'IK. Infatti, quando un osso viene ruotato, l'intera catena cinematica (a partire da quell'osso) subisce una variazione di posizione, rendendo necessario l'aggiornamento della posizione del target in base alla nuova configurazione dell'End Effector. Quando viene selezionata la modalità FK, nell'apposita sezione dell'interfaccia utente appaiono degli slider interagibili, che permettono di dimensionare gli handle (insieme ai relativi nomi) da selezionare e il Gimbal da ruotare. I nomi degli handle puntati, avvicinati o selezionati sono visibili nella stessa sezione.

Posing in IK

Quando l'utente seleziona l'opzione "Inverse Kinematic" nell'interfaccia, il sistema disattiva i gli handles presenti nelle liste delle `HandlesChain` per FK attiva tutti gli elementi della collezione dei target per la manipolazione delle catene in IK, oltre ad abilitare i componenti FABRIK presenti nella IKFABRIKChains. In questo modo, sul modello diventano visibili le sfere corrispondenti agli End Effector delle catene, accompagnate dai relativi nomi, consentendo all'utente di riconoscere facilmente il target che sta scegliendo quale catena manipola, senza la necessità di ricorrere all'interfaccia. L'utente può quindi afferrare, tramite controller o

[17]Strumento visivo utilizzato in software di grafica 3D per manipolare oggetti, come spostarli, ruotarli o scalarli.

Figure 3.6: Posing tramite cinematica diretta: catene di handle (con nome a vista delle corrispondenti ossa) con Gimbal per il controllo delle rotazioni

mani, il target che controlla la catena, potendolo spostare nello spazio liberamente; automaticamente il componente FABRIK calcolerà la posizione e la rotazione che le ossa della catena devono assumere in relazione alla manipolazione del target relativo all'End Effector, lasciando piena libertà all'utente mettere in posa l'intera catena interagendo esclusivamente con il target. In questo caso, il controller centrale del sistema non deve preoccuparsi di aggiornare la posizione degli handle per la FK, poiché questi ultimi sono stati integrati nella gerarchia dell'armatura del personaggio. Quando una catena cinematica IK viene manipolata, l'intero ramo dell'armatura subisce una variazione di posizione e rotazione (grazie all'azione del componente FABRIK), applicando tali trasformazioni a tutti gli elementi presenti in quel ramo. Di conseguenza, gli handle per la FK vengono aggiornati automaticamente. Quando viene selezionata la modalità IK, nell'apposita sezione dell'interfaccia utente appare uno slider interagibile, che permette di dimensionare i target afferrabili. I nomi dei target delle catene puntati, avvicinati o selezionati sono visibili nella stessa sezione.

Figure 3.7: Posing tramite cinematica inversa: target (con nome a vista delle relative catene) in corrispondenza degli end effector per il controllo delle catene cinematiche

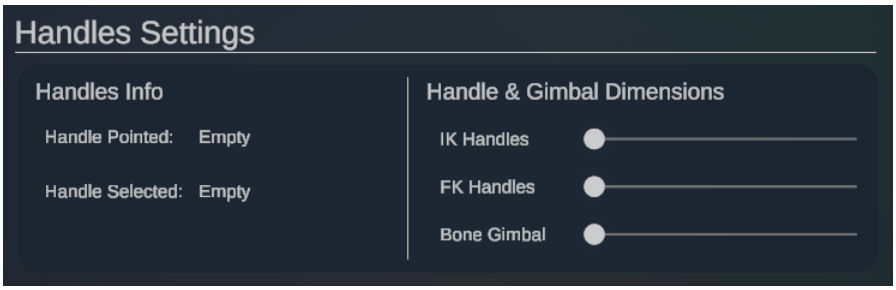

Figure 3.8: Sezione del pannello di controllo che mostra con quali handle si sta interagendo e gli slider per controllare la loro dimensione e quella del Gimbal delle rotazioni

Gestione delle Pose Realizzate

L'ultima funzionalità che il sistema implementa è quella di poter gestire localmente le pose che si stanno realizzando, fornendo la possibilità di salvare una posa

temporanea realizzata, tornare indietro all'ultima posa che è stata salvata o riportare la posa assunta dal personaggio a quella di partenza. Le funzioni che gestiscono la logica delle tre funzioni così definite sono implementate nel controller centrale stesso, oltre a registrare le informazioni relative alla posa da salvare nel descrittore del personaggio. Nello specifico, il controller centrale implementa le funzioni:

- `SavePose()`: Salva le informazioni di base (posizione, rotazione e scala) dei target per IK nell'attributo `List<Vector3> savedIKTargetPositions`, e le rotazioni delle ossa dell'armatura nell'attributo `List<BonesTransform> savedBoneTransforms` del controller centrale, memorizzandole automaticamente nei corrispondenti attributi del descrittore del personaggio.
- `BackToSavedPose()`: Imposta i valori di posizione, rotazione e scala dei target per IK, nonché le rotazioni delle ossa dell'armatura, ai valori precedentemente salvati negli attributi `List<Vector3> savedIKTargetPositions` e `List<BonesTransform> savedBoneTransforms` del descrittore del personaggio.
- `ResetPose()`: Imposta i valori di posizione, rotazione e scala dei target per IK, nonché le rotazioni delle ossa dell'armatura, ai valori che possedevano durante la costruzione del GameObject associato all'FBX, ottenendo tali valori dagli attributi `List<Vector3> savedIKTargetPositions` e `List<BonesTransform> startBoneTransforms` del descrittore del personaggio.

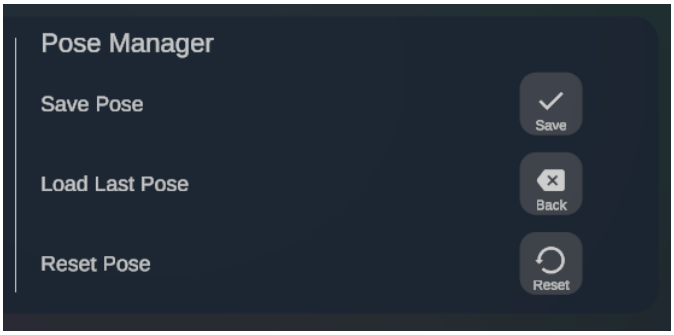

Figure 3.9: Sezione del pannello di controllo le funzioni di salvataggio di una posa, caricamento di una posa precedentemente salvata e reset alla posa di partenza

3.6 Sottosistema delle Trasformazioni

Connesso al sistema centrale della cinematica e della posa, il sistema delle trasformazioni base applicabili consente di gestire e definire la posizione, la rotazione e

le dimensioni del personaggio selezionato. A definire la logica e a implementare le funzioni che realizzano tali operazioni è stato creato uno script, che funge da controller per il sistema; esso prende il nome di `TransformSystemController` ed è associato come componente a un GameObject nella scena. A comporre il sottosistema e ad applicare le trasformazioni base sono tre GameObject separati che, impostati come valori degli attributi `positionSystem`, `rotationSystem` e `scaleSystem` del controller, sono integrati nel sottosistema e la cui logica è definita nel controller; tramite essi, è possibile applicare le tre trasformazioni base. La costruzione dei GameObject relativi all'applicazione di traslazione e rotazione è ispirata ai Gizmos usati nei software di computer grafica. La scelta di quale trasformazione applicare può essere effettuata tramite la UI, nella sezione apposita, che è mutuamente esclusiva rispetto alle altre trasformazioni. Non è possibile applicare una delle tre trasformazioni contemporaneamente al posing in IK o FK, per evitare la caoticità derivante dalla presenza di troppi elementi interagibili in scena e per l'utente. Vediamo nel dettaglio la logica applicata dal sottosistema per applicare ogni trasformazione.

Figure 3.10: Sezione del pannello di controllo per la scelta della trasformazione da applicare tra traslazione, rotazione o scalamento.

3.6.1 Gestore della Posizione

Il prefab responsabile della traslazione del personaggio è un GameObject costituito da un anello alla base e da quattro frecce che indicano le direzioni cardinali. La sua forma è ispirata all'interfaccia fornita dall'add-on **Rigify** [**34**] per Blender, prendendo ispirazione dalla grafica alla base del personaggi su cui si effettua il rigging. Alla selezione della relativa opzione nel menu di scelta, il controller del sottosistema esegue la funzione `SetPosition()`, la quale adatta le dimensioni

del GameObject a quelle del personaggio da traslare, lo posiziona alla sua base (sfruttando i punti cardine del personaggio) e lascia il controllo della posizione al GameObject. Questo può essere interagito tramite controller, mani o raggio da quest'ultimi; afferrandolo, è possibile traslarlo nello spazio, ma non ruotarlo o scalarlo. Grazie alla logica implementata nel controller del sottosistema, ogni traslazione del GameObject nel sistema delle posizioni farà sì che il personaggio segua le sue traslazioni, mantenendo sempre la stessa distanza dalla base. Quando il GameObject è indicato o selezionato, cambia colore per segnalare la possibilità di interazione o che lo si sta manipolando.

Figure 3.11: Personaggio virtuale con alla base il Gizmo per applicargli le traslazioni desiderate

3.6.2 Gestore della Rotazione

Il prefab responsabile delle rotazioni del personaggio è un GameObject costituito da tre anelli, uno per ogni asse di rotazione, ciascuno del colore corrispondente all'asse che rappresenta. Questo GameObject è equivalente a quello utilizzato per le rotazioni degli handle in FK, per cui si ha un Gimbal con le stesse caratteristiche, ma con una funzione differente. In questo caso infatti, il Gimbal ruoterà l'intero personaggio. Alla selezione della relativa opzione nel menu, il controller del sottosistema esegue la funzione `SetRotation()`, che adatta le dimensioni del Gimbal a quelle del personaggio da ruotare, lo posiziona in modo tale da inglobare visivamente al suo interno l'intero personaggio, facendo corrispondere i centri di

entrambi. Successivamente, lascia il controllo della rotazione al Gimbal. Questo può essere interagito esclusivamente tramite controller o mani: è possibile afferrare i suoi anelli e ruotarli, ma non traslarli né scalarli. Grazie alla logica implementata nel controller del sottosistema, ogni volta che uno degli anelli del GameObject viene ruotato, il personaggio seguirà la rotazione e la copierà fedelmente. Quando uno degli anelli è vicino al controller o alle mani, oppure viene selezionato, viene attivato il suo Outline, indicando che l'anello è stato puntato e che può essere ruotato.

Figure 3.12: Personaggio virtuale inserito all'interno del Gimbal per applicargli le rotazioni desiderate

3.6.3 Gestore della Scala

Il GameObject responsabile del dimensionamento del personaggio è uno Slider presente nel menu delle trasformazioni, ma inizialmente inattivo. Alla selezione della relativa opzione nel menu, il controller del sottosistema esegue la funzione `SetScale()`, che attiva lo slider e imposta il suo valore iniziale in base alla scala attuale del personaggio, visualizzando il valore tramite un testo a schermo e

spostando l'indicatore dello slider in relazione al valore stesso. Successivamente, lascia il controllo delle dimensioni allo slider, che può essere interagito tramite controller, mani o raggio. Grazie alla logica implementata nel controller del sottosistema, il personaggio selezionato cambia le sue dimensioni in base al valore assunto dallo slider, modificandone la scala.

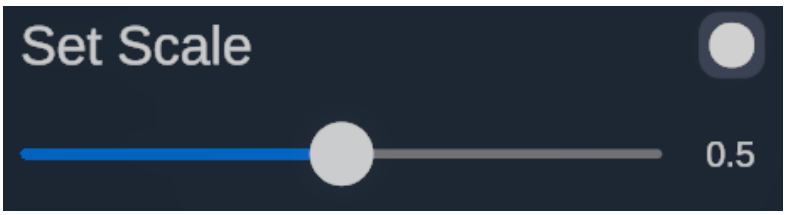

Figure 3.13: Slider a comparsa per regolare liberamente la dimensione del personaggio

3.7 Sistema di Posing basato su Animazioni Predefinite

Oltre all'analisi delle diverse modalità di cinematica adottabili per il posing di un personaggio virtuale, un ulteriore confronto proposto in questo elaborato riguarda l'utilizzo di un sistema di posing basato su animazioni preesistenti associate ai personaggi in scena. Questo approccio si ispira alla modalità già presente in un applicativo di storyboard in realtà virtuale, al quale il sistema sviluppato in questa tesi fa riferimento. Il sistema di posing in questione permette di selezionare un personaggio in scena e, tramite un pannello interattivo, scegliere un'animazione predefinita da riprodurre. Il controllo dell'animazione è limitato esclusivamente alle funzioni di **"Play & Stop"**. A partire da questa base preesistente, una parte secondaria del lavoro di tesi si è concentrata sul **porting** di tale sistema in un applicativo stand-alone per Meta, ad affiancare il sistema basato su cinematica presentato in questa tesi. Questo processo ha richiesto la conversione dell'intero sistema di gestione delle animazioni, adattandolo ai componenti e alle interfacce fornite dal Meta SDK. Inoltre, è stata introdotta una nuova funzionalità non presente nella versione originale. Di seguito, si propone un approfondimento sul sistema di posing realizzato, con particolare attenzione al controller principale e all'interfaccia grafica.

3.7.1 Sistema di Controllo delle Animazioni

Come in ogni sistema di controllo, la gestione della logica di animazione dei personaggi è stata affidata a uno script dedicato, denominato `AnimationController`. Questo script è assegnato come componente a un GameObject all'interno della scena Unity. Il controller del sistema rimane in attesa che un personaggio animato venga selezionato. Per essere riconosciuto come tale, il personaggio deve avere associato al proprio GameObject il componente *Animator*, responsabile della gestione del ciclo di animazioni proiettate sull'avatar del personaggio. Fino alla selezione, l'outline del personaggio rimane attivo, segnalando all'utente la possibilità di interagire con esso. Le modalità di interazione, conformi ai principi esposti in precedenza, prevedono l'utilizzo di controller, mani e raggio originato da loro. Alla selezione di un personaggio, il GameObject al suo interno, denominato *InteractableComponent*, si occupa di eseguire la funzione `GetCharacterAnimations(Transform character)`. Questa funzione, implementata all'interno dello script `AnimationController`, ha il compito di inizializzare le variabili necessarie per il controllo del personaggio selezionato, il cui riferimento viene passato come parametro alla funzione stessa. L'interazione con il personaggio avviene tramite un'apposita interfaccia utente (UI), realizzata mediante un Canvas interattivo. Attraverso questa UI, l'utente può accedere a diverse informazioni relative al personaggio, tra cui: Nome, Attivazione/disattivazione dell'outline di demarcazione, Lista delle animazioni associate. Le modalità di controllo dell'animazione offerte dall'`AnimationController` includono due approcci distinti, che possono essere utilizzati anche in modo complementare:

- Riproduzione e interruzione dell'animazione tramite i pulsanti "Play" e "Stop", accessibili dal pannello di controllo.
- Gestione della timeline dell'animazione in esecuzione, mediante uno slider interattivo. Quest'ultimo consente di selezionare manualmente un punto specifico della timeline, sia tramite interazione diretta con lo slider, sia attraverso i pulsanti laterali, che permettono di avanzare o arretrare di un valore predefinito.

Le due modalità coesistono, garantendo all'utente massima flessibilità nell'interazione: è possibile interrompere l'animazione in qualsiasi momento e scorrere la timeline per posizionare il personaggio nel punto esatto della sequenza animata in cui assume la posa desiderata. Di seguito viene riportato il menù di interazione.

Figure 3.14: Riproduzione dell'animazione di camminata di un personaggio controllato mediante sistema di posing basato su animazioni

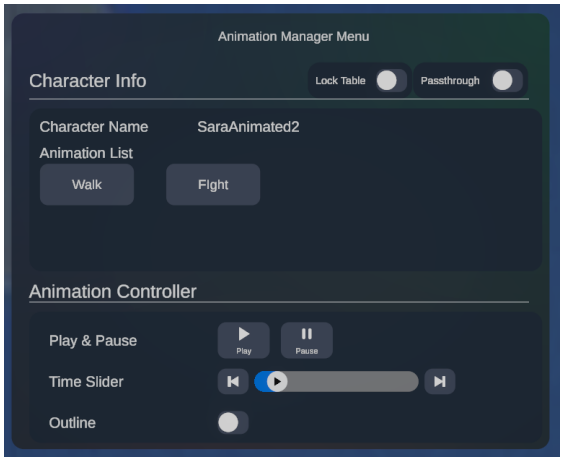

Figure 3.15: pannello di controllo centrale del siistema di posing basato su animazioni e creato impiegando componenti di Meta SDK

Chapter 4

Valutazione Sperimentale e Risultati

Il quarto capitolo ha l'obiettivo di illustrare il processo di test del sistema, condotto da utenti esperti, al fine di valutarne l'usabilità. In questo capitolo verranno chiariti gli obiettivi e le metriche di valutazione scelte, nonché il procedimento adottato per la conduzione dei test. Verrà presentato l'Use Case utilizzato durante il test, insieme ad una prima raccolta e analisi dei dati provenienti dalle valutazioni degli utenti. Inoltre, saranno evidenziati i punti di forza e le criticità emerse durante l'utilizzo del sistema. Infine, si fornirà una panoramica dei test futuri, che coinvolgeranno utenti generici e neofiti di realtà estesa e animazione.

4.1 Progettazione dei test

Il primo passo da compiere quando si parla di user test è definire le informazioni che si desidera estrapolare dagli utenti, scegliendo le metriche più adatte a catalogare i dati provenienti dai test. Questo permette di identificare se gli obiettivi prefissati sono stati raggiunti e quali criticità sono emerse. I test condotti sono stati limitati alla modalità di interazione con il sistema tramite **controller**, escludendo l'interazione tramite hand tracking. Tale scelta è stata dettata dalla complessità statistica nell'analizzare tutte le possibili configurazioni dei risultati derivanti da test che confrontano un sistema con diverse tecniche di cinematica (e non) per il posing. Inoltre, le due modalità di interazione (controller e hand tracking) comportano variabili che, se incluse, avrebbero reso il confronto troppo complesso, nonostante il sistema permetta l'interazione con gli oggetti anche tramite hand tracking.

4.1.1 Obiettivi

La prima fase di testing, i cui risultati saranno riportati in questo elaborato di tesi, ha come obiettivo principale la valutazione dell'usabilità generale del sistema proposto e della UI per interfacciarsi con essa, seguendo i principi espressi da Jakob Nielsen in [35]. Nel paper viene esplicitato che, per avere una valutazione consistente dell'usabilità e interfaccia di un sistema, è ottimale sottoporre la valutazione a un certo numero di **domain experts**[1] (tra tre e cinque), poiché è dimostrato che questa quantità di esperti è sufficiente a rilevare il 70-80% dei problemi di usabilità di un sistema e dare un contributo sostanziale alla sua valutazione e validazione. L'obiettivo di questo primo test è individuare eventuali correzioni da apportare ai problemi riscontrati e possibili migliorie. Il test viene condotto sia sul sistema di posing basato sull'impiego della cinematica, che sul sistema di posing basato su animazioni, ponendosi gli stessi obiettivi valutativi. La sperimentazione condotta seguirà quindi il medoto dell'*heuristic evaluation*.

4.1.2 Metriche scelte

Le metriche scelte per l'*heuristic evaluation* proposta da Nielsen e sottoposta agli esperti di dominio individuati si basano sui dieci principi euristici di usabilità, che vengono di seguito riportati:

- **Visibilità dello stato del sistema.**
- **Corrispondenza tra il sistema e il mondo reale.**
- **Coerenza e standard.**
- **Prevenzione degli errori.**
- **Riconoscimento anziché richiamata.**
- **Flessibilità ed efficienza d'uso.**
- **Design estetico e minimalista.**
- **Aiutare gli utenti a riconoscere, diagnosticare e recuperare dagli errori.**
- **Aiuto e documentazione.**

[1] Persona con conoscenze approfondite e specializzate in un determinato campo o settore, che possiede esperienza sia pratica che teorica.

4.2 Conduzione del Test

I test sono stati condotti nell'arco di una giornata, durante la quale agli esperti di dominio è stata fornita una panoramica generale del test che avrebbero svolto e delle sue finalità, insieme a un breve tutorial illustrativo sull'usabilità del sistema specifico. Successivamente, sono stati chiamati a svolgere un task utilizzando sia la metodologia di posing basata su animazioni che quella che si basa sul controllo della cinematica, sistema sviluppato in questo lavoro di tesi. Al termine del test, è stato somministrato un apposito questionario per raccogliere i dati necessari alla valutazione dell'usabilità.

4.2.1 Campione scelto

Sono stati scelti degli esperti di dominio per essere sottoposti al test, nello specifico dottorandi e ricercatori provenienti da laboratori di ricerca che si occupano di Computer Grafica e Realtà Estesa.

4.2.2 Tutorial

Il tutorial proposto ai tester è strutturato in tre parti, finalizzate a far acquisire confidenza con l'ambiente di lavoro e a illustrare le funzionalità offerte dai sistemi e il loro funzionamento. Nello specifico:

- Meta Ambient Tutorial: Sezione del tutorial che introduce le modalità di interazione con gli oggetti in scena tramite controller o raggio emesso da esso. In particolare, illustra l'interazione con il Canvas, gli elementi grafici, la funzione di Passthrough, il tavolo di lavoro virtuale e gli oggetti presenti nella scena (escludendo quelli animati o manipolabili tramite cinematica diretta o inversa).

- Animation Mode Tutorial: Sezione del tutorial che spiega il funzionamento del sistema di posing basato sulle animazioni, illustrando l'uso del menu di selezione delle funzioni e mostrando una demo interattiva con un personaggio.

- Kinematic Mode Tutorial: Sezione del tutorial che spiega il funzionamento del sistema di posing basato sulla cinematica, illustrando l'uso del menu di selezione delle funzioni e mostrando una demo interattiva con un personaggio.

4.2.3 Use Case

Lo Use Case somministrato consiste in una parte di script ispirata al copione originale del film d'animazione *Toy Story* [36]. Questo funge da canovaccio per il

test, indicando le azioni e le pose da ricreare fedelmente. Di seguito si riporta il testo dello script:

Scene 1: Woody sits on the edge of the bed observing all the activity. A plastic army man, SARGENT, is walking across the nightstand.

Scene 2: Woody confidently proceeds in his direction. He passes a toy ROBOT and SNAKE partially hidden under the bed, near one of his legs. Robot and Snake come out from under the bed and reluctantly follow Woody.

Scene 3: Woody spots the doodle pad on the floor and walks over to it. As he reaches down to pick it up... REX, the plastic dinosaur, jumps out to scare Woody.

Scene 4: Woody and Buzz jump back in fright, taking a fighting stance. Buzz points his hands at Woody's forehead with a fighting stance, while Woody gets into position.

4.2.4 Questionario finale

Il questionario finale, realizzato tramite la piattaforma Google Moduli, è organizzato in un'unica sezione che raccoglie tutte le domande da somministrare ai tester, oltre a una sezione dedicata ai commenti liberi. In questa casistica, non è ritenuto rilevante raccogliere dati anagrafici. Il questionario include un totale di 10 requisiti da validare, per i quali i partecipanti sono chiamati a esprimere la propria opinione e valutazione attraverso la scala Likert, una scala ordinale di misurazione che consente di indicare il grado di accordo o disaccordo con determinate affermazioni. I 10 requisiti da valutare riflettono le metriche precedentemente descritte e, per maggiore chiarezza, ogni metrica è accompagnata da una breve spiegazione del suo significato. I requisiti richiesti nel questionario sono i seguenti:

- **Visibility of system status:** The system should always keep users informed about what is going on through appropriate feedback within reasonable time.
- **Match between system and the real world:** The system should use language and concepts familiar to the user, rather than system-oriented terms.
- **User control and freedom:** Users should have the ability to undo and redo actions, as well as the option to exit unwanted states.
- **Consistency and standards:** he system should follow established conventions and provide a consistent interface, making it easier for users to understand.

- **Error prevention:** The system should be designed to prevent errors from occurring in the first place, or offer clear instructions to help users avoid them.
- **Recognition rather than recall:** Minimize the user's memory load by making options, actions, and information visible or easily accessible.
- **Flexibility and efficiency of use:** The system should be adaptable to both novice and expert users, offering shortcuts or accelerators for advanced users while remaining easy for beginners.
- **Aesthetic and minimalist design:** The design should avoid unnecessary elements that do not add value, focusing only on the essential information or controls.
- **Help users recognize, diagnose, and recover from errors:** Error messages should be clear, providing constructive feedback and guiding users to correct the problem I was able to complete the task without unnecessary interruptions or confusion.
- **Help and documentation:** While it's better if the system is usable without documentation, it's important to provide helpful and easily accessible assistance if needed.

Come domanda aggiuntiva per raccogliere eventuali commenti liberi, è stata inserita la seguente: **"Do you have any additional comments or suggestions regarding the usability of the system?"**. Per motivi di accessibilità il questionario è posto in inglese, con la possibilità di rispondere anche in italiano.

4.2.5 Analisi dei risultati

I primi test condotti con esperti di dominio, utilizzando le Euristiche di Usabilità di Nielsen come criterio di valutazione, hanno riportato risultati generalmente positivi. I feedback raccolti evidenziano un buon livello di soddisfazione per l'impostazione che possiedono entrambi i sistemi di messa in posa, seppur rilevando delle migliorie sulle interfacce di gestione del singolo sistema. Diversi sono stati i commenti e i suggerimenti, fedeli ai punteggi ottenuti raccogliendo i valori alla fien dei questionari. In particolare, il focus è stato posto gli **elementi grafici** all'interno delle interfacce di controllo. In determinati casi alcuni elementi non sono perfettamente dimensionati e creano difficoltà alle vista e alla precisione di input per interagirvi, portando a generare comportamenti non desiderati; inoltre si dovrebbe rivedere qualche terminologia applicata. Consigliato vivamente anche un maggior **coinvolgimento** e **sfruttamento** di tutti gli input di cui dispongono i controller, ad esempio con l'aggiunta di **shortcut** per la selezione delle funzioni

disponibili o impiegando la pulsantiera per la navigazione dei menù. Un aspetto particolarmente rilevante riguarda il processo di posing: l'utilizzo esclusivo della cinematica diretta è stato percepito come più **macchinoso** e lento rispetto alla cinematica inversa o di animazioni predefinite. Queste ultime sono risultate, da una prima analisi, il metodo più rapido per mettere in posa, seppur rimanendo un metodo limitato alle animazioni predefinite. Suggerito un approccio **ibrido**, che combini l'uso delle animazioni per definire una base di posa, affinata successivamente mediante cinematica diretta per il perfezionamento dei dettagli. Infine, un ulteriore suggerimento di rilievo riguarda un'**alternativa** alla logica di rotazione degli handle in FK attuale. È stato proposto di sganciare tale logica dal Gimbal, consentendo agli utenti di sfruttare direttamente la rotazione del controller per pilotare il comando dell'handle selezionato. Analogamente, per i target della cinematica inversa, è stata suggerita una maggiore libertà di manipolazione, includendo la possibilità di ampliare il loro grado di rotazione oltre alla semplice traslazione. Questo miglioramento potrebbe offrire un controllo più intuitivo ed efficace, aumentando la flessibilità del sistema. Prima di procedere con la somministrazione di uno user test su misura per utenti generici, è consigliato raccogliere ulteriori feedback e valutazioni da parte di altri domain expert, per avere certezza che quanti più issues nell'usabilità vengano scovati e, possibilmente, corretti.

4.2.6 Ampliamento dei test

La seconda fase di testing amplia lo studio e gli obiettivi prefissati, introducendo l'utilizzo di metriche differenti per una valutazione più approfondita. In questa fase, il **campione** analizzato è costituito da utenti con conoscenze nel campo della computer grafica e dell'animazione, ma non necessariamente esperti in tecnologie di realtà estesa. Si tratta potenzialmente di studenti universitari del Politecnico di Torino provenienti da corsi di studio specifici di settore. Gli **obiettivi** di questa seconda fase sono molteplici e mirano a valutare diversi aspetti dell'esperienza utente, tra cui

- Usabilità dell'applicazione e dell'interfaccia utente
- Misura delle prestazioni del'appliczione relativamente al tempo richiesto per completare un task, accuratezza del risultato finale, errori commessi durante l'interazione, facilità d'uso e grado di intuitività del sistema.
- Utilità del sistema nell'ambito del task di posing di più personaggi in scena,

Lo **svolgimento** del test sarà diverso rispetto alla conduzione effettuata nel caso dei domain expert, in quanto per questa seconda fase di test, gli user saranno divisi in due macrogruppi, in ognuno dei quali ogni user che ne fa parte sarà chaiamto a svolere il task di posing (seguqneod lo script proposto in precedenza) ma potengo

impiegare esclusivamente una modalità di posing, ovvero escsivamente tramite cinematica o animazioni. In qeusto modo possiamo più facilemtne confrontare statisticamente i risultati che si ottengono. le modalità di interazione con il sistema rimangono invariate. Per ottenere risultati confrontabili, al termine del test verrà somministrato un questionario **SUS** (System Usability Scale), uno strumento standardizzato per la valutazione dell'usabilità.Il questionario è composto da 10 affermazioni, a cui gli utenti rispondono utilizzando una scala Likert a 5 punti, che va da 1 (fortemente in disaccordo) a 5 (fortemente d'accordo). Le affermazioni sono formulate in modo alternato, includendo sia enunciati positivi che negativi, al fine di ridurre il rischio di risposte poco riflessive. I dati raccolti vengono successivamente elaborati secondo la seguente formula di calcolo:

- Per le affermazioni dispari, si sottrae 1 al valore della risposta.
- Per le affermazioni pari, si sottrae il valore della risposta da 5.
- La somma di tutti i punteggi ottenuti viene moltiplicata per 2,5, in modo da ottenere un punteggio normalizzato su una scala da 0 a 100.

Il punteggio ottenuto è un valore nel range 0-100 da non confondere con una percentuale o un percentile: Il punteggio medio del SUS è pari a 68 corrispondente al 50º percentile. Un punteggio superiore a 68 indica un livello di usabilità buono; inferiore suggerisce delle criticità. L'analisi comparativa dei punteggi SUS ottenuti consentirà di elaborare le conclusioni finali e di rispondere in modo mirato agli obiettivi prefissati.

Chapter 5

Conclusione

Il progetto presentato in questo lavoro di tesi ha introdotto un nuovo approccio nella risoluzione del task di posing per la realizzazione di uno storyboard virtuale, sfruttando tecnologie avanzate di Realtà Estesa. Il sistema sviluppato consente la manipolazione di personaggi virtuali tramite l'applicazione delle tecniche di Cinematica Diretta e Inversa, permettendo all'utente di controllare liberamente e con alta precisione le pose dei personaggi, passando liberamente da una tipologia di cinematica all'altra liberamente. Oltre alla sua funzione principale, il sistema offre strumenti aggiuntivi che ne arricchiscono l'uso, come la possibilità di traslare, ruotare o dimensionare il personaggio, oltre alla gestione delle pose attraverso operazioni di salvataggio, rollback e reset. Queste caratteristiche ampliano notevolmente la flessibilità del sistema, consentendo di adattarlo a esigenze specifiche e di gestire il processo creativo in modo dinamico. Il sistema è stato quindi confrontato con approcci esistenti, in particolare con quelli basati sul "Play & Stop" di animazioni predefinite. Mentre questi approcci possono risultare più rapidi, limitano la libertà dell'utente, vincolando le pose alle animazioni preconfigurate e alla loro durata, senza possibilità di personalizzazione. Al contrario, il sistema proposto mira a dare all'utente una maggiore libertà creativa, consentendo la definizione di pose personalizzate senza le restrizioni imposte da animazioni fisse. La sfida principale del progetto è stata quella di sviluppare uno strumento intuitivo e versatile, capace di supportare professionisti e creativi nella realizzazione di pose complesse senza imporre limiti alla creatività dell'utente. Grazie a queste caratteristiche, il sistema si propone come un valido strumento per lo storyboarding virtuale, offrendo un controllo preciso e flessibile sulla costruzione delle scene e sulla definizione delle pose dei personaggi.

5.1 Sviluppi Futuri

Il sistema così sviluppato offre ampi margini di espansione, sia per il miglioramento delle funzionalità attuali, sia per l'integrazione di nuove tecniche di posing. L'integrazione di un supporto AI non è da escludere in quanto potrebbe velocizzare e semplificare lo svolgimento dei task che si vogliono compiere o fornire supporto in sede di riconoscimento delle catene utili. Infine, il sistema potrebbe evolversi verso un'esperienza collaborativa o arricchirsi con altre tecnologie di supporto all'interno dell'ambiente in Realtà Estesa, le quali espandono il parco di possibilità e funzionalità che è possibile realizzare e integrare all'interno del sistema.

5.1.1 Sviluppi legati alla gestione delle pose

Attualmente, il sistema consente di salvare temporaneamente una sola posa alla volta, che può essere sovrascritta in qualsiasi momento senza mantenere uno storico delle pose precedenti. Di conseguenza, eventuali pose salvate inizialmente e successivamente modificate non sono più recuperabili. Un'evoluzione interessante del sistema potrebbe essere l'introduzione di una **Pose List**, che contenga tutte le pose salvate durante la sessione, permettendo agli utenti di nominarle in modo personalizzato e di accedervi facilmente per riutilizzarle. Un'ulteriore estensione di questa funzionalità potrebbe essere l'integrazione di un database dedicato (o una modalità di storage dei dati utile), capace di archiviare le informazioni relative ai giunti e agli end effector. In questo modo, le pose salvate potrebbero essere recuperate anche tra sessioni diverse di utilizzo del sistema di posing, garantendo così una persistenza nel tempo.

5.1.2 Sviluppi legati all'impiego dell'AI

Essendo sempre più pervasiva nelle tecnologie con cui interagiamo quotidianamente, si può pensare di introdurre un supporto basato sull'intelligenza artificiale al sistema, per apportare diversi benefici. Di seguito sono riportate alcune possibili applicazioni individuate.

Autoposing prima delle manipolazioni

Creare una posa complessa da zero, soprattutto partendo da una posa standard come la classica "T Pose[1]" o "A Pose[2]", può risultare un processo lungo, soprattutto se si utilizza la tecnica di Cinematica Diretta. Un possibile supporto a questo compito potrebbe essere fornito da un'intelligenza artificiale (AI) che, essendo stata addestrata appositamente su una serie di pose standard applicabili in diversi contesti, potrebbe fornire una 'base' da cui partire per creare la posa desiderata. A partire dai dati relativi a armature deformate per assumere una determinata posa, l'AI potrebbe mappare queste informazioni sull'armatura del personaggio e autocalcolare i valori da attribuire ai giunti dell'armatura, creando così una base modificabile che si avvicina molto alla posa finale. In questo modo, il tempo impiegato per creare la posa si ridurrebbe drasticamente, poiché l'utente avrebbe già una base da cui partire, che potrà personalizzare secondo il proprio gusto e la propria creatività.

No precondizioni su modello (AI)

Il funzionamento dell'intero sistema si basa sull'applicazione di specifiche condizioni iniziali che devono essere rispettate dai modellatori durante la fase di rigging, al fine di garantire che, successivamente, in Unity, le catene cinematiche siano correttamente riconosciute. Queste condizioni sono fondamentali per il corretto funzionamento dell'applicazione. Per eliminare questa imposizione iniziale, potrebbe essere introdotto un supporto basato su intelligenza artificiale. Un sistema AI potrebbe, infatti, analizzando l'FBX del modello, individuare automaticamente i rami dello scheletro del personaggio che rappresentano possibili catene cinematiche, senza la necessità di utilizzare "tag" nei nomi delle ossa. Successivamente, l'AI potrebbe notificare tali catene nel sistema e, seguendo la logica definita, creare automaticamente le istanze della classe BoneChain, assegnando correttamente le ossa utili individuate. Questa soluzione permetterebbe maggiore libertà durante le fasi di modellazione e rigging, evitando errori umani come sbagli di battitura o omissioni nei nomi delle ossa, riducendo inoltre il lavoro aggiuntivo che i modellatori dovrebbero svolgere prima dell'implementazione finale.

[1]Posa standard utilizzata nel rigging in cui un modello ha le braccia estese lateralmente, formando una "T".

[2]Posa standard utilizzata nel rigging in cui un modello ha le braccia leggermente distese, ma con un angolo verso il basso, formando una "A" maiuscola.

5.1.3 Sviluppi legati ai modelli

Attualmente, lo script generatore del file XML legato al modello creato in Blender include anche la descrizione dei possibili limiti alle rotazioni delle ossa dello scheletro del personaggio. Tuttavia, queste informazioni non vengono sfruttate dal sistema, poiché si è deciso di non imporre vincoli specifici, in particolare per consentire la creazione di pose più stravaganti in contesti surreali. Un possibile ampliamento delle funzionalità del sistema potrebbe essere l'introduzione di una condizione che permetta di decidere liberamente se applicare tali limiti alle rotazioni, mantenendo la coerenza con i limiti che uno scheletro reale dovrebbe avere. A tal fine, il componente **Rotation Limits**[**37**] fornito dal package Final IK (utilizzato per il componente FABRIK) potrebbe rivelarsi utile. Questo componente offre tre varianti, permettendo di applicare diversi tipi di limiti sulle rotazioni dei GameObject seguendo specifici angoli. Durante la creazione di un nuovo modello in Unity, definendo le catene cinematiche appropriate, si potrebbero leggere le informazioni contenute nell'XML per identificare i limiti delle ossa e, successivamente, applicare il giusto componente di tipo Rotation Limit alle relative BoneChains. L'utente potrebbe quindi attivare liberamente questi limiti durante la manipolazione del personaggio, a sua discrezione. In questo modo, si fornirebbe un ulteriore strumento utile per gli utenti che desiderano essere guidati verso un posing più realistico, seguendo i limiti naturali degli scheletri nel mondo reale.

5.1.4 Sviluppi legati alla tecnica di posing

Nel sistema attualmente progettato, l'utente agisce come un agente "passivo", manipolando liberamente le catene cinematiche del personaggio virtuale in base alla modalità scelta (cinematica diretta o inversa). Tuttavia, una possibile evoluzione interessante del sistema potrebbe essere quella di rendere l'utente un agente **attivo** nel processo di posing. Utilizzando l'infrastruttura di controllo delle catene cinematiche sviluppata in Unity, sarebbe possibile implementare un sottosistema che consenta all'utente di **impersonificare** il personaggio che desidera mettere in posa. In pratica, l'utente, emulando fisicamente la posa desiderata, permetterebbe al personaggio virtuale di replicare automaticamente la postura assunta. Questa funzionalità rappresenterebbe una naturale estensione dell'interazione, migliorando l'esperienza utente e facilitando la creazione di pose complesse. La possibilità di "impersonare" la posa aumenterebbe la fedeltà e l'automazione del processo di posing, consentendo una maggiore espressività creativa. Tuttavia, l'implementazione di tale funzionalità richiederebbe l'adozione di tecnologie avanzate, come il riconoscimento del movimento, per garantire un alto livello di precisione nel rilevamento delle posizioni e dei movimenti nello spazio.

5.1.5 Sviluppi legati alle tecnologie

Le tecnologie su cui si fonda il progetto di tesi appartengono alle più recenti innovazioni nel campo della Realtà Estesa. Tuttavia, tali tecnologie presentano alcuni limiti intrinseci relativi alle modalità di interazione e manipolazione degli oggetti virtuali. In particolare, un visore avanzato come il Meta Quest Pro offre diverse modalità di interazione con gli oggetti virtuali, ma queste sono limitate: l'interazione avviene principalmente tramite controller (dotati di una pulsantiera relativamente semplice), tramite hand tracking con il riconoscimento di pochi e limitati gesti, o infine tramite il raggio di interazione dai controller o dalle mani. Per superare queste limitazioni, si è pensato di integrare ulteriori tecnologie in grado di fornire al sistema informazioni aggiuntive, ampliando così le funzionalità e migliorando l'interazione con gli oggetti virtuali. Tecnologie come il Leap Motion Controller o i guanti aptici (Haptic Gloves) potrebbero arricchire il sistema permettendo di acquisire informazioni dettagliate sui giunti delle mani, con cui l'utente interagisce. Questi dispositivi non solo incrementerebbero il grado di fedeltà dell'interazione, ma potrebbero anche consentire nuove possibilità di controllo e manipolazione. In particolare, una delle evoluzioni future che potrebbe derivare dall'integrazione di queste tecnologie sarebbe la possibilità di mappare i giunti delle mani, rilevati tramite dispositivi come il Leap Motion o i guanti aptici, direttamente sui giunti delle catene cinematiche dei personaggi virtuali. In questo modo, l'utente potrebbe mettere in posa il personaggio semplicemente muovendo le proprie mani, nel caso di personaggi con scheletri semplici e facilmente mappabili.

5.1.6 Sviluppi legati alla collaborazione

Il sistema, così come è attualmente progettato, è pensato per l'uso da parte di un singolo utente, con un'interfaccia costituita da un unico Canvas contenente il menu con le operazioni disponibili. Tuttavia, un possibile sviluppo futuro potrebbe essere quello di ampliare l'utilizzo del sistema, permettendo a più utenti di collaborare sia localmente che, attraverso un'apposita infrastruttura da sviluppare, via web. In questo scenario, sarebbe possibile creare un ambiente condiviso in cui diversi utenti possano interagire simultaneamente, mettendo in posa modelli diversi. Ogni utente avrebbe il proprio spazio di lavoro e un menu personalizzato, al quale associare qualsiasi personaggio virtuale scelto, e potrà manipolare e posizionare il proprio modello liberamente. Questa configurazione permetterebbe la creazione di scene condivise complesse e articolate in modo collaborativo, favorendo il lavoro di gruppo nella realizzazione di pose per personaggi diversi, ma all'interno dello stesso progetto. Un tale approccio stimolerebbe un ambiente creativo di scambio di idee e ispirazione reciproca tra gli utenti, oltre a incentivare il supporto e l'aiuto reciproco.

Appendix A

Codice Sorgente

Di seguito si riporta il codice completo delle componenti e funzioni di maggior rilievo.

A.1 XML Generator

Il seguente codice Python permette di poter riavare un file in formato XML a partire dall'armatura di un modello riggato in Blender.

Codice A.1: Script `XMLGenerator.py` per ricavare l'XML da un modello

```
import bpy
import xml.etree.ElementTree as ET

def export_bone_data_to_xml(armature_object, file_path):

    root = ET.Element("ArmatureData")

    for bone in armature_object.pose.bones:
        bone_element = ET.SubElement(root, "Bone")

        ET.SubElement(bone_element, "Name").text = bone.name

        transform_element = ET.SubElement(bone_element, "Transform")

        loc_element = ET.SubElement(transform_element, "Location")
        loc_element.set("x", str(bone.location.x))
        loc_element.set("y", str(bone.location.y))
        loc_element.set("z", str(bone.location.z))

        rot_element = ET.SubElement(transform_element, "Rotation")
        rot_element.set("w", str(bone.rotation_quaternion.w))
        rot_element.set("x", str(bone.rotation_quaternion.x))
```

```
23          rot_element.set("y", str(bone.rotation_quaternion.y))
24          rot_element.set("z", str(bone.rotation_quaternion.z))
25
26          scale_element = ET.SubElement(transform_element, "Scale")
27          scale_element.set("x", str(bone.scale.x))
28          scale_element.set("y", str(bone.scale.y))
29          scale_element.set("z", str(bone.scale.z))
30
31          for constraint in bone.constraints:
32              constraint_element = ET.SubElement(bone_element, "
    Constraint", type=constraint.type)
33
34              if constraint.type == 'LIMIT_ROTATION':
35                  for axis in ['x', 'y', 'z']:
36                      axis_element = ET.SubElement(constraint_element,
    "Axis", value=axis)
37                      axis_element.set("min", str(getattr(constraint, f
    "min_{axis}")))
38                      axis_element.set("max", str(getattr(constraint, f
    "max_{axis}")))
39
40              elif constraint.type == 'LIMIT_LOCATION':
41                  for axis in ['x', 'y', 'z']:
42                      axis_element = ET.SubElement(constraint_element,
    "Axis", value=axis)
43                      axis_element.set("min", str(getattr(constraint, f
    "min_{axis}")))
44                      axis_element.set("max", str(getattr(constraint, f
    "max_{axis}")))
45
46              elif constraint.type == 'IK':
47                  target_element = ET.SubElement(constraint_element, "
    Target", name=constraint.target.name)
48                  chain_length_element = ET.SubElement(
    constraint_element, "ChainLength", value=str(constraint.
    chain_count))
49
50                  if constraint.pole_target:
51                      pole_target_element = ET.SubElement(
    constraint_element, "PoleTarget", name=constraint.pole_target.name
    )
52
53      tree = ET.ElementTree(root)
54
55      tree.write(file_path)
56
57      print(f"XML file saved to: {file_path}")
58
59  armature_obj = bpy.context.active_object
```

```
60
61 file_path = "E:\MetaQuestPosing\Assets\Resources\XML\ArmatureData.xml
       "
62
63 export_bone_data_to_xml(armature_obj, file_path)
```

A.2 KinematicChains

Il seguente codice, scritto in C#, definisce le tre classi che costituiscono le istanze di una nuova catena di ossa, catena di hadle per FK manipolabili e posizione assunta da un personaggio; rispettivamente le classi prendono il nome di `BonesChain`, `HandlesChain` e `BonesTransform`.

Codice A.2: Script `KinematicChains.cs` contenente le classi custom

```
1  using System.Collections.Generic;
2  using UnityEngine;
3
4  namespace KinematicChains
5  {
6      [System.Serializable]
7      public class BonesChain
8      {
9          public List<Transform> bones;
10         public List<string> boneNames;
11
12         public BonesChain()
13         {
14             bones = new List<Transform>();
15             boneNames = new List<string>();
16         }
17
18         public void AddBone(Transform handle)
19         {
20             bones.Add(handle);
21         }
22
23         public void AddBoneName(string boneName)
24         {
25             boneNames.Add(boneName);
26         }
27
28         public Transform GetBone(int index)
29         {
30             if (index >= 0 && index < bones.Count)
31             {
32                 return bones[index];
```

```
33              }
34              else
35              {
36                  Debug.LogError("Index out of range.");
37                  return null;
38              }
39          }
40
41          public string GetBoneName(int index)
42          {
43              if (index >= 0 && index < boneNames.Count)
44              {
45                  return boneNames[index];
46              }
47              else
48              {
49                  Debug.LogError("Index out of range.");
50                  return null;
51              }
52          }
53
54          public List<Transform> GetBonesList
55          {
56              get { return bones; }
57          }
58
59          public List<string> GetBoneNamesList
60          {
61              get { return boneNames; }
62          }
63
64          public int BoneCount
65          {
66              get { return bones.Count; }
67          }
68
69          public int BoneNameCount
70          {
71              get { return boneNames.Count; }
72          }
73
74      }
75
76      [System.Serializable]
77      public class HandlesChain
78      {
79          public List<Transform> handles;
80          public List<string> handleNames;
81
```

```
82          public HandlesChain()
83          {
84              handles = new List<Transform>();
85              handleNames = new List<string>();
86          }
87
88          public void AddHandle(Transform handle)
89          {
90              handles.Add(handle);
91          }
92
93          public void AddHandleName(string handleName)
94          {
95              handleNames.Add(handleName);
96          }
97
98          public Transform GetHandle(int index)
99          {
100             if (index >= 0 && index < handles.Count)
101             {
102                 return handles[index];
103             }
104             else
105             {
106                 Debug.LogError("Index out of range.");
107                 return null;
108             }
109         }
110
111         public string GetHandleName(int index)
112         {
113             if (index >= 0 && index < handleNames.Count)
114             {
115                 return handleNames[index];
116             }
117             else
118             {
119                 Debug.LogError("Index out of range.");
120                 return null;
121             }
122         }
123
124
125         public List<Transform> GetHandlesList
126         {
127             get { return handles; }
128         }
129
130         public List<string> GetHandleNameList
```

```
131         {
132             get { return handleNames; }
133         }
134 
135         public int HandleCount
136         {
137             get { return handles.Count; }
138         }
139 
140         public int HandleNamesCount
141         {
142             get { return handleNames.Count; }
143         }
144 
145 
146     }
147 
148     [System.Serializable]
149     public class BonesTransform
150     {
151         public List<Vector3> positions;
152         public List<Quaternion> rotations;
153 
154         public BonesTransform()
155         {
156             positions = new List<Vector3>();
157             rotations = new List<Quaternion>();
158         }
159 
160         public void AddPosition(Vector3 position)
161         {
162             positions.Add(position);
163         }
164 
165         public void AddRotation(Quaternion rotation)
166         {
167             rotations.Add(rotation);
168         }
169 
170         public Vector3 GetPosition(int index)
171         {
172             if (index >= 0 && index < positions.Count)
173             {
174                 return positions[index];
175             }
176             else
177             {
178                 Debug.LogError("Index out of range.");
179                 return new Vector3();
```

```
180             }
181         }
182 
183         public Quaternion GetRotation(int index)
184         {
185             if (index >= 0 && index < rotations.Count)
186             {
187                 return rotations[index];
188             }
189             else
190             {
191                 Debug.LogError("Index out of range.");
192                 return new Quaternion();
193             }
194         }
195 
196 
197         public List<Vector3> GetPositionsList
198         {
199             get { return positions; }
200         }
201 
202         public List<Quaternion> GetRotationsList
203         {
204             get { return rotations; }
205         }
206 
207         public int PositionsCount
208         {
209             get { return positions.Count; }
210         }
211 
212         public int RotationsCount
213         {
214             get { return rotations.Count; }
215         }
216 
217 
218     }
219 
220 
221 }
```

A.3 FindBoneSequence

Il seguente codice, scritto in C#, rappresenta le due funzioni che leggono l'XML relativo a un determinato personaggio e, associandogli il rispettivo file FBX,

procedono a istanziare le `BonesChain` del personaggio.

Codice A.3: Script `FindBoneSequence.cs` che crea dlle `BonesChain` dall'FBX dl modello

```
public void FindBoneSequence(XmlNode rootNode, GameObject
    characterFBXSkeleton, GeneralCharacterManager characterManager)
{
    bool isInSequence = false;

    BonesChain possibileBoneChain = new();
    BonesChain finalBoneChain = new();
    BonesChain combinedChain;

    bool newNestedChain = false;

    foreach (XmlNode characterBoneNode in rootNode.SelectNodes("Bone
    "))
    {
        string boneName = characterBoneNode.SelectSingleNode("Name")
    .InnerText;

        if (boneName.Contains("[ROOT]"))
        {
            possibileBoneChain = new();
            isInSequence = true;
        }

        if (isInSequence)
        {
            Transform boneTransform = FindChildRecursively(
    characterFBXSkeleton.transform, boneName);

            if (boneName.Contains("[ROOT]"))
            {
                boneTransform.name = boneTransform.name.Replace("[
    ROOT]", "");
                boneTransform.name.TrimEnd();

                possibileBoneChain.AddBone(boneTransform);
                possibileBoneChain.AddBoneName(boneTransform.name);
            }
            else if (boneName.Contains("[LEAF]"))
            {
                boneTransform.name = boneTransform.name.Replace("[
    LEAF]", "");
                boneTransform.name.TrimEnd();

                if (newNestedChain)
                {
```

```
40                      finalBoneChain.AddBone(boneTransform);
41                      finalBoneChain.AddBoneName(boneTransform.name);
42                  }
43                  else if (!newNestedChain)
44                  {
45                      possibileBoneChain.AddBone(boneTransform);
46                      possibileBoneChain.AddBoneName(boneTransform.
    name);
47                  }
48              }
49              else if (boneName.Contains("[END EFFECTOR]"))
50              {
51                  boneTransform.name = boneTransform.name.Replace("[
    END EFFECTOR]", "");
52                  boneTransform.name.TrimEnd();
53
54                  characterManager.CharacterIKEndEffectors.Add(
    boneTransform);
55                  characterManager.CharacterIKTargetNames.Add(
    boneTransform.name);
56
57                  if (newNestedChain)
58                  {
59                      combinedChain = new();
60                      combinedChain.bones = new List<Transform>(
    possibileBoneChain.bones);
61                      combinedChain.boneNames = new List<string>(
    possibileBoneChain.boneNames);
62
63                      combinedChain.bones.AddRange(finalBoneChain.
    bones);
64                      combinedChain.boneNames.AddRange(finalBoneChain.
    boneNames);
65
66                      characterManager.CharacterBoneChains.Add(
    combinedChain);
67
68                      finalBoneChain = new();
69
70                      newNestedChain = false;
71
72                      isInSequence = true;
73                  }
74                  else if (!newNestedChain)
75                  {
76                      characterManager.CharacterBoneChains.Add(
    possibileBoneChain);
77
78                      isInSequence = false;
```

```
79                      }
80                  }
81                  else if (boneName.Contains("[CHAIN]"))
82                  {
83                      boneTransform.name = boneTransform.name.Replace("[
    CHAIN]", "").TrimEnd();
84
85                      finalBoneChain.AddBone(boneTransform);
86                      finalBoneChain.AddBoneName(boneTransform.name);
87
88                      newNestedChain = true;
89                  }
90                  else if (boneName.Contains("[BONE]"))
91                  {
92                      boneTransform.name = boneTransform.name.Replace("[
    BONE]", "").TrimEnd();
93
94                      if (newNestedChain)
95                      {
96                          finalBoneChain.AddBone(boneTransform);
97                          finalBoneChain.AddBoneName(boneTransform.name);
98                      }
99                      else if (!newNestedChain)
100                     {
101                         possibileBoneChain.AddBone(boneTransform);
102                         possibileBoneChain.AddBoneName(boneTransform.
    name);
103                     }
104                 }
105             }
106         }
107 }
108
109
110 Transform FindChildRecursively(Transform parent, string name)
111 {
112     foreach (Transform child in parent)
113     {
114         if (child.name == name)
115         {
116             return child;
117         }
118
119         Transform found = FindChildRecursively(child, name);
120         if (found != null)
121         {
122             return found;
123         }
124     }
```

```
125         return null;
126     }
```

A.4 SetUpCharacterSettings

Il seguente codice, scritto in C#, mostra le tre funzioni cardine nel passaggio di dati da descrittore a controller centrale nel momento in cui viene selezionato un nuovo personaggio da manipolare.

Codice A.4: Script `SetUpCharacterSettings.cs` che raccoglie le funzioni base eseguite alla selezione di un personaggio

```
1  private void SetUpCharacterKinematicManager()
2  {
3      _character = characterManager.Character;
4      _characterBody = characterManager.CharacterBody;
5      _characterKeyPoints = characterManager.CharacterKeyPoints;
6
7      _characterName = characterManager.CharacterName;
8      _characterBones = characterManager.CharacterBoneChains;
9
10     _characterEndEffectors = characterManager.CharacterIKEndEffectors
       ;
11
12     _FKHandleChains = characterManager.FKHandleChains;
13
14     _characterKinematicChains = characterManager.IKFABRIKChains;
15     _IKTargets = characterManager.IKTargets;
16
17     _characterSkinnedMeshRenderer = characterManager.
       CharacterSkinnedMeshRenderer;
18     _characterMeshCollider = characterManager.CharacterMeshCollider;
19
20
21     _startBoneTransforms = characterManager.StartBoneTransforms;
22     _savedBoneTransforms = characterManager.SavedBoneTransforms;
23     _lastBoneTransforms = characterManager.LastBoneTransforms;
24
25     _startIKTargetPositions = characterManager.StartIKTargetPositions
       ;
26     _savedIKTargetPositions = characterManager.SavedIKTargetPositions
       ;
27     _lastIKTargetPositions = characterManager.LastIKTargetPositions;
28
29     _characterRealTimeMesh = characterManager.RealTimeMesh;
30
```

```
31      _characterOriginalScale = characterManager.CharacterOriginalScale
        ;
32      _IKTargetsOriginalScale = characterManager.IKTargetsOriginalScale
        ;
33
34
35      _characterUpper = characterManager.CharacterUpper;
36      _characterCenter = characterManager.CharacterCenter;
37      _characterLower = characterManager.CharacterLower;
38
39      _IKTargetsNewScaleInCharacter = characterManager.
        IKTargetNewScaleInCharacter;
40      _FKHandlesNewScaleInCharacter = characterManager.
        FKHandlesNewScaleInCharacter;
41
42        chainsActivation = true;
43  }
44
45  private void SetUpCharacterTransformManager()
46  {
47
48      _transformSystemController.Character = _character;
49      _transformSystemController.CharacterBody = _characterBody.
        transform;
50      _transformSystemController.CharacterKeyPoints =
        _characterKeyPoints;
51      _transformSystemController.CharacterOriginalScale =
        _characterOriginalScale;
52
53      _transformSystemController.IsSystemActive = false;
54      _transformSystemController.transformNumber = -1;
55
56      _transformSystemController.CharacterUpper = _characterUpper;
57      _transformSystemController.CharacterCenter = _characterCenter;
58      _transformSystemController.CharacterLower = _characterLower;
59
60      _transformSystemController.SetUpTransformSystem();
61
62  }
63
64  private void SetFromGimbalHandleSliderValue()
65  {
66      _targetScaleMultiplier = 1.9f * _IKTargetsNewScaleInCharacter.x;
67      float normalizedScaleIK = (_IKTargets[0].localScale.x -
        _IKTargetsNewScaleInCharacter.x) / _targetScaleMultiplier + 0.5f;
68      _IKTargetSlider.GetComponentInChildren<Slider>().value = Mathf.
        Clamp01(normalizedScaleIK);
69
70      _handleScaleMultiplier = 1.9f * _FKHandlesNewScaleInCharacter.x;
```

```
71      float normalizedScaleFK = (_FKHandleChains[0].handles[0].
        localScale.x - _FKHandlesNewScaleInCharacter.x) /
        _handleScaleMultiplier + 0.5f;
72      _FKHandleSlider.GetComponentInChildren<Slider>().value = Mathf.
        Clamp01(normalizedScaleFK);
73
74      float normalizedScaleGimbal = (gimbalSystem.transform.localScale.
        x - _gimbalOriginalScale.x) / _gimbalScaleMultiplier + 0.5f;
75      _gimbalSlider.GetComponentInChildren<Slider>().value = Mathf.
        Clamp01(normalizedScaleGimbal);
76
77  }
```

A.5 BoneHandleMapping

Il seguente codice, scritto in C#, riporta la funzione che definisce il mapping tra le liste presenti nelle `BonesChain` e `HandlesChain`.

Codice A.5: Script `BoneHandleMapping.cs` che mappa ogni handle presente in una lista sull'osso presente nell'altra

```
1   private void HandleBoneMapCreation()
2   {
3       //Siccome n handles e n ossa sono uguali per ognicatena, uso uno
        dei due rimepimenti
4       _handlesBonesMap.Clear();
5       if (_FKHandleChains.Count == _characterBones.Count)
6       {
7           for (int i = 0; i < _FKHandleChains.Count; i++)
8           {
9               HandlesChain handleChain = _FKHandleChains[i];
10              BonesChain bonesChain = _characterBones[i];
11
12              if (handleChain.handles.Count == bonesChain.bones.Count)
13              {
14
15                  for (int j = 0; j < handleChain.handles.Count; j++)
16                  {
17                      Transform handle = handleChain.handles[j];
18                      Transform bone = bonesChain.bones[j];
19
20                      _handlesBonesMap.Add(handle, bone);
21                  }
22              }
23              else
24              {
25                  Debug.LogError($"Handles and Bones lists in chain {i
    } do not have the same length.");
```

```
26                  }
27              }
28
29              Debug.Log("Dictionary has been populated with handles and
    bones.");
30          }
31          else
32          {
33              Debug.LogError("The lists _FKHandleChains and
    _characterBones do not have the same number of chains.");
34          }
35
36
37  }
```

Bibliography

[1] Blender Foundation. *Features*. https://www.blender.org (cit. on pp. 1, 20).

[2] Autodesk Foundation. *Autodesk Maya Overview*. https://www.autodesk.com/it/products/maya/overview (cit. on p. 1).

[3] Wikipedia. *Presto (Animation Software)*. https://en.wikipedia.org/wiki/Presto_(animation_software) (cit. on p. 1).

[4] John Hart. «Chapter 4 - Basic Components and Principles of the Storyboard». In: *The Art of the Storyboard (Second Edition)*. Ed. by John Hart. Second Edition. Oxford: Focal Press, 2008, pp. 37–68. ISBN: 978-0-240-80960-1. DOI: https://doi.org/10.1016/B978-0-240-80960-1.50008-9. URL: https://www.sciencedirect.com/science/article/pii/B9780240809601500089 (cit. on p. 4).

[5] Jorge Angeles, Günter Hommel, and Peter Kovács. *Computational kinematics*. Vol. 28. Springer Science & Business Media, 2013 (cit. on p. 9).

[6] Reza N. Jazar. «Forward Kinematics». In: *Theory of Applied Robotics: Kinematics, Dynamics, and Control*. Cham: Springer International Publishing, 2022, pp. 225–311. ISBN: 978-3-030-93220-6. DOI: 10.1007/978-3-030-93220-6_5. URL: https://doi.org/10.1007/978-3-030-93220-6_5 (cit. on p. 9).

[7] Jacques Denavit and Richard S Hartenberg. *A kinematic notation for lower-pair mechanisms based on matrices*. American Society of Mechanical Engineers, 1955 (cit. on p. 9).

[8] Ken Shoemake. «Animating rotation with quaternion curves». In: *SIGGRAPH Comput. Graph.* 19.3 (July 1985), pp. 245–254. ISSN: 0097-8930. DOI: 10.1145/325165.325242. URL: https://doi.org/10.1145/325165.325242 (cit. on p. 10).

[9] K. Hirasawa, M. Ohbayashi, M. Koga, and M. Harada. «Forward propagation universal learning network». In: *Proceedings of International Conference on Neural Networks (ICNN'96)*. Vol. 1. 1996, 353–358 vol.1. DOI: 10.1109/ICNN.1996.548917 (cit. on p. 10).

[10] Marko B. Popovic and Matthew P. Bowers. «2 - Kinematics and Dynamics». In: *Biomechatronics*. Ed. by Marko B. Popovic. Academic Press, 2019, pp. 11–43. ISBN: 978-0-12-812939-5. DOI: https://doi.org/10.1016/B978-0-12-812939-5.00002-1. URL: https://www.sciencedirect.com/science/article/pii/B9780128129395000021 (cit. on p. 10).

[11] A. Aristidou, J. Lasenby, Y. Chrysanthou, and A. Shamir. «Inverse Kinematics Techniques in Computer Graphics: A Survey». In: *Computer Graphics Forum* 37.6 (2018), pp. 35–58. DOI: https://doi.org/10.1111/cgf.13310. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13310. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13310 (cit. on p. 11).

[12] Samuel R Buss. «Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods». In: *IEEE Journal of Robotics and Automation* 17.1-19 (2004), p. 16 (cit. on p. 11).

[13] Ben Kenwright. «Inverse Kinematics - Cyclic Coordinate Descent (CCD)». In: *J. Graph. Tools* 16 (2012), pp. 177–217. URL: https://api.semanticscholar.org/CorpusID:26498902 (cit. on p. 11).

[14] Andreas Aristidou and Joan Lasenby. «FABRIK: A fast, iterative solver for the Inverse Kinematics problem». In: *Graphical Models* 73.5 (2011), pp. 243–260. ISSN: 1524-0703. DOI: https://doi.org/10.1016/j.gmod.2011.05.003. URL: https://www.sciencedirect.com/science/article/pii/S1524070311000178 (cit. on p. 11).

[15] Jing Li, Ji-hang Cheng, Jing-yuan Shi, and Fei Huang. «Brief Introduction of Back Propagation (BP) Neural Network Algorithm and Its Improvement». In: *Advances in Computer Science and Information Engineering*. Ed. by David Jin and Sally Lin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 553–558. ISBN: 978-3-642-30223-7 (cit. on p. 11).

[16] Jean-Baptiste Bordier, Anthony Mirabile, Robin Courant, and Marc Christie. «Smart Motion Trails for Animating in VR». In: *2022 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*. 2022, pp. 64–72. DOI: 10.1109/AIVR56993.2022.00016 (cit. on p. 13).

[17] Quentin Galvane, I-Sheng Lin, Fernando Argelaguet, Tsai-Yen Li, and Marc Christie. «VR as a Content Creation Tool for Movie Previsualisation». In: *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. 2019, pp. 303–311. DOI: 10.1109/VR.2019.8798181 (cit. on p. 13).

[18] Alberto Cannavò, Congyi Zhang, Wenping Wang, and Fabrizio Lamberti. «Posing 3D characters in virtual reality through in-the-air sketches». In: *International Conference on Computer Animation and Social Agents*. Springer. 2020, pp. 51–61 (cit. on p. 14).

[19] Cheng Yao Wang, Qian Zhou, George Fitzmaurice, and Fraser Anderson. «VideoPoseVR: Authoring Virtual Reality Character Animations with Online Videos». In: *Proc. ACM Hum.-Comput. Interact.* 6.ISS (Nov. 2022). DOI: 10.1145/3567728. URL: https://doi.org/10.1145/3567728 (cit. on p. 15).

[20] Andreia Valente, Augusto Esteves, and Daniel Lopes. «From A-Pose to AR-Pose: Animating Characters in Mobile AR». In: *ACM SIGGRAPH 2021 Appy Hour*. SIGGRAPH '21. Virtual Event, USA: Association for Computing Machinery, 2021. ISBN: 9781450383585. DOI: 10.1145/3450415.3464401. URL: https://doi.org/10.1145/3450415.3464401 (cit. on p. 16).

[21] Ye Pan and Kenny Mitchell. «PoseMMR: A Collaborative Mixed Reality Authoring Tool for Character Animation». In: *2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*. 2020, pp. 758–759. DOI: 10.1109/VRW50115.2020.00230 (cit. on p. 18).

[22] Li Zhang, Weiping He, Huidong Bai, Qianyuan Zou, Shuxia Wang, and Mark Billinghurst. «A hybrid 2D–3D tangible interface combining a smartphone and controller for virtual reality». In: *Virtual Reality* 27.2 (2023), pp. 1273–1291 (cit. on p. 18).

[23] Mikko Kytö, Krupakar Dhinakaran, Aki Martikainen, and Perttu Hämäläinen. «Improving 3D Character Posing with a Gestural Interface». In: *IEEE Computer Graphics and Applications* 37.1 (2017), pp. 70–78. DOI: 10.1109/MCG.2015.117 (cit. on p. 19).

[24] Masaki Oshita, Yuta Senju, and Syun Morishige. «Character motion control interface with hand manipulation inspired by puppet mechanism». In: *Proceedings of the 12th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*. VRCAI '13. Hong Kong, Hong Kong: Association for Computing Machinery, 2013, pp. 131–138. ISBN: 9781450325905. DOI: 10.1145/2534329.2534360. URL: https://doi.org/10.1145/2534329.2534360 (cit. on p. 19).

[25] Unity Technologies. *Unity Overview*. https://unity.com (cit. on p. 21).

[26] Microsoft. *Visual Studio Overview*. https://visualstudio.microsoft.com/it/ (cit. on p. 22).

[27] Sketchfab. *Sketchfab Overview*. https://sketchfab.com (cit. on p. 23).

[28] Adobe. *Mixamo Overview*. https://www.mixamo.com (cit. on p. 23).

[29] RootMotion. *Final IK Overview*. http://www.root-motion.com/finalikdox/html/index.html (cit. on p. 23).

[30] Unity. *Outline Overview*. https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/script-Outline.html (cit. on p. 24).

[31] Microsoft. *MRTK Overview*. https://learn.microsoft.com/it-it/windows/mixed-reality/mrtk-unity/mrtk2/?view=mrtkunity-2022-05 (cit. on p. 24).

[32] Meta. *Meta SDK Overview*. https://developers.facebook.com/docs/ (cit. on p. 24).

[33] Meta. *Meta Overview*. https://about.meta.com/it/ (cit. on p. 24).

[34] Blender Foundation. *Rigify Overview*. https://docs.blender.org/manual/en/2.81/addons/rigging/rigify.html (cit. on p. 43).

[35] Jakob Nielsen and Rolf Molich. «Heuristic evaluation of user interfaces». In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '90. Seattle, Washington, USA: Association for Computing Machinery, 1990, pp. 249–256. ISBN: 0201509326. DOI: 10.1145/97243.97281. URL: https://doi.org/10.1145/97243.97281 (cit. on p. 50).

[36] Wikipedia. *Toy Story Page*. https://it.wikipedia.org/wiki/Toy_Story_-_Il_mondo_dei_giocattoli (cit. on p. 51).

[37] RootMotion. *Rotation Limits Overview*. http://www.root-motion.com/finalikdox/html/page14.html (cit. on p. 59).