POLITECNICO DI TORINO

Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Progettazione e sviluppo di un sistema per la gestione dati di componenti e di manichini per l'ergonomia in ambito aerospaziale

Relatore

Prof. Andrea SANNA

Candidato

Giuseppe VIRZÌ

Anno accademico 2020 - 2021

Sommario

La Realtà Virtuale è utilizzata in diversi ambiti, che spaziano dall'intrattenimento, alla progettazione e simulazione. Grazie allo sviluppo di questa tecnologia, l'industria aerospaziale ha iniziato a utilizzarla per migliorare la progettazione, lo sviluppo e la simulazione dei propri sistemi. In questo ambito, la Realtà Virtuale permette di abbattere diversi costi dei processi di progettazione e diminuire notevolmente i tempi di sviluppo. Infine, risulta fondamentale per l'analisi, per l'addestramento e per la formazione.

La società Thales Alenia Space Italia (TASI), grazie al COSE - COllaborative System Engineering - Centre di Torino, ha sviluppato una serie di applicazioni in Realtà Virtuale che sfruttano principalmente il motore grafico Unreal Engine con diversi scopi, tra cui l'ingegneria collaborativa e il design ergonomico dei propri moduli. Le applicazioni VR permettono di migliorare il processo di progettazione dei moduli spaziali tramite la creazione di ambienti virtuali che li rappresentano. Inoltre, grazie all'uso di manichini virtuali è possibile studiare l'ergonomia dei sistemi spaziali, abbattendo i costi e i tempi delle tecniche tradizionali. È importante che i manichini rispettino gli standard definiti dalla NASA nel documento GP 10017, con titolo Gateway Program Human System Requirements (HSR).

Il lavoro di tesi si pone due obiettivi: la progettazione e lo sviluppo di un plugin per il motore grafico Unreal Engine 4, che permetta la gestione di dati appartenenti a reali componenti in ambito aerospaziale, e la progettazione e lo sviluppo di manichini virtuali per il design ergonomico di moduli spaziali.

Il primo modulo della tesi nasce dall'esigenza di rendere più efficiente e intuitivo il sistema d'importazione ed esportazione di parametri ingegneristici dei vari componenti che compongono un modulo spaziale. È necessario che questi dati vengano associati ai corrispettivi modelli virtuali, posizionati all'interno degli ambienti. I dati sono generalmente raccolti in un foglio di calcolo, ottenibile tramite lo strumento OCDT (Open Concurrent Design Tool) messo a disposizione dall'ESA (European Space Agency).

Il plugin, progettato per risolvere le precedenti problematiche, permette la gestione dei dati tramite una semplice interfaccia grafica. Grazie a questo, i parametri sono visualizzati e gestibili direttamente dall'engine. Per la realizzazione

sono stati sviluppati degli script in Python, integrati nativamente nell'editor di Unreal Engine 4. Il motore infatti permette di gestire e creare nuove funzionalità dell'editor tramite degli script, mettendo a disposizione una serie di strumenti, librerie e API per interagire con esso.

Inizialmente, per lo sviluppo del modulo, è stato creato un componente di Unreal Engine, per aggiungere le funzionalità di gestione dati agli oggetti di scena. Quindi sono stati creati quattro script in Pyhon per l'aggiunzione del componente personalizzato, per l'importazione e per l'esportazione. Infine, si è creata un'interfaccia grafica del plugin, inserendo un bottone che apre un menu a tendina, nella barra degli strumenti del motore.

È stato verificato il funzionamento del plugin (importazione ed esportazione) con qualsiasi tipologia di dato e parametro. Inoltre sono stati condotti dei test soggettivi presso Thales, grazie all'aiuto di collaboratori interni, per verificare l'usabilità del software. I risultati del questionario proposto sono più che positivi.

Il secondo modulo permette il miglioramento degli attuali manichini utilizzati da TASI. Attualmente la società utilizza una serie di manichini sviluppati tramite lo strumento MakeHuman, modificati tramite Blender e poi resi compatibili con Unreal Engine 4. Questo processo presenta diverse criticità: non è scalabile, non è permessa la modifica dei modelli creati in precedenza e i manichini risultano poco realistici. Inoltre, sono presenti alcune incompatibilità con i sistemi dell'engine. Infine, un ulteriore problema è il cambiamento, avvenuto nel 2019, dello standard HSR della NASA, precedentemente citato. A causa di questo, gli attuali manichini non sono più conformi ad esso.

Di conseguenza, questa seconda parte del lavoro di tesi si concentra sulla progettazione e sviluppo di un sistema automatico per la creazione dei manichini direttamente all'interno di UE4, risolvendo le criticità presenti nella precedente soluzione. Vengono utilizzati come base i modelli del nuovo strumento MetaHuman di Unreal Engine, con la quale è possibile creare dei manichini altamente realistici e animabili facilmente grazie al nuovo sistema di rigging, Control Rig. Dopodiché, sfruttando l'Animation Blueprint di UE4 si è creato lo strumento per la generazione automatica dei modelli virtuali.

Per dimostrare il funzionamento e l'efficienza della nuova soluzione, si è scelto di realizzare un sottoinsieme delle misure antropometriche definite dallo standard NASA. In particolare, per ogni misura si è sviluppato uno specifico sistema, all'interno di una classe Animation Blueprint, che agisce sullo scheletro dei manichini, in modo da poter modificare il modello dell'umanoide ad esso associato.

Il risultato di questa seconda parte sono due manichini (1° e 99° percentile) che rappresentano i due percentili unisex definiti dal documento NASA, creati grazie allo strumento sviluppato. Per la verifica delle misure degli umanoidi, è stato realizzato ed utilizzato un sistema per la misurazione di distanze e circonferenze, completamente integrato all'interno di Unreal Engine 4. Infine, sfruttando il

Control Rig del motore grafico, si è simulata la messa in posa dei manichini tramite cinematica diretta e inversa. In questo modo è possibile riprodurre, a scopi ergonomici, le pose degli astronauti che utilizzeranno gli ambienti dei moduli spaziali.

Ringraziamenti

Desidero ringraziare il professore Andrea Sanna per il grosso sostegno ricevuto prima e durante la stesura di questa tesi. Un sentito grazie va anche a Lorenzo Rocci e Eugenio Sorbellini di Thales Alenia Space, per avermi guidato in questa importante fase del mio percorso accademico.

Ringrazio Marco, Emanuele, Alessandro e gli amici di sempre, per avermi fatto ridere ed essere stati d'aiuto per tutto questo tempo. Ringrazio Ciccio, amico e collega in questi due anni a Torino. Un grazie va anche agli amici dell'asse, fondamentali in questo periodo di pandemia. Grazie al mio amico e socio Luca, per il sostegno, per le risate e per la sua disponibilità in ogni momento dell'anno.

Uno speciale ringraziamento va a mio padre Michelangelo, mia sorella Marianna e mio fratello Davide, che nonostante tutto, continuano a supportarmi in tutto ciò che faccio.

Infine, il grazie più importante va ad Irene che, in questi lunghi ma brevi anni, mi ha sempre sostenuto ed aiutato ad essere una persona migliore.

Indice

Elenco delle tabelle xi

Elenco delle figure xii

Elenco degli acronimi xvii

I Panoramica generale 1

1 Introduzione 2
- 1.1 Realtà Virtuale . . . 2
  - 1.1.1 Dispositivi di input . . . 3
  - 1.1.2 Dispositivi di output . . . 4
- 1.2 Thales Alenia Space Italia . . . 5
  - 1.2.1 COSE Centre . . . 6
  - 1.2.2 Ergonomia in ambito aerospaziale . . . 8
  - 1.2.3 Applicazioni VR in TASI . . . 8
- 1.3 Unreal Engine . . . 9
  - 1.3.1 Settore aerospaziale . . . 10
  - 1.3.2 Funzionalità principali . . . 11
  - 1.3.3 Sistema di programmazione visiva: Blueprint . . . 14
  - 1.3.4 Attori e componenti . . . 16
  - 1.3.5 Creazione di un plugin . . . 17

2 Stato dell'arte 20
- 2.1 Applicazioni di Realtà Virtuale in ambito aerospaziale . . . 20
  - 2.1.1 AeroVR . . . 20
  - 2.1.2 VR International Space Station Application . . . 21
  - 2.1.3 NASA Telexploration Project . . . 22
  - 2.1.4 NASA Marshall Space Flight Center . . . 23
- 2.2 VERITAS . . . 24

2.2.1 Ambienti sviluppati tramite V4U 25
2.3 Gestione dei dati di componenti 26
2.3.1 Requisiti ed obiettivi 27
2.4 Utilizzo di manichini realistici 28
2.4.1 Standard NASA per le misure antropometriche 29
2.4.2 Soluzioni attuali 29
2.4.3 Requisiti ed obiettivi 31

II Modulo 1 **32**

3 Strumenti utilizzati: Python in UE4 33
3.1 Python 33
3.1.1 Librerie utilizzate 33
3.2 Supporto per Unreal Engine 4 34
3.2.1 Abilitare Python in un progetto 35
3.2.2 Eseguire script in Python 35
3.2.3 Libreria *unreal* 35

4 Sviluppo del modulo per la gestione dati di componenti aerospaziali 37
4.1 Obiettivo 37
4.2 Sviluppo del modulo di gestione dati 38
4.2.1 Creazione del componente 38
4.2.2 Aggiunzione del Component 39
4.2.3 Importazione dei dati 40
4.2.4 Esportazione dei dati 40
4.2.5 Libreria *thales* 41
4.3 Creazione del plugin 41
4.3.1 Sviluppo dell'interfaccia grafica 42

III Modulo 2 **46**

5 Strumenti utilizzati: Animation Blueprint, Control Rig e MetaHuman 47
5.1 Animation Blueprint 47
5.1.1 Composizione dell'editor 48
5.1.2 EventGraph 49
5.1.3 AnimGraph 50
5.2 Control Rig 50
5.2.1 Composizione dell'editor 51

5.3 MetaHuman . . . 52
5.3.1 MetaHuman Creator . . . 52
5.3.2 Personalizzazione del MetaHuman . . . 53
5.3.3 Importare un MetaHuman su Unreal Engine . . . 56

6 Sviluppo dello strumento di misurazione 57
6.1 Obiettivo . . . 57
6.2 Sviluppo degli strumenti . . . 57
6.2.1 Controllo dello strumento e calcolo della misura . . . 58
6.2.2 Visualizzazione della misura . . . 59
6.2.3 Visualizzazione della mesh . . . 62
6.3 Risultato . . . 63

7 Sviluppo dello strumento per la creazione di manichini digitali 64
7.1 Obiettivo . . . 64
7.2 Creazione dei manichini base . . . 65
7.3 Misure dello standard NASA . . . 66
7.4 Sviluppo dello strumento per la creazione di manichini digitali . . . 75
7.4.1 Altezza . . . 77
7.4.2 Ampiezza vita . . . 78
7.4.3 Ampiezza schiena . . . 80
7.4.4 Profondità busto . . . 82
7.4.5 Lunghezza spalle . . . 83
7.4.6 Circonferenza collo . . . 84
7.4.7 Altezza caviglia, Lunghezza piede e Ampiezza piede . . . 86
7.5 Parametri e interfaccia dell'Animation Blueprint . . . 88

8 Creazione e messa in posa dei manichini 89
8.1 Creazione dei manichini . . . 89
8.1.1 Creazione del 1° percentile . . . 90
8.1.2 Creazione del 99° percentile . . . 91
8.2 Messa in posa dei manichini . . . 93
8.2.1 Replica delle pose degli astronauti . . . 94

IV Conclusione e risultati **97**

9 Risultati Modulo 1 98
9.1 Risultati . . . 98
9.1.1 Questionario utente . . . 98
9.2 Sviluppi futuri . . . 100

10 Risultati Modulo 2 102
10.1 Risultati 102
10.1.1 Verifica delle misure antropometriche 102
10.1.2 Caratteristiche e confronto con soluzioni precedenti 103
10.2 Sviluppi futuri 105

Bibliografia 106

Elenco delle tabelle

7.1 Altezze standard dei manichini di MetaHuman 65
7.2 Dimensioni antropometriche dello standard NASA riferite alla figura 7.2 67
7.3 Dimensioni antropometriche dello standard NASA riferite alla figura 7.3 68
7.4 Dimensioni antropometriche dello standard NASA riferite alla figura 7.4 69
7.5 Dimensioni antropometriche dello standard NASA riferite alla figura 7.5 70
7.6 Dimensioni antropometriche dello standard NASA riferite alla figura 7.6 71
7.7 Dimensioni antropometriche dello standard NASA riferite alla figura 7.7 72
7.8 Dimensioni antropometriche dello standard NASA riferite alla figura 7.8 73
7.9 Dimensioni antropometriche dello standard NASA riferite alla figura 7.9 74
7.10 Dimensioni antropometriche scelte dallo standard NASA per la creazione del manichino 76

8.1 Dimensioni antropometriche dello standard NASA per il 1° percentile 91
8.2 Dimensioni antropometriche dello standard NASA per il 99° percentile 92

9.1 Domande del questionario utente per il Modulo 1 99
9.2 Risposte possibili del questionario utente per il Modulo 1 99
9.3 Risposte del primo utente al questionario di gradimento 100
9.4 Risposte del secondo utente al questionario di gradimento 100

10.1 Dimensioni antropometriche dello standard NASA e strumenti utilizzati per la verifica dei percentili 103

Elenco delle figure

1.1 Esempio di una scena in Realtà Virtuale della Luna 3
1.2 Dispositivi di input di un HTC Vive 4
1.3 Dispositivo di output (HMD e output audio) di un HTC Vive . . . 5
1.4 Logo di Thales Alenia Space 6
1.5 Piantina del COSE Centre 7
1.6 Logo del COSE Centre 8
1.7 Logo di Unreal Engine 10
1.8 Ambiente simulato della ISS 11
1.9 Ambientazione virtuale di Marte 11
1.10 Esempi di ambienti virtuali creati con Unreal Engine 12
1.11 Esempio di Control Rig applicato ad un manichino 13
1.12 Esempi di rendering realistico tramite i sistemi di Unreal Engine . . 13
1.13 Esempi di simulazione ed effetti su Unreal Engine 14
1.14 Esempio di nodi nel Blueprint Visual Scripting 15
1.15 Esempio di un attore e i propri componenti su Unreal Engine 16
1.16 Interfaccia grafica per la creazione di un nuovo plugin in Unreal Engine 4 [14] 18

2.1 Esempio di scena visualizzata dall'utente in AeroVR [15] 21
2.2 Esempi di scene dell'applicazione di simulazione della ISS [16] . . . 22
2.3 Utenti esplorano ambienti virtuali di Marte tramite un HDM [17] . 22
2.4 Un utente utilizza una tuta per il motion capture, i dati catturati sono utilizzati per muovere il manichino all'interno dell'applicazione VR [18] 23
2.5 Logo del plugin VERITAS 4U 25
2.6 Ambiente virtuale del Modulo Colombus 26
2.7 Panoramica schematica dell'architettura di OCDT [20] 27
2.8 Illustrazione della stazione orbitante Gateway della NASA [21] . . . 28
2.9 Esempio delle misure antropometriche definite dallo standard NASA [22] 30

2.10 Manichini utilizzati attualmente da TASI (5° percentile donna, 95° percentile uomo) 31

4.1 Plugin *Thales Mesh Data Manager* nella lista plugin di Unreal Engine 41
4.2 Bottone base del plugin visualizzato nella barra degli strumenti di Unreal Engine 42
4.3 Interfaccia grafica del plugin *ThalesDataMeshManager* 45

5.1 Interfaccia grafica di un Animation Blueprint [40] 48
5.2 Nodo *Blueprint Initialize Animation* dell'EventGraph [41] 49
5.3 Nodo *Blueprint Update Animation* dell'EventGraph [41] 49
5.4 Esempio di un flusso di esecuzione di un AnimGraph [42] 50
5.5 Interfaccia grafica di un Control Rig Blueprint [44] 51
5.6 Interfaccia grafica del MetaHuman Creator [45] 53
5.7 Editor degli occhi di MetaHuman 54
5.8 Editor dei denti di MetaHuman 55
5.9 Editor dei capelli di MetaHuman 55
5.10 Sezione MetaHuman dell'applicazione Quixel Bridge 56

6.1 *Construction Script* dei Blueprint di misurazione *BP_Ruler* 58
6.2 Punti di controllo dello strumento *BP_Ruler* 59
6.3 Attributi del Text Render Component 60
6.4 Esempio dei componenti di testo in un attore 60
6.5 Nodi del materiale *M_Text* 61
6.6 Nodo *MathExpression* per calcolare la circonferenza di una ellisse . 61
6.7 Materiale traslucido *M_Translucent* 62
6.8 Esempi degli strumenti di misura 63

7.1 MetaHuman *Thales* all'interno del MetaHuman Creator 65
7.2 Figura esplicativa delle dimensioni antropometriche riguardanti altezze e ampiezze 67
7.3 Figura esplicativa delle dimensioni antropometriche riguardanti mani e seduta 68
7.4 Figura esplicativa delle dimensioni antropometriche riguardanti ulteriori altezze e ampiezze 69
7.5 Figura esplicativa delle dimensioni antropometriche riguardanti il cranio e ulteriori altezze e ampiezze 70
7.6 Figura esplicativa delle dimensioni antropometriche riguardanti diverse circonferenze 71
7.7 Figura esplicativa delle dimensioni antropometriche riguardanti ulteriori lunghezze 72

7.8 Figura esplicativa delle dimensioni antropometriche riguardanti i piedi e ulteriori dimensioni . . . 73
7.9 Figura esplicativa delle dimensioni antropometriche riguardanti ulteriori lunghezze e altezze . . . 74
7.10 Scheletro base dei manichini di MetaHuman . . . 75
7.11 Nodo *Transform (Modify) Bone* dell'AnimGraph di Unreal Engine . 76
7.12 Gruppo di nodi dell'AnimGraph per la dimensione *Altezza* . . . 77
7.13 Ossa fondamentali della gerarchia di MetaHuman per la definizione della misura *Ampiezza vita posteriore* . . . 78
7.14 Gruppo di nodi dell'AnimGraph per la dimensione *Ampiezza vita posteriore* . . . 79
7.15 Ossa fondamentali della gerarchia di MetaHuman per la definizione della misura *Ampiezza schiena* . . . 80
7.16 Gruppo di nodi dell'AnimGraph per la dimensione *Ampiezza schiena* 81
7.17 Gruppo di nodi dell'AnimGraph per la dimensione *Profondità busto* 82
7.18 Ossa fondamentali della gerarchia di MetaHuman per la definizione della misura *Lunghezza spalle* . . . 83
7.19 Gruppo di nodi dell'AnimGraph per la dimensione *Lunghezza spalle* 83
7.20 Osso *neck_01* della gerarchia di MetaHuman utile per la definizione della misura *Circonferenza collo* . . . 84
7.21 Gruppo di nodi dell'AnimGraph per la dimensione *Circonferenza collo* 85
7.22 Ossa fondamentali della gerarchia di MetaHuman per la definizione della misura *Altezza caviglia* . . . 86
7.23 Ossa fondamentali della gerarchia di MetaHuman per la definizione delle misure *Altezza caviglia*, *Lunghezza piede* e *Larghezza piede* . . 87
7.24 Gruppo di nodi dell'AnimGraph per le dimensioni *Altezza caviglia*, *Lunghezza piede* e *Larghezza piede* . . . 87
7.25 Parametri dell'Animation Blueprint per la creazione di manichini personalizzati . . . 88

8.1 Componenti e dettagli del Blueprint di un manichino MetaHuman . 90
8.2 Parametri dell'Animation Blueprint per il manichino rappresentante il 1° percentile . . . 91
8.3 Parametri dell'Animation Blueprint per il manichino rappresentante il 99° percentile . . . 92
8.4 Esempio dei manichini rappresentanti il 1° e il 99° percentile 93
8.5 Esempio di posa e dei controlli del Control Rig per il manichino rappresentante il 1° percentile . . . 94
8.6 Replica della posa di Samantha Cristoforetti . . . 95
8.7 Replica della posa di Daniel Burbank . . . 96

10.1 Esempi di verifica delle misure del manichino rappresentate il 1° percentile, tramite gli strumenti di misurazione sviluppati 103
10.2 Confronto tra gli attuali manichini di Thales e i manichini sviluppati in questa tesi 104

Elenco degli acronimi

UE4 Unreal Engine 4

UI User Interface

ISS Stazione Spaziale Internazionale

NASA National Aeronautics and Space Administration

TASI Thales Alenia Space Italia

COSE Centre COllaborative System Engineering Centre

VR Virtual Reality

AR Augmented Reality

DOF Degree Of Freedom

VERITAS Virtual Environment Research In Thales Alenia Space

V4U VERITAS 4U

AB Animation Blueprint

CR Control Rig

MH MetaHuman

DB Database

DBMS Database Management System

OCDT Open Concurrent Design Tool

ECSS European Cooperation for Space Standardization

Parte I

Panoramica generale

Capitolo 1

Introduzione

1.1 Realtà Virtuale

La Realtà Virtuale è un'interfaccia uomo-computer di alto livello che comprende la simulazione in real-time di un mondo realistico e le molteplici interazioni con gli oggetti di tale ambiente attraverso canali sensoriali multipli [1].

Viene quindi utilizzata la computer grafica per creare un mondo che sia percettivamente realistico. Bisogna precisare che il mondo virtuale è interattivo, quindi risponde agli input dell'utente e viceversa. Inoltre è multimodale, infatti la percezione del mondo avviene attraverso diversi sensi, non solo la vista.

Due concetti fondamentali della VR sono l'immersione e la presenza. Per *immersione* si intende la percezione del mondo virtuale come reale, anche detta immersione fisica. Maggiore è il numero di canali sensoriali stimolati, maggiore è il senso di immersione.

Invece, per *presenza* si intende la sensazione di appartenenza la mondo virtuale, anche detta immersione mentale. Affinché si provi questa sensazione, l'ambiente virtuale deve essere concepito e realizzato tenendo in mente questo fine. Inoltre è importante che l'utente sia predisposto ad accettare la sensazione di presenza in un ambiente non reale.

Per questo motivo, la misurazione oggettiva della presenza risulta molto difficile. Si possono quindi valutare alcuni fattori soggettivi:

- Misure *quantitative*: come la registrazione dei parametri fisiologici o dei tempi di risposta dell'utente, durante l'utilizzo dell'applicazione VR;

- Misure *qualitative*: tramite l'utilizzo di questionari.

Infine, gli elementi fondamentali per aumentare la presenza di un mondo VR sono: la qualità delle informazioni sensoriali (modellazione e rendering), la mobilità, il

controllo dei sensori e il controllo sull'ambiente. Tuttavia questi fattori sono limitati da vincoli tecnologici.

Un altro fattore molto fondamentale è l'immaginazione, infatti, la Realtà Virtuale è una simulazione complessa di ambienti verosimili, di conseguenza il mondo reale è solamente approssimato. Per superare queste approssimazioni, l'immaginazione umana risulta necessaria. Tuttavia, con l'evoluzione della tecnologia VR, questo fattore tende ad essere sempre meno importante.

Figura 1.1: Esempio di una scena in Realtà Virtuale della Luna

1.1.1 Dispositivi di input

La Realtà Virtuale richiede l'utilizzo di hardware specializzato, spesso il successo o il fallimento di una applicazione dipende del tipo di hardware scelto. È quindi necessario trovare un compromesso tra costi e budget.

Esistono diversi modi per categorizzare i dispositivi di input. Una prima classificazione può essere fatta in funzione delle caratteristiche dei dati che generano:

- Dati discreti: generano un evento per volta, in base alle azioni dell'utente. Ad esempio la tastiera;
- Dati continui: generano informazioni in modo autonomo o in base alle azioni dell'utente. Ad esempio dei tracker. I tracker sono dispositivi che permettono di catturare informazioni di posizione ed orientamento di oggetti reali. Utilizzano quindi dei sensori;
- Ibridi: generano dati sia discreti che continui. Ad esempio il mouse.

I dispositivi possono anche essere classificati in base ai gradi di libertà. Si passa da 1 DOF (Degree Of Freedom), come i bottoni di una tastiera, fino a 6+ DOF, come i sistemi che ottengono informazioni su posizione, orientamento oltre a gestire eventuali controlli aggiuntivi.

Esistono diverse tecnologie e tipi di sensori utilizzati per i dispositivi di input: elettronici, magnetici, basati sul tempo di volo, ottici, inerziali, meccanici, ibridi.

Figura 1.2: Dispositivi di input di un HTC Vive

1.1.2 Dispositivi di output

Essendo le applicazioni VR multimodali, anche le interfacce di output sono spesso multi-sensoriali per aumentare il realismo della simulazione stessa.

Generalmente il senso più importante è la vista e le interfacce di output collegate a questo senso sfruttano il concetto di stereoscopia. Per stereoscopia si intende la visione volumetrica del sistema visivo umano, il quale combina le due prospettive degli occhi per formare un'immagine stereoscopica, ottenendo dalle due viste informazioni su profondità e distanza. La differenza delle due immagini prodotte dagli occhi viene chiamata disparità binoculare.

La percezione di profondità è ottenuta tramite diversi fattori, sia fisiologici che psicologici. Generalmente i sistemi di stereoscopia sfruttano principalmente la disparità binoculare.

I display stereoscopici generano quindi due immagini diverse per i due occhi. È possibile catalogare questi dispositivi di output in due macro categorie:

- Sistemi basati su occhialini: gli utenti utilizzano degli occhialini, i quali possono sfruttare tecnologie di diverso tipo. Dato l'utilizzo di questi, i dispositivi possono risultare intrusivi per l'utente. Esistono due tipo di sistemi:

  - Head-Mounted Display (HMD): sfruttano due display diversi per i due occhi (figura 1.3) (es. HTC Vive, vedi paragrafo 1.2.1);
  - Sistemi multiplexed: sfruttano un singolo display.

- Sistemi auto-stereoscopici: non fanno uso di tecnologie e strumenti invasivi.

Figura 1.3: Dispositivo di output (HMD e output audio) di un HTC Vive

1.2 Thales Alenia Space Italia

Thales è una società francese che opera in molteplici settori, dalla sicurezza all'aerospazio, ed è attiva in Italia dal 1988, contando fino a 2800 dipendenti nel paese e 7 siti attivi. Le principali attività della società sono divise in 5 grandi settori [2]:

- Aerospazio: sviluppo e distribuzione di sistemi di sicurezza per il traffico aereo per il mercato nazionale e per quello internazionale;
- Difesa: progettazione, distribuzione e installazione di sistemi e reti di telecomunicazione infrastrutturali e tattiche per il mercato della difesa;
- Sicurezza: progettazione di soluzioni mission-critical e applicazioni per la sicurezza di beni e cittadini;
- Trasporti: progettazione di sistemi di sicurezza, telecomunicazione, supervisione, gestione e controllo delle informazioni per treni, tram e metropolitana;

- Spazio: progettazione e realizzazione di sistemi ed equipaggiamenti per telecomunicazioni spaziali, navigazione satellitare, osservazione della Terra, scienza, esplorazione e trasporto.

In particolare per il settore spaziale, la compagnia francese è attiva con Thales Alenia Space, una società nata dalla joint-venture tra Thales (67%) e Leonardo (33%), la quale opera in Italia attraverso Thales Alenia Space Italia (TASI) e conta fino a 2300 dipendenti distribuiti in quattro siti principali: Roma, Torino, L'Aquila e Milano. La filiale italiana ha fornito un determinante apporto allo sviluppo delle infrastrutture orbitanti, contribuendo, fino ad oltre il 50%, alla realizzazione dei moduli pressurizzati della Stazione Spaziale Internazionale. Thales infatti vanta la collaborazione delle maggiori industrie spaziali internazionali e nei programmi delle più prestigiose istituzioni quali la NASA, l'Agenzia Spazile Europea (ESA) e l'Agenzia Spaziale Italiana.

Inoltre, la società in Italia investe notevolmente nella Ricerca e Sviluppo, per sostenere un flusso continuo di innovazioni in tutti i settori, anche attraverso una vasta rete di partner accademici e industriali, che include molte piccole imprese innovative [3].

Figura 1.4: Logo di Thales Alenia Space

1.2.1 COSE Centre

L'evoluzione del settore della Realtà Virtuale ha inciso sui processi di progettazione dei sistemi spaziali. La VR può essere infatti impiegata in molte fasi del ciclo di vita di un progetto, partendo dalla progettazione, fino ad arrivare alla simulazione.

Il centro COSE (COllaborative System Engineering) è un centro di ricerca di Thales Alenia Space Italia, presente presso la sede di Torino e operativo dal 2003. Il gruppo è specializzato nella ricerca e sviluppo di tecnologie per la creazione di

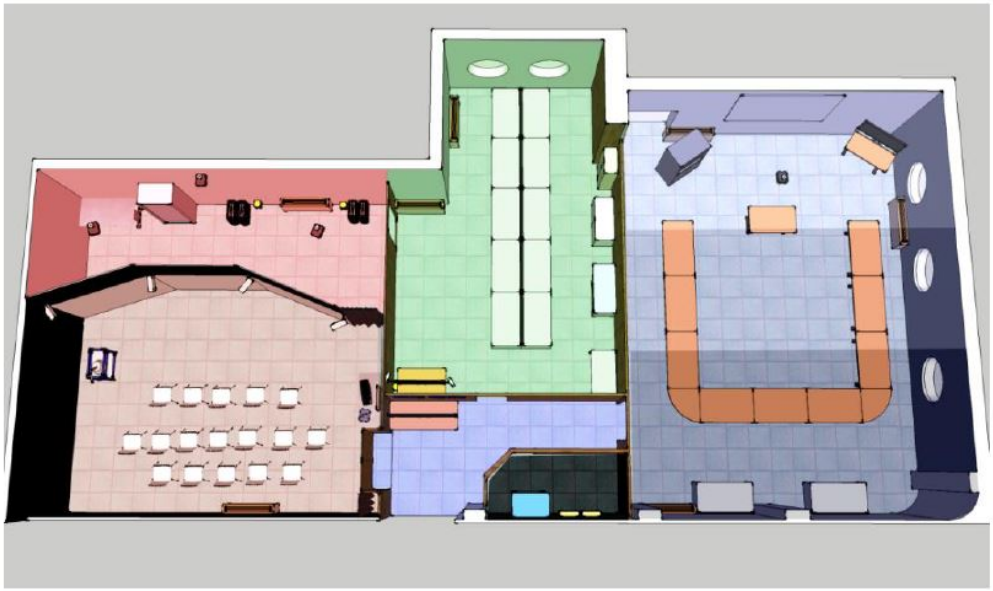


Figura 1.5: Piantina del COSE Centre

applicazioni di *Realtà Virtuale* (VR) e *Realtà Aumentata* (AR). Le occupazioni principali sono: definizione dei requisiti, progettazione, prototipazione e testing di ambienti virtuali e di ingegneria collaborativa.

Il centro è composto da tre ambienti, in figura 1.5 è possibile osservare una piantina di questo:

- Technology Research Office (TRO): ufficio suddiviso in due sezioni. Il primo team (TRO-1) si occupa della realtà virtuale. Il secondo team (TRO-2) si occupa delle applicazioni ingegneristiche;

- Collaborative Room: stanza collaborativa, dedicata a riunioni per il team e attrezzata per le videoconferenze. Viene utilizzata per attivilità di Collaborative Engineering, quindi per sessioni di progettazione, indagini, confronto e analisi;

- Virtual Reality Laboratory (VR-Lab): nel quale sono presenti diverse apparecchiature.

  – Sistema CAVE a 3 pareti: le pareti sono affiancate e inclinate tra loro di 30 gradi. Ogni parete è retroilluminata tramite un proiettore. Per ottenere l'effetto di stereoscopia, ogni proiettore riceve due flussi per i due occhi. Tramite l'uso di particolari occhialini, l'utente può osservare una scena tridimensionale;

- Dispositivi HTC Vive: due postazioni;
- Dispositivi Microsoft Hololens: usati principalmente per progetti di Realtà Aumentata.

Per maggiori informazioni consultare il paragrafo sui dispositivi di output per la Realtà Virtuale (paragrafo 1.1.2);

Figura 1.6: Logo del COSE Centre

1.2.2 Ergonomia in ambito aerospaziale

L'International Ergonomics Association (IEA) è una associazione internazionale di organizzazioni che operano nel settore dell'ergonomia. Secondo questa, l'ergonomia è la disciplina scientifica che si occupa di capire l'interazione tra umani e altri elementi di un sistema, e la professione che applica teorie, principi, dati e metodi di progettazione al fine di ottimizzare il benessere dell'uomo e le prestazioni complessive del sistema [4].

L'ergonomia è quindi una disciplina fondamentale in ambito aerospaziale. Sono infatti definite una serie di misure che i componenti ed oggetti di un modulo devono rispettare, in modo da poter valutarne l'ergonomia. Generalmente, in passato, i dati venivano raccolti in formato cartaceo, dopodiché la progettazione prevedeva l'utilizzo di questi dati per la progettazione. Tuttavia questo processo risulta molto complesso, non preciso e altamente soggetto ad errori, i quali sono generalmente commessi durante la fase di progettazione e quindi difficili da recuperare, soprattutto se si è arrivati alle fasi finali del progetto.

Una pratica spesso utilizzata è la creazione di un prototipo del modulo e l'utilizzo di una persona reale, la quale serve ad identificare le criticità del sistema. Questa prima soluzione risulta abbastanza dispendiosa e non efficiente.

1.2.3 Applicazioni VR in TASI

Con l'evoluzione della Realtà Virtuale, l'industria aerospaziale ha rivolto la propria attenzione allo sviluppo di applicazioni VR applicate alla progettazione e analisi di

moduli spaziali, tra queste anche Thales. Un particolare vantaggio dell'uso della Realtà Virtuale, rispetto alle tradizionali metodologie, è la possibilità di poter studiare un modulo ancora in fase di progettazione e soprattutto in un ambiente simulato di microgravità, anche grazie alla creazione di manichini virtuali per la simulazione. Un ambiente del genere è infatti altamente dispendioso e complicato da riprodurre fisicamente, generalmente ciò è fatto tramite l'utilizzo di speciali velivoli che effettuano voli parabolici per simulare l'assenza di peso.

La Realtà Virtuale, inoltre, permette di ottenere altri importanti vantaggi durante la fase di progettazione. Innanzitutto è possibile mostrare a collaboratori e clienti, una versione virtuale del prodotto finale. Questi possono infatti esplorare l'ambiente creato tramite appositi dispositivi. Di conseguenza, tramite gli strumenti citati, è possibile implementare il concetto di ingegneria collaborativa.

In TASI, la Realtà Virtuale è stata ampiamente utilizzata per lo sviluppo di diversi moduli e sistemi, tra questi abbiamo il modulo spaziale Columbus, la Chinese Space Station (CSS) e il modulo lunare Lunar Orbital Platform-Gateway (LOP-G).

1.3 Unreal Engine

Una delle applicazioni più diffuse per la creazione di applicazioni in Realtà Virtuale, e ampiamente utilizzata da Thales Alenia Space, è Unreal Engine. UE è un motore di gioco sviluppato da Epic Games, cui prima versione risale al 1996. Il motore è stato sviluppato in C++ e attualmente supporta una gran numero di piattaforme. Tra queste ci sono piattaforme desktop come Windows e macOS, piattaforme mobile come Android e iOS, console e altri vari sistemi come le piattaforme per la realtà virtuale [5].

È stato sviluppato dal fondatore di Epic Games, Tim Sweeney, per la produzione del loro videogioco 3D in prima persona *Unreal*. Successivamente sono state rilasciate continue versioni, fino ad arrivare a quella attuale, ovvero Unreal Engine 4. Attualmente è in sviluppo una nuova versione, ovvero Unreal Engine 5, disponibile in early access [5].

L'uso principale del motore è lo sviluppo di videogiochi, per cui è stato inizialmente creato. Contiene infatti moltissime funzionalità specifiche per la creazione di questi. Tuttavia, da qualche tempo, UE4 è diventato uno standard in molti settori, segue una lista di alcuni:

- Film e televisione: tramite il motore grafico e la possibilità di avere rendering in tempo reale, si ha la possibilità di velocizzare i tempi di produzione, ottenendo un riscontro immediato. Inoltre permette di usare gli stessi asset in più fasi della produzione [6].
- Architettura: il vantaggio principale è la facilità d'uso e la facilità con cui si può passare da un software CAD o 3D ad un UE4. Inoltre, tramite le funzionalità

del game engine, è possibile coinvolgere facilmente i clienti nella progettazione, tramite la VR o semplicemente tramite gli strumenti di collaborazione messi a disposizione [7].

- Automotive: permette di creare esperienze immersive, in modo da mostrare ai clienti ogni opzione del catalogo o far fare un test drive tramite la Realtà Virtuale. Inoltre, UE4 permette di effettuare simulazioni, utili al processo di produzione, di diverso genere, all'interno di un singolo software [8].
- Trasmissione ed eventi dal vivo: permette di integrare ambienti reali con la computer grafica, in tempo reale. Inoltre, il motore supporta molti degli hardware utilizzati per la trasmissione di eventi dal vivo, oltre ad avere un'alta affidabilità [9].
- Simulazione: tramite UE4 è possibile creare simulazione altamente realistiche, sia per umani che per macchine. Esistono quindi diverse applicazioni di questo settore come la medicina, il settore navale, il settore aerospaziale.

Figura 1.7: Logo di Unreal Engine

1.3.1 Settore aerospaziale

Attualmente Unreal Engine è ampiamente utilizzato in ambito aerospaziale, tra le più rinomate agenzie spaziali abbiamo sicuramente quella degli Stati Uniti d'America, ovvero la NASA. Una delle applicazioni più importanti è la ricostruzione della Stazione Spaziale Internazionale (ISS) all'interno di UE4, utilizzata per simulazione ed addestramento di futuri astronauti. L'ambiente virtuale (figura 1.8) è infatti una perfetta replica funzionante della stazione spaziale, che comprende i suoi complessi moduli e sistemi. In questo modo è possibile migliorare la formazione di astronauti e, contemporaneamente, diminuire i costi di addestramento [10].

Un'altra applicazione è la creazione di un ambiente virtuale che simula il terreno marziano (figura 1.9). L'obiettivo è replicare l'esperienza di trovarsi veramente su Marte tramite l'utilizzo di un caschetto di Realtà Virtuale. L'applicazione è stata sviluppata in vista dell'atterraggio sul pianeta rosso, pianificata dalla NASA per gli anni 30 [11].

Figura 1.8: Ambiente simulato della ISS

Figura 1.9: Ambientazione virtuale di Marte

1.3.2 Funzionalità principali

Unreal Engine 4 possiede una vastità di funzionalità che spaziano moltissimi settori, di seguito se ne elencano le principali [12]:

- Integrazione con la pipeline di produzione: il motore infatti permette di

convertire intere scene dalla maggior parte dei software 3D (es. 3ds Max, Revit, SketchUp Pro, Cinema 4D, Rhino, SolidWorks, Catia, ecc.). Inoltre supporta i più diffusi formati di interscambio come FBX, USD e Alembic.

Per ottimizzare la pipeline, UE supporta nativamente Python, con la quale è possibile creare script in grado di comunicare con l'editor dell'engine stesso o per comunicare con altre applicazioni della pipeline. Questa funzionalità è vastamente utilizzata per lo sviluppo del primo modulo di questa tesi;

- Costruzione di mondi virtuali: a supporto di questo, Unreal Engine propone diverse funzionalità. Il motore permette di gestire modelli complessi tramite un sistema di generazione di LOD (Level of Detail) automatico. Inoltre fornisce una serie di strumenti per la creazioni di scene realistiche come: un sistema gestione di fogliame, un sistema di gestione di flussi d'acqua (fiumi, laghi), un sistema di gestione illuminazione, cielo e illuminazione ambientale, oltre ad un insieme di strumenti che permettono di definire e modificare i terreni dell'ambiente;

Figura 1.10: Esempi di ambienti virtuali creati con Unreal Engine

- Animazione: il motore permette di creare animazioni molto realistiche all'interno di UE4 stesso. Tra gli strumenti più utilizzati ci sono l'*Animation Blueprint* e il *Control Rig* (figura 1.11), entrambi affrontati nel capitolo 5 di questa tesi;

- Rendering e illuminazione: altamente realistici grazie all'uso di diverse tecniche, utilizzabili dipendentemente dal fine dell'applicazione in sviluppo (figura 1.12a). Inoltre è presente un editor dei materiali molto flessibile, che permette la creazione di materiali realistici (figura 1.12b);

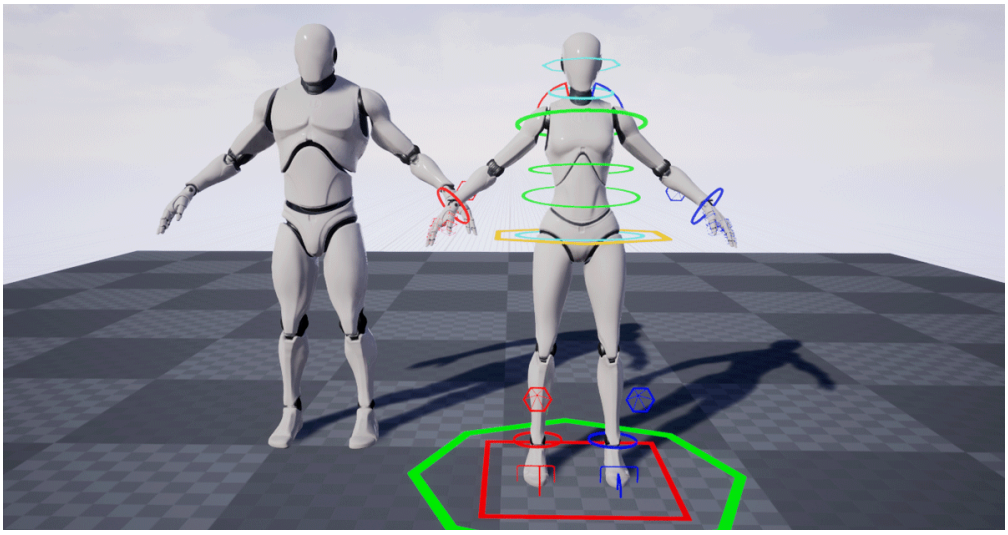
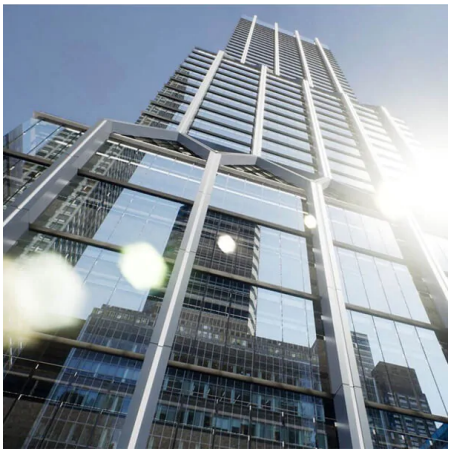


Figura 1.11: Esempio di Control Rig applicato ad un manichino



(a) Esempio di illuminazione

(b) Esempio di materiale

Figura 1.12: Esempi di rendering realistico tramite i sistemi di Unreal Engine

- Simulazione ed effetti: UE4 integra dei sistemi avanzati per la simulazione ed effetti. Tra questi sicuramente *Niagara* e *Chaos*. Il primo è un sistema che permette di creare effetti realistici come fuoco, fumo, polvere e acqua, ma anche degli effetti più astratti. Il secondo, Chaos, è un motore fisico che permette di gestire distruzioni e frantumazione di oggetti di scena (figura 1.13a). Il motore grafico inoltre supporta degli strumenti per la simulazione di tessuti e capelli (figura 1.13b);

(a) Motore fisico *Chaos*

(b) Simulazione di capelli

Figura 1.13: Esempi di simulazione ed effetti su Unreal Engine

1.3.3 Sistema di programmazione visiva: Blueprint

Una delle funzionalità più utilizzate su Unreal Engine è il *Blueprint Visual Scripting*, ovvero un sistema di programmazione visiva basata sull'uso di nodi. Con questi è possibile creare elementi di gameplay e funzionalità per la propria applicazione, ed essendo estremamente flessibile e completo, è un'ottima alternativa all'utilizzo di C++.

Bisogna comunque sottolineare come, per alcuni specifici casi, l'uso del classico linguaggio di programmazione risulta migliore e più efficiente. Tuttavia è possibile definire classi base in C++, che le classi blueprint possono estendere, e nodi personalizzati tramite le tradizionali funzioni. In questo modo è possibile estendere, il già esteso, sistema di programmazione visiva.

Tipi di Blueprint Esistono tre tipi di Blueprint:

- *Blueprint Class*: spesso riferita solamente come *Blueprint*. È un asset che permette ai programmatori di definire nuove funzionalità, tramite l'uso della programmazione visiva. Possono ereditare da una qualsiasi classe, sia C++ che un'altra classe blueprint. Nel caso in cui la classe padre si un attore, l'asset si comporterà come un normale attore e sarà possibile posizionarlo all'interno della scena.

- *Data-Only Blueprint*: classe blueprint che contiene solo la logica, le variabili e i componenti ereditati dalla classe padre. I valori delle proprietà possono essere modificati e aggiornati.

- *Level Blueprint*: associato ad un singolo livello, permette di definire la logica di quest'ultimo. Non è possibile crearne altri dall'editor.
- *Blueprint Interface*: collezione di una o più funzioni, di cui non si definisce il contenuto. Concetto simile alle interfacce dei classici linguaggi di programmazione. Può essere aggiunta ad altre classi blueprint.
- *Blueprint Macro Library*: collezione di una o più macro, ovviamente definite. Le *Macro* sono degli insiemi di nodi collegati, raggruppati in un singolo nodo. Vengono utilizzate per evitare duplicazione di grafi e migliorare la manutenibilità.
- *Blueprint Utilities*: permette di aggiungere funzionalità all'editor di Unreal Engine, non è quindi legato a funzionalità dell'applicazione che si sta sviluppando con il motore di gioco.

Struttura dei nodi Ogni nodo possiede un insieme di pin che determinano dati in input e/o dati in output. Inoltre, i nodi che eseguono operazioni hanno un pin in input e in output per il flusso di esecuzione.

Figura 1.14: Esempio di nodi nel Blueprint Visual Scripting

Vediamo l'esempio in figura 1.14, il nodo in alto permette di ottenere il valore memorizzato nella variabile *Player Health*, infatti possiede un singolo pin di output. Quello in basso permette di impostare il valore della variabile *Player Health*, infatti possiede un pin di input del valore e due pin, input e output, per il flusso di esecuzione.

Per approfondimenti si consiglia la consultazione della documentazione ufficiale di Unreal Engine [13].

1.3.4 Attori e componenti

Per comprendere al meglio il lavoro svolto in questa tesi, è necessario introdurre e spiegare il funzionamento di attori e componenti all'interno di Unreal Engine 4.

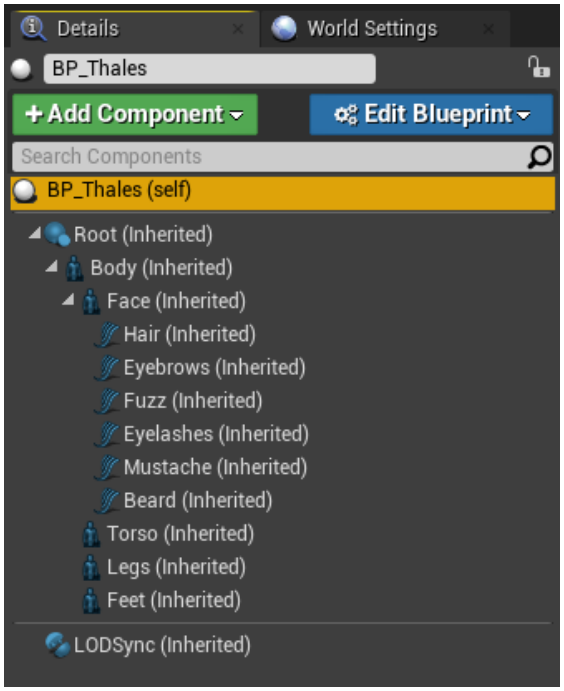


Figura 1.15: Esempio di un attore e i propri componenti su Unreal Engine

Attori Gli *Actor* (o attori) di UE4 sono oggetti che possono essere posizionati e creati all'intero di una scena e supportano le trasformazioni 3D di traslazione, rotazione e scala. Sono oggetti vari e di diverso tipo, come camere, mesh statiche, personaggi controllabili. Per la creazione di un attore si fa riferimento alla classe AActor, estendibile tramite classe C++ o tramite classe Blueprint.

Componenti I *Component* (o componenti), invece, sono uno speciale tipo di oggetti i quali sono attaccabili ad un qualsiasi tipo di attore e cui utilizzo principale è la condivisione di comportamenti comuni tra attori diversi. Questi infatti possono possedere un numero variabile di componenti che permettono di estendere il comportamento e le funzionalità (figura 1.15). La classe UActorComponent permette di creare nuovi componenti tramite il concetto di ereditarietà, sfruttabile tramite classe C++ o tramite classe Blueprint.

I component sono quindi l'unico modo per renderizzare modelli e immagini, gestire collisioni, riprodurre un suono, in generale corrispondono a tutto ciò che

l'utente vede e con cui interagisce. Esistono tre classi principali da poter estendere per la creazione di questi:

- *Actor Component*: derivato dalla classe *UActorComponent*, è il component migliore per lo sviluppo di comportamenti più astratti come movimenti, inventari e gestione degli attributi. Non possiedono una transform (insieme dei tre vettori traslazione, rotazione e scala);
- *Scene Component*: derivato dalla classe *USceneComponent*, che estende la classe precedente. Permettono di sviluppare comportamenti basati sulla posizione all'interno della scena, senza avere una vera e propria rappresentazione geometrica;
- *Primitive Component*: derivato dalla classe *UPrimitiveComponent*, che estende la precedente classe, sono dei Scene Component con una rappresentazione geometrica in scena.

1.3.5 Creazione di un plugin

Un plugin in Unreal Engine è una collezione di script e dati che gli sviluppatori possono abilitare o disabilitare per i singoli progetti. In generale, i pacchetti possono aggiungere funzionalità all'applicazione che si sta creando con il motore, modificare o estendere le funzionalità dell'editor di UE, tra cui l'interfaccia grafica (user interface, UI), e creare nuovi tipi di file [14].

Le cartelle di un plugin sono le seguenti:

- Cartella Source: strutturata in una o più cartelle per contenere il codice sorgente, tuttavia non è obbligatorio che un plugin possegga del codice;
- Cartella Binaries: contiene il codice compilato della cartella precedente, nel caso ci fosse;
- Cartella Content: contiene gli eventuali Asset dello specifico plugin.

Per gestire e visualizzare i plugin abilitati per un particolare progetto, viene messa a disposizione la finestra Plugins accessibile dal menù principale di Unreal Engine.

Per la creazione di un plugin, il motore di gioco mette a disposizione una semplice interfaccia grafica (figura 1.16). Per potervi accedere bisogna cliccare nel bottone *New Plugin* visualizzato nella finestra dei Plugins, citata precedentemente. All'interno di questa è possibile impostare i dati che descrivono il contenuto del pacchetto, come l'autore e la descrizione. Inoltre, viene data la possibilità di scegliere da una lista di template base da cui partire per i propri plugin, permettendo di velocizzare l'intero processo:

- *Blank*: plugin senza alcun tipo di contenuto, con il codice essenziale utile per il corretto funzionamento. È generalmente usato nel caso di pacchetti non visuali;
- *Content Only*: plugin senza codice sorgente, contiene quindi solo la cartella Content, nella quale poter inserire eventuali asset;
- *Blueprint Library*: utilizzato per definire nuovi nodi (quindi funzioni) per la programmazione visiva tramite blueprint;
- *Editor Toolbar Button*: crea un plugin con un semplice buttone inserito nella barra degli strumenti dell'editor di UE. Tramite questo è possibile definire le istruzioni da eseguire nell'evento "*OnButtonClick*" del bottone;
- *Editor Standalone Window*: crea un plugin che aggiunge un semplice bottone nella barra degli strumenti di UE. Una volta cliccato, questo aprirà una nuova finestra vuota;
- *Editor Mode*: contiene un insieme di strumenti per definire l'interfaccia grafica di una nuova *Editing mode* di UE4 (es. Foliage, Landscape, ecc.);
- *Third Party Library*: plugin che include una libreria di terze parti.

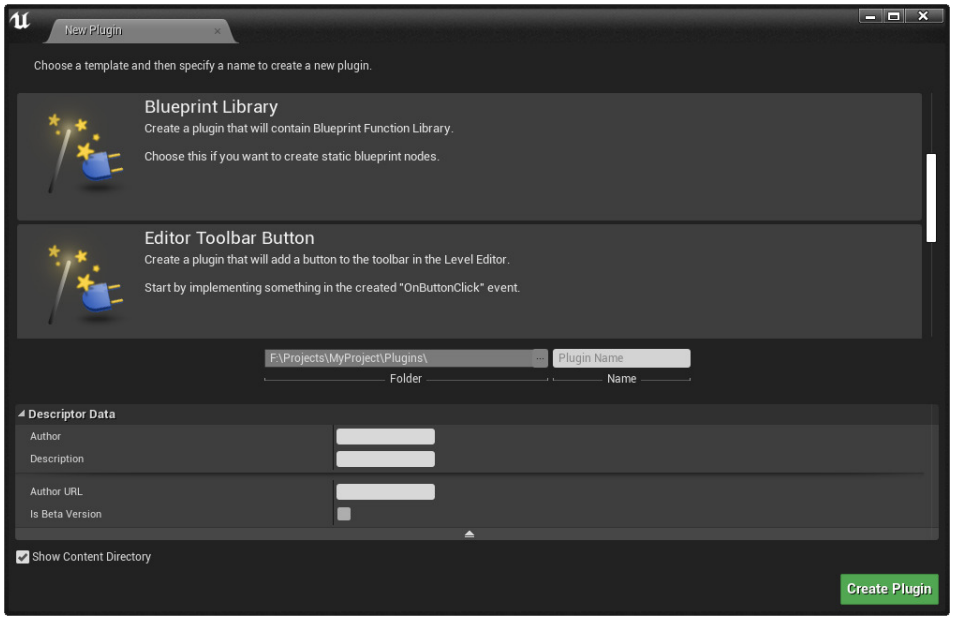


Figura 1.16: Interfaccia grafica per la creazione di un nuovo plugin in Unreal Engine 4 [14]

Una volta creato, il plugin sarà già abilitato e disponibile all'interno del proprio progetto. Inoltre, sarà possibile esportarlo facilmente copiando la cartella contenuta nella cartella Plugins del progetto.

Capitolo 2

Stato dell'arte

2.1 Applicazioni di Realtà Virtuale in ambito aerospaziale

Come accennato in precedenza, la Realtà Virtuale è ampiamente utilizzata in ambito aerospaziale. Nel corso di questi ultimi anni molte applicazioni sono state sviluppate per migliorare e ottimizzare i processi di progettazione, sviluppo e revisione dei sistemi spaziali. In questa sezione verranno presentati alcuni esempi di applicazioni VR in ambito aerospaziale.

2.1.1 AeroVR

AeroVR è un ambiente di progettazione aerospaziale immersivo con l'obiettivo di aiutare il processo di progettazione dei componenti in ambito aerodinamico, visualizzando in modo interattivo le prestazioni e la geometria dei componenti stessi [15].

L'applicazione è stata sviluppata dal Dipartimento di Ingegneria dell'Università di Cambridge per sfruttare le tecnologie VR ed applicarle al campo dell'aerospazio, in modo da migliorarne i processi di progettazione. In particolare, nell'articolo [15], si utilizza AeroVR per lo studio del prototipo di un una pala di un compressore del motore di un velivolo.

Una delle scene dell'applicazione è mostrata in figura 2.1, in cui vengono visualizzati il modello completo del motore, con la geometria nominale delle pale in blu, e due grafici a dispersione tridimensionali.

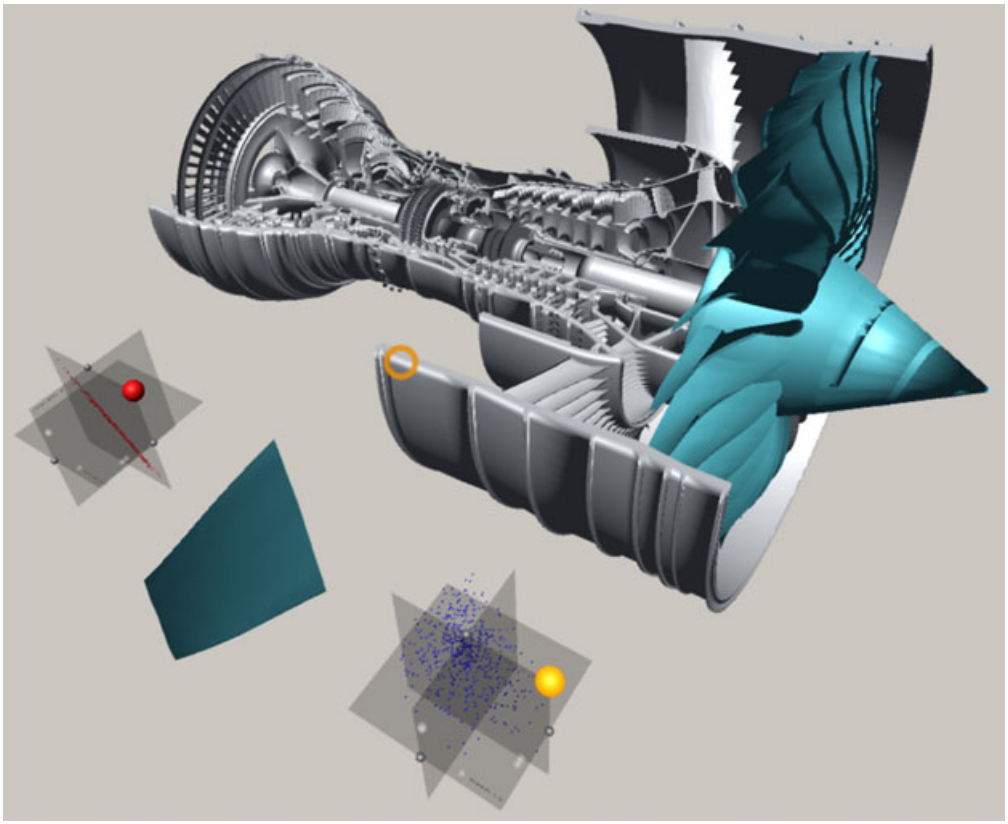


Figura 2.1: Esempio di scena visualizzata dall'utente in AeroVR [15]

2.1.2 VR International Space Station Application

Il progetto della Stazione Spaziale Internazionale è il programma più largo dell'Agenzia Spaziale Russa (Rosaviakosmos). Il Centro di addestramento cosmonauti Jurij Gagarin (GCTC) è concentrato nello sviluppo di diversi moduli di addestramento dei cosmonauti russi per missioni spaziali.

Uno dei sistemi sviluppati dall'agenzia è un ambiente in Realtà Virtuale il quale simula l'esterno e l'interno dell'ISS, in modo da aiutare i cosmonauti a familiarizzare con la configurazione della stazione spaziale. All'interno dell'ambiente è possibile interagire con diversi oggetti dell'equipaggiamento, visitare i compartimenti, percorrere le linee di servizio principali e molto altro [16].

Nelle figure 2.2 sono mostrati tre diversi ambienti della stessa applicazioni, è possibile notare come gli interni della stazione siano altamente realistici e ricchi di reale componentistica.

(a) (b) (c)

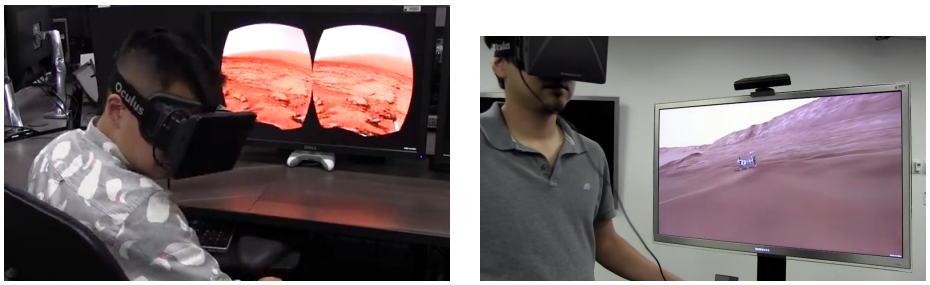
Figura 2.2: Esempi di scene dell'applicazione di simulazione della ISS [16]

2.1.3 NASA Telexploration Project

Anche la NASA è coinvolta nella creazione di molteplici applicazioni in Realtà Virtuale, sia per la simulazione e addestramento, che per la progettazione. L'obiettivo del Progetto Telexploration è il ricercare come ambienti immersivi e interazioni uomo-macchina intuitive possano consentire scienziati e ingegneri, ma anche il pubblico generale, di interagire in modo efficace con i veicoli spaziali della NASA e con ambienti alieni [17].

Grazie alla raccolta di molteplici dati bidimensionali e tridimensionali del terreno marziano, l'agenzia spaziale americana ha creato diversi ambienti virtuali immersivi di Marte. Dopodiché, utilizzando un Head Mounted Display (HMD), permette l'esplorazione di questi (figura 2.3).

Il processo di creazione di queste applicazioni è stato accelerato dalla collaborazione con diverse aziende private dell'industria dei videogiochi.



(a) (b)

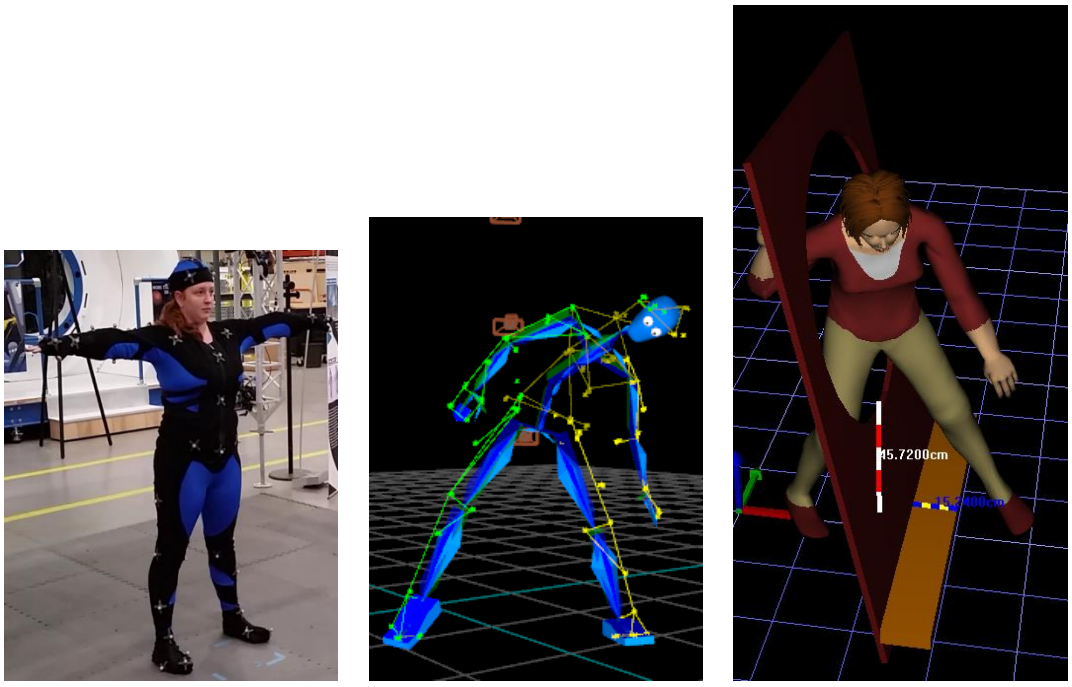
Figura 2.3: Utenti esplorano ambienti virtuali di Marte tramite un HDM [17]

2.1.4 NASA Marshall Space Flight Center

Il gruppo HFE (Human Factors Engineering) nel Marshall Space Flight Center della NASA ha sviluppato delle tecniche che uniscono la realtà virtuale e il motion capture per lo sviluppo di applicazioni. Per motion capture si intende la registrazione del movimento del corpo umano tramite appositi dispositivi di cattura, i dati sono puoi sfruttati in tempo reale o meno. Nel caso di questa applicazione, i dati registrati sono utilizzati in tempo reale per simulare il movimento di un essere umano all'interno dell'ambiente di realtà virtuale [18].

Queste tecniche sono state applicate per lo sviluppo e analisi dei Deep Space Habitas (DSH), una serie di prototipi di moduli spaziali, utili all'esplorazione spaziale con equipaggio verso la Luna, Marte e asteroidi.

Tramite questi strumenti, ingegneri e utenti possono analizzare, studiare e provare in prima persona i prototipi, senza la necessità che questi vengano fisicamente costruiti.



(a) **(b)** **(c)**

Figura 2.4: Un utente utilizza una tuta per il motion capture, i dati catturati sono utilizzati per muovere il manichino all'interno dell'applicazione VR [18]

2.2 VERITAS

Per la simulazione di scenari in Realtà Virtuale, Thales Alenia Space ha progettato e sviluppato il framework *VERITAS* (Virtual Environment Research In Thales Alenia Space). Il COSE Centre utilizza VERITAS per lo sviluppo di diverse applicazioni, si citano alcune delle più importanti:

- *TraVis* (TRAjectory VISualization): usato per l'esplorazione del sistema solare e dell'universo, oltre che per la simulazione delle traiettorie di pianeti, satelliti ed altri corpi;

- *CVAST* (Virtual Cargo Accomodation Support Tool): strumento usato per effettuare analisi sui vari contenitori che vengono collocati nei moduli della ISS;

- *VIRTooS* (VIrtual Radiation TOOlSet): permette di analizzare e visualizzare le radiazioni propagate sugli spacecraft e attorno alla Terra;

- *DesiRe* (DESIgn REview application): applicazione utilizzata per la revisione del design e per l'analisi dei mock-up virtuali;

- *RoSi* (ROver SImulator): usato per simulare il movimento di un rover su Marte;

- *LandS* (LANDer Simulator): permette di simulare l'atterraggio di un lander su una superficie planetaria;

- *PaTATrack* (PArse To Ascii TRACKing data application): traccia il movimento di un dispositivo di tracking e ne memorizza l'output.

Thales ha sviluppato VERITAS 4U (V4U), ovvero un'evoluzione di VERITAS sotto forma di plugin per il motore Unreal Engine (paragrafo 1.3). V4U contiene quindi tutti gli strumenti necessari alla creazione di ambienti virtuali esplorabili, sia in modalità desktop che con dispositivi completamente immersivi per la Realtà Virtuale (es. HTC Vive, il COSE Centre possiede infatti 2 postazioni). È importante sottolineare che il plugin è in continuo sviluppo e evoluzione.

Gli strumenti messi a disposizione dal pacchetto permettono all'utente di impersonare un astronauta a bordo di un modulo spaziale, di simulare gli ambienti virtuali e particolari condizioni come la microgravità. I fini del plugin sono molteplici, tra cui: effettuare verifiche e analisi di progettazione, studiare l'ergonomia dei sistemi e effettuare l'addestramento per astronauti veri e propri.

Figura 2.5: Logo del plugin VERITAS 4U

2.2.1 Ambienti sviluppati tramite V4U

Tra gli ambienti sviluppati tramite VERITAS 4U abbiamo la Stazione Spaziale Cinese (CSS) e il modulo Columbus.

Stazione Spaziale Cinese Similmente alla Stazione Spaziale Internazionale, la CSS è una stazione spaziale in orbita intorno alla Terra. L'obiettivo principale della missione è lo sviluppo e il miglioramento della tecnologia aerospaziale al fine di una futura esplorazione dello spazio più profondo. Inoltre, la Cina usa la CSS per favorire la cooperazione internazionale, annunciando infatti l'opportunità per tutti i paesi delle Nazioni Unite di partecipare alle missioni scientifiche della stazione [19].

Il modello virtuale della Stazione Spaziale Cinese è stato creato per scopi progettuali, in collaborazione con l'agenzia spaziale cinese. L'ambiente creato permette numerose interazioni tra cui la possibilità di aprire vani, prendere e posizionare oggetti, attivare e disattivare gli schermi del modulo. Infine, si ha la possibilità di interagire con il modulo Cupola, che consente di poter osservare lo spazio a 360 gradi.

Colombus Il secondo modulo, Colombus, è un laboratorio di ricerca sviluppato dall'Agenzia Spaziale Europea. È stato costruito a Torino e successivamente utilizzato per la Stazione Spaziale Internazionale. Il modulo è stato lanciato in orbia il 7 Febbraio 2008 ed attualmente risulta attivo nella ISS.

Tramite l'utilizzo di V4U si è potuto creare un ambiente altamente realistico e accurato. È infatti permessa l'esplorazione interna del modulo, oltre ad una serie di interazioni possibili con l'ambiente. Infatti è possibile prendere e spostare una serie di oggetti, interagire con le maniglie in modo da ruotare l'intero modulo e interagire con KUBIK, un incubatore a temperatura controllata utilizzato per esperimenti che sfruttano la microgravità.

Il lavoro di questa tesi si è concentrato su due aspetti della Realtà Virtuale di TASI, la gestione dei dati di componenti reali di sistemi spaziali e lo sviluppo di manichini realistici per il design ergonomico in ambito aerospaziale.

Figura 2.6: Ambiente virtuale del Modulo Colombus

2.3 Gestione dei dati di componenti

Lo scambio di dati ed informazioni è fondamentale per le fasi di progettazione, revisione e verifica di moduli spaziali. In particolare, l'Agenzia Spaziale Europea mette a disposizione per tutti i collaboratori (agenzie spaziali nazionali e aziende privati che collaborano con essa) una struttura dati nella quale vengono inserite informazioni dei sistemi spaziali. L'architettura del database è definita negli standard della ECSS (European Cooperation for Space Standardization), un gruppo formato dall'ESA stessa e l'industria spaziale europea, la quale si occupa di definire una serie di standard in ambito aerospaziale e che coprono l'intero ciclo di vita dei sistemi spaziali.

OCDT (Open Concurrent Design Tool) è uno strumento che implementa gli standard per la condivisione dei dati definiti dall'ECSS. Questo è un pacchetto software distribuito con architettura client/server, per consentire ingegneria collaborativa multidisciplinare nella produzione dei sistemi spaziali, fondamentale nelle prime fasi del loro ciclo di vita [20]. I dati richiesti sono quindi esportati all'interno di fogli di calcolo, essendo uno degli strumenti più utilizzati dall'industria per la gestione dati.

Di conseguenza, il client OCDT è realizzato tramite un semplice plugin per Microsoft Excel, un software dedicato alla produzione e alla gestione di fogli di calcolo elettronici. Viene utilizzato per eseguire semplici operazioni di analisi e simulazione. Anche altri strumenti possono essere integrati con OCDT, grazie

all'uso degli adattatori OCDT. D'altra parte, Il server OCDT è composto da un front-end web che usa delle REST API e un back-end PostgreSQL, ovvero un DBMS (Database Management System) per la memorizzazione persistente dei dati condivisi.

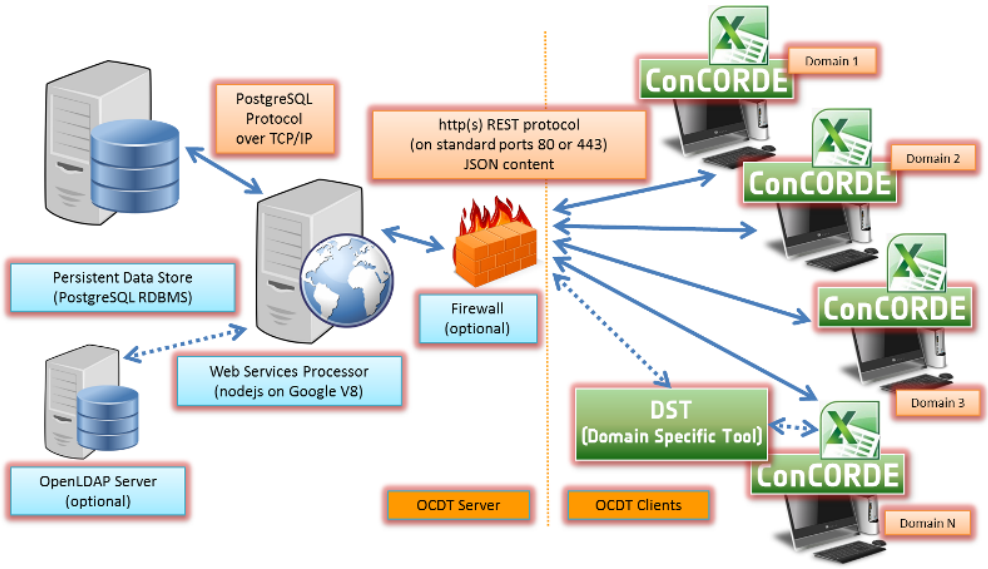


Figura 2.7: Panoramica schematica dell'architettura di OCDT [20]

I dati inseriti sono relativi a tutti i sistemi che compongono uno specifico modulo spaziale (es. massa, quantità, caratteristiche dei materiali, ecc.) e vengono usati principalmente in fase di progettazione per effettuare diversi calcoli (es. calcolo di consumi, massa totale, ecc.), ma anche e soprattutto per la revisione, verifica e validazione dei sistemi.

2.3.1 Requisiti ed obiettivi

La prima parte del lavoro di tesi si concentra nella progettazione e sviluppo di nuovo modulo per il plugin V4U di TASI, che permetta l'importazione e l'esportazione, in modo automatico, dei dati ottenuti tramite OCDT, all'interno del motore Unreal Engine. Come detto precedentemente, una volta aver interrogato il database centrale, questi vengono raccolti in fogli di calcolo. L'obiettivo è la creazione di un sistema, che tramite una semplice interfaccia grafica, permetta di associarli automaticamente ai modelli virtuali (che rappresentato i veri componenti di un sistema spaziale) di una scena costruita all'interno di UE4.

Generalmente, infatti, una volta progettato un modulo con certe dimensioni, componenti e parametri, ed aver ricevuto le prime revisioni, è possibile creare un ambiente virtuale che lo simuli. Tramite l'utilizzo della Realtà Virtuale (con

dispositivi di input e output adatti), è possibile mostrare e far provare l'ambiente sviluppato a coloro che dovranno valutare il lavoro, mostrando gli spazi e le dimensioni direttamente tramite il concetto di immersione, ma anche i dati dei vari componenti (precedentemente importati grazie al nuovo plugin) tramite un'interfaccia grafica.

2.4 Utilizzo di manichini realistici

Come spiegato nel paragrafo 1.2.2 di questa tesi, il design ergonomico dei moduli spaziali è fondamentale nel settore. In particolare, per migliorare i processi tradizionali si creano ambienti virtuali, che riproducono i reali sistemi, nella quale vengono utilizzati dei manichini digitali che ne rappresentano gli utilizzatori.

Per la modellazione dei manichini è necessario seguire con precisione i dati forniti dal documento GP 10017 della NASA, ovvero il Gateway Program Human System Requirements (HSR), definito per il programma Gateway e pubblicato nel 2019, come parte del programma Artemis. Gateway sarà una stazione spaziale orbitante intorno alla Luna, la quale fornirà supporto all'essere umano per un ritorno a lungo termine sulla superficie lunare, oltre che un punto di sosta fondamentale per le future missioni su Marte e per l'esplorazione dello spazio profondo.

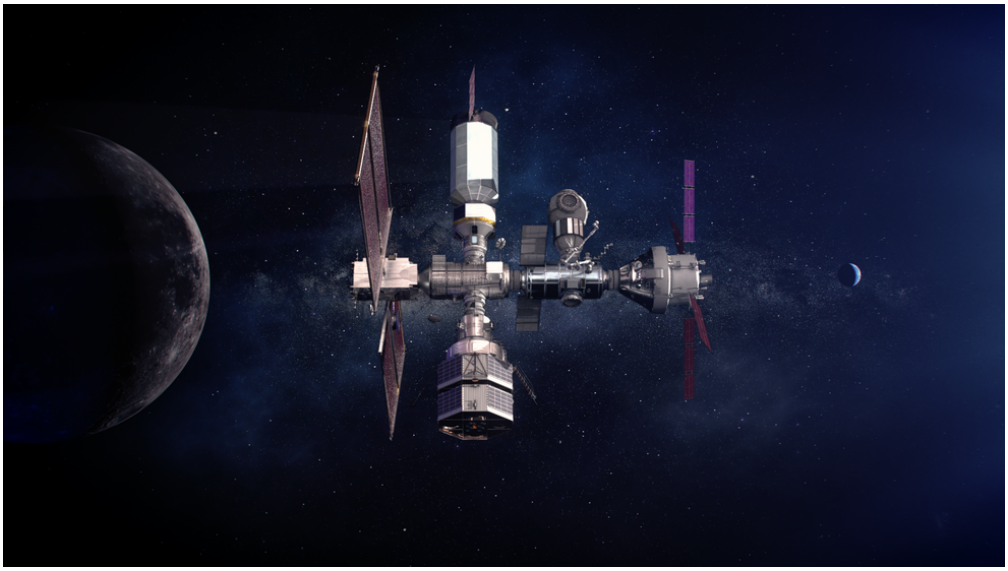


Figura 2.8: Illustrazione della stazione orbitante Gateway della NASA [21]

2.4.1 Standard NASA per le misure antropometriche

Il documento citato definisce i requisiti dei sistemi spaziali in rapporto agli essere umani che li utilizzeranno, specificando una serie di requisiti ergonomici ed antropometrici, i manichini devono rispettare per replicare correttamente le pose dei futuri reali astronauti.

In particolare, i dati dimensionali antropometrici sono stati definiti suddividendo le misure della popolazione in due percentili senza indicare un sesso, il 1° percentile e il 99° percentile. Un percentile in statistica rappresenta il valore minimo sotto al quale ricade una data percentuale degli elementi in osservazione. Per esempio, una misura dell'1° percentile indica che l'1% della popolazione avrà misura pari o minore a quelle indicata, e contemporaneamente, il 99% della popolazione avrà misure maggiori; lo stesso concetto si applica al 99° percentile.

In figura 2.9 si mostra un esempio delle molteplici misure definite, per un maggiore approfondimento si rimanda alla sezione 7.3.

2.4.2 Soluzioni attuali

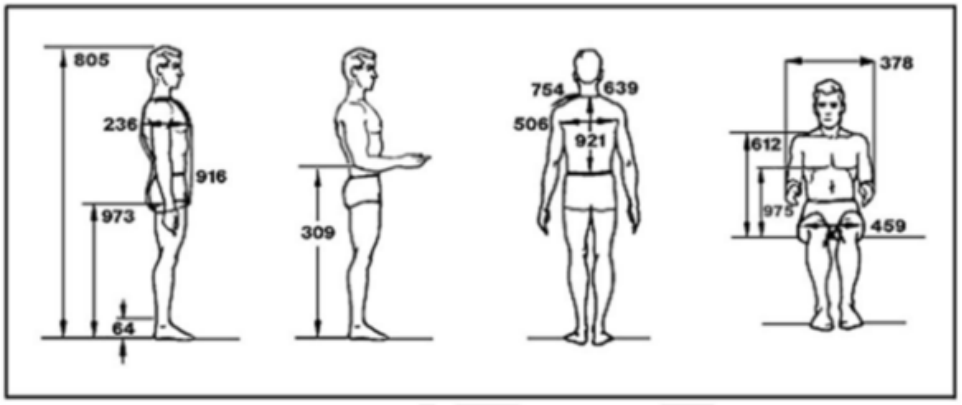
Attualmente TASI utilizza una serie di manichini sviluppati anche grazie a lavori di tesi precedenti a questo, gli umanoidi tuttavia presentano diverse problematiche.

Innanzitutto il processo di creazione non è scalabile, in quanto ogni singolo manichino richiede un processo molto lungo di modifica e revisione manuale, passando da diversi software 3D. Infatti i modelli sono inizialmente creati con lo strumento MakeHuman, un'applicazione open source per la creazione di prototipi umanoidi digitali. All'interno di esso è possibile impostare e definire le dimensioni del proprio manichino, tuttavia il risultato non è sempre soddisfacente. Nel caso del design ergonomico, i manichini possono risultare non troppo realistici e, soprattutto, non è possibile definire e impostare tutte le misure richieste dallo standard NASA.

Per questo, il modello è poi esportato all'interno di Blender, un software open source multi-piattaforma, utilizzato principalmente per modellazione, rigging, animazione e rendering. Il manichino viene quindi modificato manualmente in modo da soddisfare i vari requisiti, inoltre l'applicazione viene usata per verificare che le misure del modello siano effettivamente quelle impostate all'interno di MakeHuman.

Infine, il modello è importato all'interno di Unreal Engine. Anche questo processo risulta laborioso in quanto il rig (lo scheletro) del modello va reso compatibile con il rig standard di UE4. Per mettere in posa il manichino è poi necessario creare manualmente dei vari controlli dello scheletro. In generale, quindi, sono presenti diverse incompatibilità con il motore grafico.

Dato questo lungo processo, i manichini creati, e le singole misure di questi, non sono modificabili, spesso risultano poco realistici a causa di MakeHuman e le sembianze del modello sono permanenti, non permettendone l'adattamento.



No.	Dimension	Min (cm, (in))	Max (cm, (in))
805	Stature	148.6 (58.5)	194.6 (76.6)
973	Wrist height	70.3 (27.7)	96.3 (37.9)
64	Ankle height	4.8 (1.9)	8.1 (3.2)
309	Elbow height (rest height standing)	89.9 (35.4)	120.7 (47.5)
236	Bust depth (chest depth)	19.1 (7.5)	30.2 (11.9)
916	Vertical trunk circumference	134.9 (53.1)	181.9 (71.6)
612	Mid-shoulder height, sitting	52.6 (20.7)	71.1 (28.0)
459*	Hip breadth, sitting	31.5 (12.4)	46.5 (18.3)
921	Waist back	39.1 (15.4)	55.9 (22.0)
506	Interscye	29.3 (11.5)	48.0 (18.9)
639	Neck circumference	27.8 (10.9)	43.4 (17.1)
754	Shoulder length (side neck-to-acromion horizontal distance)	12.0 (4.7)	18.0 (7.1)
378	Forearm-forearm breadth	38.9 (15.3)	66.0 (26.0)
975	Chest height, sitting	33.8 (13.3)	50.3 (19.8)

*For seated measurements, the largest female hip breadth is larger than the largest male hip breadth, and the smallest male hip breadth is smaller than the smallest female hip breadth; therefore, male data are used for the Min dimension, and female data are used for the Max dimension.

Figura 2.9: Esempio delle misure antropometriche definite dallo standard NASA [22]

Ciononostante, il problema fondamentale che ha portato TASI ad un cambiamento dei propri manichini è il cambiamento dello standard NASA. I precedenti umanoidi erano infatti basati sul documento NASA-STD-3000 e il documento Human Integration Handbook, tuttavia in corrispondenza del nuovo programma Artemis e del programma Gateway, l'agenzia governativa americana ha deciso di aggiornare lo standard e le misure antropometriche, correggendo gli errori commessi con il precedente.

Figura 2.10: Manichini utilizzati attualmente da TASI (5° percentile donna, 95° percentile uomo)

2.4.3 Requisiti ed obiettivi

Il secondo modulo sviluppato in questa tesi consiste nella creazione di manichini altamente realistici e animabili facilmente, senza l'uso di programmi esterni. Infatti vengono utilizzati solo strumenti messi a disposizione da Unreal Engine stesso, come MetaHuman e i Blueprint Animation, entrambi affrontati nel capitolo 5. Il vantaggio principale deriva dalla conformità al nuovo standard NASA definito nel documento citato nei paragrafi precedenti.

Questo processo, rispetto a quello precedentemente esposto, permette di ottenere dei manichini altamente realistici e personalizzabili tramite MetaHuman, senza passare da programmi di modellazione esterni. Inoltre viene garantita la possibilità di animare il manichino stesso tramite il già esteso Control Rig creato da Unreal Engine per i propri umanoidi MH.

Dato che le misure antropometriche sono realizzate agendo direttamente sulle ossa dei manichini, è possibile utilizzare un qualsiasi modello creato tramite MetaHuman, in questo modo è possibile creare degli umanoidi che assumano le sembianze di reali utilizzatori dei moduli spaziali (come astronauti). Inoltre, grazie a questa soluzione, si ha la possibilità di creare velocemente nuovi percentili o modificare le singole misure dei modelli attualmente creati, nel caso in cui si volessero adattare a nuovi e diversi standard.

Parte II

Modulo 1

Capitolo 3

Strumenti utilizzati: Python in UE4

Python è il linguaggio di programmazione utilizzato maggiormente per la realizzazione del primo modulo.

Questo capitolo introduttivo viene diviso in due sezioni principali. Nella prima sezione viene presentato Python, mentre nelle sezioni successive si spiega come e perché il linguaggio viene integrato e supportato all'interno di Unreal Engine 4.

3.1 Python

Python è un linguaggio di programmazione interpretato, orientato ad oggetti [23]. Questo possiede un interprete, ovvero un programma in grado di eseguire altri programmi direttamente dal codice sorgente, senza la necessità di utilizzare un compilatore. La traduzione in linguaggio macchina viene infatti effettuata durante l'esecuzione del programma stesso [24].

Essendo un linguaggio di programmazione generico ad alto livello, Python viene utilizzato in molti diversi ambiti [25].

Questo supporta diversi paradigmi di programmazione, oltre alla programmazione orientata ad oggetti, come la programmazione funzionale e procedurale. Possiede diversi moduli e interfacce per molte librerie e system call, oltre ad essere estendibile tramite C e C++. Inoltre, essendo un linguaggio portabile, può essere eseguito in molte varianti di Unix e su Windows [23].

3.1.1 Librerie utilizzate

Le librerie di Python utilizzate per lo sviluppo di questo primo modulo sono *tkinter* e *csv*.

TKinter TKinter è un wrapper delle librerie grafiche Tcl\TK, le quali permettono la gestione di interfacce grafiche. Il pacchetto è ampiamente supportato dalla maggior parte dei sistemi operativi ed è composto da diversi moduli.

Permette di creare finestre, aggiungere bottoni, testo ed in generale, la maggior parte degli elementi di un'interfaccia grafica utente. Nello specifico caso di questa tesi, la libreria è utilizzata per generare delle finestre di selezione, per l'importazione e l'esportazione di file posizionati nella memoria del computer.

Nel caso in cui si voglia approfondire il funzionamento della libreria tkinter, si consiglia la lettura della documentazione ufficiale di Python [26].

csv Il modulo csv di Python permette di gestire il cosidetto formato csv, ovvero il formato più diffuso per la gestione di fogli di calcolo e database. La libreria è composta da classi per la lettura e scrittura di dati tabellati nel formato CSV.

Bisogna precisare che esistono diverse versioni di questo formato, in quanto non esiste uno standard ufficiale che lo definisce; esistono comunque delle linee guida che si possono rispettare. Il modulo di Python permette di definire il tipo di formato CSV che si vuole utilizzare, sia per l'importazione che per l'esportazione.

Viene data la possibilità ai programmatori di leggere o scrivere i dati sotto forma di dizionario, tramite l'utilizzo delle specifiche classi *DictReader* e *DictWriter*, le quali sono ampiamente utilizzate all'interno dei questa tesi.

Per maggiori informazioni si consiglia la lettura della documentazione ufficiale di Python [27].

3.2 Supporto per Unreal Engine 4

Recentemente, Unreal Engine ha cominciato il supporto di Python direttamente all'interno di esso. Questo perché negli ultimi anni, il linguaggio di programmazione è diventato uno standard per le pipeline di produzione e per l'interoperabilità di applicazioni 3D. La diffusione è dovuta principalmente grazie al gran numero di applicazioni che lo supportano e alla necessità di avere un unico linguaggio comune tra queste [28].

Python viene utilizzato in UE4 per automatizzare diversi tipi di flussi di lavoro all'interno dell'editor di dell'engine stesso [28]. Può essere sfruttato per facilitare il lavoro di figure appartenenti a diversi ambiti lavorativi, come: artisti, level designer, game designer e programmatori. Tuttavia, non può utilizzato come linguaggio di scripting dell'applicazione che si sta sviluppando con UE4, ma solo per sviluppare funzionalità di supporto legate all'editor del game engine.

La versione utilizzata attualmente da UE4 è Python 3.7.7. Questa è direttamente integrata all'interno di un plugin ufficiale di Epic Games. Di conseguenza non sarà necessario installare Python separatamente nella propria macchina [28].

3.2.1 Abilitare Python in un progetto

Per abilitare Python in un progetto di Unreal Engine 4, Epic Games ha sviluppato un plugin che contiene una versione ufficiale di Python, chiamato *Python Editor Script Plugin*. Il pacchetto può essere abilitato dal menù *Plugins* di UE4, a cui segue un riavvio necessario dell'editor.

È consigliabile abilitare un secondo plugin chiamato *Editor Scripting Utilities*, il quale offre delle semplici API per l'esecuzione di molte operazioni comuni nell'editor di UE4.

Per una guida più dettagliata, consultare la bibliografia [28].

3.2.2 Eseguire script in Python

Dopo aver abilitato i plugin sopracitati, è possibile eseguire uno script in Python in diversi modi. I principali vengono presentati di seguito:

- Output Log: tramite il pannello di output di UE4, è possibile selezionare la console di Python per eseguire direttamente linee di codice o inserire il percorso di uno script che si vuole eseguire;
- File *init_unreal.py*: l'editor di UE4 individua automaticamente un file Python chiamato in questo modo, il quale viene immediatamente eseguito. Questo metodo viene spesso utilizzato per eseguire codice di inizializzazione durante l'avvio dell'editor;
- Script di avvio: similmente al metodo precedente, tramite la finestra *Project Settings*, nella sezione *Python*, è possibile inserire manualmente una lista di script che si vogliono eseguire all'avvio dell'editor;
- Blueprint: l'esecuzione di script Python è possibile anche tramite l'utilizzo di nodi utilizzabili nell'ambiente di scripting visivo di UE4, ovvero il sistema dei Blueprint.

Per maggiori dettagli riguardo l'esecuzione di script Python all'interno dell'editor di UE4, consultare la bibliografia [28].

3.2.3 Libreria *unreal*

Epic Games mette a disposizione la libraria *unreal*, importabile su qualsiasi script Python che si vuole eseguire su UE4. Questa consiste in una serie di API che permettono di interagire con l'editor. Tra le funzionalità disponibili tramite la libreria ci sono:

- Interazione con l'editor.

- Interazione con il livello: attualmente attivo nell'editor. Tramite questo è possibile manipolare attori e componenti posizionati all'interno dell'ambiente virtuale.
- Interazione con asset: del progetto attualmente aperto. Tramite questo è possibile manipolare asset e oggetti che sono stati caricati all'interno del progetto, dando la possibilità di creare, modificare e eliminare asset.

Per una visione approfondita delle API disponibili nella libreria *unreal*, si consiglia la consultazione della documentazione ufficiale [29].

Capitolo 4

Sviluppo del modulo per la gestione dati di componenti aerospaziali

4.1 Obiettivo

L'obiettivo di questo primo modulo è la realizzazione di un plugin per la gestione di dati informativi, i quali sono relativi a mesh inserite all'interno di applicazioni 3D, create tramite Unreal Engine 4.

Come detto precedentemente, Thales Alenia Space sfrutta il motore grafico UE4 per la creazione di ambienti virtuali per fini di simulazione e progettazione di sistemi in contesto aerospaziale. All'interno degli ambienti virtuali sono inserite delle mesh corrispondenti a reali componenti fisici di sistemi aerospaziali. Ogni mesh è caratterizzata da una serie di misure fisiche, le quali sono generalmente gestite, condivise e modificate tramite dei fogli di calcolo.

Questo modulo si pone come obiettivo la realizzazione di funzionalità di importazione ed esportazione dei dati delle mesh, in modo che l'intero processo risulti veloce ed efficace. Nello specifico, il sistema deve permettere l'importazione automatica di dati a partire da dei fogli di calcolo, l'esportazione automatica dei dati e la visualizzazione dei dati all'interno dell'applicazione 3D.

Le informazioni da gestire sono coppie di chiavi-valore, dove la chiave è indicata da una stringa, mentre i valori sono di tre tipi: valori interi, valori decimali e stringhe.

4.2 Sviluppo del modulo di gestione dati

Per la realizzazione di questo plugin si è ampiamente utilizzato Python, integrato all'interno di UE4. Se si vuole approfondire l'integrazione del linguaggio di programmazione nel game engine, si consiglia la lettura del capitolo 3.

4.2.1 Creazione del componente

Il primo passo consiste nella creazione di un component personalizzato, il quale va aggiunto alle mesh per implementare le funzionalità del modulo. Come classe padre del component è stata creata una classe C++ chiamata *ThalesWidgetComponent*. Quest'ultima estende a sua volta la classe *WidgetComponent* di UE4, la quale si riferisce al Widget Component di UE4 [30]. Questo particolare componente permette la visualizzazione di un Widget durante l'esecuzione dell'applicazione 3D, di conseguenza si adatta perfettamente agli scopi di questa tesi.

Il component personalizzato, che come detto precedentemente estende la classe *ThalesWidgetComponent*, è stato creato come una Blueprint Class, questo permette di sfruttare il sistema di scripting visivo di UE4. Per la memorizzazione dei dati, è necessario avere una struttura dati in grado di contenerli. Per questo si è optato per una serie di mappe chiave-valore, nelle quali la chiave è una variabile di tipo String, che permette di descrivere il nome del parametro. Di conseguenza sono state aggiunge 4 variabili fondamentali:

- *ID*: un id univoco del component, generato automaticamente per evitare collisioni di dati tra mesh diverse. È stata lasciata la possibilità di modificarlo manualmente, in modo da adattarsi ad eventuali esigenze diverse. La generazione automatica dell'ID è esplicata nel paragrafo 4.2.2;
- *IntegerMap*: mappa che permette di contenere parametri di tipo intero.
- *FloatMap*: mappa che permette di contenere parametri di tipo decimale.
- *StringMap*: mappa che permette di contenere parametri di tipo stringa.

Come detto precedentemente, il nuovo component definito permette la visualizzazione di un Widget. I widget di UE4 consentono di gestire la visualizzazione dell'interfaccia grafica utente [31]. In questo caso è stato creato un widget, chiamato *W_Thales_ShowData*, che permette la visualizzazione del parametro *ID*. Si vuole precisare che il widget creato è solo un segnaposto, per dimostrare il funzionamento del plugin. Può quindi essere sostituito con un widget più curato.

Il component creato è quindi memorizzato, insieme al widget, all'interno della cartella Content del progetto, con il nome *BP_Thales_Data*.

4.2.2 Aggiunzione del Component

Per aggiungere il component creato nella precedente sezioni alle mesh dell'ambiente virtuale, è stato creato uno specifico script in Python chiamato *AddComponent.py*. Questo ottiene la lista di tutti gli attori attivi nel livello attualmente attivo nell'editor di UE4, tramite la libreria *unreal*. Dopodiché si procede e selezionare tutti quegli attori che possiedono una mesh statica, infine ad ognuno di questi viene aggiunto il component creato precedentemente.

L'aggiunzione del component non è un'operazione banale, infatti la libreria di UE4 non fornisce sufficienti strumenti a riguardo. Di conseguenza questa funzionalità è stata divisa in due funzioni, una all'interno dello script in Python ed una all'interno di una classe C++, la quale può sfruttare tutti gli strumenti messi a disposizione da Epic Games. Infatti, qualsiasi classe creata all'interno di un progetto di UE4 (o importata tramite un plugin esterno) è accessibile attraverso la libreria *unreal* automaticamente.

La classe creata in C++ si chiama *ComponentsHelper*, che eredita dalla classe *UBlueprintFunctionLibrary*, la quale permette di definire funzionalità aggiuntive per le classi blueprint. All'interno dio questa viene definito un singolo metodo, chiamato *AddCustomComponent*, che permette di aggiungere un component ad un attore specificato:

```
1 void UComponentsHelper::AddCustomComponent
2         (AActor* a, USceneComponent* component)
3 {
4     a->AddInstanceComponent(component);
5     component->RegisterComponent();
6     a->InitializeComponents();
7 }
```

La funzione prende come input un attore e un componente da aggiungere, il quale è stato creato tramite lo script Python. L'istruzione alla riga 4 aggiunge l'istanza del component all'array dei component dell'attore. La riga successiva (riga 5) si occupa della registrazione del componente, questa è un'operazione fondamentale che serve per informare l'Engine dell'esistenza del component, in modo che questo possa essere aggiornato ad ogni frame e possa influenzare la scena attualmente attiva [32]. Infine, il componente è inizializzato tramite l'istruzione della riga 6, la quale inizializza tutti i component dell'attore indicato.

L'ultima operazione effettuata all'interno dello script è l'impostazione della proprietà *ID*. Il valore viene generato ottenendo il nome dell'attore, il quale è univoco ed impostato automaticamente da UE4, concatenato con la stringa "ID_". Il risultato quindi è simile al seguente "ID_NomeAttore". Come detto nel paragrafo 4.2.1, il parametro può essere successivamente modificato manualmente.

4.2.3 Importazione dei dati

Per l'importazione dei dati si utilizza un secondo script denominato *Import.py*. Per quanto riguarda il formato del file e del suo contenuto sono state formulate diverse soluzioni. Il formato scelto è il CSV, in quanto ampiamente supportato dalla maggior parte dei software di gestione di fogli di calcolo. È stata lasciata la possibilità di estendere i formati supportati.

Per quanto riguarda il formato del contenuto dei fogli di calcolo è stata scelta una semplice struttura con 4 attributi:

- *ID*: del component, quindi della mesh all'interno della scena di UE4;
- *Property*: il nome della proprietà;
- *Type*: il tipo del dato, utilizzato per semplificare il parse del file;
- *Value*: dato vero e proprio.

Per semplificare le operazioni che l'utente deve effettuare per caricare un file, una volta eseguito lo script, questo apre una finestra di selezione file. Tramite questa l'utente può cercare e selezionare il file CSV da caricare. La finestra è stata realizzata tramite la libreria *tkinter* di Python.

Dopo aver ottenuto i dati dal foglio di calcolo, si ottiene la lista di tutte le mesh attualmente attive nella scena. Dopodiché si procede con la lettura dei dati, questo viene fatto tramite la classe *DictReader* della libreria *csv*. Questa classe permette di leggere i dati sotto forma di dizionario, la quale semplifica la scrittura del codice, ma anche la leggibilità di quest'ultimo.

Per ogni riga, si identifica l'ID e si cerca la mesh ad esso corrispondente. Se la mesh non possiede il component *BP_Thales_Data*, allora questo viene aggiunto. Infine si procede con l'aggiunzione della proprietà e del dato al component stesso. Dipendentemente dall'attributo *Type*, si aggiunge un nuovo elemento alla corrispettiva mappa, come indicato nel paragrafo 4.2.1. L'aggiunzione della proprietà viene realizzata tramite la funzione *set_editor_property* della libreria *unreal*.

4.2.4 Esportazione dei dati

L'esportazione dei dati è realizzata tramite due distinti script in Python, i quali hanno due funzionalità diverse. Lo script *Export.py* permette di esportare tutti i dati contenuti nei vari component associati alle mesh della scena, mentre lo script *ExportIDs.py* permette di esportare solo gli ID delle varie mesh. L'utilità principale di questo secondo script è la possibilità di generare un foglio di calcolo base, da riempire con le proprietà relative ai vari oggetti della scena.

In generale, il funzionamento è molti simile per entrambi gli script. Inizialmente si ottene la lista di tutte le mesh attualmente attive nella scena. Dopodiché, per

ognuna di queste si controlla se queste possiedono il component *BP_Thales_Data* o meno. In caso positivo, si estraggono i dati e si inseriscono all'interno di un foglio di calcolo. Il foglio di calcolo è creato tramite la classe *DictWriter* della libreria *csv*, che permette di scrivere i dati sotto forma di dizionario. Come detto prima, questo semplifica la scrittura e la leggibilità del codice.

4.2.5 Libreria *thales*

Dopo aver sviluppato i vari script descritti in questa sezione, è stata definita una libreria denominata *thales*, la quale racchiude tutte le funzionalità principali in comune tra i vari file. Inoltre sono incluse alcune costanti come il percorso della classe Blueprint del component e le stringhe che identificano i nomi dei parametri nel component stesso, utilizzate per aggiornarle o per ottenere i dati che memorizzano.

La definizione di questo modulo permette di ottenere diversi vantaggi. Si riduce la ridondanza del codice, migliorando la manutenibilità, dando la possibilità di effettuare modifiche e aggiornamenti al codice.

Inoltre la libreria potrà essere utilizzata per estendere il plugin o come base per un progetto con scopi diversi.

4.3 Creazione del plugin

Una volta aver sviluppato tutte le varie funzionalità e script del modulo, si è proseguito con la creazione di un pacchetto (o plugin) per Unreal Engine 4. In questo modo l'intero modulo potrà essere trasferito e abilitato facilmente in nuovi o vecchi progetti.

Il primo passo consiste nella creazione di un plugin base di UE tramite l'apposita interfaccia messa a disposizione. Il nome scelto del plugin è "*Thales Mesh Data Manager*", mentre come template base è stato utilizzato l'Editor Toolbar Button, inoltre è stata scelta una icona personalizzata. Nel caso in cui si volesse approfondire il concetto di plugin e come questi sono creati e gestiti nel motore grafico, si rimanda al paragrafo 1.3.5 di questa tesi. Il pacchetto è quindi visualizzato nella lista dei plugin di Unreal Engine nel seguente modo (figura 4.1).

Figura 4.1: Plugin *Thales Mesh Data Manager* nella lista plugin di Unreal Engine

Il risultato dell'operazione è mostrato nella successiva figura (4.2), ovvero un semplice bottone che non esegue alcuna operazione. Bisogna quindi definire l'interfaccia grafica e i bottoni necessari ad eseguire i molteplici script in Python. In particolare, l'obiettivo è la creazione di un menu a tendina collegato al bottone principale creato con il plugin. Il menu dovrà contenere i quattro bottoni utili all'esecuzione dei quattro script definiti nel capitolo precedente, ovvero:

- Aggiungere il componente per la gestione dati alle mesh selezionate;
- Esportare tutti i dati all'interno di un foglio di calcolo;
- Esportare gli id delle mesh all'interno di un foglio di calcolo;
- Importare dei dati da un foglio di calcolo, all'interno dei componenti.

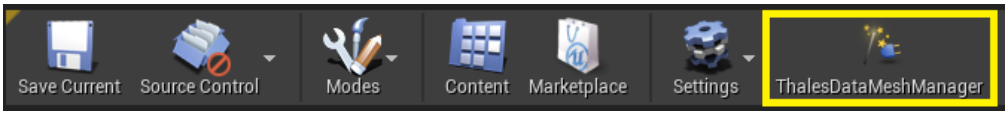


Figura 4.2: Bottone base del plugin visualizzato nella barra degli strumenti di Unreal Engine

4.3.1 Sviluppo dell'interfaccia grafica

Per la realizzazione dell'interfaccia grafica, bisogna prima studiare le classi create da Unreal Engine:

- *ThalesDataMeshManagerModule*: esegue le funzioni principali utili all'interfaccia grafica come le funzione di inizializzazione e funzione di chiusura. Inoltre viene usata per definire nuove componenti dell'UI come bottoni e menu;
- *ThalesDataMeshManagerCommands*: usato per registrare i comandi associati ai diversi bottoni dell'interfaccia grafica;
- *ThalesDataMeshManagerStyle*: permette di modificare lo stile della UI.

Inizialmente bisogna registrare i nuovi quattro comandi all'interno della funzione *RegisterCommands* della classe *ThalesDataMeshManagerCommands*, utilizzando la funzione *UI_COMMANDS*. Per ogni bottone, è stato registrato un comando con un id univoco.

```
1 UI_COMMAND(
```

```
    AddComponentAction,
    "Add Component to Selected Actors",
    "Add the Data Component to the Selected Actors",
    EUserInterfaceActionType::Button,
    FInputGesture());
```

I paramatri richiesti sono:

- Id del comando;
- Test visualizzato nel bottone;
- Descrizione del bottone;
- Tipo di comando;
- Input Chord: un tasto e i modificatori ad esso associati.

Le istruzioni da far eseguire ai comandi sono definite all'interno della classe *ThalesDataMeshManagerModule*, definendo delle nuove funzioni. Per avviare l'esecuzione degli script Python si utilizza il puntatore globale all'engine (*GEngine* [33]), accedendo alla funzione *Exec* [33], utilizzata per l'esecuzione di un comando.

```
1 void FThalesDataMeshManagerModule::AddComponent()
2 {
3     GEngine->Exec(NULL, TEXT("py AddComponent.py"));
4 }
```

Dopo aver fatto ciò, è necessario effettuare l'associazione tra il comando creato e la funzione da eseguire, grazie all'uso della funzione *MapAction* [34] della classe *FUICommandList* [34], che mantiene la lista di tutti i comandi del modulo. In particolare, è importante che la funzione venga eseguita all'interno della funzione di inizializzazione del modulo, ovvero *StartupModule* [35] di *ThalesDataMeshManagerModule*.

```
1 PluginCommands = MakeShareable(new FUICommandList);
2
3 PluginCommands->MapAction(
      FThalesDataMeshManagerCommands::Get().AddComponentAction,
      FExecuteAction::CreateRaw(
          this, &FThalesDataMeshManagerModule::AddComponent));
```

Il prossimo passo consiste nella registrazione del bottone da inserire nella barra degli strumenti. Questo è fatto all'interno della funzione *RegisterMenus* [36] di *ThalesDataMeshManagerModule*, creando un estensore dell'interfaccia grafica. Con

questo è possibile estendere la barra di Unreal Engine e indicare una nuova funzione per l'inizializzazione di questo (in questo caso *AddToolbarButton*).

```
TSharedPtr<FExtender> NewToolbarExtender =
    MakeShareable(new FExtender);

NewToolbarExtender->AddToolBarExtension("Settings",
    EExtensionHook::After,
    PluginCommands,
    FToolBarExtensionDelegate::CreateRaw(this, &
    FThalesDataMeshManagerModule::AddToolbarButton));

LevelEditorModule.GetToolBarExtensibilityManager()->AddExtender(
    NewToolbarExtender);
```

Dopo aver registrato il bottone, è importante creare il menu con cui accedere ai vari comandi. A tale scopo viene sfruttata la funzione di inizializzazione *AddToolbarButton* del bottone precedentemente creato. Questo viene fatto aggiungendo un *ComboButton*, piuttosto che un normale *Button*, con la quale poter aprire il menu a tendina. Grazie alla funzione *AddComboButton* [37] del Builder di Unreal Engine (strumento utile alla costruzione dell'interfaccia grafica) è possibile fare questo, definendo il nome da visualizzare, la descrizione, l'icona e una funzione di inizializzazione (*FillComboButton*) che permetta di creare il contenuto nel menu a tendina.

```
Builder.AddComboButton(
    TempCompileOptionsCommand,
    FOnGetContent::CreateRaw(
        this,
        &FThalesDataMeshManagerModule::FillComboButton,
        PluginCommands),
    LOCTEXT("ComboButton", "Mesh Data"),
    LOCTEXT("ComboButtonTootlip", "Actions to handle mesh data"),
    FSlateIcon(
        FThalesDataMeshManagerStyle::GetStyleSetName(),
        TEXT("ThalesDataMeshManager.MenuButton")),
    false
);
```

Infine, tramite la funziona sopracitata (*FillComboButton*) è possibile inserire i quattro bottoni, associati ai quattro comandi che permettono di eseguire gli script in Python. In particolare si fa uso della funzione *AddMenuEntry* del *MenuBuilder* [38] di Unreal Engine.

```
1 FMenuBuilder MenuBuilder(true, Commands);
2 MenuBuilder.AddMenuEntry(
      FThalesDataMeshManagerCommands::Get().AddComponentAction);
```

Lo stesso processo, presentato in questa sezione, viene ripetuto per tutti e quattro i comandi.

Il risultato ottenuto è un bottone con icona, con la quale poter aprire un menu a tendina. All'interno di questo è possibile interagire con i quattro bottoni che eseguono gli script creati nella sezione 4.2.1. Nella figura 4.3 è mostrato il risultato dell'interfaccia grafica all'interno di Unreal Engine.

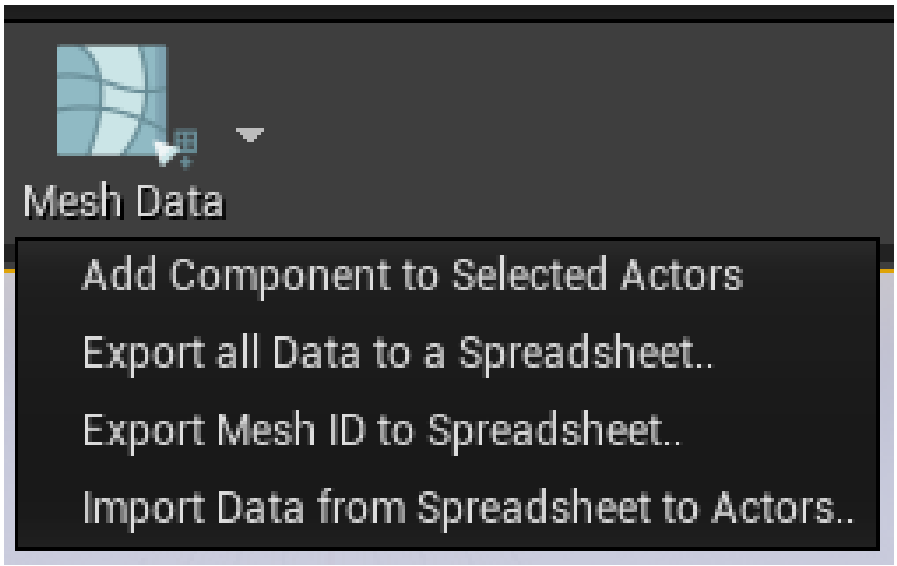


Figura 4.3: Interfaccia grafica del plugin *ThalesDataMeshManager*

Parte III

Modulo 2

Capitolo 5

Strumenti utilizzati: Animation Blueprint, Control Rig e MetaHuman

In questo capitolo introduttivo della Parte 3 sono presentate una serie di tecnologie e strumenti ampiamente utilizzati per lo sviluppo dei manichini. In particolare, gli strumenti presentati sono tutti appartenenti al motore grafico Unreal Engine per la creazione, gestione e animazioni di manichini virtuali all'interno dell'engine stesso.

5.1 Animation Blueprint

L'*Animation Blueprint* (AB) è uno speciale Blueprint (paragrafo 1.3.3) di Unreal Engine che permette di controllare le animazioni di uno scheletro, ovvero una gerarchia di ossa che consente di deformare la mesh a cui è associata. Nella computer grafica, una mesh poligonale corrisponde ad un reticolo di punti o vertici, con la quale si definisce un oggetto in uno spazio tridimensionale.

Ad ogni osso viene collegato un insieme di vertici della mesh. Modificando le proprietà dell'osso (come posizione, orientamento e scala), la porzione delle mesh ad esso associato viene deformata.

Tramite un Animation Blueprint è possibile effettuare fusione di animazioni, controllare direttamente le ossa di uno scheletro e creare logica di controllo per determinare la posizione finale dello scheletro, tutto questo per ogni frame dell'applicazione [39].

5.1.1 Composizione dell'editor

L'editor dell'Animation Blueprint è molto simile a quello di un classico Blueprint. Per maggiore chiarezza se ne indicano le componenti principali [40]:

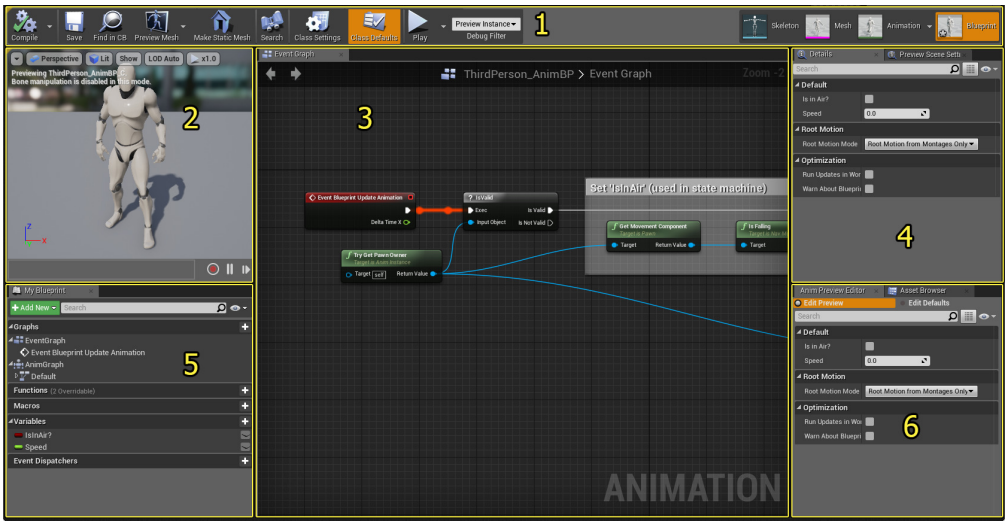


Figura 5.1: Interfaccia grafica di un Animation Blueprint [40]

- *Toolbar* (1): permette di compilare e salvare l'AB e gestire le impostazioni della classe. Inoltre è possibile spostarsi tra gli strumenti per la gestione delle animazioni, messi a disposizione da UE4;
- *Viewport* (2): questa finestra permette di riprodurre l'animazione definita tramite l'AB sullo scheletro ad esso associato;
- *Graph* (3): include due grafi diversi, l'*Event Graph*, che contiene i nodi usati per innescare l'aggiornamento della posa dello scheletro, e l'*Anim Graph*, che viene usato per determinare la posa finale dello scheletro al frame corrente. Entrambi sono utilizzati per definire la logica e la posa finale dello scheletro. I due grafi saranno approfonditi nei prossimi paragrafi (5.1.2, 5.1.3);
- *Details/Preview Scene Settings* (4): dal pannello Details è possibile modificare le proprietà dell'oggetto selezionato. Dal pannello Preview Scene Settings è possibile modificare le impostazioni di visualizzazione della finestra Viewport;
- *My Blueprint* (5): contiene la lista dei grafi, funzioni, macro, variabili e altre proprietà legate all'Animation Blueprint;
- *Anim Preview Editor/Asset Browser* (6): nella prima finestra è possibile modificare i valori delle proprietà che verranno usati per la riproduzione

dell'AB nel Viewport. Nella seconda finestra è possibile modificare i valori di default delle proprietà dell'AB.

5.1.2 EventGraph

Ogni Animation Blueprint possiede un singolo *EventGraph*, ovvero un grafo standard che contiene una serie di speciali eventi legati alle animazioni, usati per innescare l'esecuzione di una sequenza di nodi. L'uso comune di questo grafo è l'aggiornamento dei valori che verranno poi utilizzati per la fusione di animazioni, effettuato tramite l'AnimGraph (paragrafo 5.1.3) [41].

Gli eventi sono nodi che sono chiamati dal codice dell'applicazione per cominciare l'esecuzione della sequenza di nodi ad esso collegato.

Vari eventi sono chiamati dal sistema delle animazioni per inizializzare ed aggiornare l'Animation Blueprint. In particolare, i principali eventi sono due:

- *Blueprint Initialize Animation*: questo evento è eseguito una singola volta, quando l'istanza dell'Animation Blueprint è creata. Viene utilizzato per eseguire i nodi di inizializzazione;

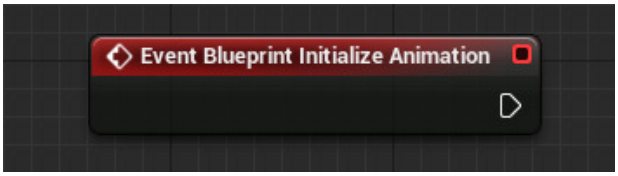


Figura 5.2: Nodo *Blueprint Initialize Animation* dell'EventGraph [41]

- *Blueprint Update Animation*: questo evento è eseguito per ogni frame, in questo modo è possibile calcolare ed aggiornare i valori delle proprietà dell'Animation Blueprint.

  Oltre al pin di output di esecuzione, possiede un pin (*DeltaTimeX*) che fornisce il tempo passato dall'esecuzione dell'ultimo update. Questo valore è utile per definire interpolazioni dipendenti dal tempo.

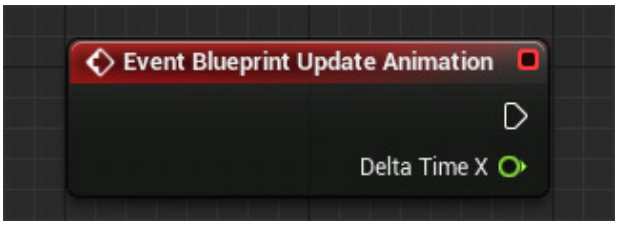


Figura 5.3: Nodo *Blueprint Update Animation* dell'EventGraph [41]

5.1.3 AnimGraph

L'*AnimGraph* è un grafo dell'Animation Blueprint usato per determinare la posa finale che assumerà lo scheletro per il frame corrente. Tramite gli *Animation Node* (nodi legati alle animazioni) è possibile eseguire diverse operazioni come controllare posizione, orientamento e scala delle singole ossa, effettuare la fusione di animazioni (*animation blending*) o anche eseguire una normale sequenza di animazioni. All'interno del grafo è possibile utilizzare i valori delle proprietà calcolati nell'EventGraph [42].

Nell'AnimGraph il flusso di esecuzione è rappresentato dalle pose che passano da un nodo ad un altro. Nell'esempio in figura 5.4 si utilizza un nodo Blend che prende due pose come input, dopodiché in funzione del valore Alpha si effettua la fusione (*blending*) delle due pose.

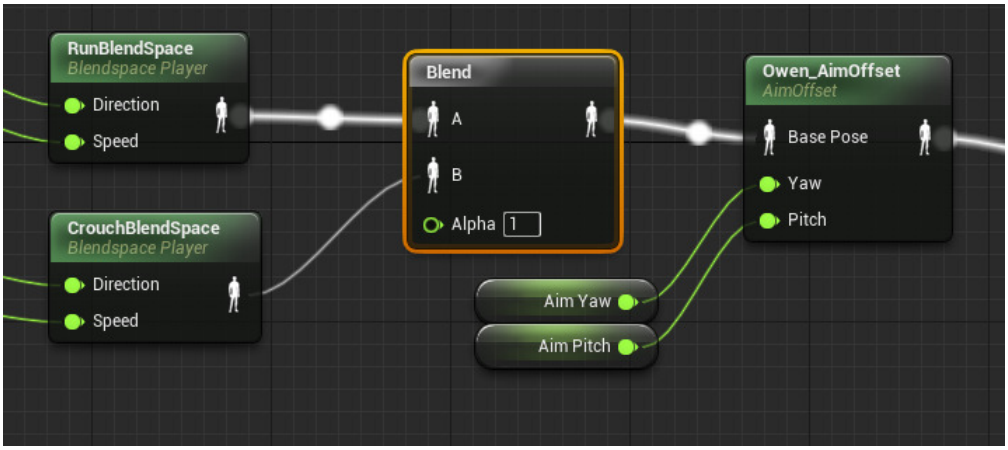


Figura 5.4: Esempio di un flusso di esecuzione di un AnimGraph [42]

Questo grafo è fondamentale per lo sviluppo dei manichini affrontato in questa parte della tesi. In particolare viene utilizzato per modificare i valori di posizione e scala delle singole ossa, in modo da poter deformare a proprio piacimento la mesh associata.

5.2 Control Rig

Il *Control Rig* (CR) è un sistema di rigging di Unreal Engine basato su nodi, il quale offre una serie di strumenti agli animatori per creare animazioni flessibili, dinamiche e procedurali. Il rigging è una tecnica di animazione che consiste nell'associazione di ossa e controlli ad un modello virtuale, con la quale controllare le deformazioni e le pose. Nel contesto di questa tesi, il sistema viene utilizzato per mettere in posa tramite cinematica diretta ed inversa i manichini virtuali creati.

Tramite il Control Rig è possibile creare e aggiungere dei controlli personalizzati agli scheletri, in modo da poter creare nuove animazioni o per modificarne esistenti, tutto all'interno dell'editor di Unreal [43].

Prima di poter utilizzare il CR all'interno di UE4, è necessario attivare il plugin ad esso associato dalla finesta dei plugin dell'engine.

Gli asset nella quale utilizzare il sistema Control Rig vengono chiamati Control Rig Blueprint.

5.2.1 Composizione dell'editor

L'editor di un Control Rig Blueprint è composto dalle seguenti sezioni [44]:

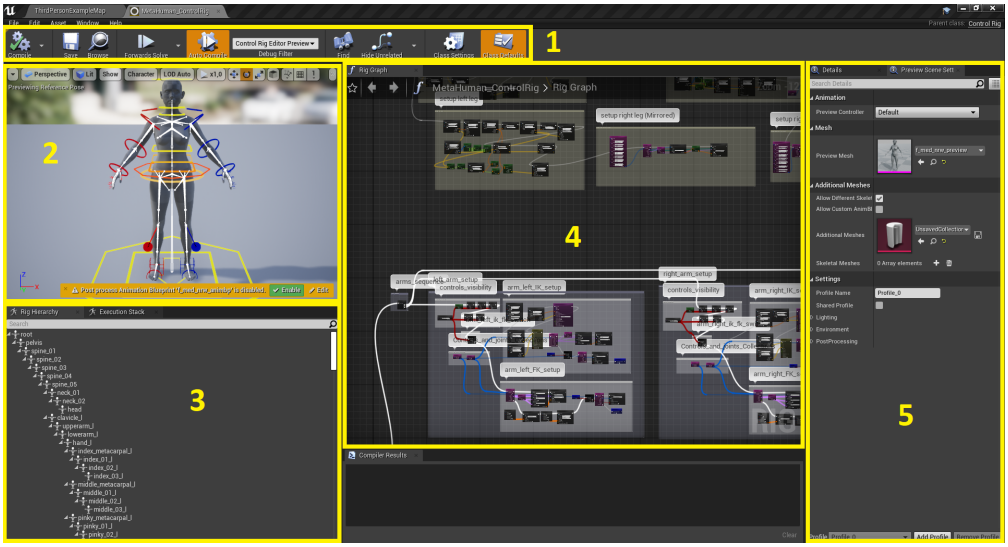


Figura 5.5: Interfaccia grafica di un Control Rig Blueprint [44]

- *Toolbar* (1): similmente all'editor di un normale Blueprint, permette di accedere alle funzionalità base necessarie durante l'editing. Sono quindi presenti bottoni per salvare, compilare e accedere alle impostazioni della classe;
- *Viewport* (2): similmente all'Animation Blueprint, permette di riprodurre l'animazione definita tramite il Control Rig;
- *Rig Hierarchy/Execution Stack* (3): la prima finestra permette di visualizzare la gerarchia dello scheletro e quella dei controlli inseriti. Viene utilizzata anche per la creazione di nuova ossa, controlli e spazi.

  La seconda finestra permette di visualizzare l'ordine con la quale le operazioni del grafo vengono eseguite;

- *Rig Graph* (4): rappresenta il grafo nella quale definire il comportamento del Control Rig tramite l'utilizzo della programmazione visiva, quindi grazie all'uso di nodi. Di conseguenza è molto simile all'AnimGraph di un Animation Blueprint;

- *Details/Preview Scene Settings* (5): dal pannello Details è possibile modificare le proprietà dell'oggetto selezionato. Dal pannello Preview Scene Settings è possibile modificare le impostazioni di visualizzazione della finestra Viewport.

5.3 MetaHuman

Lo strumento *MetaHuman* di Unreal Engine 4 fornisce un sistema per creare dei modelli umani virtuali altamente realistici tramite un'interfaccia intuitiva. La necessità di avere uno strumento di questo calibro nasce principalmente dall'industria dell'intrattenimento, in particolare i videogiochi e il cinema. Queste industrie spingono sempre di più per ottenere risultati visivamente realistici e animazioni fluide. Lo strumento in questione è in grado di creare umanoidi utilizzabili in applicazioni in tempo reale, mantenendo le stesse caratteristiche grafiche.

MetaHuman permette una vastissima personalizzazione dei propri manichini, il tutto disponibile grazie ad uno strumento web chiamato MetaHuman Creator. È possibile personalizzare volto, capelli, corpo e vestiario dei modelli.

5.3.1 MetaHuman Creator

Prima di spiegare nel dettaglio le funzionalità offerte dall'editor, si presentano le componenti principali dell'interfaccia grafica [45]:

- *Barra del titolo* (1): mostra il nome del MetaHuman creato;

- *Selezione di Attributi e Proprietà* (2): menù principale con la quale è possibile navigare tra le varie funzionalità e attributi messi a disposizione dall'editor. È possibile accedere alle diverse sezioni dello strumento per modificare volto, capelli e corpo;

- *Impostazioni del Viewport* (3): permette di modificare le imposizioni di illuminazione e qualità della finestra nella quale il MetaHuman è visualizzato;

- *Viewport* (4): finestra nella quale il MetaHuman è visualizzato e animato;

- *Riferimento alle hotkey* (5): mostra una serie di tasti di scelta rapidi da tastiera, per muoversi velocemente tra le finestre e funzionalità del MetaHuman Creator;

Figura 5.6: Interfaccia grafica del MetaHuman Creator [45]

- *Barra degli strumenti* (6): permette di selezionare le modalità di visualizzazione del viewport e di gestire l'animazione di anteprima del MetaHuman che si sta creando.

5.3.2 Personalizzazione del MetaHuman

Come detto in precedenza, l'editor di MetaHuman permette di modificare diverse caratteristiche dell'umanoide come il volto, i capelli e il corpo.

Volto La sezione dell'editor del volto contiene le seguenti funzionalità e attributi:

- *Blend*: lo strumento Blend viene utilizzato per effettuare una fusione (*blending*) di volti da una selezione di MetaHuman predefiniti;
- *Sculpt* e *Move*: due potenti strumenti che permettono di modificare punto per punto, la forma e composizione del volto umano. Tramite questi è possibile creare una vastissima moltitudine di umanoidi;
- *Proprietà della pelle*: permettono di modificare la composizione e il colore della pelle. È possibile aggiungere e modificare lentiggini del volto agendo su diversi parametri come densità, intensità, saturazione e colore. Infine, è possibile modificare il colore della pelle in particolari zone del volto (fronte, zigomi, guancia, labbra, mento), in modo da differenziarle leggermente dal colore base, modificando i parametri di arrossamento, saturazione e luminosità;

- *Attributi degli occhi*: il pannello fornisce una serie di occhi predefiniti da cui scegliere come punto di partenza, dopodiché è possibile agire su iride e sclera.

  Gli attributi dell'iride sono molteplici ed è possibile modificare gli occhi separatamente, se necessario. In particolare gli attributi riguardano il colore base, il colore dei dettagli dell'iride, la dimensione, la saturazione e il tipo di iride stesso (figura 5.7).

  Per quanto riguarda gli attributi della sclera sono disponibili parametri per modificarne il colore, brillantezza, rotazione e presenza di vene;

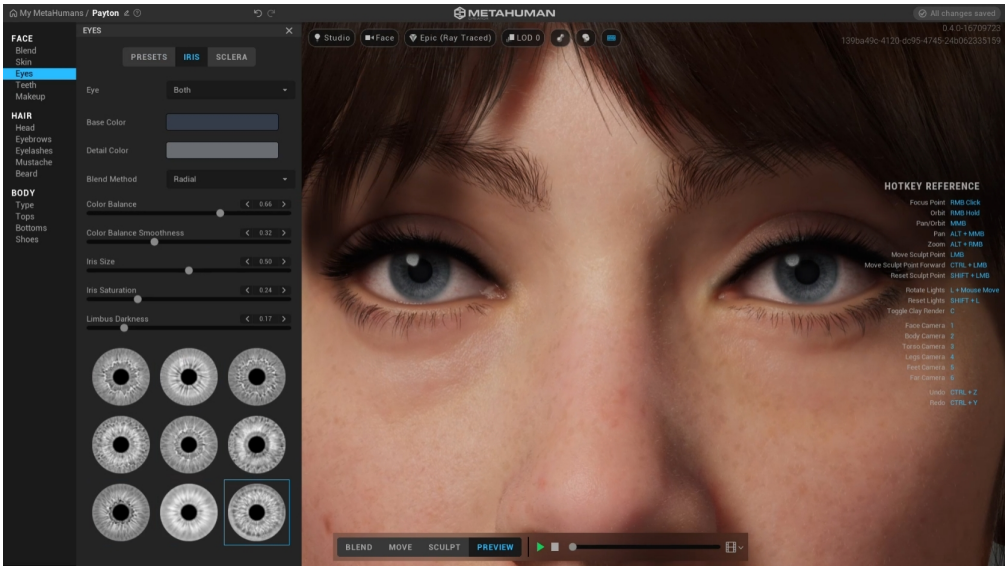


Figura 5.7: Editor degli occhi di MetaHuman

- *Proprietà dei denti*: permettono di modificare la forma e il colore dei denti. Per quanto riguarda la forma è possibile agire su lunghezza dei denti, spazio tra denti ed altre imperfezioni che rendono la dentatura più realistica. Inoltre è possibile modificare colore dei denti e della gengiva (figura 5.8);

- *Proprietà dei cosmetici*: permettono di aggiungere dei cosmetici agli occhi e alle labbra, agendo sulla forma e colore.

Capelli e peluria La categoria dei capelli e peluria di MetaHuman permette di modificare diverse proprietà: capelli, sopracciglia, ciglia, baffi e barba.

In tutte le sezioni è possibile agire su tipo e forma, colore, rugosità e percentuale di effetto brizzolato (figura 5.9).

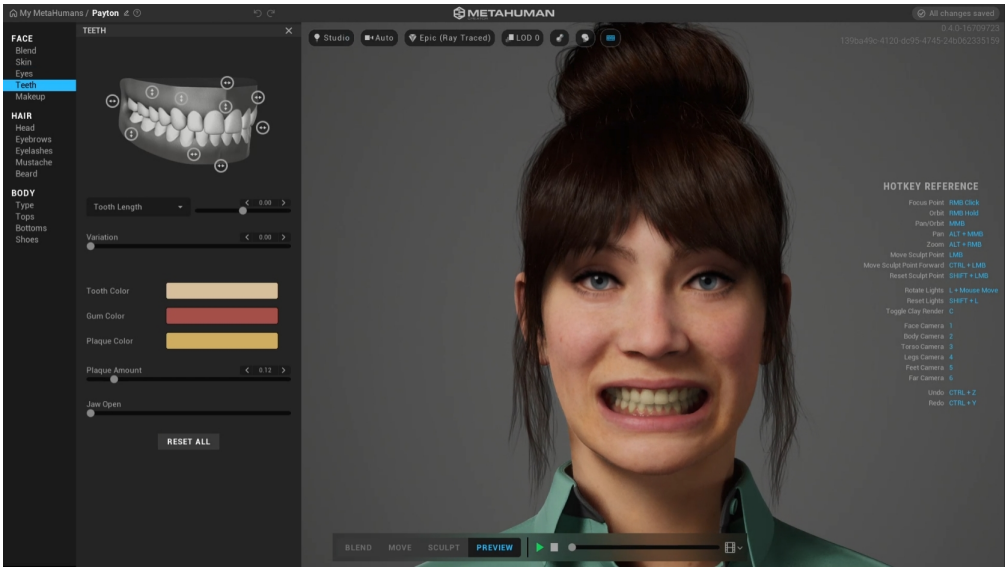


Figura 5.8: Editor dei denti di MetaHuman

Figura 5.9: Editor dei capelli di MetaHuman

Corpo La sezione del corpo permette di agire principalmente sul tipo del corpo e sul vestiario.

In particolare, per quanto riguarda il tipo, non è consentita molta personalizzazione, limitando alla scelta di sesso, altezza e peso, per un totale di 18 tipi di corpi. Data questa limitazione, per lo sviluppo dei manichini per il design ergonomico è

stato necessario modificare la composizione dello scheletro dei MetaHuman tramite l'Animation Blueprint.

5.3.3 Importare un MetaHuman su Unreal Engine

Il processo di importazione di un MetaHuman, creato tramite l'apposito editor, su Unreal Engine è abbastanza semplice. Per l'operazione è necessario essere in possesso di un account Epic Games e dell'applicazione Quixel Bridge.

Quixel Bridge è un'applicazione di Epic Games la quale permette di accedere alla libreria Quixel. La libreria citata contiene un elevatissimo numero di asset e scan per la creazione di ambienti virtuali realistici e, soprattutto, in tempo reale. L'intero pacchetto è gratuito per gli sviluppatori che utilizzano Unreal Engine.

L'applicazione Bridge funge inoltre da ponte tra MetaHuman e gli utenti, permettendo di ottenere, scaricare e esportare gli umanoidi creati in precedenza tramite il MetaHuman Creator. Una volta creato il proprio manichino, questo sarà disponibile all'interno dell'apposita sezione (figura 5.10).

Una volta selezionato l'umanoide, è possibile selezionare la risoluzione della texture. Tra le possibilità ci sono: 1K, 2K e 8K. Dopodiché, cliccando sul bottone "*Download*", l'intero pacchetto del MetaHuman viene scaricato nella propria macchina. Infine, cliccando il bottone "*Export*", il modello viene importato nel progetto di Unreal Engine attualmente aperto.

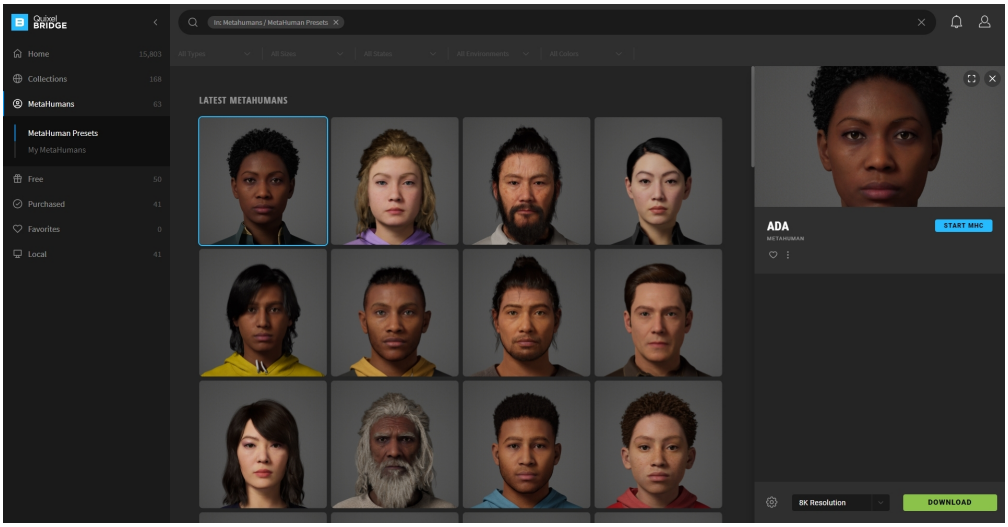


Figura 5.10: Sezione MetaHuman dell'applicazione Quixel Bridge

Capitolo 6

Sviluppo dello strumento di misurazione

6.1 Obiettivo

Dato lo sviluppo dei manichini in questa Parte 3 della tesi, è necessario utilizzare degli strumenti di misurazione in modo da verificare che le dimensioni degli umanoidi virtuali corrispondano alle misure definite dagli standard NASA.

L'obiettivo di questo capitolo è lo sviluppo di un sistema di misurazione vario, flessibile e facile da utilizzare, direttamente all'interno del motore grafico Unreal Engine 4. Lo strumento deve essere quindi un attore di UE4 inseribile all'interno di una scena, in modo da poter gestire i punti di controllo e visualizzare i valori delle misure effettuate (es. distanza, circonferenza).

Data la tipologia delle misurazioni si è inizialmente pensato di sviluppare due strumenti, la misura di distanze e la misura di circonferenze, a cui si è poi aggiunta la misura della circonferenza di ellissi. Quest'ultima è necessaria principalmente per la misurazione della circonferenza del collo dei manichini, il quale risulta non essere una circonferenza perfetta.

6.2 Sviluppo degli strumenti

Per la realizzazione degli strumenti si è scelto di utilizzare un Actor Blueprint, un particolare blueprint che permette di creare un Actor (o attore) ovvero un oggetto posizionabile all'interno di una scena virtuale che supporta le trasformazioni 3D (traslazione, rotazione e scala). Per un maggiore approfondimento si rimanda al paragrafo 1.3.4. Ogni Actor Blueprint creato possiede un SceneComponent che

permette di gestire le trasformazioni 3D ed è, generalmente, il component radice a cui vengono associati tutti gli altri.

Le classi blueprint sono quindi tre, per le tre differenti misure:

- *BP_Ruler*: per la misurare la distanza tra due punti;
- *BP_Ruler_Radius*: per misurare la circonferenza di un cerchio;
- *BP_Ruler_Ellipse*: per misurare la circonferenza di una ellisse.

Lo sviluppo, in comune per i tre blueprint, si divide in tre parti: il controllo dello strumento e il calcolo della misura, la visualizzazione della misura e la visualizzazione di una mesh che rappresenti la misura (es. circonferenza, elisse o linea che collega i due punti). Questo viene fatto tramite la definizione di tre funzioni (*CalculateMeasure*, *UpdateText*, *UpdateMesh*), cui esecuzione viene legata a quella dell'evento *Construction Script*.

L'evento citato viene chiamato dell'editor ogni volta che si effettua una modifica all'istanza della classe (es. cambio posizione dei punti di controllo), quindi risulta adatto per l'aggiornamento del blueprint. Ad esso sono legate le tre funzioni precedentemente citate, avendo un flusso di esecuzione come quello in figura 6.1.

Figura 6.1: *Construction Script* dei Blueprint di misurazione *BP_Ruler*

6.2.1 Controllo dello strumento e calcolo della misura

Per il controllo dello strumento all'interno dell'editor del livello di UE4, sarebbe conveniente avere una serie di punti di controllo da muovere manualmente all'interno della scena stessa. Una soluzione efficace consiste nell'uso di variabili vettori all'interno del blueprint, a cui viene abilitata l'opzione "Show 3D Widget".

Questa opzione permette di visualizzare nell'editor del livello un gizmo 3D, con cui controllare il valore del variabile dell'istanza. In questo modo, muovendo il punto all'interno della scena, il valore che la variabile contiene viene aggiornato in corrispondenza alla posizione del punto stesso rispetto all'asse cartesiano del mondo.

I tre blueprint utilizzano dei punti di controllo differenti, in funzione delle esigenze:

- Lo strumento per la misura della distanza tra due punti, utilizza due variabili "*Start*" e "*End*" le quali indicano gli estremi del segmento di cui si vuole misurare la lunghezza (esempio dei punti di controllo in figura 6.2);

- Lo strumento per la misura della circonferenza di un cerchio utilizza un singolo punto di controllo "*Radius*" che indica la distanza dal centro della circonferenza, ovvero il raggio;

- Lo strumento per la misura della circonferenza di una ellisse utilizza due punti di controllo, "*RadiusA*" e "*RadiusB*", i quali indicano la lunghezza degli assi dell'ellisse.

Dopodiché, con la funzione *CalculateMeasure* si calcolano le misure base come distanza tra i due punti, raggio della circonferenza e raggi principali dell'ellisse. I dati vengono poi usati come input per il sistema di visualizzazione delle misure.

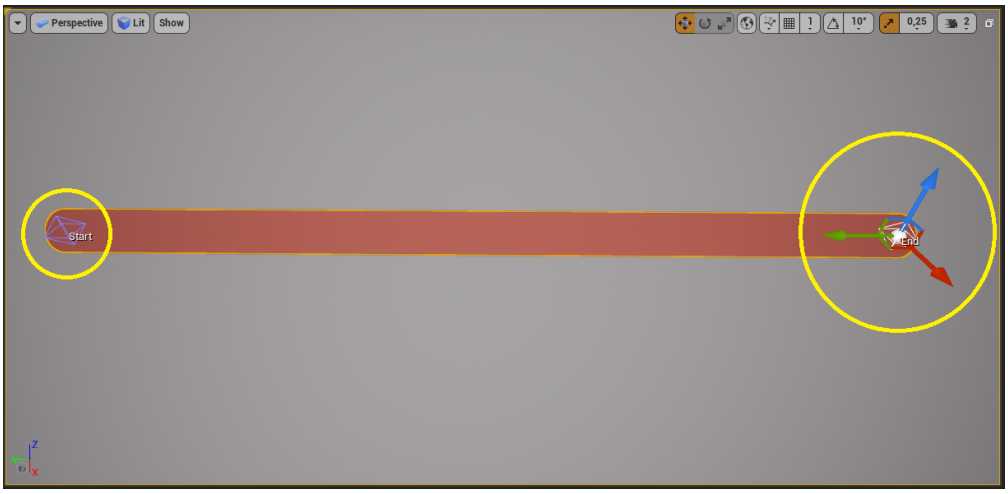


Figura 6.2: Punti di controllo dello strumento *BP_Ruler*

6.2.2 Visualizzazione della misura

Per questa seconda parte si è inizialmente pensato di utilizzare un *Widget Component* e, di conseguenza, creare un Widget in grado di visualizzare i dati. Questa soluzione risulta sicuramente più flessibile dato che i Widget di UE4 mettono a disposizione una vastità di funzionalità, tuttavia una volta realizzato, si è verificato un problema di visualizzazione. Infatti, i Widget vengono solo renderizzati durante l'esecuzione dell'applicazione virtuale, quindi non durante la modifica delle scene all'interno dell'editor. Dato che è importante che le misure siano disponibili anche

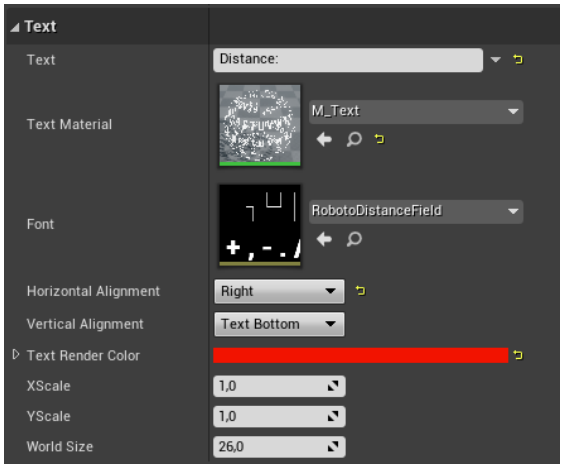


Figura 6.3: Attributi del Text Render Component

e soprattutto durante i processi di modifica, si è optato per una soluzione meno flessibile, ma sicuramente migliore, basata sull'uso dei Text Render Component.

I *Text Render Component* sono dei componenti di UE4 che permettono di renderizzare del testo all'interno del mondo. Gli attributi associati ad esso per la gestione del testo sono diversi: testo, materiale utilizzato dal testo, font, allineamento, colore e scala (figura 6.3).

Sono quindi utilizzati due componenti per visualizzare su schermo la misura effettuata dal Blueprint, uno per il nome della misura e uno per il valore della misura stessa, vediamo un esempio in figura 6.4.

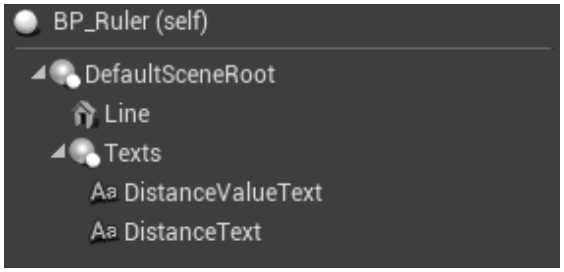


Figura 6.4: Esempio dei componenti di testo in un attore

Come mostrato in figura 6.3, si è impostato un particolare materiale chiamato *M_Text* per facilitare la leggibilità del testo anche in ambienti non illuminati. Il nuovo materiale, rispetto a quello predefinito di UE4 per il testo, ha un colore emissivo, di conseguenza il testo emetterà luce propria, migliorando la lettura dei dati (figura 6.5).

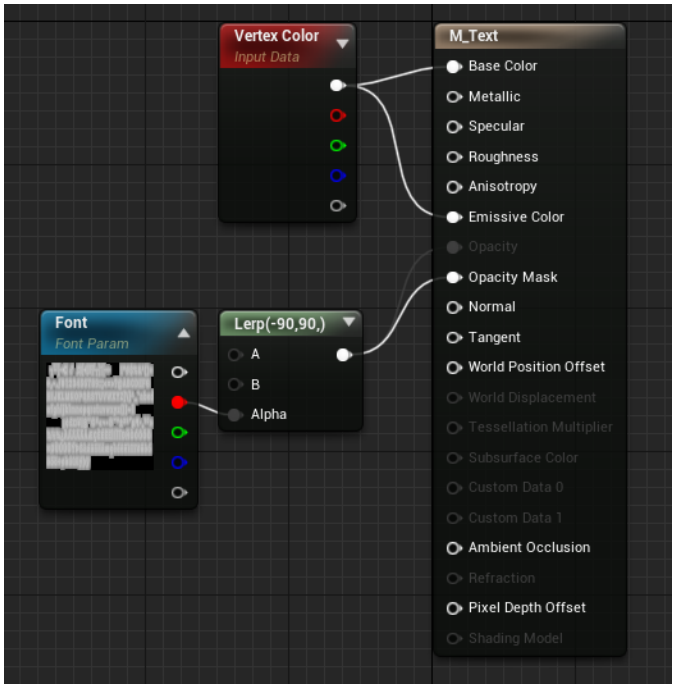


Figura 6.5: Nodi del materiale *M_Text*

Per aggiornare il testo visualizzato dall'attore è stata definita la funzione *UpdateText*, collegata al flusso di esecuzione del Construction Script del blueprint. All'interno di essa viene chiamata la funzione *SetText* legata al RenderTextComponent.

Nel caso del primo blueprint, viene semplicemente incollata la distanza tra i due punti calcolata precedentemente. Per il secondo blueprint, quello relativo ai cerchi, vengono visualizzate due misure: la prima per il raggio, la seconda per la circonferenza, calcolata facilmente. Infine, per il terzo blueprint, il calcolo della circonferenze delle ellissi viene effettuato tramite un particolare nodo chiamato *MathExpression* il quale permette di definire formule matematiche complesse (figura 6.6).

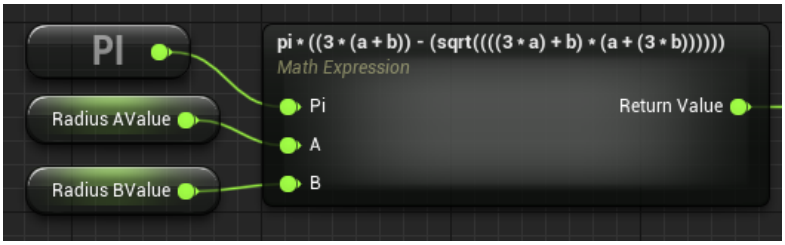


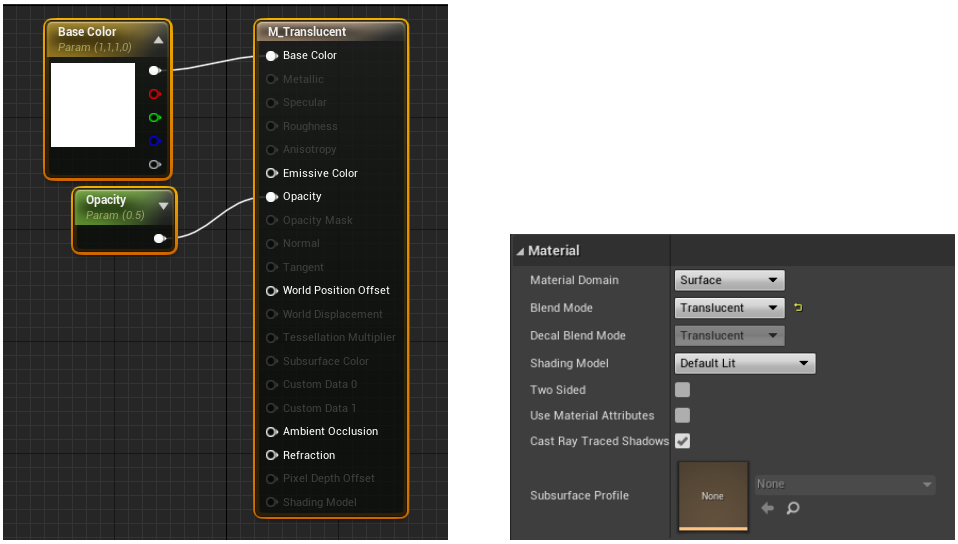
Figura 6.6: Nodo *MathExpression* per calcolare la circonferenza di una ellisse

6.2.3 Visualizzazione della mesh

L'ultimo passo consiste nella visualizzazione di una mesh rappresentativa della misura effettuata, come linee, circonferenze ed ellissi. La soluzione applicata consiste nell'utilizzo di un generico modello virtuale, il quale viene scalato in funzione delle esigenze. Per garantire un'ottima rappresentazione delle varie misure la scelta di un cilindro è risultata naturale.

Il modello del cilindro è aggiunto ai blueprint tramite uno *Static Mesh Component*, scegliendo un materiale diverso da quello predefinito. Questo particolare componente di UE4 permette di usare e visualizzare una geometria all'interno di un attore.

Come detto precedentemente, è stato creato un particolare materiale traslucido che permette una buona visione della mesh e si adatta a vari ambienti e scenari, senza dar troppo fastidio al resto dell'ambiente in quanto è possibile vedere attraverso di esso. Nella pratica, è stato definito un nuovo *Material* di UE4, impostando la *Blend Mode* in *Traslucent* (figura 6.7b), questa proprietà permette di specificare come il colore del materiale si fonde con i colori di background. Dopodiché si è impostato un valore di opacità del materiale diverso da 1 (figura 6.7a).



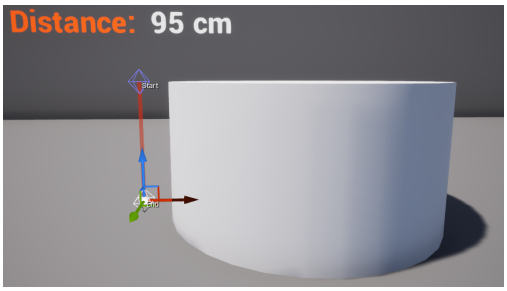
(a) Nodi del materiale traslucido **(b)** Parametri del materiale traslucido

Figura 6.7: Materiale traslucido *M_Translucent*

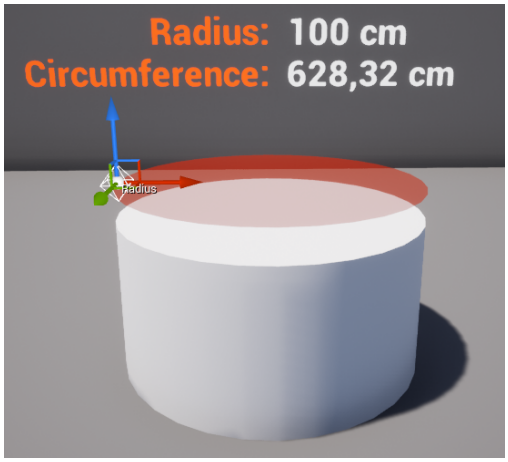
Infine, per aggiornare la mesh, quindi i corrispondenti valori di scala, si è definita una funzione *UpdateMesh*, legata al flusso di esecuzione del *Construction Script*. Questa permette di ruotare e scalare il modello in funzione dei punti di controllo.

6.3 Risultato

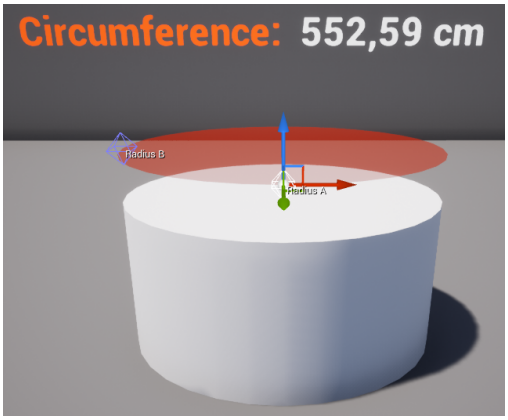
Il risultato ottenuto consiste in tre asset importabili in qualsiasi scena di Unreal Engine 4, che permettono le tre differenti misure citate. L'utilizzo è molto semplice in quando basta agire sui punti di controllo dello strumento, direttamente all'interno dell'editor del livello. Inoltre, la visualizzazione è semplice ma efficace. Si mostrano, nelle tre figure successive, i tre blueprint all'interno dell'editor di UE4.



(a) Strumento per distanze



(b) Strumento per circonferenze di cerchi



(c) Strumento per circonferenze di ellissi

Figura 6.8: Esempi degli strumenti di misura

Capitolo 7

Sviluppo dello strumento per la creazione di manichini digitali

7.1 Obiettivo

L'obiettivo del secondo modulo di questa tesi è la realizzazione di uno strumento che permetta di creare dei manichini digitali, utili al design ergonomico di sistemi spaziali.

Come base vengono utilizzati gli umanoidi generati dallo strumento MetaHuman, il quale è stato esposto nel capitolo 5.3. Dopodiché sono importati all'interno di Unreal Engine 4 per essere modificati in modo che le loro misure rispettino le misure antropometriche definite dallo standard NASA.

I manichini creati devono essere completamente animabili tramite cinematica diretta ed inversa, sfruttando lo strumento Control Rig di UE4 (capitolo 5.2), in modo da poter definire le pose utili al design ergonomico. Inoltre, lo strumento deve permette la scalabilità del processo di creazione e deve essere garantita la possibilità di poter modificare le misure singolarmente, anche dopo il processo di creazione. Il tutto deve essere possibile all'interno del motore di gioco.

L'obiettivo finale è la creazione di due manichini, uno che rappresenti il 1° percentile e uno il 99° percentile, definiti dal documento GP 10017 della NASA. Si rimanda al paragrafo 2.4 per maggiori dettagli sullo stato dell'arte e sugli obiettivi.

7.2 Creazione dei manichini base

Per la creazione dei manichini da utilizzare come base per lo strumento sviluppato, viene utilizzata l'applicazione MetaHuman, messa a disposizione gratuitamente da Unreal Engine 4.

Il manichino creato tramite il MetaHuman Creator (capitolo 5.3), rinominato Thales, è un semplice umanoide per cui si è scelto di utilizzare i parametro di altezza media (Average) e di massa corporea media (Average). Inoltre si è scelto il sesso maschile.

Le altezze dei manichini, in funzione del sesso e del parametro scelto, sono mostrati in tabella 7.1. Questi dati sono acquisiti dalla documentazione ufficiale di MetaHuman [45], purtroppo l'altezza è l'unica misura ufficiale disponibile.

Parametro altezza	Sesso femminile	Sesso maschile
Basso	1.49 m	1.65 m
Medio	1.60 m	1.72 m
Alto	1.65 m	1.80 m

Tabella 7.1: Altezze standard dei manichini di MetaHuman

L'umanoide creato ha quindi un'altezza di 1.72 m, questa è stata scelta in quanto una misura media di altezza, in questo modo sarà più semplice creare manichini più alti o più bassi con lo strumento creato nelle prossime sezioni di questo capitolo. In figura 7.1 viene mostrato il volto altamente realistico del modello creato.

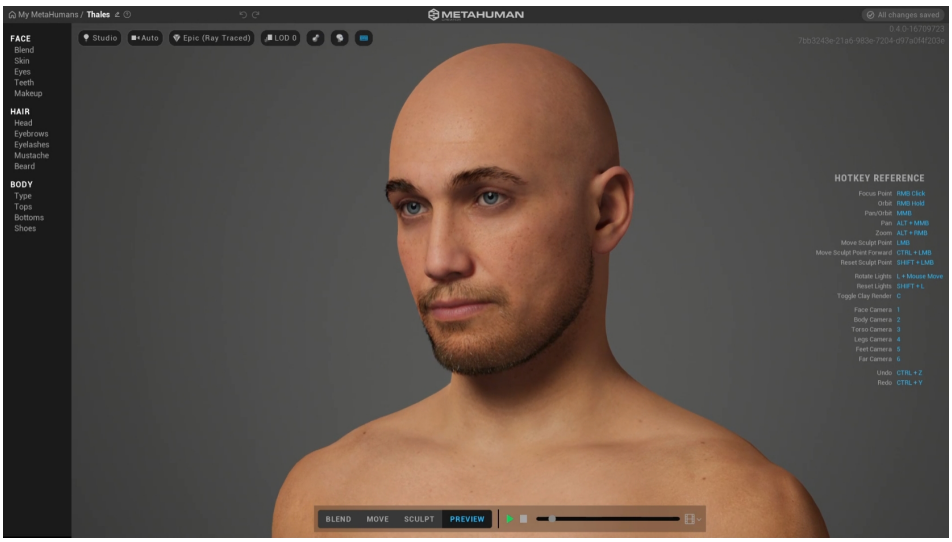


Figura 7.1: MetaHuman *Thales* all'interno del MetaHuman Creator

Successivamente l'umanoide è importato all'intero del progetto di Unreal Engine 4, tramite l'utilizzo di Quixel Bridge. L'intero processo è esplicato nel paragrafo 5.3.3 di questa tesi. Gli asset e file importati all'interno del motore sono molteplici: mesh statiche dei manichini, scheletri, texture del volto e delle parti del corpo, materiali, Control Rig Blueprint, Animation Blueprint e molto altro.

Gli asset principalmente utili per lo sviluppo dello strumento sono lo scheletro del manichino e il Blueprint dell'umanoide stesso, inoltre per le fasi successive sarà necessario sfruttare il completo Control Rig messo a disposizione da UE per i propri MetaHuman.

7.3 Misure dello standard NASA

Lo standard per le misure antropometriche definito dalla NASA nel documento GP 10017, ovvero il Gateway Program Human System Requirements, è stato creato per permettere la modellazione di manichini standardizzati, utilizzabili per il design ergonomico (sezione 2.4.1).

In particolare, l'agenzia spaziale americana ha definito due gruppi di misure, corrispondenti al 1° (minimo) e al 99° (massimo) percentile, associate ad un genere unisessuale (diverso, rispetto ai precedenti standard), prendendo quindi i minimi e i massimi dei due generi. Le misure, e i dati associati ad esse, sono utilizzati per lo sviluppo dello strumento e per la creazione dei manichini stessi. In questa sezione vengono presentati i dettagli delle varie dimensioni.

Per ogni tabella delle grandezze è associata una figura di riferimento che mette in evidenza ogni voce della tabella a cui è legata (es. la tabella 7.2 è associata alla figura 7.2).

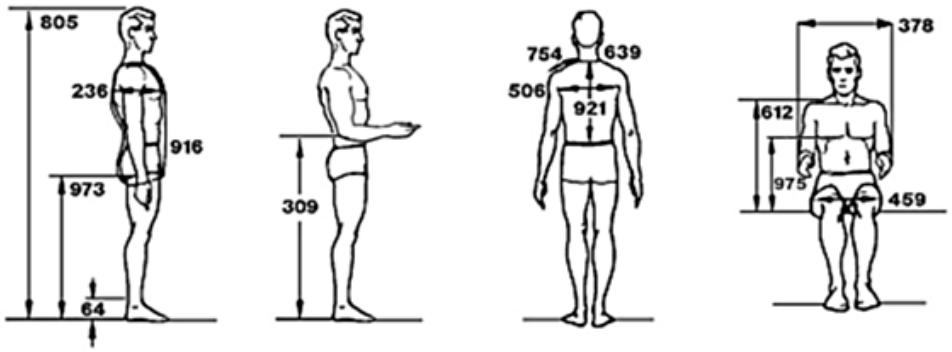


Figura 7.2: Figura esplicativa delle dimensioni antropometriche riguardanti altezze e ampiezze

Numero	Descrizione	Minimo (cm)	Massimo (cm)
805	Altezza	148.6	194.6
973	Altezza polso	70.3	96.3
64	Altezza caviglia	4.8	8.1
309	Altezza gomito	89.1	120.7
236	Profondità busto	19.1	30.2
916	Circonferenza del tronco verticale	134.9	181.9
612	Altezza piano-spalla in seduta	52.6	71.1
459	Ampiezza anca in seduta	31.5	46.5
921	Ampiezza vita posteriore	39.1	55.9
506	Ampiezza schiena	29.3	48.0
639	Circonferenza collo	27.8	43.4
754	Lunghezza spalle	12.0	18.0
378	Ampiezza avambraccio-avambraccio	38.9	66.0
975	Altezza petto in seduta	33.8	50.3

Tabella 7.2: Dimensioni antropometriche dello standard NASA riferite alla figura 7.2

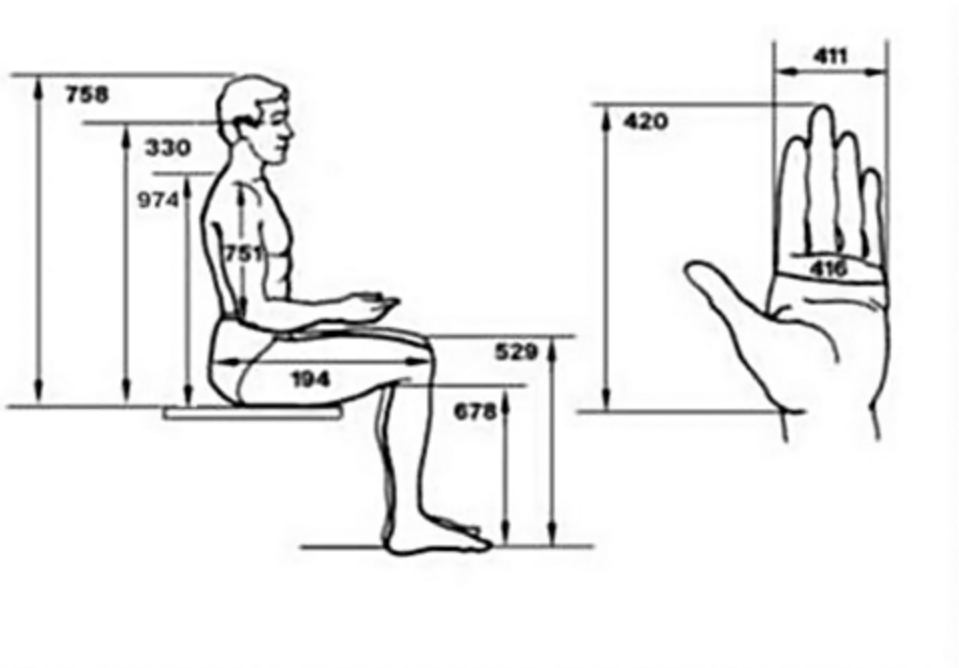


Figura 7.3: Figura esplicativa delle dimensioni antropometriche riguardanti mani e seduta

Numero	Descrizione	Minimo (cm)	Massimo (cm)
758	Altezza in seduta	77.7	101.3
330	Altezza occhi in seduta	66.6	88.9
529	Altezza ginocchia in seduta	45.5	63.5
678	Altezza popliteo	33.0	50.0
751	Lunghezza spalla-gomito	29.6	41.9
194	Lunghezza natica-ginocchio	52.1	69.9
420	Lunghezza mano	15.8	22.1
411	Ampiezza mano	7.1	10.2
416	Circonferenza mano	16.8	24.1
974	Altezza cervicale in seduta	56.6	76.2

Tabella 7.3: Dimensioni antropometriche dello standard NASA riferite alla figura 7.3

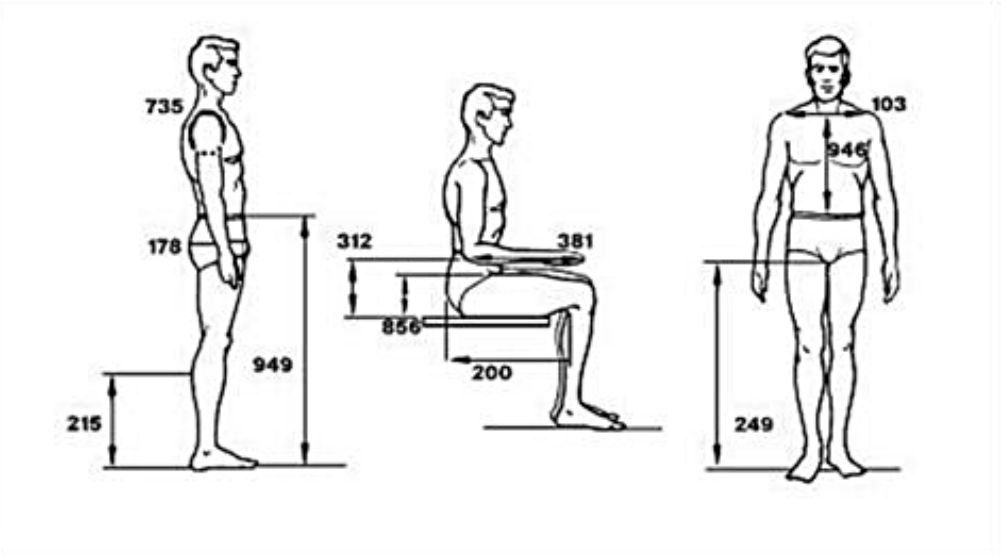


Figura 7.4: Figura esplicativa delle dimensioni antropometriche riguardanti ulteriori altezze e ampiezze

Numero	Descrizione	Minimo (cm)	Massimo (cm)
949	Altezza vita	86.6	119.6
249	Altezza cavallo	66.5	95.8
215	Altezza polpaccio	25.9	41.4
103	Ampiezza biacromiale	32.3	44.5
946	Ampiezza vita frontale	34.1	48.8
735	Circonferenza spalla	31.9	52.1
178	Circonferenza glueti	84.1	114.8
312	Altezza gomito a riposo	16.2	30.0
856	Altezza coscia in seduta	13.0	20.1
381	Lunghezza avambraccio-mano	38.7	54.6
200	Lunghezza natiche-popliteo	42.2	57.2

Tabella 7.4: Dimensioni antropometriche dello standard NASA riferite alla figura 7.4

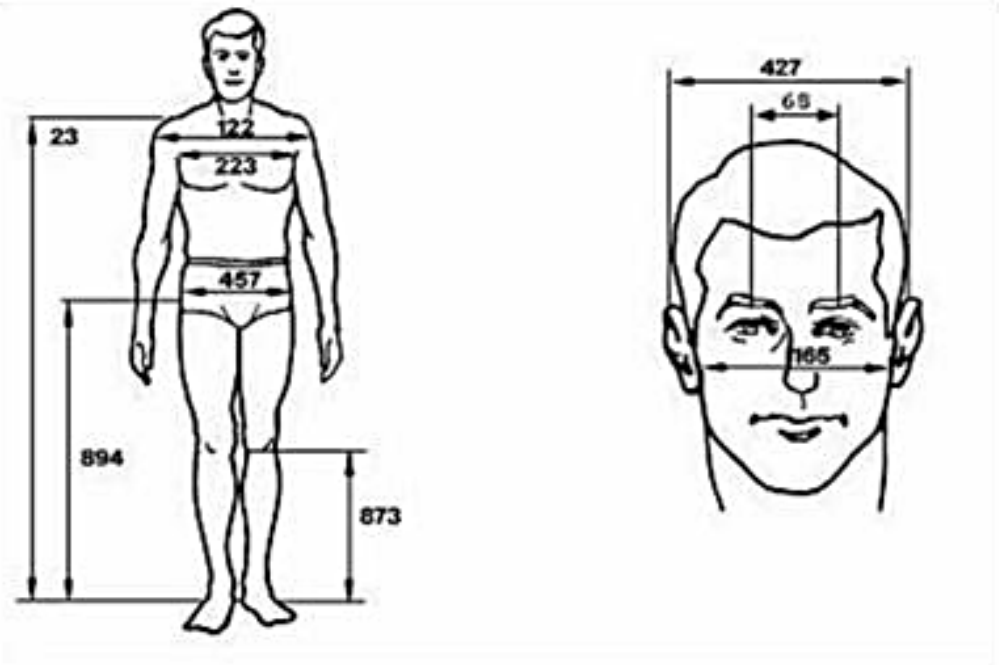


Figura 7.5: Figura esplicativa delle dimensioni antropometriche riguardanti il cranio e ulteriori altezze e ampiezze

Numero	Descrizione	Minimo (cm)	Massimo (cm)
23	Altezza acromiale	120.4	161.8
894	Altezza bacino	75.2	105.4
873	Altezza tibiale	39.6	57.9
122	Ampiezza deltoidi	37.8	56.1
223	Ampiezza vita frontale	23.5	39.4
457	Ampiezza petto	29.8	40.6
165	Ampiezza viso	12.0	15.5
427	Ampiezza cranio	13.3	16.5
68	Ampiezza pupilla-pupilla	5.3	7.4

Tabella 7.5: Dimensioni antropometriche dello standard NASA riferite alla figura 7.5

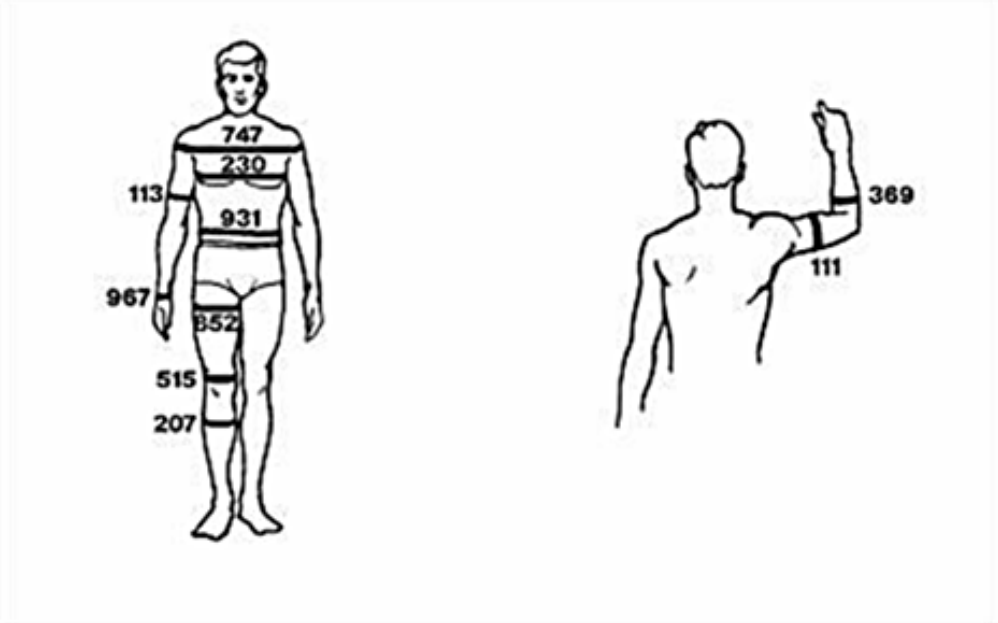


Figura 7.6: Figura esplicativa delle dimensioni antropometriche riguardanti diverse circonferenze

Numero	Descrizione	Minimo (cm)	Massimo (cm)
747	Circonferenza spalle	90.4	133.9
230	Circonferenza petto	75.7	118.6
931	Circonferenza vita	61.2	110.5
852	Circonferenza cosce	47.8	71.9
515	Circonferenza ginocchia	30.7	44.5
207	Circonferenza polpaccio	29.5	44.5
967	Circonferenza polso	13.5	19.8
111	Circonferenza bicipiti	22.9	40.4
369	Circonferenza avambraccio	21.6	35.3

Tabella 7.6: Dimensioni antropometriche dello standard NASA riferite alla figura 7.6

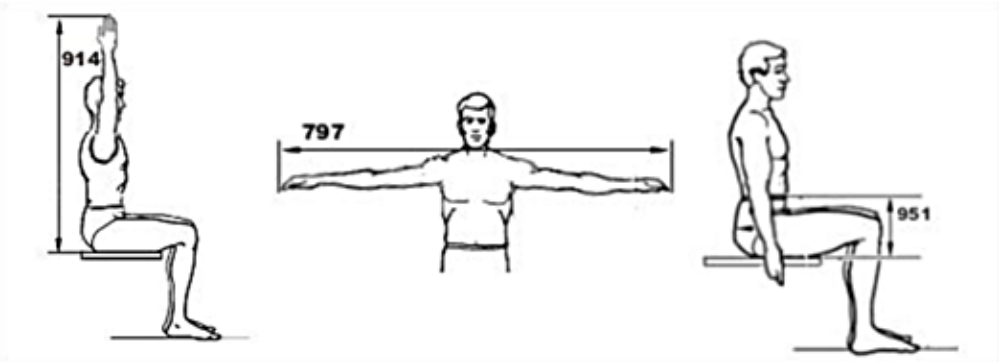


Figura 7.7: Figura esplicativa delle dimensioni antropometriche riguardanti ulteriori lunghezze

Numero	Descrizione	Minimo (cm)	Massimo (cm)
951	Altezza vita in seduta	19.3	27.2
797	Estensione braccia	147.8	204.7
914	Portata vert. dell'indice in seduta	118.9	158.2

Tabella 7.7: Dimensioni antropometriche dello standard NASA riferite alla figura 7.7

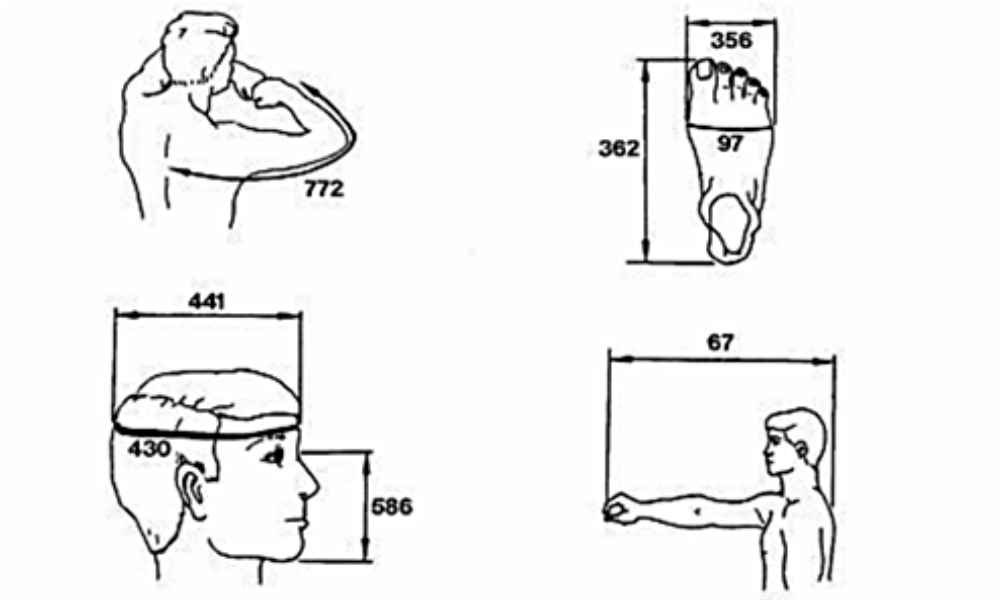


Figura 7.8: Figura esplicativa delle dimensioni antropometriche riguardanti i piedi e ulteriori dimensioni

Numero	Descrizione	Minimo (cm)	Massimo (cm)
67	Portata del pollice	19.3	27.2
772	Lunghezza manica	147.8	204.7
441	Lunghezza cranio	118.9	158.2
430	Circonferenza cranio	19.3	27.2
586	Lunghezza menton-sellion	147.8	204.7
362	Lunghezza piede	118.9	158.2
356	Ampiezza piede	118.9	158.2
97	Circonferenza piede	118.9	158.2

Tabella 7.8: Dimensioni antropometriche dello standard NASA riferite alla figura 7.8

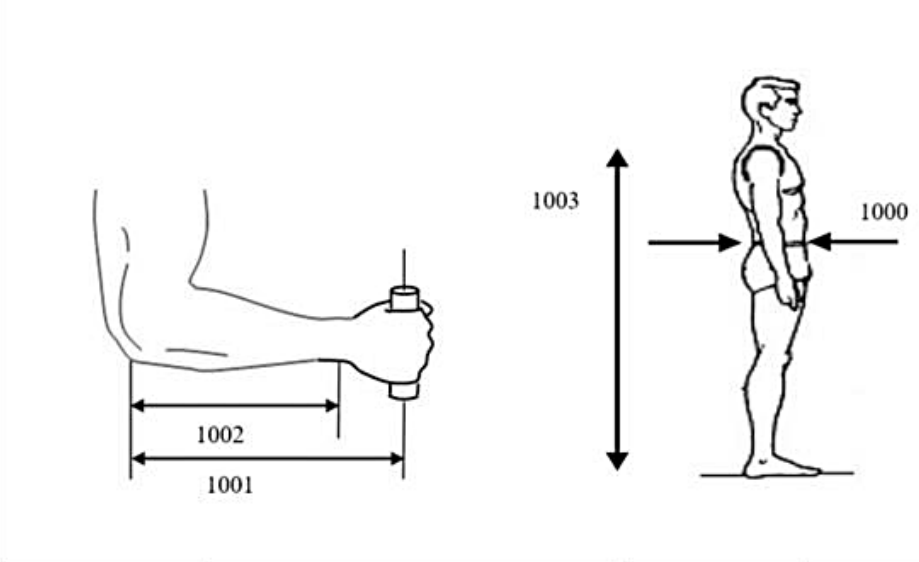


Figura 7.9: Figura esplicativa delle dimensioni antropometriche riguardanti ulteriori lunghezze e altezze

Numero	Descrizione	Minimo (cm)	Massimo (cm)
1000	Profondità vita	15.0	30.0
1001	Lunghezza gomito-centro di presa	28.7	40.8
1002	Lunghezza gomito-polso	22.5	33.1
1003	Altezza cervicale, in piedi	127.7	169.8

Tabella 7.9: Dimensioni antropometriche dello standard NASA riferite alla figura 7.9

7.4 Sviluppo dello strumento per la creazione di manichini digitali

Lo sviluppo dello strumento per la creazione di manichini digitali si concentra all'interno di un Animation Blueprint, un particolare Blueprint messo a disposizione da Unreal Engine 4 (sezione 5.1).

Come spiegato nelle precedenti sezioni, l'umanoide importato da MetaHuman possiede un gran numero di Asset. L'asset utilizzato principalmente per il fine di questa tesi è lo scheletro del corpo del manichino, chiamato *metahuman_base_skel* (figura 7.10), in quanto i valori traslazione, rotazione e scala delle singole ossa vengono alterati per ottenere le precise misure antropometriche definite dallo standard.

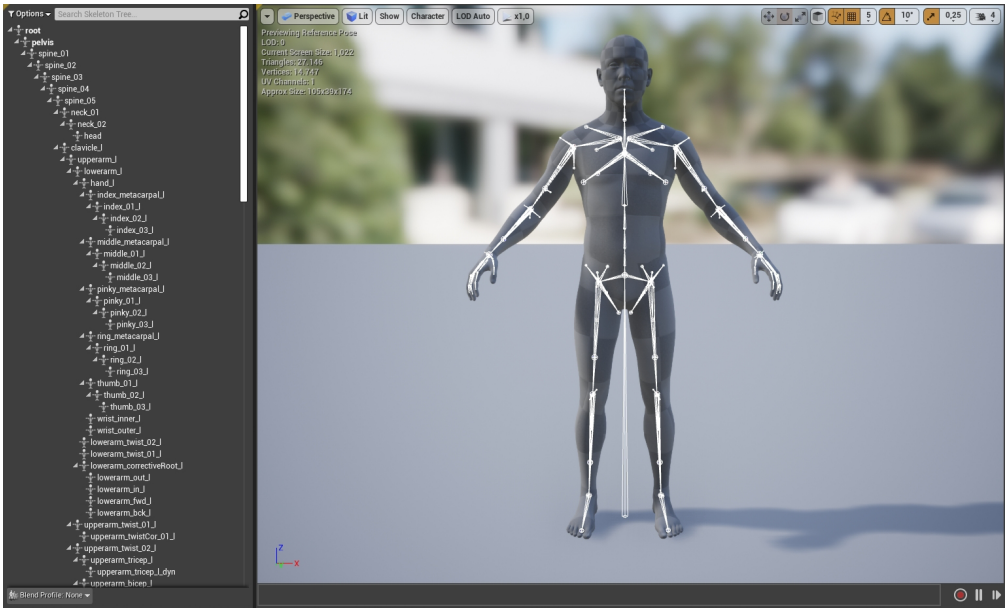


Figura 7.10: Scheletro base dei manichini di MetaHuman

Viene quindi creato un nuovo Animation Blueprint associato allo scheletro di MH, chiamato *ABP_Thales_CustomMannequin*. Per modificare lo scheletro e le trasformazioni 3D bisogna lavorare principalmente con la posa del manichino, sfruttando un particolare nodo dell'AnimGraph, ovvero il *Transform (Modify) Bone* (figura 7.11).

Questo nodo fa parte della categoria Skeleton Control Nodes, la quale raggruppa funzioni che permettono di agire direttamente sulle singole ossa dello scheletro. Il Transform Bone prende in input tre vettori per modificare la traslazione, rotazione o scala, sommando i valori alle quantità attuali o sostituendole (la scelta è definibile

all'interno della finestra Details del nodo stesso), i valori sono poi propagati ai figli dell'osso nella gerarchia. Inoltre prende in input la posa da modificare e un parametro float chiamato *Alpha*, che permette di effettuare una fusione tra la posa di input e i valori definiti nel nodo.

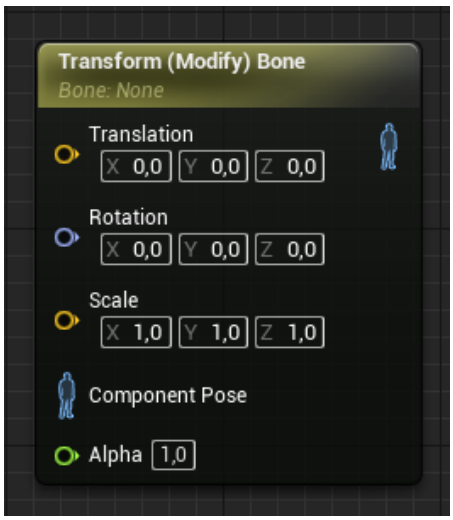


Figura 7.11: Nodo *Transform (Modify) Bone* dell'AnimGraph di Unreal Engine

Questo metodo, rispetto al metodo tradizionale di modellazione della mesh statica, permette di avere più flessibilità nella creazione e modifica delle misure e dei manichini, tuttavia ogni misura richiede un lavoro non trascurabile. Per questo sono state scelte un insieme di dimensioni antropometriche dallo standard NASA per verificare il corretto funzionamento di questa nuova metodologia. Le misure scelte sono relative alle figure 7.2 e 7.8 e mostrate nella successiva tabella 7.10.

Numero	Descrizione	Minimo (cm)	Massimo (cm)
805	Altezza	148.6	194.6
236	Profondità busto	19.1	30.2
921	Ampiezza vita posteriore	39.1	55.9
506	Ampiezza schiena	29.3	48.0
639	Circonferenza collo	27.8	43.4
754	Lunghezza spalle	12.0	18.0
64	Altezza caviglia	4.8	8.1
362	Lunghezza piede	118.9	158.2
356	Ampiezza piede	118.9	158.2

Tabella 7.10: Dimensioni antropometriche scelte dallo standard NASA per la creazione del manichino

Nelle prossime sezioni viene presentata la realizzazione di ogni dimensione all'interno dell'AnimGraph dell'Animation Blueprint creato in precedenza.

7.4.1 Altezza

La prima dimensione, e anche la più semplice da realizzare, è il parametro Altezza. Per questa misura si è semplicemente scelto di applicare una scala, in tutte e tre le dimensioni, all'osso *root*, padre della gerarchia dell'intero scheletro. La scelta è motivata dal fatto che una modifica alla scala solamente lungo l'asse verticale avrebbe creato delle deformazioni notevoli alla mesh del manichino, in questo modo le proporzioni naturali sono mantenute.

Si definisce una variabile float *Stature Scale*, utilizzata per creare un vettore che andrà in input ad un singolo nodo *Transform Bone* (figura 7.12).

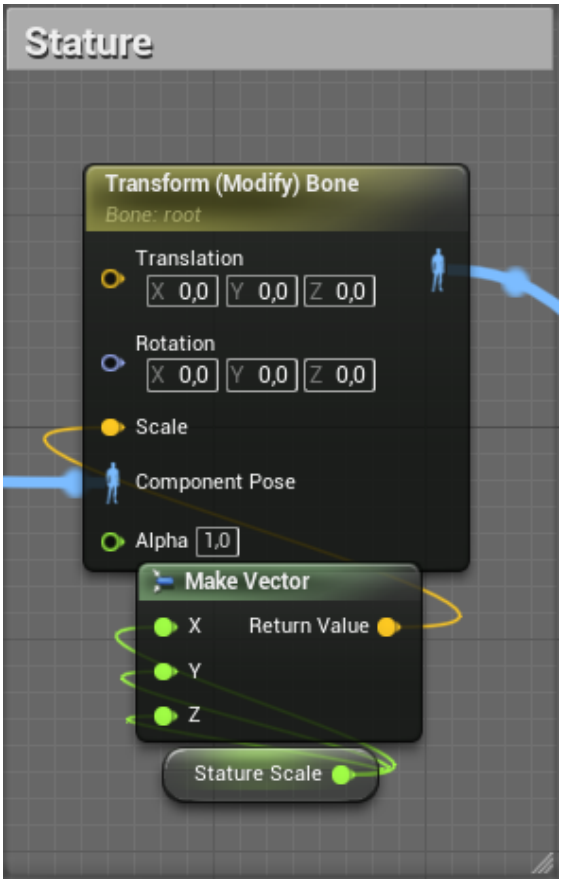
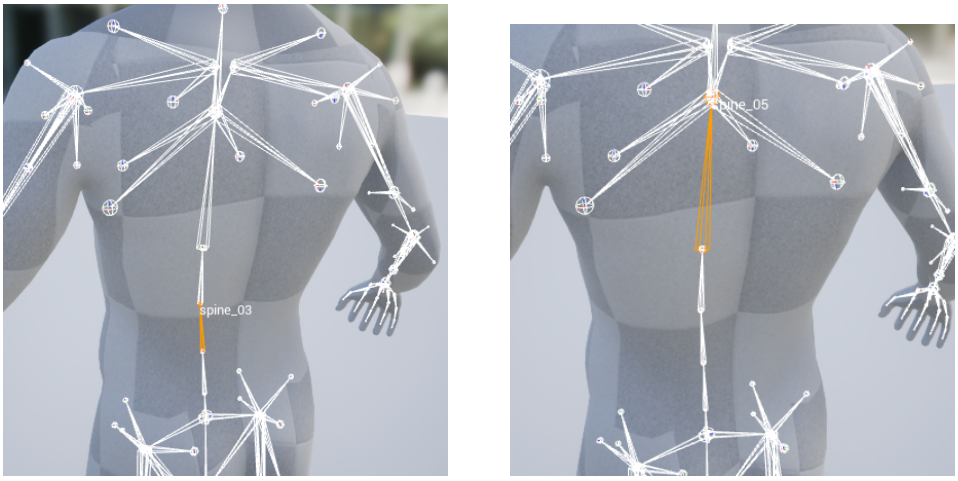


Figura 7.12: Gruppo di nodi dell'AnimGraph per la dimensione *Altezza*

7.4.2 Ampiezza vita

Per agire sull'ampiezza della vita posteriore (o schiena) è stato scelto un particolare osso della gerarchia chiamato *spine_03* (mostrato in figura 7.13a), ovvero una delle ossa che compongono l'asse vertebrale dello scheletro virtuale.



(a) spine_03 **(b)** spine_05

Figura 7.13: Ossa fondamentali della gerarchia di MetaHuman per la definizione della misura *Ampiezza vita posteriore*

Modificando la scala dell'asse X dell'osso citato, è possibile aumentare la dimensione verticale della schiena a proprio piacimento, agendo ovviamente su una variabile chiamata *Waist Back Scale*. Gli altri valori di scala sono lasciati al valore predefinito 1, in quanto non necessari.

Tuttavia questo valore di scala viene propagato anche al resto delle ossa figlie della gerarchia, generando deformazioni ed un effetto non voluto. Per questo si utilizza un secondo nodo *Transform Bone* che inverte la scala in un osso successivo nella gerarchia. L'osso scelto che permette di evitare le deformazioni del volto, spalle e braccia, ma mantenga la possibilità di applicare una scala all'intera schiena, è *spine_05* (figura 7.13b). Il gruppo di nodi che permette di realizzare ciò è mostrato nella figura 7.14.

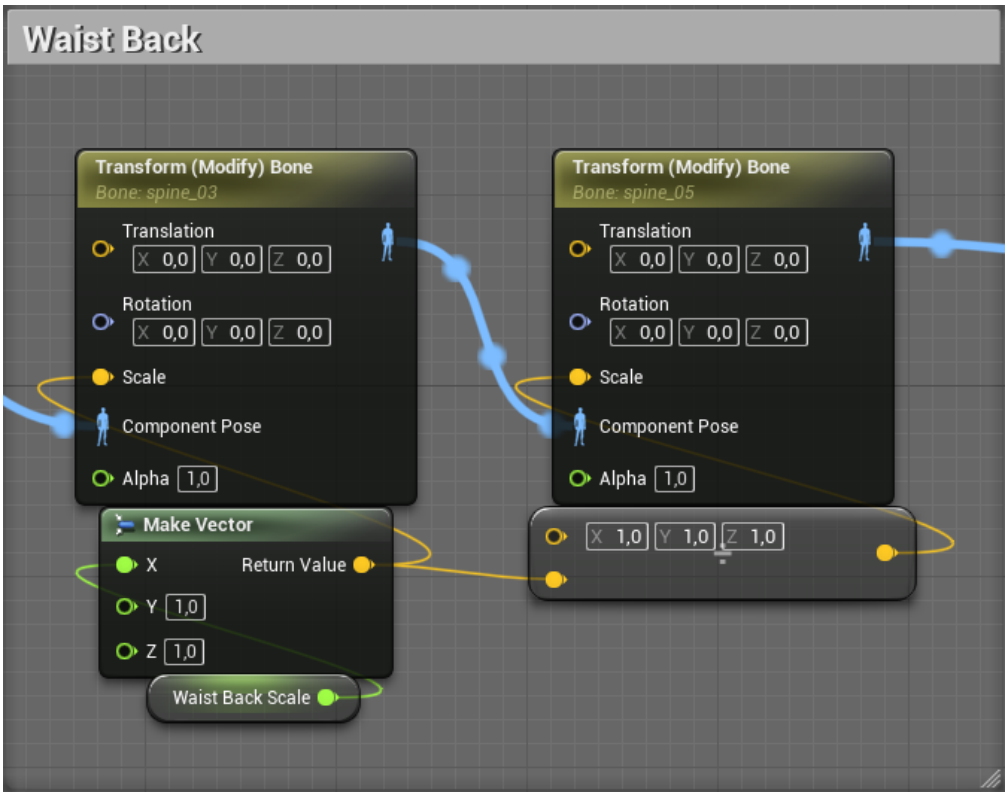
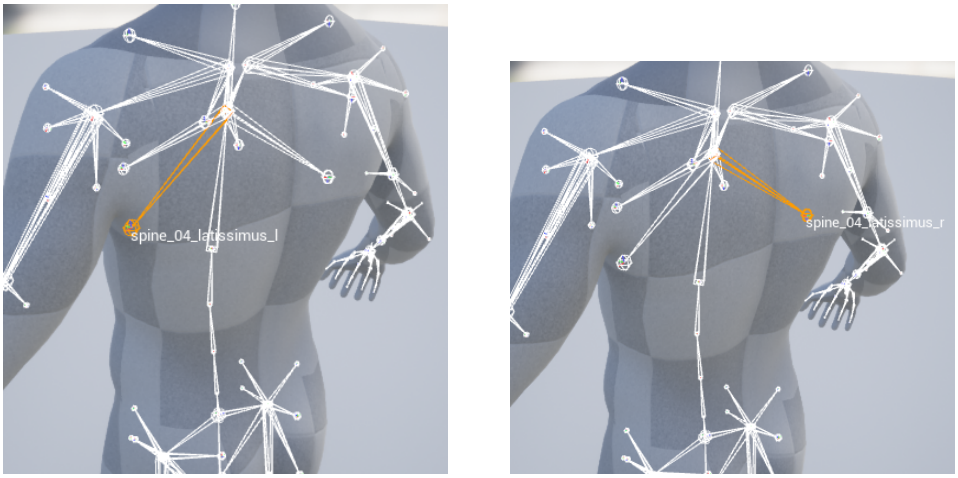


Figura 7.14: Gruppo di nodi dell'AnimGraph per la dimensione *Ampiezza vita posteriore*

7.4.3 Ampiezza schiena

Per modificare l'ampiezza della schiena si è scelto di agire su una coppia di ossa foglie (elementi di una gerarchia senza figli): *spine_04_latissimus_l* per il lato sinistro e *spine_04_latissimus_r* per il lato destro (figura 7.15).



(a) spine_04_latissimus_l **(b)** spine_04_latissimus_r

Figura 7.15: Ossa fondamentali della gerarchia di MetaHuman per la definizione della misura *Ampiezza schiena*

In particolare, traslando queste ossa orizzontalmente è possibile allargare o restringere l'ampiezza della schiena (lungo l'asse X locale di queste). Di conseguenza si definisce una variabile di tipo float chiamata *Interscye Offset*, la quale permette di muovere la coppia di nodi in direzione opposta. In questo caso, si utilizzano due *Transform Bone*, nella quale in uno di essi l'input di traslazione viene invertito.

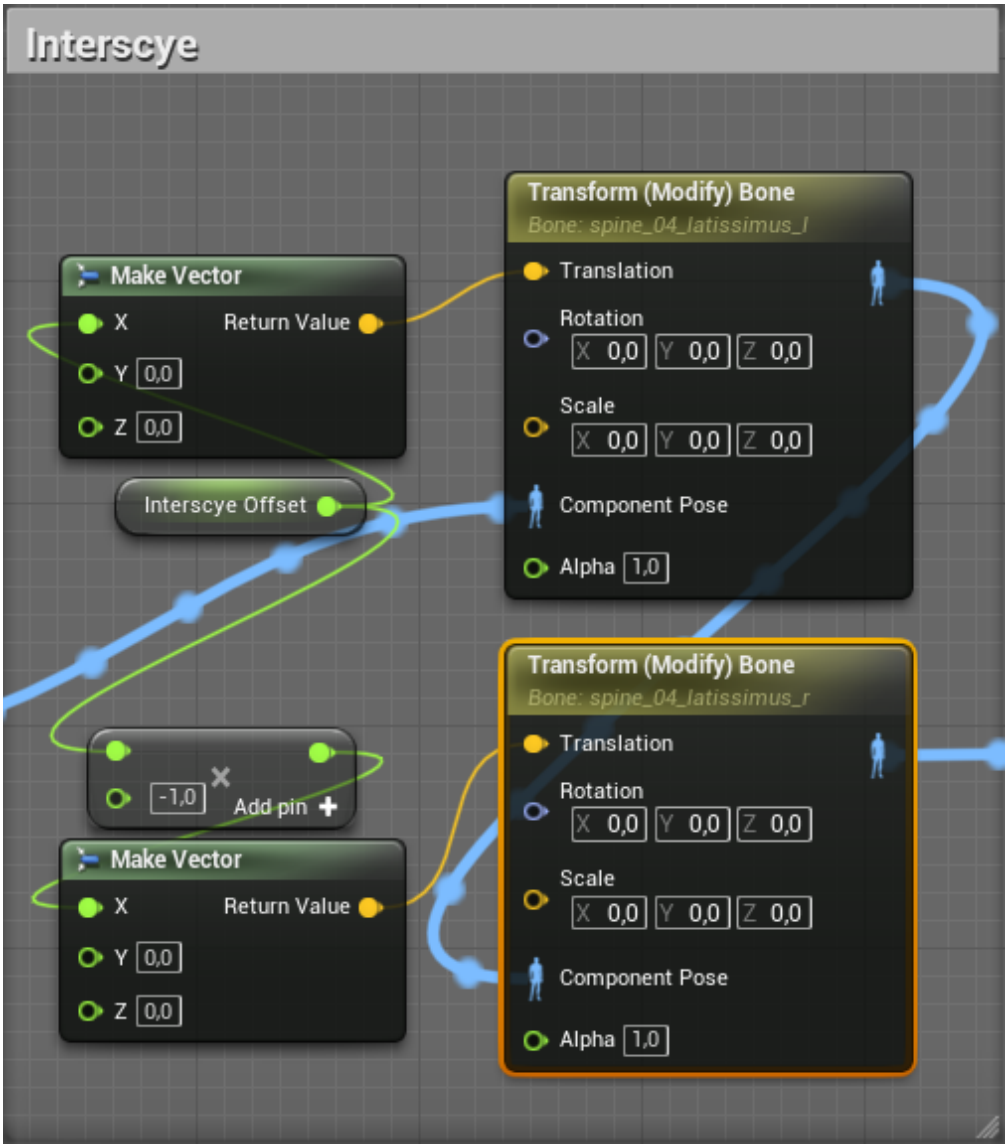


Figura 7.16: Gruppo di nodi dell'AnimGraph per la dimensione *Ampiezza schiena*

7.4.4 Profondità busto

La profondità del busto, o ampiezza petto-schiena, può essere regolata facilmente applicando un valore di scala diverso da 1 all'osso *spine_05*, il quale si trova in corrispondenza del petto del manichino. In particolare la scala viene applicata sull'asse Y, definendo una variabile chiamata *Bust Depth Scale*.

Tuttavia, come per casi precedenti, la scala viene propagata a tutti i nodi figli dell'osso selezionato, creando una serie di deformazioni inopportune. Per questo, si imposta il valore di scala sull'asse Y pari all'inverso del valore della variabile definita, ad un certo numero di ossa. In questo caso, le ossa a cui applicare la scala inversa sono ben sette: neck_01, *clacicle_l*, *clacicle_r*, *spine_04_latissimus_l*, *spine_04_latissimus_r*, *clavicle_pec_l* e *clavicle_pec_r*. I nodi utilizzati per definire questa dimensione sono mostrati in figura 7.17.

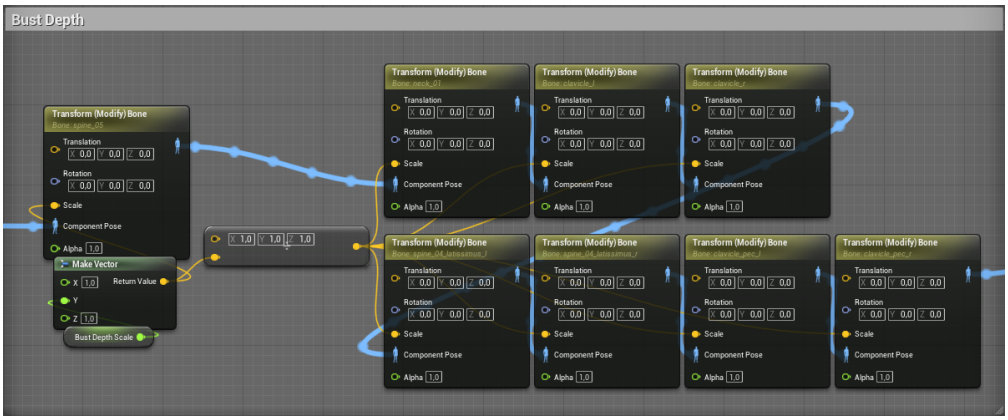
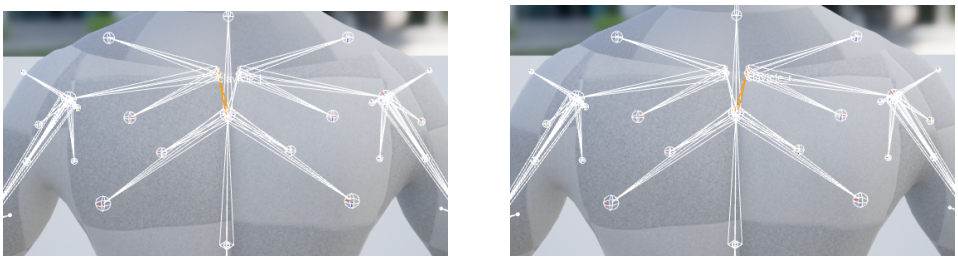


Figura 7.17: Gruppo di nodi dell'AnimGraph per la dimensione *Profondità busto*

7.4.5 Lunghezza spalle

La lunghezza o larghezza delle spalle è principalmente dipendente da due ossa, uno per lato: *clavicle_l* per la spalla sinistra e *clavicle_r* per la spalla destra (mostrate in figura 7.18). Agendo sulla scala di questi nodi, in particolare lungo il loro asse X, è possibile modificare la dimensione delle spalle nella mesh statica del manichino. Per questo fine è definita la variabile di tipo float chiamata *Shoulder Length Scale*.



(a) clavicle_l **(b)** clavicle_r

Figura 7.18: Ossa fondamentali della gerarchia di MetaHuman per la definizione della misura *Lunghezza spalle*

Inoltre, come per altre misure, il valore di scala viene propagato, di conseguenza per evitare le deformazioni delle braccia e mani, si applica una scala inversa a tutti i figli immediatamente successivi alle due ossa citate, come mostrato in figura 7.19.

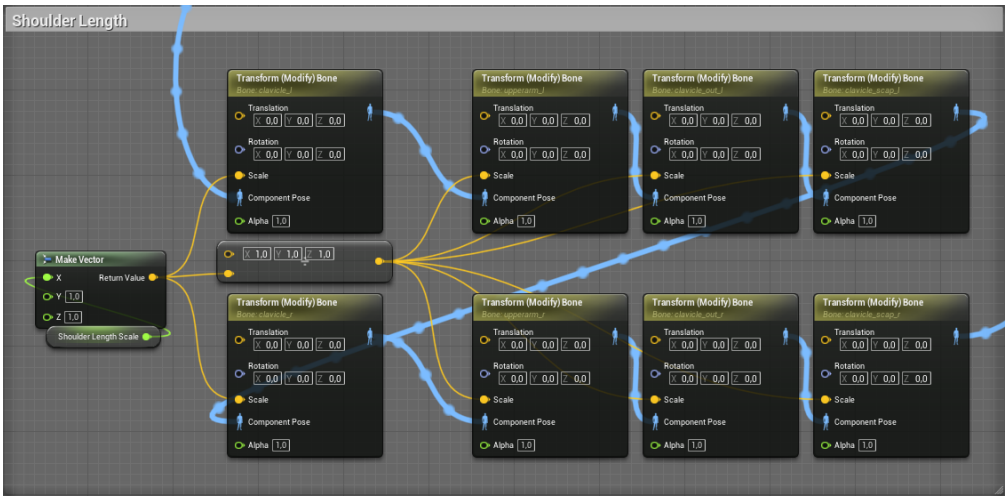


Figura 7.19: Gruppo di nodi dell'AnimGraph per la dimensione *Lunghezza spalle*

7.4.6 Circonferenza collo

La circonferenza del collo è una delle dimensioni che richiede meno nodi. È modificabile applicando una scala all'osso *neck_01* della gerarchia di MetaHuman (figura 7.20). La scala va applicata sui due assi orizzontali dell'osso, in modo da poterne ingrandire o ridurre il perimetro. Questo è fatto definendo una variabile di tipo float chiamata *Neck Circumference Scale*.

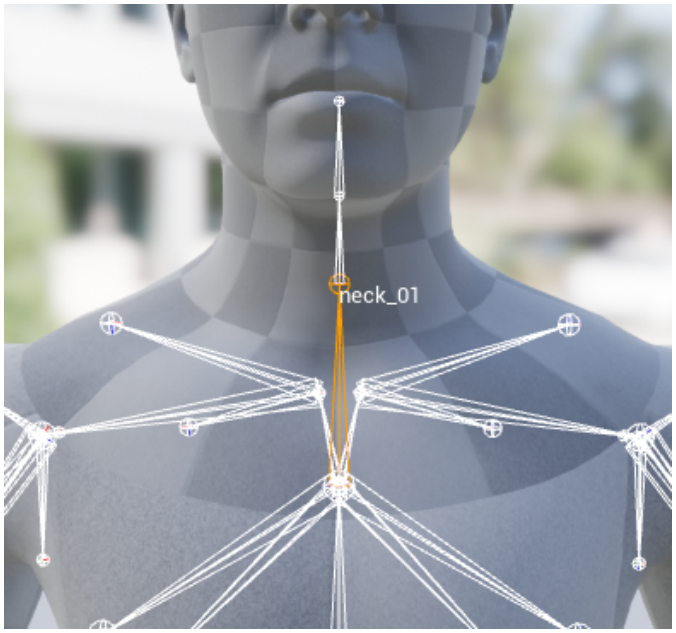


Figura 7.20: Osso *neck_01* della gerarchia di MetaHuman utile per la definizione della misura *Circonferenza collo*

Anche in questo caso va applicata una scala inversa all'osso immediatamente successivo chiamato *neck_02* (figura 7.21), per evitare deformazioni al volto e al cranio.

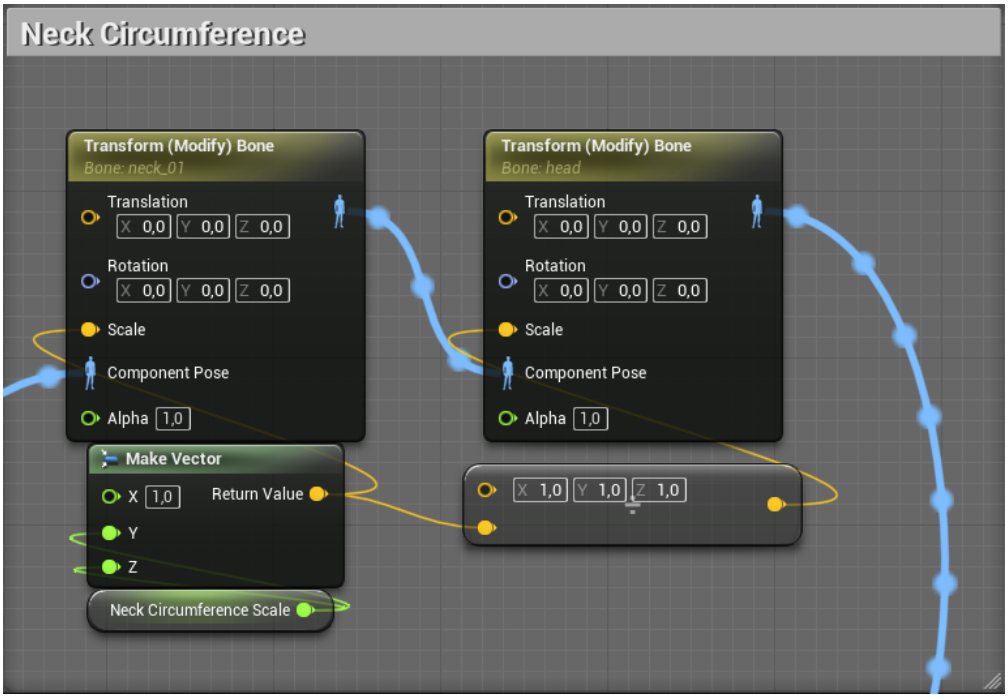
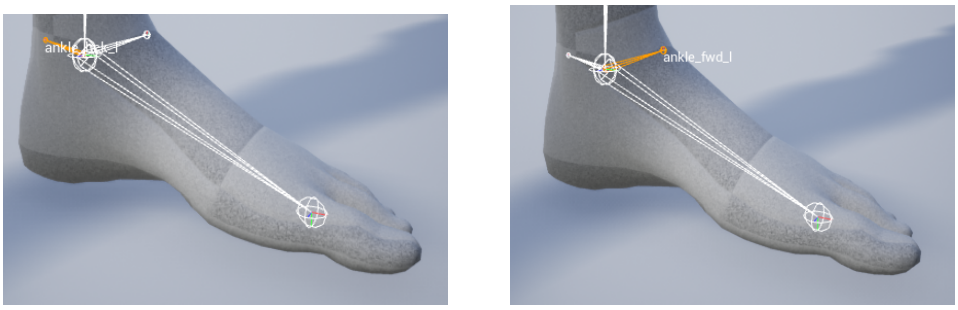


Figura 7.21: Gruppo di nodi dell'AnimGraph per la dimensione *Circonferenza collo*

7.4.7 Altezza caviglia, Lunghezza piede e Ampiezza piede

Le misure antropometriche di altezza caviglia, lunghezza piede e ampiezza piede vengono affrontate nella stessa sezione in quanto strettamente legate nella loro realizzazione.

La misura più complessa è sicuramente l'altezza caviglia, in quanto legata a più ossa dello scheletro di MetaHuman. La prima soluzione consiste nell'applicare una traslazione alle ossa ankle_bck_l e *ancke_fwd_l* per il piede sinistro, *ankle_bck_r* e *ancke_fwd_r* per il piede destro. In figura 7.22 vengono mostrate le ossa solo per il piede sinistro. La traslazione è applicata lungo l'asse verticale, qui valore è indicato all'interno della variabile di tipo float *Ankle Height Offset*.



(a) ankle_bck_l **(b)** ankle_fwd_l

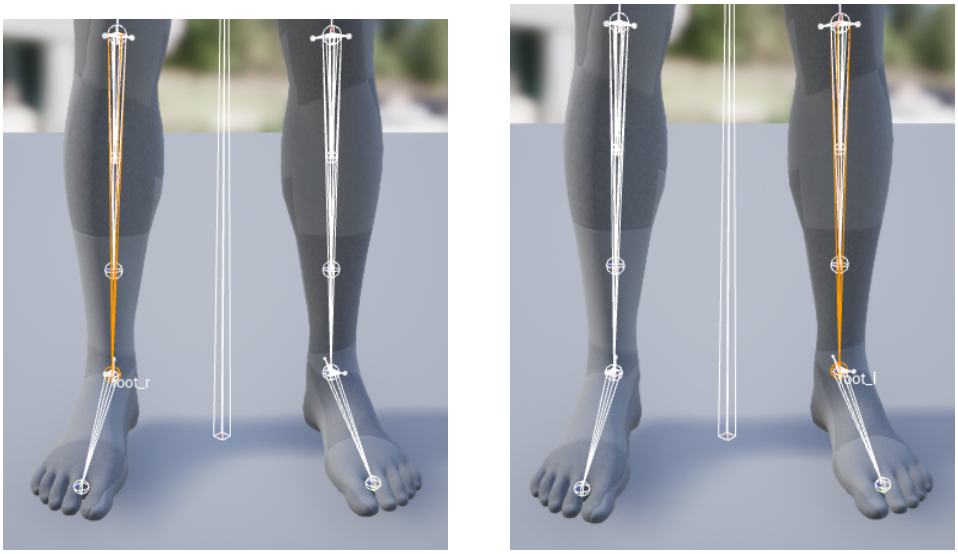
Figura 7.22: Ossa fondamentali della gerarchia di MetaHuman per la definizione della misura *Altezza caviglia*

Tuttavia questa singola operazione non permette di ottenere un risultato soddisfacente, per questo si è scelto di applicare anche un valore di scala alle ossa *foot_l* e *foot_r* (figura 7.23), cui risultano essere i nodi padre delle ossa mostrate in figura 7.22. La scala è applicata sempre lungo l'asse verticale della coppia di ossa, tramite la variabile *Ankle Height Scale*.

La scala applicata alle ossa principali del piede, provoca una deformazione dell'intero arto, il quale tende a distaccarsi dal terreno o allungarsi verso di esso. Per ovviare a questo problema si è realizzato un nuovo controllo tramite la variabile *Foot Position*, la quale permette di regolare la posizione del piede. Questo è fatto applicando una traslazione lungo l'asse verticale (asse Z), in funzione del valore della variabile precedente.

Infine, per le ultime due misure, ovvero la lunghezza piede e la larghezza piede, viene applicata una scala lungo i due assi orizzontali dei piedi alle precedenti ossa *foot_r* e *foot_l*. Anche in questo caso si definiscono due variabili per la regolazione della scala: *Foot Length Scale* e *Foot Breadth Scale*.

L'intero gruppo di nodi nell'AnimGraph per la realizzazione delle tre dimensioni di questa sezione è mostrato in figura 7.24.



(a) foot_r **(b)** foot_l

Figura 7.23: Ossa fondamentali della gerarchia di MetaHuman per la definizione delle misure *Altezza caviglia*, *Lunghezza piede* e *Larghezza piede*

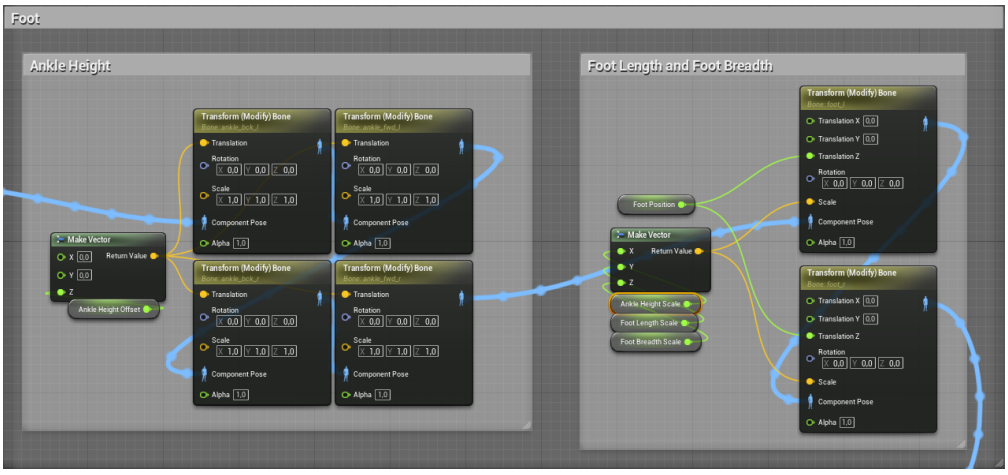


Figura 7.24: Gruppo di nodi dell'AnimGraph per le dimensioni *Altezza caviglia*, *Lunghezza piede* e *Larghezza piede*

7.5 Parametri e interfaccia dell'Animation Blueprint

Il risultato ottenuto dallo sviluppo del precedente capitolo è un Animation Blueprint estendibile per la creazione di specifici manichini, il quale possiede una serie di parametri modificabili per agire direttamente sulle ossa dello scheletro di MetaHuman. La lista dei parametri e l'interfaccia utente per la gestione di questi è mostrata nella prossima figura (7.25).

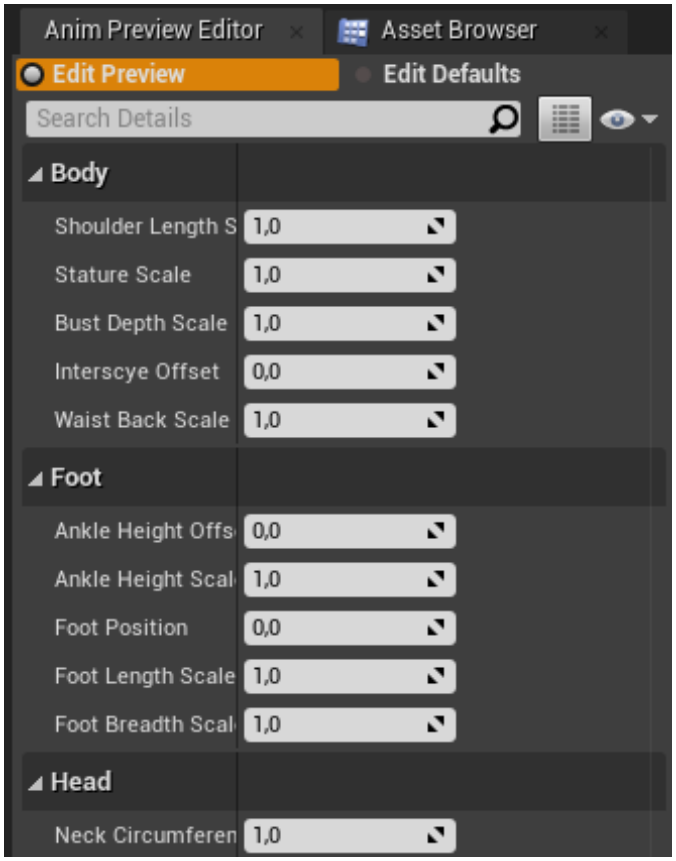


Figura 7.25: Parametri dell'Animation Blueprint per la creazione di manichini personalizzati

Capitolo 8

Creazione e messa in posa dei manichini

8.1 Creazione dei manichini

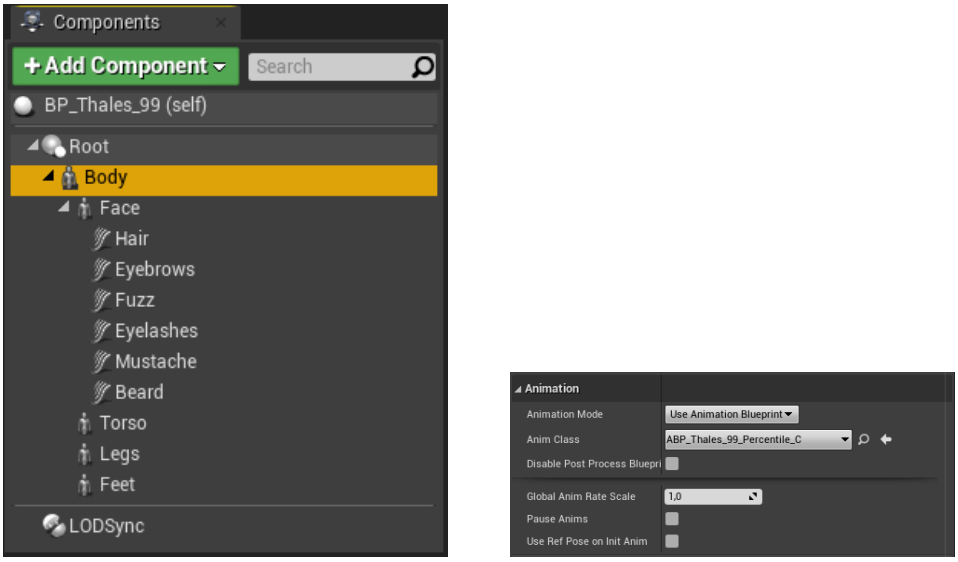
Nel capitolo 7 di questo tesi si è affrontato lo sviluppo dello strumento di creazione dei manichini, di conseguenza è possibile procedere con la creazione degli umanoidi veri e propri. La logica dello strumento è contenuta all'interno dell'Animation Blueprint chiamato *ABP_Thales_CustomMannequin*, nella quale è possibile modificare i valori dei parametri che permettono di modellare lo scheletro e, di conseguenza, la mesh statica del manichino stesso.

Per comodità, dato un particolare manichino con certe dimensioni, è possibile creare una nuova classe Animation Blueprint, che estende la classe precedentemente citata, in modo da ereditarne l'intera logica ed utilizzarla semplicemente come pannello di controllo per la modifica dei parametri dell'umanoide, i quali rimarranno memorizzati all'interno di questa. Si procede quindi con la modifica delle variabili della classe, in modo da ottenere uno scheletro, e una mesh, con le dimensioni volute.

Infine, per la visualizzazione del manichino, è bene ricordare che l'Animation Blueprint è legato solamente allo scheletro del manichino, di conseguenza è possibile scegliere un qualsiasi umanoide creato tramite lo strumento MetaHuman, permettendo alta flessibilità. Infatti, definito un certo scheletro con precise dimensioni, è possibile applicarlo a umanoide con sembianze del tutto diverse.

Nella pratica, si può creare una copia del Blueprint del manichino messo a disposizione da MetaHuman, dopodiché selezionare all'interno di esso lo Skeletal Mesh Component chiamato *Body* in modo da poterne modificare i dettagli (figura 8.1a). Quindi, nella finestra Details cambiare l'*Animation Mode* in *Use Animation Blueprint* e, tramite il campo *Anim Class*, selezionare la particolare classe AB che

contiene i parametri del manichino (figura 8.1b). Dopodiché è possibile posizionare in scena il manichino (contenuto all'interno della classe Blueprint) come un qualsiasi asset di Unreal Engine.



(a) Componenti **(b)** Dettagli del componente *Body*

Figura 8.1: Componenti e dettagli del Blueprint di un manichino MetaHuman

Seguendo il processo spiegato in questa sezione si è proseguito con la creazione dei manichini rappresentanti i due percentili definiti dallo standard NASA, presentato nel precedente capitolo 7.

8.1.1 Creazione del 1° percentile

Come presentato precedentemente, le misure che il manichino rappresentante il 1° percentile deve rispettare sono mostrate in tabella 8.1. Il primo passo consiste nella creazione di un Animation Blueprint, il quale estende la classe *ABP_Thales_CustomMannequin* in modo da ereditarne la logica, chiamato *ABP_Thales_1_Percentile*.

Per impostare i valori dei vari parametri dell'AB, si è utilizzato lo strumento di misurazione sviluppato nel capitolo 6. Lo strumento viene non solo utilizzato per aiutare il processo di scelta dei valori dei paramatri, ma anche per verificare il corretto funzionamento dello strumento e i risultati di questo.

Dopo un attento studio delle dimensioni e misure, si è scelto di utilizzare i seguenti parametri, mostrati in figura 8.2, i quali permettono di generare un manichino che rispetta ampiamente i requisiti dello standard NASA.

Numero	Descrizione	Misura (cm)
805	Altezza	148.6
236	Profondità busto	19.1
921	Ampiezza vita posteriore	39.1
506	Ampiezza schiena	29.3
639	Circonferenza collo	27.8
754	Lunghezza spalle	12.0
64	Altezza caviglia	4.8
362	Lunghezza piede	118.9
356	Ampiezza piede	118.9

Tabella 8.1: Dimensioni antropometriche dello standard NASA per il 1° percentile

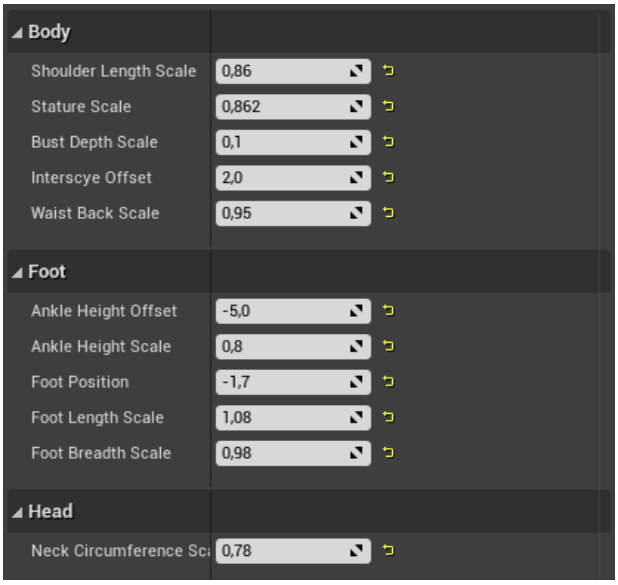


Figura 8.2: Parametri dell'Animation Blueprint per il manichino rappresentante il 1° percentile

Infine, è stato creato una classe Blueprint, copia di quella di un manichino standard importato tramite MetaHumane e Quixel Bridge, nella quale si è impostato l'AB appena definito come *Anim Class*. La classe Blueprint è chiamata *BP_Thales_1*. Il risultato è mostrato in figura 8.4.

8.1.2 Creazione del 99° percentile

Per la creazione del 99° percentile si è seguito uno stesso processo, utilizzando però le misure indicate in tabella 8.2, ovvero quelle associate a questo particolare manichino

dallo standard NASA. La classe AB è chiamata *ABP_Thales_99_Percentile*, mentre la classe Blueprint, per il manichino vero e proprio, è chiamata *BP_Thales_99*. I parametri utilizzati nell'Animation Blueprint di questo secondo umanoide sono mostrati presentati in figura 8.3. Infine, anche per questo manichino, il risultato è mostrato in figura 8.4.

Numero	Descrizione	Misura (cm)
805	Altezza	194.6
236	Profondità busto	30.2
921	Ampiezza vita posteriore	55.9
506	Ampiezza schiena	48.0
639	Circonferenza collo	43.4
754	Lunghezza spalle	18.0
64	Altezza caviglia	8.1
362	Lunghezza piede	158.2
356	Ampiezza piede	158.2

Tabella 8.2: Dimensioni antropometriche dello standard NASA per il 99° percentile

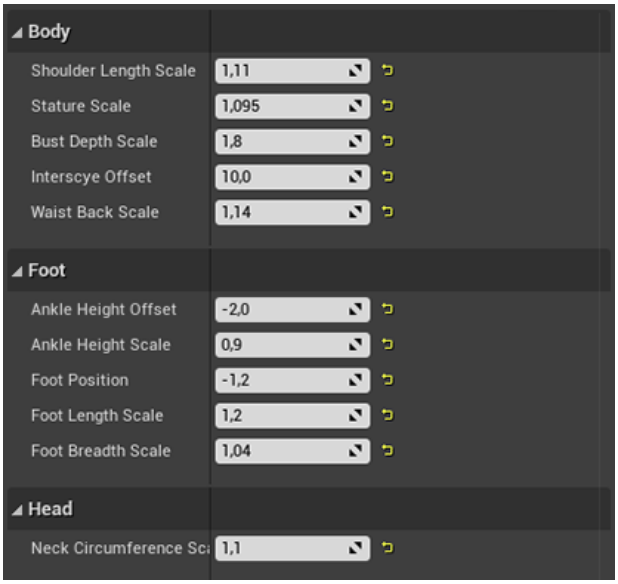


Figura 8.3: Parametri dell'Animation Blueprint per il manichino rappresentante il 99° percentile

Figura 8.4: Esempio dei manichini rappresentanti il 1° e il 99° percentile

8.2 Messa in posa dei manichini

La messa in posa dei manichini è una parte fondamentale per i processi di design ergonomico che li sfruttano. Tramite diverse pose degli umanoidi virtuali è possibile simulare le reali pose degli astronauti e utilizzatori dei moduli, di conseguenza maggiore è la qualità e precisione della posa, migliore risulterà l'intero processo di progettazione.

Per fare ciò si sfrutta il Control Rig di Unreal Engine, presentato nella sezione 5.2 di questa tesi. Dato che i manichini base sono creati tramite MetaHuman, è possibile sfruttare il Control Rig già pronto e altamente flessibile messo a disposizione da MH stesso. L'asset, ovvero il Control Rig Blueprint, è chiamato *MetaHuman_ControlRig* e viene importato nel momento in cui si importa un qualsiasi umanoide di MH in UE4.

Bisogna inizialmente introdurre l'oggetto *Level Sequence* di Unreal Engine. Questi permettono di creare dei filmati e, in generale, di creare una animazione all'interno di una scena di UE4, sfruttando il *Sequencer Editor*. Tramite questa coppia di strumenti è possibile utilizzare i controlli definiti nel Control Rig per definire una particolare posa dell'umanoide, in modo veloce ed intuitivo. Per un maggiore approfondimento si rimanda alla documentazione ufficiale di Unreal Engine [46].

Dopo aver creato il Level Sequence, ed aperto il Sequencer Editor, è possibile aggiungere una serie di asset dal livello attualmente attivo. Nel caso del nostro manichino, si importa semplicemente l'istanza della classe Blueprint attiva in

scena. Dopodiché, per mantenere le misure definite tramite l'Animation Blueprint, bisogna effettuare un *bake* dello scheletro/posa attuale al Control Rig indicato. Per bake si intende la trasformazione della posa attuale del manichino, in una posa definita tramite i controlli del Control Rig. Basta quindi cliccare con il tasto destro sull'oggetto Body del manichino all'interno del Sequencer e selezionare *Bake to Control* Rig, dopodiché selezionare il CR di MetaHuman citato precedentemente.

In questo modo, i controlli del CR sono perfettamente attivi ed è possibile interagire con essi direttamente dal Sequencer, permettendo la messa in posa del proprio manichino. Vediamo un esempio di posa e dei controlli visualizzati su schermo, in figura 8.5. Sono disponibili un vasto numero di controlli, sia per la cinematica diretta che inversa, permettendo di agire su tutti gli arti e parti del corpo. Inoltre è possibile sfruttare gli avanzatissimi ed evoluti controlli del volto messi a disposizione per i manichini di MetaHuman.

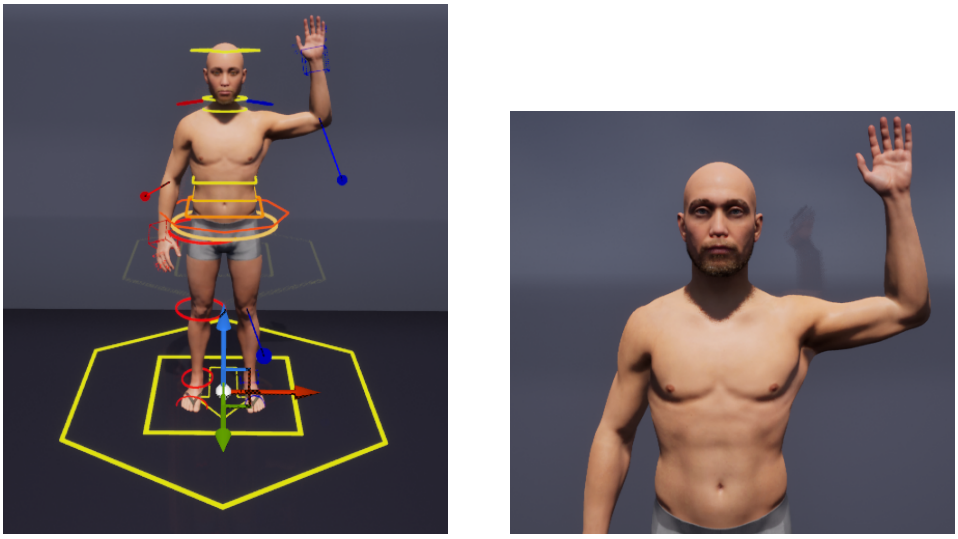


Figura 8.5: Esempio di posa e dei controlli del Control Rig per il manichino rappresentante il 1° percentile

8.2.1 Replica delle pose degli astronauti

Per mettere alla prova i manichini sviluppati e il sistema di messa in posa, gli umanoidi sono posizionati in modo che replicassero alcune delle reali pose degli astronauti all'interno della Stazione Spaziale Internazionale.

In particolare, sono state prese due immagini ritraenti l'ex astronauta statunitense, Daniel Burbank, e l'astronauta italiana, Samantha Cristoforetti, entrambi all'interno della stazione spaziale. Per ogni figura, è stata riprodotta la posa, utilizzando il sistema di rigging creato tramite Control Rig.

Si può notare come le pose siano altamente realistiche e fedeli a quelle originali, tramite l'utilizzo di controlli posti nelle gambe, braccia, mani, piedi e, addirittura, nelle dita delle mani. Anche in presenza di difficile pose, come quella dell'astronauta Cristoforetti in figura 8.6, il risultato ottenuto è più che soddisfacente e accurato.

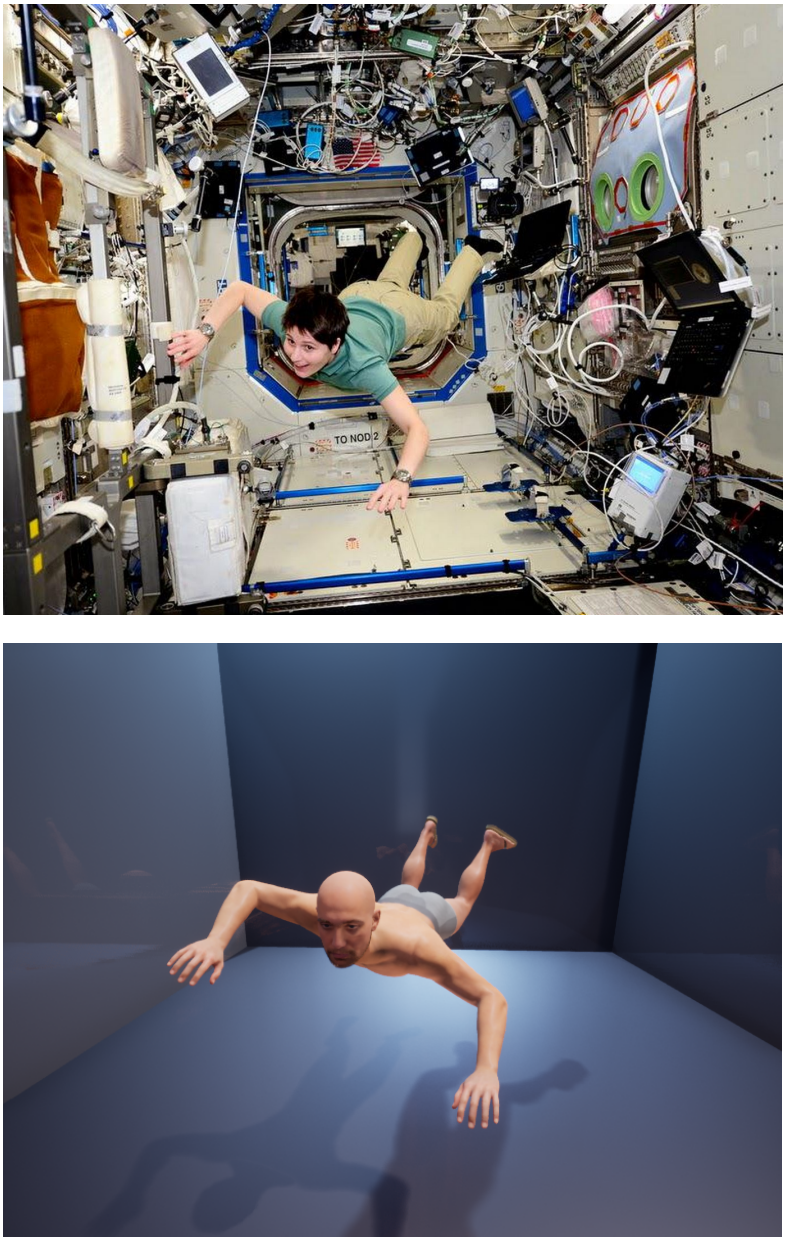


Figura 8.6: Replica della posa di Samantha Cristoforetti

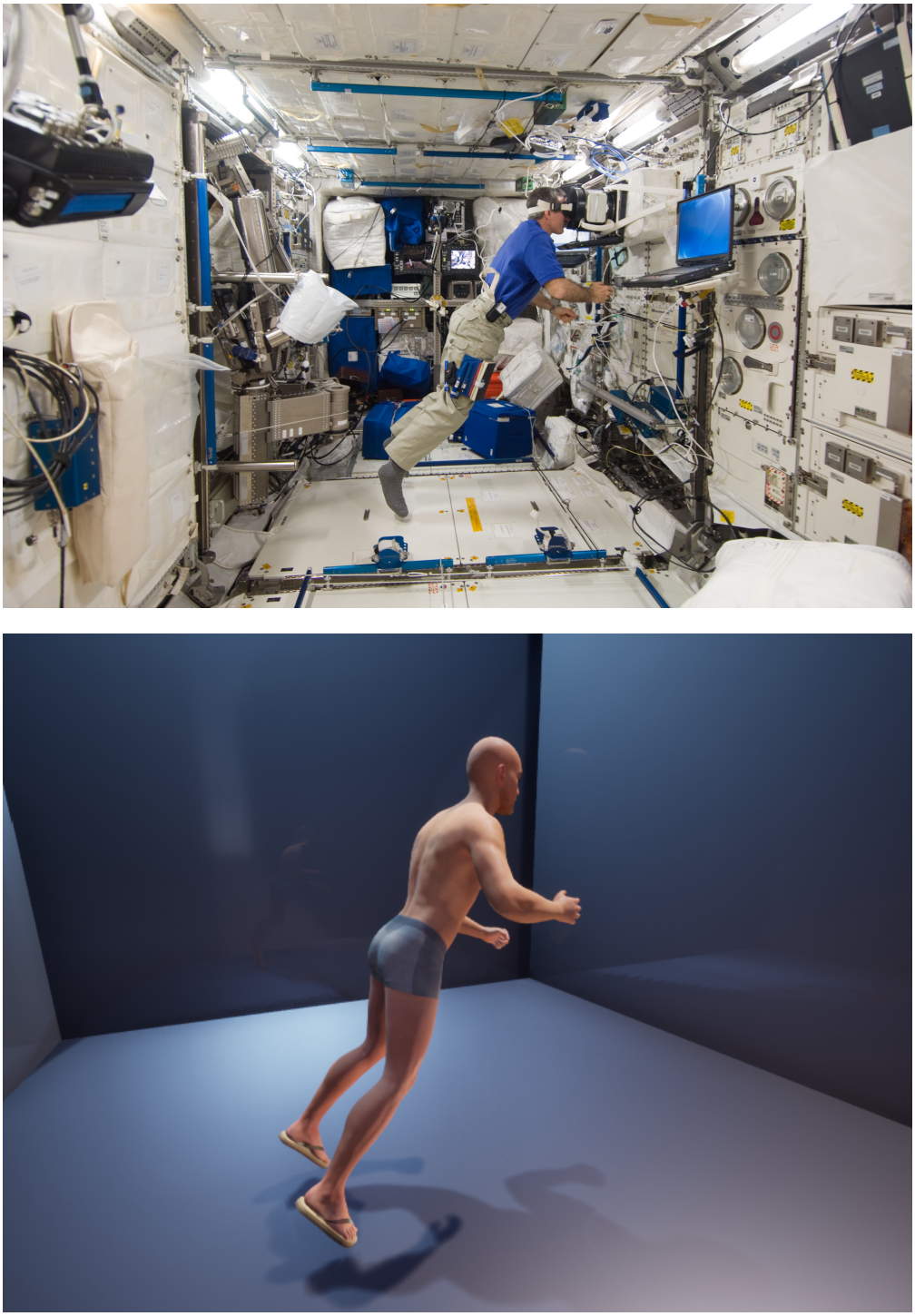


Figura 8.7: Replica della posa di Daniel Burbank

Parte IV

Conclusione e risultati

Capitolo 9

Risultati Modulo 1

9.1 Risultati

Il plugin sviluppato nel capitolo 4, per la gestione di parametri e dati di componenti in ambito aerospaziale, soddisfa gli obiettivi e i requisiti esposti all'inizio di questa tesi e richiesti da Thales.

In particolare, si è verificato il corretto funzionamento del modulo con un qualsiasi foglio di calcolo, che rispetta il formato standard definito, e con qualsiasi tipo di parametro. Il plugin sviluppato per Unreal Engine 4 permette una facile gestione dei dati, permettendo di importarli ed esportarli tramite un'intuitiva interfaccia grafica.

Per verificarne l'usabilità, sono stati svolti dei test soggettivi presso Thales, grazie all'aiuto di collaboratori interni. È stato quindi fornito un progetto di Unreal Engine con il plugin sviluppato già attivo, insieme ad una serie di file csv con la quale poter provare le varie funzionalità messe a disposizione dal modulo. Infine, gli utenti hanno compilato un questionario di gradimento, nella quale poter fornire un proprio riscontro ed opinione sull'usabilità del software.

9.1.1 Questionario utente

Il questionario utente, cui domande sono raccolte nella tabella 9.1, è stato proposto a dei collaboratori del COSE Centre che utilizzeranno attivamente il plugin sviluppato. Ad ogni domanda è possibile rispondere con una delle risposte mostrate in tabella 9.2. I riscontri ottenuti presentano dei risultati simili tra di loro.

L'utente del primo questionario ritiene di conoscere sufficientemente l'ambiente di sviluppo Unreal Engine 4. Inoltre, è pienamente d'accordo con tutte le affermazioni proposte nel questionario (tranne quella sull'affidabilità), di conseguenza afferma che il sistema sia molto facile da usare e molto intuitivo, confermando anche l'utilità del plugin nel proprio flusso di lavoro.

Domande questionario
1. Conosco molto bene Unreal Engine 4
2. Il sistema è molto facile da usare
3. L'interfaccia grafica è molto intuitiva
4. Le funzionalità offerte dal software sono molto semplici da capire
5. Trovo molto utile il software
6. Mi piacerebbe usare il software proposto per il mio lavoro
7. Il software è molto stabile (nessun bug o crash)
8. Hai dei suggerimenti per migliorare il software o altro da commentare?

Tabella 9.1: Domande del questionario utente per il Modulo 1

Risposte questionario
1. Pienamente d'accordo
2. D'accordo
3. Neutrale
4. Disaccordo
5. Pienamente in disaccordo

Tabella 9.2: Risposte possibili del questionario utente per il Modulo 1

Per quanto riguarda l'affidabilità del sistema, l'utente ritiene che il sistema è sufficientemente stabile e propone una serie di suggerimenti e migliorie. In particolare, il suggerimento si concentra sulla creazione di un controllo di integrità degli ID, associati agli oggetti di scena. Questo sarebbe utile per evitare eventuali collisioni che porterebbero i dati a non essere consistenti in seguito alle operazioni di importazione ed esportazione. Viene quindi proposto un controllo aggiuntivo ed una serie di messaggi e finestre, per notificare gli utenti di una eventuale collisione. Le risposte complete del primo utente sono mostrate in tabella 9.3.

Il secondo utente afferma di possedere un'ottima conoscenza di Unreal Engine. Inoltre è d'accordo con le affermazioni di intuitività e facilità di utilizzo, e pienamente d'accordo sull'utilità e stabilità del software. Non propone suggerimenti o commenti aggiuntivi. Le risposte complete del secondo utente sono mostrate in tabella 9.4.

Complessivamente i risultati mostrano un riscontro più che positivo, sia per l'usabilità, che utilità e affidabilità del plugin.

Domande	Risposte
1. Conosco molto bene Unreal Engine 4	D'accordo
2. Il sistema è molto facile da usare	Pienamente d'accordo
3. L'interfaccia grafica è molto intuitiva	Pienamente d'accordo
4. Le funz. del software sono molto semplici da capire	Pienamente d'accordo
5. Trovo molto utile il software	Pienamente d'accordo
6. Mi piacerebbe usare il software	Pienamente d'accordo
7. Il software è molto stabile (nessun bug o crash)	D'accordo

Tabella 9.3: Risposte del primo utente al questionario di gradimento

Domande	Risposte
1. Conosco molto bene Unreal Engine 4	Pienamente d'accordo
2. Il sistema è molto facile da usare	D'accordo
3. L'interfaccia grafica è molto intuitiva	D'accordo
4. Le funz. del software sono molto semplici da capire	Pienamente d'accordo
5. Trovo molto utile il software	Pienamente d'accordo
6. Mi piacerebbe usare il software	Pienamente d'accordo
7. Il software è molto stabile (nessun bug o crash)	Pienamente d'accordo

Tabella 9.4: Risposte del secondo utente al questionario di gradimento

9.2 Sviluppi futuri

Il Modulo 1 di questa tesi può essere ulteriormente migliorato sotto diversi punti vista:

- Integrazione con VERITAS: un primo semplice miglioramento potrebbe essere l'integrazione con l'intero pacchetto VERITAS, in modo da poter integrare il modulo sviluppato con tutte le altre funzionalità messe a disposizione da V4U;

- Interfaccia grafica dei dati: attualmente i dati sono memorizzati all'interno delle mesh statiche di un livello, tramite un componente creato appositamente. Durante l'esecuzione dell'applicazione, ciò che viene visualizzato è un Widget di UE4 nella quale viene stampato l'id della mesh stessa. Dei possibili sviluppi futuri sono sicuramente la visualizzazione dei dati veri e propri, in modo che siano facilmente accessibili dagli utilizzatori dell'applicazione;

- Miglior supporto per la Realtà Virtuale: il Widget per la visualizzazione dei dati potrebbe essere sviluppato in modo che sia facilmente accessibile ed utilizzabile tramite l'utilizzo di dispositivi per la Realtà Virtuale. Un esempio

potrebbe essere la visualizzazione dei dati del componente nel momento in cui un utente esegue un'azione, tramite puntatore, verso la mesh stessa, utilizzando dei controller (per esempio i dispositivi di input di un HTC Vive).

Capitolo 10

Risultati Modulo 2

10.1 Risultati

I manichini per il design ergonomico sono stati creati tramite lo strumento sviluppato all'interno di questa tesi. La nuova soluzione, che consiste nell'utilizzo di MetaHuman, Animation Blueprint e Control Rig di Unreal Engine 4, è stata proposta per risolvere alcuni dei problemi delle soluzioni attuali. I problemi principali riguardano realismo, animazione e usabilità/compatibilità dei manichini.

Inoltre, è importante verificare la correttezza delle misure dei nuovi umanoidi creati.

10.1.1 Verifica delle misure antropometriche

Gli umanoidi definiti devono ovviamente rispettare le misure antropometriche definite dallo standard NASA (sezione 7.3). Per fare ciò è stato sviluppato uno strumento di misurazione completamente integrato all'interno di Unreal Engine 4, lo sviluppo è presentato nel capitolo 6.

Per ogni misura dei due percentili creati (1° e 99°) è stato utilizzata un'istanza degli strumenti di misurazione e si è verificato che la misura corrispondesse a quella richiesta dal documento NASA. Nella tabella 10.1 vengono mostrate le misure e i corrispondenti strumenti utilizzati. Inoltre, in figura 10.1 vengono mostrate alcune delle verifiche delle misure antropometriche dei manichini.

L'alta precisione delle dimensioni è dovuta alla flessibilità dello strumento sviluppato, il quale da la possibilità di manipolare la mesh statica grazie a dei semplici parametri numerici. Tramite questi è possibile modificare le dimensioni a proprio piacimento, con una precisione che va oltre il millimetro.

Misura	Strumento utilizzato	1° Percentile	99° Percentile
Altezza	Righello	Verificato	Verificato
Profondità busto	Righello	Verificato	Verificato
Ampiezza vita post.	Righello	Verificato	Verificato
Ampiezza schiena	Righello	Verificato	Verificato
Circonferenza collo	Circonferenza ellisse	Verificato	Verificato
Lunghezza spalle	Righello	Verificato	Verificato
Altezza caviglia	Righello	Verificato	Verificato
Lunghezza piede	Righello	Verificato	Verificato
Ampiezza piede	Righello	Verificato	Verificato

Tabella 10.1: Dimensioni antropometriche dello standard NASA e strumenti utilizzati per la verifica dei percentili

(a) Altezza **(b)** Lunghezza spalla **(c)** Lunghezza piede

Figura 10.1: Esempi di verifica delle misure del manichino rappresentate il 1° percentile, tramite gli strumenti di misurazione sviluppati

10.1.2 Caratteristiche e confronto con soluzioni precedenti

Rispetto ai precedenti manichini utilizzati da Thales, la nuova soluzione proposta presenta diverse caratteristiche migliori e permette di risolvere diversi problemi presenti.

Realismo La prima miglioria visibile è sicuramente il maggior realismo dei nuovi manichini. Lo strumento MetaHuman infatti permette la creazione di umanoidi altamente realistici da utilizzare per applicazioni in tempo reale. Il realismo visivo dei modelli è importante soprattutto per i sistemi di ingegneria collaborativa, nella quale questi sono utilizzati come avatar virtuali degli utenti, che generalmente utilizzano dei dispositivi per la Realtà Virtuale.

In figura 10.2 possiamo vedere dei confronti tra i precedenti manichini e gli attuali. È bene ricordare che nuovi volti possono essere creati tramite MetaHuman

ed utilizzati in coppia allo strumento sviluppato in questa tesi, senza la necessità di dover ridefinire i percentili, permettendo quindi di poter creare un numero elevatissimo di modelli con sembianze (e sesso) del tutto diversi.

Figura 10.2: Confronto tra gli attuali manichini di Thales e i manichini sviluppati in questa tesi

Animazione I manichini attualmente in uso presso Thales sono animati e messi in posa tramite una logica di controllo definita all'interno di Blueprint e Animation Blueprint. Tuttavia questa soluzione non è del tutto completa e può portare delle incompatibilità con i sistemi di Unreal Engine.

Un'alternativa migliore consiste nell'uso del nuovo strumento di UE per il controllo degli scheletri, ovvero il Control Rig. La nuova soluzione infatti sfrutta le potenzialità di questo sistema, permettendo la messa in posa altamente realistica dei manichini, con una serie di controlli per la cinematica diretta e inversa. I controlli sono molteplici, tra cui abbiamo il controllo del busto, delle spalle, degli arti superiori, degli arti inferiori, delle dita delle mani, della testa. Inoltre è possibile mettere in posa il volto dei manichini in modo altamente realistico, tramite i controlli messi a disposizione per i manichini di MetaHuman.

Compatibilità e usabilità Infine, un grosso vantaggio sta nella compatibilità e usabilità dello strumento e dei manichini creati tramite questo. Infatti, come detto precedentemente, la creazione dei percentili viene fatta tramite la manipolazione degli scheletri degli umanoidi. Questo permette di poter definirlo una singola volta, utilizzando poi lo scheletro come base per tutti gli altri umanoidi con sembianze diverse.

Inoltre, gli strumenti utilizzati in questa tesi, come MetaHuman e Control Rig, sono completamente compatibili e supportati da Unreal Engine, permettendo un

miglioramento di questi grazie anche ai frequenti aggiornamenti rilasciati da Epic Games.

10.2 Sviluppi futuri

Il lavoro fatto in questa tesi può essere ulteriormente migliorato sotto diversi punti di vista, in questa sezione ne vengono presentati alcuni:

- Maggiori misure antropometriche: il miglioramento principale sta nella realizzazione e supporto di un maggior numero di misure antropometriche definite dallo standard NASA. In questa tesi si è scelto di utilizzare solo un sottoinsieme di queste, di conseguenza per una maggior conformità è possibile estendere lo strumento di creazione dei manichini, utilizzando il medesimo modello e processo presentato nei capitoli precedenti;
- Supporto per il sistema di ingegneria collaborativa: attualmente Thales utilizza un sistema per l'ingegneria collaborativa in Realtà Virtuale, il quale è disponibile all'interno del pacchetto VERITAS per Unreal Engine 4. In futuro sarebbe possibile integrare i nuovi manichini sviluppati all'interno di questo sistema, utilizzandoli come avatar virtuali degli utenti.

Bibliografia

[1] Gregor Burdea. *Definizione di Realtà Virtuale* (cit. a p. 2).

[2] Thales Group. *Attività in Italia*. URL: https://www.thalesgroup.com/it/italia/global-presence-europe/italia (cit. a p. 5).

[3] Thales Group. *Thales Alenia Space Italia*. URL: https://www.thalesgroup.com/it/worldwide/global-presence-europe-italy/spazio (cit. a p. 6).

[4] International Ergonomics Association. *What Is Ergonomics?* URL: https://iea.cc/what-is-ergonomics/ (cit. a p. 8).

[5] Wikipedia. *Unreal Engine*. URL: https://en.wikipedia.org/wiki/Unreal_Engine#Unreal_Development_Kit (cit. a p. 9).

[6] Epic Games. *Reimagined Film & Television Production Storytelling*. URL: https://www.unrealengine.com/en-US/solutions/film-television (cit. a p. 9).

[7] Epic Games. *Architecture Design Software*. URL: https://www.unrealengine.com/en-US/solutions/architecture (cit. a p. 10).

[8] Epic Games. *Advanced Automotive and Car Design Software and Visualization*. URL: https://www.unrealengine.com/en-US/solutions/automotive-transportation (cit. a p. 10).

[9] Epic Games. *Produce Broadcast & Live Events Mixed Reality Experience*. URL: https://www.unrealengine.com/en-US/solutions/broadcast-live-events (cit. a p. 10).

[10] Keef Sloan. *How NASA Trains Astronauts with Unreal Engine*. URL: https://www.unrealengine.com/en-US/developer-interviews/how-nasa-trains-astronauts-with-unreal-engine (cit. a p. 10).

[11] John Gaudiosi. *NASA Is Using Unreal Engine 4 To Make Mars A Virtual Reality*. URL: https://www.unrealengine.com/en-US/blog/nasa-is-using-unreal-engine-4-to-make-mars-a-virtual-reality (cit. a p. 10).

[12] Epic Games. *Features - Unreal Engine*. URL: `https://www.unrealengine.com/en-US/features` (cit. a p. 11).

[13] Epic Games. *Blueprint Visual Scripting*. URL: `https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/` (cit. a p. 15).

[14] Epic Games. *Plugins on Unreal Engine 4*. URL: `https://docs.unrealengine.com/4.27/en-US/ProductionPipelines/Plugins/` (cit. alle pp. 17, 18).

[15] Kristensson P. O. Tadeja S. K. Seshadri P. «AeroVR: An immersive visualisation system for aerospace design and digital twinning in virtual reality. The Aeronautical Journal». In: (2020) (cit. alle pp. 20, 21).

[16] Shukshunov V. E. Stone R. J. Panfilov P. B. «Evolution of aerospace simulation: From immersive Virtual Reality to serious games». In: (giu. 2020) (cit. alle pp. 21, 22).

[17] Davidoff S. Norris J. «Nasa telexploration project demo». In: (mar. 2014) (cit. a p. 22).

[18] Wallace B. Andrews T. Searcy B. «Using Virtual Reality and Motion Capture as Tools for Human Factors Engineering at NASA Marshall Space Flight Center». In: (lug. 2020) (cit. a p. 23).

[19] Andrew Jones. *What we know about China's space station: modules, crew, launch plans and more*. Mar. 2018. URL: `https://findchina.info/what-we-know-about-chinas-space-station-modules-crew-launch-plans-and-more` (cit. a p. 25).

[20] European Space Agency (ESA). *OCDT - Open Concurrent Design Tool*. URL: `https://ocdt.esa.int/` (cit. alle pp. 26, 27).

[21] National Aeronautics e Space Administration (NASA). *Gateway Program*. URL: `https://www.nasa.gov/gateway/overview` (cit. a p. 28).

[22] National Aeronautics e Space Administration (NASA). *Gateway Program Human System Requirements (HSR) - GP 10017* (cit. a p. 30).

[23] Python Software Foundation. *General Python FAQ - What is Python?* URL: `https://docs.python.org/3/faq/general.html#what-is-python` (cit. a p. 33).

[24] Wikipedia. *Interprete (informatica)*. URL: `https://it.wikipedia.org/wiki/Interprete_(informatica)` (cit. a p. 33).

[25] Python Software Foundation. *General Python FAQ - What is Python good for?* URL: `https://docs.python.org/3/faq/general.html#what-is-python-good-for` (cit. a p. 33).

[26] Python Software Foundation. *tkinter - Python interface to Tcl/Tk*. URL: `https://docs.python.org/3/library/tkinter.html` (cit. a p. 34).

[27] Python Software Foundation. *csv - CSV File Reading and Writing*. URL: https://docs.python.org/3/library/csv.html (cit. a p. 34).

[28] Epic Games. *Scripting the Editor using Python*. URL: https://docs.unrealengine.com/4.27/en-US/ProductionPipelines/ScriptingAndAutomation/Python/ (cit. alle pp. 34, 35).

[29] Epic Games. *Unreal Python API Documentation*. URL: https://docs.unrealengine.com/4.27/en-US/PythonAPI/ (cit. a p. 36).

[30] Epic Games. *Widget Components*. URL: https://docs.unrealengine.com/4.26/en-US/Basics/Components/Widget/ (cit. a p. 38).

[31] Epic Games. *Creating and Displaying UI*. URL: https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/UMG/HowTo/CreatingWidgets/ (cit. a p. 38).

[32] Epic Games. *Components - Registering Components*. URL: https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/Actors/Components/#registeringcomponents (cit. a p. 39).

[33] Epic Games. *GEngine - Unreal Engine Documentation*. URL: https://docs.unrealengine.com/4.27/en-US/API/Runtime/Engine/Engine/UEngine/ (cit. a p. 43).

[34] Epic Games. *FUICommandList - Unreal Engine Documentation*. URL: https://docs.unrealengine.com/4.27/en-US/API/Runtime/Slate/Framework/Commands/FUICommandList/ (cit. a p. 43).

[35] Epic Games. *IModuleInterface::StartupModule - Unreal Engine Documentation*. URL: https://docs.unrealengine.com/4.27/en-US/API/Runtime/Core/Modules/IModuleInterface/StartupModule/ (cit. a p. 43).

[36] Epic Games. *UToolMenus::RegisterMenu - Unreal Engine Documentation*. URL: https://docs.unrealengine.com/4.26/en-US/API/Developer/ToolMenus/UToolMenus/RegisterMenu/ (cit. a p. 43).

[37] Epic Games. *FToolBarBuilder - Unreal Engine Documentation*. URL: https://docs.unrealengine.com/4.27/en-US/API/Runtime/Slate/Framework/MultiBox/FToolBarBuilder/ (cit. a p. 44).

[38] Epic Games. *FMenuBuilder - Unreal Engine Documentation*. URL: https://docs.unrealengine.com/4.27/en-US/API/Runtime/Slate/Framework/MultiBox/FMenuBuilder/ (cit. a p. 44).

[39] Epic Games. *Animation Blueprints*. URL: https://docs.unrealengine.com/4.27/en-US/AnimatingObjects/SkeletalMeshAnimation/AnimBlueprints/ (cit. a p. 47).

[40] Epic Games. *Animation Blueprint Editor*. URL: https://docs.unrealengine.com/4.27/en-US/AnimatingObjects/SkeletalMeshAnimation/AnimBlueprints/Interface/ (cit. a p. 48).

[41] Epic Games. *EventGraph - Animation Blueprint*. URL: https://docs.unrealengine.com/4.27/en-US/AnimatingObjects/SkeletalMeshAnimation/AnimBlueprints/EventGraph/ (cit. a p. 49).

[42] Epic Games. *AnimGraph - Animation Blueprint*. URL: https://docs.unrealengine.com/4.27/en-US/AnimatingObjects/SkeletalMeshAnimation/AnimBlueprints/AnimGraph/ (cit. a p. 50).

[43] Epic Games. *Control Rig*. URL: https://docs.unrealengine.com/4.27/en-US/AnimatingObjects/SkeletalMeshAnimation/ControlRig/ (cit. a p. 51).

[44] Epic Games. *Control Rig Blueprints - Control Rig*. URL: https://docs.unrealengine.com/4.27/en-US/AnimatingObjects/SkeletalMeshAnimation/ControlRig/Blueprint/ (cit. a p. 51).

[45] Epic Games. *MetaHuman Creator*. URL: https://docs.metahuman.unrealengine.com/en-US/UserGuide/ (cit. alle pp. 52, 53, 65).

[46] Epic Games. *Sequencer Overview - Unreal Engine Documentation*. URL: https://docs.unrealengine.com/4.26/en-US/AnimatingObjects/Sequencer/Overview/ (cit. a p. 93).