Università degli studi di Cagliari

Facoltà di Scienze MM. FF. NN.

Corso di Laurea in Informatica

H_2Mo_4
a Virtual Laboratory for Direct Manipulation

Docenti di riferimento
Prof. Riccardo Scateni
Prof. Davide Spano

Candidati
Alessandro Carcangiu
Alessio Murru

Anno Accademico 2012/2013

Titolo: H_2Mo_4: a Virtual Laboratory for Direct Manipulation

Relatori: Prof. Riccardo Scateni e Davide Spano

Autori: Alessandro Carcangiu (60/61/44961) e Alessio Murru (60/61/45689)

Tipo tesi: Progetto ed implementazione di un prototipo

Abstract: The *Leap Motion* is a new device that permits the detection and recognition of hands and their fingers, on the three-dimensional space; The leap is consists of two cameras and three *IR leds* which allow reading objects until one meter of distance from the device. The objective of thesis is to study and prove the capability of this new device, by building an environment that can be used only the objects detected from leap motion. The selected environment is an atomic laboratory, through which the user can manipolate the atoms choosen for building, analizing and visualizing general information about molecules and other objects (such as valence electrons, relative atomic mass, electron configuration and ecc.). The application aims to create a pleasant environment for learning created for a quickly and funny learning a how atoms connect to each other, but also for having a look at the chemistry world wchich can be adapted to an educational context for students. This system is implemented using *Unity3D*, a cross-platform game engine with a built-in IDE developed by *Unity Technologies*, whereas for relevated the movements detection, we exploited the library *GestIT*.

A Gianlu

Indice

Capitolo 1

Introduzione

1.1 Obiettivo della Tesi

Obiettivo primario della tesi è quello di studiare e mostrare quali possono essere le opportunità offerte dal nuovo dispositivo *Leap Motion* nell'ambito dell'interazione uomo-macchina. Per fare ciò, è stato deciso di sviluppare un laboratorio atomico virtuale, nel quale l'utente può interagire con il sistema tramite il nuovo device. Per sfruttare le capacità del *leap motion*, abbiamo integrato nell'applicazione una versione aggiornata della libreria *GestIT* (che include le funzioni per la comunicazione con il *leap*) che è stato utilizzata per la creazione e il riconoscimento di nuovi gesti (*gesture*).

Come ambiente di sviluppo per la realizzazione del laboratorio è stato utilizzato *Unity3D*, uno strumento di authoring integrato multipiattaforma per la creazione di videogiochi 3D e altri contenuti interattivi. Il laboratorio è stato pensato per permettere all'utente di avvicinarsi al mondo dell'atomo e delle molecole, e poter ricevere informazioni su di esso in una maniera nuova ma allo stesso tempo intuitiva grazie all'utilizzo del *leap motion*, offrendogli tutti gli strumenti necessari per la navigazione nell'ambiente virtuale e la selezione degli atomi presenti. Selezionando un atomo, l'utente potrà focalizzare la visuale su di esso, quindi poterne osservare le sue proprietà chimiche, spostare la visuale rispetto alle coordinate X, Y e Z (sfruttando, quindi, le possibilità offerte da un ambiente tridimensionale) in base alla posizione delle dita, visualizzare tutti i suoi elettroni o solo quelli di valenza, oppure tramite delle gestures, effettuare l'ingradimento della visuale (tramite il gesto pinch, che consiste nell'allontanare o avvicinare due dita tra loro), oppure ruotare la visuale in base alla posizione delle due mani o ancora applicare una forza centripeta all'atomo e ai suoi elettroni. Inoltre all'utente viene data la possibilità di poter creare dei legami tra i vari atomi, semplicemente selezionando prima un atomo, e poi l'altro atomo con cui si vuole formare il legame. La tesi è suddivisa in 6 capitoli. Nel capitolo 2 vengono esposti il concetto di interazione uomo-macchina, lo stato dell'arte, le caratteristiche dei device utilizzati e degli ambienti di sviluppo utilizzati; nel capitolo 3 viene analizzata l'applicazione sviluppata e si esaminano le scelte progettuali fatte; nel capitolo 4 vengono descritti i risultati ottenuti e i test effettuati; nel capitolo 5 vengono esposti i possibili sviluppi futuri dell'applicazione e del *leap*; infine nel capitolo 6 vengono discussi i problemi riscontrati durante lo sviluppo e le conclusioni a cui si è arrivati.

Capitolo 2

Strumenti e Background

2.1 Interazione Uomo-Macchina

Con il termine interazione uomo-macchina (o *HCI*, *Human-Computer Interaction*), si indica quella disciplina dell'informatica che si occupa di studiare e analizzare gli aspetti fisici, psicologici e teorici dell'interazione tra un utente, o gruppi di utenti, e una macchina, allo scopo di poter aumentare l'usabilità e l'affidabilità di quest'ultimi e dei sistemi interattivi che li utilizzano; per utente si intende chiunque tenti di eseguire un compito o un lavoro con l'utilizzo della tecnologia, mentre per macchina si può intendere un computer, un cellulare, un sistema di controllo del processo o un sistema incorporato.

L'interazione uomo-macchina è una materia multidisciplinare, che coinvolge non solo l'informatica ma anche altre discipline, come la psicologia e le scienze cognitive, per lo studio sulle capacità concettuali, cognitive e di risoluzione dell'utente, l'ergonomia per conoscere le capacità fisiche degli utenti, il design in modo da poter sviluppare delle interfacce più efficaci ed efficienti, l'intelligenza artificiale ecc.
11 Il termine *HCI* è diventato di utilizzo generico già dagli anni '80, ma pone le sue basi sulle ricerche effettuate negli anni '40, durante la Seconda Guerra Mondiale, allo solo scopo di produrre sistemi d'armi sempre più efficienti. Queste ricerche portarono a un'ondata di interesse nel campo dell'interazione uomo-macchina tra i ricercatori, tanto che nel 1949 venne fondata la Società di Ricerca in Ergonomia. Inoltre con il diffondersi dei computer, all'inizio solo nel mondo del lavoro ma ben presto anche in tutti gli altri aspetti della vita quotidiana, la disciplina in questione acquistava sempre più importanza, attraendo un numero crescente di ricercatori specializzati nello studio dell'interazione tra persone e computer.

2.2 Stato dell'Arte

Il *leap motion*, come già accennato, è un dispositivo di nuova generazione che non è ancora entrato in commercio. Per questo motivo non esistono ancora delle applicazioni ufficiali, tuttavia la società produttrice ha già messo a disposizione dei ricercatori e degli sviluppatori che ne hanno fatto richiesta, l'SDK (*Software Development Kit*, ovvero un insieme di strumenti per lo sviluppo e la documentazione di applicazioni) e un prototipo

quasi definitivo del device, quindi è possibile trovare sulla rete diversi esempi e demo di applicazioni che si poggiano sull'utilizzo del *leap motion* per l'interazione dell'utente con il sistema. Ad ogni modo si può affermare che la tesi in oggetto è la prima ad applicare l'utilizzo del *leap* in un laboratorio atomico. È possibile trovare sulla rete diverse applicazioni che simulano un laboratorio molecolare, come ad esempio il *Virtual Molecular Dynamics Laboratory [3]* sviluppato dalla *CPS (Center for Polymer Studies)* dell'Università di Boston, che permette di poter visualizzare il movimento di un atomo, di manipolare le interazioni atomiche, nonché accedere alle proprietà biologiche, chimiche e fisiche di un atomo o di una molecola.

2.3 Leap Motion

Il *Leap Motion*[4] è un nuovo dispositivo, ancora in fase di sviluppo, in grado di rilevare e tracciare il movimento delle mani e delle dita dell'utente, nonché strumenti simili a quest'ultimi, come penne o bacchette, in maniera accurata. Obiettivo del *leap motion* è quello di aumentare le possibilità di interazione uomo-macchina, soprattutto nel mondo 3D, nella quale l'utente trova sicuramente piú naturale poter interagire direttamente attraverso l'uso delle mani, piuttosto che con mouse e tastiera. Una prima versione del dispositivo per gli sviluppatori è stata rilasciata già nel Gennaio scorso, attualmente il device e la relativa SDK sono alla versione 0.7.9, e per fine Luglio è previsto un primo lancio commerciale del prodotto.

Figura 2.1: Il *Leap Motion*

Il *leap* si presenta come un piccolo box in alluminio, è dotato di due sensori *CMOS* (acronimo di *Complementary Metal-Oxide Semiconductor*), che funzionano come due telecamere e che sono indispensabili per la visione stereoscopica, e di tre LEDs all'infrarosso, che hanno il compito di irradiare gli oggetti che rientrano nel campo visivo delle due telecamere; il campo di rilevamento del dispositivo è costituito da una piramide inversa, centrata sullo stesso, e che si estende, all'incirca, dai 25 ai 600 mm di distanza dal device.

Figura 2.2: Una visuale di tutte le componenti del *leap motion*

2.3.1 Funzionamento e SDK

Sistema di Coordinate

Il *leap* opera sulla parte destra del piano cartesiano, e il punto d'origine coincide con il centro del *leap motion*: l'asse X è orizzontale rispetto a quest'ultimo, quindi cresce da sinistra verso destra, l'asse Z invece assume valori positivi con l'allontanarsi dal dispositivo, e infine l'asse Y si sviluppa in maniera verticale, non può assumere valori negativi e, come nel caso dell'asse Z, cresce con l'aumentare della distanza dal *leap*.

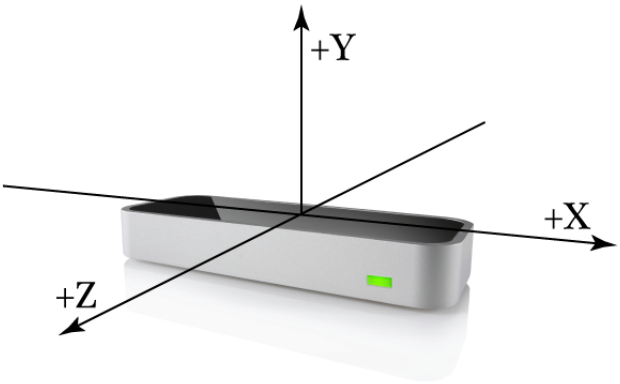

Figura 2.3: Il sistema di coordinate del *Leap*

Rilevamento Dati

Tutte le informazioni degli oggetti, dei movimenti e delle gesture rilevate dal *leap motion*, vengono registrate su dei frame. All'interno di ogni frame, ai vari oggetti e gesture tracciati, vengono assegnati degli ID univoci, che vi rimarranno associati per tutto il tempo che resteranno visibili al dispositivo, ciò significa che se l'oggetto viene perso e poi di nuovo tracciato, gli verrà assegnato un nuovo identificativo.
Un frame, più precisamente, contiene al suo interno una lista per ciascuno di questi oggetti:

- **Hands** - contiene tutte le mani rilevate; da essi è possibile accedere alle dita associate a quella mano;
- **Pointables** - in questa lista vengono elencati tutte le dita e gli strumenti rilevati;
- **Fingers** - qui troviamo le dita rilevato dal dispositivo;
- **Tools** - una lista degli strumenti o oggetti che hanno una forma simile alle dita, come penne o bacchette;
- **Gesture** - in questa lista vengono riportati le informazioni riguardanti tutte le *gesture* iniziate, finite o ancora in aggiornamento registrate.

La comunicazione *leap*/programma avviene tramite l'istanziazione di un oggetto della classe *Controller*, appartenente alla libreria del dispositivo, attraverso cui è possibile accedere alla lista di tutti i frame inviati dal *leap* da quando è stato attivato, la lista degli schermi collegati alla macchina da cui si utilizza il *leap motion* e a cui si può accedere tramite la classe *Screen*, e altre funzioni come la possibilità di ricalibrare lo stesso *leap*.
Per poter accedere alle liste indicate sopra, la libreria del *leap motion*, mette a disposizione sostanzialmente due metodi: il primo è specificare direttamente l'ID dell'oggetto a cui si vuole accedere, il secondo è effettuare una chiamata a funzione che, restituisce una lista, e contenente tutti gli oggetti dello stesso tipo rilevati:

- **Frame.hand()**
- **Frame.finger()**
- **Frame.tool()**
- **Frame.pointable()**
- **Frame.gesture()**

Nel qual caso il *leap* non rilevi uno di questi oggetti, restituirà semplicemente una lista vuota.

Frame Motion

Il *leap* rileva i movimenti di mani e strumenti confrontando il frame attuale con quelli immediatamente precedenti, per ottenere le informazioni necessarie per determinare la traslazione, la rotazione e la scala dei fattori di tutti gli oggetti che rientrano nel campo visivo del *leap motion*. Gli attributi che descrivono il movimento sono:

- **rotationAxis** - esprime l'asse di rotazione;
- **rotationAngle** - l'angolo di rotazione;
- **rotationMatrix** - una matrice di trasformazione che esprime la rotazione;
- **scaleFactor** - rappresenta la contrazione o l'espansione del movimento;
- **translation** - esprime il movimento lineare.

Oggetti Rilevabili

Hand - L'SDK del *leap motion* permette di poter accedere a molte informazioni relative a una mano rilevata, ovvero, la sua posizione, la norma e la velocità, i movimenti eseguiti, nonché le dita della mano stessa (se queste sono visibili). Il dispositivo è in grado di rilevare e gestire sia i casi in cui la mano risulta essere è chiusa (e in questo caso non vengono rilevate delle dita), sia quando sono visibili più di due mani, però in quest'ultimo caso a discapito di velocità e qualità delle informazioni; tuttavia non è in grado di determinare se la mano visualizzata è sinistra o destra. Gli attributi di una mano accessibili tramite il frame inviato dal *leap* sono:

- **palmPosition** - indica la distanza del centro della mano dal *leap motion*, espresso in mm;
- **palmVelocity** - rappresenta la velocità di movimento del palmo della mano, in mm/s;
- **palmNormal** - il valore della norma ricavato dalla posizione del palmo della mano rispetto al dispositivo;
- **direction** - la direzione del palmo della mano, calcolata a partire dalle sue dita, rispetto al dispositivo;
- **sphereCenter** - il centro della sfera ottenuta dalla curvatura della mano;
- **sphereRadius** - il raggio della sfera ottenuta dalla curvatura della mano;
- **Pointables** - restituisce la lista di tutte le dita e gli strumenti associati a quella mano;
- **Fingers** - restituisce solo le dita;
- **Tools** - restituisce solo gli strumenti.

Finger e Tool - Come già accennato, il *leap* distingue gli oggetti orientabili (o pointable) in dita e strumenti (o tools); questa distinzione avviene in base alla forma, il *leap* riconosce uno strumento come un oggetto piú lungo, piú sottile e piú dritto di un dito.
Tutte le caratteristiche delle dita e degli strumenti, sono eriditate dalla classe Pointable, e sono:

- **length** - la lunghezza, in millimetri, della porzione visibile dell'oggetto in questione.
- **width** - indica la larghezza media, in millimetri, della porzione visibile di un oggetto orientabile
- **direction** - restituisce la direzione a cui punta l'oggetto, rispetto al dispositivo
- **tipPosition** - restituisce la posizione della punta dell'oggetto, in millimetri, dal *leap motion*
- **tipVelocity** - riporta la velocità della punta dell'oggetto in mm/s.

Gesture - Il *leap motion* è in grado di riconoscere, tracciare e registrare alcune sequenze di movimenti; per poter tracciare le *gesture*, occorre prima attivarle tramite la funzione *enableGesture()* della classe *Controller*.
Quando il *leap* rileva una di queste sequenze di movimenti, inserisce la gesture all'interno di un frame. Se la gesture è continua, e si prosegue ad eseguirla, il dispositivo provvede ad aggiornare la gesture nel frame successivo. Le *gesture circle* e *swipe* sono continue, mentre il *key tap* e lo *screen tap* sono discrete, quindi il *leap motion* riferirà di ogni tap come una singola gesture.
Le gesture che il dispositivo è in grado di riconoscere di default sono:

Circle - un singolo pointable traccia un cerchio nello spazio; la libreria del *leap motion* permette di poter determinare il vettore del cerchio tracciato, il numero di volte che il cerchio è stato aggiornato (e quindi eseguito), il suo raggio e il dito che lo ha eseguito.

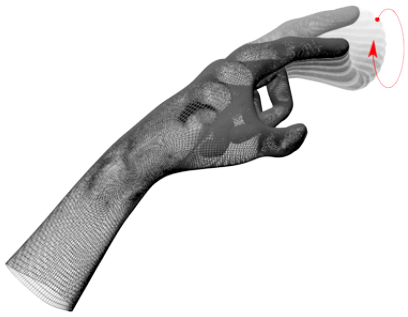

Figura 2.4: La *gesture circle* eseguita con un dito

Swipe - un movimento lineare della mano lungo l'asse X; in questo caso attraverso la libreria possiamo accedere alla sua direzione, la velocità in mm/s, il dito o il tool che lo ha eseguito, la posizione attuale dello swipe e la sua posizione di partenza.

Figura 2.5: La *gesture swipe*

Key Tap - un movimento di tapping del dito, in maniera simile alla pressione su un tasto della tastiera; gli attributi pubblici della classe *key tap* sono la direzione dell'oggetto che esegue la *gesture*, la posizione in cui viene eseguito e il *pointable* che lo esegue.

Figura 2.6: La *gesture key tap* eseguita con un dito

Screen Tap - un movimento di tapping del dito, in maniera simile alla pressione verticale su touch-screen; come già visto con il *key tap*, gli attributi dello *screen tap* sono direzione, posizione e *pointable*.

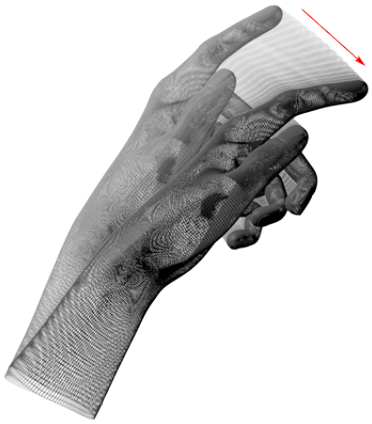

Figura 2.7: Lo *screen tap* eseguita con un dito

2.3.2 Applicazioni

Non essendo ancora entrato in commercio, non esistono delle applicazioni ufficiali per il *leap motion*, tuttavia in questi otto mesi di prova gli sviluppatori e i ricercatori che hanno già ricevuto il dispositivo, hanno potuto saggiare le proprietà del nuovo device e sviluppare degli applicativi che spaziano su diversi ambiti, da quello video-ludico (con la riproduzione virtuale del famoso gioco carta-forbici-sasso), al painting (dove si può disegnare direttamente a mano), da quello robotico/spaziale (la *NASA* ha presentato una versione del robot *ATHLETE* [7], acronimo di *All-Terrain Hex-Legged Extra-Terrestrial*

Explorer e progettato per l'esplorazione spaziale, telecomandato a distanza da un *leap motion*), all'esplorazione di realtà virtuali (come nel caso di *Google Earth*), o ancora, applicativi che permettono all'utente di poter interagire, tramite il *leap*, con un sistema operativo, e sulla rete è possibili trovare tanti altri applicativi in sviluppo, o che attendono solo il lancio ufficiale del *leap motion*. Quindi la caratteristica in comune offerta da tutti questi applicativi, è la possibilità di poter interagire direttamente e completamente con il sistema tramite il solo *leap motion*.

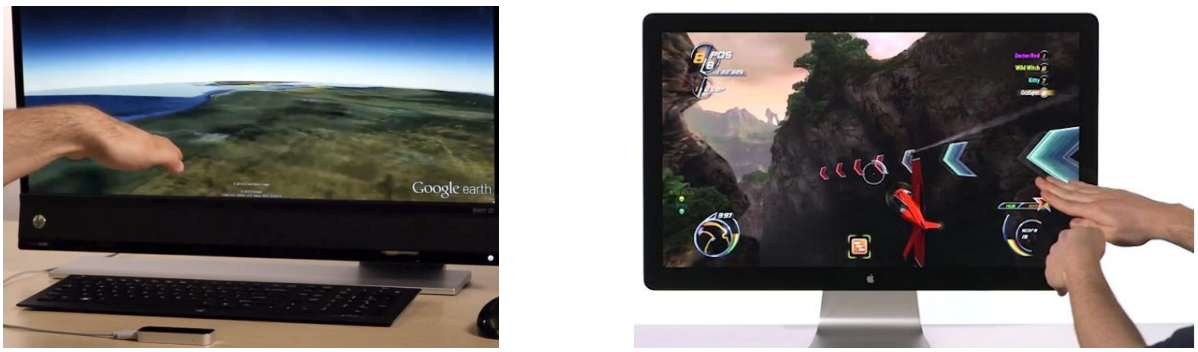

Figura 2.8: Esempi di applicazioni per il *Leap Motion*

2.4 Unity3D

Unity3D, sviluppato dalla *Unity Technologies*, è un sistema multi-piattaforma che ha aperto la frontiera dello sviluppo video-ludico, o più in generale, delle applicazioni multimediali bidimensionali e tridimensionali anche ai più piccoli sviluppatori, offrendo un motore grafico molto potente e intuitivo, accompagnato da una documentazione ufficiale che illustra in maniera esaudiente il funzionamento e le meccaniche di utilizzo di *Unity3D*, un Asset Store dal quale scaricare o comprare script già pronti per l'utilizzo con innumerevoli funzioni, ed una community ufficiale sempre attiva nella quale gli sviluppatori esperti e non si aiutano per migliorarsi a vicenda.
I progetti sviluppati in *Unity3D* godono di una grande portabilità, infatti lo sviluppatore potrà compilare il proprio progetto in più versioni senza apportare modifiche al codice, il che facilita ad esempio la creazione di un applicazione compatibile sia per Windows che per Mac cambiando solo le caratteristiche di compilazione, fattore che ha permesso una così grande notorietà di questo motore grafico, abbattendo le barriere imposte dei vari sistemi operativi.
Nello specifico *Unity3D* permette lo sviluppo nelle seguenti piattaforme:

- **Desktop**
  - Windows
  - Linux (dalla versione 4.0 di *Unity3D*)
  - Mac

- **Mobile**
  - Android
  - iOS
- **Console**
  - Sony PS3 (richiesta licenza del produttore hardware)
  - Microsoft XBox360 (richiesta licenza del produttore hardware)
  - Nintendo WiiU (richiesta licenza del produttore hardware)
- **Browser**
  - Chrome
  - Firefox
  - Safari
  - Internet Explorer

Unity3D è disponibile per il download in due versioni, la versione Free e la versione Pro. Le due versioni si differenziano per il numero di servizi offerti, ad esempio la versione Pro permette di integrare nell'applicazione delle librerie esterne, opzione non prevista nella versione Free.

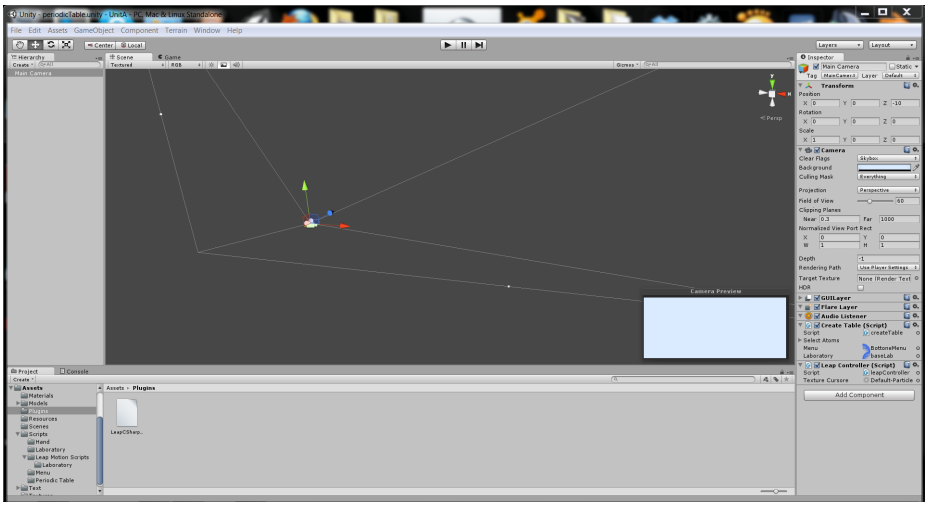

Figura 2.9: Esempio dell'interfaccia di *Unity3D Free*

2.4.1 Funzionamento

Unity3D offre due tipologie di interazione di sviluppo da parte dell'utente, la manipolazione diretta della scena e l'uso di script collegati ad oggetti della scena. La manipolazione diretta della scena offre diverse viste dell'ambiente, l'elenco dei vari *GameObject* presenti e un *Ispector* contenente l'insieme di *Components* del *GameObject* selezionato, con script associati e proprietà. Questa modalità viene usata principalmente per creare il momento iniziale del livello, vengono disposti i *GameObject* statici nell'ambiente, si associano gli script che dovranno essere eseguiti immediatamente e si potranno cambiare le loro caratteristiche iniziali, non sarà possibile però far interagire gli oggetti tra di loro o modificare dinamicamente la posizione, insomma non si può dare la vita alla scena.

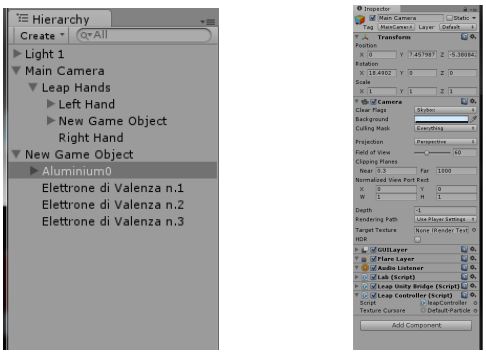

Figura 2.10: A sinistra viene mostrato in dettaglio un'esempio di lista gerarchica di *Unity3D*, a destra i *Components* associati ad una *Main Camera*

L'uso degli Script è di fondamentale importanza per lo sviluppo in *Unity3D*, gli script permettono qualsiasi modifica alla scena come la creazione della *GUI* e la manipolazione dei *GameObject*, la loro creazione o distruzione e l'associazione di nuovi *Component*. Gli script rendono dinamica la scena statica creata tramite la manipolazione diretta. I linguaggi di programmazione supportati da *Unity3D* sono:

- **C++**;
- **C#**;
- **JavaScript**;
- **Boo**.

2.4.2 GameObject

L'elemento più importante di un progetto sviluppato in *Unity3D* è il *GameObject*, è la classe base di qualsiasi entità presente nella scena, che sia una semplice sfera, il personaggio di un eventuale videogioco, la *Main Camera* o una fonte di luce. Questa sua flessibilità di utilizzo, che spazia in tipi di oggetti così differenti tra loro, è dovuta alla sua struttura, infatti il *GameObject* può essere visto come un contenitore di *Components* inizialmente vuoto, con l'aggiunta di nuovi *Components* l'oggetto acquisisce nuove caratteristiche fisiche e le azioni.

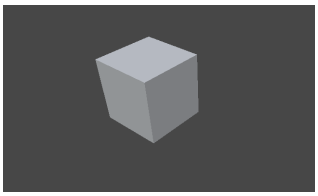

Figura 2.11: Esempio di un *GameObject* con la *mesh* di un cubo

Un *Component* può rappresentare la forma GameObject cui è associato, come una nuova *Mesh*, o delle reazioni e interazioni con l'ambiente, come ad esempio il *RigidBody*, che conferisce le caratteristiche fisiche come la massa o la gravità. Uno Script viene considerato da *Unity3D* come un *Component* creato dall'utente.
L'inserimento, la modifica o l'eliminazione di un *Component* può avvenire tramite manipolazione diretta o tramite Script. Il primo metodo risulta statico, quindi il *Component* avrà sin dall'avvio del programma i valori impostati, mentre la modifica tramite script permette una modifica dinamica, condizionata da eventi e situazioni diverse.

2.5 Blender

Blender [9] è un software opensource per la modellazione, il ridding, il rendering, l'animazione e il post-produzione di immagini tridimensionali.
Ideato come applicazione proprietaria freeware dallo studio di animazione NeoGen, ne fu poi ceduto l'incarico di sviluppo all'azienda creata dal suo ideatore Ton Roosendaal, la *Not a Number*, dal 1998 al 2002, anno della bancarotta della società. Da quell'anno, *Blender* divenne un progetto opensource della *Blender Corporation*, con il passaggio da software proprietario freeware a opensource si ebbe un notevole miglioramento di funzionalità ed usabilità dovuta alla grande community che lo segue e lo migliora in maniera costante.

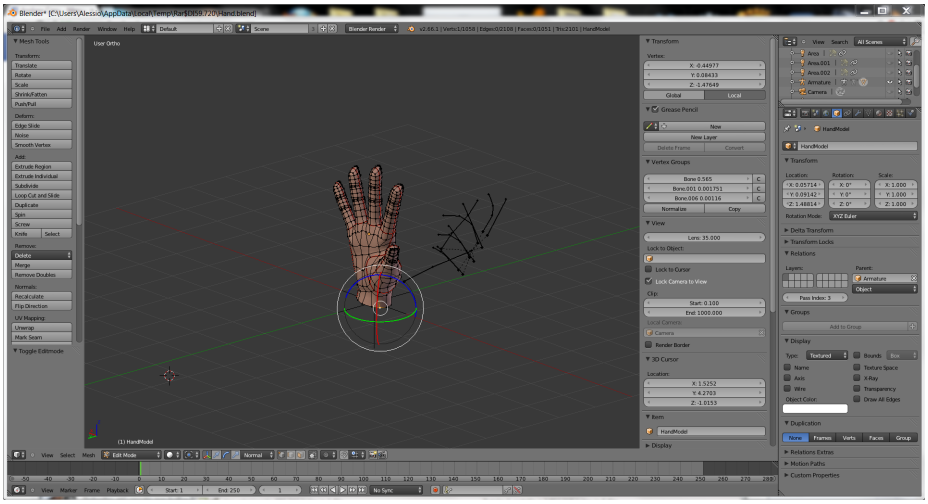

Figura 2.12: Interfaccia in *Edit Mode* di *Blender*

2.5.1 Funzionamento

Si tratta di un programma molto leggero e multipiattaforma, ma nonostante ciò possiede un gran numero di funzionalità presenti anche in noti programmi di modellazione grafica professionali e molto costosi, come *3D Studio Max* o *Maya*.
Blender ha la fama di essere un software di modellazione complesso a causa della miriade di combinazioni di tasti utilizzate in principio per richiamare le funzioni, senza aver la possibilità di una *GUI* sufficientemente esaustiva per compiere le medesime operazioni, problematica risolta da quando il progetto è diventato opensource, e che ha permesso l'avvicinamento al software di un'utenza più ampia.

L'utente può interagire con l'oggetto che vuole costruire, visibile tramite una o più viste della scena di costruzione, attraverso due modalità, la *Modalità oggetto* e la *Modalità modifica*. La prima di queste modalità permette la modifica dei singoli oggetti della scena come il suo spostamento, rotazione o scalatura. La seconda modalità permette una modifica più radicale dell'oggetto selezionato, andando a variare direttamente una sottoparte dei vertici o facce della mesh. Tutte le operazioni possono essere compiute sia in *Modalità oggetto* che in *Modalità modifica*, possono essere richiamate attraverso delle combinazioni di tasti, con variazioni nei parametri della funzione attraverso lo spostamento del mouse per modifiche approssimative e rapide, o dalla GUI con l'inserimento diretto dei valori desiderati, permettendo una precisione più elevata o il "*dragging*"del medesimo campo per un inserimento più rapido e più interattivo che mostra il progredire della modifica.
Inoltre la *GUI* di *Blender* è completamente personalizzabile, può essere scomposta in più sezioni per la visione simultanea di varie viste della scena ed ogni vista permette la normale interazione con gli oggetti. Ogni elemento dell'interfaccia può essere modificato o disposto a discrezione dell'utente, alla quale viene offerta la possibilità di salvare le varie disposizioni da utilizzare a seconda del lavoro da fare.

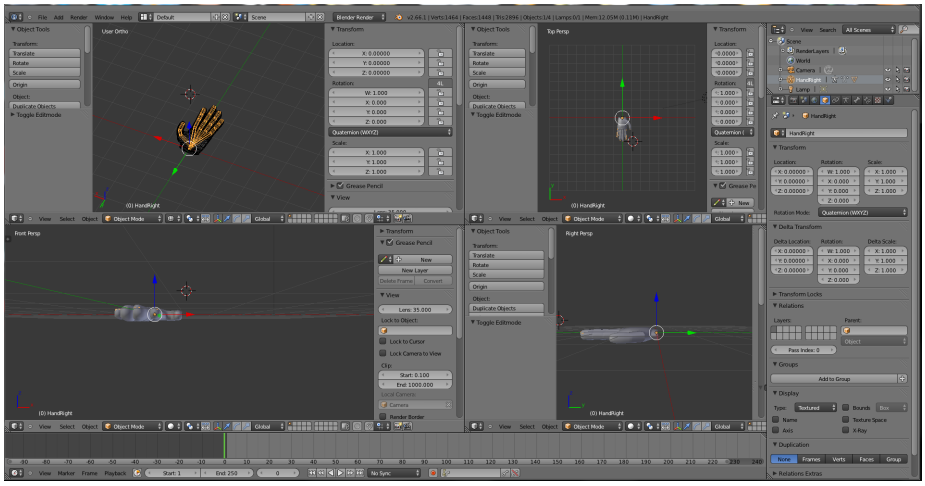

Figura 2.13: *GUI* di *Blender* suddivisa in 4 viste, permette una più completa osservazione del modello in lavorazione

2.5.2 Servizi Offerti

Come accennato in precedenza, *Blender* si offre agli utenti non solo come un programma di creazione e modellazione di oggetti tridimensionali, ma permette anche l'animazione di questi modelli e la creazione di veri e propri videogiochi, attraverso l'uso di script per gestire le interazioni.
Alcune delle funzioni offerte da *Blender* sono:

- Supporto per un vasto assortimento di primitive geometriche come *mesh* poligonali, *curve di Bézier*, *metaball*, e font vettoriali;
- Importazione ed esportazione di modelli per varie applicazioni di modellazione come *3D Studio Max* e *Wings 3D*;
- Strutture per la gestione di animazioni come la cinematica inversa, l'uso delle armature o scheletri, i vincoli utilizzati per rimediare all'impossibilità di unire due armature tra loro, la gestione dei *keyframe*, le animazioni non lineari (anche se non raggiunge ancora il livello dei software concorrenti) e il calcolo pesato dei vertici;
- Il *Blender Game Engine*, che permette l'aggiunta di caratteristiche interattive al modello (come la collisione con gli ostacoli e la programmazione della logica), la creazione di programmi indipendenti, e applicazioni real-time tramite scripting in linguaggio *python*.

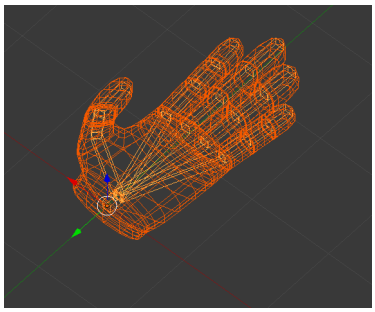

Figura 2.14: Vista del modello di una mano in con modalità di visualizzazione *wireframe* ed in *Object Mode*

2.6 GestIT

L'idea alla base dei Framework *GestIT* [13] è quella di dare la possibilità agli sviluppatori, che utilizzano device con tecnologia multitouch o comunque in grado tracciare parti del corpo, di creare delle *gesture* tramite l'utilizzo di una grammatica context-free. *GestIT* è quindi una libreria che permette di poter definire e gestire delle *gesture* in maniera compositiva e dichiarativa, consentendo sia una definizione della *gesture* ad alto livello, sia la possibilità di poterle definire come un insieme di gesture di più piccole dimensioni (le *subgesture*), ognuna con il proprio gestore. La libreria è stata sviluppata in modo da risultare astratta rispetto ad ogni piattaforma, e quindi permettere di poter descrivere delle gesture che possano essere riconosciute da più dispositivi, come il *touch-screens* per l'interazione *multi-touch*, oppure con la *Microsoft Kinect* per le cosiddette *full-body gesture*.

2.6.1 Organizzazione

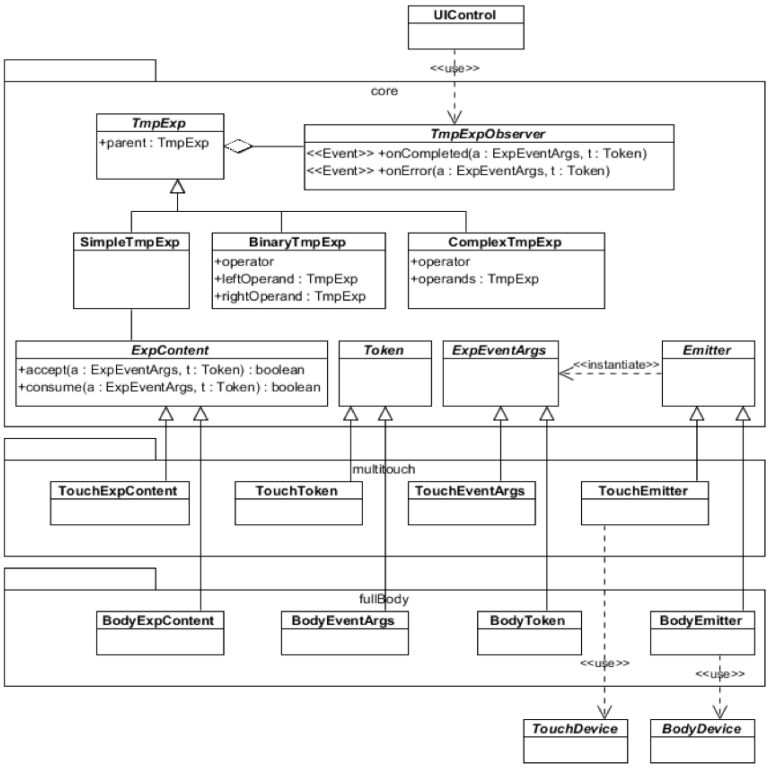

Figura 2.15: L'insieme di classi della libreria *GestIT*

La libreria ha un nucleo indipendente dall'effettivo supporto per rilevamento delle *gesture*, oltre a una serie di estensioni, che supporta gli attuali dispositivi più diffusi come *iOS* devices (multitouch package) e *Microsoft Kinect* (fullbody package). Il nucleo della libreria contiene le classi che provvedono a definire le *gesture* tramite delle espressioni, dei gesti atomici di base che possono essere considerati discreti (i cosiddetti ground terms) e gesti composti creati tramite un insieme di operatori di composizione. La classe astratta TmpExp rappresenta una generica espressione.

La classe *SimpleTmpExp* rappresenta un ground term, ed estende la classe *TmpExp*. Le loro effettive funzionalità, nonché i loro possibili stati, sono definiti tramite dei *delegate object* collegati a un'istanza di *SimpleTmpExp*, e che ovviamente dipendono anche dal dispositivo a cui si riferiscono.
La libreria contiene un interfaccia astratta *ExpContent* che definisce il protocollo per le generic delegate. Questo consiste di due metodi d'istanza:

- *accept*, che riceve lo stato della *gesture* attualmente riconosciuta (rappresentata dalla classe *ExpEventArgs*) e il Token della rete di Petri (che contiene le informazioni sullo stato della prossima *gesture* che si prevede di rilevare). Il delegate è implementato in modo da restituire un valore booleano che indica se è stato riconosciuto o meno un cambiamento nelle funzionalità, in base al valore dei parametri passati.

- *consume*, che consente al programmatore di poter specificare quanti e quali *gesture* gestire durante la fase di riconoscimento delle stesse, dato che non possibile

mantenere l'intera sequenza dei valori a causa della grande quantità di dati che richiede.

Nella classe *TmpExp* troviamo due sottoclassi, *BinaryTmpExp* e *ComplexTmpExp*, che danno la possibilità di combinare *subgesture* per formare *gesture* più complesse. La prima sottoclasse implementa tutte le espressioni con operatori binari (sequence, parallel, choice e disabling). Ovviamente un'istanza di questa classe si comporta in maniera diversa, a seconda delle proprietà dell'operatore, nonché degli operandi che stanno alla sua sinistra e alla sua destra, e che fanno parte della classe *TmpExp*; La seconda sottoclasse invece si occupa di implementare per l'unico operatore ennario, ovvero l'*order independence* che contiene una lista di operandi, anch'essi appartenenti alla classe *TmpExp*, mentre l'operatore *iterative* è rappresentato in *TmpExp* da un flag di tipo booleano. Ricapitolando gli operatori implementati sono:

Iterative (*) - riconoscimento della gesture un numero indefinito di volte;

Sequence (») - permette la creazione di una sequenza di termini base, che deve essere eseguita in ordine, da sinistra a destra

Parallel (||) - stabilisce il rilevamento contemporaneo di due o più gesture;

Choice ([ ]) - consente il riconoscimento di una sola gesture tra quelle collegate;

Disabling ([>) – ferma la rilevazione di un'altra gesture, utilizzabile per fermare un ciclo iterativo;

Order Independence (| = |) – permette di collegare sub-gesture ed eseguirle in maniera casuale, ma costringe ad eseguire tutta la sequenza una volta iniziata prima di poter eseguire un'altra gesture.

La definizione di una *gesture* è rappresentata da un *TmpExp tree*, dove tutte le foglie sono un'istanza di *SimpleTmpExp*, mentre i nodi appartengono alle classi *BinaryTmpExp* o *ComplexTmpExp*. A runtime, l'albero è gestito dal dispositivo dipendente di implementazione della classe Emitter, che ha il compito di ascoltare i dati inviati dal device, e di inviarle a sua volta alle foglie che in quel momento contengono un Token; per ognuno di questi, Emitter invoca il metodo *accept*, se questo restituisce il valore *true*, allora l'emettitore invoca anche il metodo *consume*. Dopodiché se *SimpleTmpExp* rileva il riconoscimento di una delle espressioni definite tramite esso, e in accordo con la semantica di Petri Net, allora provvede a spostare il Token, propagandando la modifica per tutto l'albero, e procedendo con il riconoscimento della *gesture*.
Tuttavia, può capitare che il device invii dei dati che non devono essere accettati da nessuna foglia, in questo caso si deve disattivare la *gesture recognition*, e lo sviluppatore ha la possibilità di definire come l'interfaccia debba reagire in caso di interruzione. La libreria offre anche la possibilità di associare un handler non solo quando viene riconosciuta una *gesture*, ma anche quando questa operazione fallisce (e questo è possibile perché anche quando il riconoscimento fallisce, questo viene propagandato ai livelli superiori del tree).

Le librerie aggiunte a *GestIT*, e che gli permettono di poter operare con il *Leap Motion* sono due:

ClonableLeapFrame - definisce un tipo di oggetto Frame, a cui si aggiungono delle funzione per la clonazione dei dati in essi contenuti. Questa libreria presenta cinque classi: *ClonableFrame, ClonableHand, ClonablePointable*, e *FakeId*.

LeapDriver - che invece si occupa di implementare un modello per elaborazione dei dati provenienti direttamente dal *leap*. È costituita da quattro classi (*ClonableHandCleaner, ClonablePointableCleaner, LeapEventArgs e LeapSensor*) e da un'interfaccia, *IStateCleaner(T,U)*.

Capitolo 3

H_2Mo_4 - Two Hands for Moving

La sigla H_2Mo_4 vuole sottolineare all'utente come l'applicazione sia stata pensata e progettata per essere utilizzata tramite interazione diretta con le mani, permettendo all'utente di poter selezionare degli oggetti presenti nella scena, muoversi nell'ambiente e accedere a delle informazioni aggiuntive, infatti H_2Mo_4 sta per *two hands for moving*, letteralmente traducibile come due mani per muoversi; ma con H_2Mo_4 si vuole anche indicare il legame del sistema con il mondo della chimica, infatti l'utente durante la sua esperienza potrà visualizzare una rappresentazione dell'atomo di diversi elementi della tavola periodica corredato, per ora, dai suoi elettroni e in futuro anche dai suoi neutroni e protoni (quindi il suo nucleo), nonché creare dei legami tra degli atomi per formare delle molecole.

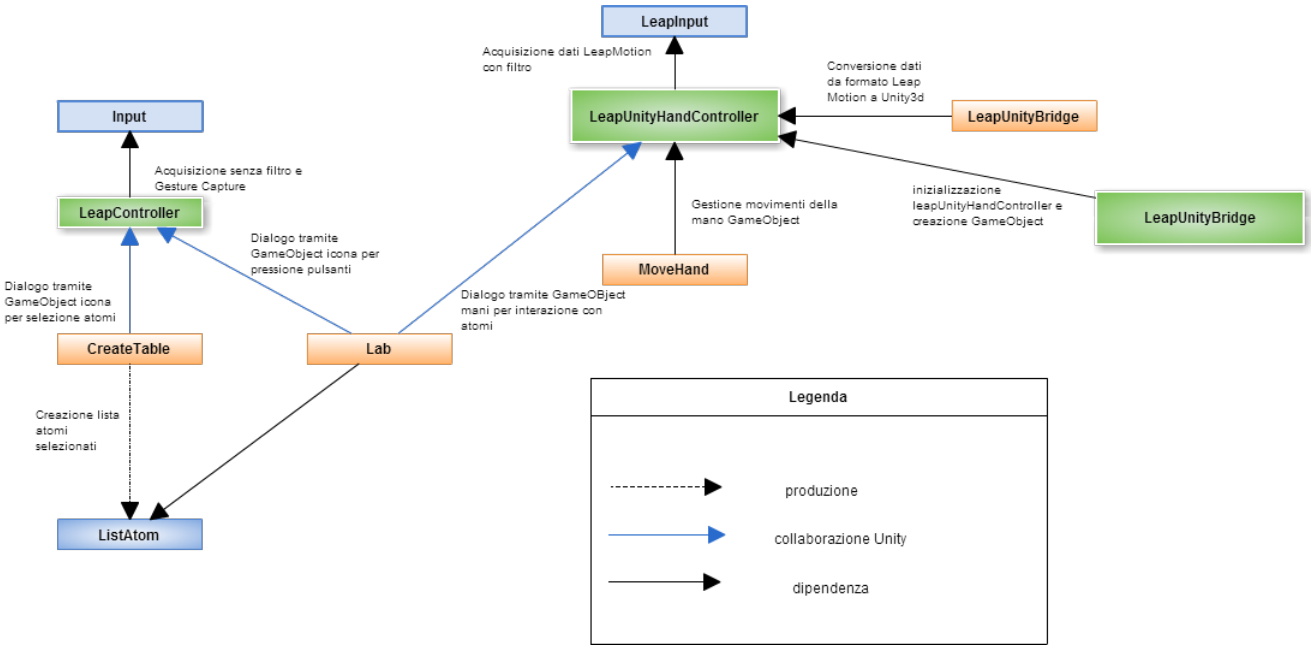

Figura 3.1: Diagramma delle principali classi che compongono l'applicazione

Le principali classi del programma sviluppato sono:

CreateTable - si occupa di disegnare sulla gui i tasti associati ad ogni elemento della tavola periodica, nonché di mantenere una lista degli atomi scelti dall'utente,

e riportare sulla sinistra dello schermo quali sono gli atomi che sono stati già selezionati;

Lab - ha il compito di stampare il laboratorio e di creare nella scena tutti gli atomi precedentemente selezionati; provvede anche a mantenere una lista dei legami creati, e di gestire tutte le opzioni implementate;

LeapUnityBridge - si occupa dell'inizializzazione della classe LeapUnityController, impone il numero di mani gestibili e provvede a creare, nella schermata, le mani rilevate;

LeapUnityController - aggiorna la posizione del GameObject che rappresenta la mano a seconda dei dati ricevuti dal *leap motion*, provvede a far corrispondere le dita rilevate con quelle della mano virtuale.

LeapInput - dialoga con il *leap* e genera un evento quando viene rilevato, aggiornato o perso un dito o una mano.

LeapController - provvede a stampare il puntatore utilizzato dall'utente;

CollisionFinger - gestisce la selezione degli atomi tramite le dita della mano virtuale, nonché la creazione di un legame tra due atomi.

Atom - riporta tutte le informazioni generali di un elemento;

Electron - crea gli elettroni di un atomo e i suoi elettroni di valenza.

3.1 Design dell'interazione

Nella tesi in oggetto sono stati studiati diversi modi per permettere all'utente di interagire con il sistema, dalla rappresentazione di un'icona corrispondente a un dito dell'utente, e che viene usato in maniera simile al puntatore di un mouse per puntare ed evidenziare degli oggetti della scena, alla creazione nello scenario di una o due mani virtuali (a seconda di quante mani il *leap motion* riconosca) che rispondono ai movimenti eseguiti dall'utente, e quindi la possibilità di poter eseguire delle *gesture* attraverso cui è possibile interagire con il sistema.

Puntatore - L'idea che sta alla base del puntatore, come già detto, è quello di permettere all'utente di poter evidenziare degli elementi presenti sulla scena, in maniera simile a come avviene con il puntatore del mouse, quando questo viene passato sugli degli oggetti con cui l'utente può interagire.
Una volta che l'utente ha evidenziato l'elemento scelto, potrà selezionarlo tramite una particolare *gesture* definita attraverso la libreria *GestIT*, e di cui i dettagli dell'implementazione verranno descritti in maniera più dettagliata nella sezione dedicata alle *gesture*, per ora basti sapere che questa *gesture* sfrutta le tre dimensioni del *leap*, e consiste sostanzialmente nello spostare in avanti lo stesso dito o *tool* che comanda l'icona, in maniera simile a come avviene quando si clicca un tasto in uno schermo *touch-screen*.
In origine l'idea era quella di permettere all'utente di poter spostare e controllare direttamente il puntatore del mouse tramite una delle dita rilevate dal *leap motion*.

Tuttavia il sistema di controllo di *Unity3D* impedisce di poter modificare le coordinate dell'icona del mouse. Per questo motivo, si è deciso di creare direttamente un'icona a partire dal dito più lungo tra quelli rilevati dal dispositivo. L'operazione di creazione dell'icona viene eseguita nella classe *LeapConroller*, e dal punto di vista grafico consiste nella rappresentazione di una texture in 2D; la texture viene disegnata sulla GUI (*Graphical User Interface*) dell'applicazione attraverso il metodo *OnGUI* di *Unity3D*; tuttavia prima di creare e visualizzare il puntatore, si provvede ad effettuare due controlli:

1. Si controlla se lo scenario in questione prevede la visualizzazione dell'icona.
2. Se lo prevede, allora si contano il numero delle dita rilevate dal *leap motion*, e se questo è superiore a 0, viene richiamata la funzione *returnFingerLength* (a cui si passa la lista delle dita rilevate), e che restituisce il dito più lungo tra quelli tracciati dal dispositivo, sfruttando l'attributo *frontmost* della classe *FingerList*.

Il dito ottenuto viene poi utilizzato per determinare la posizione dell'icona nello scenario, e anche in questo caso si sfrutta una funzione della libreria del *leap*, ovvero il metodo *Intersect* della classe *Screen*, che prende come parametro un oggetto di tipo *pointable* e restituisce l'intersezione tra lo schermo e la posizione del *pointable*.

Mani - Hanno come obiettivo principale una più profonda immersione da parte dell'utente nell'ambiente costruito, simulando i movimenti delle mani rilevate dal *leap motion*. Vengono usate nella scena successiva alla selezione degli atomi e sono il principale strumento d'interazione diretta con l'ambiente: permettono la selezione degli atomi attraverso il tocco con le dita. La selezione evidenzierà l'atomo soggetto dell'azione, che cambierà colore, in modo da poter eseguire su di esso varie operazioni come lo zoom, la visualizzazione delle informazioni o la possibilità di creare un legame con un altro atomo attraverso una seconda selezione.
Possono essere presenti nella scena solo due mani virtuali, questo per impedire che errori dovuti a rilevazioni fasulle da parte del *leap motion* interferiscano durante l'interazione dell'utente. Per evitare queste rilevazioni falsate, è stato applicato un filtro nella classe addetta alla rilevazione degli input inviati dal *leap motion*, chiamata *LeapInput*, con lo scopo di aumentare la stabilità delle rilevazioni. Questo filtro fa uso di un array circolare di grandezza *MaxFrameList*, questo array verrà riempito con i frame che il *leap motion* rileva e, una volta raggiunta la sua capacità massima, i frame contenuti al suo interno verranno confrontati con l'ultimo frame acquisito, che a sua volta verrà inserito nell'array, andando a sostituire il frame più vecchio. Il sistema del filtro agisce in due modi distinti a seconda del caso in cui ci troviamo: L'implementazione del filtro agisce nelle funzioni addette alla rivelazione di eventi come descritto di seguito:

DispatchFoundEvents - Il filtro controlla che le mani e le dita presenti nel nuovo frame ricevuto dal *leap motion* siano presenti in ogni frame contenuto nell'array circolare del filtro: in caso positivo l'oggetto verrà considerato dal sistema come valido e inserito in un'altra lista, (*listHandConvalidate* o *listFingerConvalidate* a seconda che sia stata rilevata una mano o un dito, in caso di non soddisfazione del requisito l'oggetto rilevato non verrà considerato;

DispatchLostEvents - La verifica dell'uscita dal campo visivo del *leap motion* di una mano o dito dell'utente, avviene tramite il controllo della validità degli elementi contenuti nelle *ArrayList listHandConvalidate* e *listFingerConvalidate* riempite in precedenza in *DispatchFoundEvents*. Questa operazione consiste nel valutare ogni mano o dito presente nel frame appena ricevuto dal device e confrontarli con il contenuto delle *ArrayList*; quando un oggetto confrontato col nuovo frame non è presente in quest'ultimo, questo verrà cancellato dalla lista di appartenenza.

Un altro problema rilevato nella creazione della mano è legata al fatto che il *leap* non è capace di distinguere il tipo di dito o mano rilevato, infatti non vi sono differenze tra le varie dita (es. è impossibile distinguere normalmente un pollice da un mignolo), così come è impossibile distinguere la mano destra dalla mano sinistra. Per ovviare al problema delle mani si ricorre al controllo del parametro *Hand.PalmNormal.x*, che indica con un range da (1) a (-1) la direzione del palmo della mano. L'ipotesi è che una persona normalmente non tiene le mani in modo perfettamente orizzontale, ma la mano sinistra e la mano destra avranno una propria naturale inclinazione. Questa caratteristica viene sfruttata per la distinzione, assumendo una posizione del *leap motion* frontale rispetto all'utente. Una volta acquisita una mano questa potrà assumere l'inclinazione desiderata senza causare errori di riconoscimento, tenendo conto delle caratteristiche del sensore.
Per il riconoscimento delle dita della mano, e quindi l'associazione di queste con le dita del modello, ci basiamo sulle dita appartenenti alle mani visualizzate e non su ogni dito rilevato dal *leap motion*.
Per la risoluzione del problema del riconoscimento delle dita verrà controllata la posizione lungo l'asse x occupata dal dito rilevato, tramite il parametro *Finger.TipPosition.x*. Quest'informazione, confrontata con tutte le altre posizioni delle dita interessate, indicherà la sua posizione relativa rispetto alla mano di appartenenza. Verrà quindi creato un'array per ogni mano, nella quale saranno inserite le dita associate alla mano in questione. Solo quando questo array sarà pieno, ovvero tutte le dita della mano saranno inserite, verrà eseguito l'algoritmo di ordinamento secondo la posizione del dito sull'asse x ed in funzione della mano di appartenenza. Si avrà un ordinamento crescente per la mano destra ed un ordinamento decrescente per la mano sinistra. Se un dito uscirà dalla visuale rilevata dal *leap*, quest'ultimo verrà eliminato dall'array, ma non verrà eseguito alcun ordinamento. Si è scelto questo approccio per fare in modo che, nel caso in cui un dito venga eliminato dall'array, si eviti un cambiamento di posizione degli elementi contenuti nell'array ed una conseguente incoerenza tra la corrispondenza delle dita dell'utente e le dita del modello. Viene così implementato un metodo per riconoscere quale dito viene rilevato, che rende possibile un più accurato controllo del modello della mano.
Il metodo descritto è influenzato da due fattori che limitano la sua efficacia. Il primo è la necessità di riempire completamente l'array delle dita della mano per il corretto funzionamento dell'algoritmo e quindi il riconoscimento delle dita. Per questo problema allo stato attuale e con le funzioni a nostra disposizione è impossibile porre rimedio, occorrerebbe una modifica a basso livello delle librerie proprietarie.
Il secondo fattore limita le rotazioni sull'asse y della mano da parte dell'utente. Un'eccessiva rotazione renderebbe l'utilizzo del parametro *Finger.TipPosition.x* inutile, in quanto le dita risulterebbero praticamente tutte sullo stesso punto dell'asse

x. Una possibile soluzione si troverebbe tenendo conto, oltre al valore della posizione sull'asse x, anche la direzione della mano lungo l'asse x e la posizione delle dita nell'asse z.
Infine per un movimento realistico delle dita del modello, la classe *MoveHand* accederà agli array contenenti gli oggetti *Finger* delle due mani e modificherà l'armatura del modello a seconda del parametro *Finger.Direction.y*, per la direzione del dito sull'asse y, e *Finger.Hand.Direction.y*, per impedire che tenendo le dita aperte ed inclinando la mano queste si chiudano.
Inizialmente per il *GameObject* della mano veniva utilizzato un modello di nostra creazione, ma non rispettava i normali canoni di bellezza, ed è stato optato un cambio stilistico ricercando tra i vari forum di *Blender* un modello più realistico.

Gesture - Come già visto nel capitolo 2, attualmente il *leap motion* permette di riconoscere solo quattro *gesture*, ovvero il *circle*, lo *swipe*, il *key tap* e lo *screen tap*. Tuttavia, per come l'applicazione è stata pensata e progettata, queste *gesture* risultano essere inadatte, procurando dei veri e propri problemi nell'interazione, come nel caso dello *screen tap* e del *key tap*, che se applicate al puntatore che l'utente utilizza per evidenziare gli oggetti della scena, non risultano essere molto efficienti dato che non permettono un'efficace selezione degli elementi evidenziati. Ciò è dovuto al fatto che le due *gesture* prevedono un movimento verso il basso del dito, movimento che influenza anche la posizione dell'icona e che possono portare l'utente a selezionare degli oggetti diversi da quello voluto.
Per superare questi ostacoli, e permettere all'utente un'interazione più naturale, si è deciso di definire delle nuove *gesture* attraverso l'integrazione nella tesi di una versione aggiornata della libreria *GestIT*. Questa versione presenta delle modifiche per adattarsi al *leap motion*, è ancora in fase di sviluppo, e quindi per l'applicazione in oggetto è stata utilizzata una versione non definitiva.
Le nuove *gesture* possono essere divise in due gruppi, quelle che lavorano sul frame attuale, e quelle che invece devono operare su una lista degli ultimi frame registrati. Il primo gruppo di *gesture* sono molto semplici da eseguire e implementare, si eseguono tramite un solo dito, e controllano se il dito dell'utente si trova in una determinata posizione rispetto al *leap motion*; il secondo gruppo invece controllano se la *gesture* che implementano, si verifica negli ultimi frame inviati dal dispositivo.
Le nuove sequenze di movimenti definiti tramite *GestIT* sono:

movefinger_goup - permette all'utente di spostare la visuale verso l'alto. Viene attivata quando l'utente pone il dito rilevato oltre una certa soglia rispetto all'asse Y, ovvero quando la posizione del dito, sul piano verticale, si trova oltre la posizione +350 dall'origine degli assi;

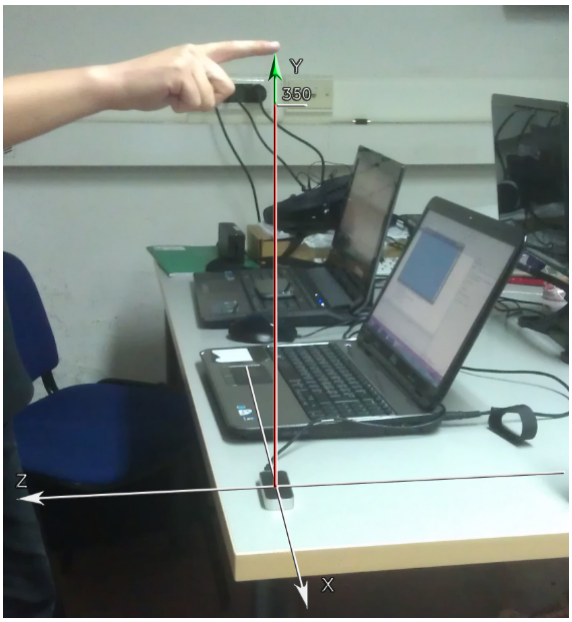

Figura 3.2: *movefinger_goup*

movefinger_godown - con questa *gesture* l'utente sposta la visuale verso il basso, e viene rilevata quando il dito dell'utente si avvicina al centro degli assi, a una posizione inferiore a +50 sull'asse Y calcolato dal *leap*;

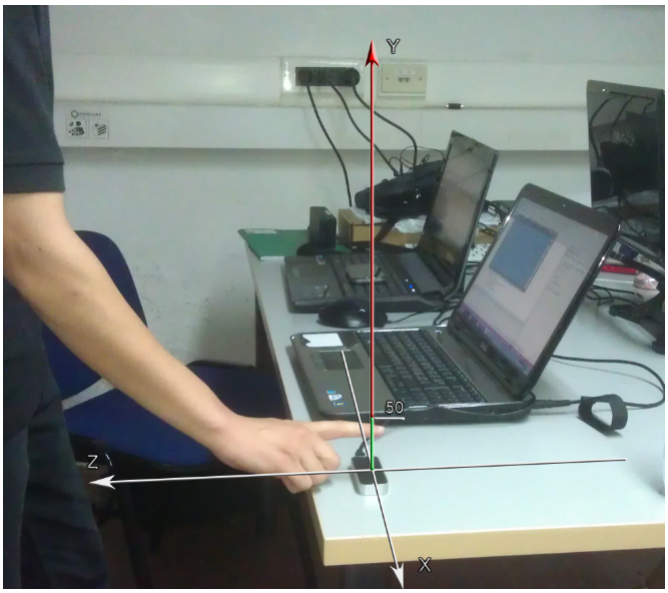

Figura 3.3: *movefinger_godown*

movefinger_front - è stata definita per permettere all'utente di spostare la visuale in avanti. Si esegue ponendo il dito a una distanza inferiore a -100 lungo l'asse Z rilevato dal device.

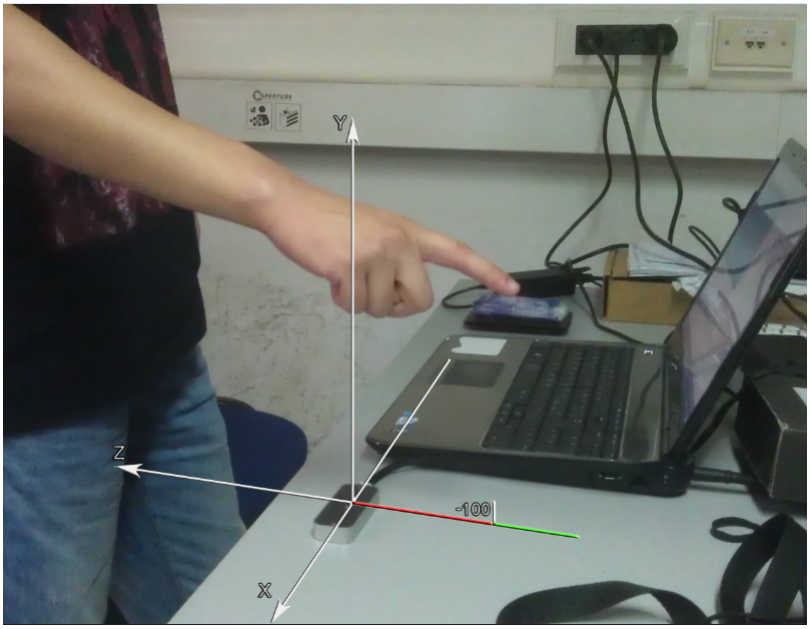

Figura 3.4: *movefinger_front*

movefinger_behind - al contrario della *gesture* precedente, *movefinger_behind* è stata implementata per spostare la camera all'indietro rispetto alla sua posizione attuale. Si ottiene ponendo il dito a una distanza non inferiore a +170 dell'asse Z.

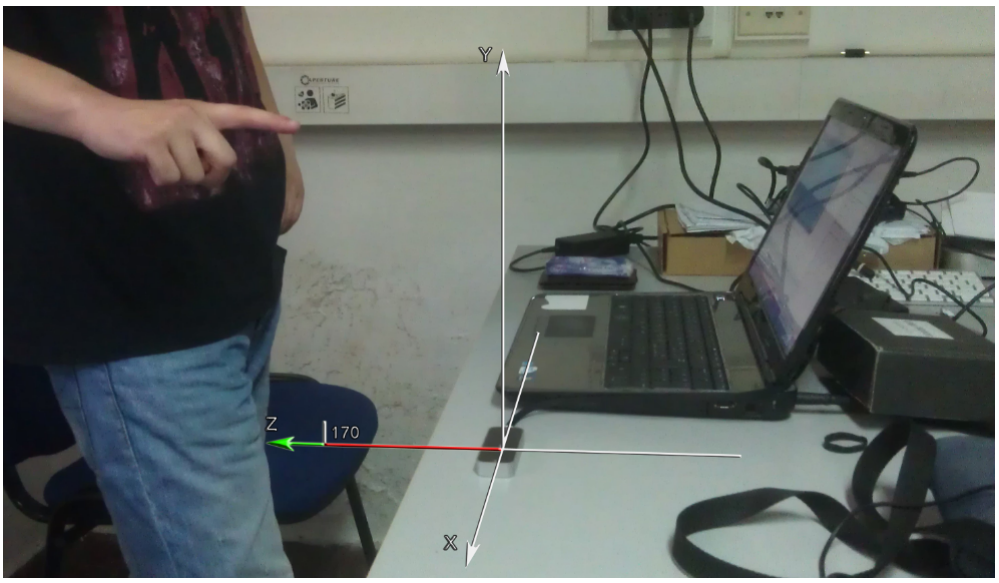

Figura 3.5: *movefinger_behind*

movefinger_right - sposta la visuale dell'applicazione a destra, viene attivata se l'utente pone il dito alla destra del *leap*, a una posizione superiore a +180 rispetto all'asse X.

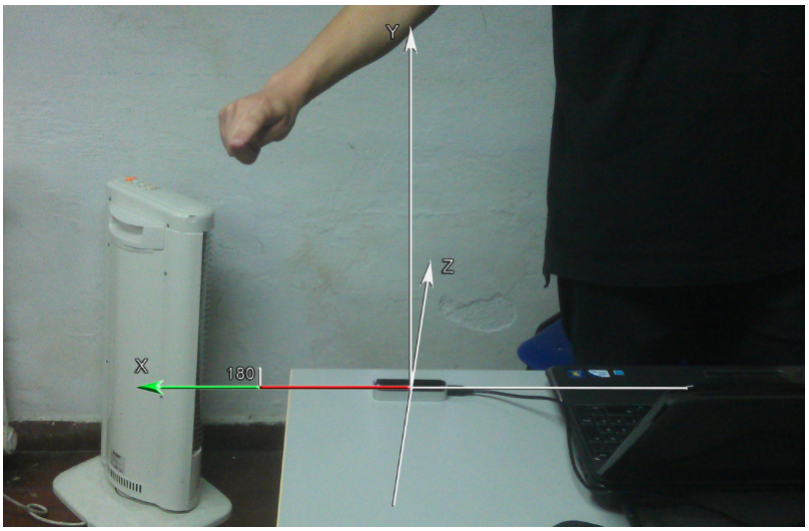

Figura 3.6: *movefinger_right*

movefinger_left - permette all'utente di spostare la visuale a sinistra, viene rilevata quando l'utente pone il dito alla sinistra del *leap*, a una posizione inferiore a -180 rispetto all'asse X.

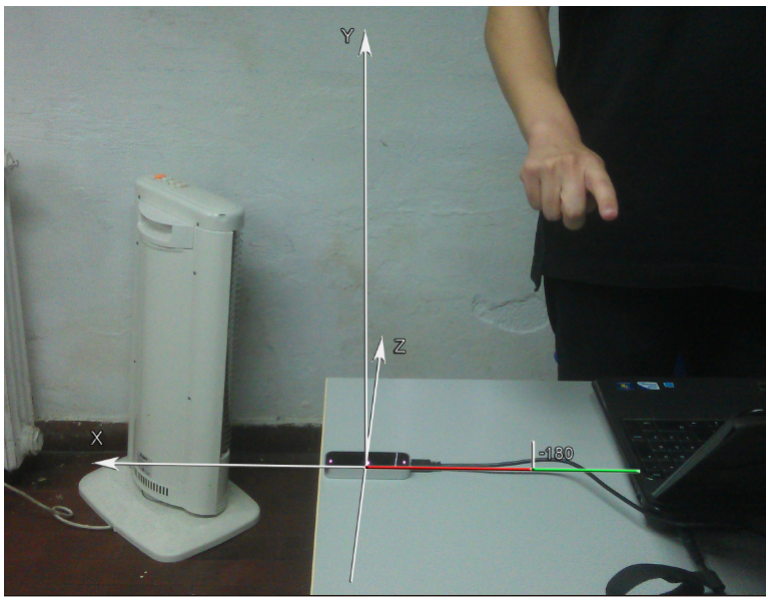

Figura 3.7: *movefinger_left*

movefinger_gofront - questa *gesture* è stata implementata per consentire all'utente di selezionare gli oggetti evidenziati. Consiste sostanzialmente nello spostare il dito in avanti, verso il device (praticamente lungo l'asse Z), con un'interazione simile a quella che si ha con uno schermo posto di fronte all'utente. Il riconoscimento di questa *gesture* si basa nel controllare se negli ultimi venti frame, il dito più lungo si sia progressivamente spostato, per una certa distanza, lungo il piano di Z.

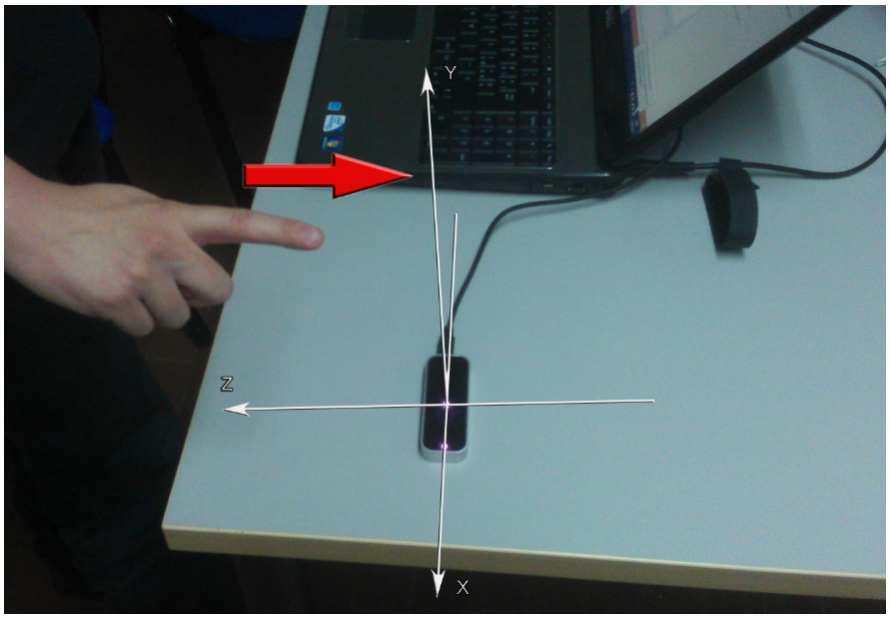

Figura 3.8: *movefinger_gofront*

movefinger_pinchOpen - questa *gesture*, come si può comprendere dal nome, implementa il pinch open, permettendo all'utente di poter ingrandire la visuale in maniera molto più rapida rispetto al semplice movimento di mandarla avanti. Il pinch open si ottiene quando si pongono due dita abbastanza vicine, e poi progressivamente le si muovono in senso opposto, allontanandole. Teoricamente questa sequenza di movimenti può essere eseguita con due dita della stessa mano, tuttavia per la tesi in oggetto, si è preferito consentire l'utilizzo di questa *gesture* solo attraverso l'uso di due mani. Il pinch open viene rilevato dal sistema quando nella lista dei frame più recenti si verifica che sono presenti due dita, e che quello più a sinistra diminuisce progressivamente la sua posizione rispetto all'asse X, mentre quello più a destra, di contro, aumenta il valore delle coordinate rispetto sempre all'asse orizzontale.

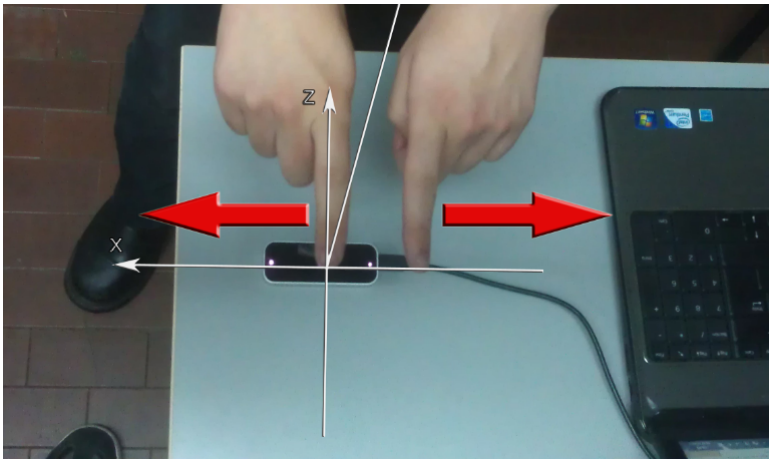

Figura 3.9: *movefinger_pinchOpen*

movefinger_pinchClose - il pinch close, al contrario del pinch open, consente all'utente di ridurre la visuale, sul piano Z, in maniera più rapida rispetto alla *gesture movefinger_behind*. Per eseguire il pinch close, l'utente deve posizionare due dita a una certa distanza l'una dall'altra, e quindi avvicinarle sempre più in base a quanto vuole allontanare la visuale della schermata visualizzata. L'implementazione del pinch close è esattamente il contrario del pinch open, si controlla nella lista dei frame più recenti se ci sono due dita, appartenenti a due mani diverse, quindi se la posizione del dito più a sinistra aumenta rispetto all'asse X, e se invece il dito più a destra diminuisce progressivamente la sua posizione lungo lo stesso asse.

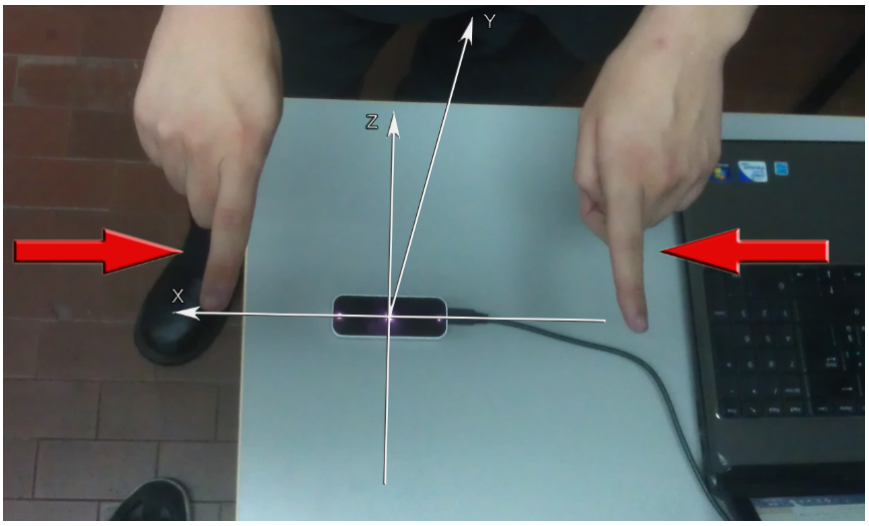

Figura 3.10: *movefinger_pinchClose*

handclose - questa *gesture* consiste nel chiudere la mano per formare un pugno, in modo tale che il *leap* riconosca una sola mano ma non rilevi alcun dito.

3.2 Laboratorio Atomico

Possiamo sostanzialmente suddividere l'applicazione in due parti, una prima parte in cui viene rappresentata la tavola periodica, dalla quale l'utente potrà scegliere gli elementi che vuole visualizzare e con cui vuole formare dei legami, e una seconda parte che invece consisterà in un laboratorio nella quale l'utente potrà interagire con gli elementi selezionati.

Tavola Periodica - Questa schermata dell'applicazione, come già specificato in precedenza, permette all'utente di scegliere quali elementi si vorranno visualizzare nel laboratorio. Nella scena sono presenti il puntatore guidato dall'utente tramite *leap motion*, una lista che riporta gli atomi selezionati dall'utente e due gruppi di pulsanti, un primo gruppo per la rappresentazione degli elementi della tavola periodica, e

un secondo gruppo di pulsanti d'opzione che permettono all'utente di uscire dall'applicazione oppure di procedere alla visualizzazione del laboratorio (previa selezione di almeno un elemento); tutti gli elementi elencati vengono stampati nella GUI dell'applicazione.
Per la rappresentazione del primo gruppo di pulsanti si è scelto di utilizzare due stili differenti, uno per quando il tasto non e selezionato, e uno invece per quando l'utente lo evidenzia.

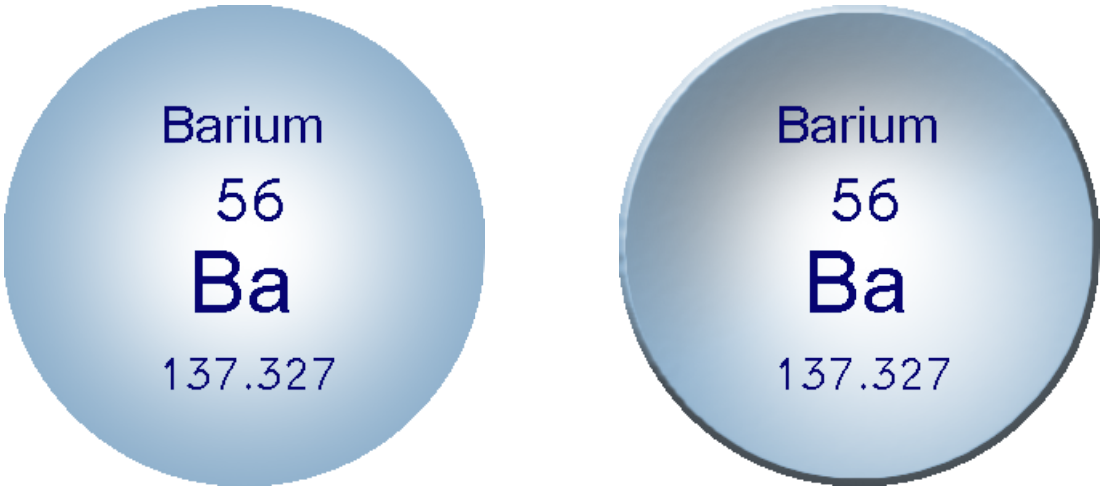

Figura 3.11: A sinistra un esempio di texture applicata al pulsante quando questo non è evidenziato dall'utente, mentre a destra un esempio dell'effetto 3D che viene applicato alla texture quando il puntatore è posato sul corrispondente pulsante

Ad ogni pulsante utilizzato per rappresentare gli elementi della tavola periodica viene applicata una texture in due dimensioni, in cui vengono visualizzate le informazioni basilari dell'elemento, ovvero il nome, il numero atomico, il simbolo e il peso atomico, inoltre lo sfondo di ciascuna texture ha un preciso colore a seconda del tipo di elemento da rappresentare:

- Azzurro - per i principali gruppi metallici;
- Blu - per i metalli di transizione;
- Giallo - per i non metalli;
- Verde - per i metalloidi;

Invece, per quanto riguarda i tasti d'opzione, questi presentano un design differente, e inoltre per la loro selezione non occorre effettuare la *gesture* movefinger_goup, descritta poco fa, ma basta che l'utente vi posi sopra il puntatore per pochi secondi.

Figura 3.12: Il design applicato per i pulsanti d'opzione

Per visualizzare tutte le pagine in cui è suddivisa la tavola periodica, l'utente si avvale delle *gesture movefinger_ right* e *movefinger_ left*, rispettivamente per sfogliare a destra o a sinistra le pagine.

Laboratorio Nel schermata laboratorio vengono rappresentati tutti gli atomi selezionati dall'utente precedentemente, e viene creato un oggetto di tipo GameObject per ognuno di essi. Ogni atomo viene creato insieme a suoi elettroni (anche quelli di valenza), e viene rappresentato come una sfera a cui viene applicata una texture contente il simbolo dell'atomo, in modo da indicare a quale elemento appartiene. Gli altri oggetti che vengono visualizzati nella schermata sono dei pulsanti d'opzione che l'utente si serve per attivare diverse modalità d'interazione e che vengono stampati sulla GUI, e le mani virtuali di cui si è parlato nel paragrafo **3.1**.

Modalità selezionabili: Le modalità che l'utente può attivare variano a seconda che abbia selezionato o meno un atomo. Se non ha selezionato alcun atomo, nella GUI verranno visualizzati i pulsanti per attivare le gesture e per cambiare la visuale. Invece, nel caso in cui l'utente selezioni un atomo, nella schermata appariranno i pulsanti per attivare le gesture, quello per cambiare la visuale e infine quello per visualizzare informazioni aggiuntive sull'elemento dell'atomo selezionato:

- **Active Gesture** - la selezione di questo pulsante permette di attivare la modalità in cui vengono riconosciute le gesture. Per ora le sequenze di movimenti gestite sono il *circle*, che attiva la rotazione dell'atomo intorno al suo asse nel caso in cui sia stato selezionato un atomo, altrimenti la rotazione viene applicata a tutti gli atomi presenti nel laboratorio; il movimento rotatorio viene applicato anche agli elettroni dell'atomo, che ruotano intorno al nucleo (ma questo non vale per gli elettroni di valenza). Le altre due gesture riconosciute sono la rotazione tramite i due pugni, il pinch open e il pinch close, il cui funzionamento e scopo si è già ampiamente parlato nel paragrafo **3.1**.
- **Zoom** - attivando questa modalità, l'utente ha la possibilità di modificare la visuale, facendo avanzare o indietreggiare la telecamera, oppure spostandola ai due lati o ancora facendola alzare o abbassare. Se si attiva questa modalità dopo aver selezionato un atomo, la posizione della telecamera viene modificata, in modo da centrare la visuale rispetto a quest'ultimo.

- **View Information** - questa opzione è disponibile solo se l'utente seleziona un atomo, e permette la visualizzazione di alcune proprietà chimiche e fisiche dell'elemento a cui appartiene, tra cui la temperatura di ebollizione, i suoi stati di ossidazione e il suo raggio atomico. Le informazioni aggiuntive vengono stampate sulla GUI, in una tabella posta alla sinistra dello schermo, in modo tale da interferire il meno possibile con gli oggetti visualizzati.

Per poter attivare queste opzioni, l'utente deve eseguire la *gesture handclose* per poco più di un secondo, indi verrà visualizzato sulla scena un puntatore con la quale potrà attivare o di disattivare le varie opzioni, semplicemente posandosi sopra.

Operazioni eseguibili: Quando l'utente seleziona un atomo, il feedback a livello grafico consiste nel modificare il colore di quest'ultimo, che passa dal grigio al verde. Una volta selezionato un atomo, l'utente può o deselezionarlo (e quindi il colore del GameObject viene riportato al grigio originale), oppure selezionare un altro atomo, con ancora almeno un atomo di valenza a disposizione, e formare così un legame tra i due atomi. Il legame viene rappresentato attraverso un GameObject di forma lineare che collega i due atomi, e che viene creato nel punto medio calcolato in base alle coordinate delle due sfere. Una volta che il legame viene creato, si determina la sua lunghezza e l'angolo di rotazione da applicare, il primo si determina calcolando la differenza lungo l'asse X della posizione tra i due atomi, il secondo invece si ottiene determinando l'arcotangente tra le due sfere coinvolte nel legame. Quindi si provvede a inserirlo in una lista, e quindi a ridurre di un'unità gli elettroni di valenza a disposizione dei due atomi. All'utente viene data anche la possibilità di eliminare un legame creato in precedenza, semplicemente eseguendo la stessa procedura descritta per la sua creazione: si selezionano i due atomi tra cui esiste un legame, si distrugge il GameObject che li collega, e infine si recuperano gli elettroni di valenza degli atomi interessati.

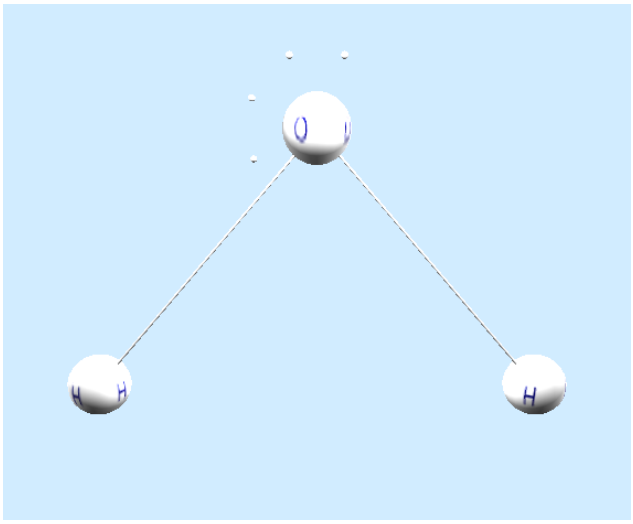

Figura 3.13: Un esempio di come viene rappresentata nel laboratorio virtuale una molecola di H_2O

Capitolo 4

Risultati e Test

4.1 Risultati

I risultati ottenuti dallo studio delle possibilità offerte dal *leap motion* sono andate ben oltre le aspettative iniziali, infatti il device risulta essere molto preciso, estremamente portatile, intuitivo e facile da utilizzare. Tuttavia è possibile riscontrare una certa difficoltà nella creazione di un sistema pensato solo per funzionare con il *leap*, soprattutto per un sistema molto complesso, in quanto comunque il dispositivo soffre ancora di problemi relativi all'occlusione (a seconda di come sono posizionate le dite, il device potrebbe non rilevarle), risulta essere ancora troppo sensibile alla luce (soprattutto quella artificiale) e di default è in grado di riconoscere solo quattro *gestures*. Ad ogni modo si può affermare che il device ben si adatta a un utilizzo complementare con tastiera e mouse.

Per quanto riguarda l'utilizzo della libreria *GestIT* nella tesi, i risultati ottenuti possono considerarsi per metà soddisfacenti, perché se da una parte si è riusciti nell'intento di definire delle nuove *gestures*, tuttavia non è stato possibile integrare la libreria direttamente nell'applicazione. Infatti l'idea originale era quella di utilizzare direttamente la libreria .dll di GestIt con *Unity 3D* attraverso uno Script in *F#* che permettesse la rilevazione dell'esecuzione di una gesture, comunicando poi gli eventi agli script scritti in *C#* di *Unity3D*. Tuttavia la libreria *GestIT* utilizza l'ultima versione della libreria *Microsoft.Net*, cosa che invece non è disponibile nella versione a disposizione di Unity 3D. Per superare questo ostacolo e poter permettere il rilevamento delle gestures implementate, è stato sviluppato un applicazione scritta in *F#*, che si occupa per l'appunto di rilevare quando una di queste gestures viene eseguita, e che dev'essere mandata in esecuzione contemporaneamente al programma in oggetto.

Infine per quanto concerne il laboratorio molecolare in sé, i risultati ottenuti sono stati soddisfacenti, infatti il prototipo risponde in maniera molto rapida ai comandi inviati tramite *leap* e il sistema risulta essere, in generale, molto semplice da utilizzare.

4.2 Test

Per testare le effettive capacità del sistema, appurare se un utilizzo prolungato del *leap* risulti essere faticoso dal punto di vista fisico, nonché dimostrare la naturalezza e la

semplicità delle *gestures* implementate, sono stati organizzati diversi test preliminari, in cui si richiedeva all'utente di:

1. selezionare quattro atomi di idrogeno e due di ossigeno dalla schermata rappresentante la tavola periodica;
2. quindi accedere al laboratorio;
3. attivare il rilevamento delle gestures, ed eseguire nell'ordine circle, pinch open e pinch close;
4. disattivare la modalità gestures, e formare una molecola di acqua unendo due atomi di idrogeno a uno di ossigeno;
5. attivare la modalità zoom e provare a centrare la visuale su un atomo di ossigeno;
6. selezionare un atomo, e attivare quindi l'opzione per visualizzare le informazioni aggiuntive;
7. dopodiché selezionare l'opzione zoom e modificare la posizione della camera in modo da visualizzare tutti gli atomo;
8. deselezionare l'atomo e quindi creare un'altra molecola con gli atomi rimanenti;
9. chiudere l'applicazione.

I risultati dei test hanno messo in evidenza diversi elementi:

- una volta di più come il *leap* sia fortemente sensibile alle fonti di luce, che provoca un certo "flickering"nel riconoscimento degli oggetti rilevati;
- la semplicità delle *gestures* implementate;
- infine, a differenza di quanto previsto, un utilizzo prolungato dell'applicazione non provoca un forte indolenzimento degli arti coinvolti.

Capitolo 5

Sviluppi Futuri

5.1 Sviluppi Futuri dell'Applicazione

Gli sviluppi futuri previsti per il laboratorio molecolare sono diversi: innanzitutto si vuole provvedere all'implementazione di alcuni dei fattori chimici e fisici che entrano in gioco durante la creazione di un legame, ovvero correggere la posizione e l'angolo degli atomi in base al legame creato e agli elementi in gioco, inserire la possibilità di formare legami multipli, implementare l'energia di legame, rendere più verosimile la rappresentazione dei legami, e gestire un maggior numero di atomi nella scena. Inoltre altri sviluppi futuri riguardano:

- implementare tutti gli elementi presenti nella tavola periodica (per ora l'applicazione permette di scegliere solo tra gli atomi di 30 elementi);
- definire più realisticamente la rappresentazione di un atomo, sia per quanto riguarda la sua dimensione che la configurazione elettronica, in base all'elemento a cui è associato;
- inserire la possibilità di poter rimuovere o aggiungere un elettrone ad un atomo, e quindi gestire la ionizzazione di un'entità molecolare neutra;
- permettere all'utente di poter salvare una molecola creata in laboratorio, nonché aggiungere la possibilità di poter caricare delle altre molecole, create dall'utente stesso oppure da altri;
- consentire all'utente, durante la scelta degli atomi da rappresentare, di poter eliminare un atomo precedentemente selezionato.

Alcune di queste features richiedono un maggior grado di interazione con l'utente, perciò un ulteriore elemento da sviluppare sarà l'implementazione e il riconoscimento di nuove gestures più complesse (come ad esempio la chiusura della mano dell'utente oppure il pinch con le dita di una sola mano). Infine integrare il riconoscimento delle gestures direttamente su *Unity 3D*, tramite definizione di una dll, appena verrà rilasciata una versione più aggiornata di quest'ultimo, oppure una versione definitiva della libreria *GestIT* utilizzata.

5.2 Il futuro del Leap Motion

Il *leap motion* ha grandissime potenzialità; in questa tesi abbiamo riscontrato anche dei difetti, come l'elevata sensibilità alle fonti di luce. Tuttavia l'alta qualità del frame rate, la possibilità di poterlo integrare in diversi linguaggi e il suo basso costo rappresentano degli ottimi vantaggi, soprattutto se si tiene in conto che il dispositivo non è stato ancora ultimato definitivamente. Ad ogni modo, il futuro del *leap* dipenderà fondamentalmente da due fattori, *in primis* come la critica, e soprattutto gli utenti finali, lo accoglieranno e in secondo luogo da come l'azienda produttrice continuerà a sostenere il suo progetto, se il device verrà costantemente aggiornato e migliorato e se verrà sempre più incentivato lo sviluppo di applicazioni, in modo da scongiurare il pericolo di mandare nel dimenticatoio un dispositivo innovativo e interessante (anche se con gli ingenti investimenti ricevuti dalle principali realtà economiche nel campo informatico, questo rischio sembra essere scongiurato).

Capitolo 6

Conclusioni

In conclusione si può affermare che i risultati ottenuti sono stati soddisfacenti, anche se non completamente in linea con gli obiettivi fissati all'inizio dei lavori:

- il *leap motion* può tranquillamente integrarsi con i sistemi attuali, ovviamente con delle piccole modifiche a quest'ultime;
- i vantaggi apportati dal dispositivo, a beneficio di un'interazione più naturale, sono maggiori dei suo difetti;
- l'applicazione risponde ai comandi eseguiti dall'utente tramite *leap motion*;
- sono state implementate nuove gestures eseguibili per mezzo del *leap*;
- è possibile interagire con gli oggetti presenti sulla scena.

Rimane comunque il rammarico di non essere riusciti a implementare nel dettaglio anche tutti i fattori chimici e fisici che stanno alla base degli atomi e della formazione delle molecole; ad ogni modo si prevede di implementare queste proprietà prossimamente, con la speranza di poter collaborare con qualche esperto nel campo.
Inoltre un altro aspetto negativo dei risultati ottenuti, riguarda l'impossibilità di poter integrare la libreria *GestIT*, a disposizione, con *Unity3D*, che ha costretto ad implementare il riconoscimento di una *gesture* tramite la simulazione della pressione del tasto di una tastiera.

Ringraziamenti

Innanzitutto un ringraziamento va a Davide, detto anche Lucio, perché anche dopo quattro mesi di rotture, ci saluta ancora. Grazie Davide, un giorno ricambieremo, forse. Ma un ringraziamento speciale va anche al prof. Scateni per avermi accettato nella sua "corte" (e per il cellulare). Però in futuro vi consiglio di alzare i requisiti per accedere all'internato.... altrimenti rischiate di trovarvi un altro coglione come me (mica son tutti come Dagi o Giammy!).
Ovviamente non possono mancare i ringraziamenti a tutti i ragazzi della Batcaverna®: Stefano, Guga, Cino, Mary, Samuel, Checco, Fabio, Silvia, Daniela, Giorgio, Ricky, Francesca, Kocho, Jenny, Elena, Stefano, Sabrina, Simone, Angelica e ecc.... senza di voi la caverna sarebbe ancora un obitorio (comunque, quando la ripuliamo la caverna?).
E dulcis in fundo un ringraziamento particolare ai, cito testualmente, "rimanenti 4 dei fantastici 5" (cherchi, hai una fantasia del cazzo, ma veramente): Giammy sono stati tre anni di divertimento, di insulti e preghiere, te ne sono grato... tuttavia del tuo essere stronzo ne farei volentieri a meno, ah salutami la gastrite quando puoi. Bizio ringrazio anche te (ma anche no), ammiro il tuo interesse per gli anime et manga, però non hai dei buoni gusti musicali, cioè dovresti osare di più, punta alla Digital Hardcore e abbi fede in me (e falla ogni tanto una dieta, daì!). Dagi, grazie a te nessuno mi chiama per nome, grazie a te la gente mi considera un droghino (ah no, questo è colpa mia, pardon), grazie per non avermi abbandonato sul ciglio della strada quando mi serviva un passaggio in macchina (e lo potevi fare, ma non lo hai fatto) e grazie per avermi sempre fatto compagnia in chiesa la domenica mattina; ah preparati, settimana prossima facciamo la settimana da vegani, così poi ci manca solo il fruttarismo e le abbiamo provate tutte. Sara, grazie infinite per l'aiuto che ci hai dato in tutto questo tempo, per le corse fatte insieme, perché mi chiami per nome e per non averci ancora mandato a quel paese..... (sai, potresti chiedere di convertire tutte queste ore passate con noi come lavori socialmente utili, magari ti pagano pure, e inveeeece? [cit.]). Un non grazie ad Alessio per le ore di fallimento passate insieme (pensa alla seconda serie di guru guru per esempio), un bel modo di sprecare minuti preziosi della nostra vita, non grazie di nuovo.
Visto che abbiamo ancora inchiostro da sprecare, vorrei anche ringraziare alcuni colleghi, per esempio il Kappa (hai passato ARE, ormai posso morire felice, però ora passi pure LIP), il Viola (sei inutile: non ci hai mai portato una cassa di birra... quanto fai schifo? Tanto, sei pure un nano, ti dedico una mia famosa citazione), il don Roberto (e studia csmn porco cane, e non giocare a LoL; tra l'altro insieme a Viola e Kappa, formi il trio dei falliti, lo sai vero?), il Vargiu (bello gears of war 4 vè?), il Dessi (naruto fa schifo), il Flavio, il Pasquale (basta, ci sono troppi juventini in questi ringraziamenti, mi avete rotto le palle), il Rio, l'Ortensio (Ken Follett è bravo, però Robert Harris è molto meglio, abbi fede) e il Concas.
Un grazie anche a Carky per le sue paste, a Sella per la sua lungimiranza politica ed

economica (da grande voglio firmare accordi con i latifondisti come lui!), alla Silvia per il suo coraggio (coraggio è un eufemismo, voglio l'autografo del papa, mi raccomando), alla Giulia (perché parlare di fare i fruttivendoli dopo sedici anni, non ha prezzo), a Rossano per la sua paxxia, a Calderoli perchè ogni volta che lo vedo in parlamento ci resto ancora male.... (e penso sempre la stessa storia: "Ma cosa ci fa una grossa penisia [cit.] in parlamento? Ah no! È solo il vice-presidente del senato! Perdonatemi!."), alla Bioware per i suoi bellissimi finali colorati (di merda) e a Jenus (incredibile, appena nomino te o qualche tuo parente vi presentate immediatamente, siete fantastici!).

Riassumendo ringrazio la mia famiglia per avermi mantenuto in tutti questi anni scolastici, gli amici per avermi procurato distrazione nei momenti più stressanti dell'università (che palle), la corsa per avermi mantenuto in forma, il post-black metal e l'avantgarde/progressive metal per avermi insegnato che nella musica non c'è mai limite al peggio (a parte joe squillo versione punk).

Ale

Voglio ringraziare prima di chiunque altra persona i miei Genitori, i quali tre anni fa insistettero per la mia iscrizione a questa facoltà e per tutto questo tempo mi hanno sostenuto moralmente e finanziariamente, senza di loro non sarei arrivato sino a questo punto e non potrei sorbirmi tutti gli stress che la laurea regala, quindi Grazie, con tutto me stesso.
Ringrazio tutti i miei colleghi del corso che hanno passato con me questi tre anni, colleghi con la quale ho passato ore e ore in facoltà per lo studio e lo sviluppo dei progetti per raggiungere il completamento di tutti, o per qualcuno quasi tutti, gli esami, Grazie.
Ringrazio il mio Relatore Davide, senza di lui starei sicuramente ancora vagando per il mondo in cerca di una tesi da fare. Spero di continuare e migliorare questo progetto sotto la tua ala, Grazie.
Ringrazio "La Borsa Rosa", il gruppo di colleghi, ma soprattutto amici che, tra risate e momenti meno felici, mi hanno aiutato a passare al meglio questi tre anni, non potevo trovare compagnia migliore, a loro voglio dire, Ho Perso, e ringraziarli tutti, anche chi ora non c'è più. Grazie.
Ringrazio anche tutti gli abitanti della Batcaverna ed il loro Leader, scusate sono un po introverso, ma sono stato bene con voi, spero mi vogliate sopportare ancora per i prossimi anni, Grazie.
Ai miei vecchi amici, che hanno sopportato le mie frasi "Non posso, devo studiare" o "Ho un esame a breve, mi spiace" tante volte. Amici che nonostante le mie innumerevoli sparizioni settimanali mi sono stati comunque vicini, Grazie.
E per ultimo voglio ringraziare una persona che ho conosciuto relativamente da poco, dall'inizio di quest'ultimo anno accademico per l'esattezza, ma che ormai sembra una vita. Una persona che mi è stata vicina in ogni momento, che mi ha tirato su quando non ce la facevo e mi ha supportato, sempre, e che è diventata sempre più speciale. Quindi, anche se probabilmente mi dirai "non dovevi ringraziarmi" lo farò lo stesso, Grazie Je.
Un grazie generale anche a tutte le persone che ho incontrato nel mio cammino, Grazie a tutti e AuguriK.

M.

Appendice A

Appendice

L'applicazione è stata sviluppata utilizzando, per il motore grafico del laboratorio atomico, *Unity3D* (alla versione 4.1.5f1), per la realizzazione del programma che rileva le *gesture* è stata utilizzata la versione ultimate di Visual Studio 2012, e infine Blender (versione 2.66) per la modellazione delle mani virtuali utilizzate nel laboratorio. Inoltre sono stati messi a disposizione del progetto, due modelli del *Leap Motion*, con il relativo kit di sviluppo aggiornato alla versione *0.7.9* (che è anche l'ultima versione rilasciata ufficialmente). Per la realizzazione del sistema in oggetto, sono state prodotte 2.860 righe di codice in C# e 283 righe di codice in F#, per un totale di 3.143 righe, e ha richiesto, all'incirca, 300 ore-uomo per la progettazione e implementazione di tutte le sue parti.

Bibliografia

[1] Dix A., Finlay J., Abowd G.D., Beale R.
Interazione uomo-macchina [2004]: McGraw-Hill, NY

[2] Wikipedia, June 2013
www.wikipedia.it

[3] Virtual Molecular Dynamics Laboratory, June 2013
www.polymer.bu.edu/vmdl

[4] Leap Motion, June 2013
www.leapmotion.com

[5] Leap Motion SDK, June 2013
leap_developer_kit/LeapSDK/docs/index.html

[6] About Leap Motion, June 2013
http://pierresemaan.com/category/leap-motion/

[7] Athlete
http://athlete.jpl.nasa.gov/

[8] Unity3D, June 2013
www.unity3d.com/Documentation/Components/index.html

[9] Blender, June 2013
www.blender.org

[10] GestIT, June 2013
www.gestit.codeplex.com

[11] Spano L.D., Cisternino, A., Paternò F., Fenu G.
GestIT: A Declarative and Compositional Framework for Multiplatform Gesture Definition [2013], pp. 187-196

[12] Spano L.D.,
Developing Touchless Interfaces with GestIT [2012], pp. 433-438

[13] Spano L.D., Cisternino A., Paternò F.
A Compositional Model for Gesture Definition [2012], pp. 34-52