ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA

TESI DI LAUREA
in
Reti di Calcolatori T

Tecniche di integrazione e modifica precisa di modelli in tempo reale su sistemi di Mixed Reality

CANDIDATO:

Paolo Caligiana

RELATORE:

Chiar.mo Prof. Ing. Antonio Corradi

CORRELATORI:
Prof. Ing. Alfredo Liverani
Dott. Gian Maria Santi
Dott. Francesco Osti

Anno Accademico 2017/18

Sessione III

Indice

Introduzione

Nel corso degli ultimi anni, si è sentito sempre più spesso parlare di Realtà Virtuale e Realtà Aumentata, due tecnologie diverse ma affini che stanno via via assumendo un ruolo chiave nelle nostre vite. La prima, impiegata maggiormente nell'industria del *gaming,* prevede l'utilizzo di visori che isolano completamente dal mondo esterno in modo da immergere chi li indossa in un ambiente totalmente simulato. La seconda invece, che interessa molteplici settori, dal design all'intrattenimento, dall'addestramento militare ai beni culturali, si serve di svariati dispositivi tra cui smartphone, tablet e speciali occhialini trasparenti per aggiungere elementi digitali e artificiali al mondo reale. Meno conosciuta ma già ampliamente utilizzata è la *Mixed Reality,* che consente, di interagire con oggetti virtuali introdotti nel mondo reale, come fossero veri. Uno degli impieghi più interessanti di questa tecnologia è sicuramente l'ambito medico, in cui si possono sfruttare particolari visori per sovrapporre al corpo dei pazienti proiezioni artificiali fornendo ai medici una visione quasi "a raggi X". In questo modo i chirurgi possono eseguire ad esempio biopsie, laparoscopie o qualunque altra operazione con una maggiore precisione.

Il progetto di tesi che si vuole qui presentare, si serve della *Mixed Reality* e nasce per l'appunto in ambito medico da una collaborazione tra l'Istituto Ortopedico Rizzoli e il Dipartimento di Ingegneria Industriale di Bologna. L'obiettivo preposto è lo studio di tecniche innovative di integrazione e modifica di oggetti in tempo reale e l'utilizzo delle stesse per lo sviluppo di un'applicazione efficiente da utilizzare all'interno dell'infrastruttura ospedaliera.

I primi tre capitoli hanno come obiettivo quello di presentare le principali tematiche trattate e fornire conoscenze di base al fine di comprendere meglio il lavoro svolto. Gli ultimi quattro presentano invece le varie fasi di sviluppo dell'applicazione finale, nominata *"Mixed Reality Tracking Cut Project"* o MRTC, proprio perché deriva dall'unione di 3 progetti: *Mixed Reality Project, Tracking Project e Cut Project*.

In particolare, nel primo capitolo vengono descritte nel dettaglio le tecnologie di Realtà Aumentata, Realtà Virtuale e *Mixed Reality,* fornendone esempi di utilizzo, spiegandone l'importanza e mettendole in fine a confronto nell'ultimo paragrafo. Il secondo capitolo introduce due degli strumenti più utilizzati per lo sviluppo di applicazioni *Mixed Reality,* vale a dire Unity e Microsoft HoloLens. Il terzo similmente al precedente descrive

dispositivi hardware e software utilizzati però per lo più nel caso specifico. Nel quarto capitolo vien presentato nel dettaglio il primo progetto sviluppato, ossia *Cut Project*, con l'obiettivo di effettuare il taglio anche multiplo di oggetti virtuali in tempo reale.
Il quinto introduce il *Tracking Project*, realizzato per associare ad un qualsiasi elemento fisico inquadrato da una telecamera, la corrispondente immagine digitale. Nel sesto capitolo, il *Mixed Reality Project* introduce l'utilizzo del Leap Motion allo scopo di consentire all'utente di vivere appunto un'esperienza di *Mixed Reality*, interagendo con gli oggetti virtuali con le sue stesse mani. Il settimo e ultimo capitolo presenta l'applicazione MRTC, risultato delle diverse fasi precedentemente descritte, nonché prodotto finito da presentare all'Istituto Ortopedico Rizzoli.

Capitolo 1

Virtual, Augmented e Mixed Reality

Il primo capitolo è volto ad introdurre il lettore al concetto di *Mixed Reality* [1] o MR (letteralmente "Realtà Mista"), ovvero il mondo esterno mediato da un dispositivo che vi inserisce oggetti virtuali coi quali è possibile interagire come fossero veri. A tal scopo, questa viene messa a confronto con altre due realtà che hanno preso sempre più piede negli ultimi anni, vale a dire quella Virtuale (dall'inglese *"Virtual Reality*", VR), che rappresenta un ambiente del tutto simulato nel quale è possibile muoversi liberamente sfruttando appositi visori oscurati, guanti e cuffie, e quella Aumentata (dall'inglese "*Augmented Reality*", AR), che rappresenta l'arricchimento della percezione sensoriale umana mediante informazioni che non sarebbero percepibili con i soli 5 sensi.

1.1 La Realtà Virtuale

Ormai da tempo si sente parlare di Realtà Virtuale, una tecnologia ancora alle sue prime fasi di sviluppo che però, nel corso degli ultimi anni, ha iniziato a dare risultati degni di nota. Questo mondo, accessibile tramite appositi visori oscurati VR, nasce dalla combinazione di dispositivi hardware e software che "collaborano" per creare uno spazio virtuale all'interno del quale l'utente possa muoversi liberamente sfruttando particolari guanti o controller (Fig. 1.1). È facile intuire che il principale settore di interesse sia quello del *gaming* (letteralmente "del gioco", quindi ciò che riguarda il mondo dei videogame), ma la Realtà Virtuale affonda le sue radici anche nel turismo, nello sport, nella sanità, sempre al fine di offrire esperienze realistiche simulate (vedi Fig. 1.2).

Figura 1.1: Visore e controller per la Realtà Virtuale.

Figura 1.2: Ambiti di utilizzo della Realtà Virtuale.

1.1 La Realtà Aumentata

Affine ma distinta dalla VR, la Realtà Aumentata [2], anziché utilizzare particolari visori per creare un ambiente del tutto fittizio simulando i cinque sensi, li "aumenta" aggiungendo contenuti visivi, sonori e addirittura olfattivi. Non si ha più quindi a che fare con un mondo del tutto virtuale, ma con la realtà circostante tipicamente arricchita da oggetti virtuali.

Per realizzare esperienze di Realtà Aumentata, vengono utilizzati i più svariati dispositivi, dal semplice smartphone o tablet a visori *head-locked*, caschetti fissati sulla testa e occhiali *see-through*, speciali lenti trasparenti.

Se già la VR stava iniziando ad influenzare le nostre vite, l'AR sta progressivamente prendendo piede in qualsiasi ambito. Molti dei cellulari attualmente in circolazione supportano questa tecnologia, svariate aziende la sfruttano ad esempio per la vendita di mobili, che gli acquirenti possono scegliere comodamente da casa proiettandoli nel proprio salotto (Fig. 1.2). Ancora ampiamente utilizzata nell'industria del *gaming*, tanto che è possibile vedere attraverso il proprio smarphone personaggi tridimensionali proiettati nella realtà circostante (Fig. 1.3).

Attualmente vengono dati in dotazione all'aeronautica militare, caschetti su cui proiettate informazioni necessarie in tempo reale, per permettere ai piloti di rimanere costantemente aggiornati senza distogliere l'attenzione dal loro lavoro [3] (Fig. 1.4).

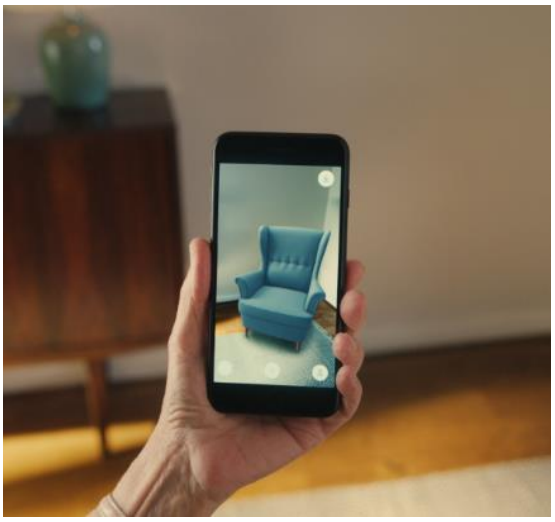

Figura 1.2: AR nell'arredamento.

Figura 1.3: AR nel *gaming*.

Figura 1.4: AR nell'aeronautica militare.

1.2 La Mixed Reality

Meno conosciuta e spesso confusa con l'AR, la *Mixed Reality*, "Realtà Mista", è la terza tipologia, così definita proprio perché "mischia" i mondi di cui si è parlato nelle due sezioni precedenti e ne costituisce il punto di incontro.

Nel 1994 Paul Milgram introduce il concetto di *Virtuality Continuum* [4], una linea che pone ai suoi estremi un ambiente completamente reale e uno del tutto virtuale che si combinano e mescolano fra di loro sempre più man mano che si avvicinano l'uno verso l'altro fino al loro punto di incontro al centro: la *Mixed Reality* (Fig. 1.5).

Quest'ultima dunque produce una realtà del tutto nuova, che non è il mondo reale e nemmeno quello virtuale, dove però questi ultimi diventano un tutt'uno, coesistono e si combinano alla perfezione. L'obiettivo finale è quello di far sì che l'utente immerso in questo ambiente non si renda quasi più conto di ciò che è reale e ciò che è virtuale.
Solitamente i dispositivi che consentono di addentrarsi in questo mondo, sono caschetti *head-locked* dotati di lenti *see-through* con sensori di profondità di vario tipo, simili a quelli utilizzati per la Realtà Aumentata, con la sola differenza che supportano applicazioni MR anziché AR. Tra queste, l'esempio più impressionante è quello della presentazione di una demo del videogame *Minecraft*, sfruttando il dispositivo Microsoft HoloLens (di cui si parlerà nel dettaglio a breve), in cui all'improvviso compare sul tavolo di legno di fronte al presentatore il mondo digitale del gioco, con cui è possibile interagire in tempo reale [5].

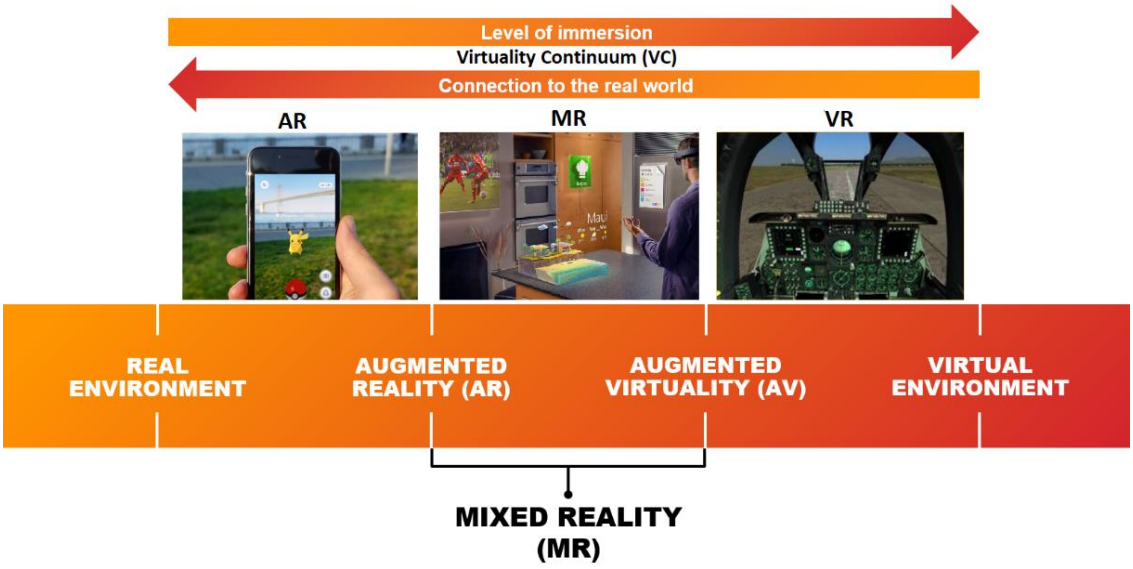

Figura 1.5: Rappresentazione della *Virtuality Continuum* (VC).

1.3 I tre mondi a confronto

Per concludere, tipicamente la Realtà Virtuale proietta l'utente in uno spazio digitale completamente artificiale senza alcuna traccia dell'ambiente esterno; la Realtà Aumentata introduce informazioni aggiuntive e sovrappone oggetti virtuali bidimensionali e tridimensionali nel mondo reale; la Realtà Mista non solo li aggiunge, ma li "ancora" alla realtà, consentendo l'interazione con questi ultimi come se fossero veri [6]. In fase di sviluppo di una qualsiasi applicazione, tutte e tre le tecnologie richiedono di solito software di modellazione 3D o sistemi CAD (strumenti di *Computer Aided Design,* ossia per la progettazione assistita di elementi digitali e reali tridimensionali), per realizzare gli oggetti virtuali da introdurre nei vari ambienti.
Si tratta di realtà molto affini, che spesso si servono di strumenti simili se non i medesimi, impiegate negli stessi settori e il più delle volte con lo stesso obiettivo.
Potrebbe essere ad esempio richiesto di trovare un metodo più efficace per istruire a livello accademico gli studenti di medicina, fornendo un'applicazione che consenta loro di vivere in prima persona un intervento chirurgico simulato. Una prima possibile soluzione più economica seppure meno efficace, potrebbe essere l'impiego della realtà aumentata, che col semplice utilizzo di uno smartphone consentirebbe di vedere nella realtà circostante, i vari step dell'operazione con relativi componenti ossei digitali tridimensionali, senza però potervi interagire. La Realtà Virtuale al contrario, consentirebbe con l'utilizzo di appositi visori oscurati, di interagire in prima persona con il paziente artificiale e i vari oggetti di interesse, ma in un ambiente completamente simulato e utilizzando dei controller per effettuare l'operazione. La *Mixed Reality,* sfruttando un dispositivo simile ma con lenti trasparenti, non oscurate, garantirebbe non solo l'interazione con il soggetto virtuale, ma il tutto proiettato nel mondo reale e facendo tutto con le proprie mani senza usare controller o guanti particolari.
Quello sopra descritto è solo uno dei possibili esempi dell'impiego delle tre diverse tecnologie, ma risulta chiaro che la MR sembra essere la più innovativa e promettente, che sarà in grado magari in futuro di annullare il confine tra reale e virtuale.

Capitolo 2

Tecnologie per *Mixed Reality*

In questo capitolo, vengono descritti due dei principali strumenti attualmente utilizzati per creare applicazioni *Mixed Reality* ed usati in particolare per la realizzazione dell'elaborato di tesi finale, di cui si parlerà ampliamente a partire dal successivo capitolo.

2.1 Microsoft HoloLens

Microsoft HoloLens [7], è il primo computer olografico (Fig. 2.1) ad offrire un'esperienza *Mixed Reality*, consentendo l'interazione con ologrammi di oggetti virtuali proiettati nel mondo reale. Proprio per questo motivo è stato scelto per la realizzazione del nostro progetto. Attualmente sono due le versioni disponibili: *Microsoft HoloLens Development Edition* e *Microsoft HoloLens Commercial Suite*. La prima, disponibile al prezzo di 3299 euro, è pensata per tutti gli sviluppatori che vogliono provare questo innovativo prodotto ed iniziare a sviluppare soluzioni che sfruttino la *Mixed Reality* di Microsoft. La seconda invece, al prezzo di ben 5489 euro, è una soluzione ideata per le aziende che intendono adottare il visore per le loro attività ed include una serie di servizi extra di valore aggiunto specifici per il mondo *enterprise*.

Figura 2.1: caricatore con porta microUsb, dispositivo HoloLens, custodia.

2.1.1 Descrizione generale

Microsoft HoloLens, all'apparenza un caschetto munito di occhiali trasparenti incorporati, altro non è che un computer olografico in grado di proiettare ologrammi di oggetti virtuali consentendo a chi lo indossa di visualizzarli nel mondo reale.
Non si tratta di un semplice visore, in effetti è molto più di questo: dotato di vari sensori di profondità e telecamere, consente di eseguire una scansione dell'ambiente circostante, una vera e propria mappatura della stanza in cui ci si trova, nella quale è possibile collocare e fissare gli oggetti virtuali a proprio piacimento attraverso delle "*gesture*" (gesti fatti con le dita dall'indossatore, che vengono riconosciuti dal visore). Tutto ciò è reso possibile grazie all'Holographic Processing Unit all'interno del dispositivo, che mantiene le informazioni sulle *gesture*, sugli ologrammi e sulla loro posizione.
Non si tratta più di semplice Realtà Aumentata bensì di Realtà Mista, che consente di modificare, spostare e interagire in tempo reale con gli oggetti virtuali proiettati nel mondo reale (Fig. 2.2).

Figura 2.2: Dispositivo HoloLens utilizzato a scopo didattico nel settore medico.

2.1.2 L’importanza di un visore MR

Questo paragrafo è volto a sensibilizzare il lettore sull’importanza e la potenza di un dispositivo del genere. Spesso vien scambiato per una nuova forma di intrattenimento, un gioco, ma una tecnologia del genere che sfrutta *Mixed Reality*, potrebbe radicalmente cambiare le nostre vite, ed essere un giorno alla portata di tutti.

Dal momento che questa tesi nasce da una collaborazione con l’Istituto Ortopedico Rizzoli, si vuole qui mostrare un esempio di utilizzo del dispositivo in ambito medico-chirurgico, nonostante sia attualmente sfruttato anche in altri settori.

In Fig. 2.3, si può vedere come dei chirurghi al St. Mary's Hospital di Londra riescano ad indossare il visore HoloLens mentre operano su pazienti sottoposti a chirurgia ricostruttiva degli arti inferiori. Ad effettuare l’esperimento e dimostrare che il visore potesse rivelarsi utile in campo medico, è stato un team dell'Imperial College di Londra, che ha utilizzato la tecnologia per sovrapporre le immagini delle scansioni TC, compresa la posizione delle ossa e dei vasi sanguigni principali, sulla gamba di ciascun paziente, consentendo a chirurgi di vedere attraverso l'arto durante l'intervento. Il dott. Philip Pratt, ricercatore presso il dipartimento di chirurgia e cancro e autore principale dello studio, pubblicato su *European Radiology Experimental* ha affermato: *"Attraverso questa serie iniziale di casi di pazienti, abbiamo dimostrato che la tecnologia è pratica e che può fornire un beneficio al team chirurgico: con gli HoloLens, guardi la gamba e in sostanza la vedi all'interno; vedi le ossa, il corso dei vasi sanguigni e puoi identificare esattamente dove si trovano i bersagli dell’operazione".*

Figura 2.3: La MR aiuta i chirurgi a vedere attraverso i tessuti.

2.1.3 Infrastrutture di contesto

Microsoft HoloLens attualmente sfrutta la piattaforma Windows Mixed Reality con sistema operativo Windows 10. Questo dispositivo dal peso di 579 grammi è dotato di una batteria di 16.5Wh, CPU e GPU, 2GB di RAM, 64GB di memoria, Wi-Fi 802.11 e Bluetooth per connettersi rispettivamente ad internet e con qualsiasi altro dispositivo.
Essendo un computer olografico lui stesso, risulta totalmente indipendente, ma il suo vero punto di forza consiste nell'analisi del mondo che lo circonda, grazie a diversi sensori, quali accelerometro, giroscopio e magnetometro, unitamente a cinque telecamere di cui una di profondità, per una visione più precisa e completa dell'ambiente circostante. L'Holographic Processing Unit precedentemente citato, permette addirittura di processare terabytes di informazioni dai vari sensori in tempo reale.
HoloLens è munito poi di una videocamera da 1.1 MegaPixel e da una fotocamera da 2.4 MegaPixel, per scattare foto in qualunque momento.
Le sofisticate lenti olografiche *see-through*, a differenza dei dispositivi di realtà virtuale, permettono di guardarvi attraverso senza coprire il mondo reale e proiettando in esso i vari ologrammi.

2.1.4 Emulatore HoloLens

L'emulatore HoloLens [8] potrebbe risultare particolarmente utile qualora si volesse realizzare un'applicazione MR che sfrutti la tecnologia Microsoft Hololens, senza però essere in possesso del dispositivo. Consente infatti di testare applicazioni olografiche sul proprio computer senza dover utilizzare il visore e viene fornito con il set di strumenti di sviluppo HoloLens. L'emulatore utilizza una macchina virtuale Hyper-V. Gli input umani e ambientali che normalmente vengono letti dai sensori sugli occhialini vengono invece simulati utilizzando la tastiera, il mouse o il controller Xbox. In fine Le applicazioni non hanno bisogno di essere modificate per essere eseguite sull'emulatore e non sanno che non sono in esecuzione su un vero HoloLens.

2.2 L’ambiente Unity

Dovendo gestire e manipolare interattivamente oggetti virtuali tridimensionali, risulta essenziale la scelta di un opportuno motore grafico.
Chiunque si imbatta nella piattaforma Unity [9], nota fin da subito che si tratta di un ambiente vastissimo che offre infinite opportunità (logo in Fig. 2.3), pertanto nel seguito non si andrà eccessivamente nel dettaglio, ma verranno esplicitate le principali caratteristiche dell’interfaccia di Unity al fine di comprendere meglio i capitoli successivi.

2.2.1 Principali caratteristiche

Unity, piattaforma notoriamente utilizzata per la realizzazione di ambienti tridimensionali, implica l’utilizzo di programmazione a oggetti, supportando 3 diversi linguaggi, C#, Javascript e Boo anche se C# è quello mediamente più utilizzato. Si tratta di un motore grafico e in quanto tale facilita lo sviluppo di applicazioni offrendo librerie con funzioni, classi e dati già definiti senza dover programmare tutto quanto da zero. Originariamente le piattaforme di gioco erano software estremamente costosi, ma Unity scaricabile gratuitamente, ha segnato un punto di svolta costituendo un’importante innovazione nella comunità scientifica.
Per aggiungere interattività tra i vari oggetti creati in un progetto è necessario ricorrere a degli script e a tal fine può essere comodo utilizzare Visual Studio come ambiente di sviluppo.

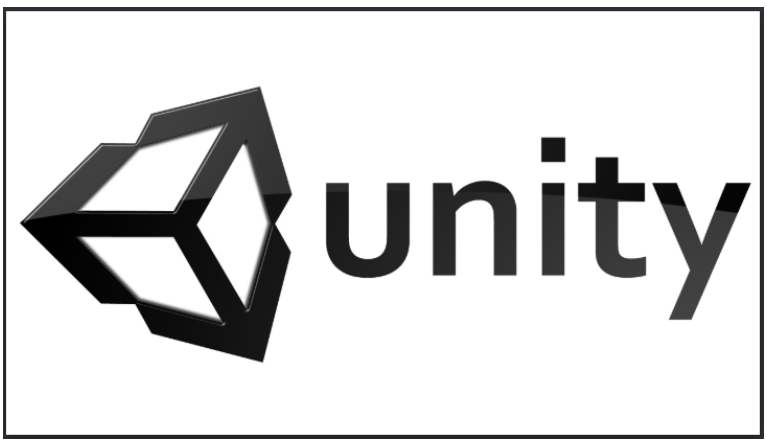

Figura 2.3: Logo di *Unity*.

2.2.2 Una piattaforma per MR

Tra i motori di gioco più utilizzati dalle aziende, vi sono Unreal Engine, che offre le migliori esperienze di gioco simulato in prima persona, Cry Engine, che vanta una grafica tra le più potenti attualmente sul mercato, Anvil Engine, apprezzato per gli elevati livelli di intelligenza artificiale e per l'interazione ambientale che offre, ed in fine il sopra citato Unity. Quest'ultimo rappresenta attualmente la piattaforma di riferimento per quanto riguarda la realizzazione e la sperimentazione in ambito VR, AR e MR e proprio per questo motivo è stato scelto tra tanti. Dalla versione 2017.2 in poi infatti, importando in un progetto il "MixedRealityToolkit" (HoloToolkit), è possibile sfruttare tutte le funzionalità in esso contenute per creare facilmente e velocemente applicazioni HoloLens.

2.2.3 Basi per lo sviluppo di un progetto Unity

Aprendo Unity la schermata iniziale che compare mostra due pannelli: *Projects* contiene tutti i progetti creati, mentre *Getting started* contiene un piccolo video introduttivo. Tre pulsanti sulla destra permettono di scegliere un progetto su un'altra directory, creare un nuovo progetto e gestire l'account Unity personale. Cliccando su *New* la schermata presentata è quella mostrata in Fig. 2.4.

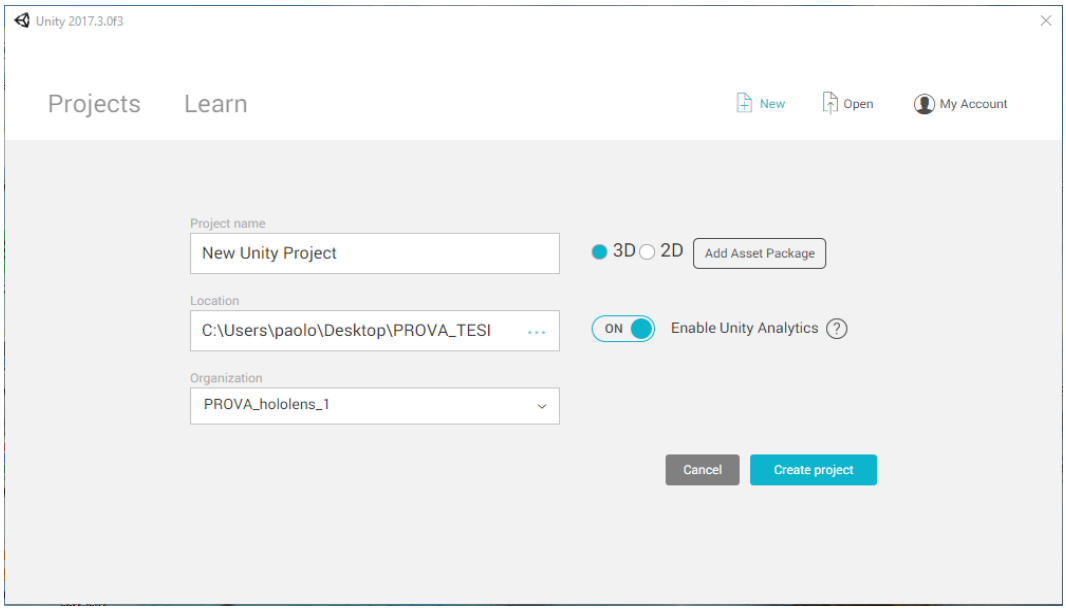

Figura 2.4: Schermata di creazione di un nuovo progetto.

Una volta scelto il nome del progetto e la directory sul quale salvarlo, è possibile inoltre scegliere se creare un progetto 3D o 2D e se importare degli *Asset Packages*.
Gli *Asset Packages* altro non sono che dei pacchetti predefiniti contenenti script, mesh, animazioni e molte altre cose che possono essere utili durante la creazione di videogiochi e applicazioni. Questi possono poi essere importati anche in un secondo momento.

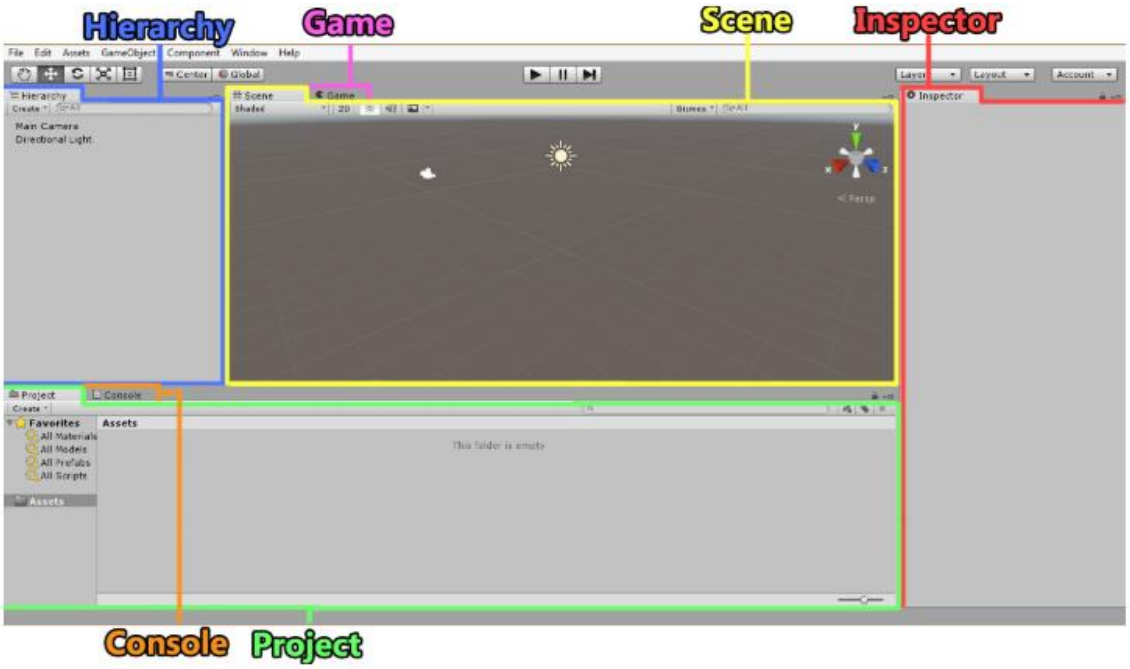

Figura 2.5: Schermata principale di un progetto.

In Fig. 2.5, vengono mostrati 6 diversi pannelli che costituiscono l'interfaccia base di Unity. Nel pannello *Scene*, viene mostrata la scena e gli oggetti che sono presenti all'interno di essa. Gli oggetti sono i vari elementi che compongono l'applicazione: telecamera, luci, modelli 3D…
All'interno di ogni progetto possono essere presenti una o più scene. Di base sono già presenti degli oggetti nella scena: una *Camera* e *una Directional light*. La prima costituisce la vista della scena, ovvero ciò che l'utente vedrà in tempo d'esecuzione, mentre la seconda specifica la direzione della luce (eventualmente si può eliminare).
Nel pannello *Hierararchy panel* vengono elencati tutti gli oggetti presenti nella scena. Con il pulsante *Create* si possono inserire nuovi oggetti nella scena, inoltre si possono creare delle gerarchie tra due o più oggetti, semplicemente selezionandoli e trascinandoli uno sull'altro per creare una parentela. L'oggetto esterno verrà chiamato "padre", mentre

quello interno sarà il “figlio”. In questo modo spostando l’oggetto padre, questo sarà seguito dall’oggetto figlio.
Nella *Console panel* è possibile testare il corretto funzionamento di uno script.
Nell’ *Inspector panel* ci sono tutte le impostazioni relative all’oggetto selezionato.
Nel *Game panel* viene mostrata un’anteprima del videogioco: la visuale che ci verrà mostrata sarà quella mostrata dalla Camera presenti nel gioco.

2.2.4 Materiali e Shader

È necessario creare un materiale diverso per ogni tipo di oggetto di cui si ha bisogno. I materiali in Unity hanno alla base uno *shader* che definisce come un oggetto venga disegnato ed ombreggiato. Per creare un nuovo materiale è sufficiente cliccare col tasto destro sul *Project Panel* e selezionare *Create > Material* dal menu a tendina.
Gli *shader* quindi altro non sono che dei modelli matematici che indicano alla scheda grafica come rendere a schermo un oggetto 3D. Descrivono essenzialmente come (e se) l’oggetto riceve la luce, che colore ha, se proietta ombre, se riceve ombre, se crea rifrazioni, se è lucido, opaco, trasparente, e una miriade di altre informazioni.
Più avanti ad esempio, si vedrà l’utilizzo di un opportuno *shader* “Intersection_Color” per evidenziare le parti di un oggetto interessate da un piano di taglio (Fig. 2.6).

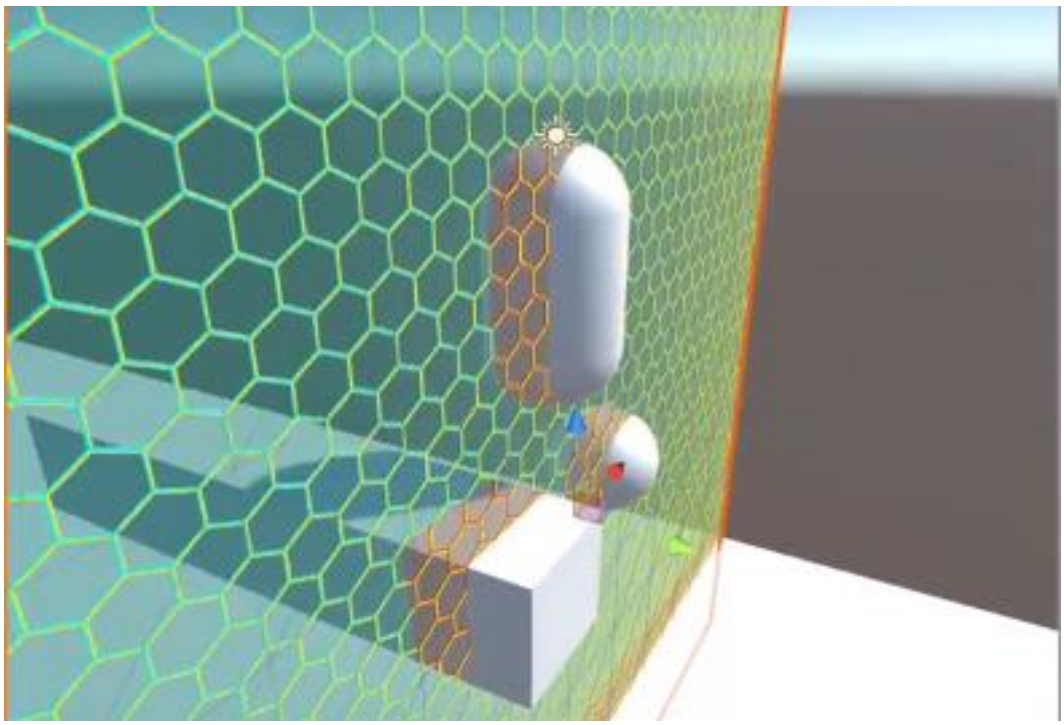

Figura 2.6: *shader* “*Intersection_Color”* applicato a tre oggetti 3D.

2.2.5 Componenti aggiuntivi

RigidBody è un componente che se aggiunto ad un oggetto, ne vincola il movimento al controllo del motore fisico di Unity. Si vedrà in più occasioni il suo utilizzo nei capitoli successivi, per consentire agli oggetti di cadere per effetto di gravità, spuntando opportunamente *"Use Gravity"*, o conferendo loro un comportamento "Is Kinematic" per ignorare forze e collisioni esterne che li scaraventerebbero via al primo contatto. Fondamentali per consentire collisioni ed interazioni sono appunto i *collider*, "collisori". *Collider* è la classe base da cui ereditano tutti i tipi di *collider* come *BoxCollider*, *SphereCollider*, *CapsuleColider* rispettivamente a forma di cubo, sfera e capsula. Non devono per forza corrispondere alla geometria reale dell'oggetto ma solitamente è bene assicurarsi che si adattino perfettamente all'oggetto per garantire un'interazione corretta. Se serve un *Collider* più dettagliato e preciso, viene messo a disposizione il componente *Mesh Collider*, componente che verrà ampliamente utilizzato per la realizzazione del progetto finale. Una *mesh* poligonale infatti altro non è che un reticolo che definisce e racchiude un oggetto nello spazio, composto da vertici, spigoli e facce. Unity in particolare, supporta maglie poligonali triangolate o quadrangolate (Fig. 2.7).

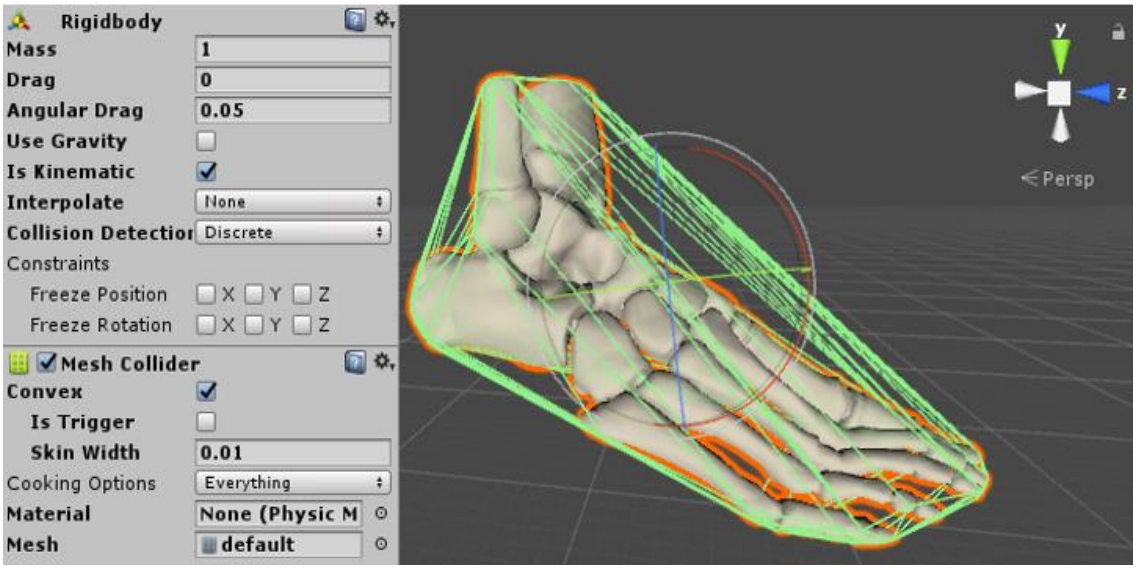

Figura 2.7: *Mesh Collider* e *RigidBody* applicati a un oggetto 3D.

Si è deciso di dedicare ampio spazio a Unity non solo perché rappresenta un ottimo strumento per lo sviluppo di applicazioni *Mixed Reality*, ma perché è stato utilizzato durante tutte le fasi di progettazione, fino alla realizzazione dell'elaborato finale.

Capitolo 3

Principali dispositivi Hardware e Software utilizzati

Come accennato nell'Introduzione, l'elaborato di tesi che si vuole presentare, si serve della *Mixed Reality* e nasce in ambito medico da una collaborazione tra l'Istituto Ortopedico Rizzoli e il Dipartimento di Ingegneria Industriale di Bologna. Lo sviluppo dell'applicazione finale prevede tre fasi, con la realizzazione dei relativi progetti:

1. *Cut Project,* per il taglio di elementi digitali tridimensionali.
2. *Tracking Project,* per eseguire il *tracking* di oggetti reali.
3. *Mixed Reality Project,* per inserire i precedenti progetti in ambito MR.

Per questo motivo l'elaborato finale prende il nome di *"Mixed Reality Tracking Cut Project",* in breve MRTC. In questo capitolo vengono però introdotti solamente gli specifici strumenti e tecnologie all'avanguardia utilizzate nelle varie fasi di sviluppo, mentre si parlerà ampliamente delle caratteristiche dei vari progetti nei capitoli successivi.

3.1 InVesalius

Per poter lavorare con le radiografie fornite dall'Ospedale Rizzoli, e poterle visualizzare ed elaborare in 3D è stato indispensabile l'utilizzo di InVesalius [10].
Si tratta di un software gratuito, disponibile per piattaforme Microsoft Windows, GNU / Linux e Apple Mac OS X, utilizzato per la ricostruzione di immagini medicali da tomografia computerizzata o apparecchiature di risonanza magnetica.
Da qualche tempo anche in Italia è prassi rilasciare al cliente un cd contenente file con estensione *DICOM* relativi ad esempio a radiologie, lastre o simili. Solitamente nel cd viene fornito anche un software in grado di aprire queste immagini, ma generalmente non va oltre la semplice visualizzazione. InVesalius invece oltre che aprire i file in formato *DICOM*, è in grado di esportare gli stessi con estensione *.OBJ* e. *.STL*

rendendone possibile la manipolazione attraverso il motore grafico Unity e la stampa in 3D. Quest'ultima viene sempre più spesso utilizzata per produrre protesi, tutori e altri dispositivi analoghi in maniera più rapida e soprattutto più personalizzabile, e nel nostro caso è stata sfruttata più e più volte per testare l'applicazione finale.

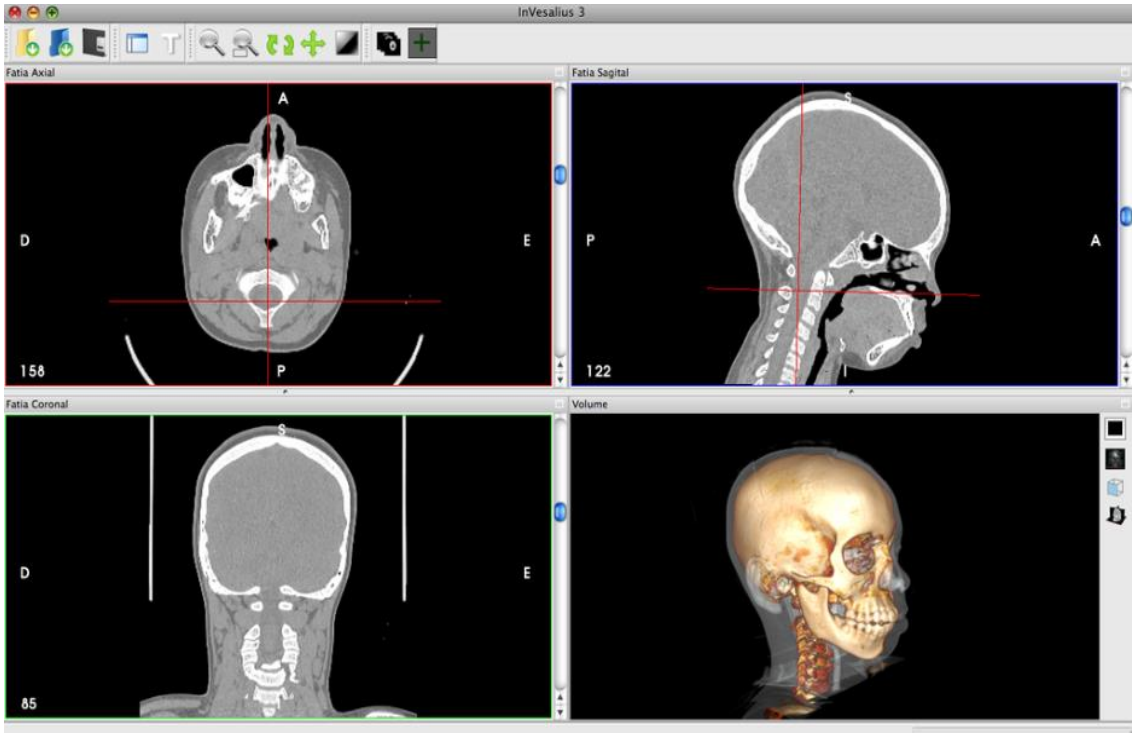

Figura 3.1: InVesalius 3, elaborazione della radiografia di un cranio.

3.2 MarkForged Mark Two

Si è visto sopra il software utilizzato per la ricostruzione di oggetti *.OBJ* e *.STL,* con supporto alla stampa 3D, ma può rivelarsi estremamente utile ai fini dello sviluppo del progetto, realizzare anche i prototipi veri e propri su cui testare l'applicazione. È necessario dunque scegliere il dispositivo più adeguato a creare fisicamente gli oggetti tridimensionali, ricostruiti con InVesalius.

Tra le varie stampanti 3D professionali, Mark Two [11], il cui prezzo si aggira attorno ai € 13.500 è in grado di realizzare prodotti finiti resistenti in Nylon, rinforzati con materiali compositi quali fibra di carbonio, fibra di vetro e kevlar. È proprio quest' ultima caratteristica a conferire agli oggetti stampati una durata, una rigidità e una resistenza impareggiabili rispetto a tutte le stampanti del settore. Nella figura sottostante,

un esempio di due oggetti ricostruiti a partire da radiografie per mezzo di InVesalius 3, successivamente stampati con Mark Two.

Figura 3.2: Stampa 3D di ossa provenienti da radiografie.

3.3 Blender e MeshLab

Più volte nella fase di sviluppo dell'applicazione si è ricorso ad altri strumenti di modellazione 3D quali Blender e Meshlab [12]. Quest'ultimo è stato utilizzato in particolare per effettuare il *remeshing* di *mesh* complesse ovvero (come si vedrà nei capitoli successivi) per semplificarle e ridurne il numero di facce, triangoli e spigoli (Fig. 3.3). Si è deciso di utilizzare MeshLab perché si tratta di un software gratuito, open-source efficace ed estremamente facile da utilizzare. Lo stesso non si può dire di Blender, software libero e multipiattaforma più complesso in quanto offre svariate opportunità tra cui modellazione, rivestimento di particelle, animazione, montaggio video, composizione e *rendering* (ossia conversione mediante apposito software del profilo di un'immagine bidimensionale in un'immagine dall'aspetto realistico e percepibile come tridimensionale). Dispone inoltre di funzionalità per mappature *UV* (dall'inglese "*UV mapping"*), dove *U* e *V* altro non sono che i nomi scelti convenzionalmente per identificare gli assi cartesiani. Nel paragrafo successivo verrà mostrata l'importanza di

queste coordinate, i cui valori variano tra 0 e 1, che vengono attribuite ai vertici degli oggetti solidi per poter lavorare le relative *mesh* secondo le specifiche del progetto.

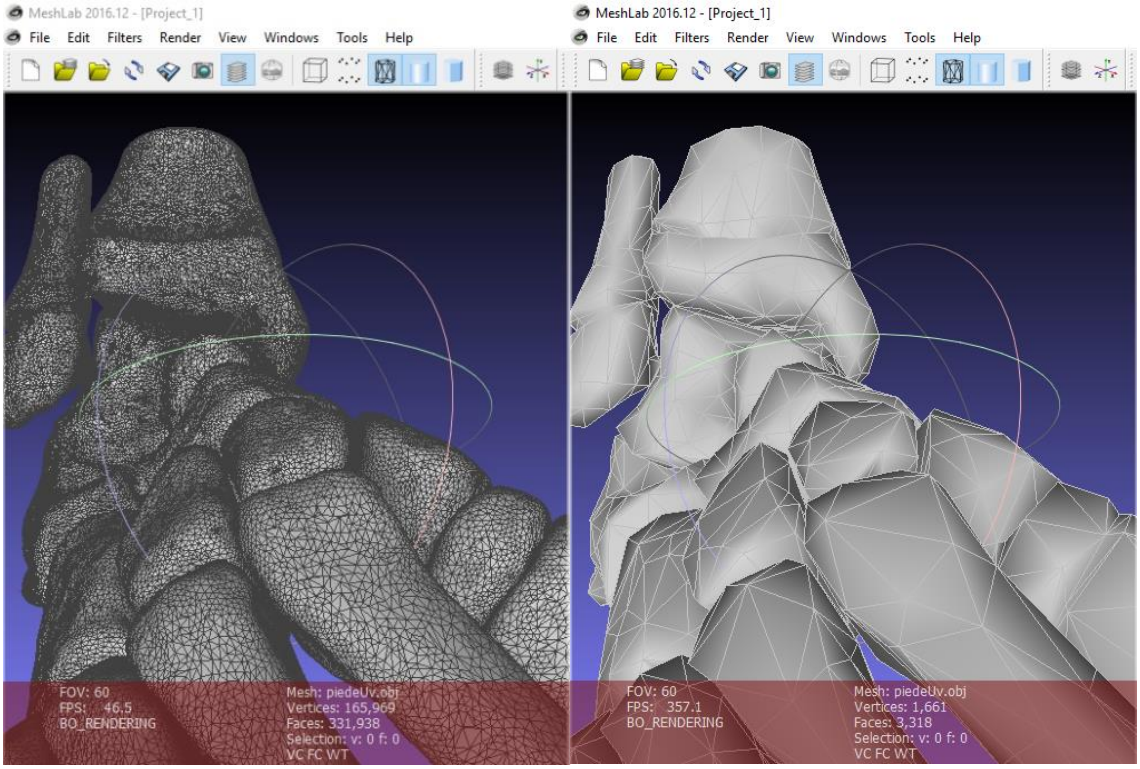

Figura 3.3: *Remeshing* delle ossa di un piede, prima e dopo.

3.3.1 Mappatura UV con Blender

La tecnica di mappatura UV (dall'inglese "*UV mapping"*, dove *U* e *V* sono semplicemente nomi convenzionali di assi cartesiani) è indispensabile per applicare un'immagine bidimensionale (detta "*texture*") ad un modello 3D. Dal momento che un oggetto *.OBJ* o *.STL* è inizialmente sprovvisto di queste speciali coordinate, è necessario trovare un mezzo con cui applicare questa operazione, che sostanzialmente consiste nell' "appiattire" la *mesh* su un piano sul quale giace la *texture* con una tecnica detta "*unwrapping*". In altri termini, è necessario sfruttare appositi strumenti (come Blender) per distribuire su di un piano cartesiano le facce che costituiscono un qualunque oggetto. Fatto ciò, ogni vertice dell'oggetto tridimensionale disporrà di un set di coordinate bidimensionali condiviso con

l'immagine e quindi le coordinate UV fungeranno da ponte tra lo spazio bidimensionale delle immagini e quello tridimensionale della *mesh* (vedi Fig. 3.4).

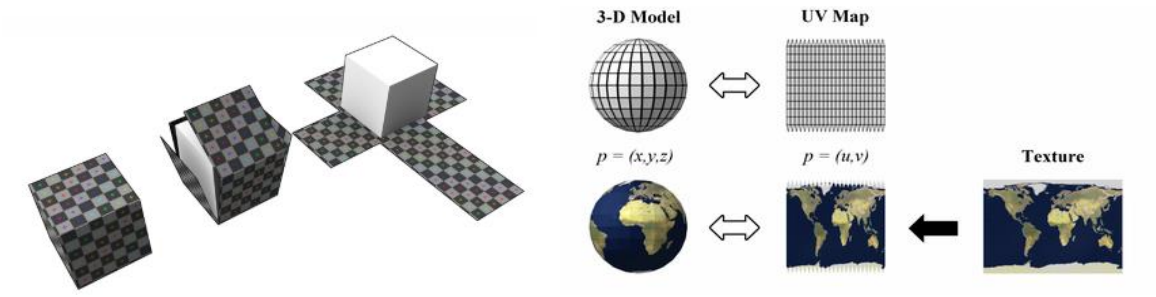

Figura 3.4: *Unwrapping* di un cubo e mappatura UV.

Blender è stato utilizzato proprio allo scopo di eseguire l'*unwrapping* di un oggetto mediante l'apposita funzione *unwrap* nella modalità *"edit mode"* (Fig.3.5), che consente di selezionare specifiche facce, applicarvi le coordinate UV e infine deformarle in modo da adattare la *texture* alla *mesh.*

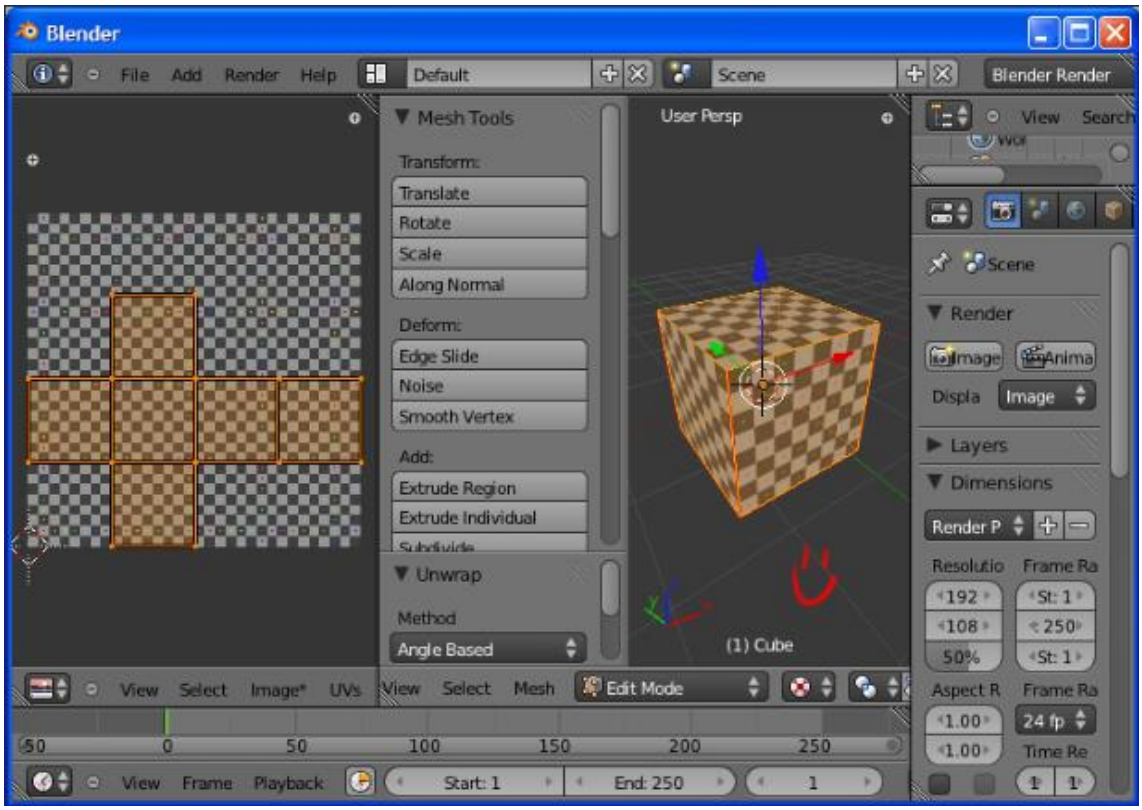

Figura 3.5: *Unwrapping* e mappatura Uv con Blender.

3.4 Vuforia

Altrettanto importante per la realizzazione della seconda fase del progetto MRTC, è stato Vuforia [13]. Si tratta di una piattaforma software per lo sviluppo di AR, che dal 2016 ha sviluppato un supporto per dispositivi Microsoft tra cui HoloLens.
È stato oggetto del mio studio in quanto la sua libreria, che può essere integrata in Unity si è rivelata di estrema utilità offrendo tecniche di *tracking* (dove per *tracking* si intende la possibilità di riconoscere ed acquisire attraverso telecamera o altro dispositivo degli oggetti in tempo reale) quali *VuMark*, *Image Target*, ma soprattutto *Model Target*.
Al fine di prendere più confidenza col software, viene inoltre offerta la possibilità ai nuovi utenti iscritti di fare diversi esperimenti di Realtà Aumentata (con scadenza dopo 30 giorni), tramite dispositivi desktop e mobili (da notare *Vuforia View Application* per cellulare).

3.4.1 *VuMark*

La più comune tra le tecniche di *tracking* è probabilmente *VuMark*, che sfrutta uno speciale codice a barre di ultima generazione, per stabilire la posizione 0 su cui appoggiare un qualsiasi oggetto virtuale specificato (Fig.3.6).

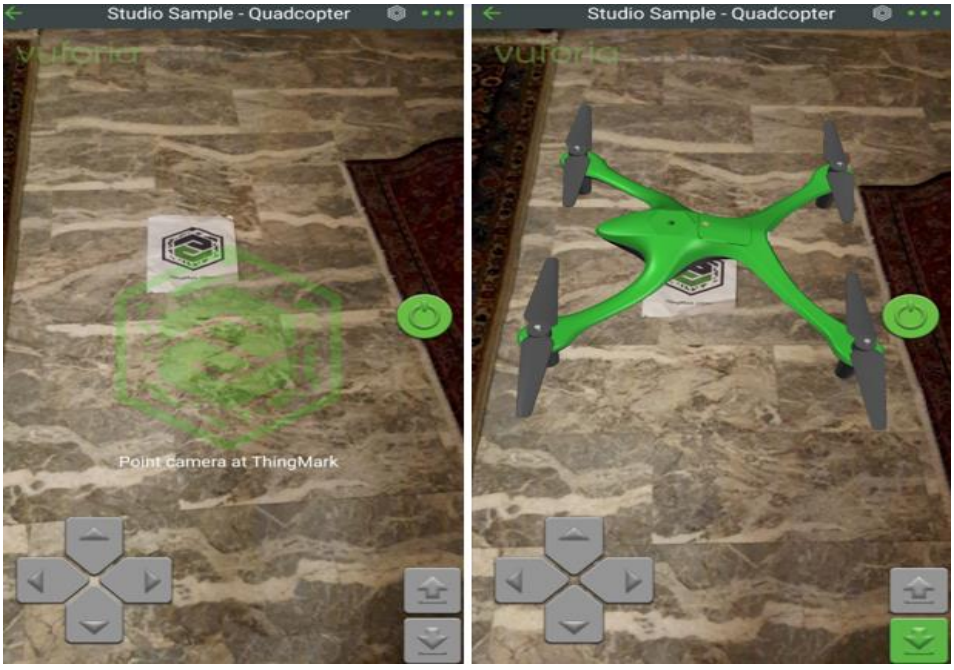

Figura 3.6: A sinistra il codice *VuMark*, a destra l'oggetto virtuale associato.

3.4.2 *Image Target*

Questa tecnica di *tracking* prevede come suggerisce il nome stesso, l'utilizzo di "*Image Target",* vale a dire immagini bersaglio che Vuforia è in grado di rilevare e tracciare. Questi, a differenza dei tradizionali indicatori fiduciali, dei codici *VuMark* e *QR*, non necessitano di speciali regioni o codici in bianco e nero per essere riconosciuti, perché il motore li rileva e ne tiene traccia (Fig. 3.7).

Figura 3.7: tracking di un *Image Target* e associazione di un astronauta virtuale.

3.4.3 *Model Target*

L'ultima tecnica di *tracking* qui descritta, rappresenta la conclusione della seconda fase del progetto MRTC, ovvero la realizzazione di *Tracking Project*.
I *Model Target* vengono utilizzati per l'acquisizione e l'elaborazione in tempo reale di elementi tridimensionali del mondo esterno. Disponendo semplicemente di un oggetto virtuale tridimensionale salvato in memoria, questa tecnica di *tracking* consente di riconoscere il corrispondere oggetto reale semplicemente inquadrando con la telecamera un lato di quest'ultimo. Una volta riconosciuto avviene il *tracking*, quindi viene sovrapposto l'oggetto virtuale manipolabile all'oggetto reale.

Vuforia a tal scopo mette a disposizione degli utenti iscritti uno strumento, *"Vuforia model target generator",* per caricare oggetti 3D in formato *.OBJ*, *.STL* e molti altri e generare i corrispondenti *Model Target*. Importando in Unity l'apposita libreria per questa tipologia di *tracking*, viene fornito un database contenente i dati di un *Model Target* d'esempio, in modo da vedere subito questa tecnologia in azione (Fig. 3.8).

Figura 3.8: *tracking* di un *Model Target*.

3.5 Leap Motion

L'ultima tecnologia acquistata, studiata ed utilizzata per la realizzazione del progetto è Leap Motion [14]. Grazie a questo dispositivo, considerato attualmente uno dei più avanzati strumenti di tracking per le mani, è stato possibile realizzare il *Mixed Reality Project* (vedi capitoli successivi), ultimo step per l'applicazione finale da presentare all'Ospedale Rizzoli. Si tratta di una piccola e leggera periferica USB la cui lunghezza sfiora gli 8 cm (vedi Fig. 3.9), attualmente acquistabile al prezzo di €70, pensata per essere posta su un tavolo, rivolta verso l'alto per identificare mani e dita con una precisione di 0.01 mm. Nonostante le sue contenute dimensioni infatti, non solo ricopre un'area semisferica di circa 1 metro, ma risulta estremamente precisa eseguendo il tracking delle mani fino a 200 fotogrammi al secondo e sfruttando 2 telecamere e 3 LED infrarossi.

Figura 3.9: *Tracking* delle mani per mezzo di Leap Motion.

Si è deciso in questo capitolo di introdurre Leap Motion così come InVesalius, MarkForged, Blender, MeshLab e Vuforia al fine di fornire le conoscenze di base necessarie alla comprensione dei capitoli successivi, che tratteranno delle principali fasi di sviluppo di progetto sfruttando appunto queste tecnologie.

Capitolo 4

Cut Project: primo step

In questo capitolo viene descritto il primo fondamentale progetto realizzato sfruttando Unity, ossia *Cut Project,* letteralmente *"Progetto Taglio"*, da cui deriva come già accennato, l'ultima lettera nella sigla MRTC e sfruttato in generale per effettuare il taglio anche multiplo di oggetti virtuali in tempo reale.

4.1 Specifiche del progetto

Questo progetto rappresenta il primo step verso la realizzazione dell'applicazione finale e ne costituisce anche il fulcro.
L'Istituto Ortopedico Rizzoli infatti ha richiesto esplicitamente di poter usufruire di uno strumento (come Microsoft HoloLens o Leap Motion) per effettuare il taglio in tempo reale di ossa virtuali affiancate a quelle reali, in un ambiente di visualizzazione interattivo (dunque non un semplice sistema CAD).
Alla base dell'applicazione finale, che dovrà sfruttare la *Mixed Reality* per garantire l'interazione con gli oggetti tridimensionali, vi sono specifici script per sezionare un qualsiasi ologramma proiettato con visore HoloLens e manipolare liberamente ogni suo frammento ottenuto. Per prima cosa quindi, mi sono occupato di trovare un modo per tagliare oggetti tridimensionali di ogni forma e dimensione gestendone poi ogni sotto-parte. A tal scopo Github, servizio di hosting per la programmazione collaborativa, si è rivelato una risorsa preziosa. Nella sua "*community*" infatti vien messo a disposizione il pacchetto *BLINDED_AM_ME* contenente diversi script per effettuare il taglio di *mesh.* In questa prima fase son stati esportati dal pacchetto citato solamente due script da riadattare completamente al diverso contesto: *ExampleUseof_MeshCut.cs* e *MeshCut.cs.*

4.2 *ExampleUseof_MeshCut.cs* in origine

Analizzando prima lo script più semplice, vale a dire *ExampleUseof_MeshCut.cs*, si può notare che nelle poche righe di codice sotto riportate (Fig. 4.1) sostanzialmente specifica come usare i frammenti ottenuti in seguito al taglio. Il comportamento ottenuto (vedi Fig. 4.2) è simile a quello alla base dell'applicazione per cellulare *Fruit Ninja*.

```
public class ExampleUseof_MeshCut : MonoBehaviour {

    public Material capMaterial;

    void Update(){

        if(Input.GetMouseButtonDown(0)){ //se premo il tasto sinistro del mouse (0)
            RaycastHit hit;

            if(Physics.Raycast(transform.position, transform.forward, out hit)){ // Raycast restituisce il primo oggetto colpito dal raggio.

                GameObject victim = hit.collider.gameObject;
                GameObject[] pieces = BLINDED_AM_ME.MeshCut.Cut(victim, transform.position, transform.right, capMaterial);
                //l'array pieces contiene ora i due pezzi (parte sinistra e destra) ottenuti dal taglio.

                if(!pieces[1].GetComponent<Rigidbody>()) //se il pezzo di destra non ha un componente RigidBody,
                    pieces[1].AddComponent<Rigidbody>(); //aggiungo il componente RigidBody che di default utilizza
                                                         //"Use Gravity" conferendo l'effetto di gravità all'oggetto.

                Destroy(pieces[1], 1); //distruggo il pezzo di destra tagliato.
            }
        }
    }
    void OnDrawGizmosSelected() { //OnDrawGizmosSelected indica che il piano di taglio è visibile solo se l'oggetto è selezionato

        Gizmos.color = Color.green;//colore del piano di taglio
        //disegno le linee che delimitano il piano di taglio
        Gizmos.DrawLine(transform.position, transform.position + transform.forward * 5.0f);
        Gizmos.DrawLine(transform.position + transform.up * 0.5f, transform.position + transform.up * 0.5f + transform.forward * 5.0f);
        Gizmos.DrawLine(transform.position + -transform.up * 0.5f, transform.position + -transform.up * 0.5f + transform.forward * 5.0f);
        Gizmos.DrawLine(transform.position, transform.position + transform.up * 0.5f);
        Gizmos.DrawLine(transform.position,  transform.position + -transform.up * 0.5f);

    }
}
```

Figura 4.1: Codice originale di *ExampleUseof_MeshCut.cs* (C#).

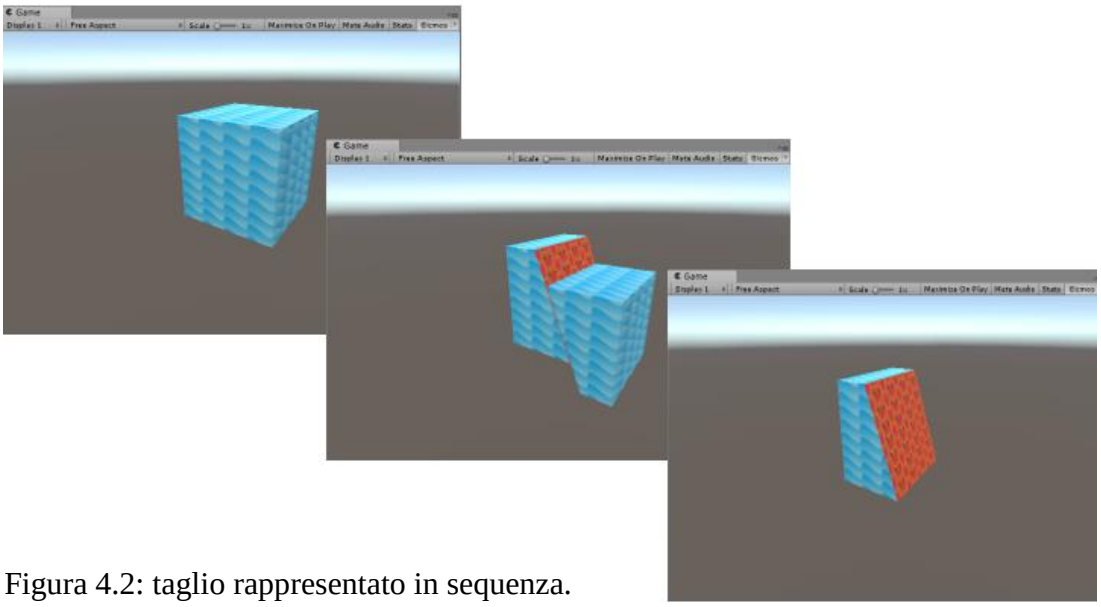

Figura 4.2: taglio rappresentato in sequenza.

Servendosi della documentazione sulle API di scripting in docs.unity3d.com, è stato facile capire la logica celata dietro alle poche linee di codice sopra riportate. Di seguito vengono analizzate passo per passo le principali caratteristiche per poi mostrare in maniera più efficiente nel paragrafo successivo le modifiche apportate.
alla settima riga di codice, Input.GetMouseButtonDown(0), indica il verificarsi dell'evento della pressione del tasto sinistro (se si specifica in ingresso 0) del mouse. Pertanto, il taglio viene effettuato con un click nella Scena di Unity. Poco più sotto, Physic.Raycast viene utilizzato per inizializzare un raggio da un punto sorgente *origin* lungo una direzione *direction* contro tutti i *collider* che incontra nella scena, restituendo il primo colpito come oggetto di tipo RaycastHit. Dunque "RaycastHit hit" rappresenta solo il primo oggetto colpito dal raggio (se presente) e qualora vi fossero più elementi uno dietro all'altro verrebbe comunque considerato dal taglio solamente il primo. GameObject victim, alla riga 12, è appunto l'oggetto vittima del taglio, che vien poi passato come parametro alla funzione Cut di *MeshCut*, che effettua il taglio vero e proprio. Dei due pezzi "pieces" creati al momento del taglio, viene aggiunto a quello di destra "pieces[1]" il componente *RigidBody* (riga 17), che conferisce all'oggetto l'effetto di gravità, attraendolo verso il basso. Subito sotto, Destroy(pieces[1],1), distrugge il frammento asportato dall'oggetto originale. I "*Gizmos*" nella funzione OnDrawGizmosSelected(), vengono utilizzati per dare il debug visivo o aiuti di configurazione nella vista della scena, quindi nel caso specifico vengono utilizzati per disegnare il piano di taglio.

4.3 *ExampleUseof_MeshCut.cs* modificato

Facendo sempre riferimento alla Figura 4.1, una prima limitazione dello script precedentemente mostrato ed analizzato, si scorge alla riga 7 del codice: Physic.Raycast inizializza un raggio che interessa solo il primo oggetto incontrato, quando invece si potrebbe voler tagliare più oggetti insieme nella stessa scena.
Ancora meglio potrebbe essere comodo offrire all'utente la possibilità di scegliere la lunghezza di un coltello virtuale entro la quale tagliare tutti gli oggetti nel raggio d'azione. Ho operato così una prima modifica sfruttando il metodo *"RaycastAll"* della classe Physic (riga 12, Fig.4.3), che restituisce un vettore di tutti i *RaycastHit* colpiti dal raggio, ed ho aggiunto una variabile "bladeLength" (riga 4 del codice sottostante) settabile dall'utente (di lunghezza pressoché infinita di default), entro la quale tagliare

tutti gli oggetti sovrapposti presenti in scena. Al fine di evitare la caduta per effetto di gravità della parte tagliata e la sua rimozione è stato sufficiente commentare le tre righe di codice relative (dalla 21 alla 23, Fig.4.3). Per quanto riguarda la funzione OnDrawGizmosSelected invece, anche se non ne è stato riportato il codice, è stato sufficiente tener conto della lunghezza settabile del coltello per disegnare il piano di taglio con le eventuali misure corrette.

```
1  public class ExampleUseof_MeshCut : MonoBehaviour {
2
3  public Material capMaterial;
4  public float bladeLength;
5
6  void Update(){
7
8      if(Input.GetMouseButtonDown(0))
9      {//se premo il tasto sinistro del mouse (0)
10
11         RaycastHit[] hits;
12         hits = Physics.RaycastAll(transform.position, transform.forward, 100.0F);//RaycastAll restituisce tutti i RaycastHit colpiti dal raggio
13         if (hits.Length>0  ){
14             for (int i = 0; i < hits.Length; i++)
15             {
16                 RaycastHit hit = hits[i];
17                 if (bladeLength>0 && hit.distance < bladeLength || bladeLength<=0 )//se ho settato la lunghezza del coltello,taglio solo gli
18                 {                                                                  //oggetti entro tale lunghezza,altrimenti tutti quelli che incontro.
19                     GameObject victim = hit.collider.gameObject;
20                         GameObject[] pieces = BLINDED_AM_ME.MeshCut.Cut(victim, transform.position, transform.right, capMaterial);
21                         //if(!pieces[1].GetComponent<Rigidbody>())
22                         //pieces[1].AddComponent<Rigidbody>(); aggiunge la gravità all'oggetto tagliato.
23                         //Destroy(pieces[1], 1); distrugge il pezzo di destra tagliato dal coltello.
24                 }
25             }
26         }
27     }
28 }
```

Figura 4.3: *ExampleUseof_MeshCut.cs* modificato (C#).

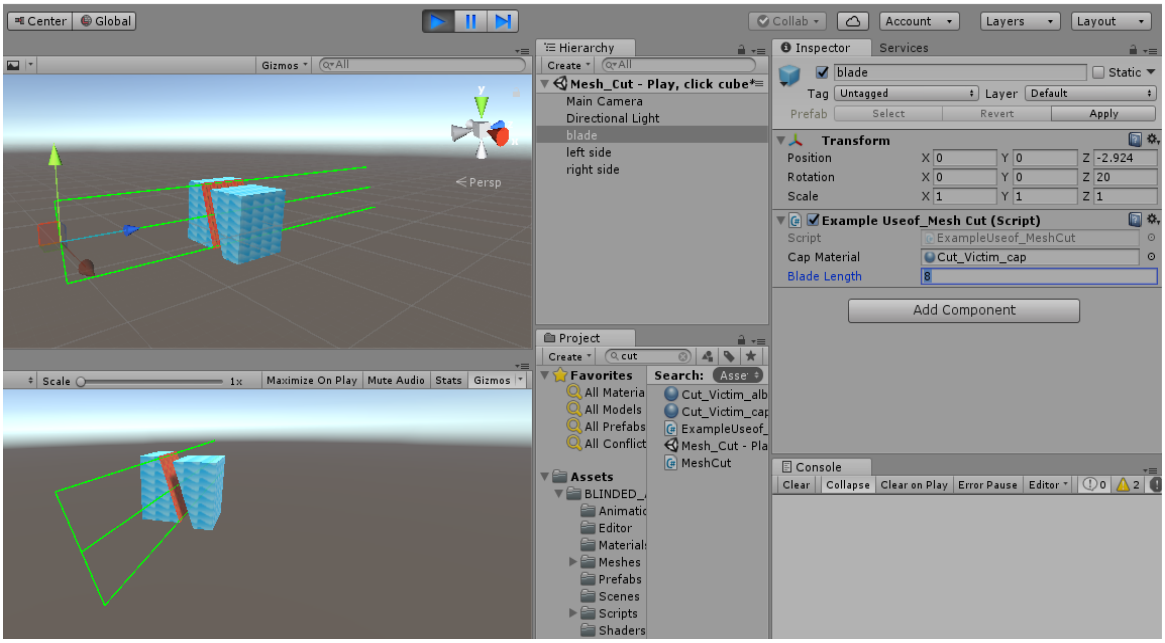

Figura 4.4: Funzione OnDrawGizmosSelected per disegnare il piano di taglio.

4.4 *MeshCut.Cut*

Il pacchetto *BLINDED_AM_ME* offre un ulteriore script scritto in C# "*MeshCut.cs"* per il taglio vero e proprio di *mesh*. Sebbene ci sarebbe tanto da dire in merito a questa classe, viene analizzata in questo paragrafo solamente la funzione Cut di nostro interesse (Fig.4.5), richiamata nello script *ExampleUseof_MeshCut.cs* appena visto. Anche qui lo script necessita di modifiche e presenta alcune limitazioni non banali.

Dal momento che il progetto originale, come si è visto, prevedeva che la parte di destra tagliata andasse distrutta, di conseguenza non vi era alcun bisogno di racchiudere la parte di destra rimossa in un opportuno *collider* (*BoxCollider, SphereCollider, CapsuleColider*, ecc…) in quanto non doveva esser fatta nessun'altra operazione su quest'ultima. Ma a noi invece interessa poter tagliare uno o più oggetti in tutte le sotto-parti che desideriamo, pertanto è bene che ogni frammento (che ora non vien più disintegrato come prima) sia racchiuso in un *collider* appropriato.

```
public static GameObject[] Cut(GameObject victim, Vector3 anchorPoint, Vector3 normalDirection, Material capMaterial){

    _blade = new Plane(victim.transform.InverseTransformDirection(-normalDirection),// set the blade relative to victim
    victim.transform.InverseTransformPoint(anchorPoint));
    _victim_mesh = victim.GetComponent<MeshFilter>().mesh;// get the victims mesh
    _new_vertices.Clear();// reset values
    _leftSide  = new Mesh_Maker();
    _rightSide = new Mesh_Maker();
    bool[] sides = new bool[3]; int[] indices; int   p1,p2,p3;

    for(int sub=0; sub<_victim_mesh.subMeshCount; sub++)... // operazioni per il taglio della mesh

    Material[] mats = victim.GetComponent<MeshRenderer>().sharedMaterials;

    Mesh left_HalfMesh = _leftSide.GetMesh();// Left Mesh
    left_HalfMesh.name =  "Split Mesh Left";

    Mesh right_HalfMesh = _rightSide.GetMesh();// Right Mesh
    right_HalfMesh.name = "Split Mesh Right";

    victim.name = "left side";//l'oggetto vittima tagliato e ridotto diventa la leftSide
    victim.GetComponent<MeshFilter>().mesh = left_HalfMesh;

    GameObject leftSideObj = victim;// assegno alla leftSide il valore di victim (modificato dal taglio)
    //l'oggetto rightSide vien creato da zero, accanto alla leftSide.
    GameObject rightSideObj = new GameObject("right side", typeof(MeshFilter), typeof(MeshRenderer));
    rightSideObj.transform.position = victim.transform.position;//si assegna alla rightSide la stessa posizione della leftSide
    rightSideObj.transform.rotation = victim.transform.rotation;//si assegna alla rightSide la stessa rotazione della leftSide
    rightSideObj.GetComponent<MeshFilter>().mesh = right_HalfMesh;

    if(victim.transform.parent != null){
        rightSideObj.transform.parent = victim.transform.parent;
    }

    rightSideObj.transform.localScale = victim.transform.localScale;

    // assign mats
    leftSideObj.GetComponent<MeshRenderer>().materials = mats;
    rightSideObj.GetComponent<MeshRenderer>().materials = mats;

    return new GameObject[]{ leftSideObj, rightSideObj };
}//end Cut
```

Figura 4.5: codice per la realizzazione della funzione Cut (C#).

Al fine di comprendere meglio le modifiche apportate al codice sopra riportato, di seguito se ne spiega brevemente il funzionamento. Come si può notare nella prima linea, la funzione Cut prende come parametri di ingresso l'oggetto soggetto al taglio "victim", la sua posizione e la sua direzione nelle coordinate x, y e z racchiuse nel Vector3, ed infine il materiale che lo costituisce. Nella riga subito sotto viene creato un piano nella direzione dell'oggetto, utilizzato dentro al ciclo for di riga 11 (che non viene però mostrato) per sezionare la *mesh* che lo racchiude in due parti. Quest'ultima si ottiene mediante l'apposita funzione *GetComponent<MeshFilter>().mesh* alla quinta riga. Dunque, una volta ottenute le due parti *left_HalfMesh* e *right_HalfMesh* (vedi linea 15 e 18), la prima viene assegnata all'oggetto "victim" originale, che quindi diventa di fatto il frammento di sinistra rinominato appositamente come "left side" (righe 21 e 22), mentre la *mesh* di destra viene attribuita ad un nuovo oggetto *rightSideObj*, chiamato "right side", costruito nella stessa posizione e con la medesima rotazione e grandezza dell'oggetto originale (dalla riga 26 alla riga 35). Nelle ultime righe di codice in fine, si provvede ad assegnare i materiali ai due frammenti e a restituirli come risultato in un apposito vettore di oggetti: *return GameObject[]{ leftSideObj, rigthSideObj}*.

4.5 *MeshCut.Cut* modificato

Dalla Fig. 4.6, si può notare che in seguito al taglio, il *BoxCollider* (cubo tratteggiato in verde) della parte di sinistra rimane della dimensione dell'oggetto originale, anziché adattarsi alle nuove misure. Per ovviare al problema, è sufficiente allora distruggere e ricreare ogni volta il *collider* aggiungendo le seguenti linee di codice nella funzione Cut, analizzata nel paragrafo precedente:

"Destroy(leftSideObj.GetComponent<BoxCollider>());"

"leftSideObj.AddComponent<BoxCollider>();"

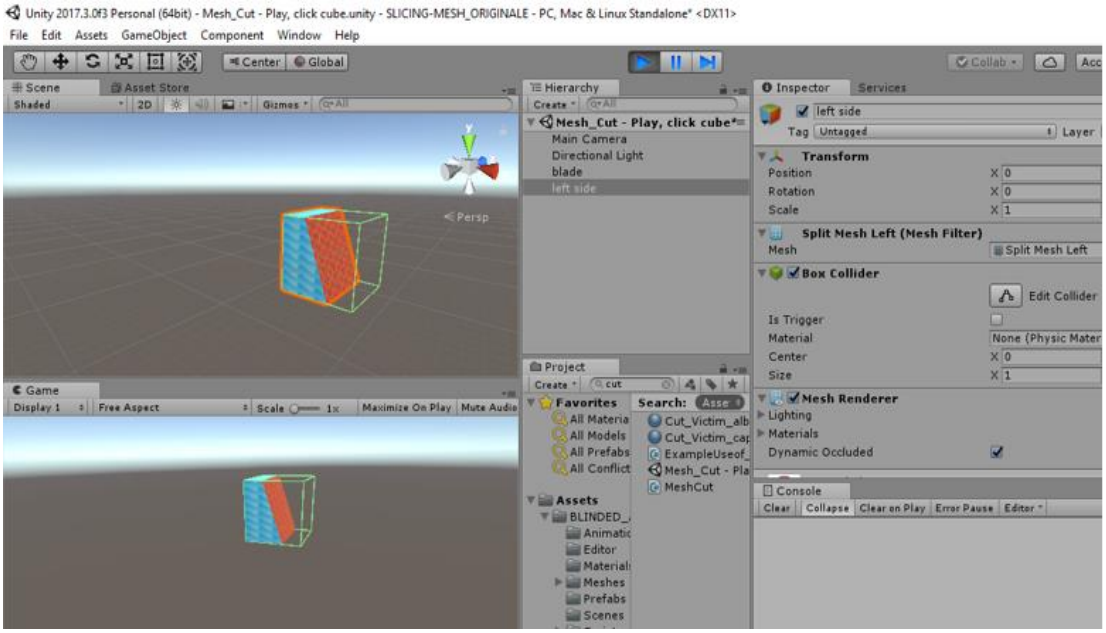

Figura 4.6: *BoxCollider* della parte sinistra non riadattato dopo il taglio.

Come si può notare in Fig.4.7, le cose sono notevolmente migliorate rispetto a prima, ma visto che il *collider* non segue perfettamente le forme dell'oggetto, persiste un problema: se si andasse a tagliare in un punto all'esterno del frammento "left side" ma interno al suo *BoxCollider*, il sistema riconoscerebbe valida l'operazione e creerebbe una sotto-parte "right side" di fatto vuota.

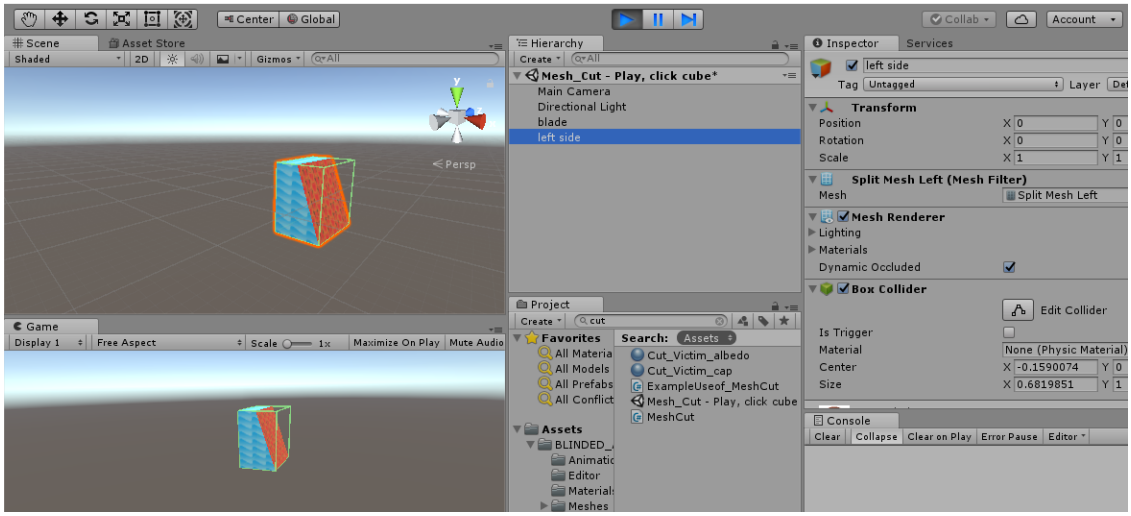

Figura 4.7: *BoxCollider* della parte sinistra riadattato dopo il taglio.

Per ovviare al problema, è sufficiente dunque al momento del taglio, distruggere qualunque tipo di collider preesistente (per evitare conflitti e sovrapposizione di più collider) e applicare un *MeshCollider*, che si adatta perfettamente alla sagoma dell'oggetto (Fig.4.8).

```
// assign the game objects

victim.name = "left side";
victim.GetComponent<MeshFilter>().mesh = left_HalfMesh;

GameObject leftSideObj = victim;

//se di default vi è uno dei seguenti collider, li distruggo
Destroy(leftSideObj.GetComponent<BoxCollider>());
Destroy(leftSideObj.GetComponent<SphereCollider>());
Destroy(leftSideObj.GetComponent<CapsuleCollider>());

//ogni volta che taglio la parte sinistra distruggo il vecchio MeshCollider (mesh dell'oggetto intero originale)
Destroy(leftSideObj.GetComponent<MeshCollider>());

//e riassegno il nuovo MeshCollider adattato perfettamente alla dimensione del nuovo frammento tagliato
leftSideObj.AddComponent<MeshCollider>();

GameObject rightSideObj = new GameObject("right side", typeof(MeshFilter), typeof(MeshRenderer));
rightSideObj.transform.position = victim.transform.position;
rightSideObj.transform.rotation = victim.transform.rotation;
rightSideObj.GetComponent<MeshFilter>().mesh = right_HalfMesh;

if(victim.transform.parent != null){
    rightSideObj.transform.parent = victim.transform.parent;
}

// in questo modo aggiungo un mesh collider anche all'oggetto rightSide ottenuto dal taglio in modo che possa anche lui esser tagliato
rightSideObj.AddComponent<MeshCollider>();
```

Figura 4.8: modifiche alla funzione Cut (C#).

Si è così riuscito ad ottenere il risultato desiderato. In Figura 4.9, vien mostrato l'esempio di due oggetti, in particolare un elicottero e un cubo, posti uno dietro l'altro, vittime di due tagli successivi ed infine manipolati.

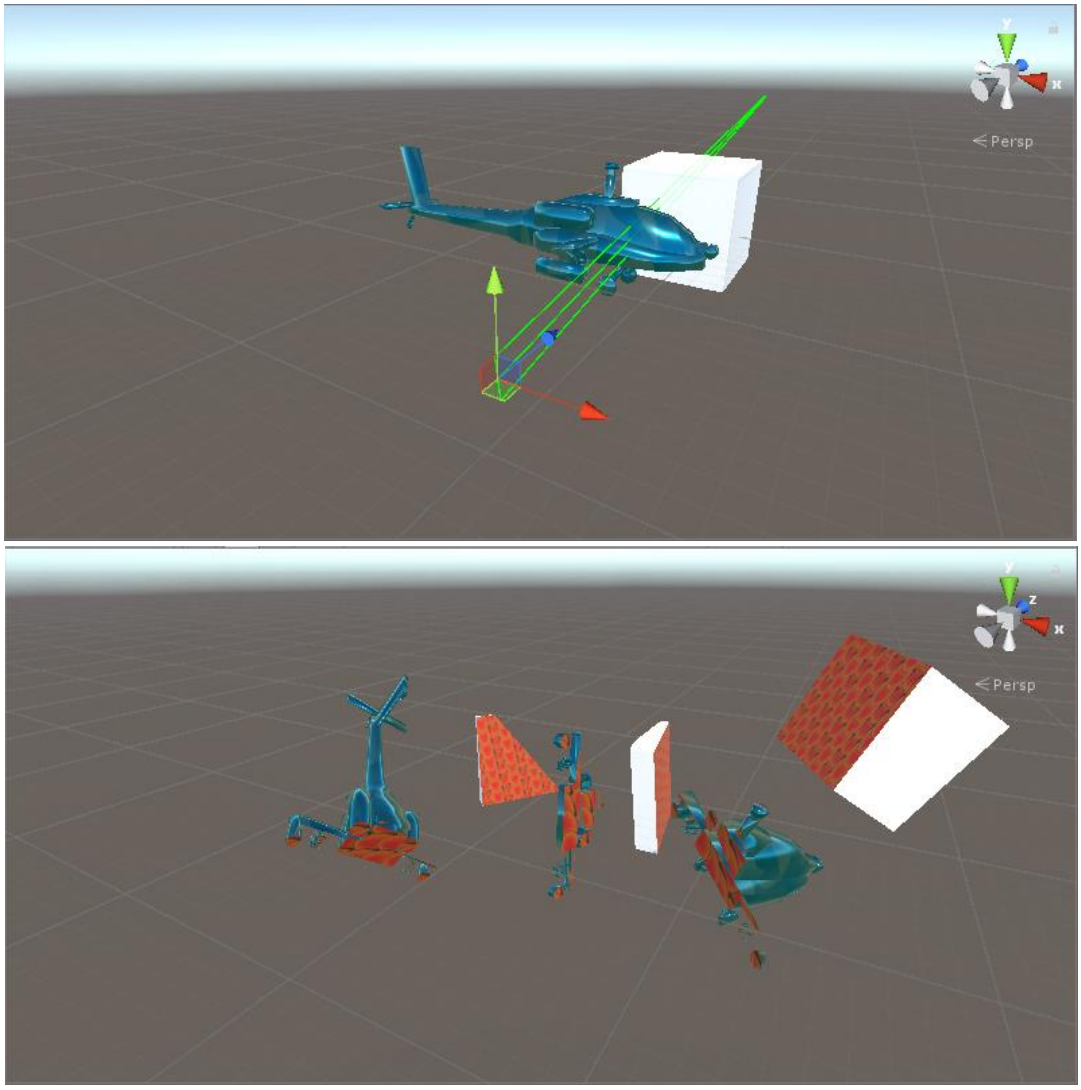

Figura 4.9: taglio e manipolazione di oggetti tridimensionali.

4.6 Operare su *mesh* complesse

Prima di proseguire nella realizzazione del progetto, è stato tuttavia necessario verificare l'efficienza dello script su *mesh* più complesse, dal momento che l'obiettivo finale era quello di sezionare ossa anche particolarmente articolate. Effettuando diversi test su oggetti caratterizzati da *mesh* molto elaborate, si è notato come previsto che la durata del taglio richiedeva qualche frazione di secondo in più (in casi critici anche qualche minuto). A tal scopo si è utilizzato InVesalius 3, di cui già si è parlato nel terzo capitolo,

per estrarre dalle radiografie, oggetti tridimensionali virtuali in formato *.OBJ* e *.STL* di varie parti del corpo e sottoporli al taglio (Fig.4.10).

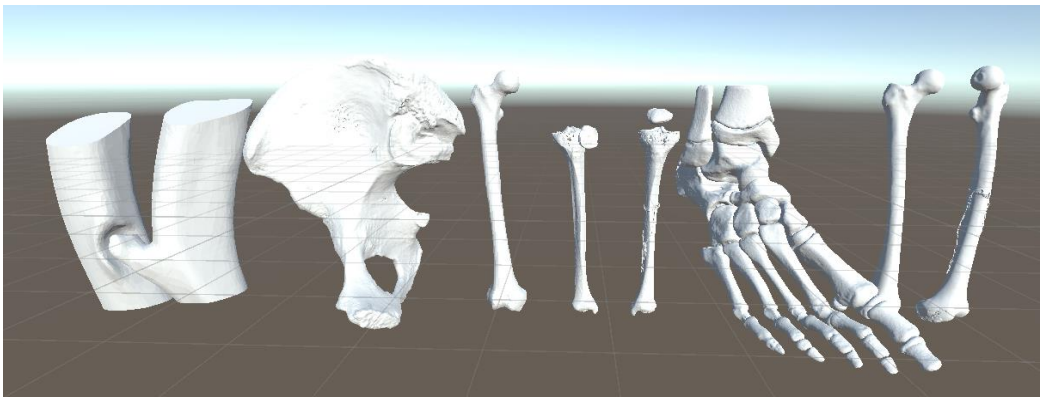

Figura 4.10: esostosi femorale, bacino, femore, tibie, piede, femori.

Dapprima è stato necessario utilizzare Blender per effettuare la mappatura UV sulle varie ossa, senza la quale non si sarebbe potuto effettuare alcuna operazione sulle *mesh*. Una volta eseguito correttamente l*'unwrapping* degli oggetti, questi sono stati importati nel progetto e sottoposti al taglio. I risultati dei test effettuati con processore Intel®Core™ i7-6500U 2.50 GHz 2.60 GHz, memoria Ram 8 GB, scheda video Intel ® HD Graphics 520, sistema operativo Windows 10 x64 bit, sono riportati nella sottostante tabella:

Oggetto (.obj)	**Vertici ("verts")**	**Facce ("tris")**	**Durata (secondi)**
Esostosi.obj	18634	11270	60
Bacino.obj	100643	198122	290
Femore.obj	4470	6866	12
Tibie.obj	6443	8973	92
Piede.obj	165969	331938	535
Femori.obj	7436	10993	101

Tabella: Risultati dei test di taglio.

Dai test riportati in tabella, risulta chiaro che gli oggetti .obj generati con InVesalius 3, non possono essere sottoposti al taglio così come sono, nemmeno nella migliore delle ipotesi: 12 secondi per sezionare un osso in tempo reale, non costituiscono un lasso di tempo molto ragionevole, figurarsi 535. Prima di importarli nel progetto *Cut Project* e sottoporli allo script dunque, occorre un adeguato *remeshing*, sfruttando MeshLab come segue: import mesh-> filters-> remeshing, simplification and reduction-> Simplification: Quadric edge collapse decimation. Senza andare a danneggiare

eccessivamente l'aspetto fisico delle ossa e trovando il giusto compromesso, si è notato che riducendo il numero di facce poligonali al di sotto di 4500, con una quantità di vertici pari a qualche decina di migliaia, si può ottenere un risultato molto soddisfacente sia in termini di qualità dell'oggetto che di tempistiche di taglio. Nella figura sottostante è riportato l'esempio dell'oggetto "Piede.obj", opportunamente ricostruito con mappatura UV, prima e dopo il *remeshing*. Come si può notare, il reticolo è molto meno fitto dopo l'operazione, il numero di triangoli e vertici è molto ridotto, ma il piede conserva comunque tutte le sue principali caratteristiche consentendo di essere tagliato in meno di 4 secondi, anziché in 9 minuti.

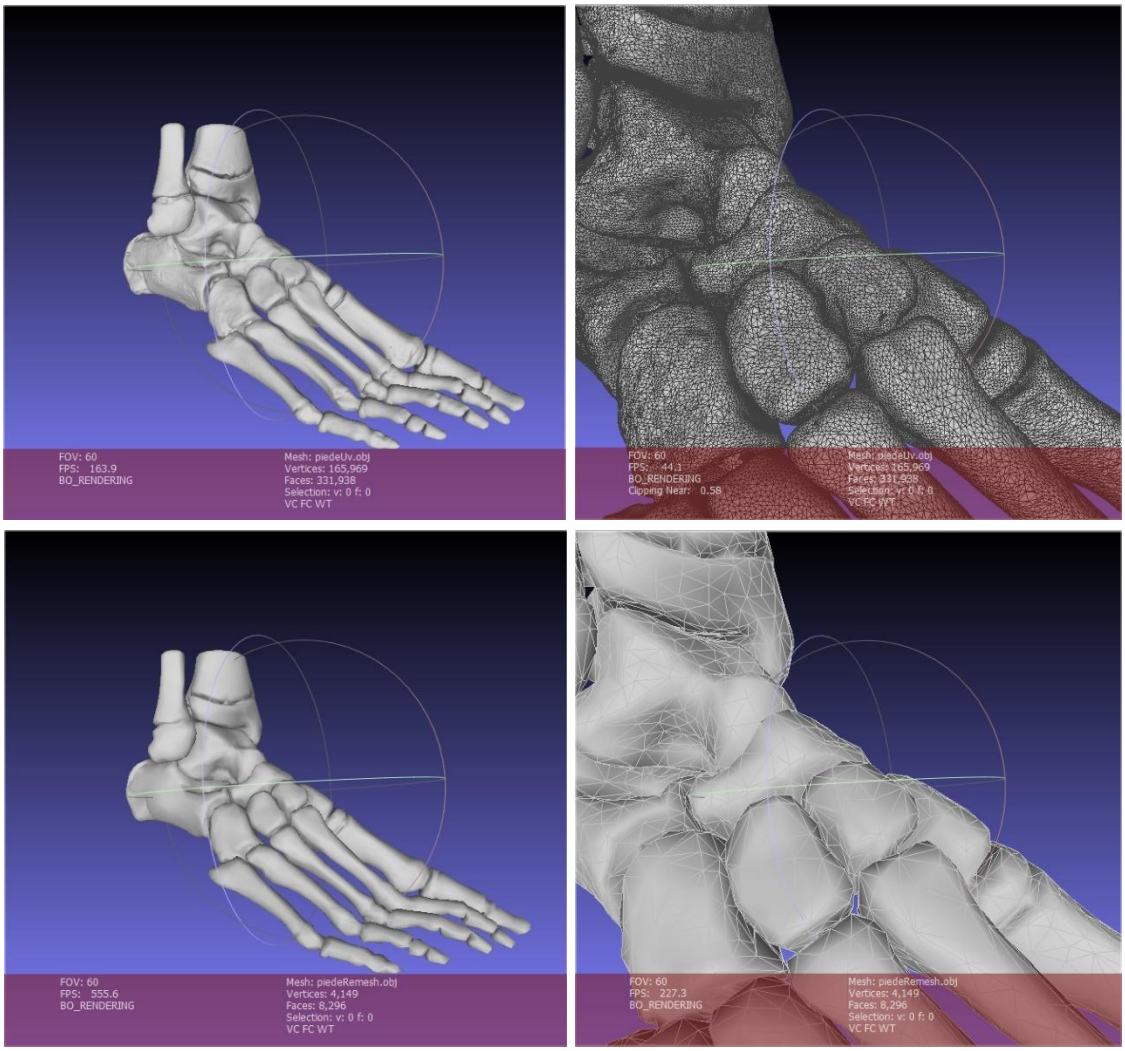

Figura 4.11: Piede.obj prima e dopo il *remeshing.*

Con quest'ultima operazione di *remeshing* si è concluso lo sviluppo di *Mesh Cut Project,* che è stato realizzato a partire da un progetto già esistente che ha subito tutte le modifiche illustrate nel capitolo, e che costituisce la prima fase dell'elaborato finale.

Capitolo 5

Tracking Project: secondo step

Si vuole presentare in questo capitolo il secondo importante step per la realizzazione dell'applicazione finale MRTC, vale a dire *Tracking Project*, utilizzato per associare ad un qualsiasi elemento fisico la corrispondente immagine digitale.

5.1 Specifiche del progetto

Ancora una volta si tratta di un progetto Unity, nato per soddisfare la richiesta di medici e chirurgi del Rizzoli di sovrapporre a specifiche parti del corpo dei pazienti, le corrispondenti ossa virtuali, "ancorandole" ad esse. Dunque, ricordando quanto detto nei capitoli introduttivi e in particolare nel paragrafo *1.4 i tre mondi a confronto,* questo progetto rappresenta il primo gradino verso la *Mixed Reality*, che non solo aggiunge oggetti virtuali nel mondo, *"ma li ancora alla realtà, consentendo l'interazione con questi ultimi come se fossero veri".* Tuttavia, sebbene molto vicini al mondo della MR, ci troviamo ancora nell'ambito della Realtà Aumentata. Qui si aggiungerà semplicemente un tassello al puzzle garantendo il *tracking* di oggetti tridimensionali e la manipolazione di questi ultimi, ma attraverso una telecamera, un monitor e un mouse, senza essere noi stessi ad interagire con essi. Per fare il passo successivo infatti, sarà necessario disporre di un opportuno strumento che lo permetta, come un visore HoloLens (Fig. 5.1).

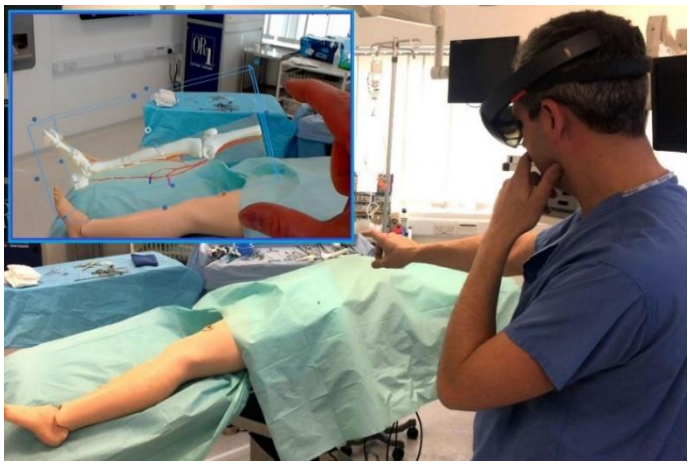

Figura 5.1: *tracking* di una gamba e interazione con l'oggetto grazie ad HoloLens.

5.2 Strumenti utilizzati

Per la realizzazione del *Trackig Porject*, sono stati impiegate praticamente tutte le tecnologie introdotte e spiegate nel terzo capitolo eccetto il Leap Motion.
Per prima cosa sono state stampate utilizzando MarkForged Mark Two due parti ossee, (vedi Fig. 3.2) prese da delle radiografie ed elaborati grazie ad InVesalius 3: un piede ed un'esostosi femorale, rispettivamente in Nylon di colore nero opaco e bianco riflettente, per stabilire quale venisse rilevato meglio.
I corrispondenti oggetti virtuali, opportunamente modificati sfruttando Blender, per l'aggiunta di coordinate UV e MeshLab, per effettuare il *remeshing*, sono stati importati poi nel *Model Target Generator*, per ottenere i rispettivi *Model Targets* in appositi database in formato *.unitypackage*. Come si può notare in Figura 5.2, prima della creazione del modello è necessario aprire un nuovo progetto caricandovi l'oggetto *.OBJ* desiderato.

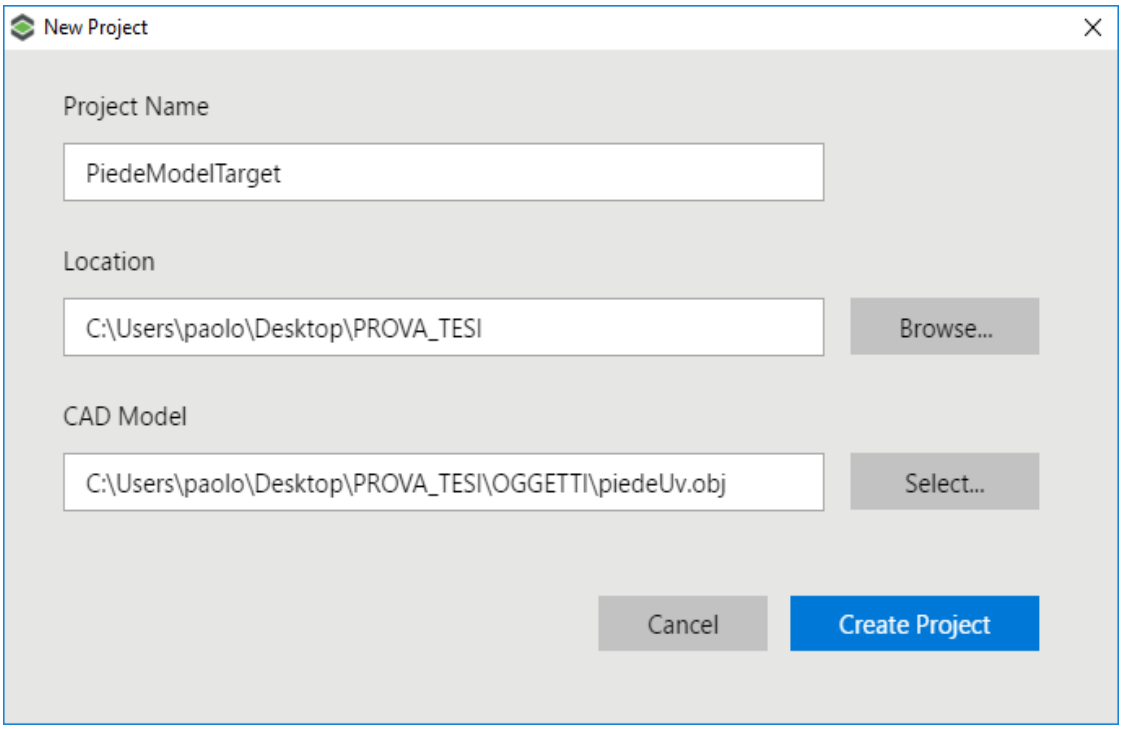

Figura 5.2: Schermata di creazione di un progetto *Model Targets*.

Fatto ciò, è necessario fare una sorta di foto dell'oggetto dalla prospettiva desiderata, premendo l'apposito pulsante "Set Detection Position", in modo da delinearne il contorno, che verrà poi utilizzato per eseguire il *tracking* (Fig. 5.3).

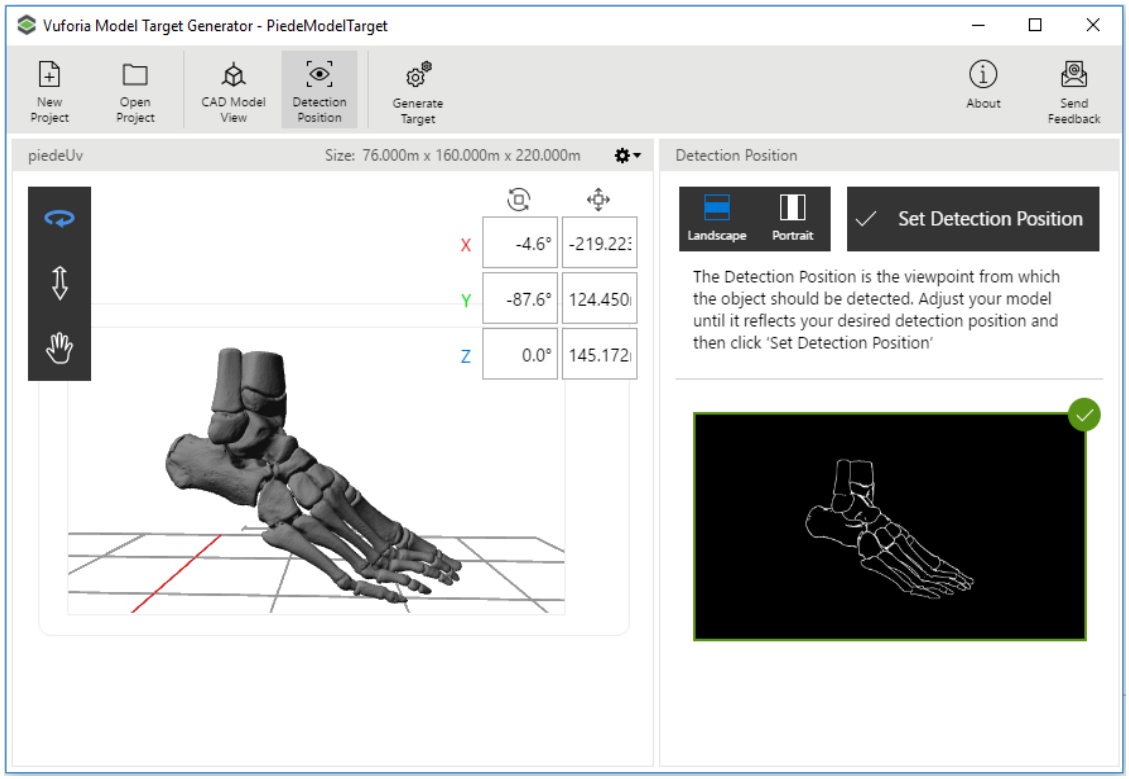

Figura 5.3: Generazione di un *Model Target*.

Importando quindi i database dei *Model Targets* in Unity e scaricando la libreria di Vuforia, è stato relativamente semplice ottenere il risultato sperato.

In fine, è stato necessario l'utilizzo di un ultimo strumento, vale a dire una telecamera esterna con risoluzione adeguata a riconoscere i dettagli degli oggetti inquadrati per poi sovrapporgli il modello virtuale (Fig. 5.4)

5.3 Osservazioni

Come si può notare in Fig. 5.4, il *tracking* degli oggetti stampati avviene con successo, tuttavia risulta più difficoltoso con l'esostosi femorale piuttosto che col piede in quanto il secondo è meno dettagliato rispetto al primo. Si è notato inoltre che la telecamera fa più fatica a mettere a fuoco e riconoscere un elemento bianco riflettente, piuttosto che uno scuro non riflettente. Al fine di garantire una migliore esperienza di *tracking*, si consiglia pertanto di utilizzare un materiale opaco per la stampa 3D degli oggetti.

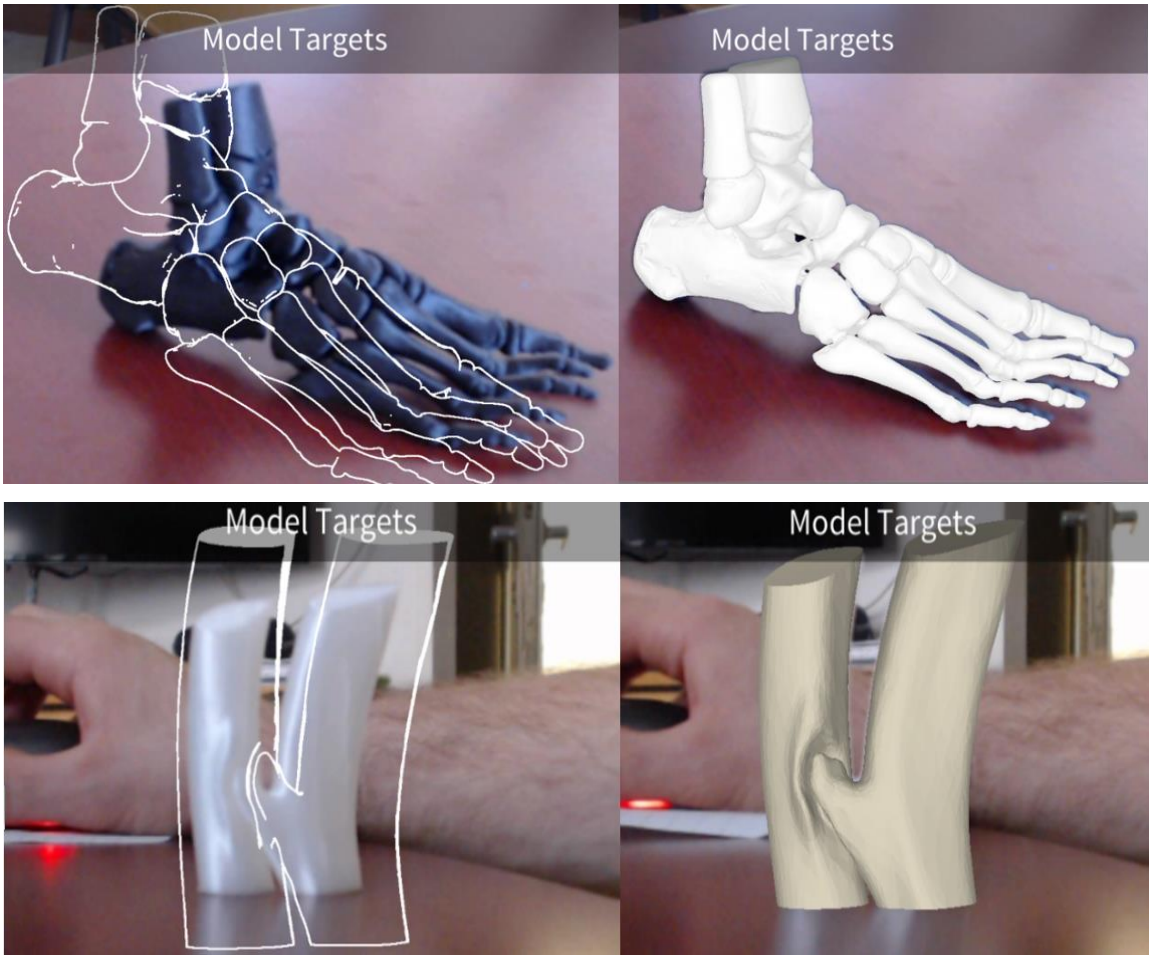

Figura 5.4: *Tracking* degli oggetti stampati.

5.4 Tracking e taglio di Model Target

Una volta messo a punto il tracking dei modelli stampati, sono stati importati nel progetto i soli script necessari al taglio e alla manipolazione di questi ultimi. Si è deciso poi di associare al piano di taglio un oggetto bisturi, come mostrato nella figura sottostante.

Figura 5.5: Tre foto in sequenza: *tracking*, taglio e manipolazione dei frammenti.

In questo capitolo sono state descritte quindi le fasi di sviluppo di *Tracking Project*, per consentire di acquisire oggetti reali ed associargli i corrispondenti modelli virtuali. È stato inoltre importata nel progetto anche la funzionalità di taglio di *Cut Project* al fine di consentire di sezionare i *Model Target* generati.

Capitolo 6

Mixed Reality Project: terzo step

In questo capitolo viene descritto il progetto *Mixed Reality*, ultimo passo per la realizzazione di MRTC. Si parlerà in particolare di Leap Motion e di come sfruttarlo per garantire l'interazione con oggetti tridimensionali virtuali per avvicinarsi poi gradualmente all'ambito della *Mixed Reality*.

6.1 Specifiche del progetto

Viene qui mostrato l'ultimo step per addentrarsi realmente nell'ambito della *Mixed Reality*, grazie all'utilizzo di uno strumento di cui già si è parlato nel terzo capitolo: il Leap Motion. Quest'ultimo infatti consente finalmente di interagire semplicemente con le proprie mani, senza cioè dover utilizzare un mouse o una tastiera, con i vari oggetti virtuali, prendendoli, spostandoli, e come si vedrà, persino tagliandoli con bisturi virtuali. È stato deciso di sfruttare questo piccolo ma potente dispositivo, al fine di fornire in tempi brevi un'applicazione funzionante da far provare ai medici del Rizzoli, per mostrare loro quello che di fatto potrebbe essere il prodotto finito una volta importato su Microsoft HoloLens. Quest'ultimo potrà così garantire un'esperienza *Mixed Reality* a 360 gradi sfruttando il progetto MRTC.

6.2 Primi passi con Leap Motion

Per prima cosa è stato necessario configurare correttamente il dispositivo ed installare nel progetto Unity *l'Asset Core per Leap Motion Orion Beta*. Fatto ciò, dopo aver preso un po' di dimestichezza con la nuova tecnologia, è stato necessario importare nel progetto un'estensione della libreria sopra citata al fine di garantire l'interazione tra mani ed oggetti virtuali: il modulo *Leap Motion Interaction Engine (1.2.0)*. Quindi è stato introdotto nella Scena Unity il Leap Motion Controller (Fig. 6.1), oggetto virtuale presente nella libreria sopra citata, tramite cui avviene il *tracking* delle mani vero e proprio sfruttando la periferica USB fisica. Collocato sopra ad esso, l'Interaction Manager garantisce l'interazione tra mani e oggetti virtuali, consentendo quindi di afferrarli e manipolarli, sfruttando opportuni script preimpostati come *InteractionHand.cs*. Al fine di visualizzare a video i propri arti, sostituiti da mani artificiali, vengono collocati in scena anche i "Capsule Hand" sinistro e destro.

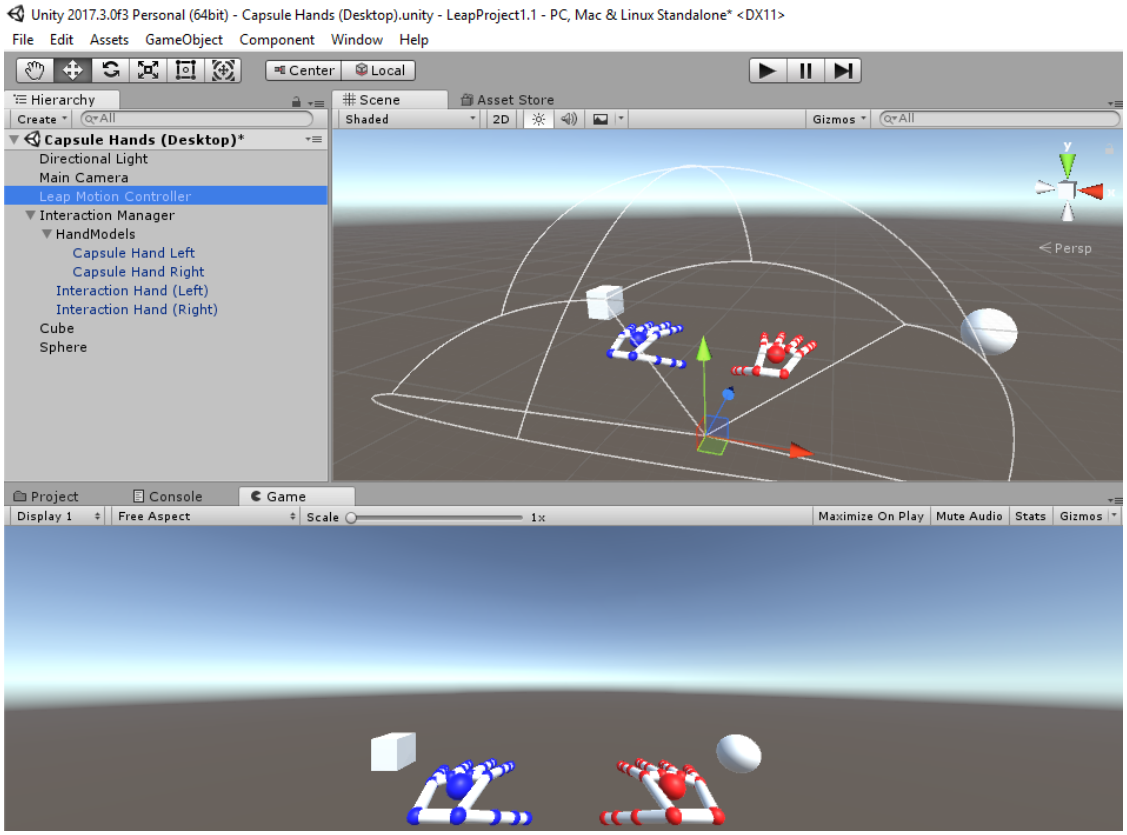

Figura 6.1: Leap Motion in Unity.

In Figura 6.1, si possono notare alla destra e alla sinistra delle mani simulate, rispettivamente una sfera ed un cubo. Per far sì che vi sia interazione con questi ultimi è

necessario un ulteriore passaggio, ovvero devono disporre di un componente *RigidBody*, di un opportuno *collider*, e dello script *InteractionBehaviour.cs* (nella figura sottostante viene mostrato l'esempio delle caratteristiche di un cubo interattivo).

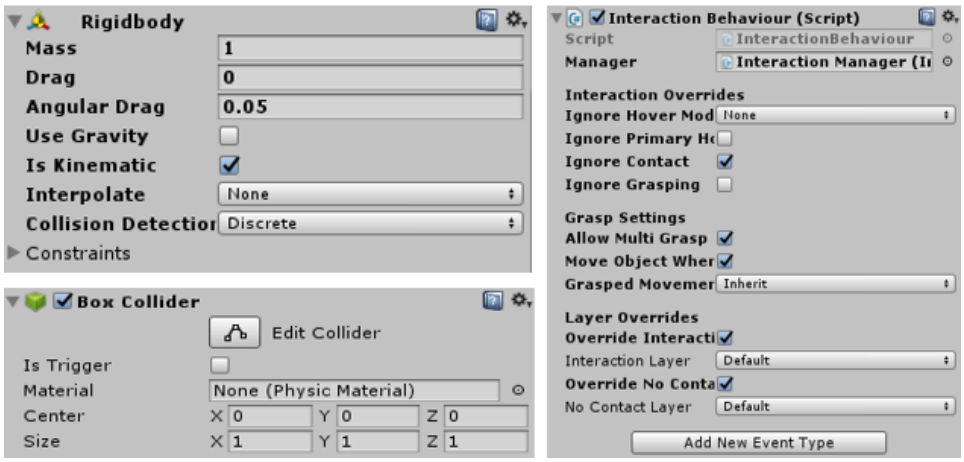

Figura 6.2: Componenti necessari per l'interazione col cubo.

6.3 Interazione con gli oggetti

Avviando il progetto con l'apposito pulsante *Play* di Unity, che consente di vedere nel pannello *Game* un'anteprima dell'applicazione in esecuzione, si può notare sovrapponendo al Leap Motion le proprie mani in carne ed ossa, la comparsa in scena delle corrispondenti "Capsule Hand" digitali. Inoltre, selezionando l'Interaction Manager, viene mostrato nel pannello *Inspector* se il *tracking* è andato a buon fine oppure no (Fig. 6.3).

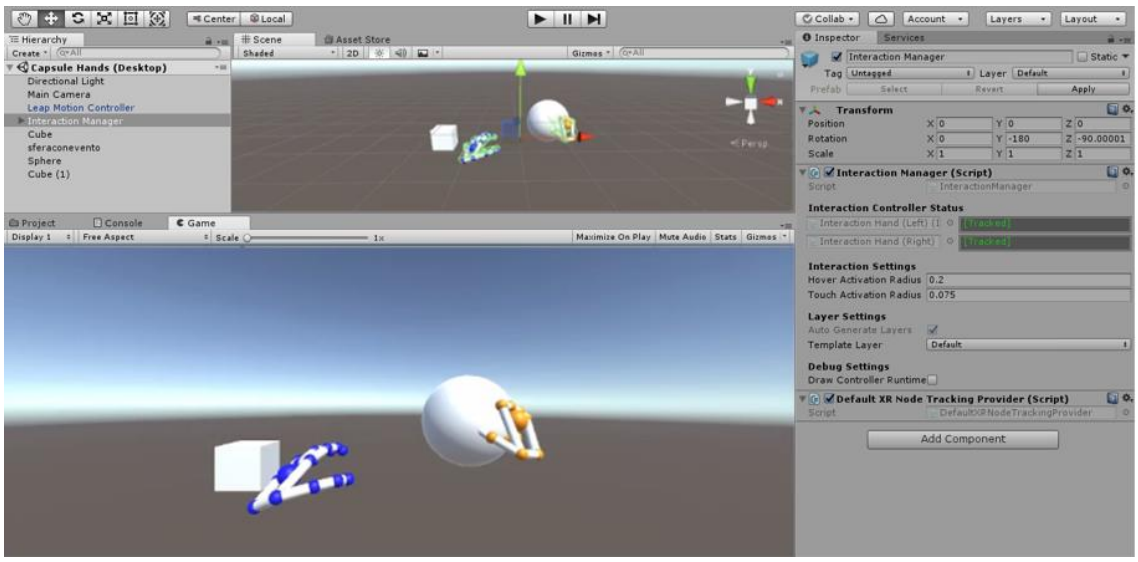

Figura 6.3: tracking delle mani e interazione con gli oggetti.

Si può scegliere poi a seconda dei casi di utilizzo, se manipolare oggetti che rimangano sospesi nel vuoto come nel caso sopra, di tipo “Kinematic”, o che cadano per effetto di gravita, “Use Gravity”. Per quanto riguarda *l’InteractionBehaviour.cs* (vedi Fig. 6.2), si può specificare su ognuna delle due mani, se consentire ad esempio di prendere un oggetto ma ignorare il semplice contatto con esso (spuntando l’apposita casella “ignore contact), per evitare che venga spostato via dalla scena erroneamente.
Quest’ultimo comportamento è proprio quello che servirà per consentire ai chirurgi del Rizzoli di prendere il bisturi virtuale di tipo “Kinematic” senza problemi.

In questo capitolo dunque è stato presentato nel dettaglio il *Mixed Reality Project*, e i vari passi per utilizzare il Leap Motion e garantire l’interazione con elementi digitali. Una volta verificato il funzionamento del progetto nel mondo virtuale, quest’ultimo è stato importato nel *Tracking Project*, risolvendo tutti gli eventuali problemi di compatibilità, per spostarsi finalmente in ambito *Mixed Reality* ed iniziare a sviluppare l’MRTC *Project* (vedi il prossimo capitolo).

Capitolo 7

MRTC Project

In questo capitolo viene finalmente presentato nel dettaglio MRTC, *"Mixed Reality Tracking Cut"*, ossia l'elaborato finale della tesi, così nominato appunto perché racchiude in sé i tre progetti di cui si è discusso finora.

7.1 *Merge* dei progetti

MRTC, nasce come "*merge"* (ovvero costruzione di un unico file, a partire da file separati) dei progetti *Cut Project*, *Tracking Project e Mixed Reality Project* (Figura 7.1).

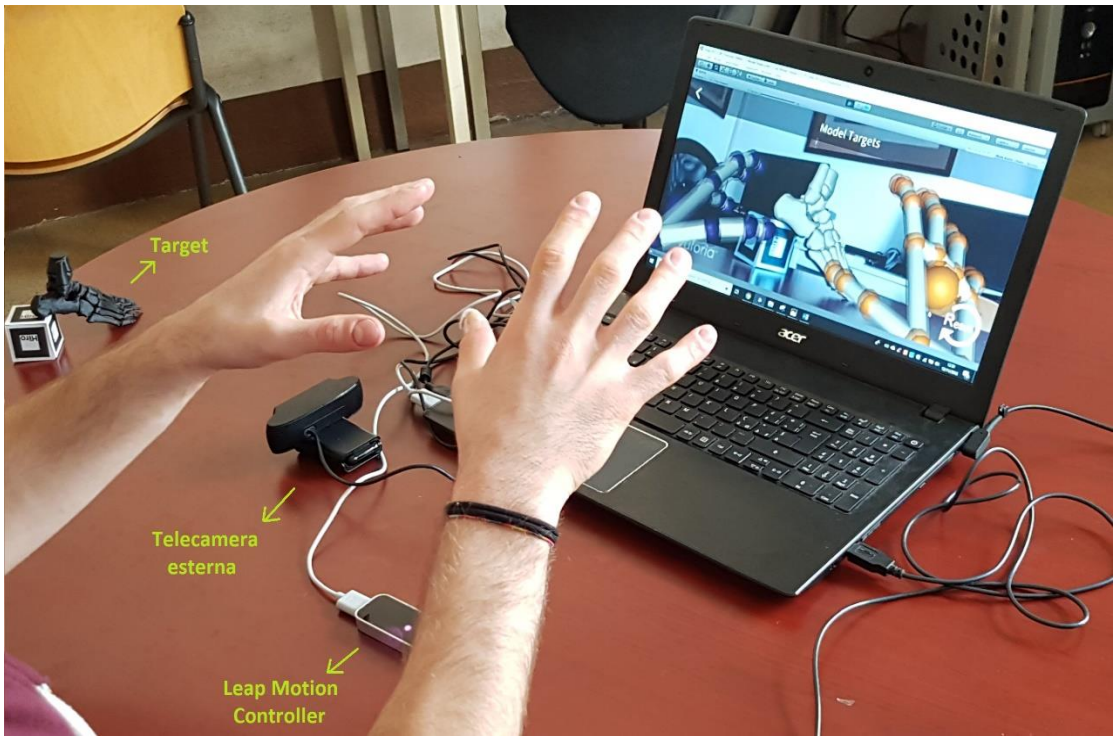

Fig. 7.1: tracking delle mani (Leap Motion) e dell'oggetto bersaglio (Vuforia).

Naturalmente una volta effettuato il *merge* dei progetti, è stato necessario andare ad apporre diverse modifiche dal momento che a muovere il bisturi virtuale non sarebbe più stato il mouse bensì le mani. Anche per effettuare il taglio vero e proprio bisognava

pensare ad un input diverso dal click del mouse o dalla pressione di un tasto sulla tastiera. Tutto questo perché come già accennato, l'obiettivo è quello di fornire un'applicazione funzionante da far provare ai medici del Rizzoli, per mostrare loro quello che di fatto potrebbe essere il prodotto finito una volta importato su Microsoft HoloLens.

7.2 Interazione coi Model Target

In primo luogo, è stato aggiunto lo script *InteractionBehaviour.cs* ai *Model Target* (e al bisturi) per garantire l'interazione con questi ultimi. In secondo luogo, dal momento che gli oggetti sono avvolti non da semplici *collider* ma da *Mesh Collider*, è stato necessario inoltre settarli come "Convex" per consentire la collisione e l'interazione con altri *Mesh Collider* (come ad esempio quelli che avvolgono le mani virtuali).
L'integrazione del Leap Motion con il *tracking* di oggetti 3D stampati, quali il piede e l'esostosi femorale, è stata quindi quasi immediata (Fig. 7.2). Lo stesso non si può dire del taglio.

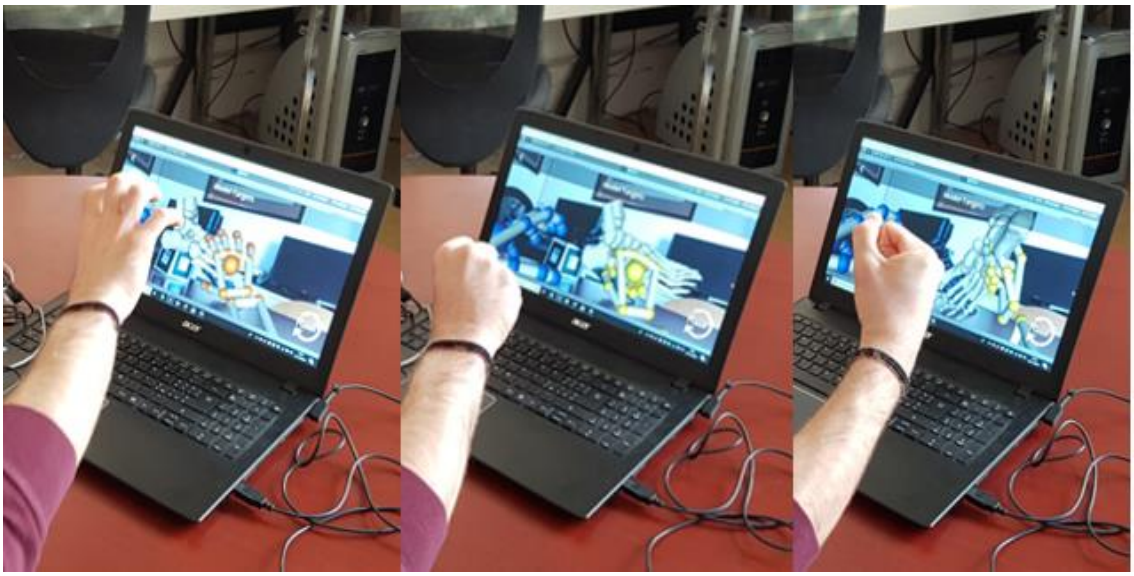

Fig. 7.2: interazione con l'oggetto virtuale generato dal *tracking* del piede.

7.3 Modifica degli script di taglio

Per prima cosa è stato necessario aggiungere alcune righe di codice allo script *MeshCut.cs* (vedi paragrafo 4.2): alle righe 4 e 12 del codice sotto riportato, si può notare l'aggiunta dello script *InteractionBehaviour.cs* per conferire il giusto comportamento ai frammenti *leftSideObj* e *rightSideObj* degli oggetti tagliati. Per il medesimo scopo è stato necessario specificare alla linea 5 e 13 un *Mesh Collider* di tipo Convex e aggiungere un componente *RigidBody* (riga 7 e 15) per entrambe le sotto-parti. Queste ultime poi (linea 8-9, 16-17) sono state rese "Kinematic" per evitare di essere scaraventate via al semplice contatto, e immuni all'effetto di gravità.

```
1 //NB: per implementare le funzionalità per utilizzare LeapMotion:
2 //************************************************************
3 //leftSide
4 leftSideObj.AddComponent<InteractionBehaviour>();
5 leftSideObj.GetComponent<MeshCollider>().convex = true; //L'interazione con i mesh collider richiede che sia Convex
6 if (!leftSideObj.GetComponent<Rigidbody>())
7     leftSideObj.AddComponent<Rigidbody>();
8 leftSideObj.GetComponent<Rigidbody>().useGravity = false;
9 leftSideObj.GetComponent<Rigidbody>().isKinematic = true; //non viene scaraventato via al contatto.
10
11 //rightSide
12 rightSideObj.AddComponent<InteractionBehaviour>();
13 rightSideObj.GetComponent<MeshCollider>().convex = true; //L'interazione con i mesh collider richiede che sia Convex
14 if (!rightSideObj.GetComponent<Rigidbody>())
15     rightSideObj.AddComponent<Rigidbody>();
16 rightSideObj.GetComponent<Rigidbody>().useGravity = false;
17 rightSideObj.GetComponent<Rigidbody>().isKinematic = true; //non viene scaraventato via al contatto.
18
19
20 //************************************************************
```

Fig. 7.3: linee di codice aggiunte allo script *MeshCut.cs.*

Oltre all'aggiunta delle linee di codice sopra mostrate, è stato necessario cambiare il funzionamento intrinseco dello script, effettuando diverse modifiche.

Se si prende in considerazione lo script originale (Fig. 4.5, capitolo quarto), si può notare che al momento del taglio, l'oggetto vittima "victim", non viene distrutto, ma gli viene semplicemente assegnata una nuova mesh rielaborata "left_HalfMesh" che rappresenta la parte sinistra. In altre parole, "victim" diventa il frammento di sinistra, mentre la parte destra vien creata da zero con la relativa "right_HalfMesh".

Lasciando le cose così come sono, al momento del taglio, l'Interaction Manager consente al limite di interagire con il frammento di destra, lanciando in continuazione errori a tempo d'esecuzione, quando si tenta di carpire quello di sinistra. Tutto questo perché all'oggetto "victim", che non viene mai distrutto, viene assegnata di colpo una

nuova mesh ridotta, non consentendo più all'Interaction Manager di riconoscerlo correttamente e garantire l'interazione.

Per risolvere il problema dunque è stato sufficiente far in modo di creare da zero al momento del taglio anche il frammento di sinistra in maniera analoga a quello di destra (vedi dalla riga 19 in poi del codice riportato sotto), cancellando tutte le linee di codice superflue (vedi Fig. 7.4, dove sono state commentate tutte le righe da eliminare), e distruggendo l'oggetto "victim" originale (Fig. 7.5).

```
1  // assign the game objects
2
3  //victim.name = "left side";
4  //victim.GetComponent<MeshFilter>().mesh = left_HalfMesh;
5
6  //GameObject leftSideObj = victim;
7
8  //se di default vi è uno dei seguenti collider, li distruggo
9  / Destroy(leftSideObj.GetComponent<BoxCollider>());
10 //Destroy(leftSideObj.GetComponent<SphereCollider>());
11 //Destroy(leftSideObj.GetComponent<CapsuleCollider>());
12
13 //ogni volta che taglio la parte sinistra distruggo il vecchio MeshCollider (mesh dell'oggetto intero originale)
14 //Destroy(leftSideObj.GetComponent<MeshCollider>());
15
16 //e riassegno il nuovo MeshCollider adattato perfettamente alla dimensione del nuovo frammento tagliato
17 //leftSideObj.AddComponent<MeshCollider>();
18
19 GameObject leftSideObj = new GameObject("left side", typeof(MeshFilter), typeof(MeshRenderer));
20 leftSideObj.transform.position = victim.transform.position;
21 leftSideObj.transform.rotation = victim.transform.rotation;
22 leftSideObj.GetComponent<MeshFilter>().mesh = left_HalfMesh;
23
24 leftSideObj.GetComponent<MeshRenderer>().materials = mats;
25 leftSideObj.transform.localScale = victim.transform.localScale;
```

Figura 7.4: modifiche apportate alla funzione Cut di *MeshCut.cs* (C#).

Importanti modifiche, anche per quanto riguarda l'input per il taglio vero e proprio, sono state apportate anche allo script *ExampleUseof_MeshCut.cs* (vedi paragrafo 4.2) e riportate in Figura 7.5 nella pagina successiva.

Si è deciso innanzitutto di definire una variabile "taglia" (riga 5), pubblica e di tipo boolean (ovvero una variabile accessibile anche dall'esterno e settabile con valore "vero" o "falso"), che catturasse l'evento del taglio in qualunque maniera esso avvenisse. Al completamento dell'azione questa viene resettata al valore "falso" (riga 23). Oltre a questo, le altre righe di codice rimangono invariate eccetto quelle dalla 23 alla 25 inerenti all'oggetto "Scalpel", che sarebbe il bisturi, per far sì che dopo il taglio esso venga ricollocato nella posizione iniziale (subito a fianco alle mani).

Questa funzionalità in realtà è stata aggiunta per ultima per migliorare l'applicazione ed avere il coltello sempre nella posizione più idonea dopo ogni taglio.

```
public class ExampleUseof_MeshCut : MonoBehaviour {

    public Material capMaterial;
    public float bladeLength;
    public bool taglia = false;

    void Update(){
        if (taglia){ //non utilizzo più l'input del mouse

            RaycastHit[] hits;
            hits = Physics.RaycastAll(transform.position, transform.forward, 100.0F);
            if (hits.Length>0  ){
                for (int i = 0; i < hits.Length; i++)
                {
                    RaycastHit hit = hits[i];
                    if (bladeLength>0 && hit.distance < bladeLength || bladeLength<=0 )
                    {
                        GameObject victim = hit.collider.gameObject;
                        BLINDED_AM_ME.MeshCut.Cut(victim, transform.position, transform.right, capMaterial);

                        if (victim != null) //
                        {
                            GameObject Scalpel = GameObject.Find("Scalpel"); //trova il bisturi nella scena
                            Scalpel.transform.position = new Vector3(0.07f, -0.03f, 0.06f);//lo rimette nella posizione iniziale
                            Scalpel.transform.rotation = Quaternion.Euler(-6, 21.8f, -92f);//con la medesima rotazione iniziale
                        }
                        Destroy(victim); //dopo aver ottenuto il frammento di destra e quello di sinistra,
                                         //distruggo l'oggetto originale
                    }
                }
            }
        }
        taglia = false;
    }
```

Fig. 7.5: Modifiche a *ExampleUseof_MeshCut.cs* (C#).

7.4 Extended Finger Detector

Viene qui introdotto *l'ExtendedFingerDetector.cs,* altro script presente nel *Leap Motion Interaction Engine (1.2.0),* utilizzato allo scopo di poter associare alla posizione delle singole dita di una mano, una qualche azione. In particolare, si è deciso di monitorare solamente l'arto sinistro (Fig. 7.6), associando alla chiusura della mano la volontà di tagliare un qualche oggetto.

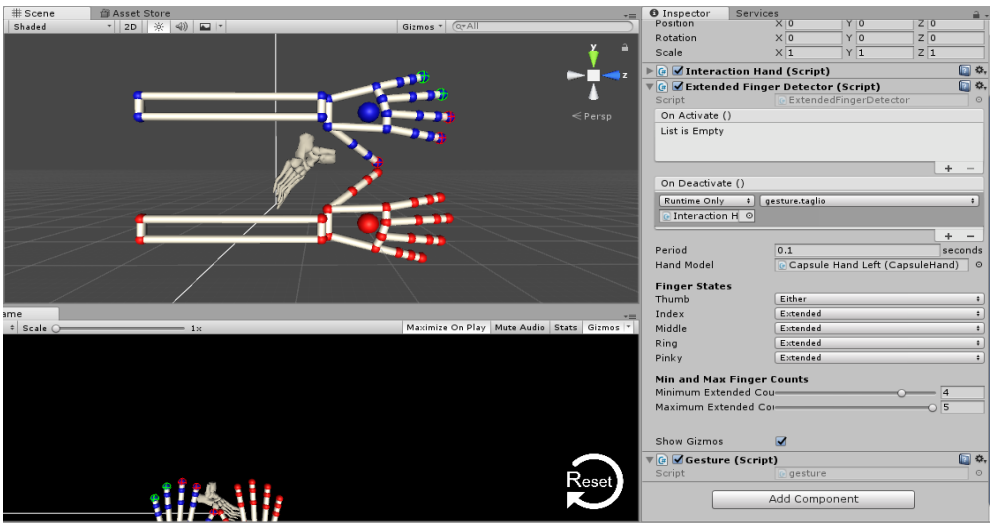

Figura 7.6: *ExtendedFingerDetector.cs* associato alla mano sinistra.

Questo rappresenta di fatto l'ultimo step per poter interagire con gli oggetti, sezionarli e manipolarli con le proprie dita, senza dover ricorrere a mouse o tastiera.
È stato quindi sufficiente creare un semplice script *gesture.cs* (sotto riportato) caratterizzato da un'unica funzione "taglio", che venisse richiamata ogni qualvolta il Leap Motion si accorgesse che la mano sinistra si fosse chiusa a pugno.

```
public class gesture : MonoBehaviour {
    public void taglio()
    {
        print("chiudo->taglio");
        GameObject Blade = GameObject.Find("blade"); //trovo il bisturi nella scena
        ExampleUseof_MeshCut script = Blade.GetComponent<ExampleUseof_MeshCut>();
        script.taglia = true; // setto a "vero" la variabile taglie dello script "ExampleUseof_MeshCut.cs"
                              //in modo tale da effettuare il taglio.

    }
}
```

Figura 7.7: *gesture.cs* (C#).

7.5 Il piano di taglio

Giunti a questo punto, tutti i progetti sono stati integrati perfettamente. L'unica cosa che manca, è un mezzo efficace per consentire all'utente di capire quale sia esattamente il piano di taglio del bisturi, che senza alcuna indicazione potrebbe risultare ambiguo (Fig. 7.8).

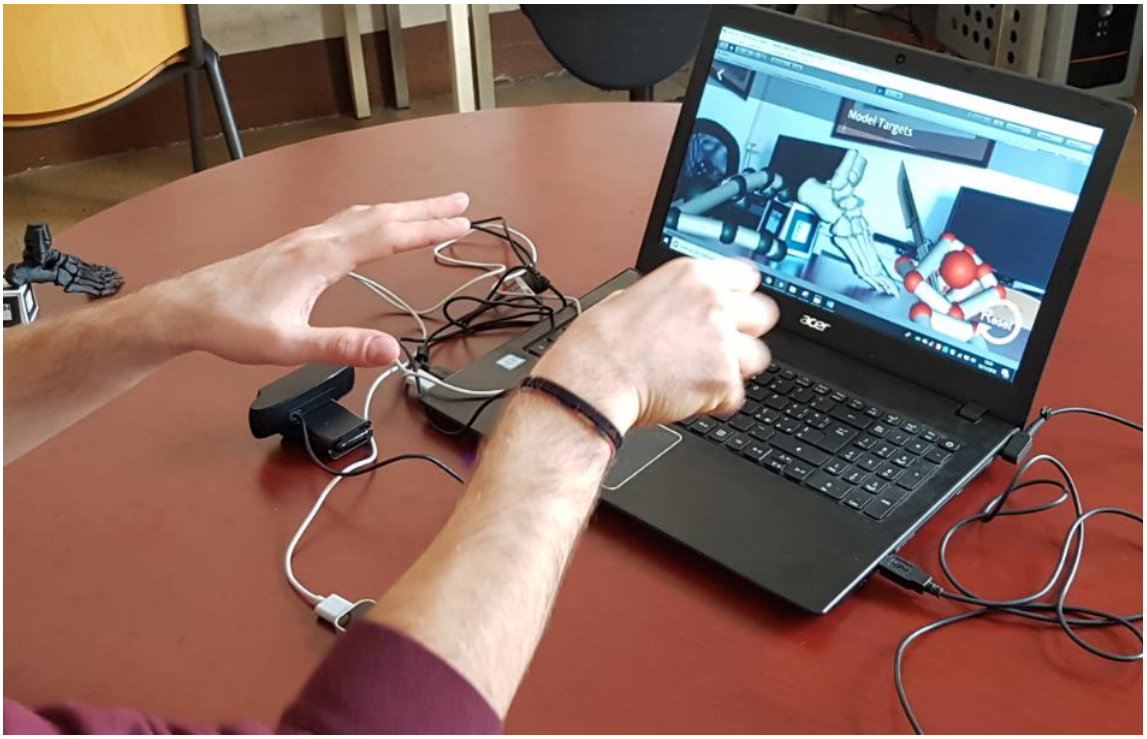

Figura 7.8: taglio con bisturi.

Si è deciso a tal scopo di applicare al coltello una superficie della medesima lunghezza. Tuttavia, un piano qualunque opaco avrebbe nascosto parte dell'oggetto ed uno trasparente non sarebbe servito a nulla. Ancora una volta il pacchetto *BLINDED_AM_ME*, si è rivelato di estrema utilità: è stato sufficiente esportare da esso lo *shader* "Intersection_Color", modificarne alcuni parametri ed applicarlo in fine al piano. Il risultato finale ottenuto è stato proprio quello desiderato (vedi Fig. 7.10 e 7.11).

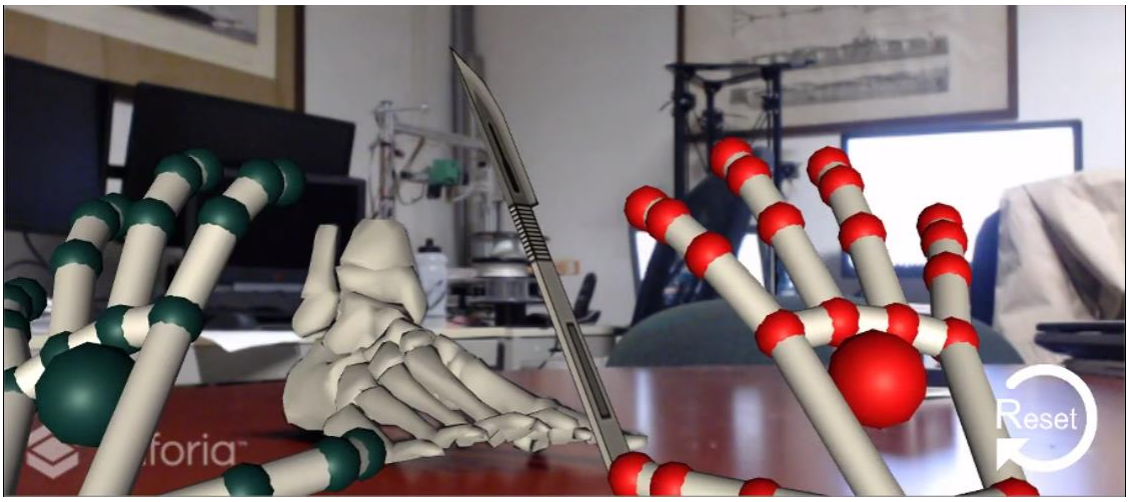

Figura 7.9: *Tracking* del Model Target.

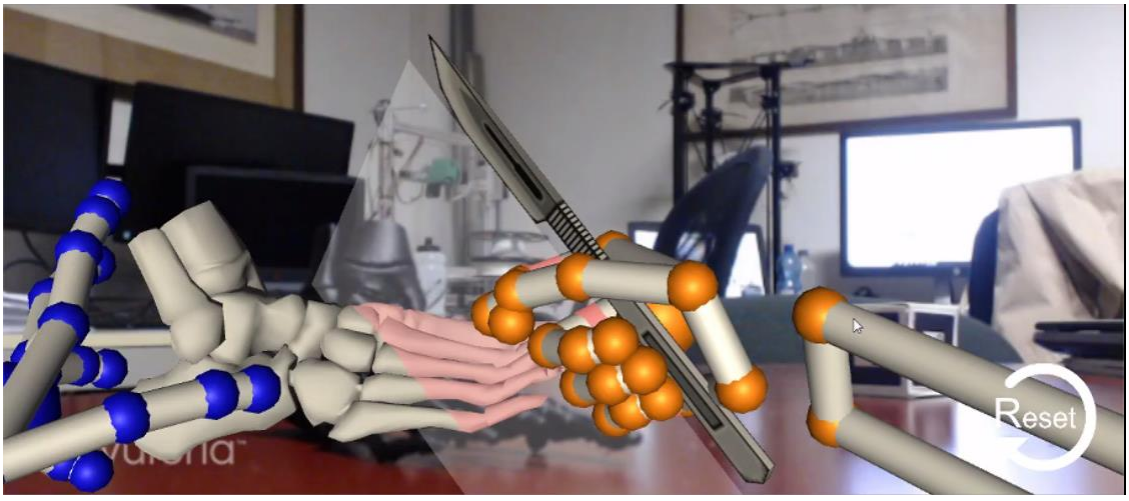

Figura 7.10: Interazione col Model Target, preparazione al taglio.

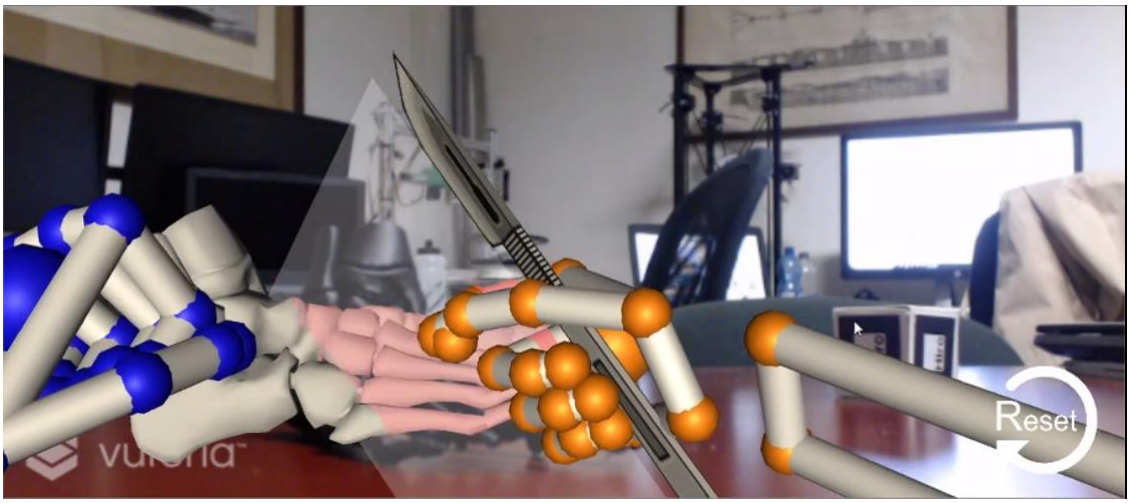

Figura 7.11: Mano sinistra chiusa, taglio effettuato.

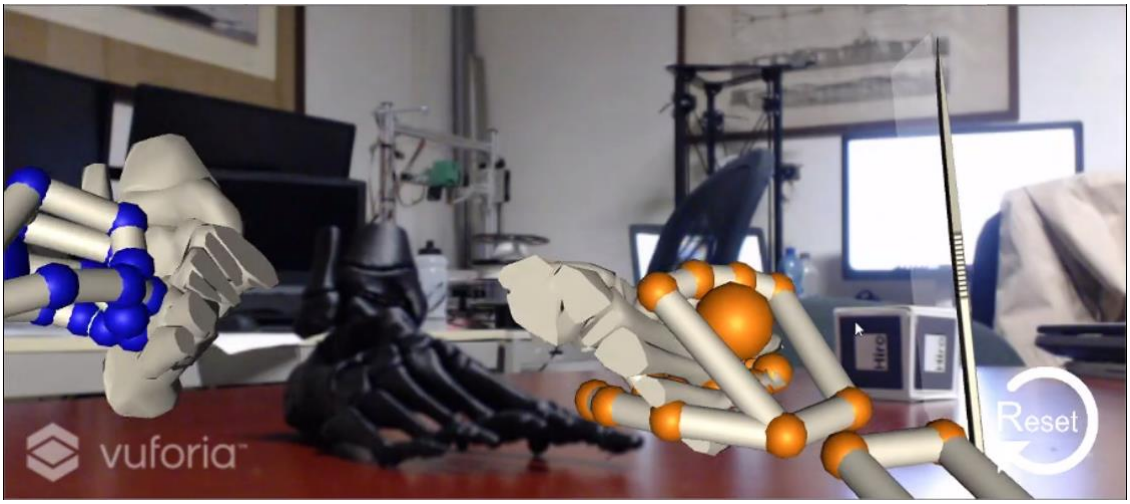

Figura 7.12: Manipolazione degli oggetti tagliati.

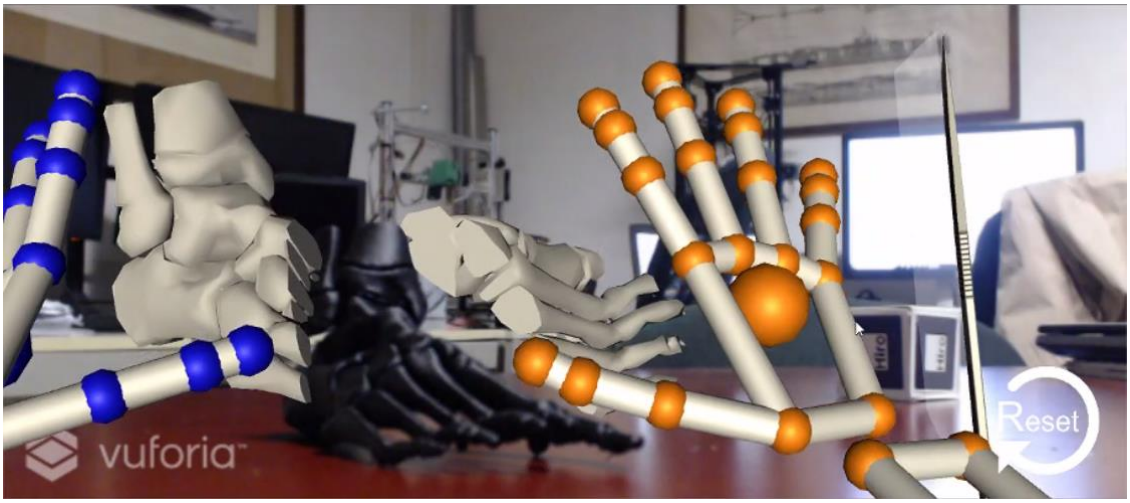

Figura 7.12: Collocazione dei frammenti nello spazio.

Il progetto MRTC rappresenta quindi il prodotto finito da presentare all'Istituto Ortopedico Rizzoli che garantisce il *tracking* di oggetti reali e relativa sovrapposizione dei corrispondenti modelli digitali, il taglio preciso e veloce degli elementi virtuali applicabile tutte le volte che si desidera e in fine l'interazione con i vari frammenti in un ambiente Mixed Reality, che consenta di simulare un'operazione chirurgica nel mondo reale.

Per soddisfare al meglio tutti i requisiti sopra citati e garantire un'ottima esperienza con l'applicazione realizzata, sono state fatte diverse considerazioni e valutazioni sulla precisione e sulle tempistiche del taglio. Prima di tutto per consentire di sezionare gli oggetti in maniera precisa nel punto di applicazione del coltello, è stato fondamentale applicare ai vari elementi tridimensionali i *Mesh Collider,* opportunamente ricostruiti e riadattati in seguito ad ogni taglio. Tuttavia, un'operazione così precisa, effettuata sulle ossa virtuali ricavate direttamente da InVesalius, avrebbe richiesto tempistiche esagerate (si vedano i risultati dei testi riportati in tabella al paragrafo 4.6).

Piuttosto che optare per una soluzione alternativa che non facesse uso di *Mesh Collider,* meno precisa ma sicuramente più veloce, si è notato che associando alle ossa i corrispondenti oggetti virtuali, modificati con Blender in modo da avere forme leggermente più smussate con un reticolo più semplificato, il taglio era pressoché istantaneo. In particolare, con un numero di facce poligonali al di sotto di 4500, un qualunque componente osseo mantiene le sue principali caratteristiche e viene sezionato nel giro di un secondo.

Un' ultima importante considerazione riguarda il materiale scelto per la stampa di oggetti tridimensionale, qualora si intendesse lavorare con prototipi e copie di oggetti reali. Seppure il *tracking* di questi ultimi avvenga in ogni caso, e una volta effettuato, persista senza problemi mantenendo l'elemento virtuale saldamente ancorato a quello reale, potrebbe tuttavia impiegare un tempo maggiore a riconoscere superfici lucide, chiare e con pochi dettagli. Sono stati effettuati infatti diversi test ed è stato dimostrato che uno stesso oggetto illuminato dalla stessa sorgente luminosa, realizzato prima in Nylon bianco lucido poi in Nylon nero opaco, richiedeva decine di secondi in più per essere riconosciuto, spostando anche diverse volte la telecamera. A parità di condizioni di luce e colore del materiale poi, il *tracking* di elementi piuttosto dettagliati avviene senz'altro più velocemente rispetto ad altri meno con meno caratteristiche.

Per una migliore esperienza con questa applicazione, si suggerisce quindi di tenere in considerazione tutte le osservazioni fatte sopra.

Conclusioni e sviluppi futuri

Questa tesi mi ha permesso di andare a toccare con mano una realtà che per me era inizialmente del tutto sconosciuta. Tecniche come il *tracking* di oggetti reali e dispositivi *Mixed Reality* come Leap Motion e Microsoft HoloLens, sono piuttosto recenti e in fase di sviluppo pertanto la documentazione su di essi scarseggia, ma una volta compresi appieno i punti di forza e le potenzialità di queste tecnologie, risulta chiaro come esse possano essere sfruttate nella vita di tutti i giorni. Basti pensare che il progetto a cui ho lavorato, è stato richiesto da un ospedale per consentire ai chirurgi di effettuare uno o più tagli su componenti ossei virtuali, sovrapposti a quelli reali, ripetendo l'operazione tutte le volte che si desidera e registrando l'esecuzione migliore in modo da rieseguirla allo stesso modo in sala operatoria. Tutto ciò mi ha reso consapevole dell'importanza di questo ramo dell'informatica, degli incredibili passi avanti che ha fatto la scienza e mi ha stimolato inoltre a ragionare sulle possibili applicazioni *Mixed Reality* che potrebbero essere sviluppate e utilizzate un indomani dalla comunità.

Per concludere posso affermare con certezza che il raggiungimento di questi risultati è stato molto gratificante ed ha contribuito ad accendere il mio interesse nei confronti della Realtà Virtuale, Aumentata e Mista e di tutto ciò che concerne la modellazione di oggetti tridimensionali digitali e l'interazione con essi.

Il progetto MRTC è stato realizzato per funzionare sul dispositivo Leap Motion e mostrare ai medici del Rizzoli un'anteprima del suo funzionamento in un ambiente *Mixed Reality*, ma è già prevista una sua estensione ed integrazione col visore Microsoft HoloLens. Inoltre, un'applicazione del genere che sfrutti Leap Motion o il visore appena citato, potrebbe essere sviluppata e riadattata per essere utilizzata in altri settori oltre a quello medico. Sicuramente potrebbe essere integrata in un videogioco *MixedReality*, per permettere di interagire in prima persona con ogni tipo di oggetto presente nella stanza, tagliandolo ad esempio con una spada virtuale. Senza apportare nessuna modifica al progetto, questo potrebbe essere utilizzato anche in ambito accademico per mostrare ad esempio agli studenti di ingegneria un qualsiasi pezzo meccanico sezionato in qualunque punto, senza dover spaccare l'oggetto fisico reale in più parti.

Chiaramente questi sono solo due dei possibili ambiti di utilizzo di MRTC project.

Ringraziamenti

Ritengo sia doveroso innanzitutto ringraziare i principali contribuenti al progetto, nonché coloro che mi hanno permesso di raggiungere e conseguire il titolo di laurea, che suggella il termine di questo primo percorso universitario.

Ringrazio quindi il Prof. Antonio Corradi, relatore di tesi, che nel corso di questi tre anni si è sempre reso disponibile verso me e i miei colleghi e anche in quest'ultima occasione ha dato prova della sua professionalità, seguendomi con precisione durante il periodo di stesura e suggerendomi fonti utili da cui attingere per migliorare il mio lavoro. Allo stesso modo vorrei ringraziare il Prof. Alfredo Liverani, correlatore di tesi, senza il quale non sarei nemmeno venuto a conoscenza di questo progetto, che mi ha ospitato più volte nel suo ufficio, fornendomi tutto il materiale necessario su cui studiare e sperimentare, dandomi supporto decisionale e guidandomi nella realizzazione dell'elaborato finale. Ci tengo fortemente a menzionare anche i dottorandi Gian Maria Santi e Francesco Osti, sempre aperti al dialogo, che fin da subito si sono rivelati fonte di ispirazione e conoscenza, spiegandomi per filo e per segno il lavoro da svolgere, elencandomi tutte le problematiche del caso e trattandomi come un collega al loro pari col quale scambiare idee e opinioni.

A questo punto, i ringraziamenti ai propri cari si sprecano, ma ci tengo ugualmente a rendere omaggio alla mia famiglia, a partire da mio padre e mia madre che mi hanno sempre supportato e appoggiato in ogni scelta, spronandomi a superare ogni ostacolo e consigliandomi sempre la cosa giusta da fare, fino ad arrivare a mio fratello e mia sorella, che sono per me due grandi modelli di riferimento da emulare per la voglia di mettersi in gioco dell'uno e la costanza e determinazione dell'altro. Ci terrei a ringraziare poi tutti i miei amici, in primis quelli di vecchia data con cui ho condiviso esperienze indimenticabili, che mi son sempre stati accanto, mi conoscono e mi hanno consolato nei momenti difficili ed elogiato nei momenti di gioia, poi quelli incontrati negli ultimi tre anni, che senz'altro hanno avuto un peso determinante nel conseguimento di questo risultato, coi quali ho stretto un forte legame affrontando e superando insieme i vari esami universitari. Un ultimo ringraziamento speciale anche a tutte le persone incontrate negli ultimi mesi che mi hanno regalato grandi emozioni ed hanno contribuito ad arricchire la mia persona.

Bibliografia

[1] https://ieeexplore.ieee.org/document/6674458, 2013.

[2] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, e B. MacIntyre. *Recent advances in augmented reality. Computer Graphics and Applications*, IEEE, 2001.

[3] https://www.amvi.it/, 2017.

[4] P. Milgram, F. Kishino, *A taxonomy of mixed reality virtual displays,* IEICE Trans. Information Systems, 1994.

[5] https://www.youtube.com/watch?v=xgakdcEzVwg&t=8s, 2015.

[6] Julia Tokareva, *What is the difference between virtual reality, augmented reality and mixed reality?*, https://it.quora.com/, 2018.

[7] Microsoft HoloLens, Sito ufficiale, https://www.microsoft.com/en-us/hololens, 2018.

[8] https://docs.microsoft.com/en-us/windows/mixed-reality/using-the-hololens-emulator, 2018.

[9] Unity, Sito ufficiale, https://unity3d.com/, 2018.

[10] https://invesalius.github.io/, 2017.

[11] Mark Two, Sito ufficiale, https://markforged.com/mark-two/, 2018.

[12] Blender, Sito ufficiale, https://www.blender.org/, 2018.

[13] Vuforia, Sito ufficiale, https://www.vuforia.com/, 2018.

[14] Leap Motion, Sito ufficiale, https://www.leapmotion.com/,2018.