UNIVERSITÀ DEGLI STUDI DI PARMA

Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria Informatica

UN SISTEMA APTICO IN REALTÀ AUMENTATA CON SIMULAZIONE FISICA

Relatore:

Chiar.mo Prof. STEFANO CASELLI

Correlatore:

Ing. JACOPO ALEOTTI

Tesi di Laurea di:

FRANCESCO DENARO

ANNO ACCADEMICO 2007-2008

Un giorno le macchine riusciranno a risolvere
tutti i problemi, ma mai nessuna di esse
potrà porne uno. (Albert Einstein)

Indice

Indice **i**

1 Introduzione **1**

1.1 Organizzazione della tesi 8

2 Tecnologie per la Realtà Aumentata **9**

2.1 Strumenti di computer graphics 10

2.1.1 Ambienti virtuali 11

2.1.2 ARToolKit 12

2.1.3 OpenGL 23

2.2 Interazione Aptica 25

2.2.1 Interfacce Aptiche 26

2.2.2 Struttura di un'interfaccia aptica con force feedback . . 28

2.2.3 Caratteristiche di un'interfaccia aptica 29

2.2.4 Classificazione delle interfaccie aptiche 32

2.2.5 Come compensare le limitazioni dell'hardware aptico . 33

2.3 Simulatori fisici 35

2.4 Strumenti hardware 39

2.4.1 Strumento aptico utilizzato 40

3 Architettura del sistema 41

3.1 Corrispondenza tra ambiente reale e virtuale 43

3.1.1 Calibrazione della telecamera 43

3.1.2 Risultati della Calibrazione 46

3.1.3 Modello di una telecamera reale 48

3.1.4 Creazione dell'immagine eseguita da un programma di computer graphics 51

3.1.5 Corrispondenza tra i due modelli di telecamera 63

3.1.6 Gestione dell'occlusione 65

3.2 Il simulatore fisico 66

3.2.1 Classi e thread dell'applicazione 72

3.3 Il dispositivo aptico 93

3.3.1 Inizializzazione del Falcon 94

3.3.2 Le "callback functions" e la sincronizzazione 99

3.3.3 Comunicazione con la classe PhysicsClass 102

4 Risultati sperimentali 104

4.1 Valutazione dell'accuratezza dell'ambiente di realtà aumentata 105

4.2 Esperimenti di interazione 107

4.3 Valutazione preliminare degli oggetti soft 112

4.4 Valutazione preliminare del riconoscimento di marker 115

5 Conclusioni 117

Bibliografia 120

Ringraziamenti 123

Capitolo 1

Introduzione

Lo scopo di questa tesi è la progettazione e la realizzazione di un sistema aptico in realtà aumentata con simulazione fisica. In questo capitolo si discutono le motivazioni della tesi, i principali obbiettivi del progetto e i risultati ottenuti. In particolare, saranno messi in evidenza i contributi innovativi del lavoro svolto rispetto alla stato dell'arte della ricerca nei settori della computer grafica e dell'interazione uomo-macchina. Il capitolo inoltre fornisce una descrizione generale dell'architettura del sistema sviluppato ed introduce i principali elementi del sistema che saranno analizzati nei capitoli successivi.

L'interazione Uomo-Macchina può essere definita in molti modi. É stata descritta come "la disciplina che si occupa della progettazione, valutazione e implementazione di sistemi interattivi e dello studio di tutti i fenomeni più importanti connessi con l'interazione tra questo tipo di sistemi e l'essere umano" [1] oppure, ponendo l'accendo su altri aspetti: "l'Interazione Uomo-Macchina si occupa della progettazione di sistemi che siano in grado di aiutare le persone che li usano a svolgere la loro attività in modo produttivo e sicuro" [2]. Dalle precedenti definizioni di interazione uomo-macchina si

deduce che un aspetto importante riguarda l'applicazione sistematica delle conoscenze sia sulle caratteristiche cognitive degli esseri umani che sulle capacità delle macchine.
Oggetto specifico di questa tesi sono le interfacce uomo-macchina che comprendono sistemi di simulazione grafica 3D e dispositivi di interazione tattile. Il termine interfaccia è una parola chiave nell'ambito dell'interazione uomo-computer e nello sviluppo stesso di sistemi interattivi, in quanto essa influisce sulla nostra capacità di utilizzarli, oltre che sulla nostra motivazione a farlo. Ciò ricopre una grande importanza perchè i sistemi interattivi vengono ormai utilizzati in tutti i settori della vita: intrattenimento, tempo libero, lavoro, servizi.

Il settore della computer grafica, applicato alla progettazione di sistemi che consentono all'utente di interagire con il computer, è chiamato computer grafica interattiva. Con il termine interattiva si vuole distinguere questo settore da altri rami della computer graphics[1] in cui le immagini sono invece generate tramite plotter digitali, film recorder, stampanti, o comunque dispositivi che generano immagini permanenti, cioè in cui le immagini vengono fruite non interattivamente. Una delle tecniche di computer grafica interattiva più avanzata è rappresentata dalla così detta realtà aumentata. Per augmented reality (AR) s'intendono quelle applicazioni il cui scopo è di incrementare la percezione visiva dello spazio fisico con immagini prese da uno spazio virtuale. Un sistema di questo tipo genera una scena composita in cui la visione dell'utente immerso nel mondo reale è "aumentata" con infor-

[1] La computer graphics (CG) è il settore dell'informatica che riguarda l'impiego del calcolatore nel campo della grafica, e quindi lo studio di tecniche di rappresentazione delle informazioni con modalità grafiche atte a migliorare la comunicazione tra uomo e macchina o tra uomo e uomo.

mazioni addizionali generate dal computer. In questo modo l'ambiente reale e quello virtuale sembrano coesistere, e l'utente si può muovere liberamente nella scena generata interagendo con essa.
Un fattore molto importante per ottenere un elevato livello di realismo in ambienti di realtà virtuale e di realtà aumentata consiste nell'utilizzo di simulazioni fisiche dell'ambiente.
Un simulatore fisico è un programma che simula i modelli fisici Newtoniani usando per i corpi dell'ambiente proprietà come massa, velocità e forze di attrito. I simulatori fisici sono usati prevalentemente in simulazioni scientifiche e nei video giochi.
Ci sono due classi principali di simulatori fisici: quelli in tempo reale e quelli ad alta precisione. I simulatori ad alta precisione richiedono più potenza di calcolo per calcolare con precisione i processi fisici e sono in genere utilizzati in settori ingegneristici e nei film di animazione.
Nei videogiochi, o in altre forme di interazione con il computer, i simulatori fisici semplificano i lori calcoli e abbassano l'accuratezza così da poter calcolare in tempo reale le informazioni garantendo un adeguato frame-rate per l'applicazione. Questi tipi di simulatori sono definiti simulatori in tempo reale. Il principale compito di un simulatore fisico è l'individuazione delle collisioni, la risoluzione delle collisioni e dei vincoli e fornire le future coordinate "mondo"[2] di tutti gli oggetti.

Un altro componente fondamentale di un sistema di simulazione grafico 3D interattivo è costituito dal dispositivo fornito all'utente per l'interazione. In questa tesi si è utilizzato un dispositivo aptico. Il termine aptico deriva

[2]Le coordinate "mondo" indicano il centro di massa per i corpi rigidi e le trasformate sui vertici per i corpi deformabili.

dal greco "aptio", che significa tocco: con questo attributo, perciò, s'intende qualcosa che abbia a che fare con il tatto. Un'interfaccia aptica, quindi, è un dispositivo robotico studiato per interagire direttamente con l'operatore umano, il quale riceve in risposta delle sensazioni tattili, come, ad esempio, le forze relative all'oggetto con cui sta avvenendo il contatto. Semplici esempi d'interfacce aptiche sono un joystick con ritorno di forza o un mouse con rotellina, in grado di bloccarsi automaticamente nel punto in cui l'utente raggiunge il margine dello schermo.

L'obbiettivo della tesi è la progettazione e realizzazione di un sistema di realtà aumentata con simulazione fisica e interazione aptica. In particolare le caratteristiche fondamentali del sistema progettato sono le seguenti:

- Progettazione di un sistema di realtà aumentata orientato alla manipolazione di oggetti. Il sistema consente all'utente di interagire con oggetti simulati mediante applicazione di forze attraverso l'uso di un dispositivo aptico.
- Integrazione di un algoritmo per il riconoscimento di marker in tempo reale.
- Progettazione di una interfaccia grafica avanzata in grado di rappresentare fedelmente l'ambiente virtuale. In particolare il sottosistema grafico supporta tecniche di illuminazione, gestione delle occlusioni, creazione di oggetti convessi e concavi.
- Integrazione di un simulatore dinamico open-source per ottenere la simulazione fisica dell'ambiente virtuale. In particolare, tale simulatore consente di definire corpi rigidi e corpi deformabili.

- Integrazione di un dispositivo aptico a tre gradi di libertà per la manipolazione della posizione di un punto nello spazio operativo dell'utente che consente il ritorno di un feedback di forza.

L'attività svolta e documentata nella tesi ha portato alla realizzazione di un sistema stabile e affidabile, le cui caratteristiche sono descritte in dettaglio nei capitoli successivi della tesi.

La parte innovativa di questa tesi è costituita dalla realizzazione di un sistema che comprende sia funzionalità di realtà aumentata, sia la simulazione fisica della scena, sia infine un'interfaccia aptica per l'operatore. In letteratura è presentata una sola ricerca con caratteristiche analoghe [3]. Tale lavoro descrive un'architettura simile a quella progettata in questa tesi, ma con significative limitazioni e differenze progettuali. Nell'articolo citato la simulazione fisica ricreata era limitata ad una semplice pallina, e veniva simulato il gioco del ping pong utilizzando come dispositivo aptico un Phantom [4], uno strumento molto più costoso di quello utilizzato in questa tesi. Inoltre venivano utilizzati due computer collegati in rete, uno dei quali si occupava solamente della simulazione fisica e l'altro della simulazione grafica, ottenendo quindi un sistema molto più costoso.

Il sistema creato in questa tesi è mostrato in figura 1.1. Tramite l'utilizzo di una telecamera si acquisisce l'immagine dell'ambiente reale che viene associato ad un corrispondente ambiente virtuale. Quest'ultimo riproduce le caratteristiche dell'ambiente reale mediante l'applicazione di un algoritmo di simulazione fisica. Tra i benefici ottenibili da questo sistema è compresa la possibilità di impostare le caratteristiche dei materiali come massa, attrito, ecc. Si ottiene quindi un ambiente virtuale in relazione $1:1$ con l'ambiente

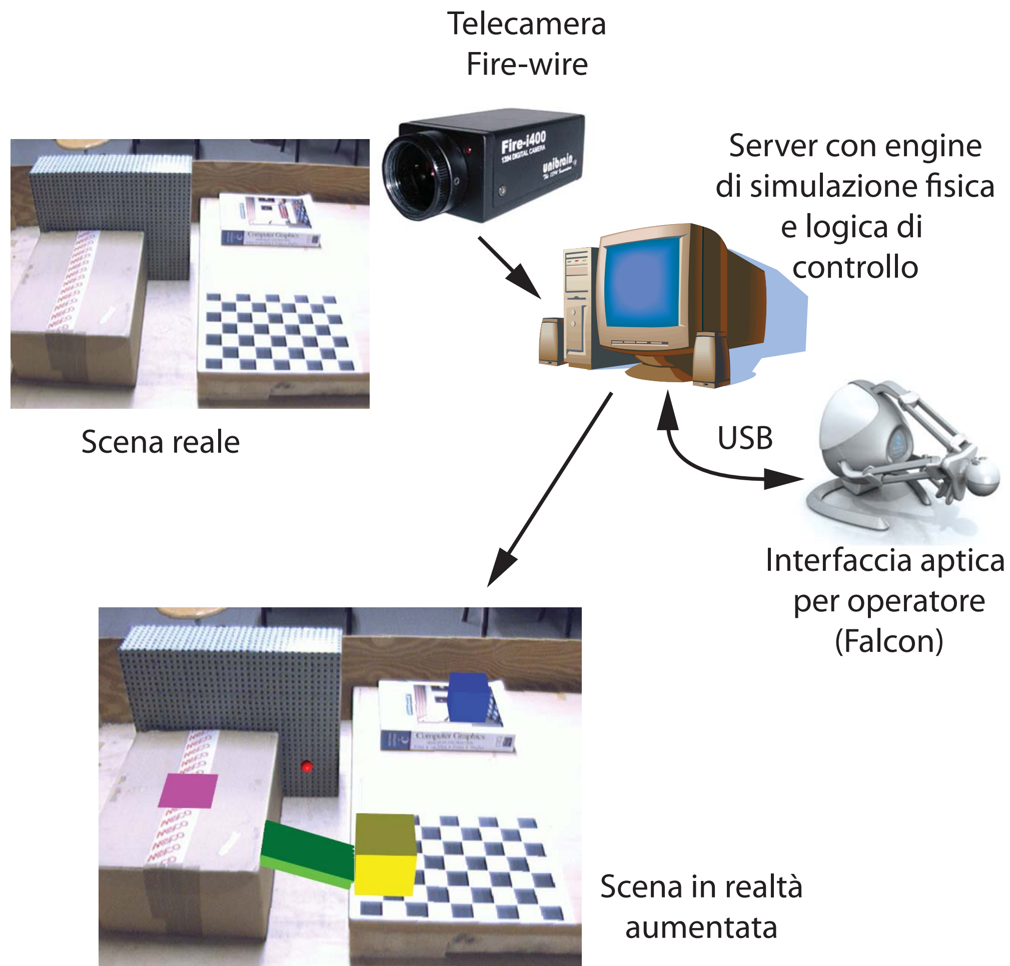
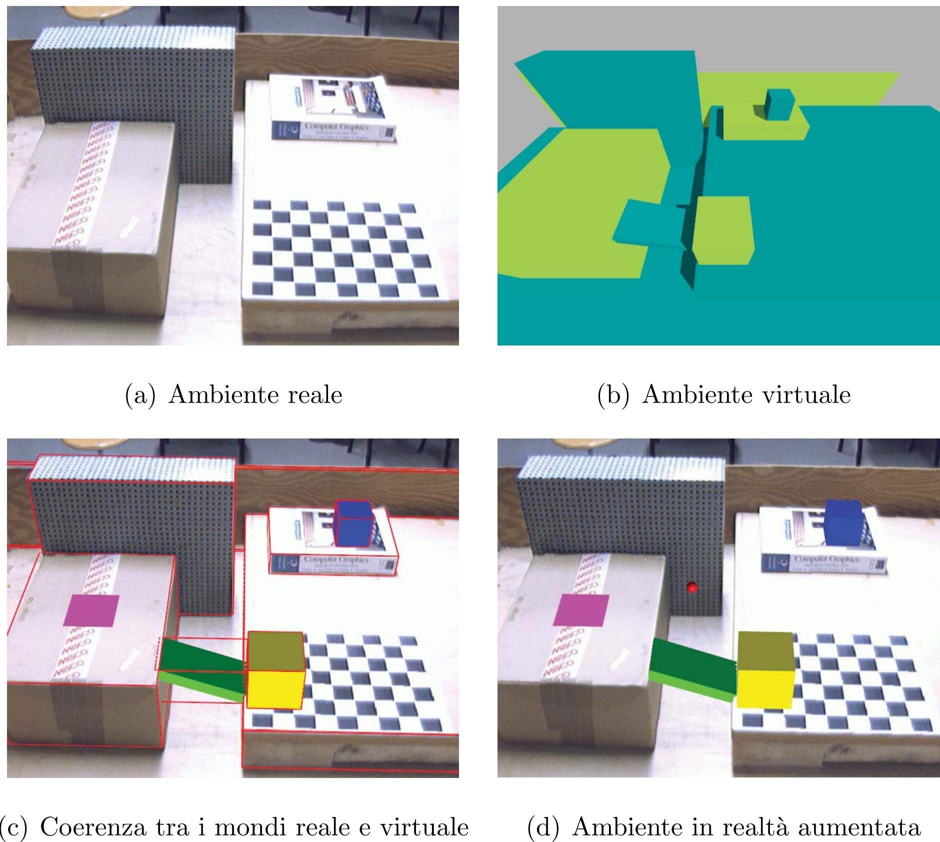


Figura 1.1: Architettura del sistema progettato in questa tesi.

reale (figura 1.2(c)), poiché ogni suo aspetto viene fedelmente riprodotto. Successivamente vengono inseriti all'interno della scena oggetti virtuali deformabili e non deformabili, alcuni dei quali sono inseriti attraverso l'utilizzo di marker.

Per interagire con l'ambiente ricreato viene utilizzato un dispositivo aptico, il cui "end effector" viene rappresentato nell'ambiente da una sfera rossa (visibile nella figura 1.2(c)). Il dispositivo aptico permette un ulteriore livello di immersione nell'ambiente ricreato, fornendo un feedback di forza. Inoltre il



(a) Ambiente reale (b) Ambiente virtuale

(c) Coerenza tra i mondi reale e virtuale (d) Ambiente in realtà aumentata

Figura 1.2: Fasi per realizzare un ambiente di realtà aumentata.

sistema gestisce correttamente il fenomeno delle occlusioni, che si manifestano quando un oggetto reale è interposto tra l'utente e un oggetto virtuale e quest'ultimo non viene visto dall'utente nella misura in cui è coperto dall'oggetto reale. La figura 1.2 mostra i passaggi necessari per ricreare una scena in realtà aumentata.

L'utilizzo della realtà aumentata risulta essere molto importante in diversi campi, tra i quali la chirurgia minimamente invasiva, i computer games [5], la guida di treni o aerei e in campo militare.

1.1 Organizzazione della tesi

La tesi è organizzata come segue:

Capitolo 2 Introduzione alle tecnologie per la Realtà Aumentata.

Capitolo 3 Descrizione dell'architettura del sistema.

Capitolo 4 Descrizione dei risultati delle prove sperimentali.

Capitolo 5 Conclusioni e possibili sviluppi futuri.

Capitolo 2

Tecnologie per la Realtà Aumentata

In questo capitolo vengono introdotte la tecnologie necessarie ai diversi moduli che compongono il sistema realizzato nella tesi: il sottosistemi d'interazione aptica, il sottosistema simulazione fisica e il sottosistema pre la realtà aumentata, creata grazie alla libreria ArtoolKit e OpenGL. La figura 2.1 mostra l'architettura generale del sistema creato.

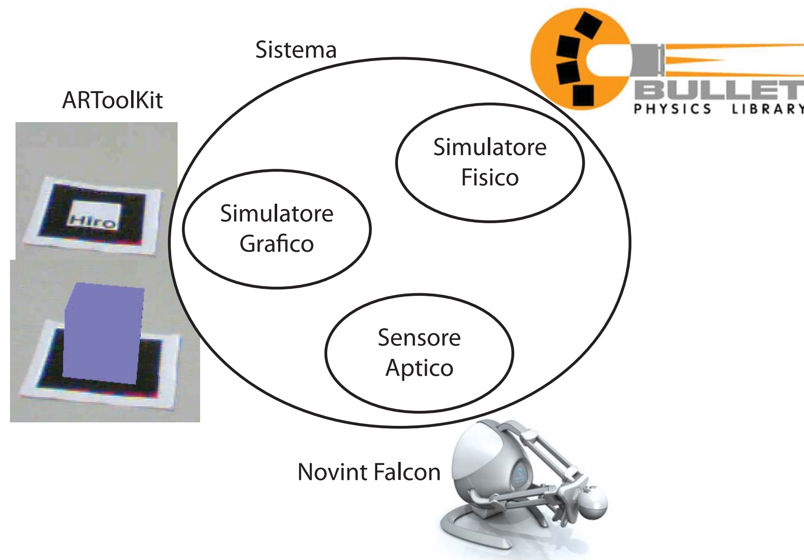


Figura 2.1: Architettura del sistema e tecnologie usate.

2.1 Strumenti di computer graphics

In questo paragrafo vengono introdotti gli strumenti utilizzati per la creazione dei sistemi di realtà aumentata, molto utilizzati nel campo della chirurgia minimamente invasiva. Per esempio le immagini viste direttamente dal chirurgo attraverso una telecamera vengono integrate con immagini TC (Computer Tomography) o MRI (Magnetic Resonance Imaging), ottenute sul paziente in precedenza. In questo modo è possibile dare al chirurgo una "visione a raggi X" e ciò riduce il trauma durante un'operazione, perché permette di effettuare un'incisione più piccola. Altre applicazioni della realtà aumentata riguardano l'ambito militare, l'intrattenimento, l'Engineering Design, ecc.
Nei paragrafi successivi viene illustrato il concetto di ambiente virtuale, e vengono descritte le librerie software Artoolkit e OpenGL, utilizzate nella tesi per creare l'ambiente di realtà aumentata.

2.1.1 Ambienti virtuali

Un ambiente virtuale è la riproduzione di un ambiente reale simulato al computer, in cui l'utente è in grado di percepire la propria presenza. In un ambiente virtuale l'utente può immergersi su tre livelli diversi. In un primo livello l'operatore è in grado di percepire la propria presenza all'interno del "virtual environment". Nel secondo livello di immersione l'operatore naviga all'interno dell'ambiente simulato. Infine, nel terzo livello d'immersione l'operatore è in grado di poter interagire con l'ambiente virtuale. Se per il raggiungimento dei primi due livelli sono sufficienti simulazioni grafiche ed audio, capaci cioè di riprodurre stimolazioni di natura visiva ed uditiva, per raggiungere pienamente il terzo livello di immersione è necessario fornire all'utente anche una stimolazione tattile, resa possibile riproducendo all'interno della simulazione tutte le forze esercitate dall'ambiente esterno. La grande differenza, infatti, tra un'immersione del primo o del secondo tipo rispetto ad una del terzo consiste nel ruolo che l'operatore assume all'interno del "virtual environment": se nel primo e nel secondo livello il suo ruolo è passivo, il terzo livello conferisce all'operatore un ruolo attivo.

Figura 2.2: Livelli d'immersione nell'ambiente virtuale.

L'esigenza di realizzare ambienti immersivi e interattivi crea la necessità di simulazioni in grado di calcolare in tempo reale tutte le situazioni

di contatto al fine di far percepire correttamente all'utente le forze agenti dall'esterno: non basta più caratterizzare gli oggetti dell'ambiente simulato solo dal punto di vista grafico, ma è necessario caratterizzarli anche dal punto di vista fisico, specificando per ciascuno i valori delle caratteristiche fisiche, come ad esempio peso, viscosità, rigidezza, inerzia. Una simulazione di questo tipo, in grado di riprodurre sensazioni tattili e di forza, è ottenuta mediante dispositivi definiti Interfacce Aptiche.

2.1.2 ARToolKit

ARToolKit [6] è una libreria software open-source in linguaggio C/C++ che permette al programmatore di sviluppare applicazioni di Realtà Aumentata ('Augmented Reality', AR), cioé applicazioni grafiche in cui si effettua una sovrapposizione di un'immagine virtuale su un'immagine reale.
La libreria ARToolKit è disponibile per diversi sistemi operativi e possiede le seguenti caratteristiche:

- Uno strumento per la creazione di marker idonei per l'utillizzo con la libreria
- Uno strumento per la calibrazione della telecamera di semplice utilizzo.
- Un' applicazione AR operante in real time
- Distribuzione per SGI IRIX, Linux, MacOS e Windows
- Distribuzione completa del codice sorgente

Per essere utilizzato ARToolKit, ha bisogno di alcuni marker, per esempio quello mostrato nella figura 2.3. I marker devono essere di forma quadrica

racchiusa in una cornice nera, applicati su superfici rigide e preferibilmente non simmetrici. Un esempio di marker simmetrico si ha quando all'interno del quadrato nero è presente una croce. L'importanza di questa libreria risiede nel suo veloce algoritmo di riconoscimento dei marker, che consente di ricostruire la scena praticamente in realtime. Inoltre le prestazioni di ARToolKit dipendono ovviamente dall'hardware grafico sottostante e dalle caratteristiche della videocamera utilizzata.

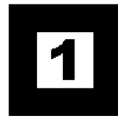
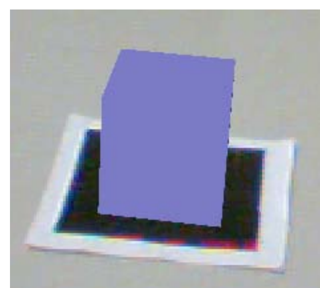


Figura 2.3: Esempio di marker creato con ARToolKit.

ARToolKit utilizza un algoritmo di visione per calcolare la reale posizione e orientazione della telecamera relativamente al marker, permettendo al programmatore di disegnare oggetti virtuali sulla scena reale. La figura 2.4(a) mostra una scena reale, mentre la figura 2.4(b) presenta l'immagine 'aumentata'. ARToolKit, per costruire l'immagine di realtà aumentata, opera nel

(a) Immagine reale.



(b) Immagine modificata.

Figura 2.4: Possibile utilizzo di ARToolKit.

seguente modo:

1. L'immagine del mondo reale viene catturata dalla telecamera e inviata al computer.

2. L'applicazione software costruita con ARToolKit ricerca i marker presenti nell'immagine:

   - Ricerca i quadrati.
   - Identifica i marker dentro i quadrati.

3. Se il marker è stato trovato, esegue i calcoli necessari per determinare la posizione della telecamera relativa al marker.

4. Posiziona la telecamera e disegna il modello grafico che è rappresentato dal marker.

5. Il modello grafico è disegnato a video sopra la texture che rappresenta il mondo reale, e in tal modo appare sul marker.

La figura 2.5 sintetizza i passi appena esposti.

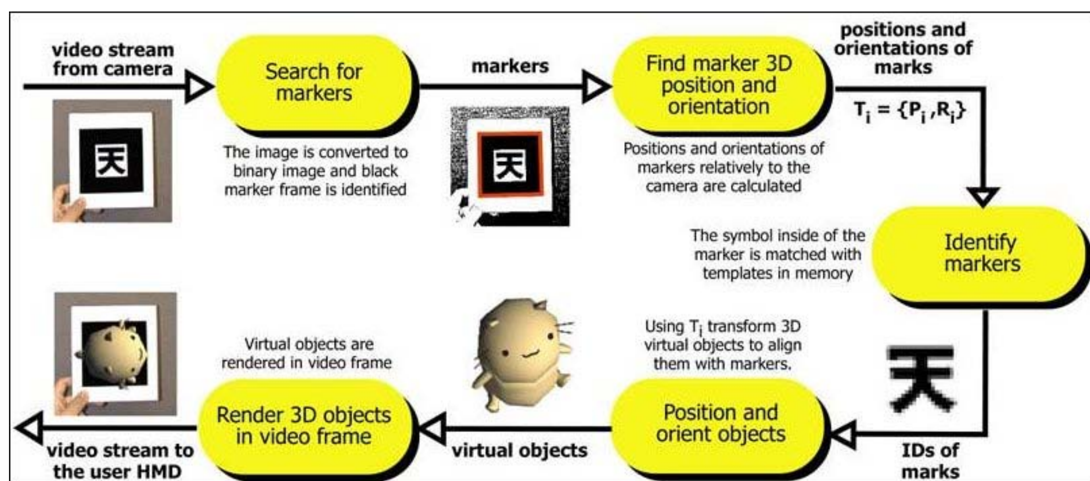
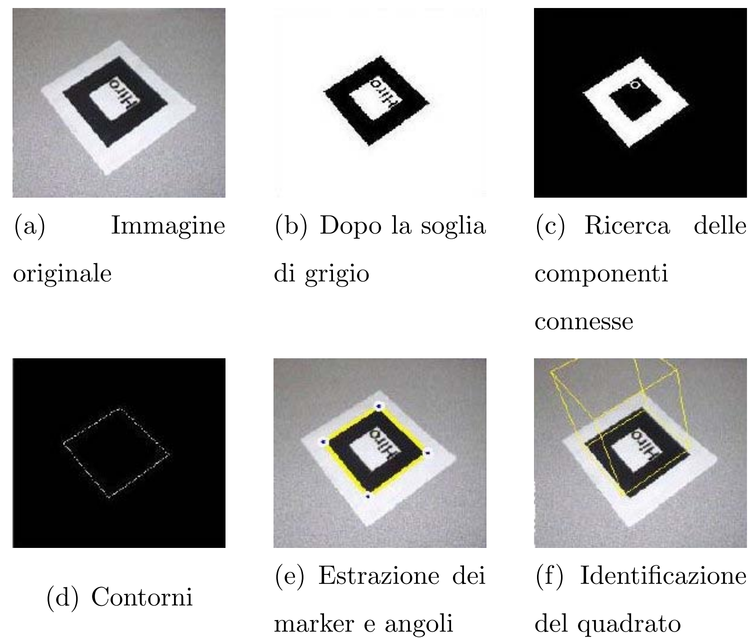


Figura 2.5: Schema di funzionamento di ARToolKit.

Muovendo il marker anche l'oggetto virtuale segue i suoi stessi spostamenti. La limitazione più rilevante di ARToolKit è legata alla visualizzazione del marker, che deve essere totalmente visibile (compresa anche la cornice bianca attorno al quadrato nero). Per questo motivo più il marker è semplice più è probabile un corretto riconoscimento.
La rilevazione del marker, oltre che dalle dimensioni dei pattern, dipende anche dall'orientazione rispetto alla telecamera. Un'altra componente di rilievo nella rilevazione è data dalle condizioni di luce, poiché luci ambientali possono creare riflessioni e punti di specchio sul marker, rendendo più difficile l'individuazione del suo quadrato. Per ridurre l'effetto riflettente i marker possono essere realizzati con materiale antiriflesso.

L'algoritmo di visione di ARToolKit è basato su una rilevazione d'angolo con un veloce algoritmo sulla posizione. La figura 2.6 mostra le principali immagini che si ottengono dall'algoritmo di visione, mentre la figura 2.7 mostra l'algoritmo di visione nei suoi passi principali. Tali passi sono:

- Binarizzazione dell'immagine acquisita.
- Applicazione della soglia di grigio.
- Ricerca dei contorni.
- Ricerca degli angoli.
- Normalizzazione del pattern.
- Confronto fra il pattern trovato e il pattern del marker in memoria.



(a) Immagine originale (b) Dopo la soglia di grigio (c) Ricerca delle componenti connesse

(d) Contorni (e) Estrazione dei marker e angoli (f) Identificazione del quadrato

Figura 2.6: Risultati dei passi di elaborazione dell'algoritmo di visione di ARToolKit.

2.1.2.1 Calibrazione della telecamera

Nel software ARToolKit è presente un file 'camera_para.dat' dove sono contenuti i parametri di default per la telecamera. Questi parametri sono sufficienti per diversi tipi di telecamere, tuttavia utilizzando una procedura di calibrazione è possibile generare dei parametri specifici per la telecamera che si sta utilizzando. La procedura di calibrazione consente di migliorare le prestazioni di riconoscimento dell'algoritmo, ed è suddivisa in due passi che si avvalgono di due programmi distinti: calib_dist e calib_cparam.

Le proprietà della telecamera sono misurate tenendo conto del punto centrale dell'immagine della telecamera, della distorsione della lente e della lunghezza del punto focale della telecamera.

Il programma 'calib_dist.exe' è usato per misurare il punto centrale della

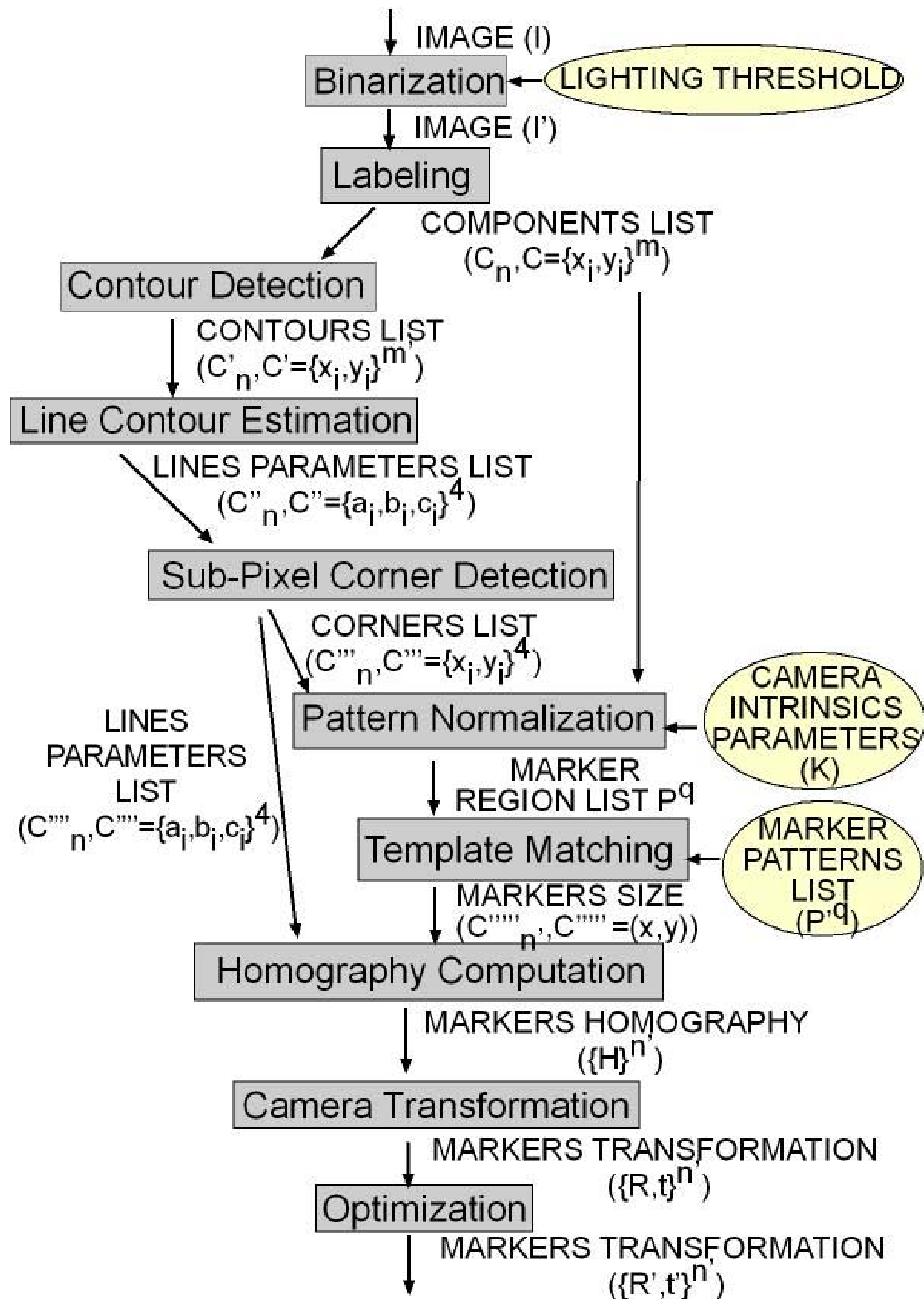


Figura 2.7: L'algoritmo di visione utilizzato in ARToolKit.

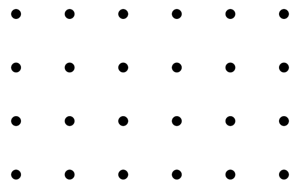


Figura 2.8: Griglia di punti.

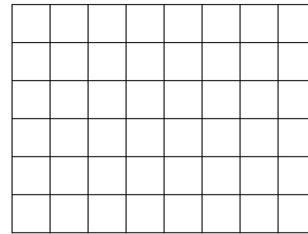


Figura 2.9: Griglia di linee.

telecamera e la distorsione della lente, e deve essere eseguito prima del programma calib_cparam.exe.

Programma calib_dist

Il programma calib_dist.exe si avvale di una griglia di punti come quella mostrata in figura 2.8, che rappresenta 6x4 punti equidistanti, e utilizza la distanza nota tra i punti per compensare la distorsione della lente della telecamera.
Quando si lancia il programma compare una finestra che mostra la scena acquisita dalla telecamera, nella quale si posiziona l'immagine della figura 2.8, in modo tale da poter vedere tutti i punti della figura. Successivamente si preme il pulsante sinistro del mouse, che causa il congelamento dell'acquisizione video da parte della telecamera.
Successivamente, su ogni punto dell'immagine si disegna un quadrilatero nero tenendo premuto il pulsante sinistro del mouse, iniziando dal primo punto in alto a sinistra e terminando con l'ultimo in basso a destra. I punti devono essere presi nel seguente ordine:

```
 1  2  3  4  5  6
 7  8  9 10 11 12
```

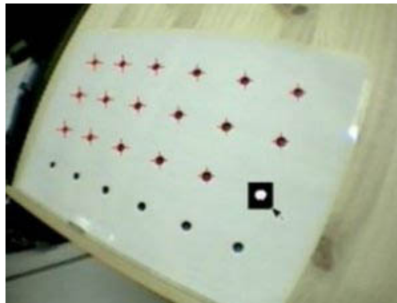


Figura 2.10: Calibrazione della telecamera.

```
13 14 15 16 17 18
19 20 21 22 23 24
```

La figura 2.10 mostra il rettangolo su uno degli ultimi punti dall'utente. Ogni volta che viene trovato un nuovo punto nell'immagine il programma ne conferma l'individuazione, disegnando una crocetta di colore rosso al centro del punto. Quando tutti i 24 punti dell'immagine sono stati trovati bisogna premere nuovamente il pulsante sinistro del mouse; in questo modo il programma memorizza la posizione dei punti e riavvia l'acquisizione dell'immagine.
Si ripete il processo per 5–10 immagini variando l'angolo e la posizione dell'immagine rispetto alla telecamera. Dopo aver preso 5–10 immagini si preme col pulsante destro del mouse per fermare la cattura dell'immagine, e si inizia il calcolo della distorsione delle immagini, che può richiedere qualche minuto. I valori calcolati, che dipendono dal tipo di telecamera, sono utilizzati dal programma calib_cparam.

Per controllare la correttezza dei parametri calcolati si preme il pulsante sinistro del mouse, ottenendo l'immagine su cui era stata eseguita la prima calibrazione, con delle linee rosse che attraversano i punti di calibrazione che

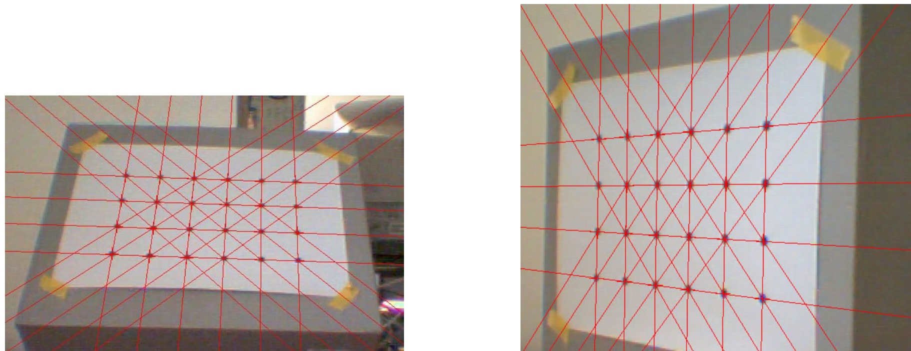


Figura 2.11: Esempi di risultati di calibrazione.

dovrebbero passare per il centro di ogni punto come mostra la figura 2.11. Ogni volta che si preme il pulsante sinistro del mouse viene mostrata l'immagine della successiva calibrazione. Se si è soddisfatti dei risultati si preme il tasto destro del mouse per terminare il programma. Con i dati ottenuti è possibile eseguire il programma calib_cparam, che fornisce le caratteristiche definitive della telecamera.

Programma calib_cparam

Il programma calib_cparam è utilizzato principalmente per determinare la lunghezza focale della telecamera e altri parametri. Esso utilizza la griglia mostrata in figura 2.9, cioè una griglia di 7 linee orizzontali e 9 linee verticali. Questa immagine deve essere applicata su una superficie rigida. All'avvio del programma vengono richiesti alcuni dati che sono stati precedentemente trovati col programma calib_dist. Successivamente compare una finestra che mostra l'acquisizione dell'immagine da parte della telecamera. A questo punto bisogna seguire la seguente procedura:

1. Posizionare l'immagine di fronte e perpendicolare alla telecamera, in modo che la griglia sia la più larga possibile e che tutte le linee siano

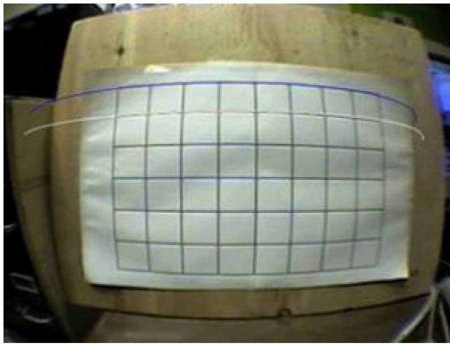


Figura 2.12: Posizionamento linee orizzontali

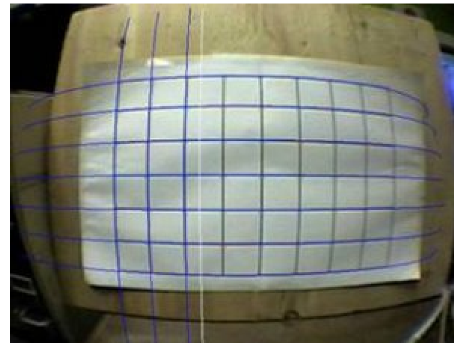


Figura 2.13: Posizionamento linee verticali

visibili.

2. Cliccare sull'immagine col pulsante sinistro del mouse. Una linea bianca comparirà a video sopra l'immagine.

3. Muovere la linea bianca in alto alla griglia e cercare di allineare il più possibile questa linea con la linea della griglia. Quando sono allineate premere il pulsante enter. La linea bianca diventa blu e un'altra linea bianca compare nell'immagine. Ripetere il processo per tutte le linee orizzontali (figura 2.12). Una volta finito con le linee orizzontali la linea bianca diventa verticale e il procedimento deve essere ripetuto per le linee verticali (figura 2.13).
   L'ordine delle linee è molto rilevante, esse devono essere disposte dall'alto verso il basso e da sinistra a destra fino a disegnare tutte le 16 linee sullo schermo.

4. Quando questo processo è stato completato per un'immagine, l'immagine deve essere allontanata dalla telecamera di 10cm (tenendola sempre perpendicolare alla telecamera). A questo punto si deve ripetere il procedimento.

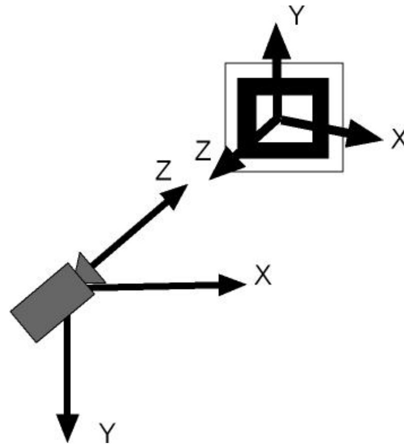


Figura 2.14: I sistemi di coordinate di ARToolKit per telecamera e marker.

5. Si ripete tutto il procedimento 5 volte, allontanando il disegno di calibrazione fino a 50cm dalla telecamera. Dopo le cinque calibrazioni il programma calcola automaticamente i parametri. Successivamente viene chiesto il nome del file sul quale memorizzare i parametri.

Ottenuti i valori di calibrazione della telecamera è possibile utilizzare il file per migliorare le prestazioni di ARToolKit.

2.1.2.2 Il sistema di coordinate

ARToolKit fornisce la posizione del marker riferito al sistema di coordinate della telecamera, e utilizza la matrice del sistema OpenGL per posizionare l'oggetto virtuale. Il sistema di coordinate (Coordinate System, CS) di ARToolKit è mostrato in figura 2.14.

È inoltre possibile calcolare la posizione della telecamera riferita al marker (invertendo la matrice di trasformazione) e anche la posizione di quest'ultimo riferita ad un altro marker. Le relazioni tra marker e telecamera forniscono pertanto la possibilità di lavorare con diversi sistemi di coordinate.

I differenti sistemi di coordinate sono usati principalmente da algoritmi di visione e dal rendering. La figura 2.15 mostra i principali sistemi di coordinate di ARToolKit. La posizione e l'orientazione dei marker vengono individuate

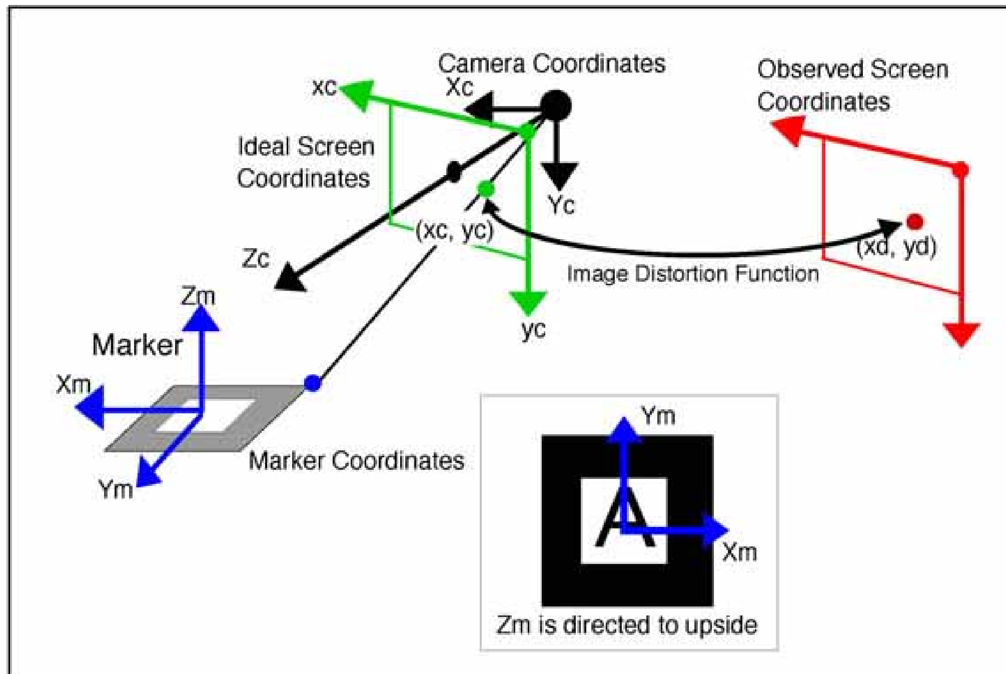


Figura 2.15: Tutti i sistemi di coordinate di ARToolKit.

con la nota forma matriciale della trasformazione omogenea (matrice 4x4).

2.1.3 OpenGL

OpenGL [7, 8] (Open Graphics Library) è una specifica che definisce una API per più linguaggi e per più piattaforme per scrivere applicazioni che producono computer grafica 2D e 3D. L'interfaccia consiste in circa 250 diverse chiamate di funzione utili per disegnare scene tridimensionali, anche complesse, a partire da semplici primitive. OpenGL è usato per sviluppare

videogiochi, applicazioni di CAD, realtà virtuale, e CAE. È lo standard di fatto per la computer grafica 3D in ambiente Unix.

OpenGL fornisce supporto via software per le caratteristiche non supportate dall'hardware, permettendo alle applicazioni di utilizzare la grafica avanzata su sistemi di potenza computazionale relativamente bassa.
OpenGL assolve a due compiti fondamentali:

- nascondere la complessità di interfacciamento con acceleratori 3D differenti, offrendo al programmatore una API unica ed uniforme;
- nascondere le capacità peculiari e disomogenee offerte dai diversi acceleratori 3D, richiedendo che tutte le implementazioni supportino completamente l'insieme di funzioni OpenGL, ricorrendo ad un'emulazione software se necessario.

Il compito di OpenGL è quello di ricevere primitive come punti, linee e poligoni, e di convertirle in pixel (rasterizing o rasterizzazione). Ciò è realizzato attraverso una pipeline grafica nota come 'OpenGL state machine'. La maggior parte dei comandi OpenGL fornisce primitive alla pipeline grafica o istruisce la pipeline su come elaborarle. Prima dell'introduzione di OpenGL 2.0, ogni stadio della pipeline realizzava una funzione fissa ed era configurabile solo entro certi limiti, ma dalla versione 2.0 molti stadi sono divenuti totalmente programmabili.

OpenGL è una API procedurale che opera a basso livello, e richiede al programmatore passi precisi per disegnare una scena. Questo approccio si differenzia dalle API descrittive ad alto livello (come ad esempio Cosmo [9]), le quali, operando su struttura dati ad albero (scene graph), richiedono al pro-

grammatore solo una descrizione geometrica della scena, occupandosi automaticamente dei dettagli più complessi del rendering. La natura di OpenGL obbliga quindi i programmatori ad avere una buona conoscenza della pipeline grafica stessa, ma al contempo lascia una certa libertà per implementare complessi algoritmi di rendering.

Storicamente, OpenGL ha esercitato una notevole influenza sullo sviluppo degli acceleratori 3D, promuovendo un livello base di funzionalità che è oggi comune nelle schede video destinate al grande pubblico:

- punti, linee e poligoni disegnati come primitive base;
- una pipeline per il transform and lighting;
- Z-buffering
- Texture mapping
- Alpha blending

Una delle caratteristiche più apprezzate in ambito professionale è la retro-compatibilità tra le diverse versioni di OpenGL: programmi scritti per la versione 1.0 della libreria sono compatibili senza modifiche su implementazioni che seguono la versione 2.1.

2.2 Interazione Aptica

Negli ultimi decenni si è potuto assistere ad una crescita sempre maggiore dell'interesse degli studiosi per la realtà virtuale (*Virtual Reality*), ed in particolare verso lo studio delle metodologie che permettono l'interazione del-

l'uomo con essa. In tale contesto un ruolo di particolare rilievo è ricoperto dalle Interfacce Aptiche, dispositivi robotici grazie ai quali nuove forme di interazione con ambienti virtuali (*Virtual Environment*) sono oggi possibili [10, 11, 12, 13].

2.2.1 Interfacce Aptiche

Il termine interfaccia deriva dal latino, "inter facies", che significa tra le facce. Nasce in contesti del tutto estranei all'informatica e alle scienze umane; essa riguardò inizialmente gli ambiti della chimica, della fisica, e della meccanica ed indica il punto di contatto, di trasmissione tra due o più elementi. Applicata successivamente all'interazione uomo-calcolatore, l'interfaccia indica in un primo momento le parti dell'hardware che permettono di interagire con il calcolatore, e successivamente acquisisce un significato più forte, oggi più diffuso, indicando anche le modalità di presentazione delle informazioni sullo schermo. Il termine interfaccia è una parola chiave nell'ambito dell'interazione uomo-computer e per lo sviluppo stesso di sistemi interattivi, in quanto essa influisce sulla nostra capacità di utilizzarli, oltre che sulla nostra motivazione a farlo. Ciò ha una grande importanza perchè i sistemi interattivi vengono ormai utilizzati in tutti i settori della vita: intrattenimento, tempo libero, lavoro, servizi.

Quindi con il termine interfaccia, solitamente, identifichiamo quel dispositivo fisico o virtuale che permette la comunicazione e, quindi, lo scambio di informazioni tra due o più entità di tipo diverso. In campo tecnologico il termine indica spesso sistemi per lo scambio di informazioni tra un essere umano e la macchina, in altre parole tra reale ed artificiale. Il termine aptico

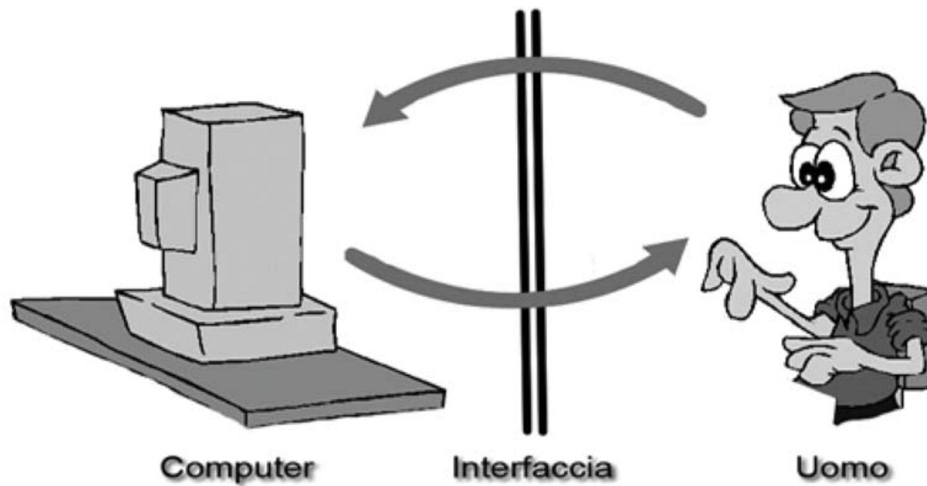


Figura 2.16: Comunicazione uomo macchina mediante interfaccia.

deriva dal greco "apto", che significa tocco: con questo attributo, perciò, s'intende qualcosa riguardante il tatto.
Un'interfaccia aptica, quindi, è un dispositivo robotico studiato per interagire direttamente con l'operatore umano, il quale riceve in risposta delle sensazioni tattili, come, ad esempio, le forze relative all'oggetto con cui sta avvenendo il contatto. Semplici interfacce aptiche sono un joystick con ritorno di forza o un mouse, la cui rotellina si blocca nel punto in cui l'utente raggiunge il margine dello schermo.
Storicamente le tecnologie aptiche sono nate negli anni '60 contemporaneamente alla nascita e allo sviluppo della robotica. Inizialmente hanno trovato applicazione nello sviluppo di sistemi per il controllo remoto di robot, di bracci robotizzati nell'ambito di voli spaziali e nel trattamento di materiali altamente nocivi per l'uomo (come i combustibili utilizzati nelle centrali nucleari). Queste prime realizzazioni impiegavano, in genere, un'interfaccia uomo macchina (Human Machine Interface, HMI) connessa fisicamente con il dispositivo controllato, ma incapace di effettuare il force feedback, cioè di

restituire in risposta sensazioni tattili. Solo a partire dalla seconda metà degli anni '80 sono state sviluppate le prime interfacce aptiche connesse ai sistemi robotici tramite la rete (LAN), svincolando l'operatore dal dispositivo che veniva controllato ed in grado di effettuare il force feedback. Questo ha permesso la nascita di innumerevoli nuove applicazioni, basate sul controllo remoto e sensorizzato, come, ad esempio, la telechirurgia. Negli ultimi anni abbiamo potuto assistere alla nascita di nuovi dispositivi aptici con capacità di force feedback, dotati di una grande flessibilità garantita sia dalla possibilità di gestione di un elevato numero di gradi di libertà da parte del dispositivo stesso, sia dall'implementazione di nuovi algoritmi per la compensazione di ampi ritardi di trasmissione.
Esistono anche dispositivi aptici orientati a restituire non un ritorno di forza ma una stimolazione tattile. Un esempio di tali dispositivi è il guanto CyberTouch [14] dell'Immersion disponibile presso il laboratorio di robotica.

2.2.2 Struttura di un'interfaccia aptica con force feedback

Un'interfaccia aptica con capacità di force feedback è un dispositivo elettromeccanico costituito da due parti principali:

Manipolatore: è la parte meccanica vera e propria, ed è costituita a sua volta da varie componenti:

- una base: fissata nell'ambiente di lavoro o costituita da una piattaforma mobile.
- una serie di links: parti rigide di collegamento.
- una serie di giunti: snodi che permettono il collegamento tra i links.

- un *end-effector*: parte terminale del manipolatore, connesso al resto del dispositivo meccanico attraverso un polso che permette all'end-effector di muoversi liberamente.

Sistema di programmazione e di controllo: solitamente è un sofisticato dispositivo di calcolo, collegato in rete ad altre risorse locali, quali sensori (che servono a misurare la posizione del dispositivo attraverso degli encoders), attuatori (che sono gli elementi, in genere elettrici o idraulici, utilizzati per effettuare i movimenti del dispositivo) e controllori (che servono per effettuare il feedback sui movimenti del dispositivo). Le funzionalità richieste ad un sistema di programmazione e di controllo sono l'immagazzinamento dati, il controllo dei giunti in tempo reale, il monitoraggio dei sensori e la capacità di interazione con l'utente.

L'utente si collega ad un'interfaccia aptica attraverso l'end-effector del manipolatore, impartendo i movimenti da effettuare, che vengono rilevati dai sensori. A questo punto viene calcolata la forza esercitata dal dispositivo (applicata attraverso gli attuatori presenti al suo interno) in funzione della posizione dell'end-effector.

2.2.3 Caratteristiche di un'interfaccia aptica

Affinché un dispositivo robotico possa essere ritenuto in grado di simulare l'interazione fisica con l'ambiente esterno, deve possedere tre caratteristiche fondamentali:

1. **Elevata trasparenza:** Se non viene rilevata nessuna interazione con l'ambiente circostante, l'utente non deve percepire la presenza del dis-

positivo; pertanto, l'interfaccia aptica deve essere realizzata in modo tale che le forze di reazione, necessarie al suo spostamento ed esercitate sull'operatore durante le variazioni di posizione effettuate nello spazio libero siano ridotte al minimo. In altre parole, occorre ridurre al minimo le forze dovute al peso del dispositivo, agli attriti interni e alle masse in movimento (cioè alle forze inerziali). Per fare in modo che questa condizione sia rispettata, occorre intervenire sia sulla componente meccanica del dispositivo (quindi sulla cinematica, sui sistemi di trasmissione, sugli attuatori, sulle parti strutturali, ecc...), sia su quella di controllo degli attuatori, mediante tecniche di compensazione nei riguardi di attrito, gravità ed inerzia.

2. **Elevata rigidezza:** Nel momento in cui avviene l'interazione tra l'utente ed il dispositivo meccanico, la percezione del contatto con oggetti dell'ambiente circostante deve essere la più realistica possibile. Supponiamo, in un caso limite, che la rigidezza di un oggetto possa essere infinita, il dispositivo utilizzato dovrà essere capace di riprodurre rigidezze sufficientemente elevate. Anche in questo caso, per garantire che tale condizione sia rispettata, si deve operare sulla componente meccanica dell'interfaccia aptica (quindi sulle trasmissioni e sulle parti strutturali), e su quella di controllo, imponendo che il guadagno d'anello sia elevato.

3. **Elevata banda passante:** Nel momento in cui avviene il passaggio da una fase di non contatto ad una fase di contatto, il dispositivo deve essere in grado di generare delle forze che abbiano una dinamica temporale elevata (cioè un alto contenuto spettrale), soprattutto nel caso in cui si abbia la necessità di simulare l'urto con oggetti rigidi.

Per garantire questa condizione è necessario agire elevando la frequenza appartenente alla componente meccanica del dispositivo.

Le componenti meccaniche che influiscono maggiormente sulle caratteristiche sopra citate sono gli attuatori, per cui possiamo tradurre tali condizioni in caratteristiche fondamentali per gli attuatori. Idealmente vorremmo avere attuatori che forniscano in uscita una forza indipendente dai movimenti del carico. Gli attuatori reali, invece, mostrano delle limitazioni, dovute soprattutto alla presenza di inerzie, di attriti e ad una banda passante limitata, che fanno sì che il loro comportamento si discosti da quello ideale. Per questo, al fine di garantire il corretto funzionamento del dispositivo, dobbiamo assicurarci che alcune condizioni siano rispettate:

- **Elevato rapporto forza/ingombro**
- **Elevato rapporto forza/peso**
- **Elevato rapporto forza/inerzia:** Nel caso in cui siano presenti inerzie, si crea una forza aggiuntiva, legata alle accelerazioni del carico. Dal momento che questa forza non deve essere avvertita dall'utente, si rende necessario compensarla, provocando un innalzamento nella complessità del controllo dell'attuatore nel caso in cui siano presenti inerzie rilevanti rispetto al carico applicato.
- **Elevato fattore dinamico:** Il fattore dinamico è definito come il rapporto tra la forza massima e la forza minima generabili dall'attuatore (la forza minima esercitabile dall'attuatore è causata dagli attriti interni che si determinano tra le componenti dell'attuatore in moto relativo). Nella progettazione degli attuatori, di solito, si tende a massimizzare il fattore dinamico, cercando, quindi, di minimizzare, per quanto possibile, la forza minima esercitabile.

- **Elevata banda passante:** La banda passante di un attuatore è la frequenza massima raggiungibile effettuando comunque un controllo di forza accurato. Questo valore dipende dal principio fisico scelto per la trasformazione da energia elettrica ad energia meccanica nonché dalle parti mobili dell'attuatore. Di solito gli attuatori utilizzati nelle interfacce aptiche sono dei servomotori elettromagnetici, che possono essere o interfacciati direttamente con il carico o usati insieme a riduttori. In entrambi i casi si presentano però degli inconvenienti.
- **Elevata controllabilità della forza:** È necessario effettuare un accurato controllo sulla forza generata affinché questa si avvicini maggiormente a quella desiderata.
- **Costi contenuti**

2.2.4 Classificazione delle interfaccie aptiche

In letteratura esistono più metodologie per la classificazione delle interfacce aptiche [15]. Una prima distinzione scaturisce dalla posizione della base, che permette di suddividere tali dispositivi in desktop e non-desktop. Appartengono alla prima categoria quelli a base fissa, che possono quindi operare dal piano di lavoro su cui sono posti. I dispositivi non-desktop sono, al contrario, quelli che non dispongono di una base fissa, ma, ad esempio, possono essere indossati. Hanno la peculiarità di poter disporre di molti gradi di libertà e si differenziano da quelli desktop perché, spesso, nel loro utilizzo non richiedono solo il movimento degli arti superiori dell'utente ma anche di quelli inferiori. Una seconda suddivisione può essere fatta analizzando il principio meccanico che ne determina il funzionamento. Possiamo trovare dispositivi a impedenza e dispositivi ad ammettenza, ma sono molto più diffusi i dispositivi aptici ad

impedenza.
Infine, può essere effettuata una suddivisione in base al numero di gradi di libertà di movimento o di forza che l'interfaccia aptica possiede. Il numero di gradi di libertà (*Degree of freedom* - **DOF**) è il numero di direzioni in cui il dispositivo è in grado di effettuare un movimento o applicare una forza.

Le interfacce aptiche trovano molteplici applicazioni. Senza dubbio, proprio per come sono state concepite, sono utilizzate in tutti i casi in cui il lavoro da compiere risulti essere rischioso sia per l'operatore che per le cose e le persone che si trovano nell'area di lavoro. Ad esempio, possono essere utilizzate nei casi in cui sia necessario manipolare materiale radioattivo nelle centrali nucleari, per installazioni sottomarine, per costruzioni spaziali e simulazioni chirurgiche.

Uno dei dispositivi più avanzati è il PHANToM [4]. Esso è dotato di sensori di forza ma anche di meccanismi di force feedback che permettono di restituire all'utente una forza programmata. Consiste di un braccio a 3 DOF di rotazione su cui è montata una penna come mano, con 3 ulteriori DOF (figura 2.17). Il braccio è in grado di fornire un ritorno di forza, per cui è possibile mediante software generare oggetti e superfici di vario genere da esplorare.

2.2.5 Come compensare le limitazioni dell'hardware aptico

Sono stati condotti alcuni esperimenti per testare la corrispondenza delle sensazioni visive con quelle tattili, e in uno di questi esperimenti i soggetti sono

Figura 2.17: L'interfaccia aptica a 6 DOF Sensable Phantom Omni.

stati posti di fronte ad un monitor che visualizzava due molle, modellate con costante elastica differente. I soggetti utilizzavano un Linear Grasper per 'comprimere' la molla, e a seconda della durezza della molla ricevevano un feedback visivo e uno tattile. In base a queste due sensazioni i soggetti dovevano riconoscere quale fosse la molla più dura, cosa che è avvenuta nell 100% dei casi. Quando però le molle sul monitor sono state scambiate all'insaputa dell'utente, la percentuale di risposte corrette è stata intorno allo 0%. Questo significa che l'uomo utilizza primariamente l'informazione visiva, mentre il tatto assume una funzione di supporto. E' un risultato importante, poiché nelle applicazioni in cui all'haptics è associata una immagine è possibile compensare le limitazioni dell'hardware aptico con opportune deformazioni grafiche degli oggetti, ottenendo comunque un effetto altrettanto realistico.

2.3 Simulatori fisici

Un simulatore fisico è un programma che simula i modelli fisici Newtoniani usando variabili come massa, velocità, forze di attrito ecc. Sono principalmente usati in simulazioni scientifiche e nei video giochi.
Ci sono principalmente due classi di simulatori fisici quelli in tempo reale e quelli ad alta precisione. I simulatori ad alta precisione richiedono più potenza di calcolo per calcolare precisamente i processi fisici e sono usualmente utilizzati in settori ingegneristici e nei film di animazione.
Nei videogiochi, o in altre forme di interazione con il computer, i simulatori fisici semplificano i lori calcoli e abbassano l'accuratezza, così da poter calcolare in tempo reale le informazioni garantendo un adeguato frame-rate per l'applicazione. Questi tipi di simulatori sono definiti simulatori in temporeale. Nei videogioci i simulatori fisici sono utilizzati per incrementare il realismo, cioé per calcolare realisticamente i movimenti dei solidi e le loro collisioni. I motori fisici migliori riescono a simulare realisticamente anche il comportamento dei fluidi e la loro interazione con gli oggetti circostanti.
I principali simulatori fisici attualmente disponibili sono Bullet, Open Dynamics Engine, Physics2D, Newton Game Dynamics, PhysX (conosciuto anche come Novodex), Havok, ecc.

Ad alto livello i simulatori sono divisi da due componenti principali, la componente di rilevazione delle collisioni e la componente di simulazione. In base all'interazione di questa architettura i simulatori vengono suddivisi in cinque paradigmi principali che sono:

- Metodo basato sui vincoli.
- Metodo basato sulle penalità.

- Metodo basato sugli impulsi.
- Metodo di sincronizzazione delle collisioni
- Ibridi

Inoltre le principali prestazioni dei simulatori fisici sono determinate da sei fattori essenziali:

Paradigma di simulazione Determina quali aspetti possono essere accuratamente simulati. Questi incidono sull'accuratezza nella risoluzione. Una presentazione generale dei paradigmi di simulazione è presentata da Erleben [16]. Mirtich [17] fornisce un confronto sui simulatori basati sui vincoli con quelli basati ad impulsi, mentre un confronto tra il metodo a penalità e quello a vincoli è presentata da Baraff [18].

L'integratore Determina l'accuratezza numerica della simulazione . Alcuni di questi metodi sono tratti da Baraff [19].

Rappresentazione degli oggetti Contribuisce all'efficenza e accuratezza delle collisioni nella simulazione. I vari aspetti per la scelta della rappresentazione degli oggetti è discussa da Ratcliff [20].

Individuazione delle collisioni e la determinazione dei contatti contribuiscono all'efficenza e alla precisione della collisione nella simulazione.

Proprietà dei materiali determina quale modello fisico di frizione applicare agli oggetti.

Implementazione dei vincoli determina come e quali giunti sono supportati e con quale precisione essi possono essere simulati si veda [16].

Molti simulatori fisici sono specializzati in particolari applicazioni. Nell'articolo "Evaluation of real-time physics simulation systems" [21] vengono confrontati sette differenti simulatori fisici: AGEIA PhysX (chiamato anche Novodex), Bullet Physics Library, JigLib, Newton Physics SDK, Open Dynamics Engine, Tokamak, True Axis Physics SDK. Nella figura 2.18 si possono visualizzare quattro tabelle che indicano le principali proprietà e caratteristiche di alcuni dei principali simulatori fisici.

Bullet è un simulatore ibrido che comprende sia il modulo ad impulsi che quello basato su vincoli, supporta un tempo di simulazione sia fisso che variabile e include un'implementazione per la GPU (graphics processing unit).

Engine	License	Cost : Edu/Com	Platform:PC	Platform:Console
AGEIA PhysX / Novodex	EULA	Free/Free	Win32/Linux/-	Xbox360/PS3/-
Bullet	Open, Zlib	Free/Free	Win32/Linux/Mac	Xbox360/PS3/-
JigLib	Open	Free/Free	Win32/Linux/-	-/-/-
Newton	EULA	Free/Free	Win32/Linux/Mac	-/-/-
Open Dynamics Engine	Open, LGPL/BSD	Free/Free	Win32/Linux/Mac	Xbox360/PS3/PSP
Tokamak	Open, BSD	Free/Free	Win32/Linux/-	-/-/-
True Axis	EULA	Free/Hobby+Full	Win32/Linux/Mac	Xbox360/-/-

Table 1 – Comparison of engines license and supported platforms

Engine	Corkscrew	Distance	Fixed	Generic (6D)	Prismatic	Revolute	Spherical	Universal	Vehicle
AGEIA PhysX / Novodex	n	y	y	y	y	y	y	n	y
Bullet	n	n	n	y	n	y	y	n	y
JigLib	y	y	y	y	n	y	y	n	y
Newton	y	n	n	y	y	y	y	y	y
Open Dynamics Engine	n	n	y	n	y	y	y	y	n
Tokamak	n	n	n	y	n	y	y	n	n
True Axis	n	n	n	y	y	y	y	n	y

Table 2 – Comparison of engines constraints support

Engine	Box	Capsule	Cylinder	Cone	Convex Mesh (Dynamic)	Compound Object	Heightfield (Static)	Plane	Sphere	Triangle Mesh (Static)
AGEIA PhysX / Novodex	y	y	n	n	y	y	y	y	y	y
Bullet	y	y	n	y	y	y	y	y	y	y
JigLib	y	y	n	n	n	y	y	y	y	y
Newton	y	y	y*	y	y	y	n	n	y	y
Open Dynamics Engine	[illegible]	y	y	n	n	y	y	y	y	y
Tokamak	y	y	n	n	y	y	n	n	y	y
True Axis	y	y	y	n	y	y	n	n	y	y

* Newton supports a Chamfer cylinder

Table 3 – Comparison of engines geometry support

Engine	Static Friction	Kinetic Friction	Restitution
AGEIA PhysX / Novodex	y	y	y
Bullet	y	n	y
JigLib	y	y	n
Newton	y	y	y
Open Dynamics Engine	y	y	y
Tokamak	y	n	y
True Axis	y	n	y

Table 4 – Comparison of engines material support

Figura 2.18: Comparazione tra i diversi simulatori fisici.

2.4 Strumenti hardware

La scelta della telecamera è molto importante, e per ottenere le migliori prestazioni bisogna tener conto di alcuni parametri principali, come la risoluzione, il frame-rate e la distorsione ottica. Un altro elemento che può influenzare la scelta della telecamera consiste nella sua compatibilità con i driver della piattaforma utilizzata. Alcune telecamere abilitano di default contrasto automatico che diminuisce le prestazioni, o offrono 25Hz di frame-rate con una diminuzione di qualità dell'immagine.

La scelta tradizionale è una telecamera USB o Firewire. Il frame-rate o color palette (RGB, YUV, compressione YUV) dipende principalmente dalla larghezza di banda della tecnologia. Una telecamera USB usa generalmente formati di compressione per la trasmissione come YUV:4:2:2 YUV:4:1:1 (perdita di compressione). Le telecamere firewire offrono migliori soluzioni, ma telecamere con palette color RGB complete sono generalmente costose (un formato di compressione VGA è una buona scelta). Le nuove telecamere USB 2.0 o IEEE1394b sono una buona scelta per progetti che utilizzano la libreria ARToolKit.

Il computer utilizzato nel progetto di tesi è un INTEL(R) Core(TM)2 Quad CPU Q9450 a 2,66 GHz con 3,25 GB di RAM. La telecamera utilizzata è una telecamera fire-wire dell'Unibrain più in specifico la Fire-i400.

2.4.1 Strumento aptico utilizzato

Per le attività di questa tesi è stato utilizzato il dispositivo Falcon prodotto da Novint mostrato in figura 2.19, ideato per essere un nuovo 'game controller'. Esso si caratterizza per i suoi 3 DOF e per il costo limitato, nonostante la buona sensorialità fornita. Le sue caratteristiche principali sono:

1. **3D Touch Workspace:** 10.16cm × 10.16cm × 10.16cm
2. **Massima forza disponibile:** 8.89 Newton
3. **Risoluzione posizione:** 400 dpi
4. **Frame Rate:** 1 KHz

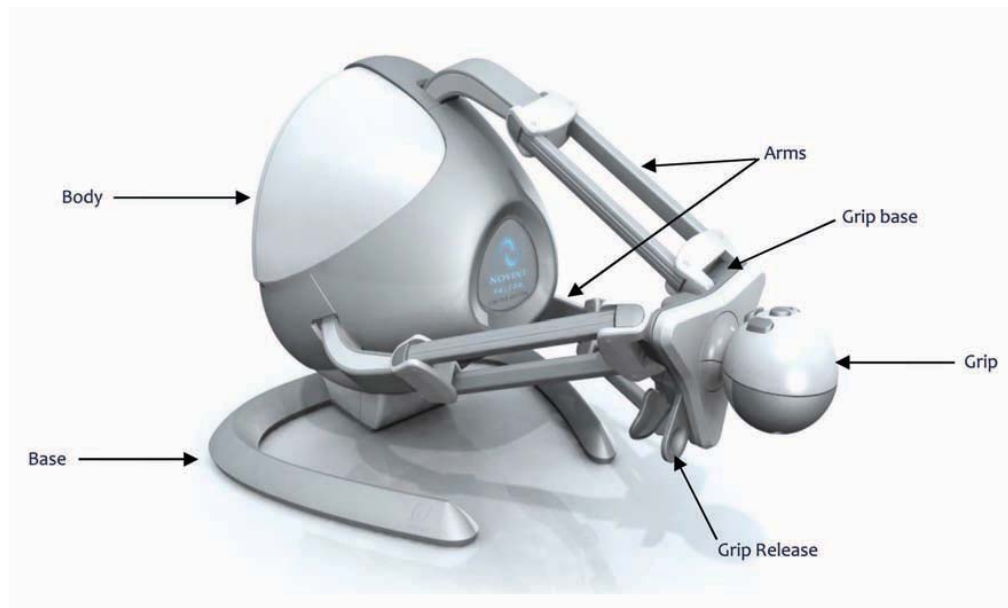


Figura 2.19: Il dispositivo Novint Falcon.

Capitolo 3

Architettura del sistema

In questo capitolo della tesi si illustrano i moduli che compongono l'architettura del sistema realizzato, mostrando anche le principali parti del codice implementato. L'architettura del sistema si struttura tramite l'interazione dei visibili in figura 3.1. Per i moduli ArtoolKit e OpenGL si rimanda ai

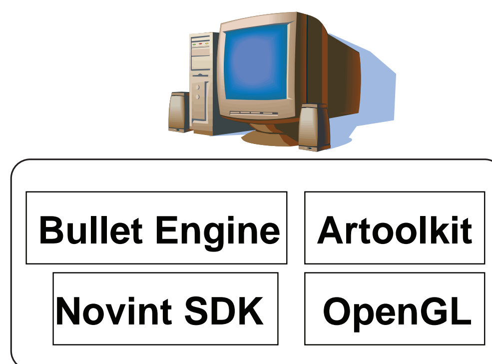


Figura 3.1: Moduli del sistema realizzato.

paragrafi 2.1.2 e 2.1.3, mentre i rimanenti moduli vengono illustrati successivamente in questo capitolo.

Un'ulteriore e complessa parte del progetto è quella che realizza la corrispondenza tra mondo reale e virtuale. Essa necessita uno studio approfondito per quanto riguarda la formazione dell'immagine da parte di un programma di computer graphics in relazione a quella acquisita da una telecamera. Infatti la perfetta corrispondenza di prospettiva tra il mondo reale e quello ricreato virtualmente necessita di parametri ben precisi. La figura 3.2 mostra sia l'ambiente reale sia quello riprodotto virtualmente, ma come si può notare le due immagini presentano prospettive diverse, e questo non permette la creazione di un ambiente di realtà aumentata. Infatti la corrispondenza di prospettiva è indispensabile per una buona visualizzazione degli oggetti creati virtualmente.

(a) Ambiente reale.

(b) Ambiente virtuale.

Figura 3.2: Ambiente reale e virtuale.

3.1 Corrispondenza tra ambiente reale e virtuale

Questo paragrafo descrive il processo che ha portato ad ottenere la coerenza tra l'immagine visualizzata dalla telecamera e gli oggetti di realtà aumentata presenti nella scena.
Per prima cosa è necessario che la telecamera venga calibrata per poter estrapolare i suoi parametri intrinseci ed estrinseci, e successivamente si devono andare a studiare i processi di formazione di un' immagine sul monitor da parte di una telecamera reale e di un programma di computer grafica. Nella figura 3.3 si puo vedere che sia la prospettiva che le dimensioni degli oggetti virtuali sono coerenti con quelli degli oggetti reali inquadrati dalla telecamera.
Nelle sezioni seguenti vengono descritti questi tre passi fondamentali. Inoltre viene riportato il procedimento matematico che porta a stabilire una corretta corrispondenza tra i due ambienti. Per finire, vengono esposti i risultati quantitativi ottenuti.

3.1.1 Calibrazione della telecamera

In questo paragrafo viene rappresentata l'operazione di calibrazione della telecamera. In un sistema di scansione 3D calibrare una telecamera significa estrarre una serie di parametri caratteristici che sono suddivisi in due classi: parametri intrinseci e parametri estrinseci.
Tale operazione è fondamentale perché bisogna cercare di minimizzare la deformazione del segnale originale (che può manifestarsi per molte ragioni). Nel caso in esame è molto importante controllare questi difetti, in quanto

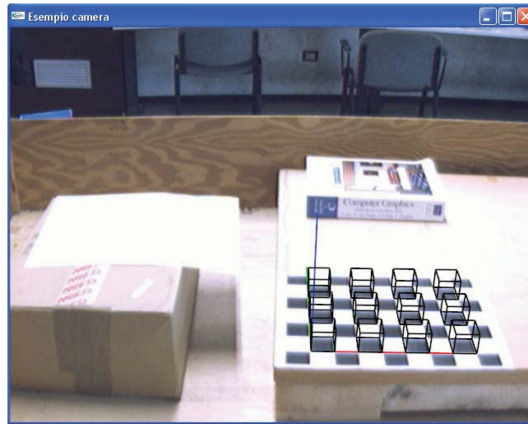


Figura 3.3: Esempio di realtà aumentata coerente con l'ambiente reale.

essi possono causare errori di allineamento che non permettono all'immagine ottenuta di coincidere con l'originale.
Per prima cosa bisogna considerare la distorsione della lente che, per sua costruzione, tende a generare immagini con qualità più bassa ai bordi rispetto che al centro, provocando una prima perdita di qualità nel momento di produzione dell'immagine. Altre distorsioni possono essere generate dal rumore elettrico, da perdite di informazione al momento della digitalizzazione, ecc. Per correggere queste distorsioni è necessario ricorrere alla calibrazione della telecamera, che permette di determinare le caratteristiche geometrico-ottiche interne ad essa, la posizione spaziale dell'oggetto telecamera, e l'orientamento della telecamera rispetto ad un sistema di coordinate. In altre parole, la calibrazione consiste nello stabilire una corrispondenza fra i punti dell'oggetto e quelli dell'immagine, secondo una funzione il più simile possibile ai principi di lavoro della telecamera.

Il processo di calibrazione si può eseguire in vari modi[1]. Esistono algorit-

[1] La calibrazione di una telecamera stazionaria si effettua mediante un sistema fisso di coordinate e un reticolo di punti aventi coordinate note nel sistema universale.

mi che permettono di ottenere le caratteristiche direttamente, a partire dalla conoscenza delle coordinate del punto dello spazio e della sua proiezione nell'immagine; altri si basano su procedure che considerano per prima cosa la matrice di proiezione, e poi i parametri intrinseci ed estrinseci a partire da questa matrice. Nel caso, dobbiamo ricordare che esistono classi di algoritmi diversi, in funzione della disposizione spaziale dei punti; questi ultimi, infatti, possono formare una matrice tridimensionale, oppure essere considerati sullo stesso piano.
Alcuni fenomeni, che si presentano al momento dell'acquisizione, rendono difficoltosa questa estrazione. Fra loro i piu significativi sono: le aberrazioni ottiche, la risposta non uniforme delle telecamere CCD, l'illuminazione, il rumore, ecc.

3.1.1.1 Parametri Intrinseci

I parametri intrinseci sono i parametri necessari a collegare le coordinate di un pixel dell'immagine con le coordinate corrispondenti nel sistema di riferimento della telecamera. Sono quindi necessari a specificare le caratteristiche ottiche, geometriche e digitali della telecamera (intesa come sensore). Essi sono:

- **Lunghezza focale:** La lunghezza focale in pixel è memorizzata in un vettore **fc** 2×1.
- **Centro dell'immagine:** Il centro dell'immagine in punti di coordinate è memorizzata in un vettore **cc** 2×1.
- **Coefficiente di Skew:** Il coefficiente di skew definisce l'angolo tra gli assi x e y memorizzati nel valore **alpha_c**.

- **Distorsione:** I coefficienti di distorsione (radiale e tangente) sono memorizzati in un vettore **kc** 5×1 (questa distorsione è causata dall'ottica).

I parametri intrinseci ci permettono di collegare le coordinate x_i, y_i, (espresse in pixel) di un punto dell'immagine, con le coordinate dello stesso punto nel sistema di riferimento della telecamera. Con il metodo di calibrazione utilizzato arriviamo ad ottenere una matrice che definisce tutti questi parametri:

$$KK = \begin{pmatrix} fc(1) & alpha_c * fc(1) & cc(1) \\ 0 & fc(2) & cc(2) \\ 0 & 0 & 1 \end{pmatrix}$$

3.1.1.2 Parametri estrinseci

I parametri estrinseci definiscono la posizione e l'orientamento del sistema di riferimento della camera rispetto al riferimento mondo, che è supposto noto. Questi parametri sono:

- **Una matrice di rotazione** 3×3
- **Una vettore di traslazione** 3×1

3.1.2 Risultati della Calibrazione

La calibrazione è stata eseguita con un toolbox di Matlab (GML MATLAB Camera Calibration Toolbox), e sono state utilizzate in totale sedici immagini. All'interno di queste immagini è presente un pattern regolare, che nel caso specifico è rappresentato da una scacchiera. In ognuna delle figure esso è stato spostato in modo tale da essere inquadrato dalla telecamera con una

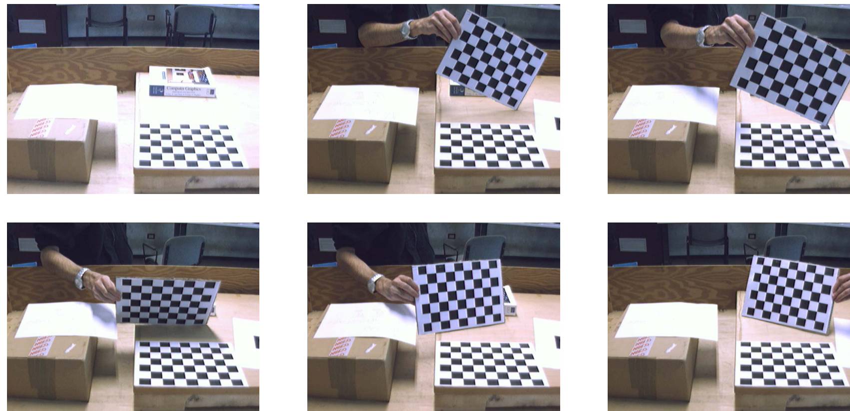


Figura 3.4: Figure utilizzate per la calibrazione della telecamera.

orientazione diversa. La figura 3.4 mostra alcune delle 16 immagini utilizzate. Le immagini vengono elaborate dal toolbox con l'aiuto dell'utente, e alla fine dalla procedura si ottengono i parametri intrinseci ed estrinseci della telecamera. La procedura di calibrazione è stata applicata ad una telecamera posizionata su un treppiedi ad una distanza di circa un metro dalla scena inquadrata. Sono ottenuti in particolare i seguenti risultati:

$$\begin{aligned}
\text{Principal point: } & cc = [319.500 \quad 239.500] \pm [0.000 \quad 0.000] \\
\text{Skew: } & alpha_c = [0.000] \pm [0.000] \\
& \quad => \text{angle of pixel axes } = 90.000 \pm 0.000 \quad \text{degrees} \\
\text{Distortion: } & kc = [0.000 \quad 0.000 \quad 0.000 \quad 0.000 \quad 0.000] \\
& \quad \pm [0.000 \quad 0.000 \quad 0.000 \quad 0.000 \quad 0.000] \\
\text{Pixel error: } & err = [0.508 \quad 0.350]
\end{aligned}$$

Con questi parametri intrinseci sono stati calcolati i parametri estrinseci della telecamera, ottenendo la figura 3.5 e i seguenti parametri:

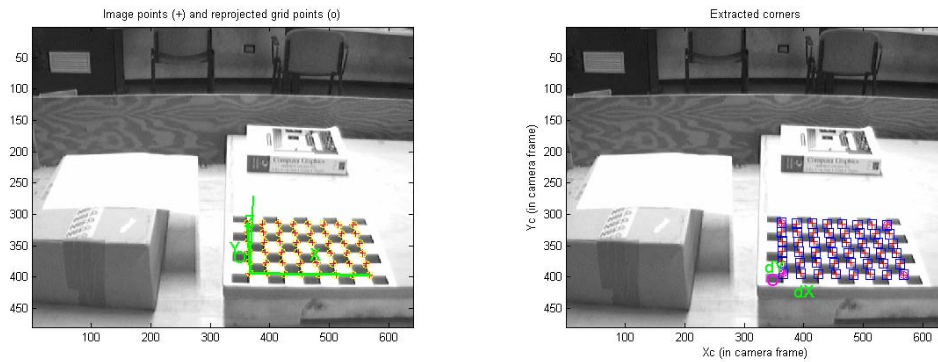


Figura 3.5: Calcolo dei parametri estrinseci della telecamera.

$$\text{Translation vector: } Tc_ext = [48.219 \quad 160.235 \quad 1003.358]$$

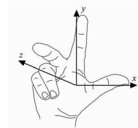
$$\text{Rotation vector: } omc_ext = [2.081 \quad 0.016 \quad 0.007]$$

$$\begin{aligned}\text{Rotation matrix: } Rc_ext = [&0.999 \quad 0.009 \quad 0.012\\ &0.015 \quad -0.489 \quad -0.872\\ &-0.002 \quad 0.872 \quad -0.489]\end{aligned}$$

$$\text{Pixel error: } err = [0.693 \quad 0.426]$$

3.1.3 Modello di una telecamera reale

La telecamera reale ha un sistema di coordinate 'right-hand' (secondo la convenzione della mano destra[2]. La direzione della visuale si trova lungo la direzione $+z$ $(0,0,1)$ come viene mostrato in figura3.6.

[2]Il nome deriva dalla possibilità di utilizzare le prime tre dita della mano destra per ricordare la disposizione reciproca dei tre assi coordinati: il pollice è l'asse delle x l'indice l'asse delle y ed il medio l'asse delle z. 

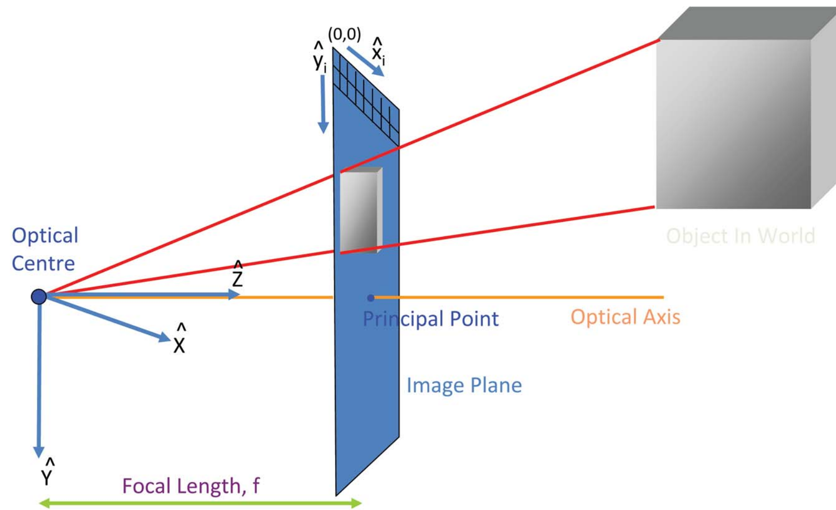


Figura 3.6: Modello della telecamera reale.

Il punto 3D proiettato su un' immagine viene ricavato seguendo la procedura illustrata in figura 3.7.

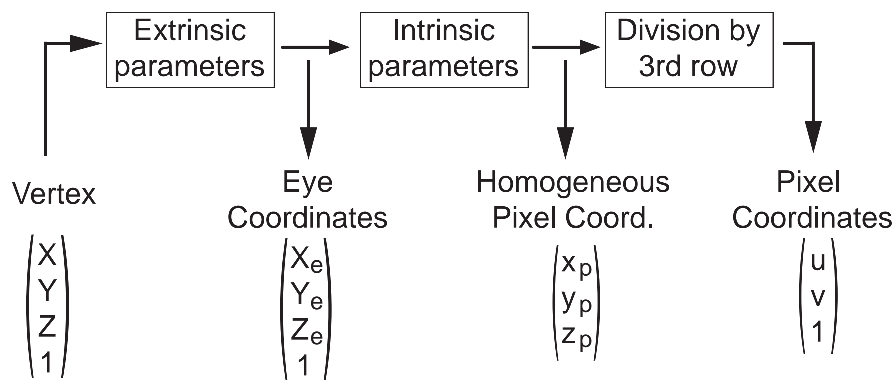


Figura 3.7: Processo per ottenere un' immagine da una telecamera reale.

Quindi $x = PX_w$ dove $X = (X, Y, Z, 1)^T$ e $x = (U_{real}, V_{real}, 1)^T$.

Un punto nello spazio $X = (x, y, z)$ è mappato in un punto sul piano dell'immagine corrispondente all'intersezione della retta passante per il punto X ed il centro di proiezione con il piano dell'immagine [22]. Si ha quindi una trasformazione dallo spazio Euclideo 3D allo spazio Euclideo 2D cioé:

$$(X, Y, Z)^T \rightarrow (f\tfrac{X}{Z}, f\tfrac{Y}{Z})$$

Si hanno tre diversi sistemi di riferimento:

1. il sistema di riferimento 3D detto anche sistema "mondo";
2. il sistema di riferimento 3D della telecamera centrato in C;
3. il sistema di riferimento 2D per l'immagine.

La matrice di proiezione P è:

$$P = K \left[\begin{array}{c|c} R & T \end{array} \right] = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \left[\begin{array}{ccc|c} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{array} \right]$$

questa matrice è detta *matrice di proiezione prospettica.* Rappresenta il modello geometrico della telecamera.
La matrice $[R|T]$ contiene i parametri estrinseci della telecamera. Essa incorpora sia la rotazione che la traslazione, e quindi identifica univocamente le trasformazioni tra il sistema di riferimento della camera, non noto, e quello del mondo, noto.
Nella matrice K si identificano[3] quattro parametri: f_x e f_y rappresentano la lunghezza focale della camera in termini della dimensione dei pixel lungo la x e la y, mentre u_0 e v_0 rappresentano le coordinate del punto principale (che

[3]Generalmente sono cinque ma si è considerato il parametro di skew= 0.

non è necessariamente nel centro). Questa matrice identifica quindi tutti i parametri intrinseci della telecamera, ed è detta pertanto 'intrinsic matrix'. Grazie a questa matrice è possibile collegare le coordinate espresse in pixel di un punto dell'immagine con le coordinate dello stesso punto nel sistema di riferimento della telecamera. Moltiplicando P e X si ottiene il punto x proiettato:

$$X = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_e \\ Y_e \\ Z_e \end{bmatrix} = \begin{bmatrix} f_x \frac{X_e}{Z_e} + u_0 \\ f_y \frac{Y_e}{Z_e} + v_0 \\ 1 \end{bmatrix}$$

Il punto proiettato ha la seguenti coordinate:

$$U_{real} = f_x \frac{X_e}{Z_e} + u_0$$
$$V_{real} = f_y \frac{Y_e}{Z_e} + v_0$$

3.1.4 Creazione dell'immagine eseguita da un programma di computer graphics

Il processo di visione in 3D è inerentemente molto più complesso di quello in 2D. In 2D si ha solamente la necessità di specificare una finestra sul mondo 2D e una viewport sulla superficie (2D) del dispositivo. Concettualmente gli oggetti nel mondo dell'applicazione devono solo eseguire un processo di clipping sulla window, e poi sono trasformati nella viewport sul display applicando la trasformazione composta window-to-viewport. L'aumento di complessità passando da due a tre dimensioni è dovuto a due fattori: la dimensione in più e l'assenza di dispositivi di output 3D. Per ovviare a questo inconveniente si introducono trasformazioni di proiezione, che mappano oggetti 3D su di un piano di proiezione 2D.

Per definire la vista in 3D, quindi, entrano in gioco molti più parametri che in 2D: il volume di vista nel mondo (che racchiude tutte e sole le primitive, o la parte di esse, visibili), la trasformazione di proiezione e la viewport sul piano. Gli oggetti, dopo aver eseguito la fase di clipping sul volume di vista, sono trasformati (mappati) sulla viewport del display. Il modello concettuale del processo di visione in 3D, quindi, è mostrato nella figura 3.8.

Il processo di formazione dell'immagine di sintesi in 3D consta in una sequenza di operazioni:

- Definizione della trasformazione di proiezione (il modo di mappare informazioni 3D su un piano immagine 2D).
- Definizione dei parametri di vista (punto di vista, direzione di vista, etc.).
- Clipping in 3D (i parametri di vista individuano un volume di vista; occorre rimuovere le parti della scena esterne a tale volume).
- Trasformazione di proiezione e visualizzazione della scena (con trasformazione "window-to-viewport" finale).

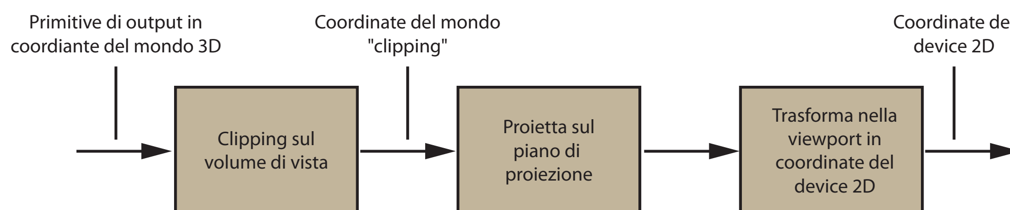


Figura 3.8: Il processo di Visione in 3D.

3.1.4.1 Il modello synthetic-camera

I fondamenti concettuali della computer graphics tridimensionale sono basati su modelli di sistemi ottici. Il processo di formazione delle immagini gener-

ate dal computer viene assimilato al processo di formazione di un'immagine da parte di un sistema ottico, quale ad esempio una macchina fotografica. Questo paradigma è chiamato Synthetic-Camera Model.

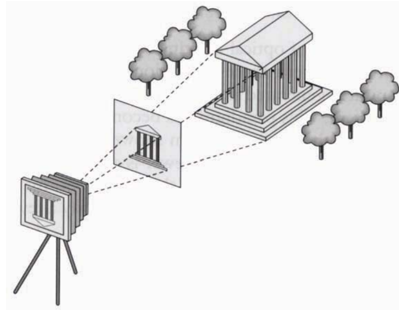


Figura 3.9: Modello di formazione dell'immagine.

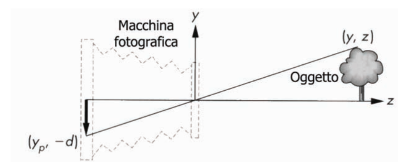


Figura 3.10: Formazione immagine in una macchina fotografica.

Si consideri l'immagine di figura 3.9 dove sono presenti una serie di oggetti ed un osservatore, in questo caso rappresentato dalla macchina fotografica. Per poter emulare questo processo in un sistema grafico, al fine di produrre immagini di sintesi, dobbiamo identificare una serie di principi di base. In primo luogo la definizione del modello, nel mondo dell'applicazione, è indipendente dalla posizione dalla quale l'utente deciderà di osservare la scena, così come gli oggetti nel mondo reale sono indipendenti dalle fotografie che se ne scattano. In un sistema grafico, quindi, ci si può aspettare che funzioni separate siano destinate alla modellazione (definizione del mondo dell'applicazione) ed al posizionamento della macchina (dell'osservatore) all'interno della scena.

Inoltre, si possono formalizzare le operazioni di generazione delle immagini utilizzando dei calcoli trigonometrici relativamente semplici. Consideriamo una vista laterale di un modello semplice (figura 3.10). I raggi luminosi, attraversando l'obiettivo, impressionano la pellicola riproducendo un'immag-

ine (ruotata di 180°) dell'oggetto fotografato. La dimensione dell'oggetto che stiamo fotografando come sarà nell'immagine (y) è proporzionale all'oggetto reale e può essere facilmente calcolata data la distanza focale d: $y_{p'} = \frac{y}{z/d}$ (questo grazie alla similitudine dei triangoli).
Se invece viene spostato il piano su cui si forma l'immagine davanti alla macchina si lavora con un modello proiettivo più semplice (figura 3.11). Si supponga quindi di portare il piano di proiezione davanti alla lente della telecamera.

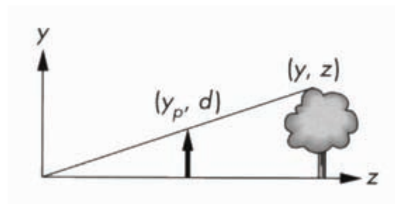


Figura 3.11: Processo di Visione.

Si deve inoltre tener conto della limitata dimensione dell'immagine, posizionando una finestra sul piano dell'immagine; su di essa (o tramite le informazioni da essa descritte) verrà fatto il clipping delle primitive tenendo solo quelle visibili interamente o in parte (figura 3.12).

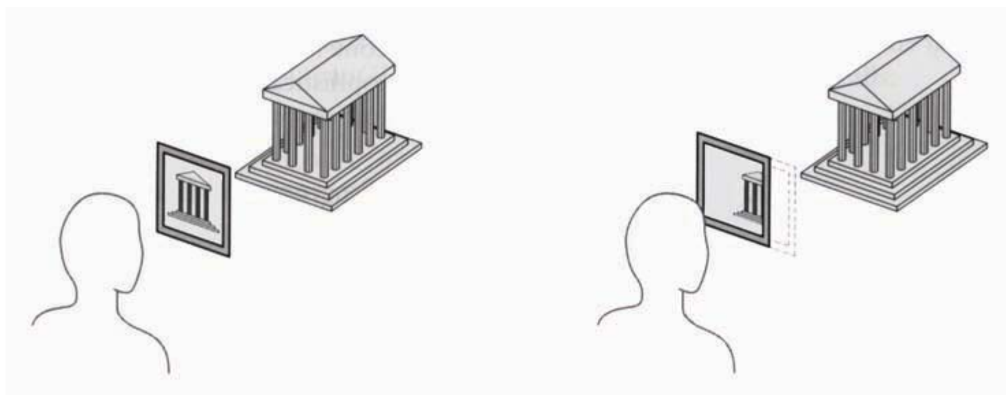


Figura 3.12: Operazione di clipping con due diverse disposizioni della finestra.

3.1.4.2 Proiezioni

Si dice proiezione una trasformazione geometrica avente il dominio in uno spazio di dimensione n ed il codominio in uno spazio di dimensione n-1 (o minore). In computer graphics le trasformazioni di proiezione utilizzate sono quelle dallo spazio 3D (il mondo dell'applicazione) al 2D (la superficie del dispositivo di output). Da un punto di vista geometrico, la proiezione è definita per mezzo di un insieme di proiettori aventi origine in un unico centro, passanti per tutti i punti dell'oggetto considerato ed intersecanti un piano. La proiezione di un segmento è a sua volta un segmento, e non è quindi necessario calcolare i proiettori di tutti i punti di una scena, ma solo quelli relativi ai vertici delle primitive che la descrivono.
Le proiezioni sono caratterizzate da: proiettori rettilinei e proiezioni giacenti su un piano, che prendono il nome di proiezioni geometriche piane.

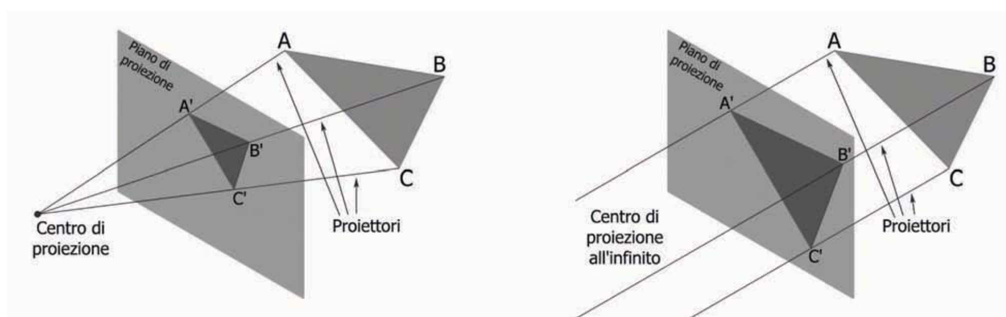


Figura 3.13: Proiezioni prospettiche.

Le proiezioni parallele prendono il nome dai proiettori che sono, appunto, tra loro tutti paralleli, mentre per quelle prospettiche si specifica un centro di proiezione. Nel primo caso si parla invece di direzione di proiezione.
La proiezione prospettica è più realistica di quella parallela in quanto riproduce una visione reale (gli oggetti appaiono di dimensione decrescenti via

via che ci si allontana dall'osservatore). Inoltre in essa ogni insieme di linee parallele (non al piano di proiezione) converge in un punto, detto punto di fuga, e se l'insieme di linee è parallelo ad uno degli assi coordinati tale punto viene detto principale.

3.1.4.3 Creazione dell'immagine con OpenGL

Come già accennato, una vista in tre dimensioni non è definita solo dalla proiezione ma anche dal volume di vista, ovvero dalla regione dello spazio tridimensionale che include tutte e sole le primitive visibili. La definizione dei parametri di una vista sintetica si basa ancora sul modello della macchina fotografica virtuale (synthetic-camera), ed è analoga a quella che si farebbe per fare una fotografia con una macchina reale. È possibile mettere in evidenza quattro passi principali e le corrispondenti analogie (figura 3.14).
Si può schematizzare il processo di scattare una fotografia nei seguenti passi:

- disporre la macchina e puntarla verso la scena,
- disporre gli oggetti da fotografare nella scena,
- scegliere una lente o aggiustare lo zoom,
- scegliere la dimensione della foto,

Prendendo in esame il metodo utilizzato da OpenGL per generare una vista di una scena sintetica mediante computer graphics, le quattro azioni corrispondono a definire, rispettivamente, le seguenti quattro trasformazioni:

- viewing transformation (posizionamento della camera nello spazio 3D),
- modelling transformation (modellare la scena),

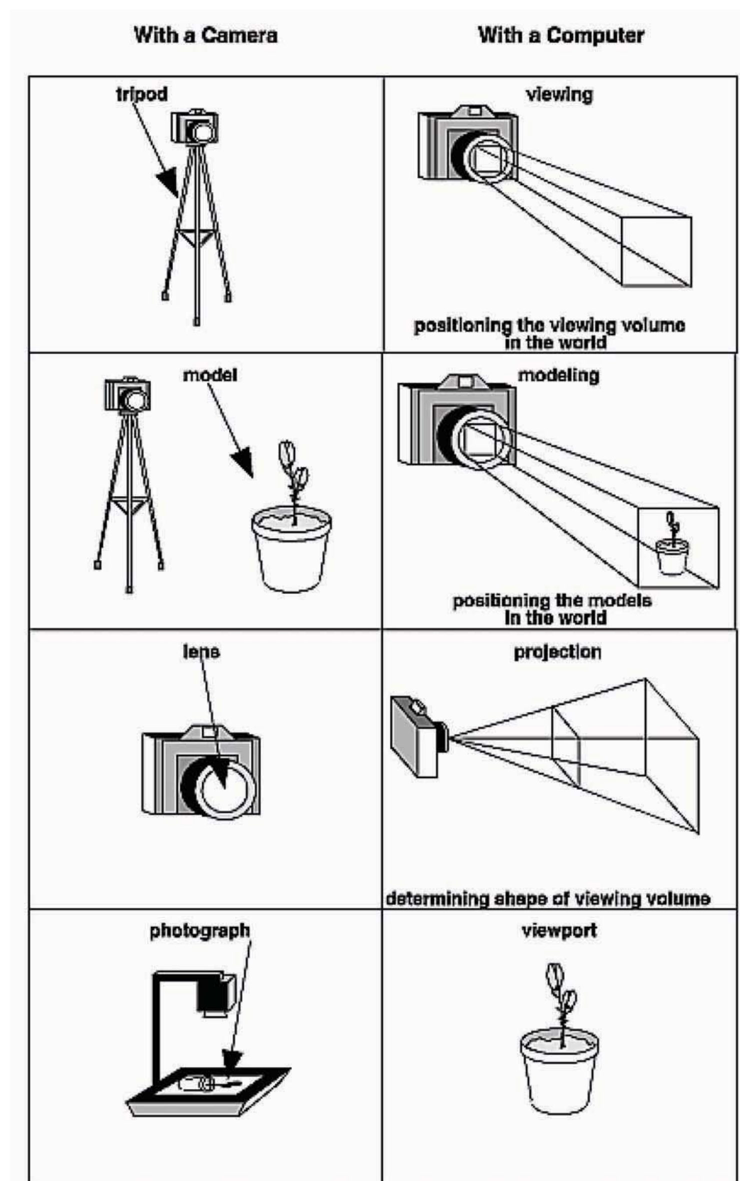


Figura 3.14: Camera Analogica.

- projection transformation (scelta del tipo di proiezione e della lente),
- viewport transformation (dimensionamento dell'immagine finale, glViewport).

In OpenGL tutte le trasformazioni necessarie a produrre la scena desiderata avvengono tramite l'applicazione di matrici agli oggetti che compongono la scena, e vengono effettuati quindi alcuni passaggi mostrati nella figura 3.15.

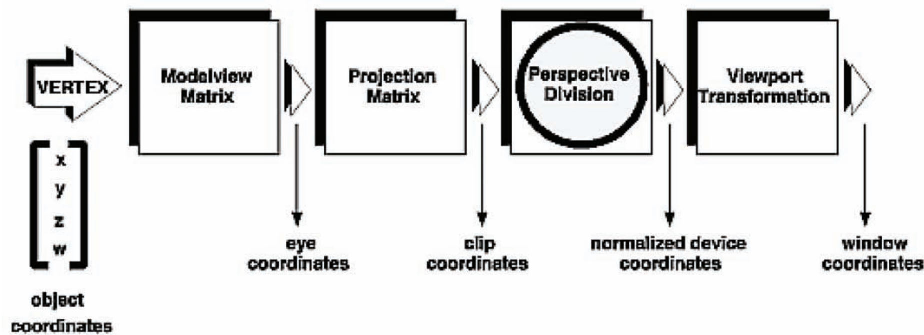
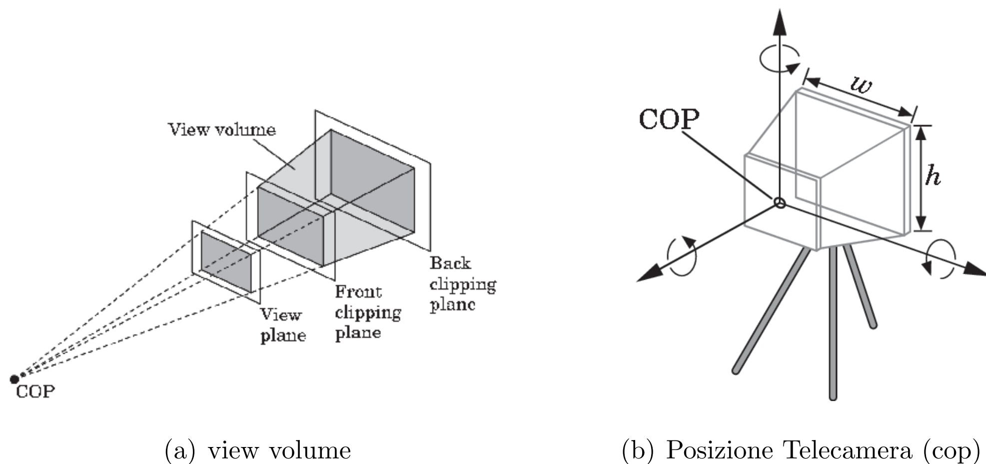


Figura 3.15: Pipeline OpenGL.

- Object coordinates: sistema di riferimento solidale con l'oggetto e utilizzato principalmente per la sua costruzione.
- Eye coordinates: sistema di riferimento "virtuale" e fisso, in riferimento al quale vengono descritte posizione e orientazione di tutti gli oggetti. Si chiama così perché è solidale con l'osservatore.
- Normalized device coordinates: coordinate normalizzate rispetto alle dimensioni della finestra.
- Window coordinates: coordinate relative alla finestra attuale.
- Device coordinates: coordinate relative allo schermo.

La matrice di trasformazione Modelview trasforma le object cordinates in eye coordinates, e viene modificata ad ogni trasformazione geometrica della scena. E' usata inoltre per definire il punto di vista, dal momento che le trasformazioni geometriche che lo determinano sono le medesime usate per modellare la scena. Invece la matrice di *projection* definisce il tipo di proiezione e trasforma un vertice dalle eye coordinates alle clip coordinate. Quindi con essa si va a modificare il volume di vista, cioè quello spazio all'interno del quale gli oggetti saranno visualizzati e proiettati sulla viewport secondo la particolare proiezione definita. Per una proiezione prospettica il viewing volume è un tronco di piramide chiamato frustum (figura 3.16).



(a) view volume (b) Posizione Telecamera (cop)

Figura 3.16: Tronco di piramide visualizzato dalla telecamera.

Per la telecamera virtuale il modello OpenGL vede la scena nella direzione $-Z$, cioé l'osservatore è posto nell'origine ed è rivolto nella direzione negativa dell'asse z come mostrato in figura 3.17.

Assunto quindi il sistema di riferimento della telecamera con il centro di proiezione nell'origine e il piano di proiezione a distanza d lungo l'asse $-Z$,

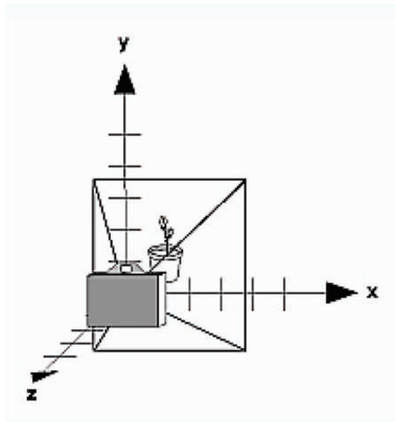


Figura 3.17: Sistema di riferimento della telecamera.

si vuole trovare la proiezione sul piano di proiezione di un punto. Per fare questo consideriamo i due modi che OpenGL mette a disposizione per creare una vista prospettica, che risulta piu realista e si avvicina maggiormente a quella realmente visualizzata da una telecamera reale.
Per trovare la proiezione di un punto sul piano di proiezione, si utilizza OpenGL che, per creare una vista prospettica, e quindi simile a quella reale, mette a disposizione due funzioni: glFrustrum e gluPerspective. Con entrambi si possono ottenere gli stessi risultati. Nel progetto si è deciso di utilizzare la gluPerspective, in cui il punto di vista è posizionato nell'origine e punta alle z negative e l'AspectRatio è tale per cui sono eliminate le distorsione sulla viewport (l'aspect ratio del frustum coincide con quello della viewport).
La funzione (figura 3.18) è così definita:

$$\text{gluPerspective}(fovy, aspect, Near, Far)$$

Aspect Ratio definisce il rapporto tra la larghezza e l'altezza dell'immagine (w/h), mentre il fovy è l'angolo sotto il quale viene vista la scena nel piano xz, ed è compreso tra 0° e 180° gradi.
Avendo quindi un punto nelle eye coordinates, OpenGL lo trasformerà nelle

clip coordinates nel seguente modo:

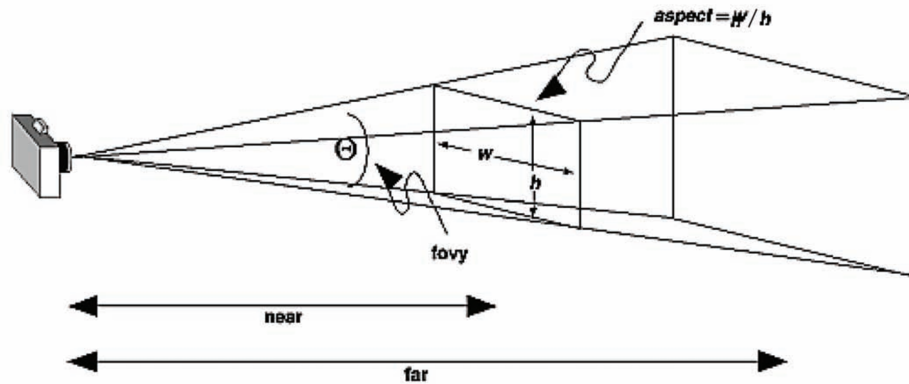


Figura 3.18: Defizione del view volume utilizzando la funzione perspective.

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \\ W_c \end{pmatrix} = \begin{pmatrix} \frac{\cot(fovy/2)}{aspect} & 0 & 0 & 0 \\ 0 & \cot(fovy/2) & 0 & 0 \\ 0 & 0 & \frac{Far+Near}{Near-Far} & \frac{2*Far*Near}{Near-Far} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} X_e \\ Y_e \\ Z_e \\ 1 \end{pmatrix}$$

OpenGL si occuperà internamente anche del 'Perspective Division', che consiste in una possibile deformazione "shear" per rendere la linea centrale del volume di vista parallela all'asse z, considerato con la variabile s.
In seguito bisogna considerare l'ultima operazione "Viewport Transformation"[4] in OpenGL che esegue tramite la seguente funzione:

glViewport(x_0, y_0, *width*, *height*)

Questa trasformazione converte un vertice da clip coordinate a window coordinates (s è il fattore di scala descritto precedentemente).

[4]Trasfomazione window to viewport: in OpenGL la finestra nelle coordinate mondo ha dimensione $x_{min} = -1$, $x_{max} = 1$, $y_{min} = -1$, $y_{max} = 1$.

$$s\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{width}{2} & 0 & \frac{width}{2}+x_0 \\ 0 & \frac{height}{2} & \frac{height}{2}+y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix}$$

Infatti:

$$\begin{pmatrix} \frac{width}{2} & 0 & \frac{width}{2}+x_0 \\ 0 & \frac{height}{2} & \frac{height}{2}+y_0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{width}{2} & 0 & 0 \\ 0 & \frac{height}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

La matrice di modelview corrisponde ai parametri estrinseci della telecamera reale, è una matrice 4×4 e la sua quarta riga è $(0, 0, 0, 1)$.

Possiamo scrivere un punto nelle window coordinates partendo da un punto nelle eye coordinates come:

$$s\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{width}{2} & 0 & \frac{width}{2}+x_0 \\ 0 & \frac{height}{2} & \frac{height}{2}+y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{\cot(fovy/2)}{aspect} & 0 & 0 & 0 \\ 0 & \cot(fovy/2) & 0 & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} X_e \\ Y_e \\ Z_e \\ 1 \end{pmatrix}$$

Infine le coordinate in pixel in OpenGL di un punto nel mondo: ponendo $s = -Z_e$, $Near = 0$ e $Far = 1$,si ottengono:

$$U_{virtual} = \frac{\cot(fovy/2)}{aspect}\left(\frac{-X_e}{Z_e}\right)\frac{width}{2} + \frac{width}{2} + x_0$$
$$V_{virtual} = \cot(fovy/2)\left(\frac{-Y_e}{Z_e}\right)\frac{height}{2} + \frac{height}{2} + y_0$$

3.1.5 Corrispondenza tra i due modelli di telecamera

Intuitivamente si può pensare che U_{real} sia uguale a $U_{virtual}$ e V_{real} eguagli $V_{virtual}$, tuttavia esistono alcuni elementi da dover tenere in considerazione. Per prima cosa la direzione di vista iniziale della reale telecamera è opposta a quella virtuale: la telecamera reale vede $(0, 0, 1)$, quella virtuale $(0, 0, -1)$. Per fare il confronto inseriamo una matrice di rotazione prima di eseguire la matrice di projection, così da avere l'asse x e l'asse delle z allineati con il sistema di OpenGL.

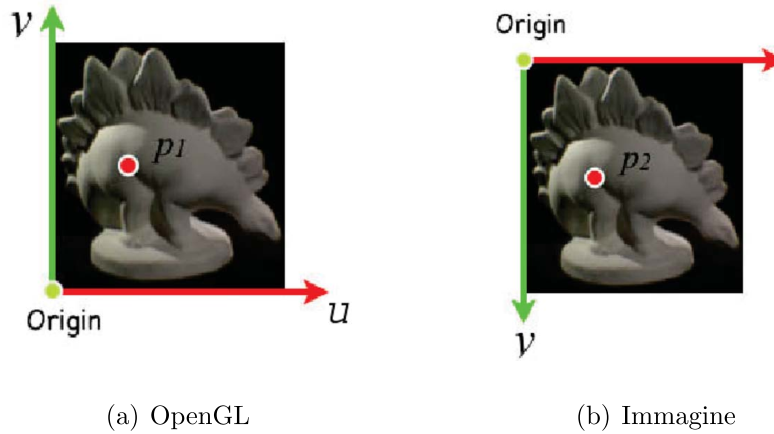
$$s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{width}{2} & 0 & \frac{width}{2} + x_0 \\ 0 & \frac{height}{2} & \frac{height}{2} + y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{\cot(fovy/2)}{aspect} & 0 & 0 & 0 \\ 0 & \cot(fovy/2) & 0 & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix}$$
$$\begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_e \\ Y_e \\ Z_e \\ 1 \end{pmatrix}$$

Quindi le coordinate di un vertice 3D proiettato con OpenGL, applicando la matrice di rotazione ponendo $s = -Z_e$, sono:

$$U'_{virtual} = \frac{\cot(fovy/2)}{aspect} \frac{width}{2} \left(\frac{X_e}{Z_e} \right) + \frac{width}{2} + x_0$$
$$V'_{virtual} = -\cot(fovy/2) \left(\frac{Y_e}{Z_e} \right) \frac{height}{2} + \frac{height}{2} + y_0$$

A questo punto occorre considerare che le coordinate del pixel tra OpenGL e quelle dell'immagine reale sono differenti, come mostrato nella figura 3.19. Perché l'origine è differente bisogna convertire il punto *P1* e *P2* nel seguente modo:

$$p_{2_y} = height - p_{1_y}$$



(a) OpenGL (b) Immagine

Figura 3.19: Sistema di riferimento delle immagini.

Equagliando i due risultati:

$$f_x \frac{X_e}{Z_e} + u_0 = \frac{\cot(fovy/2)}{aspect} \frac{width}{2} \left(\frac{X_e}{Z_e} \right) + \frac{width}{2} + x_0$$

$$V_{real} = height - V'_{virtual}$$

$$f_y \frac{Y_e}{Z_e} + v_0 = height + \cot(fovy/2) \frac{height}{2} \left(\frac{Y_e}{Z_e} \right) - \frac{height}{2} + y_0$$

In fine si ottengono i parametri per il rendering in OpenGL

$$x_0 = u_0 - \frac{width}{2}$$

$$y_0 = \frac{height}{2} - v_0$$

$$fovy = 2 * \cot^{-1} \left(\frac{2f_y}{height} \right)$$

$$aspect = \frac{width}{height} \frac{fy}{fx}$$

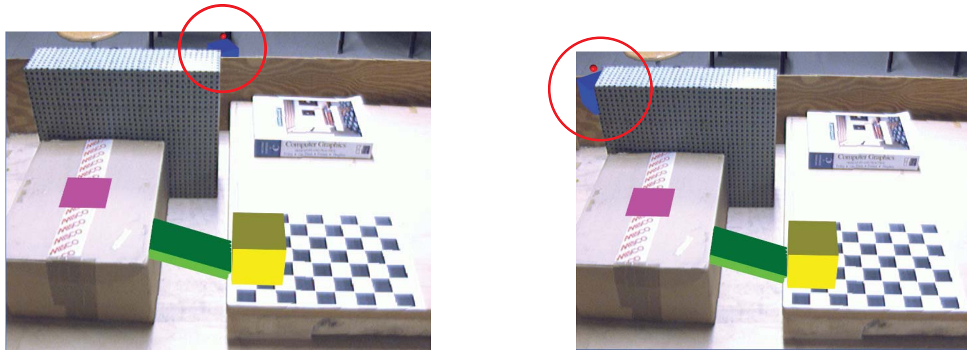


Figura 3.20: Gestione dell'occlusione.

Attraverso queste equazioni risulta possibile calcolare i parametri di OpenGL dalla telecamera reale.

3.1.6 Gestione dell'occlusione

Uno dei problemi della realtà aumentata è quello delle occlusioni, che si verificano quando un oggetto reale è interposto tra l'utente e un oggetto virtuale, e quest'ultimo non deve essere visto dall'utente nella misura in cui è coperto dall'oggetto reale. Nel lavoro di tesi la gestione dell'occlusioni è stata effettuata utilizzando le potenzialità di OpenGL. la figura 3.20 mostra come il cubo di colore blu risulti correttamente nascosta dietro agli oggetti reali della scena.

OpenGL, per disegnare una scena, utilizza un Frame Buffer. Il Frame Buffer è l'insieme della memoria che OpenGL utilizza per immagazzinare informazioni dell'ambiente 3D. Il Frame Buffer è formato da diversi componenti:

Color buffer: contiene i dati relativi al colore di ciascun pixel sullo schermo. In esso è presente il risultato di una parte delle chiamate della funzione

di drawing (redraw negli esempi precedenti).

Depth buffer: utilizzato nella rimozione delle superfici nascoste.

Stencil buffer: utilizzato per limitare il drawing ad aree di forma qualsiasi.

Accumulation buffer: utilizzato per ottenere effetti particolari come motion blur.

Per gestire il fenomeno dell'occlusione si è utilizzato il buffer Color, che è stato disabilitato prima di disegnare gli oggetti virtuali corrispondenti agli oggetti reali che compongono la scena. Così facendo OpenGL non disegnerà realmente l'oggetto ma lo considererà per gestire la rimozione delle superfici nascoste. In questo modo quando si andrà a disegnare un oggetto virtuale dietro un oggetto reale OpenGL sarà in grado di gestire la rimozione delle superfici nascoste.

3.2 Il simulatore fisico

In questo lavoro di tesi si è utilizzato il simulatore fisico Bullet [23].
Bullet Physics è una libreria professionale open source di "collision detection" per corpi rigidi e deformabili le cui principali caratteristiche sono:

- Codice C++ open source con licenza Zlib e gratuita per qualsiasi uso commerciale su tutte le piattaforme: PLAYSTATION 3, XBox 360, Wii, PC, Linux, Mac OSX e iPhone.
- Algoritmo di rilevazione delle collisioni. Sono permesse tutte le forme basi per esempio cubi, cilindri, sfere. Inoltre sono incluse forme convesse e concavi.

- Risolutore dinamico veloce e stabile per vincoli con corpi rigidi, character controller e slider, perno, giunto generico a 6DOF.
- Soft Body dinamici per vestiti, cavi e volumi deformabili con due modi di interazione con i corpi rigidi; include un supporto per i vincoli.
- Plugin per Maya Dynamica , integrazione con Blender, supporto per importare ed esportare COLLADA physics.

Il principale compito di un simulatore fisico consiste nell'individuazione delle collisioni, nella risoluzione delle collisioni e dei vincoli, e nel fornire il calcolo delle future coordinate "mondo[5]" di tutti gli oggetti.
Bullet è stato progettato per essere personalizzabile e modulare. Il programmatore può:

- utilizzare solamente il componente di collision detection,
- utilizzare le componenti per i corpi rigidi senza le componenti per i corpi deformabili,
- scegliere se utilizzare la versione della libreria a singola o a doppia precisione.

I principali componenti sono organizzati come mostra la figura 3.21.

La figura 3.22 mostra un diagramma con le più importanti strutture e i passi di esecuzione della pipeline di Bullet per i corpi rigidi. Questa pipeline è eseguita da sinistra a destra, e comprende una parte di rilevazione delle collisioni e una parte di realizzazione dei vincoli dinamici.

[5]Le coordinate mondo indicano il centro di massa per i corpi rigidi e le trasformate sui vertici per i corpi deformabili.

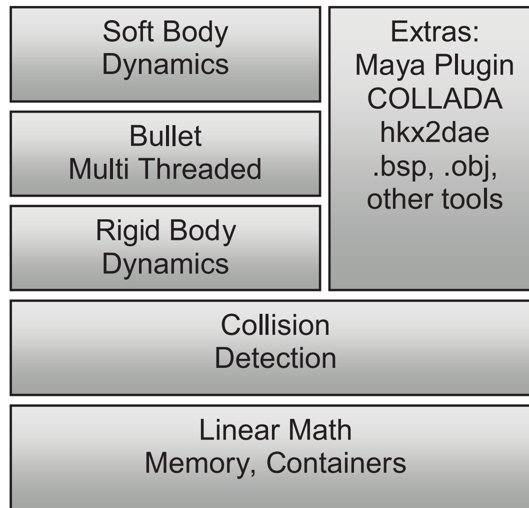


Figura 3.21: Componenti principali di Bullet.

L'intera esecuzione della pipeline di simulazione fisica e le sue strutture dati sono rappresentate in Bullet dal "dynamics world". Bullet permette al programmatore di scegliere diverse parti del dynamics world come broadphase collision detection, narrowphase collision detection (dispatcher) e constraint solver. È anche possibile definire il tempo di simulazione chiamato "stepSimulation", usualmente fissato a 60 Hertz.

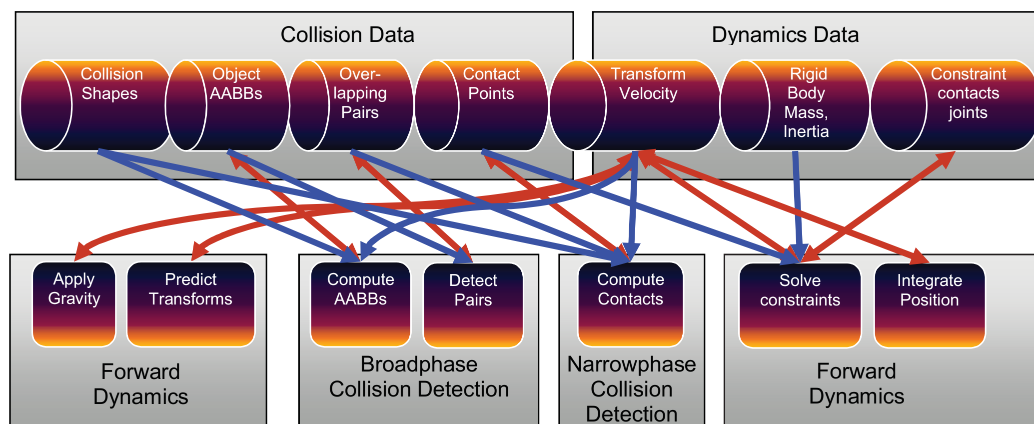


Figura 3.22: Pipeline di Bullet per i corpi rigidi.

Bullet fornisce una libreria matematica nella quale sono definite strutture dati che permettono di eseguire efficientemente operazioni matematiche. Le strutture dati principali di questa libreria sono: btScalar, btVector3, btQuaternion, btMatrix3x3 e btTransform. btTransform è la combinazione della posizione e dell'orientazione, e può essere utilizzato per trasformare punti e vettori dalle coordinate di uno spazio ad un altro (non sono permessi la scalatura o la shearing). Bullet utilizza un sistema di coordinate destrorse, come mostrato in figura 3.23.

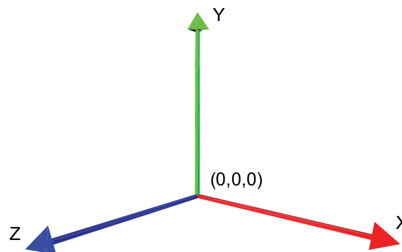


Figura 3.23: Sistema di coordinate di Bullet.

Esso inoltre fornisce differenti algoritmi per la collisione, ed è possibile aggiungerne di propri. Per migliorare le prestazioni della simulazione è importante scegliere la forma di collisione che meglio si addice all'applicazione. La figura 3.24 aiuta nella decisione della forma da attribuire all'oggetto.

Le forme basi più comuni son definite dalle seguenti classi : btBoxShape, btSphereShape, btCapsuleShape, btCylinderShape, btConeShape, btMultiSphereShape (può essere utilizzata per creare una capsula). Si possono formare corpi più complessi unendo più corpi semplici, utilizzando la classe btCompoundShape, mentre per formare corpi convessi definiti da triangoli si utilizza la classe btConvexHullShape.

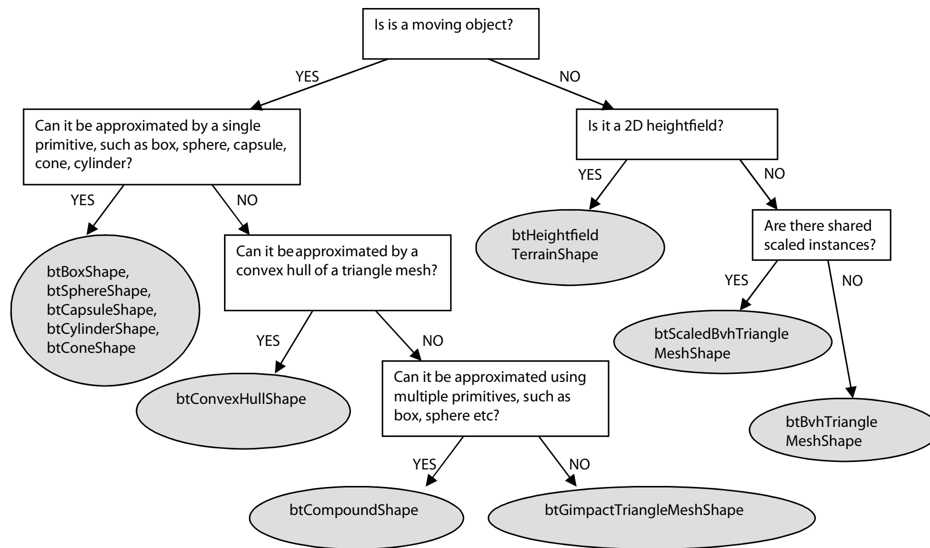


Figura 3.24: Selezione della "Collision Shape" per Bullet.

Per ogni coppia di forme che collidono, Bullet utilizzerà un algoritmo specifico servendosi del "dispatcher". Bullet utilizza un'implementazione gratuita degli algoritmi di collision detection GJK (Gilbert, Johnson e Keerthi) e EPA (Expanding Polythope Algorithm). La figura 3.25 mostra l'algoritmo di collisione utilizzato per le diverse combinazioni di forme collidenti.

In Bullet esistono tre differenti tipi di oggetti:

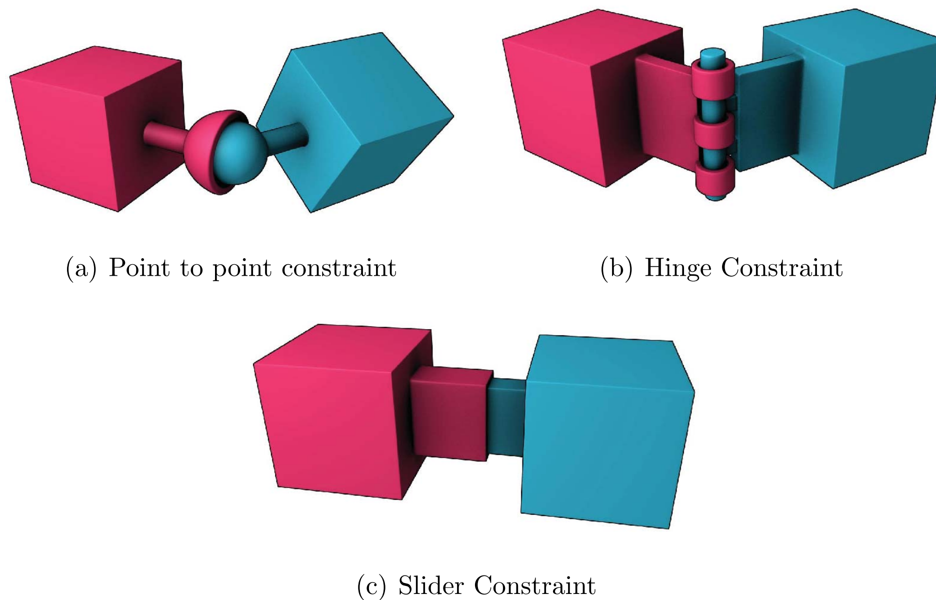
- Dinamici
  - Hanno una massa positiva
  - Ad ogni ciclo di simulazione viene aggiornata la posizione nel mondo dell'oggetto
- Statici
  - Hanno una massa uguale a zero
  - Non possono muoversi, possono semplicemente avere delle collisioni con altri oggetti

	box	sphere	convex,cylinder cone,capsule	compound	triangle mesh
box	boxbox	spherebox	gjk	compound	concaveconvex
sphere	spherebox	spheresphere	gjk	compound	concaveconvex
convex, cylinder, cone, capsule	gjk	gjk	gjk	compound	concaveconvex
compound	compound	compound	compound	compound	compound
triangle mesh	concaveconvex	concaveconvex	concaveconvex	compound	gimpact

Figura 3.25: Algoritmi di rilevazione delle collisioni di Bullet a seconda delle forme degli oggetti coinvolti.

- Cinematici
  - Hanno una massa uguale a zero
  - Sono animati dall'utente, i corpi dinamici sono allontanati da un eventuale collisione, ma l'oggetto cinematico non viene influenzato da questa collisione.

Nel lavoro di tesi si sono utilizzati tutti e tre i tipi di oggetti a disposizione. Gli oggetti reali dell'ambiente ricreato sono stati definiti come oggetti statici, infatti in nessun modo potranno essere mossi durante l'esecuzione della applicazione software. Gli oggetti virtuali che formano la scena di realtà aumentata sono definiti invece come dinamici. Infine, per il dispositivo aptico si è definito all'interno del simulatore un oggetto sferico di tipo cinematico, e sarà proprio l'utente a spostare questo oggetto e ad interagire con gli altri.



(a) Point to point constraint (b) Hinge Constraint

(c) Slider Constraint

Figura 3.26: I diversi tipi di giunti presenti in Bullet.

Bullet fornisce anche diversi tipi di vincoli tra gli oggetti; i piu importanti sono btPoint2PointConstraint, btHingeConstraint, btSlideConstraint, btConeTwistConstraint, btGeneric6DofConstraint (la figura 3.26 mostra tre di questi vincoli).

3.2.1 Classi e thread dell'applicazione

Il progetto è composto da tre thread principali mostrate in figura 3.27.
Il programma inizia dal file main.cpp, dove si inizializza il programma per la ricerca dei marker e si crea l'ambiente fisico. In base alla definizione o meno della variabile ENABLE_ARTOOLKIT definita, nel file PhysicsClass.h si può creare un ambiente di realtà aumentata, o semplicemente un ambiente di realtà virtuale.

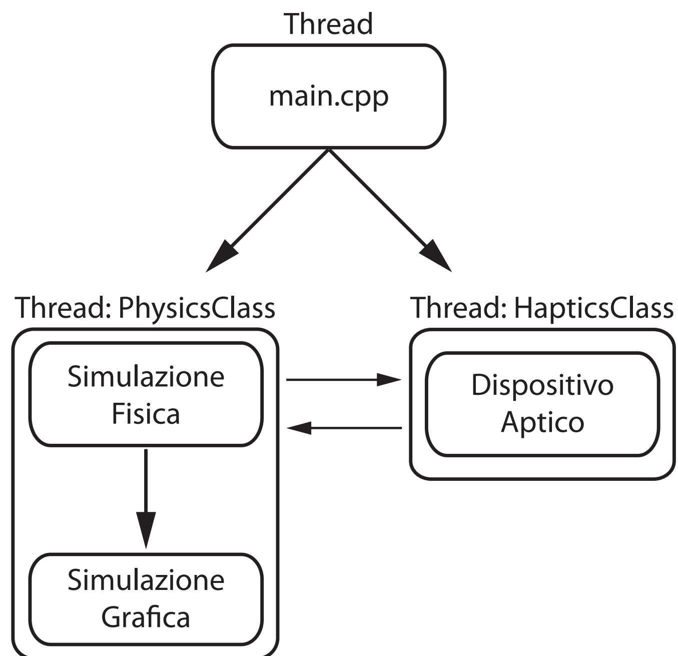


Figura 3.27: Thread e classi del progetto.

```
......
PhysicsClass physics;
......
int main(int argc,char** argv) {
   ......

 glutInit(&argc, argv);

 //inizializzazione del falcon e di artoolkit
 init();

 physics.initPhysics();
   ....
 return glutmain(argc, argv,640,480," Tesi ",&physics);

```

```
16   //default glut doesn't return from mainloop
17   return 0;
18 }
19
20 static void init( void )
21 {
22 #ifdef ENABLE_ARTOOLKIT
23     ARParam wparam;
24
25     /* open the video path */
26     if( arVideoOpen( vconf ) < 0 ) exit(0);
27     /* find the size of the window */
28     if( arVideoInqSize(&physics.xsize, &physics.ysize) < 0 ) exit
           (0);
29     printf("Image size (x,y) = (%d,%d)\n", physics.xsize, physics.
           ysize);
30
31     /* set the initial camera parameters */
32     if( arParamLoad(cparam_name, 1, &wparam) < 0 ) {
33         printf("Camera parameter load error !!\n");
34         exit(0);
35     }
36
37     arParamChangeSize( &wparam, physics.xsize, physics.ysize, &
           cparam );
38     arInitCparam( &cparam );
39     printf("*** Camera Parameter ***\n");
40     arParamDisp( &cparam );
41
42   if( (physics.patt_id=arLoadPatt(patt_name)) < 0 ) {
43         printf("pattern load error !!\n");
44         exit(0);
45     }
```

```
46
47      /* open the graphics window */
48      argInit( &cparam, 1.0, 0, 0, 0, 0 );
49
50  #else
51    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH |
        GLUT_STENCIL);
52      glutInitWindowPosition(0, 0);
53      glutInitWindowSize(640, 480);
54      glutCreateWindow(" Tesi ");
55  #endif
56
57    .....
58    Inizializzazione del Falcon
59    .....
60
61  #ifdef ENABLE_ARTOOLKIT
62      arVideoCapStart();
63  #endif
64
65  }
```

Listato 3.1: Inizializzazione dell'ambiente e del software per individuare i marker.

Il listato 3.1 mostra le principali operazioni eseguite dal main, mentre l'inizializzazione del Falcon è illustrata dal listato 3.11. Con la classe PhysicsClass si definisce l'oggetto physics che crea l'ambiente virtuale ed esegue la simulazione fisica; questa classe si occuperà anche di eseguire il rendering grafico del programma. La funzione `glutmain(argc, argv,640,480,"Tesi ",&physics)` richiama una funzione definita nel file GlutStuffm.cpp, che predispone la classe PhysicsClass ad eseguire il rendering grafico.

La classe PhysicsClass eredita da una classe base di Bullet di nome DemoAp-

plication. Essa ridefinisce molte funzioni, tra cui: updateCamera(), myinit(), initPhysics(), renderme(), renderscene(), shootBox(). La classe PhysicsClass, oltre a contenere oggetti e variabili per effettuare la simulazione fisica, contiene al suo interno un oggetto di nome camera_t, che appartiene alla classe CCamera, e un oggetto di nome gHaptics, appartenente alla classe HapticsClass. L'oggetto camera_t si occupa di effettuare i calcoli necessari per ottenere una corrispondenza tra mondo reale e virtuale, mentre gHaptics si occupa della gestione del dispositivo aptico. La scelta di inserire la classe del dispositivo aptico dentro la classe PhysicsClass può permettere in futuro di inserire più di un sensore aptico all'interno dell'applicazione creata, e proprio per questo motivo all'interno di questa classe si hanno due callback functions, mostrate nel listato 3.14, che come verrà spiegato più avanti si occuperanno della gestione del dispositivo aptico.
La figura 3.28 mostra il diagramma delle classi del sistema che mette in evidenze solamente le funzioni principali.

Alla creazione dell'oggetto physics si inizializza l'oggetto camera_t con la seguente funzione: `camera_t.LoadFile("camera.txt")`. Il file camera.txt contiene al suo interno i parametri estrinseci ed intrinseci della telecamera che sono stati precedentemente calcolati. Il file è mostrato nel listato 3.2.

```
1
2 Width 640 Height 480
3
4 INT 9.622530e+002 0.0000 .....
5
6 EXT 9.997839e−001 1.8237 .....
```

Listato 3.2: File camera.txt

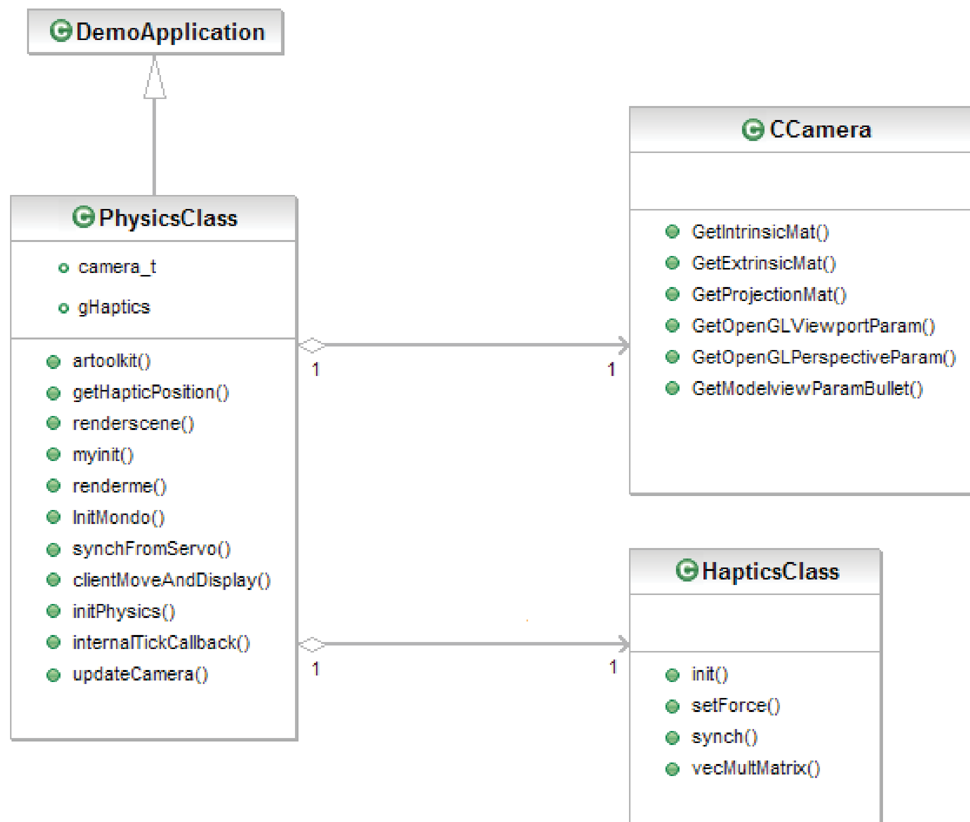


Figura 3.28: Diagramma semplificato delle classi che compongono il sistema.

I parametri estrinseci ed intrinseci della telecamera sono stati ricavati utilizzando Matlab, come discusso nel paragrafo 3.1.2. Dopo aver ottenuto i parametri vengono eseguiti due file di nome utileExt.m e utileInt.m, creati per fornire le righe di parametri INT ed EXT da inserire nel file camera.txt. Nella funzione LoadFile si esegue la lettura da file e si inizializzano le varibili all'interno dell'oggetto camera_t. Dopo l'esecuzione di questa funzione physics esegue la funzione updateCamera() mostrata nel listato 3.3.

```
1 void PhysicsClass::updateCamera() {
2
```

```
3   double pos_camera[3];
4
5   // Get OpenGL parameters
6   camera_t.GetOpenGLViewportParam(newViewport) ;
7   camera_t.GetOpenGLPerspectiveParam(fovy, aspect, fNear, fFar) ;
8   camera_t.GetModelviewParamBullet(cameraModelview) ;
9
10 #ifdef ENABLE_ARTOOLKIT
11  glMatrixMode(GL_PROJECTION);
12  glLoadIdentity();
13
14  glMatrixMode(GL_MODELVIEW) ;
15  glLoadIdentity() ;
16
17  glViewport(newViewport[0], newViewport[1], newViewport[2],
        newViewport[3]) ;
18
19  glMatrixMode(GL_PROJECTION) ;
20  glLoadIdentity() ;
21  gluPerspective(fovy, aspect, fNear, fFar*10000) ;
22
23  glMatrixMode(GL_MODELVIEW) ;
24  glLoadIdentity() ;
25
26  glLoadMatrixd(cameraModelview) ;
27
28 #else
29
30  DemoApplication::updateCamera();
31
32 #endif
33 }
```

Listato 3.3: Crezione della corretta proiezione e posizione della telecamera.

In questa funzione[6] si calcolano le sei variabili che permettono di costruire la realtà aumentata: newViewport, fovy, aspect, fNear, fFar e cameraModelview. Successivamente, utilizzando OpenGL, si crea la prospettiva corretta utilizzando le funzioni glViewport e gluPerspective (vedere paragrafo 3.1.4); infine si esegue la funzione glLoadMatrixd(cameraModelview) che imposta la posizione della telecamere. Infatti la variabile cameraModelview rappresenta una matrice di rototraslazione.

L'oggetto physics inizializza il mondo virtuale attraverso la funzione initPhysics() mostrata dal listato 3.4.

```
1  void PhysicsClass::initPhysics()
2  {
3   .......
4   .......
5
6   m_dispatcher=0;
7
8   //per gestire anche i SoftBody
9   m_collisionConfiguration = new
        btSoftBodyRigidBodyCollisionConfiguration();
10
11  ///use the default collision dispatcher.
12  m_dispatcher = new btCollisionDispatcher(
        m_collisionConfiguration);
```

[6]Nell'eventualità non sia definita la variabile ENABLE_ARTOOLKIT si chiama la funzione della superClasse.

```
13 m_softBodyWorldInfo.m_dispatcher = m_dispatcher;
14
15 btVector3 worldMin(-10000,-10000,-10000);
16 btVector3 worldMax(10000,10000,10000);
17
18 m_broadphase = new btAxisSweep3(worldMin,worldMax,maxProxies);
19
20 m_softBodyWorldInfo.m_broadphase = m_broadphase;
21 m_solver = new btSequentialImpulseConstraintSolver();
22
23 btDiscreteDynamicsWorld* world = new btSoftRigidDynamicsWorld(
       m_dispatcher,m_broadphase,m_solver,m_collisionConfiguration);
24 m_dynamicsWorld = world;
25
26 .......
27 .......
28
29 m_dynamicsWorld->getDispatchInfo().m_enableSPU = true;
30
31 m_dynamicsWorld->setGravity(btVector3(0,-1000,0));
32
33 m_softBodyWorldInfo.m_gravity.setValue(0,-1000,0);
34
35 .......
36 .......
37
38 InitMondo();
39
40 .......
41 .......
42 .......
43
```

```
44  m_dynamicsWorld->setInternalTickCallback(myTickCallback,
        static_cast<void *>(this));
45
46  m_softBodyWorldInfo.m_sparsesdf.Initialize();
47
48  clientResetScene();
49 }
```

Listato 3.4: Creazione del mondo virtuale e delle caratteristiche del simulatore fisico.

In questa funzione si inizializza il simulatore fisico andando a specificare le varie componenti come: collisionConfiguration, dispatcher, broadphase, solver, world, questa inizializzazione utilizza le classi che permettono di creare e gestire anche una simulazione con corpi deformabili, i quali sono stati inseriti in questo lavoro di tesi. La variabile più importante è m_dynamicsWorld, e rappresenta il mondo fisico e tutte le sue componenti per una corretta simulazione.
Con la funzione setInternalTickCallback si indica al simulatore fisico quale funzione invocare al termine di ogni passo di simulazione, questa sarà utilizzata per la corretta interazione con il dispositivo aptico.

Nel listato, oltre alle inizializzazioni necessarie, vengono definiti gli oggetti presenti all'interno della scena. Nella funzione InitMondo vengono definiti tutti gli oggetti reali dell'ambiente di realtà aumentata creato, un esempio di definizione di un oggetto statico è mostrato nel listato 3.5.

```
1  //definisce la forma dell'oggetto
2  btCollisionShape* groundShape;
3  btTransform groundTransform;
```

```
4
5   //li si assegna la forma di un cubo di lato 40
6   groundShape = new btBoxShape(btVector3(btScalar(20.),btScalar
       (20.),btScalar(20.)));
7
8   m_collisionShapes.push_back(groundShape);
9   groundTransform.setIdentity();
10
11  //posizione nelle coordinate mondo di Bullet
12  groundTransform.setOrigin(btVector3(0, 0,0));
13
14  //inserimento dell'oggetto nel mondo e nella simulazione
15  localCreateRigidBody(0,groundTransform,groundShape);
```

Listato 3.5: Esempio di definizione di un oggetto di tipo statico.

Il primo parametro della funzione localCreateRigidBody definisce la massa dell'oggetto, ed è impostata a zero proprio perché si vuole un oggetto statico, mentre i successivi due parametri sono la posizione nelle coordinate del mondo di bullet e la forma del'oggetto. La funzione calcola al suo interno il tensore d'inerzia, e si occupa inoltre di inserire l'oggetto nel mondo di simulazione.

Un esempio di oggetto dinamico è rappresentato dal listato 3.6. Nell'esempio si nota come si possono definire dei parametri all'oggetto come un fattore di damping e la frizione.

```
1     // Create Dynamic Objects
2
3     btCollisionShape* colShape = new btBoxShape(btVector3
         (30,30,30));
```

```
4    m_collisionShapes.push_back(colShape);
5    btTransform startTransform;
6    startTransform.setIdentity();
7    startTransform.setOrigin( btVector3(0,0,0) );
8
9    //rigidbody is dynamic if and only if mass is non zero,
        otherwise static
10   btVector3 localInertia(0,0,0);
11   int mass=10;
12   colShape->calculateLocalInertia(mass,localInertia);
13
14   //using motionstate is recommended, it provides interpolation
        capabilities, and only synchronizes 'active' objects
15
16   btDefaultMotionState* myMotionState = new btDefaultMotionState
        (startTransform);
17
18   btRigidBody::btRigidBodyConstructionInfo cInfo(mass,
        myMotionState,colShape,localInertia);
19
20   cubo = new btRigidBody(cInfo);
21
22   cubo->setFriction(1.65);
23   cubo->setDamping(0.9,0.9);
24
25   m_dynamicsWorld->addRigidBody(cubo);
```

Listato 3.6: Inizializzazione di un oggetto dinamico.

Nel programma, come già anticipato, viene anche aggiunto un oggetto di tipo ghost che identifica il dispositivo aptico utilizzato. Il dispositivo viene simulato da un oggetto di forma sferica, perché ben rappresenta il grip del Falcon, nel listato 3.7 si mostra come viene inizializzato nel programma.

```
1  btTransform startTransform;
2  startTransform.setIdentity();
3  btVector3 origin;
4  getHapticPosition(origin);
5  startTransform.setOrigin (origin);
6
7  m_ghostObject = new btPairCachingGhostObject();
8  m_ghostObject->setWorldTransform(startTransform);
9
10 m_softBodyWorldInfo.m_broadphase->getOverlappingPairCache()->
       setInternalGhostPairCallback(new btGhostPairCallback());
11
12 btSphereShape* sphere = new btSphereShape (btScalar(8));
13
14 m_ghostObject->setCollisionShape (sphere);
15 m_ghostObject->setCollisionFlags (btCollisionObject::
       CF_KINEMATIC_OBJECT);
16 m_ghostObject->setActivationState(DISABLE_DEACTIVATION);
17
18 m_dynamicsWorld->addCollisionObject(m_ghostObject,
       btBroadphaseProxy::CharacterFilter, btBroadphaseProxy::
       StaticFilter|btBroadphaseProxy::DefaultFilter);
```

Listato 3.7: Inizializzazione dell'oggetto ghost all'interno dell'ambiente di realtà aumentata.

La funzione getHapticPosition è illustrata nel listato 3.16, e principalmente si occupa di ottenere la posizione del Falcon, con la funzione addCollisionObject e setCollisionFlags si inserisce l'oggetto all'interno della simulazione e si definisco i filtri di collisione per ottenere correttamente i punti di contatto e le collisioni con gli oggetti.

Nel lavoro di tesi sono stati inseriti anche oggetti piu complessi e di tipo deformabile; nel listato 3.8 si visualizzano l'inizializzazione di oggetti trimesh e oggetti soft, il primo è il famoso coniglio triangolarizzato ed è di consistenza rigida mentre nel secondo caso si ha lo stesso coniglio ma con un corpo deformabile.

```
// Create Bunny Shape (Rigid)
{
  m_indexVertexArrays2 = new btTriangleIndexVertexArray
    (BUNNY_NUM_TRIANGLES,&gIndicesBunny[0][0], 3*sizeof(int),
        BUNNY_NUM_VERTICES, (REAL*) &gVerticesBunny[0],sizeof(REAL
        )*3);
  btGImpactConvexDecompositionShape * trimesh2 = new
      btGImpactConvexDecompositionShape(m_indexVertexArrays2,
      btVector3(80.f,80.f,80.f),btScalar(0.01));
  trimesh2->updateBound();
  //register algorithm
  btCollisionDispatcher * dispatcher = static_cast<
      btCollisionDispatcher *>(m_dynamicsWorld ->getDispatcher()
      );
  btGImpactCollisionAlgorithm::registerAlgorithm(dispatcher);
}

//Init_Bunny (Soft)
{
  psb=btSoftBodyHelpers::CreateFromTriMesh(this->
      m_softBodyWorldInfo, gVerticesBunny, &gIndicesBunny[0][0],
      BUNNY_NUM_TRIANGLES);
  btSoftBody::Material* pm=psb->appendMaterial();
  pm->m_kLST    = 0.5;
  pm->m_flags    -= btSoftBody::fMaterial::DebugDraw;
  psb->generateBendingConstraints(2,pm);
```

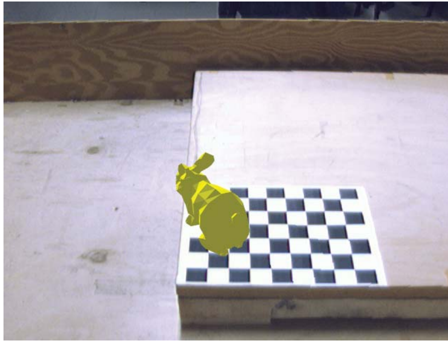


Figura 3.29: Coniglio rigido trimesh.

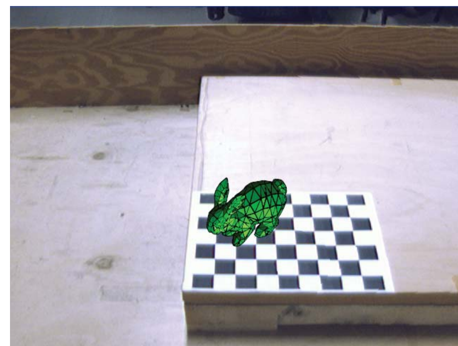


Figura 3.30: Coniglio deformabile.

```
19    psb->m_cfg.piterations = 2;
20    psb->m_cfg.kDF    = 0.5;
21    psb->randomizeConstraints();
22    psb->scale(btVector3(60,60,60));
23    psb->setTotalMass(100,true);
24
25
26    btMatrix3x3 m;
27    m.setIdentity();
28    //effettuo una rotazione dell'oggetto.
29    m.setEulerZYX(0.0,3.14,0.0);
30    psb->transform(btTransform(m,btVector3(-60,150,100)));
31
32    this->getSoftDynamicsWorld()->addSoftBody(psb);
33  }
```

Listato 3.8: Inizializzazione di un oggetto triangolarizzato e di un oggetto soft.

La figura 3.29 mostra il coniglio rigido trimesh mentre la figura 3.30 mostra il coniglio deformabile.

Il loop grafico richiama principalmente due funzioni, la clientMoveAndDisplay e la displayCallback, (successivamente vedremo principalmente la

clientMoveAndDisplay, mostrata nel listato 3.9).

```
//procedura che viene richiamata dal glutIdleFunc
void PhysicsClass::clientMoveAndDisplay()
{
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

#ifdef ENABLE_ARTOOLKIT
 if(gARTImage!=NULL){
   argDrawMode2D();
   argDispImage( gARTImage, 0,0 );
 }

 artoolkit();

 PhysicsClass::myinit();
 PhysicsClass::myCamera();
 glDepthFunc(GL_LEQUAL);

#endif

 float ms = getDeltaTimeMicroseconds();

 ///step the simulation
 if (m_dynamicsWorld)
 {
   ........

   int numSimSteps = m_dynamicsWorld->stepSimulation(ms / 10000.f
       ,6);

   m_dynamicsWorld->debugDrawWorld();

   ........
```

```
32     ........
33
34   }
35
36   m_softBodyWorldInfo.m_sparsesdf.GarbageCollect();
37
38   renderme();
39
40   glFlush();
41   glutSwapBuffers();
42 }
```

Listato 3.9: Funzione richiamata dalla glutIdleFunc che si occupa del render grafico.

Nella funzione si ricercano i marker tramite la funzione artoolkit(), e successivamente viene impostato il tempo della simulazione fisica. Infine viene eseguito il render grafico chiamando la funzione renderme(), che si occupererà di disegnare gli oggetti presenti nell'ambiente simulato. La funzione glDepthFunc(GL_LEQUAL) è indispensabile per visualizzare la scena reale ripresa dalla telecamera, infatti disabilitandola la texture prodotta da artoolkit per visualizzare il mondo reale non apparirà più.
Se si individua un marker all'interno della scena, viene inserito un cubo di tipo ghost all'interno dell'ambiente di realtà aumentata proprio nella posizione di individuazione del marker. Spostando il marker l'oggetto virtuale ricreato seguirà lo spostamento del marker, e ciò permette di creare azioni di cooperazione, e quindi di avere due utenti, il primo che manipola il Falcon e l'altro che esegue azioni muovendo il marker, (la figura 3.31 mostra un esempio di riconoscimento del marker).

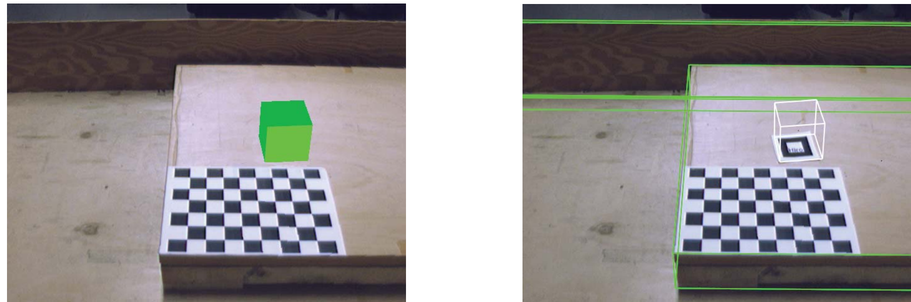


Figura 3.31: Esempio di riconoscimento di marker.

Infine si analizza la funzione internalTickCallback, mostrata solo in parte nel listato 3.10, in cui si calcolano le collisione della sfera, che rappresenta il dispositivo aptico, con gli oggetti presenti nell'ambiente.

```
1  void myTickCallback(btDynamicsWorld *world, btScalar timeStep) {
2   PhysicsClass *basic=static_cast<PhysicsClass *>(world->
       getWorldUserInfo());
3   basic->internalTickCallback(timeStep);
4
5  }
6
7  void PhysicsClass::internalTickCallback(btScalar timeStep){
8   ........
9   ........
10  ........
11  ........
12  {
13   btManifoldArray manifoldArray;
14   btBroadphasePairArray& pairArray = m_ghostObject->
        getOverlappingPairCache()->getOverlappingPairArray();
15
16   int numPairs = pairArray.size();
17
18   forceX=0;
```

```
19    forceY=0;
20    forceZ=0;
21
22    for (int i=0;i<numPairs;i++)
23    {
24     manifoldArray.clear();
25
26     const btBroadphasePair& pair = pairArray[i];
27
28     btBroadphasePair* collisionPair = m_broadphase->
         getOverlappingPairCache()->findPair(pair.m_pProxy0,pair.
         m_pProxy1);
29
30     if (!collisionPair)
31      continue;
32
33     if (collisionPair->m_algorithm)
34      collisionPair->m_algorithm->getAllContactManifolds(
          manifoldArray);
35     ........
36     ........
37     ........
38     for (int j=0;j<manifoldArray.size();j++)
39     {
40      btPersistentManifold* manifold = manifoldArray[j];
41      for (int p=0;p<manifold->getNumContacts();p++)
42      {
43       btManifoldPoint&pt = manifold->getContactPoint(p);
44
45       //btVector3 color(255,255,255);
46       btVector3 color(0,0,255);
47
48       glBegin(GL_LINES);
```

```
49       glColor3f(0, 0, 1);
50
51       btVector3 ptA = pt.getPositionWorldOnA();
52       btVector3 ptB = pt.getPositionWorldOnB();
53
54       int segno=-1;
55       if(manifold->getBody1() == m_ghostObject)
56         segno=1;
57
58       glVertex3d(ptA.x(),ptA.y(),ptA.z());
59       glVertex3d(ptB.x(),ptB.y(),ptB.z());
60
61       glEnd();
62
63       btVector3 normal = pt.m_normalWorldOnB;
64       btScalar angleX = normal.angle(btVector3(1,0,0));
65       btScalar angleY = normal.angle(btVector3(0,1,0));
66       btScalar angleZ = normal.angle(btVector3(0,0,1));
67
68       btScalar impulseX = btFabs(pt.m_distance1)*cos(angleX);
69       btScalar impulseY = btFabs(pt.m_distance1)*cos(angleY);
70       btScalar impulseZ = btFabs(pt.m_distance1)*cos(angleZ);
71       ........
72       ........
73       forceX+=((double)impulseX/ (timeStep));
74       forceY-=((double)impulseY/ (timeStep));
75       forceZ+=((double)impulseZ/ (timeStep));
76      }
77     }
78    }
79    if(forceX >MAX_FORCE) forceX = MAX_FORCE;
80    if(forceX <-MAX_FORCE) forceX = -MAX_FORCE;
81    if(forceY >MAX_FORCE) forceY = MAX_FORCE;
```

```
82     if(forceY <-MAX_FORCE) forceY = -MAX_FORCE;
83     if(forceZ >MAX_FORCE) forceZ = MAX_FORCE;
84     if(forceZ <-MAX_FORCE) forceZ = -MAX_FORCE;
85 
86     gHaptics.setForce( forceX , forceY, forceZ );
87   }
88   btVector3 walkDirection = btVector3(0.0, 0.0, 0.0);
89   getHapticPosition(walkDirection);
90   btTransform xform;
91   xform.setIdentity();
92   xform.setOrigin (walkDirection);
93   m_ghostObject->setWorldTransform (xform);
94 
95 }
```

Listato 3.10: Funzione chiamata al termine di ogni passo di simulazione.

Dopo aver ricavato le collisioni si calcola la penetrazione tra la sfera e l'oggetto con il quale sta collidendo, in base alla penetrazione si calcola la forza da settare al dispositivo aptico. Al termine di questa funzione si aggiorna la posizione del dispositivo aptico.
In questa funzione viene anche implementata la possibilità di prendere un oggetto tramite il Falcon. Per eseguire questa operazione basta tener premuto il pulsante centrale del grip del Falcon e andare a toccare un oggetto dinamico dell'ambiente. Appena si toccherà l'oggetto dinamico viene creato un giunto di tipo Point to point tra il Falcon e l'oggetto dinamico creato, in questo modo sarà possibile portare l'oggetto dove si vuole. La figura 3.32 mostra tale modalità.

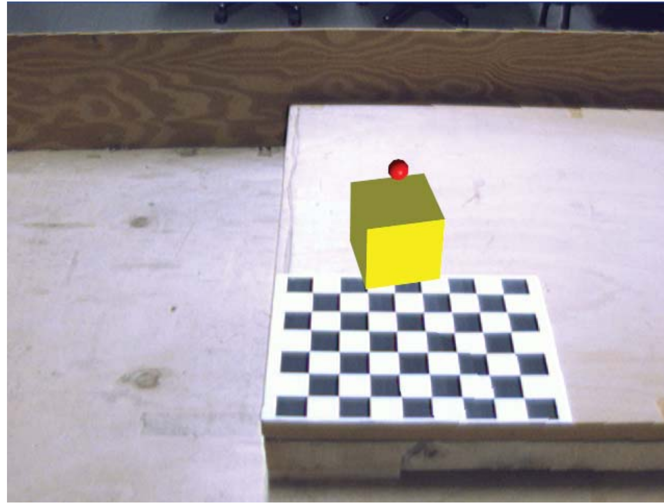


Figura 3.32: Modalità di presa di un oggetto applicando un vincolo di tipo point to point.

3.3 Il dispositivo aptico

Questa sezione della tesi illustra in dettaglio la comunicazione tra il simulatore fisico e il dispositivo aptico. La figura 3.33 mette in evidenza tale comunicazione. In questa comunicazione si deve prestare attenzione alle sincronizzazione tra le thread, perché sia Bullet che il Falcon vorranno accedere a variabili condivise come la posizione del grip, lo stato dei pulsanti e la forza da attribuire al Falcon.

Oltre ad assicurare la correttezza della sincronizzazione, è necessario utilizzare correttamente il Falcon, illustrando come selezionare il dispositivo, come inizializzarlo, come leggerne lo stato (posizione e pulsanti) e comandarne la forza.
Per usare il Falcon si utilizza la libreria software Haptic Device Abstraction Layer (HDAL), il cui obbiettivo primario è quello di fornire un'interfaccia uniforme che supporti i principali dispositivi aptici creati dalla NOVINT.

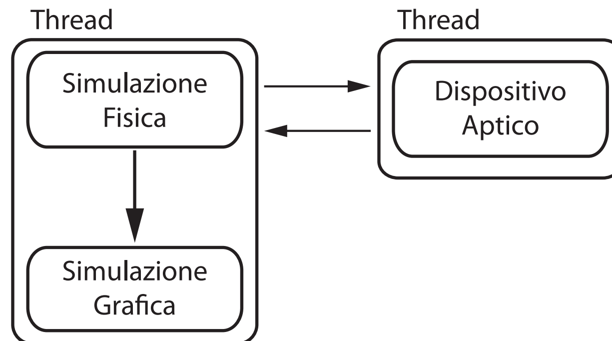


Figura 3.33: Interazione tra la thread del dispositivo aptico e quella del simulatore fisico.

HDAL è implementato a livelli, e comunica con il Falcon attraverso le HDAL API. L'elemento principale di questa comunicazione, dopo l'inizializzazione, è l'uso di una "callback function". La funzione viene chiamata all'interno dell'HDAL ad ogni interazione ("tick"), che avviene approssimativamente 1000 volte al secondo. All'interno di questa funzione il programmatore legge la posizione, calcola la forza appropriata per l'applicazione e invia questa forza al dispositivo.

Il compito principale del programmatore è quello di assicurarsi che la simulazione aptica e la simulazione grafica dell'applicazione siano sincronizzate, così da avere correlazione tra ciò che l'utente vede e sente (simulazione grafica) con la percezione della forza (simulazione aptica).

3.3.1 Inizializzazione del Falcon

La gestione del Falcon viene eseguita tramite la classe HapticsClass, definita dai file haptics.h e haptics.cpp. Questa classe si occupa della corretta inizializzazione e dell'utilizzo del dispositivo aptico, permettendone l'integrazione

nell'applicazione principale, che in questa tesi è rappresentata dal simulatore fisico e dall'architettura di realtà aumentata.
L'inizializzazione del dispositivo avviene nel file main.cpp ed è caratterizzata dalle funzioni mostrate nel listato 3.11.

```
// Set up handler to make sure teardown is done.
atexit(exitHandler);

// Call the haptics initialization function
physics.gHaptics.init(&physics,ContactCB);

// Some time is required between init() and checking status,
// for the device to initialize and stabilize. In a complex
// application, this time can be consumed in the initGL()
// function. Here, it is simulated with Sleep().
Sleep(100);

// Tell the user what to do if the device is not calibrated
if (!physics.gHaptics.isDeviceCalibrated())
   MessageBox(NULL,
           // The next two lines are one long string
           "Please home the device by extending\n"
           "then pushing the arms all the way in.",
           "Not Homed",
           MB_OK);
```

Listato 3.11: Inizializzazione Falcon.

Si consideri l'istruzione: `physics.gHaptics.init(&physics,ContactCB)` con la quale si inizializza il dispositivo. Con il nome physics viene identificato un oggetto della classe PhysicsClass che si occupa della simulazione fisica del programma. Al suo interno è presente un oggetto della classe HapticsClass

chiamato gHaptics; la funzione init si occupererà di gestire l'inizializzazione del dispositivo. I parametri passati alla funzione servono per definire la callback function, fondamentale per il funzionamento del dispositivo. Il listato 3.12 mostra la funzione init, che rappresenta una normale sequenza d'inizializzazione.

La sequenza delle istruzioni deve essere rispettata per il corretto funzionamento del dispositivo. In particolare, hdlCreateServoOp non deve essere chiamata prima di hdlInitNamedDevice, e se hdlStart non è chiamata prima di hdlCreateServoOp il dispositivo potrebbe non funzionare[7].

```
1  void HapticsClass::init(void* SuperClass, HDLServoOpExitCode
       ContactCB(void *data))
2  {
3
4     HDLError err = HDL_NO_ERROR;
5
6     // Passing "DEFAULT" or 0 initializes the default device based
          on the
7     // [DEFAULT] section of HDAL.INI. The names of other sections
         of HDAL.INI
8     // could be passed instead, allowing run-time control of
         different devices
9     // or the same device with different parameters. See HDAL.INI
         for details.
10    m_deviceHandle = hdlInitNamedDevice("DEFAULT");
11    testHDLError("hdlInitDevice");
12
13    if (m_deviceHandle == HDL_INVALID_HANDLE)
```

[7] L'errata inizializzazione del dispositivo potrebbe portare ad un'indicazione di errore istantaneamente o mentre si sta utilizzando il dispositivo.

```
14    {
15        MessageBox(NULL, "Could not open device", "Device Failure",
              MB_OK);
16        exit(0);
17    }
18
19    // Now that the device is fully initialized, start the servo
         thread.
20    // Failing to do this will result in a non-funtional haptics
         application.
21    hdlStart();
22    testHDLError("hdlStart");
23
24    // Set up callback function
25    m_servoOp = hdlCreateServoOp(ContactCB, SuperClass,
         bNonBlocking);
26    if (m_servoOp == HDL_INVALID_HANDLE)
27    {
28        MessageBox(NULL, "Invalid servo op handle", "Device Failure
             ", MB_OK);
29    }
30    testHDLError("hdlCreateServoOp");
31
32    // Make the device current. All subsequent calls will
33    // be directed towards the current device.
34    hdlMakeCurrent(m_deviceHandle);
35    testHDLError("hdlMakeCurrent");
36
37    // Get the extents of the device workspace.
38    // Used to create the mapping between device and application
         coordinates.
39    // Returned dimensions in the array are minx, miny, minz, maxx
         , maxy, maxz
```

```
40     // left, bottom, far, right, top, near)
41     // Right-handed coordinates:
42     //  left-right is the x-axis, right is greater than left
43     //  bottom-top is the y-axis, top is greater than bottom
44     //  near-far is the z-axis, near is greater than far
45     // workspace center is (0,0,0)
46     hdlDeviceWorkspace(m_workspaceDims);
47     testHDLError("hdlDeviceWorkspace");
48 
49     // Establish the transformation from device space to app space
50     // To keep things simple, we will define the app space units
          as
51     // inches, and set the workspace to approximate the physical
52     // workspace of the Falcon. That is, a 4" cube centered on the
53     // origin. Note the Z axis values; this has the effect of
54     // moving the origin of world coordinates toward the base of
          the
55     // unit.
56     double gameWorkspace[] = {-900,-900,-900,900,900,1000};
57     bool useUniformScale = true;
58     hdluGenerateHapticToAppWorkspaceTransform(m_workspaceDims,
59                                               gameWorkspace,
60                                               useUniformScale,
61                                               m_transformMat);
62     testHDLError("hdluGenerateHapticToAppWorkspaceTransform");
63 
64     m_inited = true;
65 }
```

Listato 3.12: Funzioni che permettono l'inizializzazione del Falcon.

Si noti che il gameWorkspace identifica le dimensioni virtuali (espresse in cm) nel quale si può muovere il Falcon. Tali dimensioni sono molto impor-

tanti per far si che il dispositivo interagisca correttamente con le dimensioni dell'ambiente virtuale/reale ricreato nell'applicazione

3.3.2 Le "callback functions" e la sincronizzazione

Quando l'HDAL è inizializzato correttamente crea una thread, chiamata "servo thread", che gestisce la comunicazione tra il dispositivo e l'applicazione. Questa thread successivamente invoca una chiamata all'applicazione che l'ha creata, da cui il nome callback function.
Il prototipo di questa funzione è: `HDLServoOpExitCode HDLServoOp(void* pParam)` La definizione di questa funzione si trova all'interno della classe PhysicsClass con il nome di ContactCB. Il parametro pParam viene utilizzato per passare un puntatore della classe PhysicsClass.
Nell'architettura della tesi è presente anche un'altra thread che vuole accedere ai dati del dispositivo; bisogna quindi assicurare il corretto accesso alle variabili comuni. A questo scopo viene aggiunta un'ulteriore funzione friend, e un'altra funzione che si occupererà del trasferimento delle variabili tra il dispositivo e l'applicazione. Nel listato 3.13 sono riportate le inizializzazioni delle due funzioni friend, mentre nel listato 3.14 è presente la loro definizione.

```
1  // Define callback functions as friends
2  friend HDLServoOpExitCode ContactCB(void *data);
3  friend HDLServoOpExitCode GetStateCB(void *data);
```

Listato 3.13: Inizializzazione delle funzioni friend.

```
1 // Continuous servo callback function
2 HDLServoOpExitCode ContactCB(void* pUserData)
3 {
```

```
4    // Get pointer to haptics object
5    PhysicsClass* basic = static_cast< PhysicsClass* >( pUserData );
6
7    // Get current state of haptic device
8    hdlToolPosition(basic->gHaptics.m_positionServo);
9    hdlToolButton(&(basic->gHaptics.m_buttonServo));
10
11   // Send forces to device
12   hdlSetToolForce(basic->gHaptics.m_forceServo);
13
14   // Make sure to continue processing
15   return HDL_SERVOOP_CONTINUE;
16 }
17
18 // On-demand synchronization callback function
19 HDLServoOpExitCode GetStateCB(void* pUserData)
20 {
21   // Get pointer to haptics object
22   PhysicsClass* basic = static_cast< PhysicsClass* >( pUserData );
23
24   hdlToolPosition(basic->gHaptics.m_positionServo);
25   hdlToolButton(&(basic->gHaptics.m_buttonServo));
26
27   // Call the function that copies data between servo side
28   // and client side
29   basic->gHaptics.synch();
30
31   // Only do this once. The application will decide when it
32   // wants to do it again, and call CreateServoOp with
33   // bBlocking = true
34   return HDL_SERVOOP_EXIT;
35 }
36
```

```
37 // This is the entry point used by the application to synchronize
38 // data access to the device. Using this function eliminates the
39 // need for the application to worry about threads.
40 void PhysicsClass::synchFromServo()
41 {
42   hdlCreateServoOp(GetStateCB, this, true);
43 }
44
45 void HapticsClass::synch()
46 {
47   m_buttonApp = m_buttonServo;
48   for(int i=0; i<3; i++)
49     m_positionApp[i] = m_positionServo[i];
50
51   m_forceServo[0] = m_forceServoApp[0];
52   m_forceServo[1] = m_forceServoApp[1];
53   m_forceServo[2] = m_forceServoApp[2];
54 }
```

Listato 3.14: Definizione delle funzioni per la sincronizzazione tra le thread.

La thread che vuole accedere ai dati del dispositivo chiama la funzione synchFromServo(), che farà chiamare GetStateCB dallo scheduler interno dell'HDAL al successivo ciclo servo thread. Il parametro di bBlocking forza synchFromServo() ad aspettare finché il ciclo servo thread è completo.

Quando il successivo ciclo del servo thread viene eseguito, viene chiamata GetStateCB(), che chiama a sua volta la funzione synch(), che salva una copia dei dati tra le variabili che vengono utilizzate solo dal servo thread, e le variabili che sono usate solamente dalla thread dell'applicazione.

La forza da restituire all'utente viene calcolata dal simulatore fisico, cioè dall'altra thread presente nell'applicazione. Per assicurarsi del corretto passaggio delle informazioni sarà proprio la funzione synch() ad eseguire questo

passaggio di variabili. Si hanno quindi varibili che verranno utilizzate solamente dal dispositivo aptico e altre utilizzate dall'applicazione. Il listato 3.15 mostra le dichiarazioni di queste variabili.

```
1  // Variables used only by servo thread
2  double m_positionServo[3];
3  bool  m_buttonServo;
4  double m_forceServo[3];
5  
6  // Variables used only by application thread
7  double m_positionApp[3];
8  bool  m_buttonApp;
9  double m_forceServoApp[3];
```

Listato 3.15: Definizione delle variabili comuni tra la thread del sensore aptico e del simulatore fisico.

3.3.3 Comunicazione con la classe PhysicsClass

La comunicazione con la classe PhysicsClass avviene principalmente tramite tre funzioni: getHapticStateButton, getHapticPosition e setForce.
Nella funzione getHapticPosition, come mostrato nel listato 3.16, si è provveduto a rendere conforme la posizione del Falcon con quella simulata dal simulatore fisico. Infatti gli assi del dispositivo haptico hanno le direzioni dell'asse x e z in verso opposto rispetto a Bullet. Con questa funzione si prendere la posizione attuale, e non quella memorizzata nella variabile m_positionApp, che corrisponde alla posizione del Falcon dell'ultima lettura effettuata.

```
1  void PhysicsClass::getHapticPosition(btVector3& position)
2  {
```

```
3   // Haptic cursor position in "world coordinates"
4   // transform cursor position from device coordinates
5   // to world coordinates
6   double cursorPosWC[3];
7
8   // Must synch before data is valid
9   this->synchFromServo();
10
11  HDLError err = HDL_NO_ERROR;
12    err = hdlGetError();
13    if (err != HDL_NO_ERROR)
14    {
15        char msg[200];
16        sprintf(msg, "HDAL ERROR %d", err);
17        MessageBox(NULL, "hdlCreateServoOp", msg, MB_OK);
18        exit(1);
19    }
20
21    gHaptics.vecMultMatrix(gHaptics.m_positionServo, gHaptics.
          m_transformMat, cursorPosWC);
22
23  position.setValue(-cursorPosWC[0],cursorPosWC[1],-cursorPosWC
        [2]);
24 }
```

Listato 3.16: Funzione getHapticPosition.

Un'altra funzione molto importante è la setForce. Questa funzione andrà a settare i valori della variabile m_forceServoApp, che saranno successivamente impostati nella variabile m_forceServo appena verrà chiamata la funzione synch().

Capitolo 4

Risultati sperimentali

In questo capitolo vengono illustrati i test effettuati e i loro risultati. Tutte le prove sono state eseguite con la telecamera fire-wire Unibrain Fire-i400, posta ad una distanza di 1,3m dall'ambiente. Il marker è stato stampato su normale carta, e ciò comporta alcuni problemi di riflesso della luce. Le prove sono state ottenute con luce ambientale senza particolari accorgimenti.
Come primo esperimento si è valutata la corrispondenza tra l'ambiente reale e l'ambiente virtuale, andando quindi a calcolare l'errore in pixel tra le immagini. Nel secondo test è stata eseguita una valutazione sperimentale dell'ambiente di realtà aumentata con diversi utenti. Come ultimo esperimento viene riportata una analisi preliminare della procedura di riconoscimento di marker e inoltre si mostrano alcuni risultati riguardanti l'interazione di oggetti deformabili all'interno del sistema.

4.1 Valutazione dell'accuratezza dell'ambiente di realtà aumentata

La prima valutazione sperimentale ha riguardato il calcolo dell'errore di corrispondenza tra il mondo virtuale e quello reale. L'errore viene calcolato confrontando la differenza in pixel tra l'immagine acquisita dalla telecamera e l'immagine creata da OpenGL, l'errore in pixel è calcolato con le seguenti relazioni:

$$e_x = \frac{1}{N}\sum_{k=1}^{N} |Ui_{real} - Ui_{virtual}|$$

$$e_y = \frac{1}{N}\sum_{k=1}^{N} |Vi_{real} - Vi_{virtual}|$$

L'errore medio viene calcolato su un numero di 20 punti scelti casualmente. Per calcolare con quali coordinate dell'immagine verrà proiettato un punto nel mondo, e quindi calcolare U_{real} e V_{real}, si utilizza la funzione:

```
project_points2(X, om, T, f, c, k, alpha)
```

messa a disposizione da Matlab. I parametri passati alla funzione corrispondono a:

- X: coordinate del punto nel mondo
- (om,T): Parametri di trasformazione tra le coordinate del mondo e le coordinate della telecamera. om: vettore di rotazione (3×1); T: vettore di traslazione (3×1)
- f: lunghezza focale della telecamera, orizzontale e verticale (vettore 2×1).
- c: punto principale in coordinate pixel (vettore 4×1)

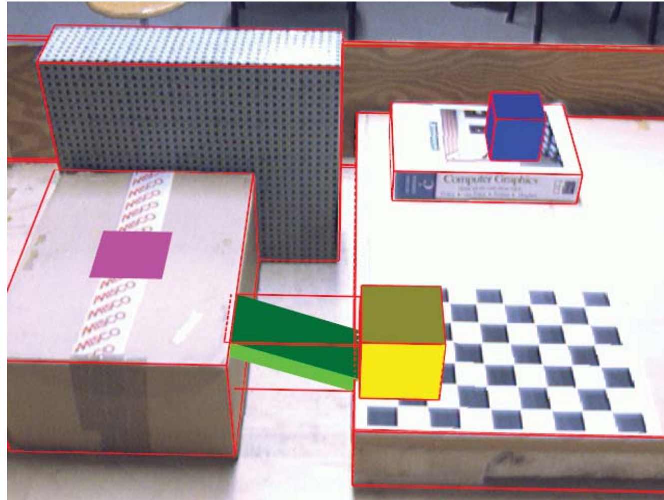


Figura 4.1: L'immagine mostra la corretta corrispondenza tra l'ambiente reale e virtuale.

- k: coefficiente di distorsione (radiale e tangenziale) (vettore 4×1)
- alpha: Coefficiente di skew tra i pixel x e y (se alpha $= 0 \iff$ pixel quadrati)

Per calcolare $U_{virtual}$ e $V_{virtual}$ viene utilizzata la funzione:

```
int gluProject( GLdouble objx, GLdouble objy, GLdouble objz,
    const GLdouble modelMatrix[16], const GLdouble projMatrix[16],
    const GLint viewport[4], GLdouble *winx, GLdouble *winy,
    GLdouble *winz)
```

Questa funzione restituisce i valori delle coordinate del punto sull'immagine nelle variabili winx e winy.

Sono state effettuate molteplici prove, valutando la corrispondenza tra l'immagine è la scena virtuale. L'errore medio trovato sull'asse x è di: 1.69 pixel mentre sull'asse y è stato rilevato un errore medio di: 1.64 pixel.

La figura 4.1 mostra la corretta corrispondenza tra l'ambiente reale e virtuale.

4.2 Esperimenti di interazione

É stata poi condotta una valutazione sperimentale dell'interazione con l'ambiente di realtà aumentata, rilevando sia le impressioni soggettive sia i tempi necessari per eseguire un compito di manipolazione da parte di alcuni utenti. A tal scopo si è utilizzato l'ambiente mostrato in figura 4.2. Agli utenti è stato chiesto di spostare un oggetto virtuale (rappresentato in figura 4.2 dal cubo blu) in una posizione finale (rappresentata dal pannello di colore viola). Il task richiede di spostare il cubo blu spingendolo con la sfera rossa controllata dall'utente mediante il Falcon.
L'interazione tra il cubo blu e gli oggetti della scena avvengono con l'ausilio della simulazione fisica. All'operatore inoltre può o meno essere restituita una sensazione di forza attraverso il Falcon.
L'utente per completare il task deve passare obbligatoriamente sul ponte virtuale che è poco più largo del lato del cubo. Per transitare sul ponte, inoltre l'utente è costretto a spostare il cubo di colore giallo che ne impedisce il passaggio.
Se l'utente spinge il cubo blu in una posizione da cui non è più possibile completare il task, per esempio quando il cubo cade dal ponte, il sistema assegna un errore. Il task viene ripetuto fino a quando l'utente non riesce a completare il percorso. Viene a questo punto memorizzata la durata del tentativo che ha avuto successo. Al raggiungimento della posizione finale, per indicare che il task è stato completato con successo, il cubo cambia colore come mostrato in figura 4.3. Il cubo cambia colore solamente quando sia la

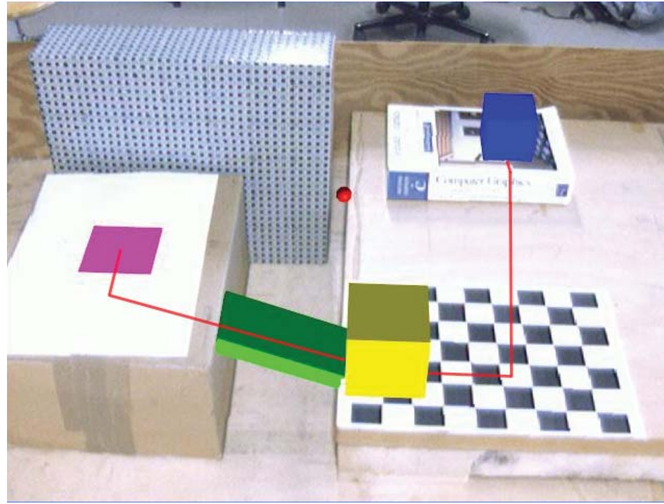


Figura 4.2: Percorso di riferimento per l'esperimento di interazione con gli utenti.

sua orientazione che la posizione sono simili a quelle del pannello che rappresenta la destinazione richiesta. La prova è organizzato in due fasi differenti: nella prima gli utenti eseguono il percorso ricevendo dal Falcon il feedback di forza, mentre nella seconda fase essi eseguono il precorso senza tale feedback. La valutazione è stata condotta da 12 utenti. L'ordine precedentemente indicato (prova con feedback, prova senza feedback) è stato seguito per sei utenti, mentre per i restanti sei sono stati invertite le fasi. La tabella 4.1 mostra i risultati ottenuti da parte di ciascun utente. La tabella 4.2 presenta invece alcuni dati riassuntivi.

Successivamente è stato fatto eseguire il percorso per tre volte a tre utenti mantenendo sempre il feedback di forza. In tal modo si è valutato l'apprendimento degli utenti per quanto riguarda l'utilizzo del Falcon. I risultati di questo secondo esperimento sono riportati in tabella 4.3.

Analizzando i risultati delle tabelle 4.1 e 4.2 si osserva che gli utenti hanno eseguito il percorso con un tempo medio di 53.83 s con deviazione standard di 17.12 s in modalità di ritorno di forza, e di 63.75 s con deviazione standard di

Utente	Modalita	Tempo (s)	Errori
1	con feedback	64	0
	senza feedback	100	0
2	con feedback	52	0
	senza feedback	64	1
3	con feedback	72	0
	senza feedback	39	1
4	con feedback	74	0
	senza feedback	76	0
5	con feedback	32	0
	senza feedback	53	1
6	con feedback	36	0
	senza feedback	53	0
7	senza feedback	75	1
	con feedback	73	0
8	senza feedback	52	0
	con feedback	75	0
9	senza feedback	60	4
	con feedback	39	0
10	senza feedback	45	0
	con feedback	42	0
11	senza feedback	65	0
	con feedback	38	0
12	senza feedback	83	1
	con feedback	53	0

Tabella 4.1: Risultati del primo test effettuato dagli utenti.

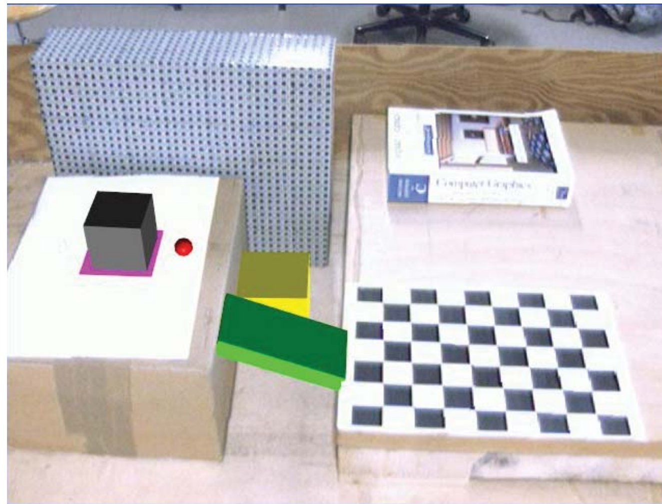


Figura 4.3: Conclusione del task d'interazione con gli utenti.

	Numero di errori	Numero utenti	Tempo media (s)	Tempo dev. std. (s)
senza feedback	9	6	63,75	17,35
con feedback	0	6	53,83	17,12

Tabella 4.2: Risultati riassuntivi del test di interazione.

17.35 s senza tale ritorno. Si sono verificati errori di caduta del cubo dal ponte solamente in modalità di assenza di feedback. Questo dato suggerisce che il passaggio sul ponte è un passaggio delicato che richiede abbastanza precisione nello spingere il cubo. Il risultato evidenzia quindi la minore precisione di movimento in assenza di feedback aptico.

Analizzando i dati in tabella 4.1 osserviamo inoltre tre tipologie di utenti:

- Per 2 utenti la percezione aptica è una sorpresa. Gli utenti eseguono più lentamente.
- Per 3 utenti i tempi sono sostanzialmente equivalenti.

Utente	Prova	Tempo (s)	Errori
1	1 prova	64	0
	2 prova	38	0
	3 prova	35	0
2	1 prova	55	0
	2 prova	51	0
	3 prova	34	0
3	1 prova	45	0
	2 prova	37	0
	3 prova	33	0

Tabella 4.3: Risultato del test di apprendimento con gli utenti.

- Per i rimanenti 7 utenti la percezione aptica fornisce un deciso vantaggio anche in termini di tempi di esecuzione del compito.

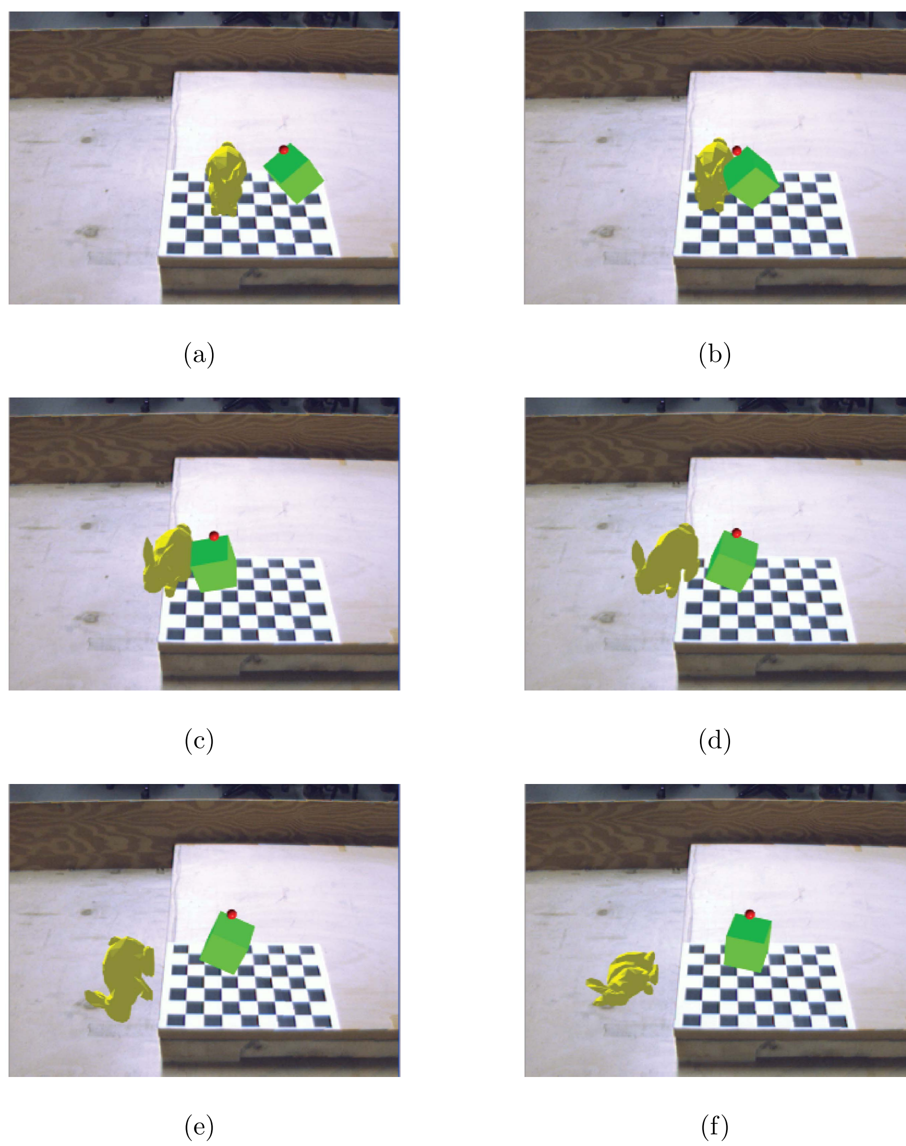
Dopo aver concluso la prova gli utenti hanno dichiarato che è molto più difficile svolgere il percorso senza il ritorno di forza, perché in questo caso non si ha la perfetta percezione dell'ambiente. Infatti senza feedback è possibile che la sfera controllata dall'utente passi attraverso gli oggetti reali della scena perché è il feeedback di forza che impedisce la penetrazione tra i corpi. L'assenza di forza permette agli utenti di spingere l'oggetto virtuale anche più velocemente ma con una precisione molto minore, causando quindi gli errori.

Analizzando i risultati della tabella 4.3 si nota che l'utente acquista maggiore confidenza con il Falcon svolgendo più volte il task. I tempi rilevati

durante tali prove, infatti diventano sempre più ridotti, assumendo un andamento decrescente. Anche se i tre utenti presentano abilità iniziali diverse alla terza prova essi eseguono il compito con tempi molto prossimi suggerendo un buon apprendimento delle modalità di interazione con il sistema.

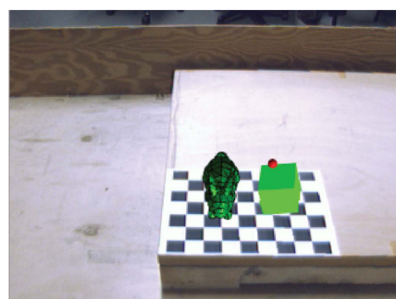
4.3 Valutazione preliminare degli oggetti soft

In questa sezione della tesi sono state eseguite delle valutazioni preliminari di simulazione dell'ambiente in presenza di oggetti deformabili. Utilizzando questi tipi di oggetti si è riscontrato un aumento dell'utilizzo del processore, questo è del tutto normale infatti simulare oggetti deformabili è molto più oneroso che i semplici oggetti rigidi. Le prove effettuate hanno comunque mostrato che il sistema rimane stabile è affidabile senza creare conflitti o rallentamenti troppo evidenti e in reali della simulazione. Nella figura 4.4 vene mostrata una sequenza di collisione tra un cubo e un coniglio rigido mentre nella figura 4.5 si illustra una sequenza di collisione tra un cubo e un coniglio deformabile.

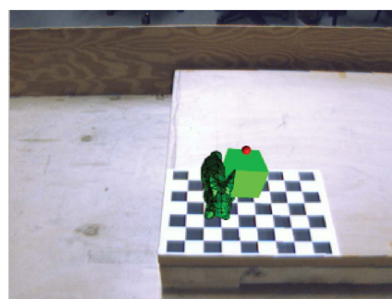


(a) (b) (c) (d) (e) (f)

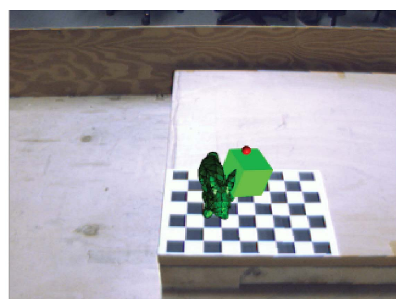
Figura 4.4: Esempio di collisione tra il cubo preso dal Falcon è un coniglio rigido.



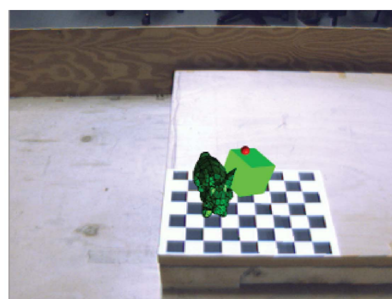
(a)



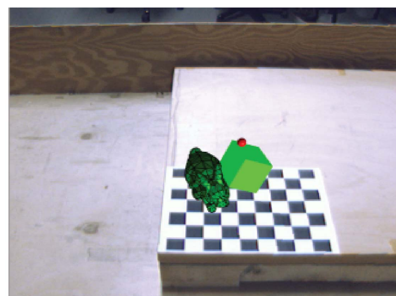
(b)



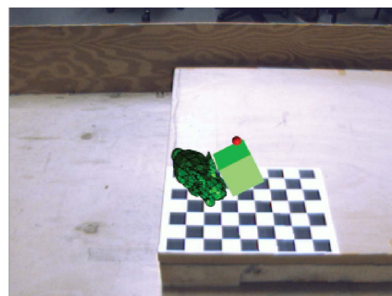
(c)



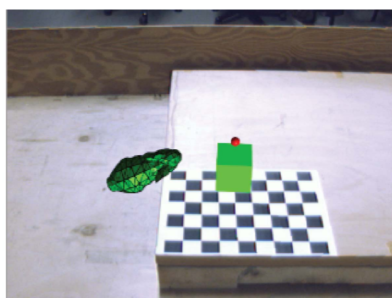
(d)



(e)



(f)



(g)

Figura 4.5: Esempio di collisione tra il cubo preso dal Falcon e un coniglio deformabile.

4.4 Valutazione preliminare del riconoscimento di marker

In questa sezione si sono eseguite delle valutazioni preliminari sul riconoscimento di marker, queste prove hanno mostrato un errore sul riconoscimento dovuto principalmente alla visione mono utilizzata nel sistema, che non permette una precisione adeguata. Si è riscontrato che l'errore cresce maggiormente cresce tanto più il marker viene posto nelle zone periferiche dell'immagine inquadrata dalla telecamera. Questo errore oltre alla visione mono è dovuto anche in parte all'algoritmo di visione utilizzato da ArtoolKit infatti alcune ricerche approfondite hanno mostrato che altri utilizzatori dello stesso sistema hanno riscontrato gli stessi errori con una visione mono. La figura 4.6 mostra comunque due esempi di individuazione di marker che presentano un errore contenuto.

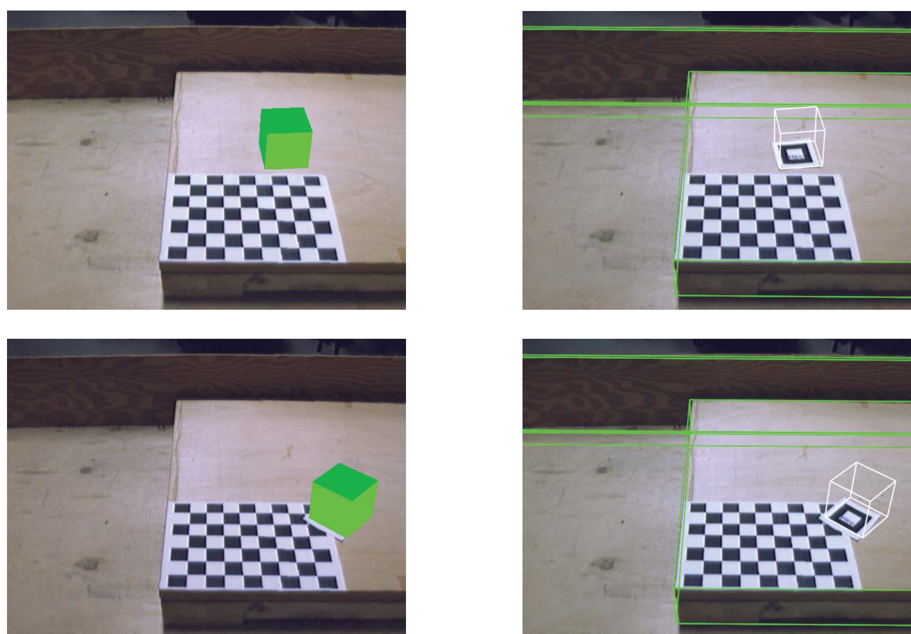


Figura 4.6: Esempi di risultati di riconoscimento dei marker.

Capitolo 5

Conclusioni

Il lavoro descritto in questa tesi ha riguardato il tema delle interfacce uomo-macchine innovative che comprendono sistemi di simulazione grafica 3D interattiva e dispositivi di interazione tattile. In particolare, è stato progettato e realizzato un prototipo di sistema di realtà aumentata, con interazione aptica e simulazione fisica, caratterizzato da un'ottima sincronizzazione tra gli elementi che lo compongono e da stabilità e robustezza.
La realtà aumentata è una delle tecniche più avanzate di computer grafica interattiva. Per migliorare il livello di realismo dell'interazione tra operatore e ambiente creato il sistema integra un simulatore fisico.
Nel prototipo realizzato, per interagire con l'ambiente l'operatore dispone di un'interfaccia aptica, cioè di un dispositivo robotico studiato per interagire direttamente con l'operatore umano, da cui riceve comandi di moto e a cui fornisce in risposta delle sensazioni tattili. Nell'applicazione sviluppata il dispositivo viene programmato per trasmettere all'operatore la forza restituita all'oggetto con cui sta avvenendo il contatto. Il sistema inoltre si avvale di un algoritmo per il riconoscimento di marker in tempo reale mediante telecamera per etichettare oggetti di interesse e consentire l'esecuzione di task in

cooperazione tra più utenti. Tutte queste componenti sono state implementate in un unico sistema con un'architettura suddivisa in moduli.
Grazie alle interazione di molteplici componenti come realtà aumentata, simulazione fisica e interazione aptica, il sistema creato è innovativo per lo stato dell'arte della ricerca nei settori della computer grafica e dell'interazione uomo-macchina. Il prototipo si caratterizza quindi come un sistema innovativo a basso costo, che permette all'utente la manipolazione di oggetti in un ambiente di realtà aumentata.
I test eseguiti hanno evidenziato una buona sincronizzazione tra i vari componenti, una volta superati i problemi di elaborazione in tempo reale e di corrispondenza tra la scena reale inquadrata dalla telecamera e quella virtuale che la riproduce. Gli esperimenti condotti con un gruppo di utenti hanno riguardato il posizionamento di un oggetto in una scena complessa di realtà aumentata mediante l'interfaccia predisposta. I risultati ottenuti hanno evidenziato in maniera molto chiara i benefici della simulazione fisica e dell'interfaccia aptica rispetto a soluzioni più convenzionali.

Uno dei possibili sviluppi futuri del lavoro descritto in questa tesi riguarda il sistema di visione artificiale utilizzato, che effettua la rilevazione di marker mediante visione monoculare. Per un sistema monoculare sia la profondità che le proporzioni fra gli oggetti presenti in una scena sono più difficili da percepire rispetto ad un sistema binoculare e si possono presentare problemi di accuratezza ai bordi della scena. Questo problema è stato riscontrato soprattutto in alcune prove di cooperazione con più utenti. L'utilizzo della visione stereo ridurrebbe gli errori nella visualizzazione dei marker, consentendo quindi anche di creare un sistema di modellazione dell'ambiente tramite la rilevazione dei marker.

Un altro potenziale sviluppo riguarda l'implementazione di feedback di forza nell'interazione con oggetti soft, che attualmente sono stati integrati nell'ambiente ma non interagisco direttamente con il dispositivo aptico utilizzato. Infine, un interessante sviluppo è rappresentato dal possibile utilizzo contemporaneo di due dispositivi aptici, in particolare due Falcon, che permetterebbe azioni combinate come il peg-in-hole.
Al di là di queste possibili estensioni, lo sviluppo più interessante e promettente della tesi è la possibilita di utilizzare il sistema e le tecnologie indagate per un problema di interesse industriale, ad esempio per l'addestramento di operatori in contesti in cui l'impianto da manovrare non è immediatamente disponibile.

Bibliografia

[1] T. Hewett, R. Baecker, S. Cad, T. Carey, et al. ACM SIGCHI Curricula for Human-Computer interaction. ACM SIGCHI Curriculum Development Group, 1992.

[2] J. Preece, J. Rogers, H. Sharp, D. Benyon, H. Holland, and T Carey. *Human-Computer Interaction.* Addison-Wesley, 1994.

[3] Bianchi G., Knoerlein B., Szekely G., and Harders M. High Precision Augmented Reality Haptics. `http://www.vision.ee.ethz.ch/~mharders/EH06a.pdf`.

[4] PHANTOM Omni® Haptic Device. `http://www.sensable.com/haptic-phantom-omni.htm`.

[5] Thomas Bruce, Close Ben, Donoghue John, Squires John, De Bondi Phillip, and Piekarski Wayne. First Person Indoor/Outdoor Augmented Reality Application: ARQuake. *Springer-Verlag London Ltd Personal and Ubiquitous Computing*, 2002.

[6] ARToolKit. `http://www.hitl.washington.edu/artoolkit/`.

[7] OpenGL. `http://it.wikipedia.org/wiki/OpenGL`.

[8] OpenGL Programming Guide or 'The Red Book'. http://fly.cc.fer.hr/~unreal/theredbook/.

[9] COSMO 3D. http://techpubs.sgi.com/library/manuals/3000/007-3445-002/pdf/007-3445-002.pdf.

[10] De Paolis Lucio Tommaso, Pulimeno M., and Aloisio G. The Simulation of a Billiard Game Using a Haptic Interface. In *11th IEEE Symposium on Distributed Simulation and Real-Time Applications*, 2007.

[11] Kitamura Yoshifumi, Douko Kenichi, Kitayama Makoto, and Kishino Fumio. Object Deformation and Force Feedback for Virtual Chopsticks. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, 7-9 November 2005.

[12] Hui Chen, Hanqiu Sun, and Hanqiu Sun. Interactive Haptic Deformation of Dynamic Soft Objects. In *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, 14-17 June 2006.

[13] Bardorfer A., Munih M., Zupan A., and Ceru B. Linear and Circular Tracking Exercises in Haptic Virtual Environments for Hand Control Assessment. In *9th International Conference on Rehabilitation Robotics*, 28 June 2005.

[14] CyberTouch - Immersion. http://www.inition.co.uk/inition/product.php?URL_=product_glove_vti_touch&SubCatID_=26.

[15] V. Hayward, O. Astle, Cruz-Hernandez R., Grant M., and Robles-De-La-Torre D. Haptic Interfaces and Devices. *Sensor Review*, 2004.

[16] K. Erleben. Stable, Robust, and Versatile Multibody Dynamics Animation. Ph.D Thesis, Department of Computer Science, University of Copenhagen, 2004.

[17] B. Mirtich. Impulse-based Dynamic Simulation of Rigid Body Systems. PhD Thesis, University of California, Berkeley, 1996.

[18] D. Baraff. Dynamic Simulation of Non-Penetrating Rigid Bodies. PhD Thesis, Computer Science Department, Cornell University, 1992.

[19] D. Baraff. Physically Based Modeling: Principles and Practice. Carnegie Mellon University, 1997.

[20] J. Ratcliff. Automatic Generation of Dynamics Models. Game Developers Conference 2007 Course Notes, 2007.

[21] Boeing A. and Bräunl T. *Evaluation of real-time physics simulation systems*. Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia, 1-4 December 2007.

[22] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision Second Edition*. CAMBRIDGE, 2003.

[23] Bullet physics library. `http://www.bulletphysics.com`.

Ringraziamenti

Eccomi giunto finalmente all'ultima pagina di questa tesi.

Per primo vorrei ringraziare il Prof. Stefano Caselli per avermi dato la possibilità di lavorare su questo fantastico progetto e per la sua disponibilità costante.
Un altro importante ringraziamento ufficiale va all' Ing. Jacopo Aleotti per i suoi preziosi consigli durante tutto lo svolgimento di questa tesi, e per avermi aiutato nella sua stesura.

Ringrazio tutte le persone conosciute nella palazzina 1 e in particolare nel Laboratorio di Robotica, e tutti i compagni di corso con cui ho condiviso questi due anni di università.

Ringrazio anche la mia famiglia per il sostegno e la fiducia dimostrata in questo lungo cammino, ed infine un ringraziamento speciale a Chiara.