UNIVERSITÀ DEGLI STUDI DI ROMA "TOR VERGATA"

FACOLTA' DI INGEGNERIA MECCANICA

DOTTORATO DI RICERCA IN
PROGETTAZIONE DEI SISTEMI MECCANICI

XXI CICLO

TESI DI DOTTORATO

La Realtà Aumentata a supporto dell'Ingegneria Virtuale

CANDIDATO

Davide Gattamelata

A.A. 2008/2009

Tutor: Prof. Eugenio Pezzuti

Coordinatore: Prof. Carlo Brutti

Abstract

The Augmented Reality is an emerging field of computer graphics. It deals with the combination, in real time, of real world images with computer generated data.
This work concerns several approaches to integrate computer aided engineering instruments into Augmented Reality Environment. Applications about real-time mechanism motion simulation, maintenance, structural and fluid dynamics analysis post-processing, reverse engineering and CAD modeling are presented as well. Each application has been developed in Augmented Reality platform in order to improve the Virtual Engineering procedures of design and simulation.

keywords: Augmented Reality, CAD, Reverse Engineering, Prototyping, Post processing, C++, Virtual Engineering, Maintenance

La Realtà Aumentata è una disciplina della Computer Graphics che studia sistemi in grado di sovrapporre informazioni e oggetti virtuali, generati al computer, ad un flusso di immagini video provenienti dal mondo reale.
Il lavoro svolto riguarda lo sviluppo di applicazioni di ingegneria virtuale su piattaforme di Realtà Aumentata. Le applicazioni si riferiscono al post-processing di analisi fluidodinamiche, cinematiche e strutturali, alle simulazioni multibody, al disegno assistito dal calcolatore e all'ingegneria inversa.
L'obiettivo è quello di migliorare, attraverso la Realtà Aumentata, l'interfaccia e le funzionalità delle applicazioni di ingegneria virtuale dedicate alla progettazione dei sistemi meccanici.

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

Albert Einstein

Indice

Introduzione

Nell'ultimo trentennio l'informatica ha contribuito notevolmente allo sviluppo dei sistemi di progettazione meccanica. Parallelamente all'evoluzione delle risorse hardware sono nate diverse applicazioni di disegno e progettazione assistita dal calcolatore.

Di fatto la nascita della *Realtà Virtuale* ha determinato quella successiva dell'*Ingegneria Virtuale*, intesa come l'integrazione di tutte le tecnologie di modellazione volte a simulare il ciclo di vita di un prodotto. I moderni software di progettazione possiedono infatti una struttura informatica in grado di contenere le fasi di prototipazione, verifica, realizzazione e documentazione relative ai componenti meccanici.

Inoltre, il progresso tecnologico dei sistemi informatici ha influito positivamente sull'evoluzione delle prestazioni e delle funzionalità dei software di progettazione assistita dal calcolatore. Tale evoluzione non ha però trovato riscontro nel miglioramento delle interfacce. La classica configurazione desktop (*tastiera, mouse e monitor*) è tuttora la più diffusa nei reparti di progettazione. I limiti di tale struttura riguardano la mancanza di un contatto tridimensionale con la scena: il progettista è limitato dall'elaborare i dati del modello virtuale nello spazio bidimensionale del monitor e non può puntare ad elementi reali se non attraverso macchinose e poco intuitive operazioni di ingegneria inversa.

La Realtà Aumentata, una disciplina informatica che si occupa della combinazione della realtà virtuale dei computer con il mondo reale, sembrerebbe avere le

caratteristiche giuste per superare la mancanza di interattività delle attuali applicazioni dedicate alla progettazione meccanica.

Senza la pretesa di aprire un nuovo settore di applicazioni informatiche per l'ingegneria, l'obiettivo di questo lavoro è quello di valutare l'utilità dell'introduzione della Realtà Aumentata nella progettazione dei sistemi meccanici. Partendo dalle problematiche di reverse engineering e prototipazione virtuale sono state sviluppate applicazioni cinematiche, fluidodinamiche e strutturali.

La prima parte del lavoro è dedicata alla descrizione della Realtà Aumentata: partendo da un confronto storico di questa disciplina con la Realtà Virtuale si passa a definirne le caratteristiche generali e lo schema di funzionamento.

Nel secondo capitolo vengono descritte le caratteristiche di uno specifico sistema di Realtà Aumentata, basato sul riconoscimento visivo di entità fiduciali, e le librerie grafiche *Artoolkit*, librerie Open Source utilizzate per sviluppare le applicazioni presentate in questo lavoro.

Il terzo capitolo è dedicato alla presentazione di applicazioni, in ambiente di Realtà Aumentata, che riguardano la simulazione dei sistemi meccanici. In particolare i settori della progettazione assistita dal calcolatore, del reverse engineering, della simulazione cinematica e dinamica e infine il post-processing, di analisi fluidodinamiche, strutturali e dinamiche.

Capitolo 1 **La Realtà Aumentata**

La Realtà Aumentata (AR, dall'inglese Augmented Reality) è una disciplina della computer graphics che studia e sviluppa sistemi in grado di proiettare oggetti e informazioni virtuali combinate con immagini provenienti dal mondo reale. L'impiego di questa tecnologia stravolge il modo di interagire con il mondo.
L'utente di un'applicazione di AR, indossando o utilizzando opportuni strumenti, è nella condizione di vivere un'esperienza sensoriale arricchita di informazioni ed elementi virtuali con cui interagire. In questo capitolo si introduce la tecnologia della Realtà Aumentata, si discutono le applicazioni maggiormente diffuse e si illustra l'evoluzione storica di questa disciplina.

1.1 Cenni Storici

Nel corso dei secoli, le tecniche di comunicazione tra gli uomini hanno subito sviluppi legati alle scoperte scientifiche contemporanee. Nell'ultimo secolo l'avvento dei computer e il nascere di discipline dedicate all'informatica grafica hanno aggiunto, ai tradizionali metodi di comunicazione, la comunicazione visiva e interattiva con la realtà virtuale generata dal computer.

La Realtà Virtuale VR (dall'inglese Virtual Reality) ha lo scopo di imitare, attraverso simulazioni sviluppate per scopi specifici, il mondo reale. Il primo ad applicare il concetto di VR, in ambito cinematografico, fu Morton Eilig, che nel 1957-1962 costruì *Sensorama*, un impianto in grado di coinvolgere tutti i sensi in maniera realistica,

durante la proiezione di un film. Qualche anno più tardi 1989, Jaron Lanier fondatore della VPL research coniò il termine Virtual Reality. Più recenti ma analoghi i termini di Virtual Reality e CyberSpace Virtual World e Virtual Environments. Si parla di realtà virtuale in riferimento a tutte quelle tecniche che permettono di indurre esperienze sensoriali di luoghi e oggetti, reali o immaginari. che vengono simulati per mezzo di tecnologie informatiche.

La AR è una tecnologia informatica strettamente legata, fin dalla sua nascita alla VR.

Infatti, nel 1965 Ivan Sutherland [3] teorizzò la possibilità di costruire ambienti sintetici generati dal computer e gettò le basi teoriche per lo sviluppo di sistemi di VR con la seguente definizione di display interattivo:

"…The ultimate display would, of course, be a room within which the computer can control the existence of matter. A chair displayed in such a room would be good enough to sit in. Hand cuffs displayed in such a room would be confining, and a bullet displayed in such a room would be fatal. With appropriate programming, such a display could literally be the Wonderland into which Alice walked. "

In altri termini evidenziò l'obiettivo di creare un sistema in grado di generare stimoli artificiali generando nell'osservatore l'illusione di vivere un'esperienza reale.

Tre anni dopo (1968) Sutherland costruì, con l'aiuto di un suo studente Bob Sproull, quello che è universalmente riconosciuto come il primo sistema di Realtà Aumentata [4] della storia (v. Figura 1.1.1).

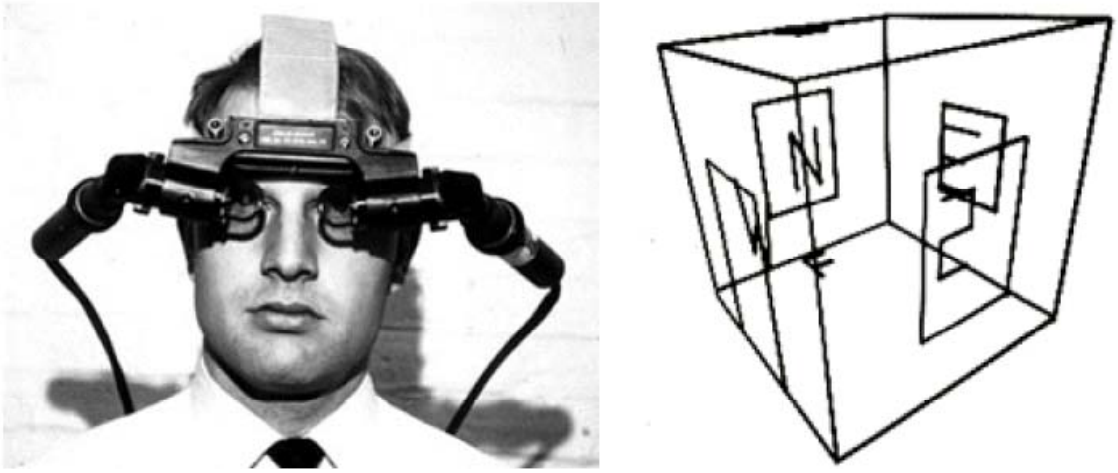


Figura 1.1.1. Il primo *Head Mounted Display* sviluppato da Ivan Sutherland (1968).

Ovvero un dispositivo a forma di casco *HMD* (dall'inglese head mouted display) in grado di proiettare, per mezzo di lenti argentate, immagini virtuali sul campo visivo dell'osservatore.

Figura 1.1.2. Realtà aumentata nelle operazioni di cablaggio della Boeing

Da allora in poi le due discipline informatiche hanno evoluzioni storico-tecnologiche differenti. La VR attira l'interesse dei ricercatori informatici e l'obiettivo di realizzare un ambiente totalmente immersivo, in grado di manipolare le sensazioni percepite dall'utente, motiva lo studio di nuove tecnologie. La AR invece non viene sviluppata fino agli inizi degli anni 90, quando Tom Caudell, riferendosi ad un applicazione della Boeing per l'assemblaggio di Aircraft (v. Figura 1.1.2), coniò la dicitura *Augmented Reality* ed è solo da allora che si è iniziato a percepirne il potenziale informatico.

La relazione tra Realtà Aumentata e Realtà Virtuale è tuttora fonte di dibattito. Nonostante le definizioni formali fornite dagli esperti di computer graphics nel corso degli anni [9][10] le due discipline, pur attingendo allo stesso bagaglio di conoscenze, sono concettualmente diverse. In sintesi si può affermare che mentre la Realtà Virtuale riguarda l'immersione in un mondo sintetico, la AR si interessa alla sovrapposizione di elementi virtuali sul mondo reale.

1.1.1 Realtà Virtuale e Realtà Aumentata

Nel corso dell'ultimo ventennio diverse definizioni e classificazioni sono state introdotte per le applicazioni di Realtà Aumentata. Formalmente esistono due correnti di pensiero: quella dei ricercatori che definiscono la Realtà Aumentata come un caso della Realtà Virtuale e quelli che sostengono il contrario.

Tuttavia per le applicazioni ingegneristiche, che interessano questo lavoro, gli aspetti da considerare, confrontando la Realtà Virtuale con la AR, sono due: il primo riguarda l'approccio delle due tecnologie con il mondo reale, mentre il secondo riguarda i sistemi hardware impiegati.

La Realtà Virtuale non tiene conto del mondo reale a differenza della AR, che invece si basa sull'ambiente reale in cui si trova l'utente dell'applicazione.

Dal punto di vista hardware la Realtà Virtuale vanta un maggior numero di applicazioni e anni di sviluppo rispetto alla più recente AR, questo spinge verso l'utilizzo dei dispositivi e delle tecnologie testate nei sistemi di Realtà Virtuale per le applicazioni di AR.

1.2 Schema generale di un sistema AR

Come accennato nelle sezioni precedenti il concetto chiave di un'applicazione di AR risiede nella corretta combinazione di immagini provenienti dal mondo reale con oggetti virtuali. Il problema fondamentale della AR è quello della corretta collimazione tra i due mondi, ciò interessa tecnologie e tecniche differenti in funzione della disponibilità di risorse economiche e del contesto di utilizzo dell'applicazione stessa.

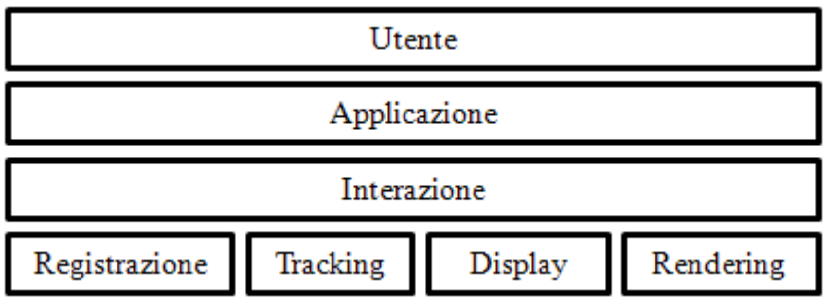


Figura 1.2.1. Schema generale di un sistema di Realtà Aumentata.

È difficile classificare i diversi sistemi finora realizzati differenziandoli in funzione dell'hardware impiegato o delle tecniche di implementazione, mentre risulta semplice

individuare uno schema di riferimento (v. Figura 1.2.1) per i processi che generalmente caratterizzano i sistemi di AR.

Seguendo lo schema di Figura 1.2.1 a livello più basso abbiamo i processi base di *Tracking, Registrazione, Visualizzazione e Rendering* che vengono di seguito illustrati.

Il *Tracking* si occupa del tracciamento della posizione dell'osservatore rispetto alla scena. In altri termini tale processo fornisce in tempo reale la posizione del punto di vista dell'osservatore rispetto ad un sistema di riferimento globale assegnato per convenzione all' ambiente reale in cui l'osservatore si trova.

Esistono sistemi di tracking di diverso genere, i più importanti verranno discussi nel paragrafo1.2.2.

La *registrazione* è il processo che si occupa di allineare gli oggetti virtuali al punto di vista dell'osservatore sulla scena, applicando le opportune trasformazioni geometriche.

Accanto alla registrazione e il tracking c'è il processo di *Visualizzazione*. Per tale funzionalità i dispositivi di visualizzazione (display) sono il supporto principale. La tipologia di dispositivo dominante è quella degli *Head Mounted Display*, i quali consentono una elevata mobilità e un veloce allineamento del punto di vista dell'osservatore con la scena proiettata.

Il terzo processo base è il *rendering* e si interessa direttamente della sovrapposizione degli elementi grafici sulle immagini reali. Le proprietà che caratterizzano tale processo sono la velocità di aggiornamento delle immagini prodotte e la capacità di produrre immagini fotorealistiche. Velocità di aggiornamento e qualità dell'immagine prodotta sono proprietà contrapposte, infatti cercare un sistema efficiente per la velocità di aggiornamento dell'immagine prodotta significa rinunciare a tecniche di rendering più sofisticate quali il ray-tracing e il radiosity[1].

Passando dal livello base dei processi delle applicazioni AR (v. Figura 1.2.1) al livello superiore si arriva al processo interattivo. Tale processo, ove presente, necessita di strumentazione sviluppata ad hoc. In tal senso sensori, sistemi di tracciamento della posizione e accelerometri consento di estendere l'interazione con la scena anche ad altri

[1] Il processo di rendering delle librerie grafiche *OpenGL* è trattato in maniera più diffusa nell'Appendice A Le librerie grafiche OpenGL.

sensi, oltre a quello visivo, e come sarà illustrato nelle sezioni successive anche di creare e manipolare oggetti virtuali puntando agli oggetti reali presenti nella scena.

1.2.1 I display

I sistemi di visualizzazione, finora sviluppati, per le applicazioni di AR, sono raggruppabili in tre categorie:

- *head-attached displays*: sono i dispositivi tipo casco indossati sulla testa dall'osservatore;
- *hand held displays*: sono i visualizzatori da tenere in mano, tipo palmari e cellulari;
- *spatial displays*: display spaziali spesso utilizzati per applicazioni destinate ai musei*;*

Tra gli *Head Attached Display* si distinguono tre metodi differenti di implementazione in funzione dei dispositivi impiegati per la visualizzazione della scena. Sono i display più efficaci e pratici, grazie alla sensazione di immersione che generano sull'utente, e sono denominati con l'acronimo *HMD* (*Head Mounted Display*). Tra questi, i display che utilizzano piccoli visori ottici sono denominati *Head Mounted Projector* e sono di due tipi :

- *Optical See Through*
- *Video See Through.*

Rientrano nella categoria anche i cosidetti *retinal display* che utilizzano i laser per generare i contenuti sintetici da visualizzare.

I dispositivi Optical See Through utilizzano un visore di fascio ottico, consistente in uno specchio traslucido che trasmette la luce in una direzione e contemporaneamente la riflette nell'altra. Si basa su una tecnologia parzialmente trasmittente che permette di guardare contemporaneamente l'immagine virtuale sovrapposta alla vista reale.

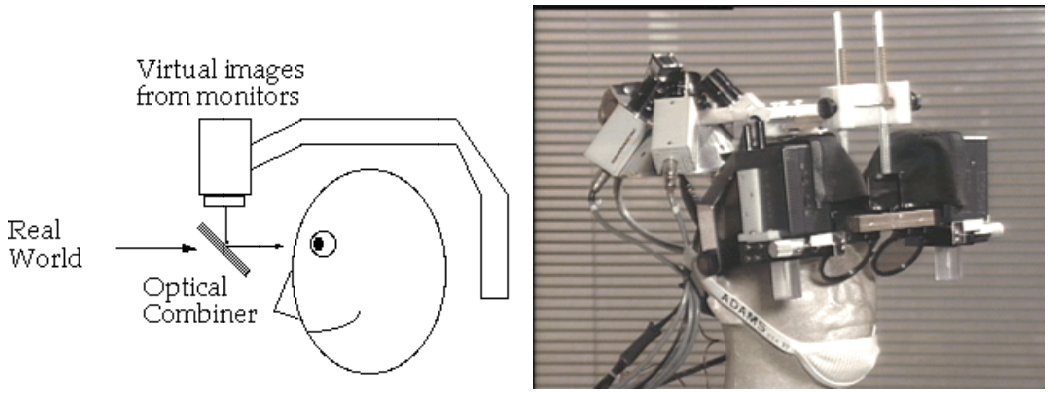


Figura 1.2.2: Esempio di HMD display di tipo optical see through. A sinistra schema di funzionamento; a destra prototipo.

Tali visori sono molto simili agli *Head Up Display* utilizzati dai piloti degli aerei americani. Una peculiarità di questi sistemi è che riducono l'intensità di luce del 30%.

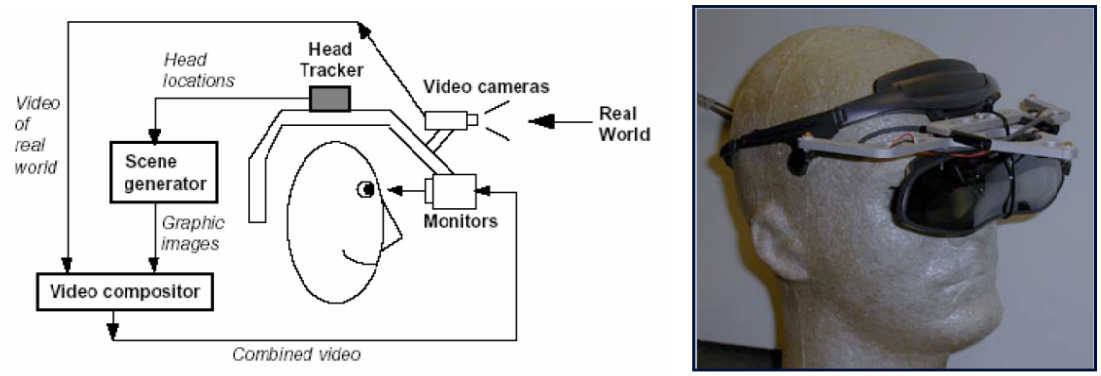


Figura 1.2.3. Esempio di HMD display di tipo Video See Through. A sinistra: schema di funzionamento; a destra: prototipo.

I visori di tipo Video See Through usano invece due telecamere, una per ciascun occhio, con le quali acquisiscono l'immagine reale, le immagini vengono inviate all'unità di elaborazione che le proietta sui display, uno per ciascun occhio, arricchite delle informazioni sintetiche. La scelta di questo tipo di dispositivo consente di realizzare effetti visivi più complessi. Tale tecnologia impone un piano di messa a fuoco costante su tutta la scena, caratteristica che ne limita il comfort.

Figura 1.2.4. Esempio di applicazione AR su display palmare.

Gli *hand held display* sono dispositivi portatili quali palmari (PDA) [17], cellulari, display a specchio e video proiettori. Tali dispositivi sebbene efficaci dal punto di vista della portabilità presentano alcuni limiti funzionali [2], quali la ristretta mobilità rispetto agli *HMD* e le basse prestazioni grafiche. L'incapacità quindi di elaborare velocemente le immagini video provenienti dalla telecamera e integrare elementi virtuali che richiedono elaborazioni grafiche onerose.
L'altro limite di questi dispositivi proviene dalle ridotte capacità delle lenti che non permettono di variare la profondità di proiezione e dall'angolo di vista limitato dal visualizzatore, che di solito è di piccole dimensioni.

A differenza dei dispositivi appena visti, i display di tipo spaziale non richiedono all'utente di indossare alcuna strumentazione. L'apparecchiatura è in questo caso installata direttamente nell'ambiente oggetto dell'osservazione. Questa tecnologia di visualizzazione prevede tre tipi differenti di visualizzatori (v. Figura 1.2.5):

- I dispositivi *Video See Through*;
- I dispositivi *Optical See Through*;
- I dispositivi diretti;

i dispositivi di tipo see through utilizzano una videocamera per l'acquisizione e una configurazione PC di tipo desktop, le immagini aumentate vengono proiettate direttamente sul monitor del PC. Tali dispositivi utilizzano combinatori ottici posizionati nell'ambiente, in grado di generare immagini allineate con il punto di vista dell'osservatore. Esistono tipologie differenti di combinatori ottici: specchi, schermi trasparenti e ologrammi (v. Figura 1.2.5). Gli ologrammi registrano una scena captando ampiezza, lunghezza d'onda e fase. Questo consente di ricostruire il fronte d'onda visivo e generare uno scenario tridimensionale, osservabile da diversi punti di vista [15].

Gli svantaggi di questo tipo di visualizzatori sono legate all'impossibilità di creare applicazioni portatili, poiché sono legati allo spazio di implementazione. Inoltre consentono un'interattività limitata.

Figura 1.2.5. Esempi di visualizzatori spaziali; a specchio (sinistra), a schermo trasparente (centro) e ad ologramma (destra).

1.2.2 Il tracking

La coerente sovrapposizione di immagini virtuali su fotogrammi reali è requisito fondamentale per le applicazioni di AR. Gli oggetti virtuali devono essere accuratamente registrati in tutte le loro dimensioni.

Questo è un requisito fondamentale per talune applicazioni. Infatti mentre la registrazione non corretta in un'applicazione di intrattenimento pecca per efficacia di comunicazione, lo stesso difetto ha conseguenze disastrose su operazioni chirurgiche, praticate con il supporto di applicazioni di AR. In tale ambito è fondamentale non solo disporre di una corretta registrazione quando l'utente è fermo, ma anche quando si trova in movimento rispetto alla scena. A tal fine è necessario che tanto l'orientamento quanto la posizione della testa dell'osservatore siano rilevate in tempo reale.

Inoltre il problema del tracking, in un applicazione di AR che preveda l'interazione con la scena, si estende anche ai dispositivi di puntamento[2].

Per il processo di tracking si distinguono due categorie di sistemi:

- *Outside-in*;
- *Outside-out*;

[2] Nel 3°Capitolo vengono discusse alcune tecnologie di interazione (magnetiche e ottiche) per applicazioni di ingegneria inversa e disegno assistito.

La distinzione tra le due tipologie è basata sulla configurazione di sensori ed emettitori rispetto agli oggetti da puntare. I sistemi di tipo Outside-in hanno i loro sensori disposti su posizione fisse all'interno della scena. Gli oggetti da seguire sono forniti di emettitori e marker[3]. D'altra parte si trovano i sistemi di tipo Outside-out che utilizzano sensori installati direttamente sugli oggetti da seguire.

I requisiti principali per i sistemi di AR, riguardo alla funzionalità del tracking, possono essere riassunti nella seguente lista [33]:

- Accuratezza di misura, sia per l'inclinazione che per la posizione;
- Buona velocità di acquisizione (minimo 30 Hz);
- Mobilità dell'utente, evitare il più possibili cavi o spazi di lavoro ristretti;

Diversi approcci sono stati seguiti per sviluppare i sistemi di tracking, ognuno dei quali presenta caratteristiche positive o negative in funzione della tecnologia impiegata. A volte una buona strategia consiste nell'integrare sistemi diversi affinché i limiti di un sistema vengano compensati dalle qualità dell'altro e viceversa.

Di seguito vengono presentati quattro approcci diversi al problema del tracking [7]:

- Il tracking inerziale;
- Il tracking acustico;
- Il tracking magnetico;
- Il tracking ottico;

Il tracking inerziale utilizza giroscopi e accelerometri per determinare la posizione e l'orientamento nello spazio di dispositivi inerziali. Questi dispositivi non forniscono una misura diretta ma necessitano di una fase di elaborazione. Le accelerazioni vengono rilevate misurando le variazioni sulla differenza di potenziale e vengono integrate due volte per ricavare i parametri cinematici dell'oggetto da tracciare. Tale processo è soggetto al problema della deriva, che conduce alle frequenti calibrazioni dell'apparecchiatura. D'altro canto il tracking inerziale non è influenzato da eventuali interferenze magnetiche, non pone restrizioni sulle dimensioni dell'ambiente di funzionamento, ma non risulta molto accurato per piccoli spostamenti. In Figura 1.2.6 si riporta l'immagine di un sistema di tracking inerziale realizzato dalla Xsens Motion Technology di dimensioni relativamente contenute.

---

[3] I marker sono dispositivi applicati agli oggetti presenti nella scena osservata. Vengono utilizzati per individuare, nelle immagini digitalizzate, la posizione degli oggetti al variare della posizione dell'osservatore e/o dell'oggetto osservato.

Figura 1.2.6. Il sistema inerziale della *Xsens Motion Technologies.*

I sistemi di tracking acustici basano il loro funzionamento sull'emissione di onde ultrasoniche alla frequenza di 20kHz, valore al di sopra delle frequenze riconosciute dall'apparato uditivo umano. Utilizzando una sola coppia di trasmettitori e ricevitori, si riesce a rilevare la distanza tra il punto da tracciare e un punto fisso. Per avere informazioni sulla posizione nello spazio è necessario disporre di un trasmettitore e tre ricevitori oppure tre trasmettitori ed un ricevitore. Mentre tre trasmettitori e tre ricevitori sono necessari per stimare sia la posizione che l'orientamento. In questo tipo di sistemi la distanza viene calcolata utilizzando la lunghezza d'onda e i dati rilevati inerenti al tempo di risposta.

I pregi principali di questi sistemi sono la leggerezza e la indipendenza da disturbi dovuti ad interferenze magnetiche. Per contro tali sistemi subiscono l'influenza di interferenze acustiche. Inoltre l'accuratezza della misura è dipendente dalle condizioni ambientali, in quanto la velocità di propagazione delle onde nell'aria varia con la densità di quest'ultima. Infine la necessità, che questi sistemi hanno, di avere sempre la visuale libera tra trasmettitori e ricevitori ne limita l'affidabilità.

Un esempio di dispositivo di tracking acustico è riportato in Figura 1.2.7. È un apparecchio realizzato dalla *intersense;* l'*IS-900* utilizza un emettitore sonoro e dei dischi per emettere e ricevere ultrasuoni.

Figura 1.2.7. Il dispositivo di tracking acustico *IS*-900, della *Intersense*.

Il tracking magnetico si basa sulla misura del campo magnetico e utilizza sia onde a bassa frequenza che pulsate. È composto da un ricevitore e da un trasmettitore. Il trasmettitore, per generare il campo magnetico, contiene tre spire avvolte su un magnete. Il ricevitore a sua volta contiene tre spire nella stessa configurazione dell'emettitore. Per ogni ciclo di misura il ricevitore rileva, per ciascuna spira dell'emettitore, l'entità del campo magnetico, per un totale di nove misurazioni. I dati delle misure vengono elaborati per calcolare la posizione e l'orientamento del ricevitore rispetto all'emettitore.
A differenza di altri dispositivi di tracking, quello magnetico non è sensibile a problemi di occlusione emettitore-ricevitore. Inoltre consente di lavorare su spazi ampi. Tuttavia i principali svantaggi connessi all'impiego di questa tecnologia riguardano la sensibilità alle interferenze magnetiche causate da onde radio e superfici metalliche. Infine l'accuratezza di misura diminuisce con l'aumentare della distanza. Un esempio di dispositivo magnetico è riportato in Figura 1.2.8 ed è costituito da un sottile ricevitore e da una unità di potenza che gestisce le misure. Il sistema in oggetto supporta fino a quattro ricevitori simultaneamente.

Figura 1.2.8. Il dispositivo di tracking magnetico realizzato da Polhemus Frastrak.

I sistemi meccanici di tracking misurano gli angoli e le lunghezze tra i giunti del dispositivo ed utilizzano una configurazione base come posizione iniziale rispetto alla quale misurare i movimenti degli oggetti puntati.
Questo tipo di dispositivi si divide in due gruppi: quelli fissi e quelli installati sui corpi. Nel primo caso un punto del tracciatore è fissato al terreno in una posizione nota, mentre nel secondo caso gli apparati di misura sono installati su un esoscheletro.

Figura 1.2.9. Esempio di dispositivo di tracking meccanico, Gypsy 4.

Questo tipo di tecnologia non è sensibile alle interferenze magnetiche e a problemi di visibilità della strumentazione rispetto all'unità di elaborazione. Inoltre è in grado di fornire misure precise, sia sulla posizione che sull'orientamento. Per contro sono sistemi piuttosto invasivi, ingombranti e consentono spazi di lavoro limitati.
Un esempio di questo tipo di tracciatori è riportato in Figura 1.2.9, il *Gypsy4*: è un sistema di tracking per il corpo umano, è costituito da 43 sensori di movimento e da 17 coppie cinematiche e supporta il tracking multiplo.

I dispositivi di tracking ottici sfruttano la luce per stimare la posizione nello spazio dell'oggetto da puntare. Ogni punto nel piano dell'immagine genera un raggio di luce dal pixel al centro di proiezione. Al crescere della distanza tra trasmettitore e ricevitore l'energia del raggio di luce diminuisce con il quadrato della distanza. Come ricevitori si possono utilizzare sensori CCD (*Charged Coupled Device*) o fotodiodi laterali, in funzione della tipologia di dispositivi che viene utilizzato, passivo o attivo. Esempi tipici di dispositivi passivi sono i marker e di dispositivi attivi i led.

Nella categoria dei sistemi di puntamento ottici rientrano i sistemi di riconoscimento dell'immagine che utilizzano algoritmi di grafica computazionale per elaborare immagini, contenenti gli oggetti da seguire, e calcolarne la loro posizione rispetto al punto di vista della camera sulla scena.
Il sistemi di tracking di tipo ottico assicurano velocità di elaborazione adeguate e non pongono limiti sullo spazio di osservazione. Per contro sono sensibili alla maggiore o minore visibilità degli oggetti da osservare e all'intensità della luce.
Nelle librerie grafiche Artoolkit, utilizzate per lo sviluppo di sistemi di Realtà Aumentata, è implementato un sistema di tracking ottico che utilizza marker per rilevare la posizione e l'orientamento della camera rispetto ai marker.[4]

L'analisi dei dispositivi di tracking finora svolta evidenzia, per ciascun sistema, limiti e qualità. Una scelta vincente potrebbe essere quella di combinare diversi dispositivi in modo tale che le caratteristiche migliori di ciascun sistema siano in grado di sopperire alle mancanze degli altri. Sebbene questa appaia una strategia vincente, l'integrazione di più strumenti fa crescere la complessità dei sistemi e conseguentemente rende più complicata la gestione e il controllo dell'applicazione.
Inoltre le tecnologie illustrate sono state sviluppate per ambienti chiusi, ovvero per ambiente limitati facilmente controllabili. L'utilizzo dei sistemi di tracking per applicazioni in ambienti aperti avviene in una situazione diametralmente opposta: gli ambienti aperti sono fuori controllo e richiedono la portabilità della strumentazione.
Infine è necessario considerare il processo di tracking per applicazioni di AR anche nell'ambito del design collaborativo, dove ciascun utente deve poter interagire autonomamente con lo scenario condiviso. Questo problema definisce un obiettivo e una funzionalità diversa rispetto alle applicazioni di AR finora trattate. In questo ambito occorre sviluppare applicazioni server in grado di gestire coerentemente il processo di tracking per più utenti e strumenti [57].

[4] Le Artoolkit sono descritte più dettagliatamente nel Capitolo 3.

1.2.3 Applicazioni

Dall'inizio degli anni '90 la possibilità offerta dalla AR di far coesistere informazioni reali e virtuali ha trovato applicazione in diversi settori di ricerca [8][20].

Come spesso accade nella ricerca, la protipazione e la sperimentazione di nuove tecnologie inizia nel settore militare e viene poi introdotta in ambito civile.

L'esercito americano per primo ha introdotto la Realtà Aumentata con una tecnologia di visualizzazione denominata *Head Up Display* (v. Figura 1.2.10) di ausilio ai piloti dell'aeronautica per proiettare informazioni relative al pilotaggio di aeromobili.

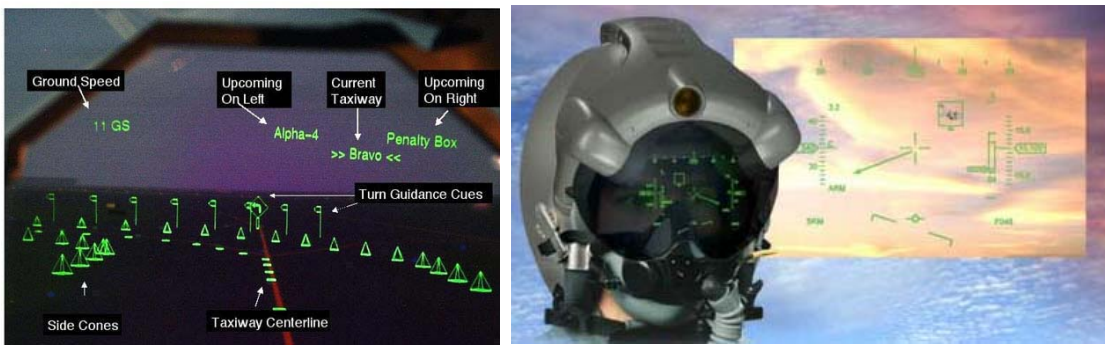


Figura 1.2.10. Display di tipo Head-Up per applicazioni militari.

Un'altra applicazione di AR in campo militare è la simulazione volta all'addestramento dei soldati. In tale ambito un esempio significativo è il SIMNET, un diffuso sistema di simulazione di giochi di guerra che prevede l'equipaggiamento del personale militare con caschi provvisti di visori, mediante i quali si proiettano le attività delle altre unità che partecipano all'esercitazione. Lo scenario di guerra può essere aumentato con informazioni supplementari o mettendo in evidenza unità nemiche nascoste. L'Università di Rochester partecipa al progetto di video sorveglianza e monitoraggio (VSAM), finanziato dalla Defense Advance Research Projects Agency (DARPA). La Figura 1.2.11 mostra lo scenario per l'utilizzo della AR in questo progetto: una unità di ricognizione aerea genera marker di riferimento per la registrazione. Autonome unità con equipaggiamento per la video sorveglianza controllano la stessa area. La vista aerea è arricchita dai dati provenienti dalle unità di sorveglianza. Un altro scenario riguarda la ricognizione aerea in grado di identificare oggetti sospetti e trasmetterne la posizione ad unità di terra. Oggetti sospetti, anche se nascosti alla vista delle forze di terra, vengono visualizzati sui display integrando alle immagini reali dati aggiuntivi.

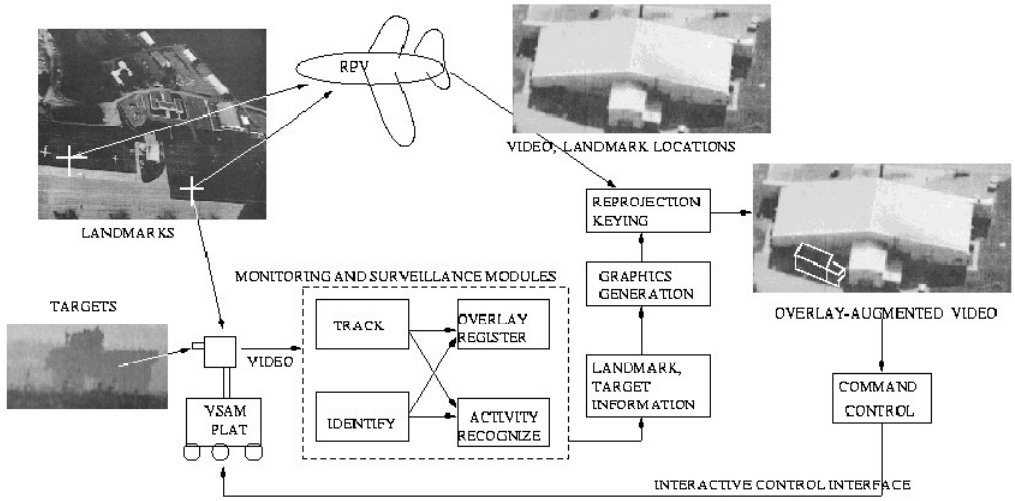


Figura 1.2.11. Il progetto di videosorveglianza e monitoraggio VSAM.

In campo medico la AR è stata adottata per migliorare l'affidabilità degli interventi chirurgici [11]. In questo settore la ricerca di tecniche chirurgiche poco invasive, come la laparoscopia, si scontra con la difficoltà da parte del medico di avere un visione adeguata della zona di intervento. D'altra parte le informazioni provenienti dai sistemi di imaging [16], quali TAC, ultrasuoni ed MRI, che forniscono informazioni dettagliate sull'anatomia e sulla fisiologia del paziente, possono essere proiettate efficacemente grazie ai sistemi di AR.

Le interfacce AR permettono di sovrapporre le immagini virtuali al corpo del paziente fornendo una visualizzazione simile ai raggi X per la zona di intervento. In questo modo il medico può praticare con maggiore precisione procedure complesse quali, la perforazione della scatola cranica per la chirurgia al cervello, biopsie o laparoscopie.

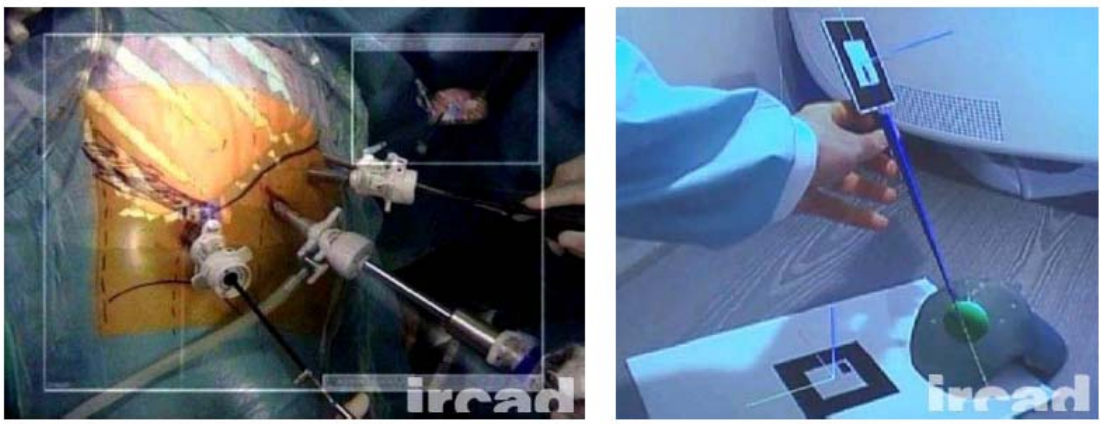


Figura 1.2.12: Esempi di operazione chirurgica in Realtà Aumentata (IRCAD).

Quindi, le applicazioni di AR in campo medico hanno come comune denominatore l'impiego della chirurgia guidata per mezzo di immagini acquisite, attraverso esami clinici, della fisiologia del paziente.

Nella progettazione architettonica le applicazioni di AR sono un valido ausilio nei processi di costruzione, di rinnovo e di ristrutturazione di opere edili [14]. In Figura 1.2.13 si riporta il caso della costruzione di un nuovo ponte. Grazie alle tecniche di AR, la nuova costruzione è proiettata sull'ambiente reale e questo permette di valutarne in maniera realistica l'impatto ambientale.

Figura 1.2.13. La Realtà Aumentata in architettura.

Nella robotica la *Realtà Aumentata* è utilizzata per simulare l'effetto di movimentazione automatizzata delle articolazioni meccaniche dei robot [13]. Croby e Nafis (1994) descrivono un sistema di *Realtà Aumentata* per la manipolazione di un robot che si occupa delle operazioni di ispezione di un reattore nucleare. In un sistema telerobotico, l'operatore utilizza un'immagine visiva dello spazio di lavoro per guidare il robot. Poiché spesso il punto di vista della scena è monoscopico, aumentare la scena, con disegni *wireframe* delle strutture presenti nella vista, facilita la percezione della geometria tridimensionale intorno al robot. Se l'operatore sta per compiere un movimento, questo viene prima simulato su un robot virtuale e visualizzato sul display (v. Figura 1.2.14). L'operatore può quindi decidere di procedere realmente dopo aver visto i risultati. Il robot quindi esegue direttamente solo i movimenti necessari, eliminando pericolose oscillazioni spesso presenti a causa di ritardi di comunicazione con il sito remoto.

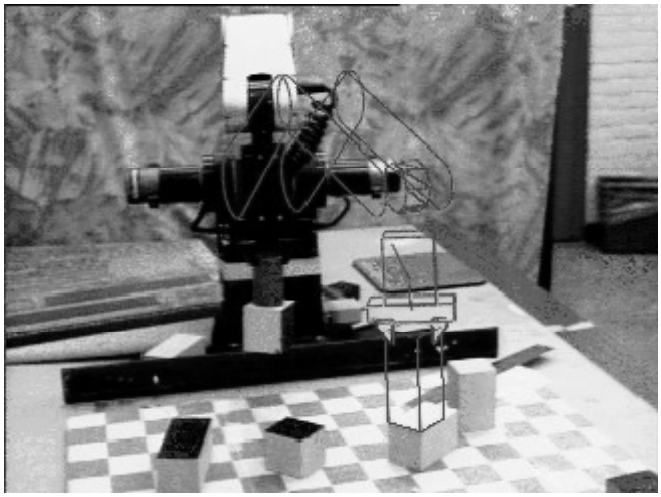


Figura 1.2.14. Applicazione di Realtà Aumentata in robotica.

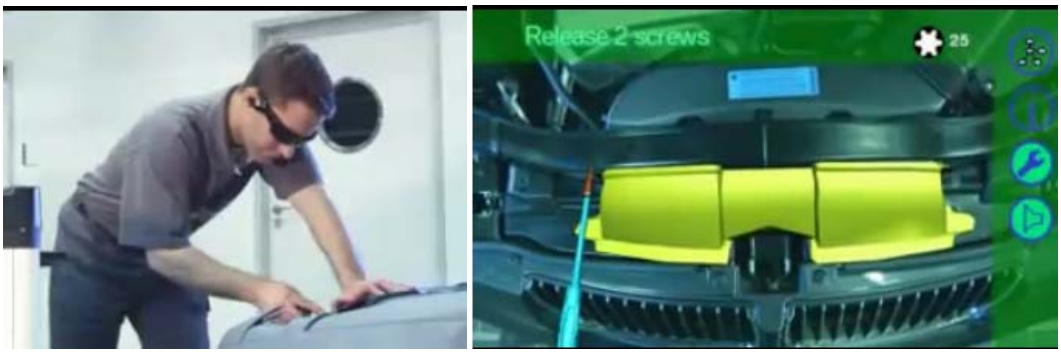


Figura 1.2.15. Applicazione di Realtà Aumentata per le operazioni di manutenzione e riparazione. (dal sito http://technorati.com/videos/youtube.com)

Sin dalla nascita dei primi sistemi (v. Figura 1.1.2), la AR è stata spesso impiegata per assistere nelle operazioni di manutenzione, riparazione ed assemblaggio [21][22][23]. In Figura 1.2.15 si riporta, come esempio, un'applicazione sviluppata di recente dalla BMW per le operazioni di manutenzione sulle macchine. L'operatore indossa occhiali di tipo Optical See Through integrati da microfono e cuffie. L'applicazione visualizza la sequenza e la tipologia delle operazioni da seguire e l'operatore, attraverso il microfono, comunica all'applicazione l'operazione da visualizzare.

In Figura 1.2.16 si riporta l'esempio di applicazione sviluppata per l'insegnamento della geometria. Di fatto l'e-learning è un settore che utilizza e sperimenta applicazioni di AR [24][25][26].

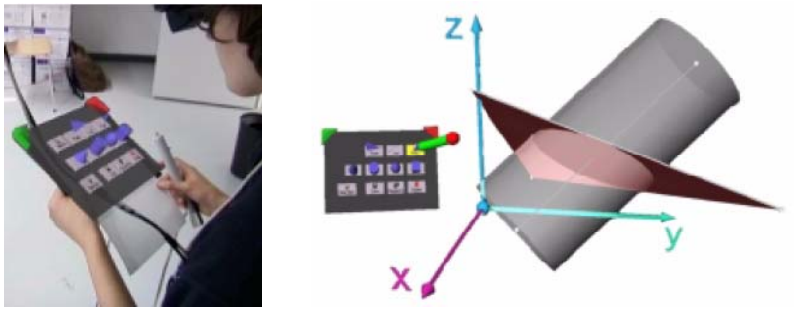


Figura 1.2.16. Applicazione di AR per l'insegnamento della geometria.

Infine si registra l'utilizzo della tecnologia della AR in settori di ricerca e attività come quello manifatturiero [27][28], logistico[29][30], arte [31] e navigazione[32].

1.3 Conclusioni

La Realtà Aumentata è un filone di ricerca emergente della Computer Graphics che trova applicazione in diversi settori di attività e ricerca. Nonostante sfrutti tecnologie proprie della Realtà Virtuale se ne distingue a livello concettuale. Mentre la Realtà Virtuale si riferisce ad un'esperienza immersiva in un ambiente totalmente sintetico, la Realtà Aumentata si riferisce ad un'esperienza mista, dove informazioni e oggetti virtuali sono visualizzati in sovrapposizione a fotogrammi provenienti dal mondo reale.
La sovrapposizione dei due scenari è tanto più efficace quanto più preciso è l'allineamento tra i due mondi e quanto più veloce è la registrazione del punto di vista dell'utente. In Figura 1.3.1 si schematizza il problema dell'allineamento evidenziando le trasformazioni geometriche e i sistemi di riferimento assegnati per convenzione alla camera e alla scena, rispettivamente *S.R. Camera* e *S.R. Mondo*.
L'applicazione deve risalire alla relazione geometrica T che definisce la posizione dell'osservatore rispetto al mondo e alla relazione geometrica P che regola la proiezione sullo spazio immagine degli oggetti virtuali. Tali trasformazioni devono essere applicate alle primitive geometriche dell'oggetto virtuale per allinearlo con la prospettiva dell'osservatore. Il processo finale di rendering riguarda la sovrapposizione delle immagini provenienti dallo spazio reale e dall'ambiente grafico virtuale.

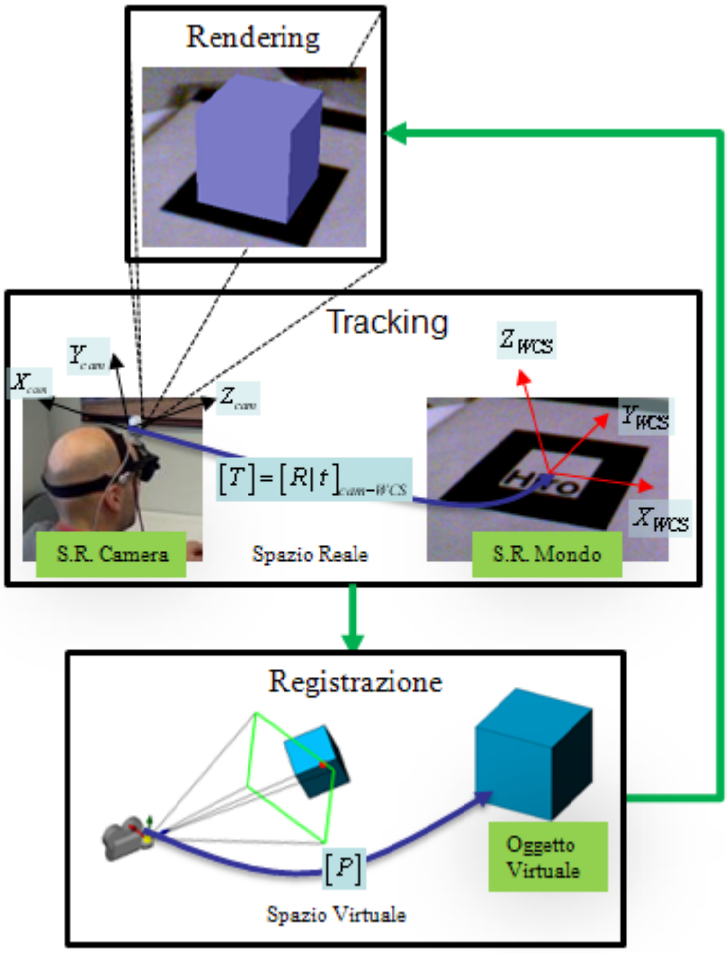


Figura 1.3.1. Sistemi di coordinate e trasformazioni: mondo virtuale e mondo reale.

Diverse sono le tecniche impiegate per la registrazione e l'allineamento, nel Capitolo 2 si discuterà delle tecniche basate sull'elaborazione dell'immagine, le quali calcolano la posizione dell'osservatore rispetto alla scena grazie al riconoscimento, nel singolo fotogramma, di geometrie fiduciali. Il problema della proiezione degli oggetti virtuali riguarderà invece le librerie grafiche OpenGL e le funzioni implementate per le trasformazioni prospettiche.

Oltre ai processi legati alla registrazione e allineamento del punto di vista un altro aspetto caratterizza e distingue tra loro le applicazioni di AR: la scelta dei dispositivi di interazione visiva denominati display o visualizzatori. Questi apparecchi, come accennato nel corso di questa sezione, costituiscono i dispositivi di output sui quali viene proiettata la scena aumentata. Per le applicazioni sviluppate in questo lavoro di tesi si è scelto di utilizzare i monitor (dispositivi spaziali) nella fase di sviluppo e occhiali stereoscopici (HMD) per la fase di prova. Insieme agli occhiali è stata installata una webcam al fine di consentire l'allineamento del punto di vista simulato con quello reale

dell'utente rispetto alla scena. In Figura 1.3.2 si riporta lo schema di funzionamento delle applicazioni che verranno presentate nelle sezioni seguenti. I fotogrammi provenienti dal mondo reale vengono elaborati dal computer e arricchiti delle immagini virtuali. Il risultato è un flusso video proiettato sul visore installato sulla testa dell'operatore.

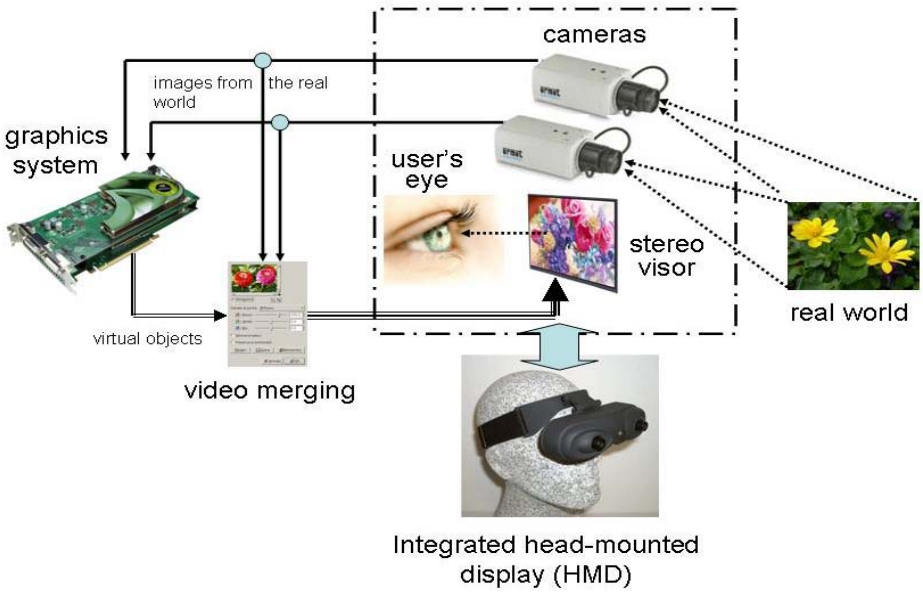


Figura 1.3.2. Schema di funzionamento di un sistema di Realtà Aumentata.

I dettagli relativi alla registrazione e all'allineamento verranno forniti nel Capitolo 2 mentre le applicazioni sviluppate nel lavoro di ricerca saranno presentate nel Capitolo 3.

Capitolo 2 **I sistemi di Realtà Aumentata basati sul riconoscimento di Marker**

Come evidenziato nella sezione precedente, le applicazioni di AR richiedono funzionalità di puntamento e allineamento con l'ambiente tridimensionale[5]. Esistono diversi sistemi e tecnologie per il tracciamento della posizione dell'osservatore (meccanici, magnetici, etc.), in questa sezione si analizzano i sistemi basati sul riconoscimento di geometrie fiduciali planari (*marker*). In particolare si introducono gli strumenti matematici connessi al problema del tracking e si descrivono le librerie grafiche *Artoolkit*, utilizzate per lo sviluppo delle applicazioni di AR presentate nel Capitolo 3.

2.1 La rappresentazione parametrica della Camera.

Alcuni dei principi fondamentali su cui si basano i sistemi AR riguardano l'interpretazione delle informazioni provenienti da sistemi di visualizzazione artificiale quali telecamere e fotocamere. La *Visione Computazionale* (*Computer Vision*) [12] è la

[5] Nel corso della trattazione si fa riferimento al processo di allineamento e puntamento come al problema del *tracking*.

disciplina informatica che studia tali processi. In questa sezione si introducono i parametri ottici che legano, attraverso trasformazioni geometriche, i punti nello spazio tridimensionale alle loro proiezioni geometriche nello spazio dell'immagine.

2.2 Il modello matematico per la proiezione prospettica

La generazione dell'immagine, da un punto di vista matematico, è una proiezione dallo spazio tridimensionale, dei punti acquisiti, allo spazio bidimensionale dell'immagine.

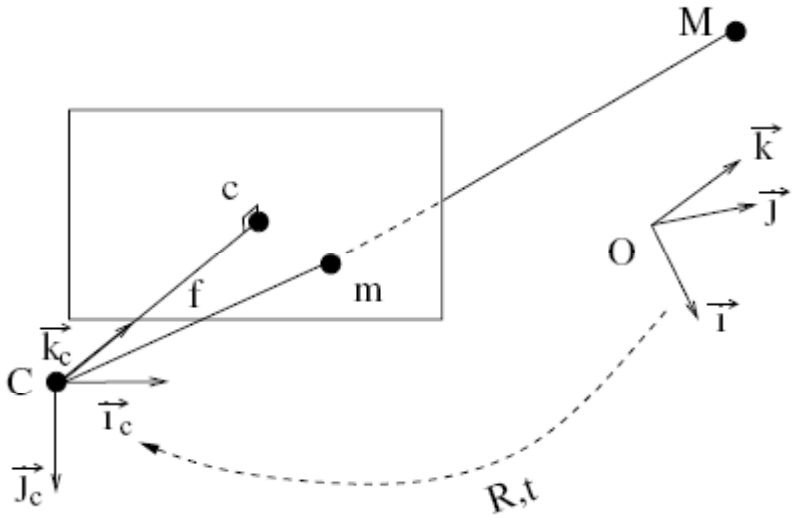


Figura 2.2.1. I sistemi di riferimento del modello di proiezione prospettica: camera *C*, immagine *I*, spazio reale *M*.

Facendo riferimento alla Figura 2.2.1, le coordinate di un punto 3D *M*, con $M = \{X, Y, Z\}^T$ nello spazio euclideo, e il punto corrispondente $m = \{u, v\}^T$ nello spazio immagine sono legati dalla seguente relazione:

$$s\{m\} = [P]_{3\times 4}\{M\} \tag{2.1}$$

dove s è un fattore di scala, mentre $\{m\}$ e $\{M\}$ sono le coordinate omogenee dei punti *m* e *M*, infine *P* è la matrice di proiezione prospettica e dipende da 11 parametri.
La matrice di proiezione *P* è la composizione di due trasformazioni ed è il risultato del seguente prodotto matriciale:

$$[P] = [K] \cdot \left[[R] \middle| \{t\} \right] \tag{2.2}$$

dove:

- $[K]$ è la matrice 3×3 di calibrazione della camera, dipende dai parametri intrinseci della camera e proietta i punti dello spazio della camera *C* nello spazio dell'immagine.
- $\left[[R] \middle| \{\vec{t}\} \right]$ è la matrice 3×4 dei parametri estrinseci della camera e rappresenta la trasformazione euclidea dallo spazio reale a quello della camera (v. Figura 2.2.1). Tale matrice è composta dalla matrice 3×3 di rotazione $[R]$ e dal vettore di traslazione $\{t\}$.

La matrice di calibrazione della camera $[K]$ contiene i parametri intrinseci della camera, detti anche parametri interni, e può essere scritta come:

$$K = \begin{bmatrix} \alpha_u & s & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

dove:

- α_u e α_v rappresentano i fattori di scala nelle direzioni *u* e *v* dello spazio immagine. Tali fattori sono proporzionali alla lunghezza focale *f* della camera: $\alpha_u = k_u \cdot f$ e $\alpha_v = k_v \cdot f$, dove k_u e k_v sono il numero di pixel per unità di lunghezza nelle direzioni *u* e *v*;
- $c = \{u_0 \; v_0\}$ è il punto principale e rappresenta l'intersezione dell'asse *z* della camera con il piano dell'immagine;
- *s* è il parametro di distorsione ed è diverso da zero solo se le direzioni *u* e *v* non sono perpendicolari, ma questa è un'eventualità molto rara nelle moderne attrezzature;

Se la dimensione dei pixel nelle direzioni *u* e *v* è uguale allora $\alpha_u = \alpha_v$, inoltre se il punto *c* è perfettamente centrato nello spazio dell'immagine i parametri $u_0 \; v_0$ sono nulli e la matrice *K* può essere semplificata:

$$K = \begin{bmatrix} \alpha_{uv} & 0 & 0 \\ 0 & \alpha_{uv} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Una camera si definisce calibrata quando sono noti i parametri interni appena elencati. Tornando alla relazione (2.2), la matrice $[R|t]$ definisce l'orientamento e la posizione della camera rispetto al sistema di riferimento *M* dello spazio reale.

Assumendo che i parametri interni siano noti attraverso un processo di calibrazione[6], il problema del tracking si riconduce alla determinazione degli elementi della matrice di posizionamento della camera $[R|t]$. Tale trasformazione consente di esprimere le coordinate di un generico punto P_m ,nel sistema di riferimento dello spazio euclideo, nel sistema di riferimento della camera (P_c):

$$\{P_c\} = [R] \cdot \{P_m\} + \{t\} \tag{2.4}$$

dalla (2.4) è possibile ottenere le coordinate del centro ottico della camera (punto *C* di Figura 2.2.1) imponendo la condizione $\{P_c\} = \{0\ 0\ 0\}^T$, imponendo tale condizione alla (2.4) si ottiene:

$$\{C\} = -[R]^{-1} \cdot \{t\} = -[R]^T \cdot \{t\}$$

In un sistema AR marker based i sistemi di riferimento necessari per risalire alla relazione spaziale che sussiste tra marker e camera, ovvero tra mondo e osservatore sono tre.

[6] Il processo di calibrazione nel caso di un sistema *AR Marker Based* è illustrato nel §2.5.1.

Figura 2.2.2. Il fenomeno della distorsione. Immagine distorta (a sinistra) immagine corretta (destra).

Il modello appena introdotto, per la proiezione dei punti dallo spazio euclideo a quello dell'immagine, risulta incompleto nel caso in cui siano presenti fenomeni di distorsione dell'immagine (v. Figura 2.2.2 a sinistra).

La distorsione può essere considerata nella (2.1) introducendo una trasformazione bidimensionale che corregga l'effetto di bombatura, sull'immagine, dovuto alle imperfezioni ottiche della camera (v. Figura 2.2.2 a destra). Gli effetti distorsivi della camera si possono sintetizzare con due aliquote, la distorsione radiale e la distorsione tangenziale.

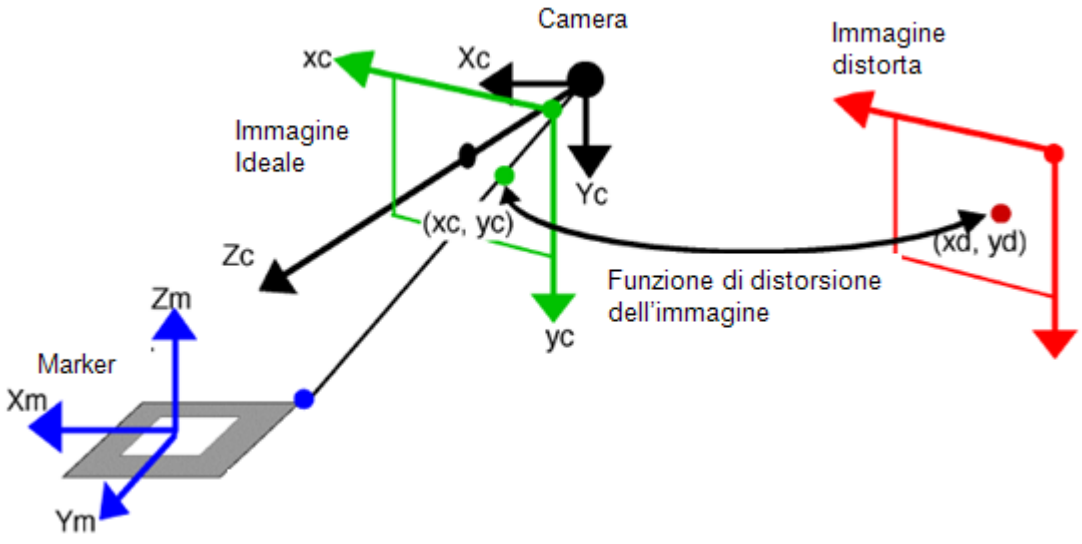


Figura 2.2.3 Correzione della distorsione nel piano dell'immagine.

Definendo $\{\tilde{U}\} = \{\tilde{u}\ \tilde{v}\}^T$ come le coordinate in pixel dell'immagine distorta e con $\{\tilde{X}\} = \{\tilde{x}\ \tilde{y}\}^T$ le corrispondenti coordinate normalizzate tali che:

$$\tilde{u} = u_0 + \alpha_u \cdot \tilde{x}$$
$$\tilde{v} = v_0 + \alpha_v \cdot \tilde{v}$$

dove $u_0, v_0, \alpha_u, \alpha_v$ sono i parametri interni di calibrazione della camera introdotti con la (2.3). Definendo invece con $\{U\} = \{u \ \mathrm{v}\}^T$ e $\{X\} = \{x \ \mathrm{y}\}^T$ i parametri corrispondenti rispettivamente a $\{\tilde{U}\}$ e $\{\tilde{X}\}$, ottenuti correggendo gli effetti della distorsione, e considerando i due contributi di distorsione radiale e tangenziale si ottiene la seguente espressione:

$$\{\tilde{X}\} = \{X\} + \{dX_{\text{radiale}}\} + \{dX_{\text{tangenziale}}\}$$

La distorsione radiale può essere approssimata come:

$$\{dX_{\text{radiale}}\} = \left(1 + k_1 \cdot r^2 + k_2 \cdot r^4 + \ldots\right) \cdot \{X\}$$

dove $r = \|X\| = \sqrt{x^2 + y^2}$. La distorsione tangenziale invece è valutata attraverso la seguente formula:

$$\{dX_{\text{tangenziale}}\} = \begin{bmatrix} 2p_1xy + p_2\left(r^2 + 2x^2\right) \\ p_1\left(r^2 + 2y^2\right) + 2p_2xy \end{bmatrix}$$

tuttavia tale contributo risulta spesso trascurabile e viene quindi ignorato.

La parametrizzazione della posizione della camera rispetto al sistema di riferimento assoluto convenzionalmente assegnato alla scena implica la determinazione dei parametri di rotazione della matrice *R* e di traslazione del vettore *t.* Al contrario della traslazione, la parametrizzazione della rotazione non è univoca, esistono, infatti, diversi metodi per il calcolo dei parametri della matrice *R*.

Le parametrizzazioni più diffuse sono tre: la parametrizzazione secondo gli angoli di Eulero, i quaternioni e le mappe esponenziali [34].

2.3 Calcolo dei parametri di posizionamento

Noti i parametri intrinseci della camera, il problema del tracking si riconduce al calcolo dei parametri estrinseci per ogni fotogramma processato da un'applicazione di AR. In pratica si cerca di stimare con sufficiente precisione i valori dei componenti della matrice $[R|t]$ della (2.2) attraverso la corrispondenza di punti nello spazio con i rispettivi punti individuati nell'immagine.

Da un punto di vista computazione il problema si riconduce alla stima dei componenti della matrice *P* attraverso *n* relazioni del tipo:

$$[P]\cdot\{M_i\} = \{m_i\} \tag{2.5}$$

Se, come in questo caso, i parametri intrinseci della camera sono noti, è possibile dimostrare [35] che se le corrispondenze riguardano punti complanari sono sufficienti 4 punti per risolvere il problema, a patto che non vi siano triplette di punti allineati.

Il problema, così posto, è denominato *DLT*[7] , i primi ad interessarsene furono i fotogrammetristi [36]. Successivamente il DLT è stato studiato dai ricercatori della Computer Vision [37].

Il DLT può essere utilizzato per stimare la matrice *P* della (2.1) anche quando i parametri intrinseci non siano noti. Dalla corrispondenza i-esima tra i punti M_i e m_i si ottengono due equazioni indipendenti:

$$\begin{aligned} \frac{P_{11}\cdot X_i + P_{12}\cdot Y_i + P_{13}\cdot Z_i + P_{14}}{P_{31}\cdot X_i + P_{32}\cdot Y_i + P_{33}\cdot Z_i + P_{34}} &= u_i \\ \frac{P_{21}\cdot X_i + P_{22}\cdot Y_i + P_{23}\cdot Z_i + P_{14}}{P_{31}\cdot X_i + P_{32}\cdot Y_i + P_{33}\cdot Z_i + P_{34}} &= v_i \end{aligned} \tag{2.6}$$

Che possono essere riscritte nella forma:

$$[A]\cdot\{p\} = 0 \tag{2.7}$$

[7] Dall'inglese *Direct Linear Transformation*

Dove p è il vettore costruito con i componenti di P, P_{ij}. Il problema si riconduce al calcolo della soluzione non banale e può essere determinata utilizzando il metodo SVD (*Singular Value Decomposition*) della matrice A con l'autovettore corrispondente all'autovalore più piccolo.

Il calcolo dei parametri di P dipende fortemente dalla disposizione dei punti processati, si passa dalla necessità di avere 15-20 corrispondenze di punti, nei casi più favorevoli, alla necessità di avere 100 corrispondenze per disposizioni particolari.

Una semplificazione efficace dell'iter computazionale consiste nel separare il calcolo dei parametri intrinseci da quelli estrinseci, in modo da applicare l'algoritmo della DLT solo a questi ultimi.

Noti i parametri intrinseci, attraverso la calibrazione, i parametri estrinseci si possono calcolare manipolando la (2.2):

$$[K]^{-1} \cdot [P] \cong \left[[R] \middle| \{t\} \right] \tag{2.8}$$

la sottomatrice R che si ottiene dalla (2.8) può non corrispondere ad una matrice di rotazione, pertanto è necessario procedere con un algoritmo di ottimizzazione iterativo.

Prima di illustrare tale algoritmo è opportuno considerare alcuni aspetti relativi all'applicazione del DLT per il problema del riconoscimento di un marker.

Il DLT consente di calcolare i parametri sia intrinseci che estrinseci del problema del tracking, tuttavia nel caso in cui i punti appartengano ad un piano e siano non allineati a gruppi di tre, il calcolo subisce delle semplificazioni.

In questo caso la relazione di proiezione tra il rettangolo che rappresenta i bordi del marker (v.Figura 2.3.1) e la sua proiezione nello spazio immagine si riconduce al calcolo dei parametri di una matrice 3x3, denominata *matrice omografica*.

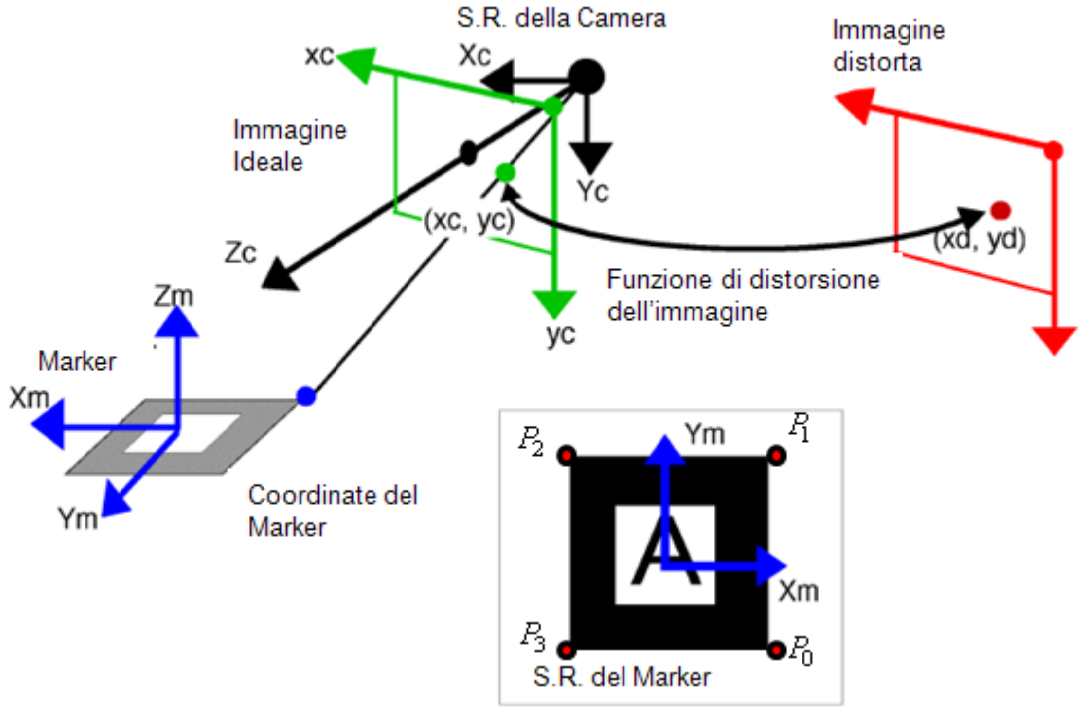


Figura 2.3.1. Calcolo dei parametri di proiezione: posizionamento relativo del marker rispetto alla camera.

Considerando un sistema di riferimento assegnato al marker come quello rappresentato in Figura 2.3.1, ovvero con il piano del marker coincidente al piano di equazione *Z=0*, la matrice omografica *H* si ricava a partire dalla matrice $\left[\left[R\right]\middle|\left\{t\right\}\right]$. Poiché si avranno solo punti del tipo:

$$\left\{M_i\right\} = \begin{Bmatrix} X_i \\ Y_i \\ 0 \\ 1 \end{Bmatrix} \tag{2.9}$$

la proiezione di tali punti M_i nello spazio immagine, secondo la (2.1), sarà:

$$
\begin{aligned}
\{m\} &= [P] \cdot \{M\} \\
&= [K] \begin{bmatrix} | & | & | & | \\ R^1 & R^2 & R^3 & t \\ | & | & | & | \end{bmatrix} \cdot \begin{Bmatrix} X \\ Y \\ 0 \\ 1 \end{Bmatrix} \\
&= [K] \begin{bmatrix} | & | & | \\ R^1 & R^2 & t \\ | & | & | \end{bmatrix} \cdot \begin{Bmatrix} X \\ Y \\ 1 \end{Bmatrix} \\
&= [H] \cdot \begin{Bmatrix} X \\ Y \\ 1 \end{Bmatrix}
\end{aligned}
\tag{2.10}
$$

Dalla (2.10) si evince che una volta note le matrici *H* e *K* è possibile calcolare la posizione relativa della camera rispetto al marker sfruttando la relazione tra matrice *H* e matrice *P*, nella seguente maniera:

$$
[K]^{-1}[H] = \begin{bmatrix} | & | & | \\ R^1 & R^2 & t \\ | & | & | \end{bmatrix} \tag{2.11}
$$

dove R^1 e R^2 rappresentano i vettori X_m e Y_m del riferimento del marker (v. Figura 2.3.1) espresse nel riferimento della camera, il vettore R^3 si ottiene dal prodotto vettoriale di R^1 e R^2 e il vettore *t*, che definisce il vettore traslazione tra il punto di vista della camera e l'origine del marker, è noto dalla (2.11). Infine i componenti della matrice *H* possono essere stimati utilizzando l'algoritmo DLT per quattro punti.

Il DLT così applicato al problema del calcolo della posizione del marker rispetto alla camera è un metodo veloce ed indipendente da soluzioni iniziali, ma lamenta alcuni difetti: è molto sensibile ai disturbi e quindi alla mancanza di precisione.

In merito alla mancanza di precisione è importante osservare che la DLT fornisce la soluzione che minimizza un errore algebrico, mentre sarebbe più corretto considerare la soluzione che minimizza l'errore geometrico. Se il punto m_i viene proiettato e rilevato con un certo rumore, la posizione della camera deve essere stimata attraverso la minimizzazione della somma degli errori di proiezione di tutti i punti, corrispondente al quadrato della distanza tra la proiezione dei punti e i valori rilevati:

$$[R|t] = \arg\min_{[R|t]} \sum_i dist^2\left([P]\{M_i\}, m_i\right) \tag{2.12}$$

Questo tipo di minimizzazione dell'errore non può essere risolta in forma chiusa ma richiede uno schema di ottimizzazione iterativo [12].

2.4 L'allineamento con geometrie fiduciali

Il problema del tracking, introdotto nelle sezioni precedenti, è riconducibile alla sequenza di due operazioni:

- Elaborazione dell'immagine, per estrarre informazioni sulle geometrie fiduciali rilevate;
- Stima della posizione relativa di camera e punti di riferimento nello spazio;

Uno dei fattori che maggiormente influenza l'intero processo è la tipologia e la disposizione rispetto alla camera dei punti di riferimento. A tal riguardo l'utilizzo di geometrie fiduciali facilmente rilevabili nella fase di elaborazione dell'immagine facilita il funzionamento e l'implementazione dell'applicazione. Solitamente si suddividono le geometrie fiduciali in due categorie :

- I punti fiduciali;
- I piani fiduciali;

In genere i punti fiduciali sono delle sferette ricoperte di materiale fosforescente.

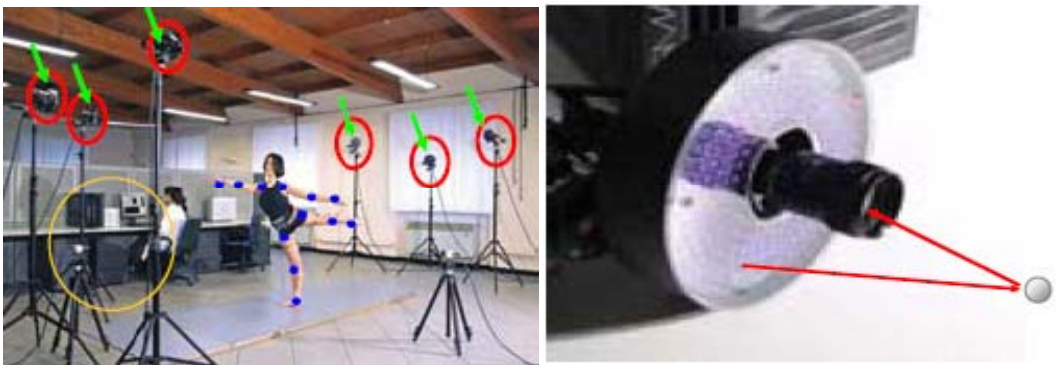


Figura 2.4.1. Marker sferici per analisi del movimento.

La riflettività del rivestimento facilita l'individuazione nelle immagini digitalizzate, l'elaborazione consiste nel calcolo del baricentro delle zone fosforescenti proiettate sui fotogrammi.

I marker planari sono invece forme quadrate o rettangolari di solito messe a contrasto di colore (nero su bianco) con la zona che le circonda [39]. Gli spigoli vengono di solito ricavati dall'intersezione di linee ottenute per interpolazione durante la fase di elaborazione. Da ogni spigolo si ricava la corrispondenza $M_i \rightarrow m_i$ e la posizione nell'immagine viene stimata con il filtro di Kalman.

I riferimenti bibliografici [40][41] e [42] dimostrano come un singolo marker di tipo planare sia sufficiente a stimare la sua posizione rispetto alla camera. Questo tipo di approccio ha avuto successo grazie ai costi contenuti, alla velocità con la quale si riesce ad avere un sistema di tracking tridimensionale e infine grazie all'implementazione di questo metodo in librerie software gratuite, le *Artoolkit* [5]. Una descrizione dettagliata di queste librerie è riportata nella sezione 2.5.

2.5 Le librerie Artoolkit

Le artoolkit sono librerie Open Source, scritte in linguaggio C, implementate per facilitare lo sviluppo di applicazioni di AR in real-time. La prima versione fu sviluppata da Dr. Hirokazu Kato dell'univesità di Osaka (Giappone), e poi sono state supportate dal HIT Lab dell'università di Washington e dal HIT Lab NZ dell'università di Canterbury, Nuova Zelanda. La diffusione di queste librerie è stata agevolata dalla codifica di versioni per sistemi operativi quali Irix, Linux, MacOS e Windows complete di codice sorgente.

Figura 2.5.1. La sovrapposizione di oggetti virtuali a marker planari nelle artoolkit.

In Figura 2.5.1 si riporta un esempio di applicazione sviluppata con le Artoolkit: al sistema di riferimento associato al marker planare viene associato un oggetto virtuale, realizzato in precedenza ed importato nell'applicazione. Lo sfondo del flusso video coincide con quello acquisito dalla telecamera ed è congruente con la percezione visiva dell'osservatore. Gli oggetti virtuali da inserire nella scena sono visualizzati utilizzando l'ambiente grafico delle OpenGL.

Le prerogative delle Artoolkit sono la semplicità e la flessibilità d'impiego. Mentre tecnicamente risolvono il problema del Tracking dell'osservatore utilizzando algoritmi di calcolo propri della *Computer Vision* e calcolando, in real time, la posizione reciproca tra marker e camera elaborando i fotogrammi di un flusso video.

Gli algoritmi di calcolo implementati nelle Artoolkit sono:

- La calibrazione della camera;
- L'elaborazione ed estrazione, dalle immagini digitali, delle geometrie fiduciali;
- La stima della posizione;
- La sovrapposizione di oggetti virtuali ai fotogrammi digitali;

mentre la trattazione matematica di questi aspetti è stata fornita nelle sezioni precedenti in questa sezione si illustrano i processi ad essi associati.

2.5.1 Il processo di calibrazione delle Artoolkit

Il processo di calibrazione (v. § 2.1) stima i parametri intrinseci della camera, i parametri di distorsione e i parametri di proiezione. Le Artoolkit contengono due applicativi per il processo di calibrazione: uno prevede una procedura con una sola operazione mentre l'altro una procedura di calcolo in due fasi. Il primo, denominato *one step*, è consigliato per applicazioni in cui la necessità principale è quella di sovrapporre oggetti virtuali ai marker. Il secondo, denominato *two step*, è più complicato ma fornisce una stima più precisa dei parametri associati alla camera ed è consigliato per applicazioni di misura.

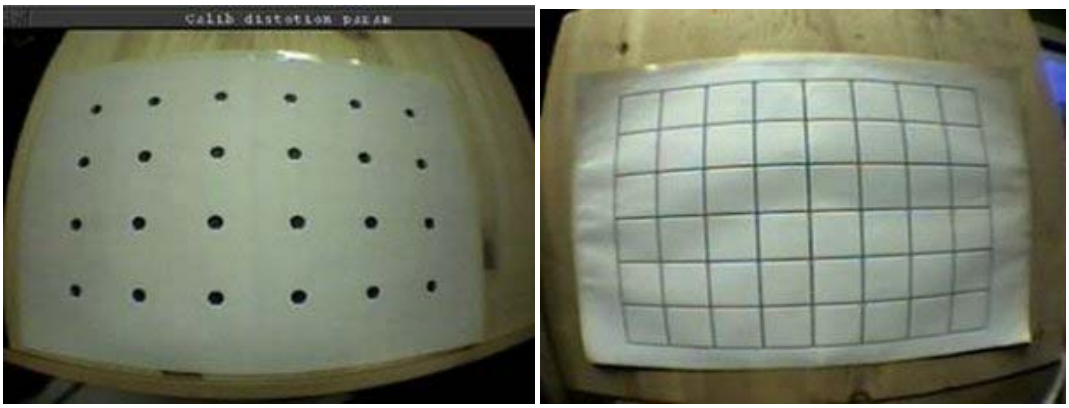


Figura 2.5.2. I supporti per la calibrazione nelle artoolkit: distorsione (a sinistra) proiezione (a destra).

I supporti di calibrazione per la procedura two step sono riportati in Figura 2.5.2, a sinistra è riportata l'immagine per il calcolo dei parametri di distorsione e del centro dell'immagine, mentre a destra quella per la valutazione dei parametri intrinseci della camera.

Il programma *calib_dist.exe* consente di stimare i parametri di distorsione della camera attraverso la selezione dei cerchi neri dell'immagine riportata a sinistra in Figura 2.5.2. Dopo aver ripetuto il processo per un minimo di 5 e un massimo di 10 volte il software

di calibrazione calcola la distanza tra i punti individuati e determina il valore del fattore di distorsione e le coordinate del centro della figura in pixel.

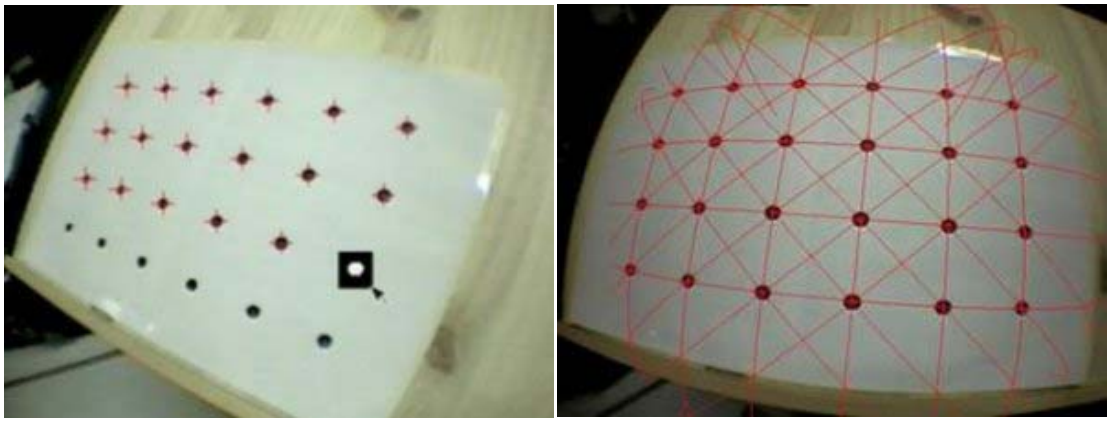


Figura 2.5.3. Il primo passo del processo di calibrazione: selezione dei punti (a sinistra), elaborazione dei parametri a destra.

La fase successiva del processo di calibrazione è affidata al programma *calib_cparam*, che si occupa del calcolo dei parametri intrinseci della camera e richiede, in avvio, i parametri di distorsione ricavati nel processo precedente. In questa fase l'utente deve disporre la griglia di calibrazione di Figura 2.5.2 ortogonalmente alla camera e deve posizionare, utilizzando i tasti freccia della tastiera, le linee blu proiettate virtualmente sulle immagini della camera in corrispondenza delle linee orizzontali e verticali dell'immagine di calibrazione (v. Figura 2.5.4). Tale operazione deve essere ripetuta per differenti valori della distanza camera-foglio di calibrazione, per cinque volte.

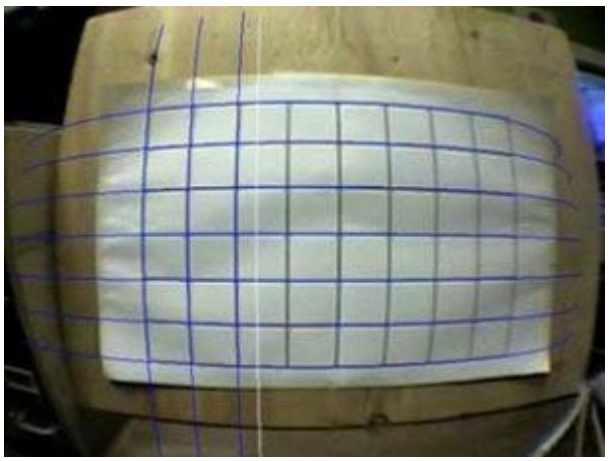


Figura 2.5.4. La seconda fase del processo di calibrazione.

Alla fine del processo il programma salva i dati di calibrazione in un file *.dat, tale file va rinominato come *camera_para.dat* e sostituito al file di calibrazione nelle cartelle delle applicazioni Artoolkit.

2.5.2 Il processo di tracking nelle Artoolkit

Il processo di tracking delle applicazioni di AR consente di rapportare il mondo virtuale a quello reale. Le Artoolkit, grazie alla presenza di un marker nel campo visivo della camera, calcolano la posizione dell'utente rispetto alla scena in questo attraverso la sequenza di operazioni riportata di seguito:

- Caricamento del video-frame contenente il marler;
- Conversione dell'immagine in binario rispetto ad un valore di soglia predefinito per identificare la forma quadrata del marker;
- Calcolo della posizione relativa e dell'orientamento dei marker individuati;
- Confronto delle immagini contenute nei marker con quelle dei marker da individuare caricati all'avvio dell'applicazione;
- Registrazione della scena virtuale con le trasformazioni geometriche dei marker rilevati;
- Sovrapposizione degli oggetti presenti nella scena virtuale al flusso di immagini provenienti dalla camera;
- Proiezione della scena aumentata nel display;

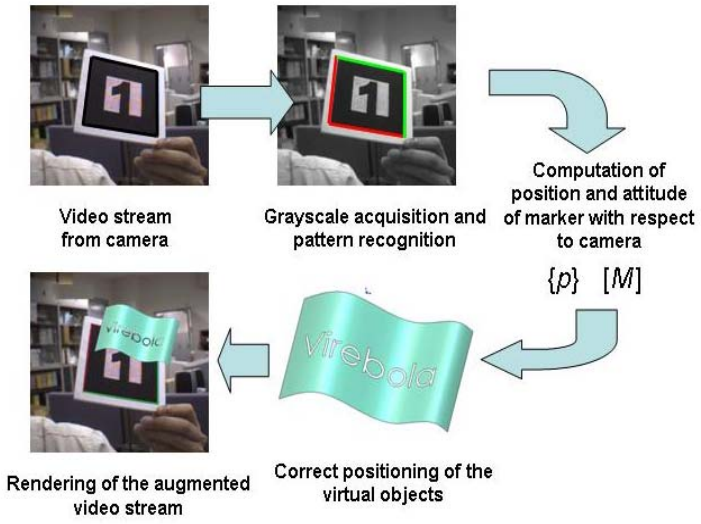


Figura 2.5.5. Schema del processo di tracking nelle Artoolkit.

La Figura 2.5.5 schematizza il processo del tracking mentre in Figura 2.5.6 si riporta l'algoritmo di elaborazione del flusso video tratto da [5] per il medesimo processo.

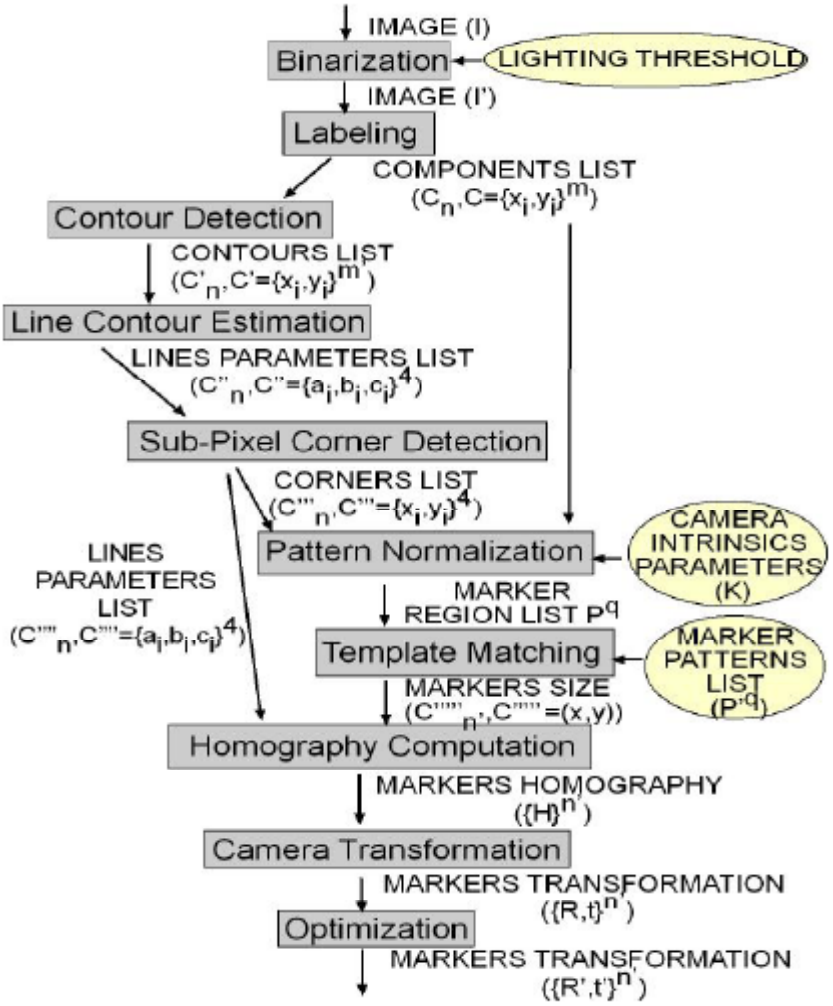


Figura 2.5.6. Algoritmo di elaborazione del flusso video.

In Figura 2.5.7 invece si riportano le immagini delle diverse fasi dell'elaborazione: l'immagine *a* rappresenta il video elaborato in scala di grigi. L'immagine *b* è il risultato dell'operazione di conversione dell'immagine originale in binaria. L'immagine *c* visualizza gli spigoli connessi. L'immagine *d* visualizza i contorni dei marker rilevati. L'immagine *e* mostra il risultato dell'estrazione di bordi e vertici mentre l'immagine *f* la fase di render finale.

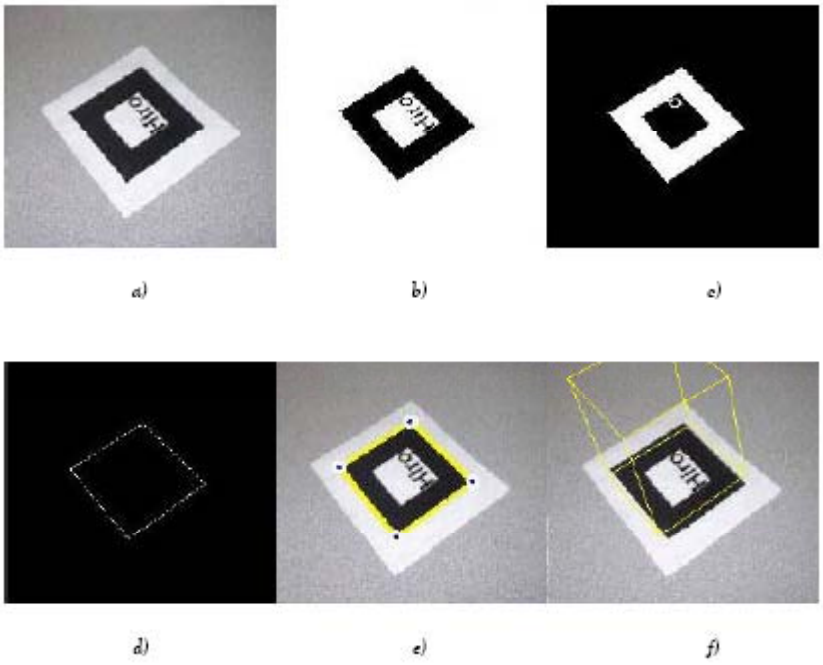


Figura 2.5.7. Le fasi di elaborazione dell'immagine per il tracking.

2.5.3 Il processo di registrazione nelle Artoolkit

Le Artoolkit consentono di sovrapporre all'immagine reale dell'osservatore dei contenuti grafici virtuali visualizzati mediante le interfacce di programmazione delle OpenGL. Ad ogni ciclo di render i parametri intrinseci della camera utilizzata (calcolati attraverso il processo di calibrazione) e le trasformazioni geometriche riguardanti la posizione relativa camera-marker (calcolati attraverso il processo di calibrazione) vengono caricati nella *Rendering Transformation Pipeline* (v. Appendice A) delle OpenGL.

La Figura 1.3.1 riporta lo schema di funzionamento dei processi di tracking e registrazione.

2.5.4 Applicazioni AR con librerie Artoolkit

Dopo aver illustrato le principali procedure e funzionalità di un'applicazione di AR, in questa sezione si analizza da vicino il codice sorgente per un programma di AR sviluppato con le Artoolkit[8].

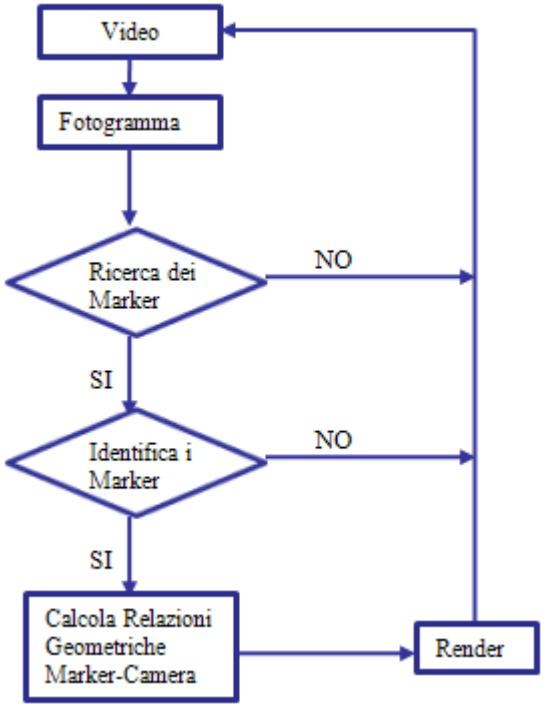


Figura 2.5.8. Algoritmo di programma di AR in ambiente Artoolkit.

Un programma sviluppato con le librerie grafiche Artoolkit è composto da una fase di inizializzazione, da un ciclo principale di elaborazione e dalla fase di chiusura dell'applicazione. Le fasi di inizializzazione e chiusura sono dedicate al caricamento e alla distruzione delle variabili utilizzate per i parametri estrinseci della camera e per i dati dei marker. Il ciclo principale invece elabora con frequenza variabile (fino 30 fps) il flusso video aumentato ed è composto dalle seguenti fasi:

- Fase di caricamento di un fotogramma proveniente dal flusso video;
- Fase di elaborazione dell'immagine per rilevare la presenza di marker;
- Fase di calcolo delle matrici di trasformazione tra riferimento del marker e riferimento della camera;

[8] Le istruzioni di seguito riportate si riferiscono al linguaggio di programmazione Visual C++, mentre le applicazioni sono state implementate sulla piattaforma Visual Studio 2005 della Microsoft.

- Fase di disegno degli oggetti virtuali associati ai marker rilevati;

In Tabella 2.5.1 si riportano le funzioni corrispondenti alle suddette fasi, mentre nella Tabella 2.5.2. Codice C++ per una applicazione di AR con librerie ArtoolkitTabella 2.5.2 si riporta il codice di esempio del programma di AR per il disegno di un cubo virtuale centrato su un marker quadrato (v. Figura 2.5.9).

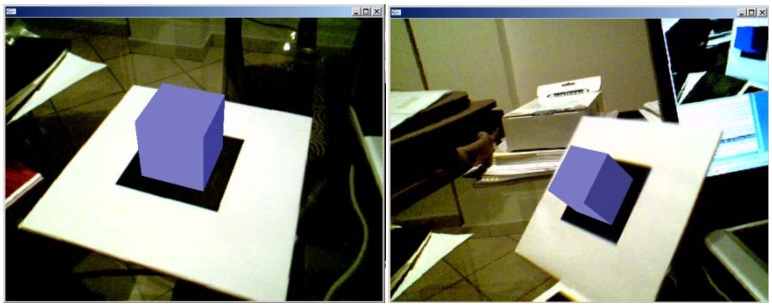


Figura 2.5.9. Applicazione AR per il disegno di un cubo virtuale.

Per capire bene il funzionamento e l'implementazione delle applicazioni che verranno presentate nel Capitolo 3 è importante capire l'utilità e il significato delle funzioni principali riportate in Tabella 2.5.1.

Tabella 2.5.1: Descrizione delle funzione richiamate nel programma Artoolkit

Funzione	Descrizione	Funzione di riferimento
`init()`	Inizializza l'applicazione: carica i parametri della camera, crea la finestra grafica.	`Main`
`arVideoGetImage()`	Carica un video frame.	`mainLoop`
`arDetectMarker ()`	Cerca i marker nel fotogramma caricato.	`mainLoop`
`arGetTransMat ()`	Calcola i parametri di trasformazione della camera.	`mainLoop`
`draw()`	Disegna gli oggetti virtuali	`mainLoop`
`cleanup`	Chiude l'applicazione e libera la memoria del computer dai dati dell'applicazione.	

Tabella 2.5.2. Codice C++ per una applicazione di AR con librerie Artoolkit

```
#ifdef _WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#ifndef __APPLE__
#include <GL/gl.h>
#include <GL/glut.h>
#else
#include <OpenGL/gl.h>
#include <GLUT/glut.h>
#endif
```

```
#include <AR/gsub.h>
#include <AR/video.h>
#include <AR/param.h>
#include <AR/ar.h>

/* set up the video format globals */

#ifdef _WIN32
char   *vconf = "Data\\WDM_camera_flipV.xml";
#else
char   *vconf = "";
#endif

int             xsize, ysize;
int             thresh = 100;
int             count = 0;
int             mode = 1;
char           *cparam_name    = "Data/camera_para.dat";
ARParam         cparam;
char           *patt_name      = "Data/patt.hiro";
int             patt_id;
int             patt_width     = 80.0;
double          patt_center[2] = {0.0, 0.0};
double          patt_trans[3][4];

static void   init(void);
static void   cleanup(void);
static void   keyEvent( unsigned char key, int x, int y);
static void   mainLoop(void);
static void   draw( double trans[3][4] );
```

```
int main(int argc, char **argv)
{
 glutInit(&argc, argv);
    init();
    arVideoCapStart();
    argMainLoop( NULL, keyEvent, mainLoop );
 return (0);
}
```

```
static void   keyEvent( unsigned char key, int x, int y)
{
    /* tasto ESC per uscire dall'applicazione */
    if( key == 0x1b ) {
        printf("*** %f (frame/sec)\n", (double)count/arUtilTimer());
        cleanup(); exit(0);
    }
if( key == 'c' )
{
        printf("*** %f (frame/sec)\n", (double)count/arUtilTimer());
        count = 0;
        mode = 1 - mode;
        if( mode ) printf("Metodo continuo arGetTransMatCont.\n");
         else      printf("Metodo discreto arGetTransMat.\n");
    }
}
```

```
/* Ciclo principale */
static void mainLoop(void)
{
    static int      contF = 0;
    ARUint8         *dataPtr;
    ARMarkerInfo    *marker_info;
    int             marker_num;
```

```
    int             j, k;
    /* Cattura un fotogramma Video */
    if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
        arUtilSleep(2);
        return;
    }
    if( count == 0 ) arUtilTimerReset();
    count++;
    argDrawMode2D();
    argDispImage( dataPtr, 0,0 );
    /* Rileva i markers nel fotogramma elaborato */
    if( arDetectMarker(dataPtr, thresh,
                    &marker_info, &marker_num) < 0 )
{
        cleanup();
        exit(0);
    }
    arVideoCapNext();
    /* Controlla le corrispondeze rettangoli-marker */
    k = -1;
    for( j = 0; j < marker_num; j++ ) {
        if( patt_id == marker_info[j].id ) {
            if( k == -1 ) k = j;
            else if( marker_info[k].cf < marker_info[j].cf ) k = j;
        }
    }
    if( k == -1 ) {
        contF = 0;
        argSwapBuffers();
        return;
    }
    /* Calcola la matrice di trasformazione tra marker e camera */
    if( mode == 0 || contF == 0 ) {
        arGetTransMat(&marker_info[k],
patt_center, patt_width, patt_trans);
    }
    else {
        arGetTransMatCont(&marker_info[k], patt_trans, patt_center,
patt_width, patt_trans);
    }
    contF = 1;
    draw( patt_trans );
    argSwapBuffers();
}
```

```
static void init( void )
{
    ARParam  wparam;
    /* Avvia l'acquisizione video */
    if( arVideoOpen( vconf ) < 0 ) exit(0);
    /*Rileva le dimensioni della finestra grafica*/
    if( arVideoInqSize(&xsize, &ysize) < 0 ) exit(0);
    printf("Image size (x,y) = (%d,%d)\n", xsize, ysize);
    /* Carica i parametri della camera (distorsione, calibrazione)
*/
    if( arParamLoad(cparam_name, 1, &wparam) < 0 ) {
        printf("Camera parameter load error !!\n");
        exit(0);
    }
    arParamChangeSize( &wparam, xsize, ysize, &cparam );
    arInitCparam( &cparam );
    printf("*** Camera Parameter ***\n");
```

```
    arParamDisp( &cparam );
    if( (patt_id=arLoadPatt(patt_name)) < 0 ) {
        printf("pattern load error !!\n");
        exit(0);
    }
    /* Genera la finestra grafica */
    argInit( &cparam, 1.0, 0, 0, 0, 0 );
}
```

```
/* Funzione per la chiusura dell'applicazione */
static void cleanup(void)
{
    arVideoCapStop();
    arVideoClose();
    argCleanup();
}
```

```
static void draw( double trans[3][4] )
{
    double    gl_para[16];
    GLfloat   mat_ambient[]     = {0.0, 0.0, 1.0, 1.0};
    GLfloat   mat_flash[]       = {0.0, 0.0, 1.0, 1.0};
    GLfloat   mat_flash_shiny[] = {50.0};
    GLfloat   light_position[]  = {100.0,-200.0,200.0,0.0};
    GLfloat   ambi[]            = {0.1, 0.1, 0.1, 0.1};
    GLfloat   lightZeroColor[]  = {0.9, 0.9, 0.9, 0.1};

    argDrawMode3D();
    argDraw3dCamera( 0, 0 );
    glClearDepth( 1.0 );
    glClear(GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);

    /* Carica la matrice di trasformazione della camera */
    argConvGlpara(trans, gl_para);
    glMatrixMode(GL_MODELVIEW);
    glLoadMatrixd( gl_para );

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambi);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lightZeroColor);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_flash);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_flash_shiny);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMatrixMode(GL_MODELVIEW);
    glTranslatef( 0.0, 0.0, 25.0 );
 /*Disegna un cubo*/
    glutSolidCube(50.0);
    glDisable( GL_LIGHTING );
    glDisable( GL_DEPTH_TEST );
                                    }
```

La funzione `main` è la prima ad essere eseguita all'avvio del programma e contiene, oltre alle istruzioni di avvio, le impostazioni dell'applicazione.

In ordine la funzione main chiama la funzione `init`, che contiene il codice per inizializzare il caricamento delle immagini video, per la lettura dei parametri del marker

e della camera e per l'impostazione della finestra grafica. Dopo la funzione init, la funzione main chiama la funzione `arVideoCapStart` e lancia il ciclo da seguire in tempo reale. Infine chiama la funzione `argMainLoop`,a cui il main passa, nel caso si vogliano utilizzare, due funzioni di tipo callback[9] e una funzione obbligatoria da eseguire per ogni fotogramma elaborato. Le funzioni callback gestiscono gli eventi associati al mouse e alla tastiera (NULL e `keyEvent` nel codice di esempio riportato in Tabella 2.5.2) mentre la terza funzione contiene le istruzioni per l'elaborazione dell'immagine, il riconoscimento dei marker e il disegno di oggetti virtuali.

Nella funzione `init` i metodi più importanti sono `arParamLoad` con il quale vengono caricati i parametri intrinseci della telecamera (registrati in un file *.DAT) e la funzione `arLoadPatt` che carica i dati relativi al marker da riconoscere nella scena.

Inoltre la funzione `argInit( &cparam, 1.0, 0, 0, 0, 0 )` al pari della funzione `gluCreateWindow` (v. Appendice A), proietta una nuova finestra grafica sul visualizzatore. Il secondo argomento di input di tale funzione definisce un fattore di zoom, nell'esempio presentato è impostato sul valore di 1.0 il che implica una proporzione 1:1 tra le dimensioni del video aumentato e quelle della finestra grafica che lo contiene. Il terzo e il quarto parametro definiscono le dimesioni della finestra, a volte (v. Capitolo 3) è necessario disporre di una finestra maggiorata per visualizzare informazioni aggiuntive e occorre impostare tali parametri su valori diversi da zero.

Il `mainLoop` è il cuore dell'applicazione, contiene le istruzioni per generare un flusso video continuo e aumentato da informazioni sintetiche.

Ad ogni ciclo il `mainLoop` carica un video frame con il metodo `arVideoGetImage()` e lo proietta nella finestra grafica con la funzione `argDispImage(…)`. Senza quest'ultima funzione non si avrebbe l'effetto realistico dell'applicazione, con l'immagine reale proiettata come sfondo della finestra grafica.

Il passo successivo al caricamento riguarda l'elaborazione dell'immagine. Il metodo `arDetectMarker` ricerca eventuali geometrie chiuse nella finestra e le confronta con le immagini di riferimento dei marker. I parametri restituiti dalla funzione `arDetectMarker` corrisponde ad una lista di `marker_num` marker e da un puntatore

[9]Una descrizione più approfondita e alcuni esempi, riguardanti OpenGL e Glut, sono riportati nell'Appendice A.

(marker_info) che punta a tale lista organizzata secondo la struttura dati *ARMarkerInfo* implementata nelle Artoolkit.

Tale struttura contiene per ogni poligono chiuso, possibile proiezione di un marker, le seguenti variabili:

- *area*: variabile intera corrispondente al numero di pixel individuati dalla geometria individuata;
- *id*: variabile intera corrispondente al numero identificativo del marker corrispondente;
- *dir*: variabile intera che registra la rotazione del marker, ovvero definisce quale delle linee registrate corrisponde alla linea di base del marker. Questa variabile può assumere solo quattro valori (0, 1, 2, 3) in funzione dell'ordine di memorizzazione delle linee;
- *cf*: è una variabile di tipo double denominata *confidence value*, e definisce la probabilità, stimata, che la geometria selezionata sia corrispondente all' id-esimo;
- *pos*[2]: è una variabile di tipo double utilizzata per registrare il valore del centro dell'immagine in pixel
- *line*[4][3]: è una matrice di valori double che contiene, su ogni riga, i valori dei coefficienti delle quattro rette di contorno del poligono. Dove le tre colonne corrispondono rispettivamente ai valori *a*, *b*, *c* della retta di equazione $ax+by+c=0$.
- *vertex* [4][2]: è la matrice delle coordinate degli spigoli del marker.

Una volta registrate le possibili geometrie corrispondenti ai marker e dopo aver assegnato ad ognuna un indice di valutazione della corrispondenza, il mainLoop passa a verificare quale delle geometrie rilevate ha la più alta probabilità di corrispondere al marker caricato con il comando `arLoadPatt(patt_name)`. La corrispondenza viene determinata per confronto dei valori di *cf* assegnati ad ogni componente della lista dei marker.

La penultima operazione eseguita dal mainLoop è il calcolo della matrice di trasformazione tra i riferimenti della camera e del marker. La trasformazione in questione viene registrata su una sottomatrice 3x4 (`patt_trans`) della sottomatrice omogenea di trasformazione *P* della (2.2). Tale matrice definisce la posizione relativa del marker rispetto alla camera Figura 2.5.10. Il comando che attiva la procedura di calcolo è `arGetTransMat` .

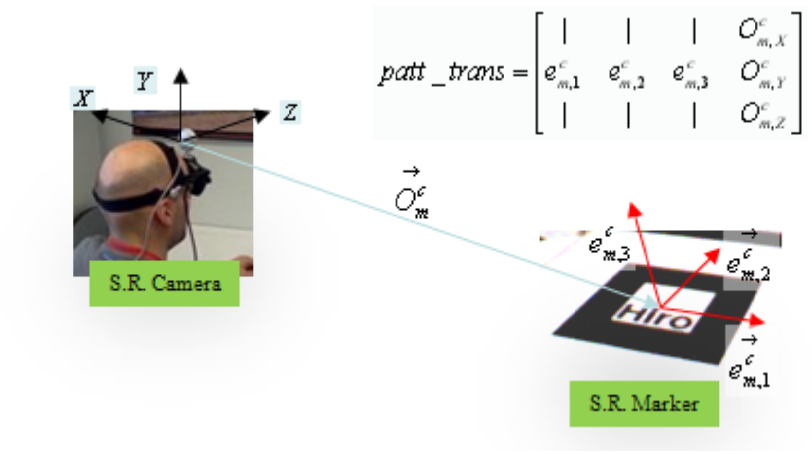


Figura 2.5.10. Trasformazione geometrica tra S.R. della camera e S.R. del marker.

Infine il `mainLoop` richiama la funzione `draw()` che contiene le istruzioni OpenGL e Glut per disegnare gli oggetti virtuali.

La funzione `draw` può essere suddivisa in tre gruppi di istruzioni per l'inizializzazione della scena, per la registrazione del punto di vista e per la fase di render dell'oggetto.

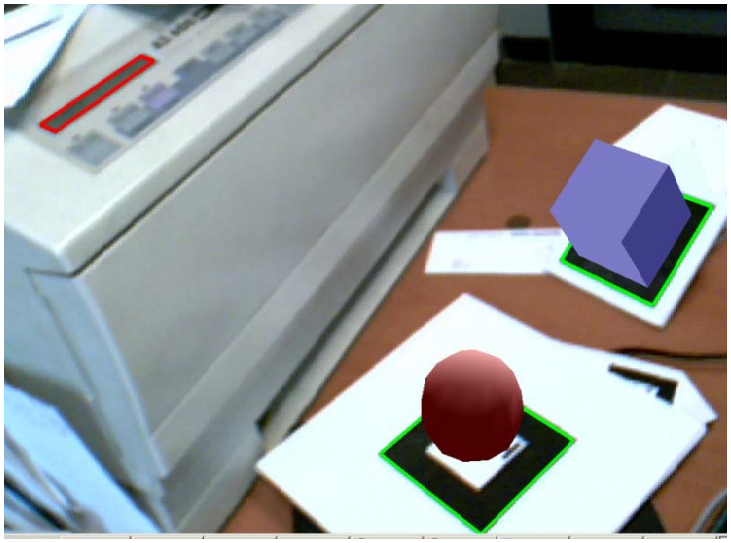


Figura 2.5.11. Esempio di applicazione di AR con più marker.

Per la collimazione del punto di vista con quello della scena reale la funzione `draw` carica la matrice di trasformazione (`patt_trans`) nella matrice OpenGL con i seguenti comandi:

```
\
 argConvGlpara(patt_trans, gl_para);
 glMatrixMode(GL_MODELVIEW);
 glLoadMatrixd( gl_para );
```

dove la funzione `argConvGlpara` converte la matrice di trasformazione nel formato utilizzato nelle OpenGL, mentre i comandi successivi effettuano la registrazione[10].

L'ultima parte del codice è dedicato all'impostazione delle luci nella scena e al disegno del cubo (`glutSolidCube`) visualizzato nelle immagini riportate in Figura 2.5.9.
Le Artoolkit supportano anche il riconoscimento di più marker presenti nella scena, in Figura 2.5.11 è riportato il caso con due marker (evidenziati con il bordo verde) ai quali sono stati associati due entità geometriche elementari (cubo e sfera).
In questo caso il programma di esempio riportato in Tabella 2.5.2 va modificato, la strategia da seguire è quella di creare un file di testo del tipo:

```
4

#marker 1
cube
Data/patt.hiro
80.0

#marker 2
…
```

dove il primo carattere letto è un numero corrispondente al numero di marker da caricare. In seguito si definiscono nome, percorso e dimensioni dei marker da riconoscere. Nel programma di esempio si dovrà prevedere la lettura dei dati per ciascun marker e una struttura dati specifica per i marker al fine di gestirli in maniera più agevole[11].

Come risulterà chiaro leggendo il Capitolo 3, per applicazioni particolari, un aspetto importante nello studio delle Artoolkit riguarda i parametri di trasformazione marker-camera. Già si è parlato, in riferimento all'esempio di Tabella 2.5.2, della matrice `patt_trans`, che contiene la trasformazione geometrica che lega i sistemi di riferimento di marker e camera. Supponendo di puntare a più di un marker nella scena, è possibile risalire alla matrice di trasformazione, quindi alla posizione relativa, di un marker rispetto all'altro.

[11] Per ulteriori dettagli sui comandi di proiezione delle OpenGL si rimanda il lettore all'Appendice A

A tal riguardo considerando quattro matrici 3x4 dove le prime due, `patt_trans1` e `patt_trans2`, coincidono con le matrici di trasformazione camera-marker1 e camera marker2, e la terza matrice `Inv_patt_trans1` coincide con la matrice inversa della `patt_trans1`. La quarta matrice, denominata `patt_trans21`, che definisce la posizione del marker2 rispetto al marker1, si ricava applicando in sequenza i metodi implementati nelle Artoolkit per l'inversione e la moltiplicazione di matrici: `arUtilMatInv` e `arUtilMatMul`. In particolare si avrà:

```
arUtilMatInv(patt_trans1, Inv_patt_trans1);
arUtilMatMul(Inv_patt_trans1, patt_trans2, patt_trans21);
```

Infine le Artoolkit contengono una libreria (arVrml.h) di metodi per il caricamento e la visualizzazione di oggetti salvati in formato *VRML*[12].

[12] VRML (Virtual Reality Modeling Language) è un formato di file progettato per un impiego sul World Wide Web, per rappresentare grafica vettoriale 3D interattiva. L'estensione relativa a tale formato è *.wrl. Un oggetto VRML sfrutta un semplice file testuale che specifica la struttura dati dell'oggetto virtuale da disegnare; un esempio della struttura dati VRML per la definizione di una geometria solida con quattro facce è riportato in Appendice A.

Capitolo 3 **La Realtà Aumentata nella progettazione dei sistemi meccanici**

Mentre nelle sezioni precedenti è stata fornita una breve panoramica dei sistemi di Realtà Aumentata ed è stato analizzato nello specifico un sistema Marker Based, in questa sezione si concentra l'attenzione sulle possibilità di impiegare tale tecnologia nella progettazione dei sistemi meccanici, illustrando applicazioni di ingegneria virtuale, su piattaforme di realtà aumentata, sviluppate dall'autore. Le applicazioni implementate riguardano la modellazione CAD, l'ingegneria inversa, la manutenzione e l'assemblaggio, le simulazioni cinematiche e dinamiche di meccanismi virtuali e il post-processing di analisi strutturali e fluidodinamiche.

3.1 Motivazioni ed obiettivi

L'informatica grafica ha condizionato fortemente, nel corso degli ultimi decenni, l'ingegneria virtuale[13] [18], di fatto sono state implementate e sviluppate diverse

[13] Ci si riferisce comunemente all' *Ingegneria Virtuale* come all'insieme di attività volte a riprodurre al calcolatore tutti i processi, gli strumenti e le risorse coinvolte nell'intero ciclo di vita del prodotto. L'output di queste attività è un prototipo virtuale, ovvero un esemplare primitivo ed immateriale di un manufatto, costituito di dati e relazioni, realizzato e gestito con tecniche informatiche, del tutto equivalente per determinati fini ad un prototipo fisico.

generazioni di programmi di progettazione assistita dal calcolatore, le cui evoluzioni sono dipese fortemente dai progressi informatici contemporanei. In questo modo si è passati dai disegni tecnici bidimensionali, che codificavano solo alcune delle informazioni relative ad un prodotto manifatturiero, e dalle procedure di calcolo automatico, ai moderni software di progettazione, che consentono di inglobare e visualizzare virtualmente tutte le informazioni relative non solo alle proprietà fisiche dell'oggetto (v. Figura 3.1.1) ma anche alla simulazione del suo ciclo di vita.

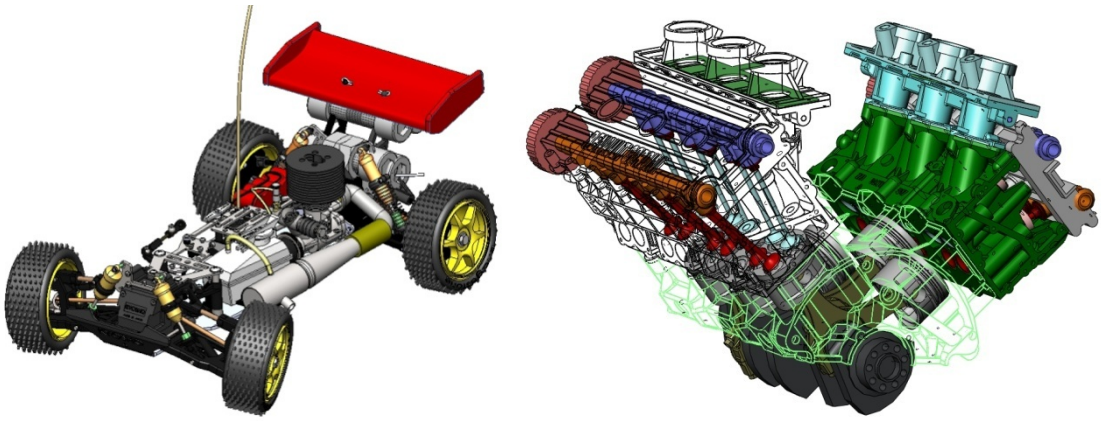


Figura 3.1.1. Esempi di prototipi virtuali.

Per un'azienda manifatturiera l'impiego dell'ingegneria virtuale è una scelta strategica, che ha l'effetto di ridurre i costi connessi alla prototipazione e il tempo di immissione sul mercato di un nuovo prodotto. Sono questi i fattori chiave che spingono verso lo sviluppo delle attività di simulazione e la ricerca di nuovi metodi e tecnologie volti a migliorare e velocizzare tali attività.

Figura 3.1.2. Esempi di Virtual Engineering. A sinistra, interazione visiva con ambiente virtuale. A destra, un esempio di interazione visiva e tattile con l'ambiente virtuale attraverso dispositivi d'interfaccia indossati dall'operatore.

Nella ricerca di possibili miglioramenti, uno degli aspetti da tenere in considerazione è fortemente legato alle possibilità di interagire con l'ambiente virtuale della simulazione.

Una soluzione ideale, da sempre obiettivo degli studiosi informatici, è lo sviluppo di una interfaccia di interazione totalmente immersiva, in cui l'utente riesce a percepire sensazioni del tutto simili a quelle registrabili nel mondo reale (v. Figura 3.1.2).

Nonostante l'esperienza accumulata dalla Realtà Virtuale nello sviluppo di ambienti immersivi [55] (v. Figura 3.1.3), per diversi settori di ricerca e attività, la possibilità di percepire l'ambiente virtuale è ancora fortemente legata alle prestazioni del sistema computerizzato e alle sue capacità di riprodurre stimoli realistici.

Figura 3.1.3. Realtà immersive. A sinistra, per valutare l'ergonomia di un'autovettura. A destra, esempio di cave a 6 muri.

Malgrado gli sforzi profusi nella progettazione di interfacce adeguate i sistemi attuali sono ancora costosi ed ingombranti. Inoltre, i dispositivi più abbordabili peccano nel realismo e i migliori comportano costi eccessivi.

La Realtà Aumentata sembrerebbe fornire gli strumenti giusti per aggirare questi problemi. Infatti, pur essendo tecnologicamente più giovane della Realtà Virtuale e vantando applicazioni industriali per la maggior parte rivolte solo alla visualizzazione di informazioni sintetiche, lo sviluppo di un software in ambiente di realtà aumentata incontrerebbe senz'altro facilitazioni rispetto all'implementazione in un ambiente totalmente virtuale. Il principale beneficio riguarda l'economicità di implementazione dell'ambiente di modellazione. Di fatto l'oggetto può essere visualizzato direttamente nel suo contesto reale e analizzato da diversi punti di vista, senza dover rilevare e riprodurre artificialmente la scena reale.

Tuttavia è bene osservare che proprio a causa della sua gioventù tecnologica la AR, per applicazioni nel settore della simulazione ingegneristica, necessita di affinamenti e miglioramenti mirati ad estendere l'interattività visiva anche al senso del tatto, in quanto il progettista nella fase di creazione o simulazione deve poter interagire con gli oggetti. Quindi è necessario dotare le applicazioni di strumenti di interazione con la scena, ovvero strumenti di puntamento indispensabili per il controllo e la manipolazione degli oggetti virtuali e reali.

Per gli scopi appena accennati i dispositivi di puntamento devono possedere i seguenti requisiti:

- Il dispositivo deve essere user friendly;
- Il dispositivo deve possedere una precisione accettabile, per il contesto di utilizzo e deve consentire un'acquisizione ad alta frequenza per facilitare l'elaborazione in tempo reale delle azioni dell'utente;
- Non deve limitare la mobilità dell'utente;
- Deve essere uno strumento di supporto per agevolare il progettista nella fase di progettazione;

Nelle applicazioni presentate in questa sezione vengono presi in considerazione due tipi di puntatori, un puntatore elettromagnetico e un puntatore marker che sfrutta, per rilevare la posizione nello spazio, le matrici di trasformazione marker-camera delle artoolkit.

L'obiettivo, che ci si è prefissi di seguire, è quello di sviluppare applicazioni ingegneristiche in ambiente di Realtà Aumentata e valutare se queste possano diventare strumenti di supporto nella progettazione dei sistemi meccanici.

3.2 Implementazione di un sistema CAD in Realtà Aumentata

Il primo passo nella creazione di un prodotto e del suo ciclo di vita è la modellazione delle sue forme. I software di modellazione CAD sono diventati, nel corso degli ultimi venti anni, uno strumento indispensabile per il progettista industriale poiché forniscono il supporto di base del ciclo di vita simulato dei prodotti manifatturieri.
La crescente richiesta di software interattivi in grado non solo di gestire il prototipo virtuale di un prodotto ma di valutarne anche l'ergonomia e l'estetica, già nella fase di concettualizzazione, spinge verso la richiesta di applicazioni sempre più immersive.
Nonostante i CAD abbiano subito evoluzioni software importanti e proporzionali all'evoluzione dei componenti hardware, l'interfaccia dei programmi CAD è rimasta la stessa fin dalla sua nascita.

Figura 3.2.1. La configurazione *Desktop* di Engelbart.

Nel 1968 Engelbart (v. Figura 3.2.1) realizzò la prima interfaccia *desktop* (tastiera, mouse e monitor) e da allora tale sistema non ha subito evoluzioni ma solo integrazioni che non ne hanno mai migliorato l'interattività (v. space-mouse).
D'altra parte se la proiezione bidimensionale, largamente utilizzata per la comunicazione web e per le applicazioni di ufficio, quali videoscrittura, è una soluzione soddisfacente, nell'ambito del disegno meccanico costituisce un limite. Il disegno industriale necessita

di una maggiore comprensione e una più completa percezione di forme complesse quali quelle delle carrozzerie automobilistiche, rispetto a quelle che un display bidimensionale riesce ad offrire. Il progettista ha la necessità di muoversi liberamente nello spazio di lavoro e deve poter creare, modificare e animare gli oggetti virtuali, espressioni della sua creatività [38]-[47]. A causa di questa necessità alcuni strumenti di prototipazione quali il bozzetto, la modellazione scultorea e il *Taping*[14] (v. Figura 3.2.2) non sono ancora stati abbandonati completamente [188].

Figura 3.2.2. Tecniche di prototipazione nel settore Automotive. A sinistra, il *Taping*. A Destra, la modellazione con sculture d'argilla.

Tuttavia queste tecniche risultano sempre più limitative nelle realtà industriali moderne, per la mancanza di codificazioni informatiche simultanee in grado di supportare la concurrent engineering. Infatti, la strategia di avere in tempo reale le bozze dei modelli virtuali consente a più gruppi di lavoro di studiare la bontà funzionale del progetto e di scartare più velocemente soluzioni impraticabili.
Inoltre, per ottenere feedback su come l'oggetto si inserisce nel suo contesto di lavoro, è spesso necessario prevedere lunghe fasi di rendering e prototipazione.

3.2.1 Tecniche di modellazione nei moderni software CAD

Le versioni più recenti dei software CAD, grazie ai progressi compiuti in campo informatico, vantano potenzialità di modellazione parametrica e visualizzazione notevoli (v. Figura 3.2.3).

[14] Il Taping è una tecnica di prototipazione molto utilizzata in campo automobilistico e motociclistico. Il designer ha a disposizione una parete sulla quale stende con le mani un nastro adesivo nero e realizza il modello in scala reale. Modifiche e aggiunte sono applicate velocemente rimuovendo o aggiungendo nastro.

Figura 3.2.3. Esempio di modello virtuale realizzato in SolidWorks. Dal sito http://www.solidworks.com

Rispetto alle precedenti versioni o applicazioni di disegno assistito dal calcolatore, che utilizzavano rappresentazioni B-Rep e C.S.G., l'associatività e la parametricità dei programmi attuali consentono di gestire in maniera veloce le modifiche da apportare a forme e dimensioni di componenti meccanici.

La logica di funzionamento dei CAD parametrici si basa sull'utilizzo di due elementi principali:

- Lo schizzo;
- Le funzioni applicate allo schizzo;

Mentre il primo rappresenta il profilo di base dell'oggetto, le funzioni applicate (sweep, loft, estrusione etc.) generano gli oggetti solidi (v. Figura 3.2.4) o le lavorazioni di taglio applicate agli oggetti stessi.

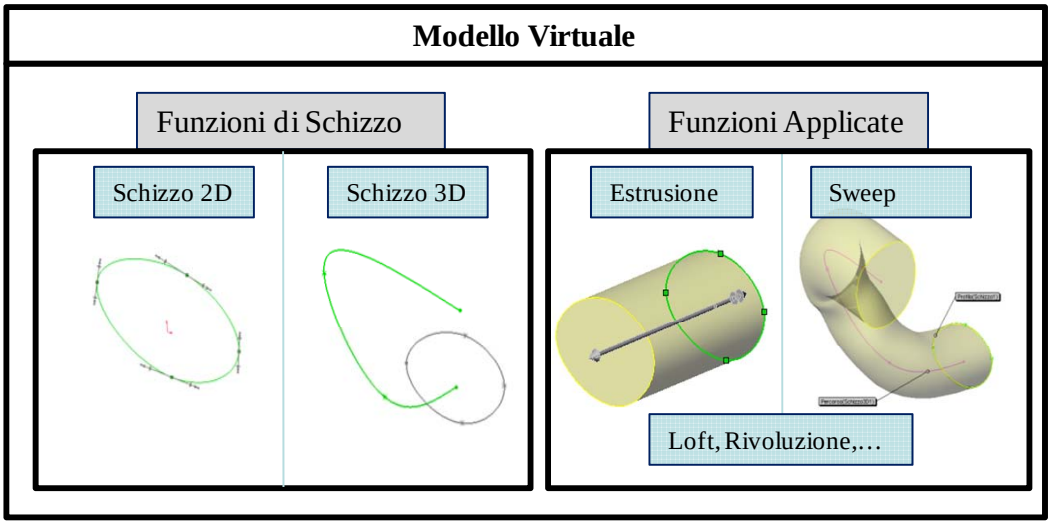


Figura 3.2.4. Logica di funzionamento dei CAD parametrici associativi: le funzioni di schizzo per crfeare i profili di base, le funzioni applicate per generare le geometrie solide e le lavorazioni dell'oggetto.

Due sono i fattori che condizionano il progettista durante la modellazione concettuale[15]:

- Valutare la forma dell'oggetto nello spazio tridimensionale durante il disegno;
- Puntare ad elementi non appartenenti ad un piano;

Il primo fattore deriva dalla rappresentazione intrinseca dei software sullo schermo, che vincola alla visualizzazione degli oggetti secondo la loro proiezione rispetto al piano ortogonale alla vista. Le funzioni di visualizzazione consento di muovere dinamicamente il punto di vista rispetto al sistema di riferimento virtuale associato all'oggetto. Anche se queste funzioni risultano utili nella fase di valutazione finale dell'oggetto creato è auspicabile un'interattività visiva migliore che consenta di percepirne in maniera più naturale la forma (v.Figura 3.2.2) durante la fase di modellazione o modifica.

A tal riguardo una tecnica efficace è quella che prevede l'utilizzo della visualizzazione simultanea di quattro viste dell'oggetto (v. Figura 3.2.5).

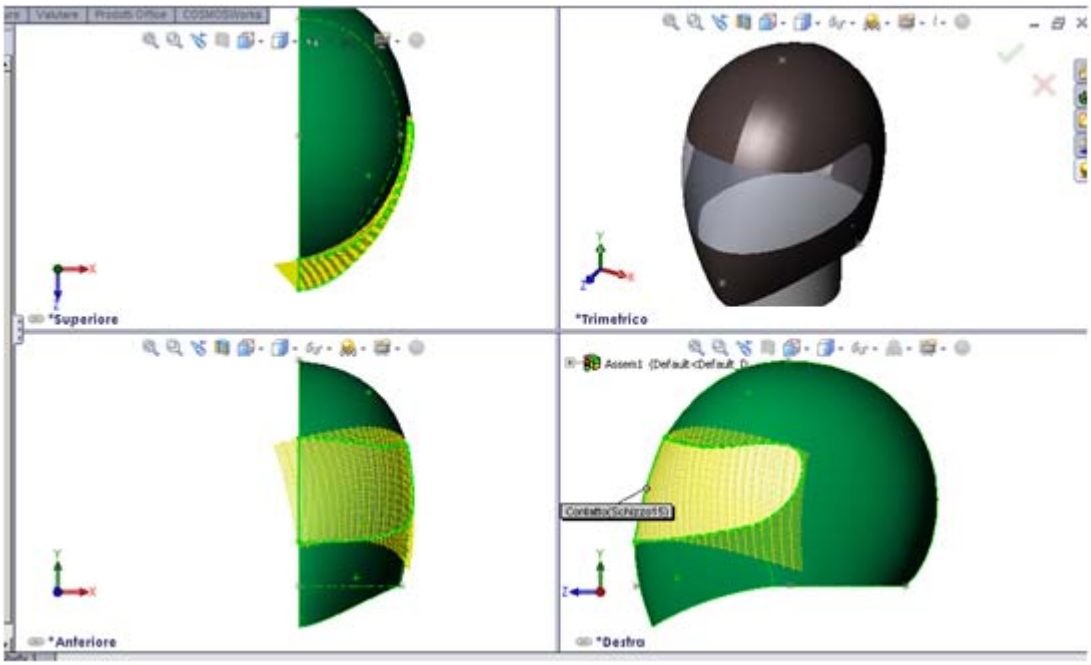


Figura 3.2.5. Modellazione con viste multiple collegate al modello.

In questo modo si percepiscono meglio gli effetti, sulla forma finale, causati dal disegno o dalla modifica delle geometrie di costruzione del modello.

Il secondo fattore che vincola il disegnatore, nella modellazione concettuale riguarda le possibilità offerte dai software di puntare ad entità geometriche nello spazio di lavoro.

[15] Affermazione dell'autore supportata dall'esperienza maturata come esercitatore del corso di Disegno Assistito dal calcolatore presso l'università di Roma Tor Vergata negli anni accademici 2006/2007 e 2007/2008.

Tale fattore è influenzato dalla dinamica di funzionamento dei programmi. La maggior parte delle funzioni per la creazione degli schizzi sono associati ai piani di appartenenza degli stessi, l'operazione di selezione di punti nello spazio è pertanto vincolata a tali piani e difficilmente con la vista di proiezione bidimensionale si riesce a selezionare il punto cercato. I limiti appena accennati vengono parzialmente risolti utilizzando le funzioni implementate per gli schizzi di entità tridimensionali (v. Figura 3.2.4). In questo caso il progettista può selezionare punti nello spazio e tracciare curve parametriche tridimensionali per creare profili di contorno o di controllo per superfici di forma complessa (v. Figura 3.2.6). Tuttavia anche in questo caso il vincolo ad utilizzare una vista bidimensionale limita la possibilità di puntare consapevolmente ai punti nello spazio. A tal proposito in Figura 3.2.6 si riporta il caso della modellazione della visiera di un casco: si noti come la curva parametrica di contorno della visiera, indicato dalla freccia rossa, per le viste Laterale e Superiore risulti correttamente tracciato nella vista laterale mentre presenti una anomalia sulla curvatura evidenziata nella vista superiore. Tale anomalia genera un effetto indesiderato sulla forma finita, visibile nella Vista 3D riportata in alto a destra in Figura 3.2.6.

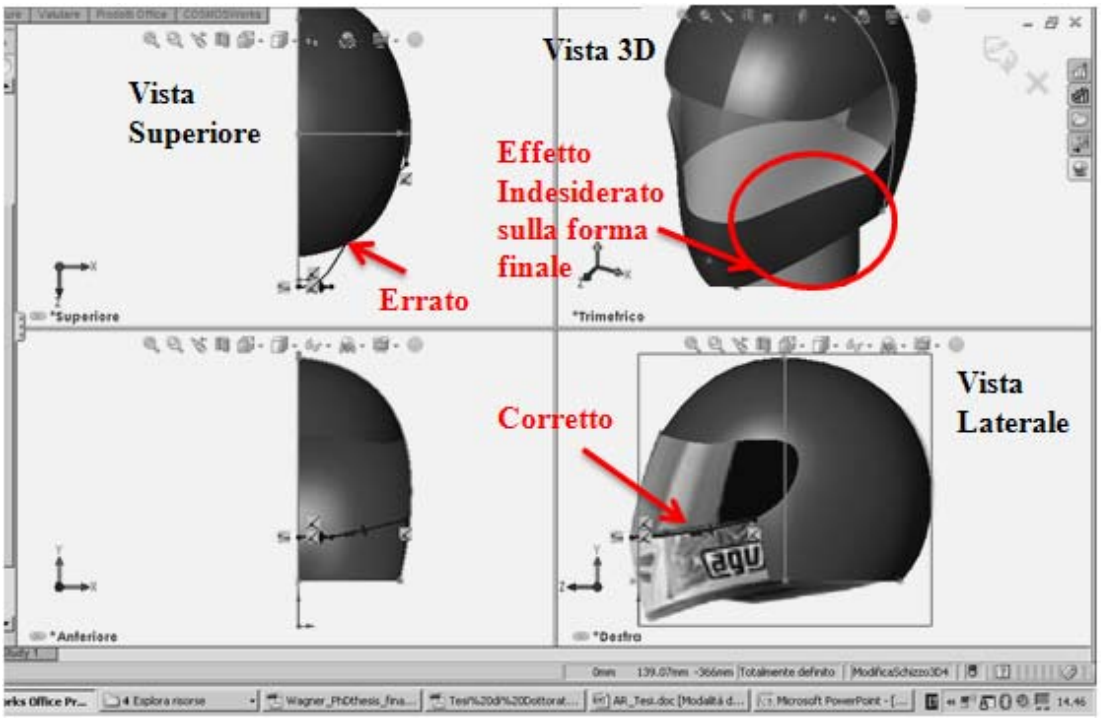


Figura 3.2.6. Errori di costruzione per la modellazione CAD di profili 3D.

Pertanto, sebbene l'utilizzo delle viste multiple collegate al modello sia uno strumento efficace che permette di valutare efficacemente eventuali effetti indesiderati sulle forme

generate, è senza dubbio un sistema macchinoso e poco intuitivo da utilizzare nella modellazione, poiché il progettista deve controllare visivamente più viste e deve disporre le entità geometriche utilizzate per la modellazione passando da una vista all'altra in un processo iterativo che si conclude quando l'entità geometrica assume, su tutte le viste, una forma congruente con quella desiderata.

Risulta pertanto evidente che nei sistemi CAD che utilizzano un'interfaccia tradizionale WIMP, cioè monitor, tastiera e mouse, la creazione e la modifica di modelli 3D è possibile solo mediante scomposizione in operazioni bidimensionali su sezioni o piani ausiliari, che, oltre ad essere non intuitive, ostacolano il flusso creativo e la rapida espressione e valutazione delle idee.

La ricerca di un ambiente di modellazione tridimensionale, in cui muovere il punto di vista e controllare la posizione dei punti nello spazio è la soluzione ideale per agevolare e semplificare il processo di disegno [188] (Figura 3.2.7).

Figura 3.2.7. Modellazione 3D in Realtà Aumentata sviluppata da Wayne Piekarsky.

In tale ambito l'interattività è uno dei componenti più importanti dell'applicazione, per questo si è scelto di utilizzare un dispositivo elettromagnetico e un HMD che permette all'utente di muoversi agevolmente nello spazio di lavoro avendo a disposizione sempre viste prospettiche allineate con il suo punto di vista e potendo selezionare in maniera intuitiva gli oggetti nello spazio senza il limite della profondità imposto dalle attuali configurazioni desktop.

3.2.2 Sviluppo di un sistema CAD in AR

Questa sezione è dedicata alla presentazione del software *AR-CAD 2.0*, un programma CAD in ambiente di AR. La descrizione del sistema parte dalle problematiche incontrate

dall'autore durante la fase di sviluppo e arriva alla presentazione dell'interfaccia del software.

Le problematiche affrontate nello sviluppo del sistema sono:

- La scelta di componenti hardware compatibili con le esigenze dell'applicazione e con le disponibilità economiche;
- La scelta di una piattaforma software per la gestione dei dispositivi e dell'applicazione;
- L'implementazione di strutture dati per ogni tipologia di entità geometrica da modellare e visualizzare;
- L'implementazione di metodi di rappresentazione per ogni tipo di struttura dati utilizzata;
- La scelta di componenti e di interfacce per garantire l'auspicata interattività;

Ciascuno degli aspetti appena menzionati viene discusso nelle sezioni seguenti.

Figura 3.2.8. Il sistema in funzione.

3.2.3 Layout del sistema

Il sistema implementato è costituito di dispositivi di input, di elaborazione e di output. I dispositivi di input acquisiscono il flusso video della scena reale e gestiscono l'interazione uomo-macchina. I dispositivi di output sono invece dedicati alla proiezione del flusso di immagini reali aumentate con gli oggetti virtuali. L'unità di elaborazione è dedicata al trattamento dei flussi di dati di input e alla loro corretta combinazione per produrre il flusso video aumentato (v. Figura 3.2.9).

I dispositivi di Input sono tre:

- la tastiera, per ricevere l'evento di attivazione dei comandi di modellazione e modifica;
- la webcam, utilizzata per catturare un flusso continuo di immagini dell'ambiente reale allineate con il punto di vista dell'operatore;
- il puntatore elettromagnetico, per selezionare i punti nello spazio o le entità virtuali appartenenti alla scena aumentata;

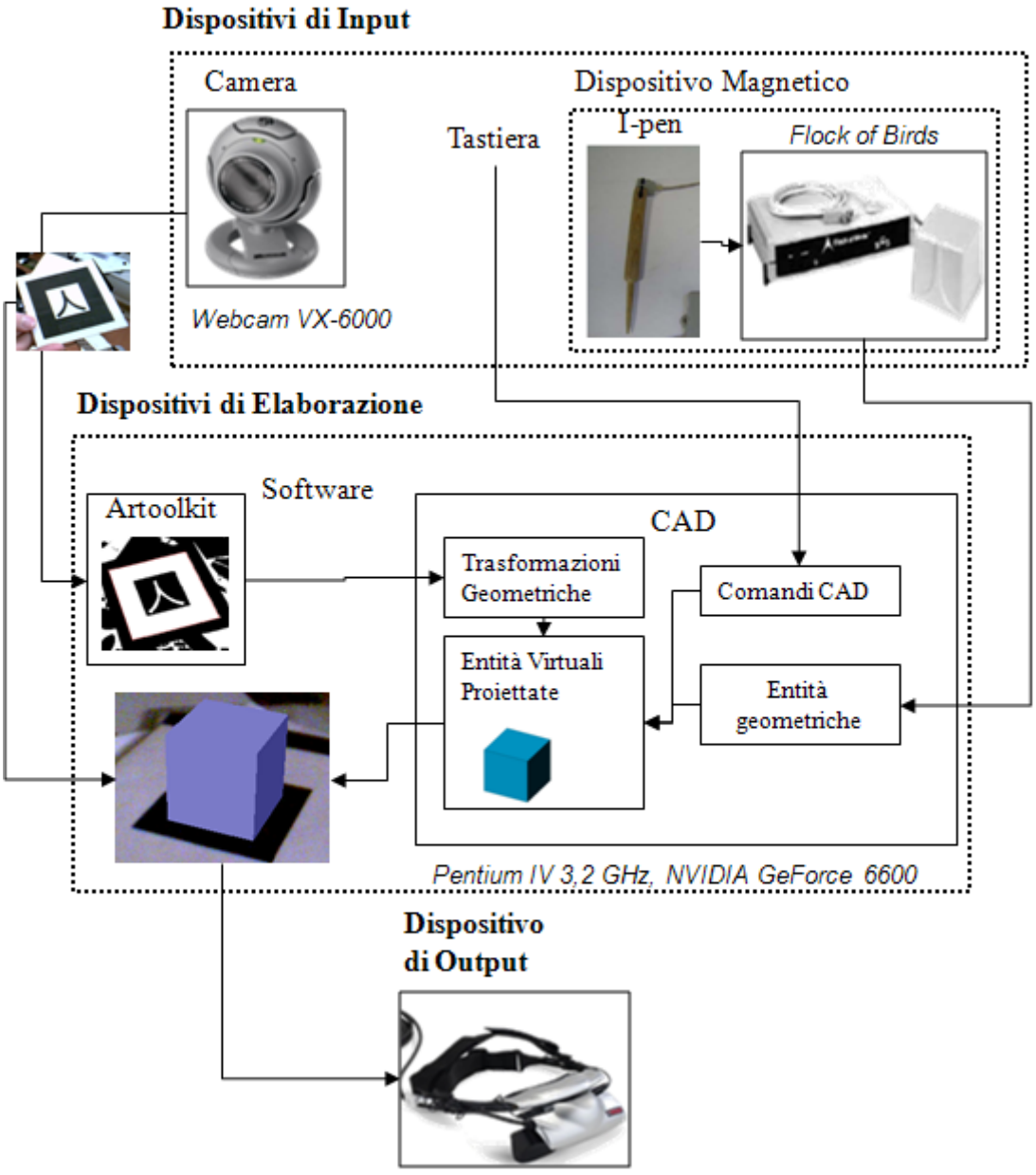


Figura 3.2.9. Layout del sistema: dispositivi di input, output ed elaborazione.

Mentre la tastiera (*Logitech*) e la webcam (*Microsoft VX-6000*), di cui vengono riportate le caratteristiche tecniche in Tabella 3.2.1, sono accessori informatici di uso comune, il dispositivo di puntamento è di tipo magnetico. Il modello scelto per l'applicazione è il

Flock of Birds realizzato dalla *Ascension Technology* le cui caratteristiche tecniche sono riportate in Tabella 3.2.2 (i dati sono stati ricavati da [60]).

Tabella 3.2.1. Caratteristiche tecniche della Webcam Microsoft VX-6000.

Dimensioni (LxPxH)	5.5 cm x 5.3 cm x 7.4 cm
Peso	95 g
Tipo sensore ottico	CMOS - 1.300.000 pixel (1280 x 1024)
Messa a fuoco	Manuale
Caratteristiche	Tecnologia tracking automatico immagine
Capacità trasmissione immagini	30 fps

Tabella 3.2.2. Specifiche tecniche del *Flock of Birds.*

Position Range	Fino a 3.05 m. in tutte le direzioni
Angular range:	±180° Azimuth ±180° Roll ±90° Elevation
Static Accuracy Position	1.8mm RMS
Static Accuracy Orientation*:	0.5° RMS
Static Resolution Position:	0.5mm @ 30.5cm
Static Resolution Orientation:	0.1° @ 30.5cm
Measurement rate:	Up to 144 measurements/second
Outputs:	X,Y,Z positional coordinates and orientation angles, rotation matrix, or quaternions
Interface:	RS-232C with selectable baud rates to 115,200, USB converter is available.
Format:	Binary
Modes:	Point or strema
PHYSICAL	
Transmitter:	Mid-Range Transmitter: 9.6cm cube with 3.05M cable, or extended range transmitter option: 30.5cm cube with 6.1m cable
Sensor:	25.4mm x 25.4mm x 20.3mm cube (or optional 3-button mouse) with 3.05M or 10.7M cable. Weight: 21 g (0.7 oz) without cable, 169 g

	(6.0 oz) with 3.05M cable, 394 g (13.9 oz) with 10.7M cable.
Enclosure:	24cm x 29cm x 6.6cm
Power:	User provided or optional external plug-in: US/European version
Environment:	Metal objects and stray magnetic fields in the operation volume will degrade performance.

Il dispositivo di elaborazione è un PC configurazione desktop, con scheda madre Asus P5, processore *Pentium IV* a 2,8 GHz, 1 Gb di RAM Kingston a 667 MHz e scheda grafica modello *NVIDIA GeForce 6600*.

Il dispositivo di Output è un display di tipo video see-trough della *eMagin* le cui specifiche tecniche principali sono riportate in Tabella 3.2.3.

Tabella 3.2.3. Specifiche tecniche visore *eMagin Z800*.

Rapporto Larghezza Altezza	4:3
Risoluzione	SVGA 800x600 (1.44 Mp)
Contrasto	200:1
Luminosità	50 cd/m^2
Tracking Orizzontale	360°
Tracking Verticale	60°

Nello scegliere i componenti da utilizzare per il sistema si è cercato il miglior compromesso tra la capacità di soddisfare le esigenze progettuali e quella di rientrare, in termini di costi, nel budget disponibile. La spesa maggiore ha riguardato l'acquisto del display (dispositivo di output) e il dispositivo magnetico per il puntamento (circa 3500 € complessivamente). Per i componenti di elaborazione dati (pc e scheda grafica) sono stati utilizzati componenti riciclati, mentre il dispositivo di acquisizione video è una webcam commerciale, acquistabile con una spesa inferiore ai 100€.

3.2.4 Collimazione del flusso dati

La fase più delicata nell'assemblaggio dei vari componenti, per l'implementazione del sistema, ha riguardato lo sviluppo del programma di gestione. Non avendo a disposizione un unico dispositivo che si occupasse della manipolazione del flusso di dati si è scelto un linguaggio ed una piattaforma di sviluppo comuni. Poiché le librerie di

gestione del puntatore magnetico e quelle di implementazione delle applicazioni di realtà aumentata sono disponibili in linguaggio C++, l'applicazione (*AR-CAD 2.0*) è stata sviluppata utilizzando tale linguaggio su piattaforma Microsoft Visual Studio.NET 2005.

È appena il caso di sottolineare che il Software è stato sviluppato sulla base del codice presentato nel Capitolo 2. Le aggiunte principali hanno riguardato l'implementazione di classi, per la gestione degli oggetti e della loro rappresentazione, l'implementazione di un' interfaccia utente e l'introduzione delle istruzioni necessarie per la collimazione del flusso dati provenienti dai dispositivi di input.

Tali istruzioni riguardano l'elaborazione dei dati provenienti dal dispositivo magnetico e dei dati provenienti dalla webcam. Mentre i primi vengono utilizzati per determinare la posizione nello spazio del puntatore i secondi consentono di calcolare le trasformazioni prospettiche per proiettare gli oggetti virtuali sulla scena.

La collimazione del flusso dati ingloba due procedure di calcolo:

- La Registrazione;
- La Proiezione;

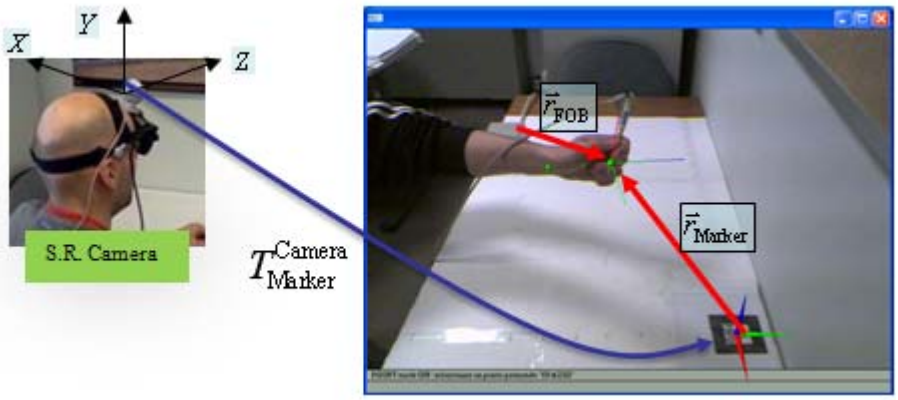


Figura 3.2.10. La relazione geometrica tra i sistemi di riferimento di camera e marker nella fase di registrazione.

La registrazione si occupa del tracciamento delle posizione dell'osservatore rispetto ad un sistema di riferimento assegnato convenzionalmente ad uno o più marker (v. Capitolo 2). Da un punto di vista computazionale si cerca la relazione *T* di trasformazione geometrica tra i sistema di riferimento della camera e quello assoluto assegnato al marker (v. Figura 3.2.10).

Contemporaneamente alla fase di registrazione la fase di proiezione calcola la posizione della punta del puntatore (*I-Pen* in Figura 3.2.11) *P* nel sistema di riferimento

globale del marker ($O_w - X_w Y_w Z_w$ in Figura 3.2.11). Tale operazione è necessaria poiché i punti selezionati, per il corretto allineamento con la scena, devono essere visualizzati nel sistema di riferimento globale delle OpenGL, il quale per default è allineato con il sistema di riferimento assegnato al marker. Lo scopo della fase di Proiezione è pertanto quello di calcolare le componenti del vettore $\vec{r}'$ nel riferimento del marker (v. Figura 3.2.11) secondo la (3.1).

$$\left\{\vec{r}'\right\} = \left[T\right]_{\text{Trasm.}}^{\text{Marker}} \left\{\vec{r} - \vec{d}\right\} \qquad (3.1)$$

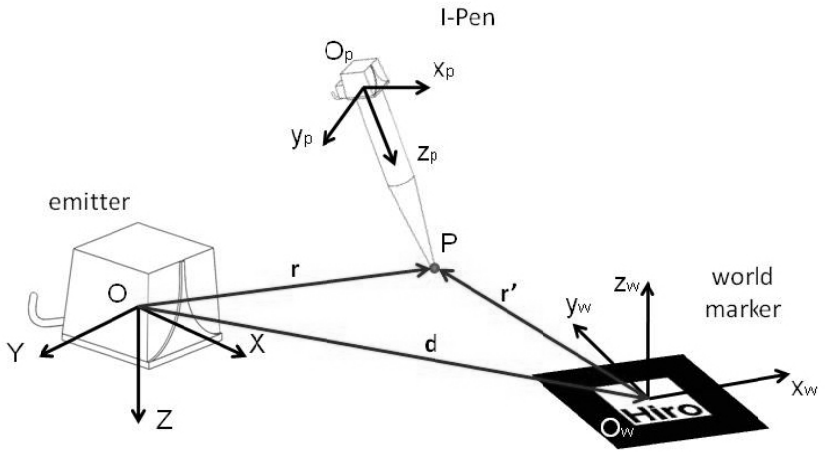


Figura 3.2.11. Relazioni geometriche tra i sistemi di riferimento di marker, puntatore ed emettitore.

Dove $\vec{r}$ rappresenta il vettore posizione che individua *P* nel sistema di riferimento dell'emettitore e $\vec{d}$ e $\vec{T}$ rappresentano rispettivamente il vettore traslazione e la matrice di rotazione tra i sistemi di riferimento del marker e dell'emettitore.

Il vettore $\vec{r}$ deriva dall'elaborazione del flusso dati provenienti dal dispositivo magnetico e da una fase iniziale di impostazione del sistema denominata qualifica.

In Tabella 3.2.4 si riportano le principali istruzioni utilizzate nell'elaborazione dei dati del dispositivo magnetico.

Tabella 3.2.4. Le istruzioni per la gestione del dispositivo di puntamento.

Istruzione	Commenti
`#include <wincon.h>` `#include "bird.h"`	Sono le dichiarazioni delle librerie necessarie per la gestione di dispositivi esterni al PC
`birdRS232WakeUp()`	Funzione di *Bird.h* per avviare la connessione con l'unità di potenza del Flock of Bird

	specificando i parametri di connessione.
`birdGetSystemConfig()`	Funzione che legge i dati relativi alla configurazione del sistema.
`birdGetFastDeviceConfig()`	Funzione che legge i dati relativi alla configurazione del dispositivo di puntamento.
`birdStartFrameStream()`	Funzione per inizializzare l'importazione del flusso di dati.
`birdGetMostRecentFrame()`	Legge i dati relativi all'acquisizione più recente e li salva in una variabile di tipo BIRDREADING.

La struttura dati *BEARDREADING*, argomento della funzione `birdGetMostRecentFrame` contiene le seguenti variabili:

- *position*, che contiene tre valori *nX*, *nY* e *nZ* che definiscono la posizione del sensore rispetto al trasmettitore.
- *angles*, che contiene tre parametri *nAzimuth*, *nElevation*, *nRoll* che definiscono l'orientamento del sensore rispetto al trasmettitore.

Tali variabili cinematiche consentono di individuare la posizione del sensore rispetto al trasmettitore. In Figura 3.3.10 si riportano i sistemi di riferimento solidali al sensore ed al trasmettitore.

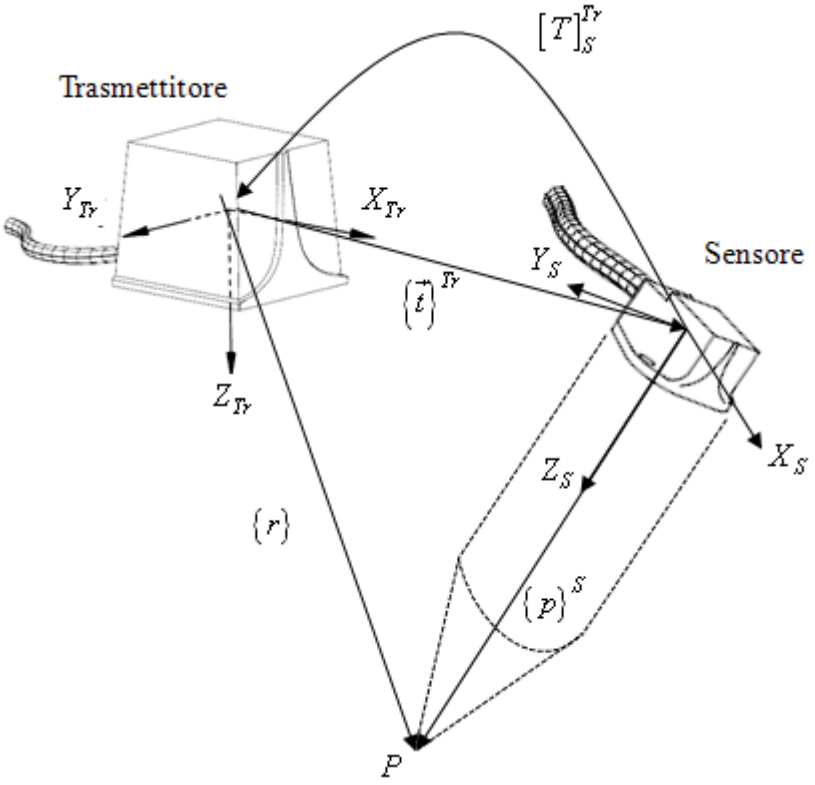


Figura 3.2.12. I parametri di posizione del sensore rispetto al trasmettitore.

Le coordinate del vettore $\vec{r}$, che rappresenta la posizione della punta *P* del puntatore nel sistema di riferimento del trasmettitore, può essere calcolata con la seguente espressione:

$$\{r\}^{Tr} = \{t\}^{Tr} + [T]_S^{Tr} \cdot \{p\}^S \tag{3.2}$$

dove $\{t\}^{Tr} = \{Xn \quad Yn \quad Zn\}^t$, e la matrice $[T]_S^{Tr}$ dipende dai parametri sopramenzionati secondo la seguente espressione:

$$[T]_S^{Tr} = \begin{bmatrix} \mathrm{c}(nE)\cdot\mathrm{c}(nA) & \begin{matrix}-\mathrm{c}(nR)\cdot\mathrm{s}(nA)+\\ \mathrm{s}(nR)\cdot\mathrm{s}(nE)\cdot\mathrm{c}(nA)\end{matrix} & \begin{matrix}\mathrm{s}(nR)\cdot\mathrm{s}(nA)+\\ \mathrm{c}(nR)\cdot\mathrm{s}(nE)\cdot\mathrm{c}(nA)\end{matrix} \\ \mathrm{c}(nE)\cdot\mathrm{s}(nA) & \begin{matrix}\mathrm{c}(nR)\cdot\mathrm{c}(nA)+\\ \mathrm{s}(nR)\cdot\mathrm{s}(nE)\cdot\mathrm{s}(nA)\end{matrix} & \begin{matrix}-\mathrm{s}(nR)\cdot\mathrm{c}(nA)+\\ \mathrm{c}(nR)\cdot\mathrm{s}(nE)\cdot\mathrm{s}(nA)\end{matrix} \\ -\mathrm{s}(nE) & \mathrm{s}(nR)\cdot\mathrm{c}(nE) & \mathrm{c}(nR)\cdot\mathrm{c}(nE) \end{bmatrix}$$

dove $\cos(.)$ e $\sin(.)$ sono stati abbreviati con le lettere $c(.)$ e $s(.)$ e i parametri cinematici *nAzimuth*, *nElevation*, *nRoll* rispettivamente con *nA*, *nE* e *nR*[16]. A differenza dei parametri cinematici che vengono elaborati alla frequenza di acquisizione e che variano al variare della posizione del puntatore, il vettore $\{p\}^S$ è costante per ogni ciclo di acquisizione. Nel caso specifico (v. Figura 3.2.13) il vettore $\{p\}^S$ è allineato con l'asse *Z* del sistema di riferimento del puntatore e quindi è del tipo:

$$\{p\}^S = \{0 \quad 0 \quad \Delta Z\}_{Sensore}$$

La fase di qualifica si riconduce pertanto al calcolo del valore di ΔZ. Valore ottenibile calcolando la differenza del valore rilevato per la coordinata *Z* in due posizionamenti successivi del sensore su un piano parallelo al piano *XY* dell'emettitore. Nel primo posizionamento si poggia la superficie piana del sensore mentre nel secondo la punta

[16] Per ulteriori dettagli sulla matrice di trasformazione si rimanda alla documentazione tecnica del dispositivo magnetico [61]e al testo [34].

dell' *I-pen* disposto, con l'asse di rivoluzione, ortogonalmente alla superficie di selezione.

Figura 3.2.13. *I-pen* puntatore in legno installato sul sensore magnetico.

Al pari dei componenti del vettore $\{p\}^S$ anche quelli della matrice $[T]^{\text{Marker}}_{\text{Trasm.}}$ della (3.1) rimane costante[17], per ogni ciclo di acquisizione, e può essere calcolata, successivamente alla qualifica del puntatore, con un apposito sistema. Un modo di procedere, è quello di selezionare, con il puntatore magnetico, tre punti appartenenti al marker nell'ordine e nella posizione di Figura 3.2.14. Dove il punto $\{P_0\}$ si trova nel centro del marker il punto $\{P_1\}$ si trova al centro del bordo destro e il punto $\{P_2\}$ al centro del bordo superiore.

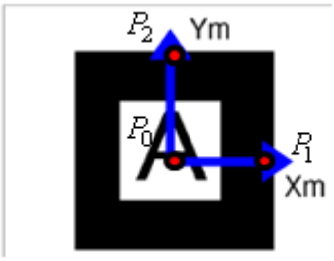


Figura 3.2.14. I punti di selezione sul marker per il calcolo della matrice di trasformazione.

Noti i valori dei componenti dei tre punti, nel sistema di riferimento dell'emettitore magnetico, la matrice di trasformazione $[T]^{\text{Marker}}_{\text{Trasm.}}$ può essere calcolata secondo la seguente:

---

[17] Tale affermazione è vera solo se la posizione relativa marker-emettitore magnetico rimane costante durante tutto il processo di acquisizione.

$$[T]_{\text{Trasm.}}^{\text{Marker}} = \begin{bmatrix} \{P_1 - P_0\}^T / \|\cdot\| \\ \{P_2 - P_0\}^T / \|\cdot\| \\ \{P_1 - P_0\}^T \wedge \{P_2 - P_0\}^T / \|\cdot\| \end{bmatrix} \qquad (3.3)$$

Mentre, per come sono state definite le entità geometriche, i componenti del vettore $\{d\}$ di Figura 3.2.11 sono le stesse del vettore $\{P_0\}$ ottenute nella fase di selezione.

Quindi la quantità di dati e l'elaborazione dei dati stessi risulta più complessa per una applicazione di AR dotata della funzionalità di puntamento. Infatti è necessario prevedere procedure di impostazione iniziale, per calcolare le costanti geometriche delle trasformazioni di collimazione dati, e funzioni di calcolo per determinare la posizione del puntatore collegato al sensore magnetico.

3.2.5 Strutture dati e metodi per le entità virtuali

La necessità di manipolare entità geometriche coincide con l'esigenza di una struttura dati e di metodi per la creazione, la modifica e la visualizzazione delle geometrie stesse. Per soddisfare tale esigenza è stata sviluppata la classe *CCAD*, in grado di contenere tutte le entità geometriche elementari e derivate, disponibili nell'ambiente di modellazione.

Tale classe, sfruttando le proprietà di polimorfismo ed ereditarietà delle classi C++, eredita la struttura dati *VRML* e i metodi della classe *Object* presentati nell'Appendice A. Inoltre il polimorfismo delle classi *C++* è stato impiegato per corredare la classe *CCAD* con i metodi per la visualizzazione e la memorizzazione delle entità geometriche più diffuse nella modellazione (curve, superfici, solidi).

Il software implementato utilizza la classe *CCAD* secondo l'algoritmo riportato in Figura 3.2.15. Attraverso l'interfaccia utente il software riceve in input l'attivazione del comando di modellazione. Sulla base dei dati provenienti dal dispositivo di puntamento genera un'istanza della classe *CCAD*, ovvero un oggetto virtuale, che memorizza nella lista degli oggetti da disegnare. Ad ogni ciclo di render, sfruttando i metodi della classe, il software proietta gli *N* oggetti sulla finestra grafica.

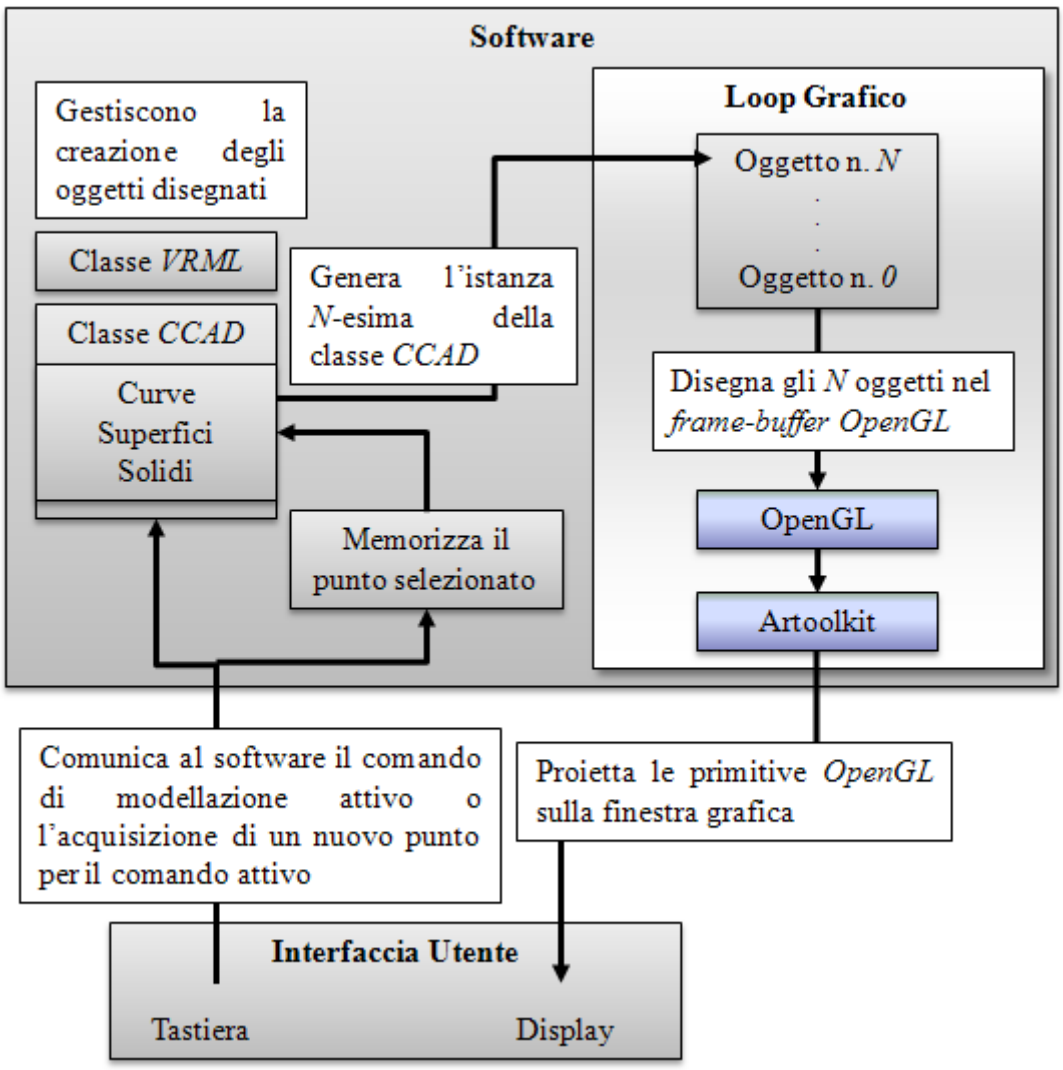


Figura 3.2.15. Logica di gestione delle entità virtuali.

Tale schema mostra l'esigenza di dotare la classe di una struttura dati utile all'esportazione/importazione e di implementare metodi in grado di ricevere dati, per inizializzare la classe, e tradurre la struttura dati in primitive grafiche OpenGL per la loro visualizzazione.

Le entità geometriche da gestire si possono raggruppare in tre famiglie:

- Linee, polilinee, curve parametriche;
- Superfici piane e tridimensionali;
- Solidi di base e/o ottenuti mediante modellazione;

Lo schema logico di utilizzo della classe è riportato in Figura 3.2.5, mentre la dichiarazione della classe in Tabella 3.2.5.

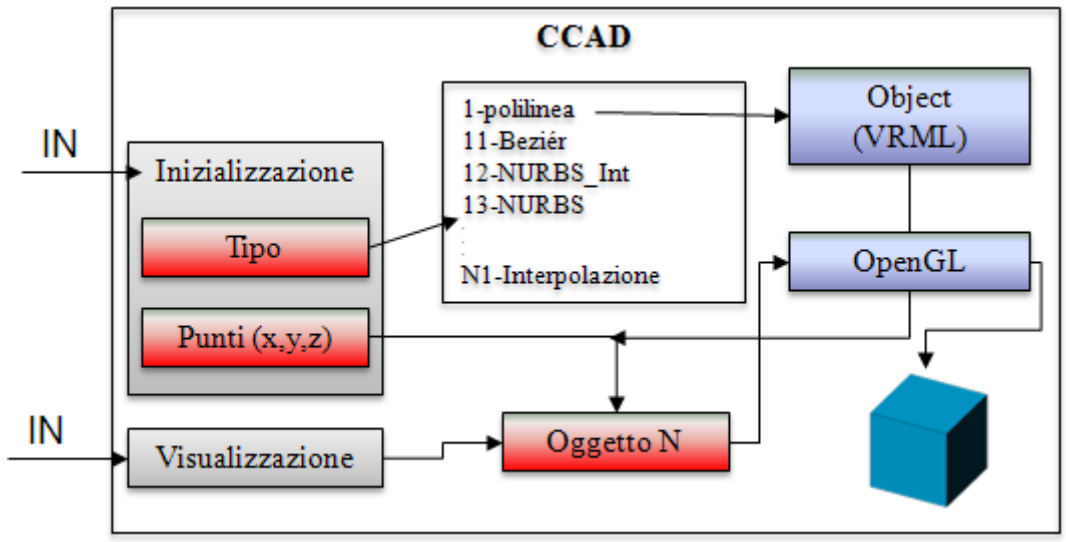


Figura 3.2.16. Logica della classe *CCAD*.

Esistono due soli metodi d'ingresso, uno relativo all'inizializzazione di una entità virtuale e l'altro alla visualizzazione dell'entità stessa mediante le primitive grafiche *OpenGL*. Secondo tale logica, una nuova istanza della classe passa attraverso il metodo di inizializzazione, che prevede la specifica del tipo e il passaggio dei punti appartenenti al nuovo oggetto (v. funzione `CCAD()` di Tabella 3.2.5), mentre un apposito metodo per la visualizzazione (v. funzione `Disegna(void)` di Tabella 3.2.5) consente al programma chiamante di visualizzare nel contesto OpenGL l'*N-esimo* oggetto creato.

Tabella 3.2.5. Dichiarazione della classe *CCAD* come sottoclasse della classe *Solid* contenente una struttura dati *VRML*.

```
#include <windows.h>
#include <gl/gl.h>
#include <gl/glu.h>
#include <gl/glut.h>
class CCAD:public Solid
{
public:
//Entità geometrica da disegnare
int TYPE;
//Inizializzazione della classe
CCAD(int tipo=1,unsigned int NPts=0,unsigned int NFacce=0,
struct Point3D * Pts=0);
//Funzione di disegno delle primitive geometriche
void Disegna(void);
//Funzioni di disegno delle entità geometriche
…
virtual ~CCAD(void);
};
```

La funzione di inizializzazione è stata implementata con la seguente struttura:

Tabella 3.2.6. Struttura della funzione di inizializzazione per la classe *CCAD*.

```
CCAD::CCAD(int tipo,
           unsigned int NPts,
           unsigned int NFacce,
           struct Point3D  *Pts)
{
 unsigned int i,j,k;

 if(!tipo)TYPE=tipo;
 //Assegnazione dei valori alla classe VRML in funzione del
tipo
 switch(TYPE)
 {
 case 1:  //Polilinea
  CCAD::vert_num=NPts;
  CCAD::face_num=1;
  //Allocazione del vettore dei punti
  CCAD::make_vertices(CCAD::vert_num);
  for(i=0;i<CCAD::vert_num;i++)
  {
   CCAD::x[i]=Pts[i].X   ;
   CCAD::y[i]=Pts[i].Y  ;
   CCAD::z[i]=Pts[i].Z ;
  }
 case 11:  //Beziér
  …
  break;
 }
```

dove in funzione del tipo di geometria che si intende creare, la classe assegna i valori dei vertici alla struttura dati *VRML* definita all'interno della superclasse *Solid* ed utilizza le strutture dati generalmente assegnate alle facce per definirne il tipo. Inoltre, utilizzando le strutture dati caricate e in funzione della tipologia di entità, la funzione `Disegna` proietta sulla finestra grafica l'oggetto creato.
Alcuni esempi relativi alla gestione delle curve parametriche e degli oggetti solidi vengono di seguito riportati.

L'elaborazione dati per la gestione di una entità geometrica di tipo polilinea si riduce alla memorizzazione dei vertici della polilinea nella sequenza di selezione. Il codice di Tabella 3.2.6 riporta le istruzioni di inizializzazione della classe, le quali definiscono un poligono ad un'unica faccia con un numero di vertici pari a quello dei punti selezionati e attraverso l'allocazione dinamica dei vettori *x*, *y*, *z* e la successiva assegnazione dei valori delle coordinate dei punti selezionati, associano gli *NPts* punti alla nuova istanza di classe.
La funzione di disegno per la polilinea (`DisegnaPolilinea`) con le relative istruzioni OpenGL di disegno è riportata in Tabella 3.2.7

Tabella 3.2.7. La funzione di disegno per la polilinea.

```
void CCAD::DisegnaPolilinea(void)
{
 unsigned int i,j,k;
 glColor3f(1.0,0.0,0.0);
switch (CCAD::face_num)
case (1):
 glBegin(GL_POLYGON);
 for(i=0;i<CCAD::vert_num ;i++)
 {
 glVertex3f(CCAD::x[i],CCAD::y[i],CCAD::z[i]);
 }
…
 glEnd();
 return;
}
```

Un caso particolare è quello della gestione di oggetti derivati da altri oggetti. In Figura 3.2.17 si riporta, a titolo di esempio, il caso di un oggetto solido generato per estrusione. In questo caso la struttura dati del poligono di base è inglobata nella struttura dati del solido estruso. Il metodo di inizializzazione della classe, partendo dagli n vertici $(P_0,...,P_{n-1})$ del poligono di base con una sola faccia, dovrà generare una lista di $2n$ vertices $(P_0,...,P_{n-1},P_n,...,P_{2n-1})$ con $n+2$ facce, dove le coordinate dei vertici $(P_n,...,P_{2n-1})$ sono definiti dalla (3.4).

$$\{P_{i+n}\}=\{P_i\}+\{v\} \tag{3.4}$$

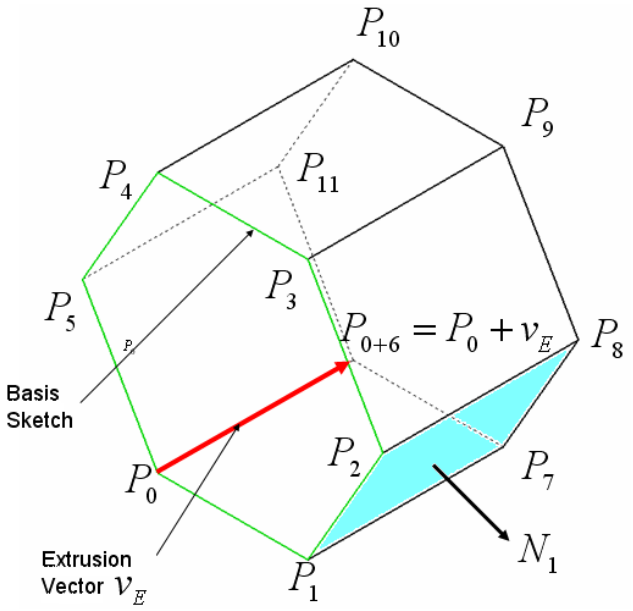


Figura 3.2.17. Struttura dati di un oggetto virtuale creato per estrusione.

L'operazione successiva consiste nel registrare le *2n* liste di vertici relative alle facce. Ovviamente rispettando la struttura dati *VRML*, la faccia di base e la faccia finale saranno definite dalle seguenti istruzioni:

```
CCAD::v_array[0]={0,1,2,…,n-1};
CCAD::v_array[1]={n,1,2,…,2n-1};
```

mentre le liste di vertici delle facce laterali del solido estruso possono essere definite mediante le seguenti:

```
for(i=0;i<n;i++)
 {
   CCAD::v_array[1+i]={i,i+1,n+1+i,n+i};
 }
```

Infine la fase di visualizzazione dell'oggetto può essere implementata come la funzione `DisegnaSolido` di Tabella 3.2.8.

Tabella 3.2.8. La funzione DisegnaFacce.

```
void CCAD::DisegnaSolido(void)
{
 unsigned int i, j, k;
  for(k=0;k<CCAD::face_num;k++)
  {
  glBegin(GL_POLYGON);
  for(i=0;i<CCAD::faces[k].v_count;i++)
  {
   glColor3f(0.2*i,0.0,0.2);
   glVertex3f(CCAD::x[CCAD::faces[k].v_array[i]],
        CCAD::y[CCAD::faces[k].v_array[i]],
           CCAD::z[CCAD::faces[k].v_array[i]]);
  }
  glEnd();
  };
 return;
}
```

Le curve e le superfici parametriche hanno richiesto un trattamento diverso. Tali entità geometriche infatti non sono caratterizzate da facce e vertici ma da parametri di controllo e punti di controllo o interpolazione.

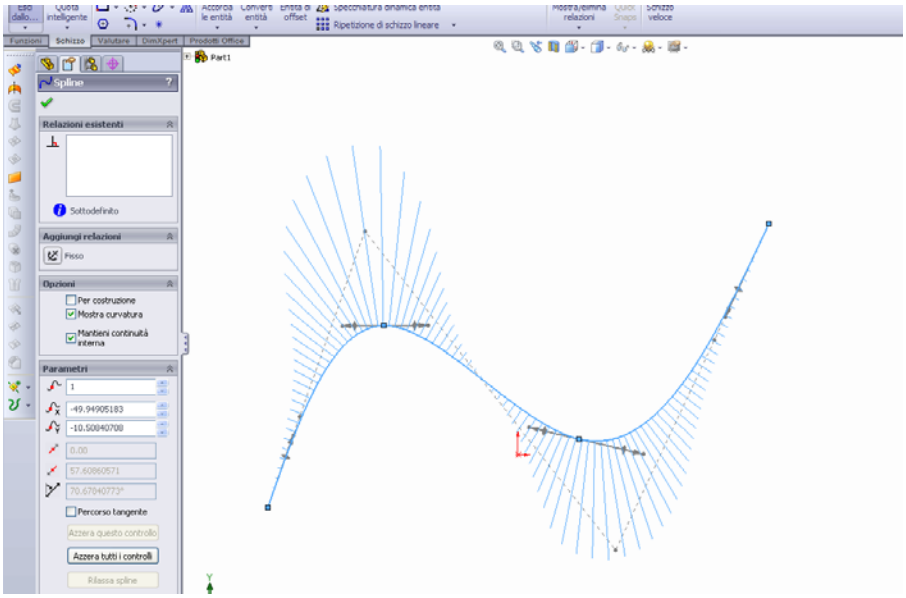


Figura 3.2.18. Esempio di *B-Spline* disegnata al CAD.

Riferendoci al caso di una curva parametrica tipo *B-Spline*, come quella riportata in Figura 3.2.18, per definirne in maniera univoca la forma occorre conoscere :

- Grado dei segmenti interpolanti della curva;
- Punti appartenenti al poligono di controllo;
- Vettore dei nodi;

In alternativa la curva è univocamente definita da una lista di punti di interpolazione e dal metodo utilizzato per calcolare i parametri sopramenzionati.

A titolo di esempio in Tabella 3.2.9 si riportano di dati relativi alla curva di Figura 3.2.18.

Tabella 3.2.9. Parametri analitici di una curva *B-Spline*.

Dim = 3
CtrlPoints = 6
NumKnots = 10
NumCtrlPts =4
CtrlPts[2] = (-49,9490518331226, -10,508407079646, 0) mm
CtrlPts[5] = (-43,5953978157604, 7,61288542305259, 0) mm
CtrlPts[8] = (-30,5861746251753, 44,7165643494273, 0) mm
CtrlPts[11] = (19,5476429798602, -19,0485086084416, 0) mm
CtrlPts[14] = (39,2106220477055, 22,8185209969568, 0) mm
CtrlPts[17] = (50,221997471555, 46,2642857142857, 0) mm
Knot[20] = 0
Knot[21] = 0
Knot[22] = 0
Knot[23] = 0
Knot[24] = 0,295792090676914
Knot[25] = 0,605639733468403
Knot[26] = 1
Knot[27] = 1
Knot[28] = 1

```
Knot[29] = 1
PtsInterp[0] = (-49,9490518331226, -10,508407079646, 0) mm
PtsInterp [1] = (-26,7486725663717, 25,7933628318584, 0) mm
PtsInterp [2] = (12,2825537294564, 3,1388748419722, 0) mm
PtsInterp [3] = (50,221997471555, 46,2642857142857, 0) mm
```

Sfruttando il polimorfismo delle classi C++ è stata sovrascritta la struttura dati *VRML*. In particolare al posto di un vettore di facce è stato definito, per la fase di inizializzazione, un vettore di 4 componenti interi che rappresentano rispettivamente:

- Grado della curva;
- Numero di punti di controllo;
- Numero di nodi;
- Numero di punti di interpolazione;

mentre i vettori *x*, *y*, *z* della classe *Object* vengono utilizzati per caricare le coordinate dei punti del poligono di controllo, il vettore dei nodi e le coordinate dei punti di interpolazione secondo l'ordine riportato in Tabella 3.2.9.

Per la fase di visualizzazione sono state utilizzate le funzioni OpenGL dedicate alla visualizzazione di curve e superfici parametriche. La sequenza di istruzioni è quella riportata in Tabella 3.2.10.

Tabella 3.2.10. Istruzioni OpenGL per la visualizzazione di una NURBS.

```
void CCAD::DisegnaNurbsCTRL (void)
{
 int i;
// Puntatore ad un oggetto di tipo NURBS
 GLUnurbsObj *pNurb = NULL;
 pNurb = gluNewNurbsRenderer();

// Numero di nodi
 int KnotNum;
// Numero di punti del poligono di controllo
 int CtrlPtsNum;
// Numero di punti di interpolazione
 int InterpPtsNum;
// Matrice dei punti di interpolazione
 float **ctrlPoints;
// Vettore dei nodi
 float *Knot;
// Grado della curva
 float Dim;

Dim=CCAD::faces[0].v_array[0];
 KnotNum=CCAD::faces[0].v_array[1];
 CtrlPtsNum=CCAD::faces[0].v_array[2];
 Knot=(float *) calloc(KnotNum,sizeof(float));
 for(i=0;i<KnotNum;i++)
 {
  Knot[i]=CCAD::x[i];
 }
```

```
 ctrlPoints=(float**)calloc(CtrlPtsNum,sizeof(float));
 for(i=0;i<3;i++)
 {
  ctrlPoints[i]=(float*)calloc(3,sizeof(float));
 }
 for(i=0;i<CtrlPtsNum;i++)
 {
  ctrlPoints[i][0]=CCAD::x[i];
  ctrlPoints[i][1]=CCAD::y[i];
  ctrlPoints[i][2]=CCAD::z[i];

 }
// Rendering
 gluBeginCurve(pNurb);
 // Evaluate the surface
    gluNurbsCurve(pNurb,KnotNum,Knot,Dim,&ctrlPoints[0][0],
        4,
        GL_MAP1_VERTEX_3);
 gluEndCurve(pNurb);
 return;
}
```

È necessario precisare che la funzione `DisegnaNurbsCTRL` si riferisce alla visualizzazione di una curva parametrica definita mediante poligono di controllo, ma è applicabile anche alle curve parametriche generate per interpolazione.

3.2.6 Interattività

Come accennato nella parte introduttiva di questa sezione, un software CAD sviluppato in ambiente di AR e dotato di uno strumento di puntamento tridimensionale consente all'operatore una maggiore interattività con lo spazio di lavoro. In un sistema di questo tipo l'operatore ha l'opportunità di selezionare liberamente punti nello spazio di modellazione, senza dover necessariamente definire piani di lavoro o inserire numericamente le coordinate dei punti da acquisire. La funzionalità di selezione deve consentire la selezione di punti reali e punti ed oggetti virtuali. Mentre per i primi il problema si riconduce all'implementazione di una funzione di acquisizione, che memorizza le coordinate della punta del sensore magnetico al verificarsi di uno specifico evento, per gli oggetti virtuali è necessario attivare una procedura più laboriosa. A tal riguardo si è deciso di modificare la funzione di selezione delle OpenGL, la quale restituisce una lista di oggetti, ordinati per profondità, che ricadono nell'area intorno al pixel selezionato. Perché il processo funzioni è necessario nominare le entità geometriche con numeri interi diversi e definire le dimensioni dell'area di selezione che si trova intorno al pixel. Mentre i dettagli della procedura di selezione si possono trovare

in Appendice A, in questa sezione si introducono le modifiche e gli accorgimenti da utilizzare per adattare tale procedura al software sviluppato.

La prima operazione riguarda la memorizzazione dei parametri intrinseci della camera, durante la fase di inizializzazione, caricati e inizializzati con le funzioni `arParamLoad` e `arInitCparam`,e la conversione dei parametri intrinseci della camera in un formato compatibile con le OpenGL. L'istruzione:

```
argConvGLcpara( &cparam, AR_GL_CLIP_NEAR, AR_GL_CLIP_FAR, gl_cpara );
```

converte i parametri `cparam` nel formato `gl_cpara`, che è un vettore di 16 elementi utilizzato per definire le trasformazioni prospettiche in ambiente OpenGL.

La seconda operazione interessa la definizione di una funzione che elabori l'evento di selezione. In Tabella 3.2.11 si riporta una funzione di esempio, `ProcessSelection()`, che riceve in input le coordinate del pixel selezionato sulla finestra grafica e restituisce nel vettore `buffer` le liste di oggetti selezionati secondo la sintassi riportata in Figura A.5.

Tabella 3.2.11. Funzione per la selezione di oggetti.

```
void ProcessSelection(int u_pix, int v_pix)
 {
 double glCamViewPars[16];
 int hits;
 double gl_para[16];
 GLsizei BUFF_SIZE = 512;
 GLuint buffer[BUFF_SIZE];
 GLint viewport[4];
 glGetIntegerv(GL_VIEWPORT, viewport);
//Comunica il vettore di selezione
 glSelectBuffer(BUFF_SIZE, buffer);
//Avvia la modalità di selezione
 glRenderMode(GL_SELECT);
 glMatrixMode(GL_PROJECTION);
 glPushMatrix();
 glLoadIdentity();
//Proiezione con i parametri intrinseci della camera
 glMultMatrixd( gl_cpara );
//Area di selezione
 gluPickMatrix(u_pix, viewport[3]-v_pix, 3.0, 3.0, viewport);
 glMatrixMode(GL_MODELVIEW);
 glPushMatrix();
//TransMatCam matrice di trasformazione dal riferimento della
//camera a quello del marker
 argConvGlpara(TransMatCam, gl_para);
 glMultMatrixd( gl_para );
//Ridisegna la scena per la selezione
 DrawObject();
 glPopMatrix();
// Rileva la lista degli oggetti selezionati
```

```
  hits = glRenderMode(GL_RENDER);
// Se almeno un oggetto è stato selezionato controlla la lista
di selezione
if(!hits)
  ProcessEvent(BUFF_SIZE,buffer);
  glPopMatrix();
  }
```

È importante sottolineare come prima delle istruzioni per ridisegnare la scena (`DrawObject()`) sia necessario allineare il punto di vista sulla scena virtuale con l'istruzione `glMultMatrixd( gl_cpara )` che carica i parametri intrinseci della camera nelle trasformazioni di proiezione della scena OpenGL.

L'ultima operazione riguarda la trasformazione delle coordinate $\{X \quad Y \quad Z\}^t$, della punta del puntatore *P* al momento della selezione, nelle corrispondenti coordinate in pixel $\{u \quad v\}^t$ dello spazio immagine da passare in argomento alla funzione `ProcessSelection`. Tale trasformazione si ottiene moltiplicando le coordinate del punto *P* per le matrici di trasformazione e proiezione delle OpenGL (v. Appendice A). Tali matrici si ricavano, per ogni ciclo di render, mediante la funzione `glGet()`.

3.2.7 Il software AR-CAD 2.0

Dopo aver descritto le problematiche e le strategie affrontate per lo sviluppo del sistema, questa sezione presenta l'interfaccia di utilizzo del software e mostra alcuni esempi di modellazione.

Come anticipato nella sezione dedicata alla descrizione del layout, il sistema è dotato di due dispositivi di input che ricevono i comandi (tastiera) e registrano la posizione del puntatore nello spazio (puntatore magnetico). Attraverso tali dispositivi (v. Figura 3.2.19) il progettista attiva i comandi di modellazione e seleziona i punti nello spazio.

L'interfaccia di visualizzazione (occhiali stereoscopici) visualizza il comando attivo sulla linea di comando e la posizione del puntatore sullo spazio visivo dell'utente.

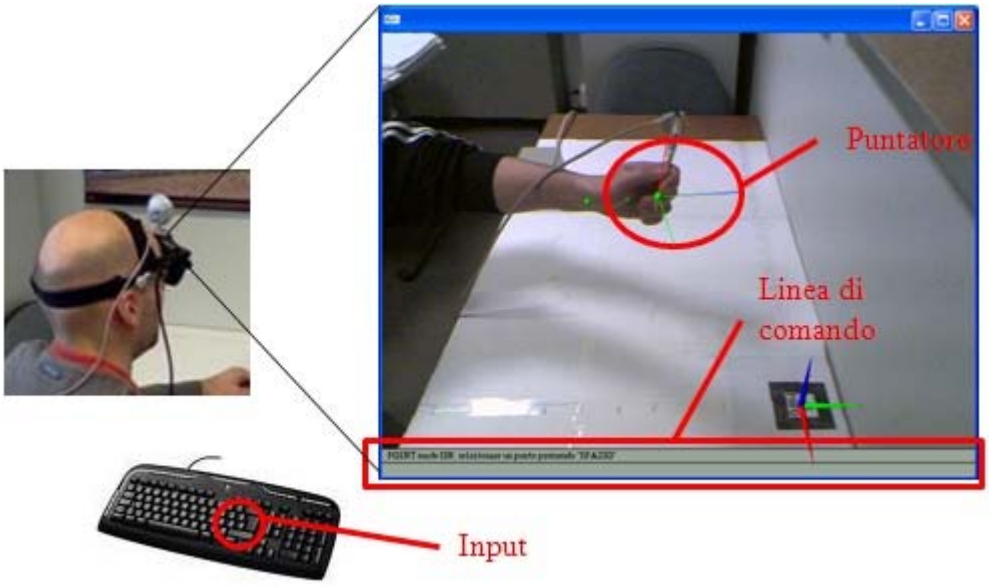
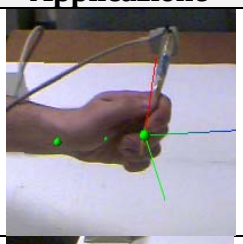
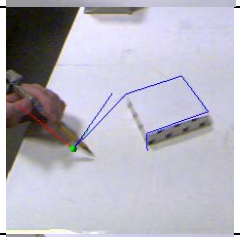
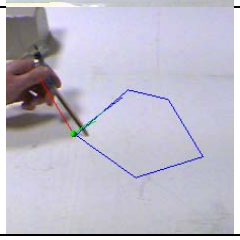


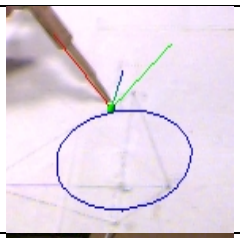
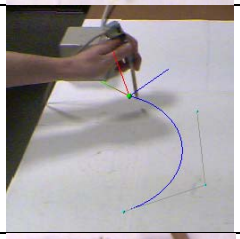
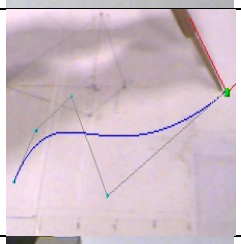
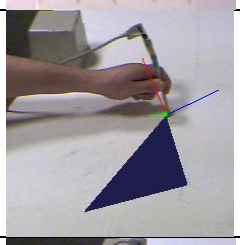
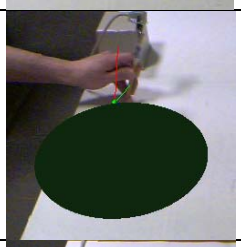
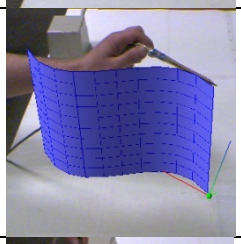
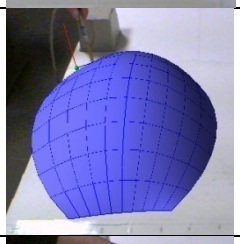
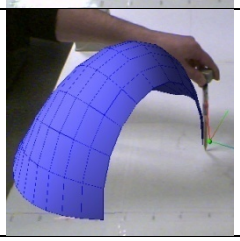
Figura 3.2.19. L'interfaccia di modellazione.

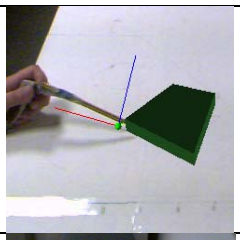
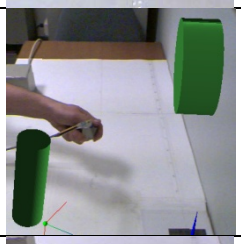
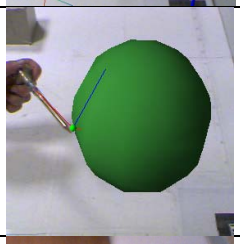
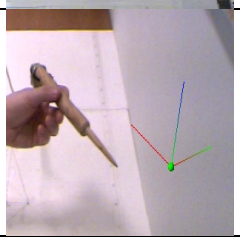
In Tabella 3.2.12 si riporta l'elenco dei comandi di modellazione e i relativi tasti di attivazione[18].

Tabella 3.2.12. Descrizione dei comandi di modellazione.

Applicazione	Comando	Evento
	Punto	"o" per attivare il comando " " per acquisire il punto
	Polilinea	"m" per attivare il comando " " per acquisire il punto
	Poligono	"p" per attivare il comando " " per acquisire i vertici

[18] I tasti associati ai comandi sono riportati tra virgolette. Con " " ci si riferisce alla barra spaziatrice, associata all'evento di selezione dei punti.

	Cerchio	“c” per attivare il comando “ “ per acquisire il centro e raggio, richiede la selezione di un piano
	Curva di Bezièr	“b” per attivare il comando “ “ per acquisire i punti del poligono di controllo
	NURBS	“u” per attivare il comando “ “ per acquisire i punti del poligono di controllo
	Superficie Poligono	“P” per attivare il comando “ “ per acquisire i vertici.
	Disco	“C”per attivare il comando. “ “ per acquisire centro e raggio, richiede la selezione di un piano.
	Estrusione di superficie	“B” per attivare il comando. “ “ per acquisire l’altezza. Applicabile solo ad una curva piana
	Superficie di Beziér	“N” per attivare il comando. Richiede l’acquisizione di una griglia di punti. “n” per passare da una fila all’altra.
	Superficie Nurbs	“M” per attivare il comando. Richiede l’acquisizione di una griglia di punti. “m” per passare da una fila all’altra.

	Estrusione di un poligono	“E” per attivare il comando. “ “ per acquisire il vettore di estrusione. Richiede un poligono di estrusione
	Cilindro	“T” per attivare il comando. “ ” per definire l’altezza di estrusione. Richiede una circonferenza o un cerchio di base
	Sfera	“E” per attivare il comando. “ “ per acquisire centro e raggio.
	Piano di schizzo	“k” per attivare il comando. Richiede la selezione di tre punti nello spazio non allineati o il tasto “k” per uscire dalla modalità selezione

Di seguito sono riportati alcuni esempi di modellazione che sfruttando l’interattività del sistema riproducono oggetti reali.

Il primo esempio riguarda la modellazione della scocca esterna di un telefono da tavolo. La Figura 3.2.20 mostra il telefono in questione visualizzato nella finestra dell’AR-CAD 2.0.

Il primo passo è stato quello di definire con il comando *K* un piano di schizzo sulla parete posteriore del telefono per riprodurre mediante due cilindri le fiancate laterali (v. Figura 3.2.21).

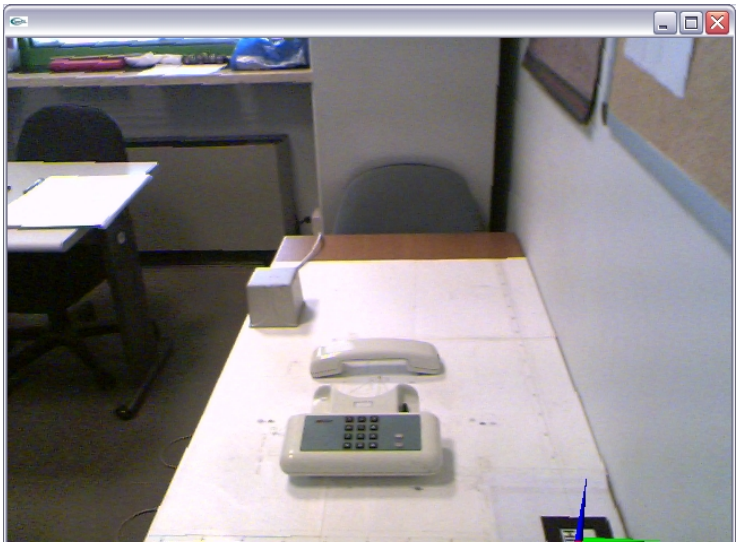


Figura 3.2.20. Modellazione di un telefono fisso.

Figura 3.2.21. Modellazione dei fianchi del telefono.

Figura 3.2.22. Modellazione della zona frontale.

Il passo successivo è stato definire un piano verticale passante per l'asse di uno dei cilindri appena creati per modellare la parte frontale con un terzo cilindro (v. Figura 3.2.22).

Successivamente sono state inserite due superfici sferiche nei punti in cui si intersecano i cilindri, in modo da ottenere una superficie continua, Figura 3.2.23.

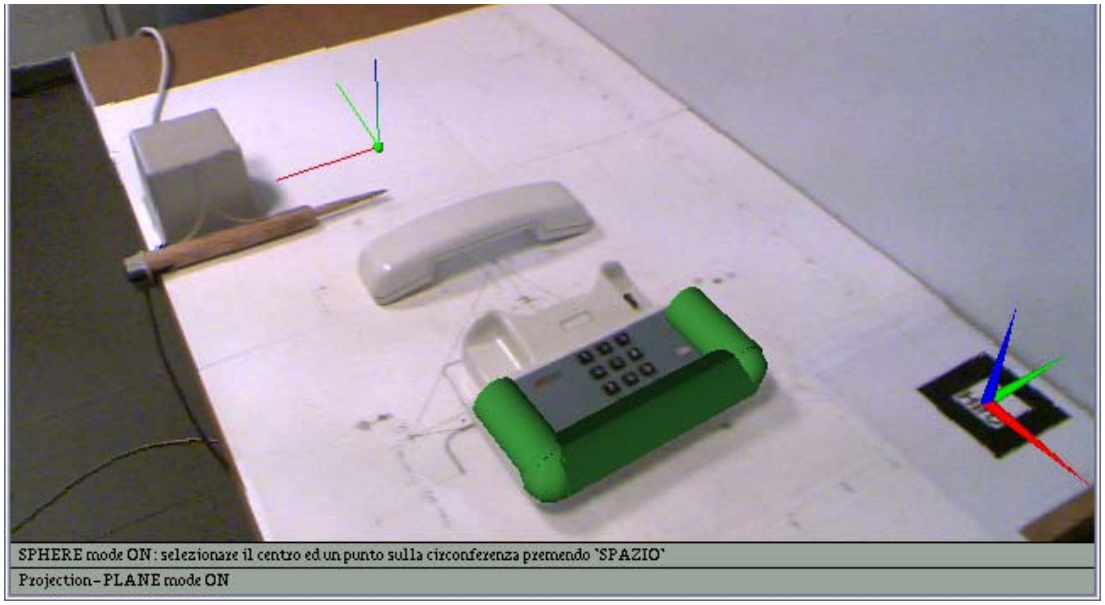


Figura 3.2.23. Modellazione degli angoli.

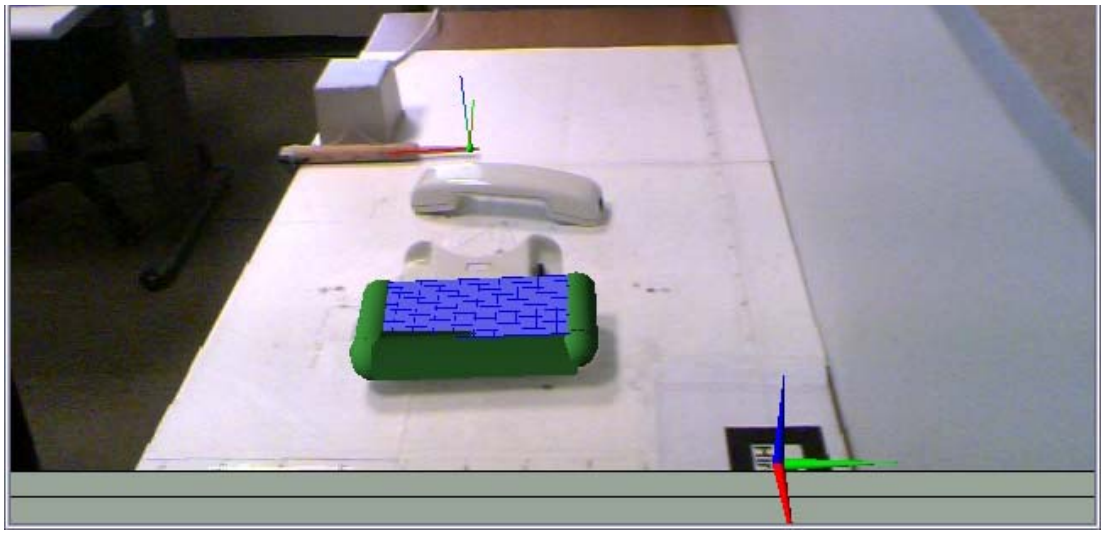


Figura 3.2.24. Modellazione della superficie della tastiera.

Al modello sono state poi aggiunte le superfici posteriore e superiore (superfici di Beziér di Figura 3.2.24).

La Figura 3.2.25 mostra il successivo step di modellazione riguardante il basamento del telefono, definito come estrusione di un quadrilatero, e l'alloggiamento della cornetta ottenuta mediante l'acquisizione dei punti di controllo di una superficie NURBS.

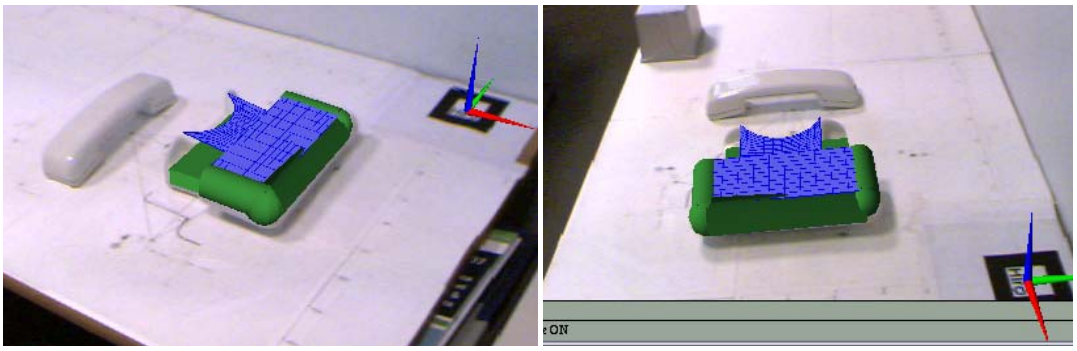


Figura 3.2.25. Fase finale della riproduzione del basamento.

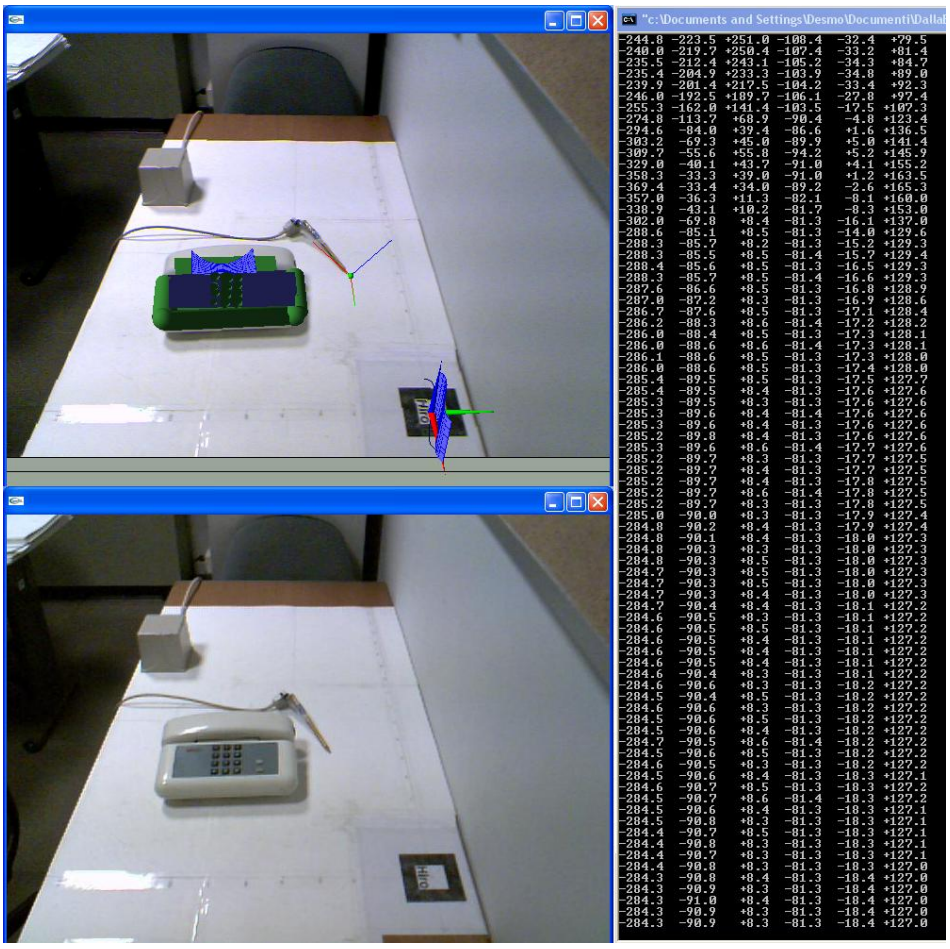


Figura 3.2.26. Confronto tra oggetto reale e oggetto virtuale.

Il modello è stato completato con l'aggiunta dei tasti modellati tramite estrusione di poligoni. In Figura 3.2.26 si riporta il confronto tra la scena reale e la scena aumentata, con la finestra 'dos' che visualizza la posizione del puntatore durante l'acquisizione.

Un altro esempio di modellazione è la riproduzione della scatola di cioccolatini riportata in Figura 3.2.27.

Figura 3.2.27. Oggetto da riprodurre.

In questo caso si è partiti dalla definizione di un piano di schizzo passante per la superficie superiore dell'oggetto in modo da trovare un facile riferimento nel piano d'appoggio (v. Figura 3.2.28).

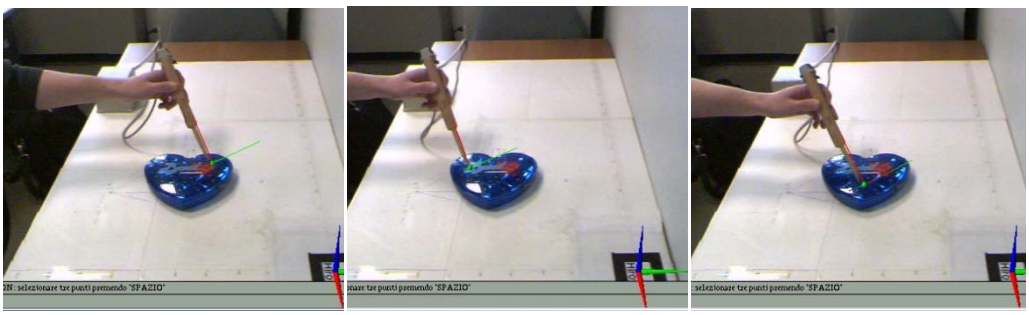


Figura 3.2.28. Selezione dei punti che definiscono il piano di schizzo.

La fase successiva ha riguardato la modellazione del profilo laterale dell'oggetto con le funzioni implementate per la creazione delle curve parametriche e la conseguente estrusione per ottenere la superficie di contorno.

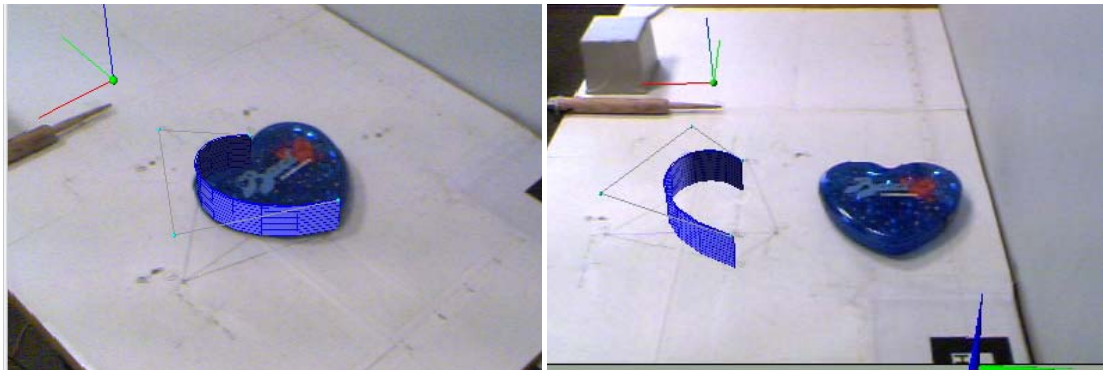


Figura 3.2.29. Modellazione delle superfici di contorno mediante estrusione di curve di Beziér.

I risultati ottenuti sono visualizzati in Figura 3.2.29. L'operazione è stata quindi ripetuta per la parte destra (Figura 3.2.30).

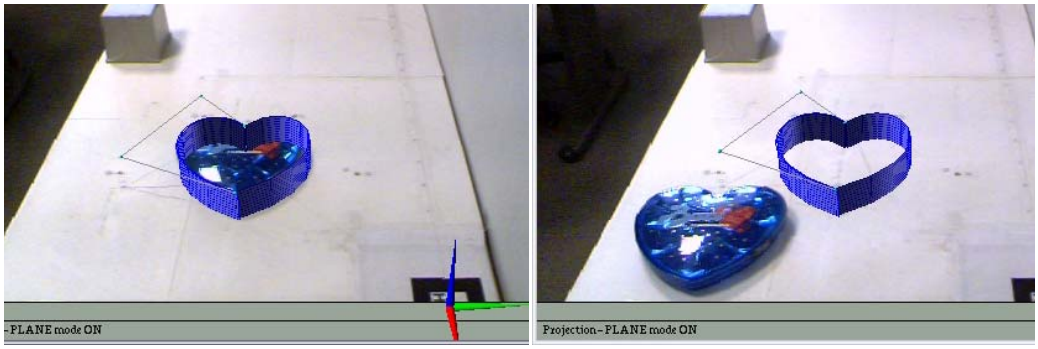


Figura 3.2.30. Completamento della fase di modellazione dei bordi.

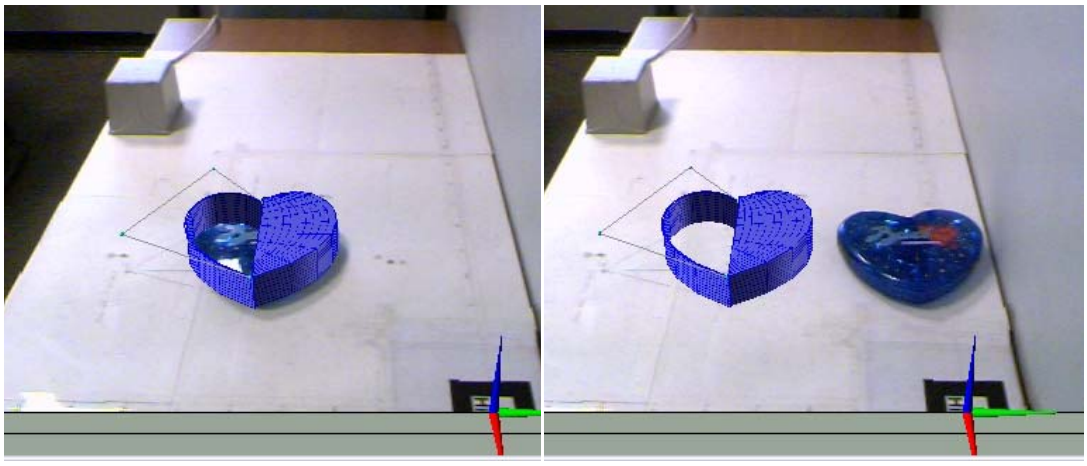


Figura 3.2.31. Modellazione della superficie superiore.

Infine La superficie superiore è stata modellata metà alla volta, in modo da sfruttare gli stessi punti di controllo precedentemente utilizzati per definire il bordo esterno, mentre

per il bordo interno è stato creato un profilo bombato che seguisse le forme della confezione.

Il risultato finale della modellazione è riportato in Figura 3.2.32, per due angolazioni diverse.

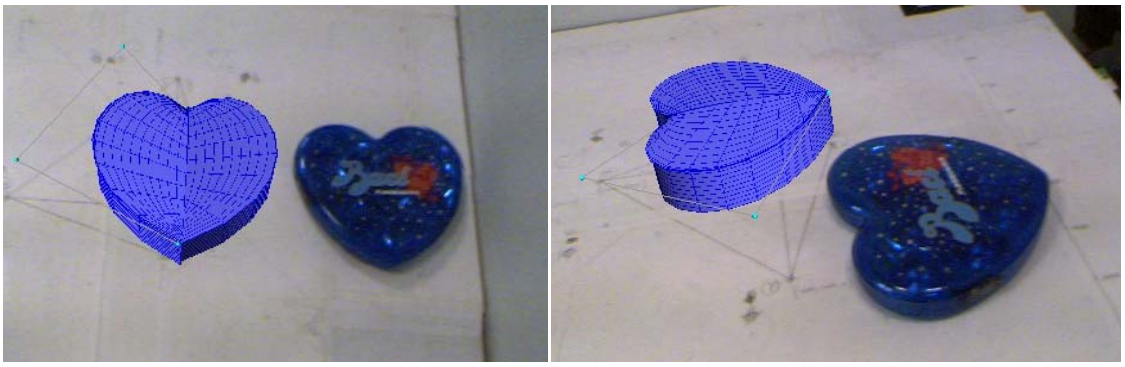


Figura 3.2.32. Confronto tra il modello virtuale e l'oggetto reale.

In riferimento allo stesso oggetto, attraverso i comandi per disegnare oggetti cilindrici, sono stati riprodotti gli elementi interni (v. Figura 3.2.33).

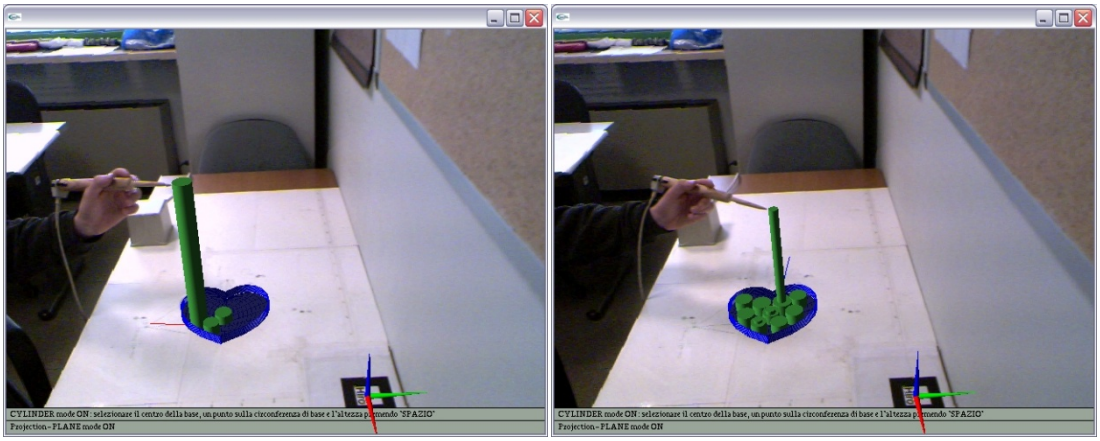


Figura 3.2.33. Modellazione delle parti interne.

L'ultimo esempio di modellazione con l'AR-CAD 2.0 presentato in questa sezione riguarda la modellazione di particolari estetici su un modellino radiocomandato, Figura 3.2.34.

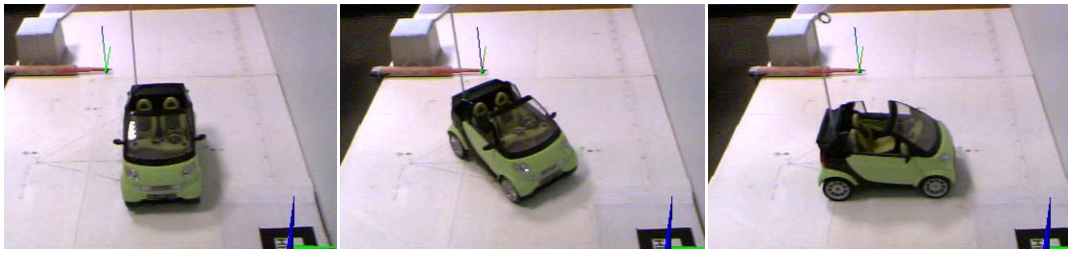


Figura 3.2.34. Modellino radiocomandato.

Il tetto è stato modellato partendo dalla definizione di un piano verticale parallelo al piano di simmetria dell'auto, passante per l'estremità destra del montante anteriore. Successivamente è stato disegnato il profilo del tetto con una curva parametrica (Figura 3.2.35).

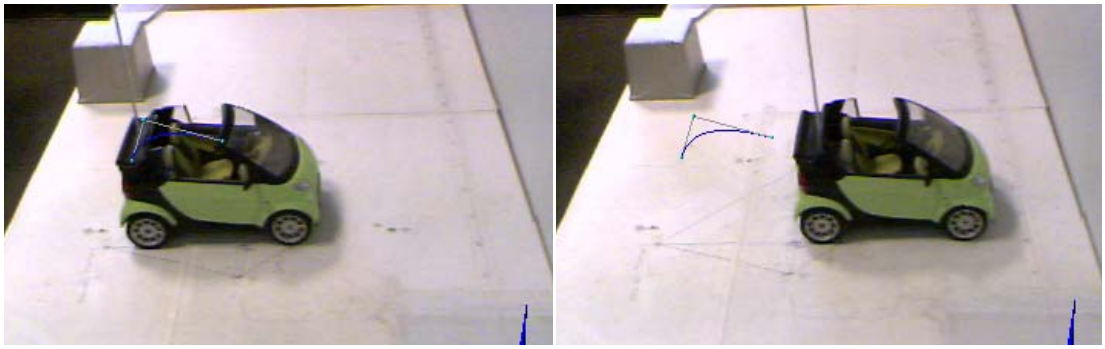


Figura 3.2.35.Profilo del tetto.

Infine, attraverso il comando di estrusione di superficie è stato modellato il tetto.

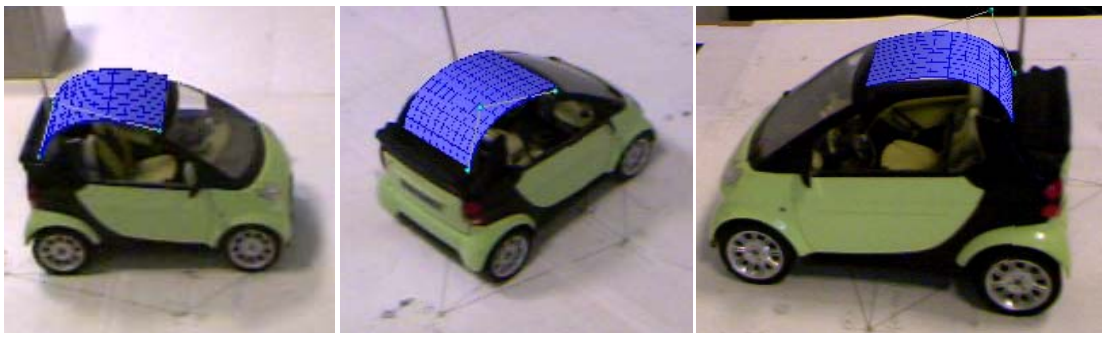


Figura 3.2.36. Risultato finale osservato da diverse angolazioni.

3.2.8 Conclusioni e sviluppi futuri

Il sistema sviluppato nasce dall'esigenza di una interfaccia più evoluta di quella utilizzata nei CAD commerciali installati su configurazioni desktop e a costi minori rispetto ai CAVE virtuali. L'adozione di un puntatore magnetico e di un sistema di visualizzazione di AR, consente al progettista di entrare nello spazio di modellazione, dove le macchinose funzioni di gestione delle viste vengono sostituite da un sistema di visualizzazione dinamico, che aggiorna la visuale sulla scena coerentemente con il punto di vista dell'utente rispetto alla scena stessa. In questo modo la selezione bidimensionale lascia il posto ad una più intuitiva selezione tridimensionale.

Sebbene l'interfaccia risulti innovativa può essere migliorata soprattutto nelle funzionalità di attivazione e gestione comandi. L'utilizzo della tastiera del computer per l'attivazione dei comandi è una scelta comoda in fase di sviluppo ma limita l'utente nelle usuali operazioni di modellazione. Attingendo ancora una volta alla tecnologia della Realtà Virtuale, si potrebbero utilizzare guanti virtuali per l'attivazione e la gestione delle funzioni di modellazione.

D'altra parte un aspetto importante da considerare nella valutazione del sistema riguarda la stima dell'errore commesso nella fase di selezione.

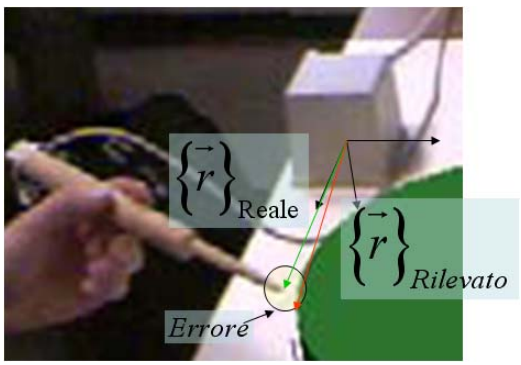


Figura 3.2.37. Stima dell'errore nel processo di selezione.

Nella pratica tre fattori condizionano la precisione del sistema:

- I movimenti incontrollati dell'utente;
- I disturbi del campo elettromagnetico generato dall'emettitore;
- L'errore di misura intrinseco del sistema;

L'ultimo contributo sopraelencato, può essere stimato note le prestazione del sistema. Per il dispositivo magnetico utilizzato l'errore si attesta intorno ai 0.5 mm per la posizione e intorno ad 1° per l'orientamento del puntatore.

Gli altri due contributi invece sono fortemente variabili in funzione dell'ambiente che circonda il sistema e dell'operatore che lo utilizza. A tal riguardo si può limitare l'errore dovuto ai movimenti incontrollati dell'operatore, considerando anche la velocità di acquisizione che si attesta intorno ai 140 fps, utilizzando bracci robotizzati o spingendo la punta del puntatore contro la superficie da acquisire.

Il limite maggiore del sistema, in accordo con le prove sperimentali, riguarda la sensibilità alle interferenze magnetiche causate da superfici metalliche. Da qui l'impossibilità di considerare, nello spazio di lavoro, la presenza di oggetti metallici, frequentemente utilizzati nella meccanica.

Da ultimo si dovrebbe potenziare il sistema con comandi di modellazione più evoluti e con una rappresentazione parametrica delle entità modellate. A tal fine una strategia vantaggiosa, attualmente in fase di elaborazione, è quella di importare le funzionalità del sistema di puntamento e di visualizzazione, mediante le interfacce di programmazione [65][63], all'interno di un software CAD di ultima generazione.

3.3 Reverse Engineering in Realtà Aumentata

Nel processo di ingegneria inversa (v. Appendice B) la scelta del metodo di acquisizione condiziona la precisione con la quale si riproducono virtualmente le geometrie acquisite.
Le tecniche di acquisizione si suddividono in due gruppi: quello dell'acquisizione per contatto e quello dell'acquisizione senza contatto. Le tecniche di acquisizione per contatto assicurano libertà di movimento all'interno della scena, pur offrendo uno spazio di lavoro limitato, e consentono all'operatore di selezionare i punti di maggiore interesse, ovvero i punti utili alla riproduzione dei bordi e dei profili delle superfici da rilevare. Questo fatto conduce alla semplificazione del processo perché consente di inglobare la fase di elaborazione dei punti nella fase di acquisizione. D'altra parte le tecniche senza contatto vantano una velocità di acquisizione dei punti superiore alle tecniche con contatto, ma necessitano di una fase di elaborazione dati più onerosa.

In questo paragrafo viene presentato un metodo di acquisizione ibrido, in grado di coniugare la libertà di selezione del metodo per contatto con la semplicità di acquisizione di un sistema senza contatto.

Il sistema in oggetto è stato valutato in termini di precisione e sperimentato su un caso realistico riguardante la riproduzione virtuale del cupolino e della carenatura anteriore di una *Ducati Multistrada 620.* Il modello virtuale riprodotto è stato utilizzato per analisi fluidodinamiche comparative.

3.3.1 Il metodo di acquisizione

Il metodo di acquisizione implementato si basa sul riconoscimento della posizione relativa tra due marker, di cui uno è considerato il sistema di riferimento fisso e l'altro, manovrato dall'operatore, il puntatore (Figura 3.3.1).
Utilizzando le librerie Artoolkit è possibile risalire alle trasformazioni geometriche che caratterizzano la posizione relativa di più marker rispetto alla camera che riprende la scena (v. § 2.5.4).

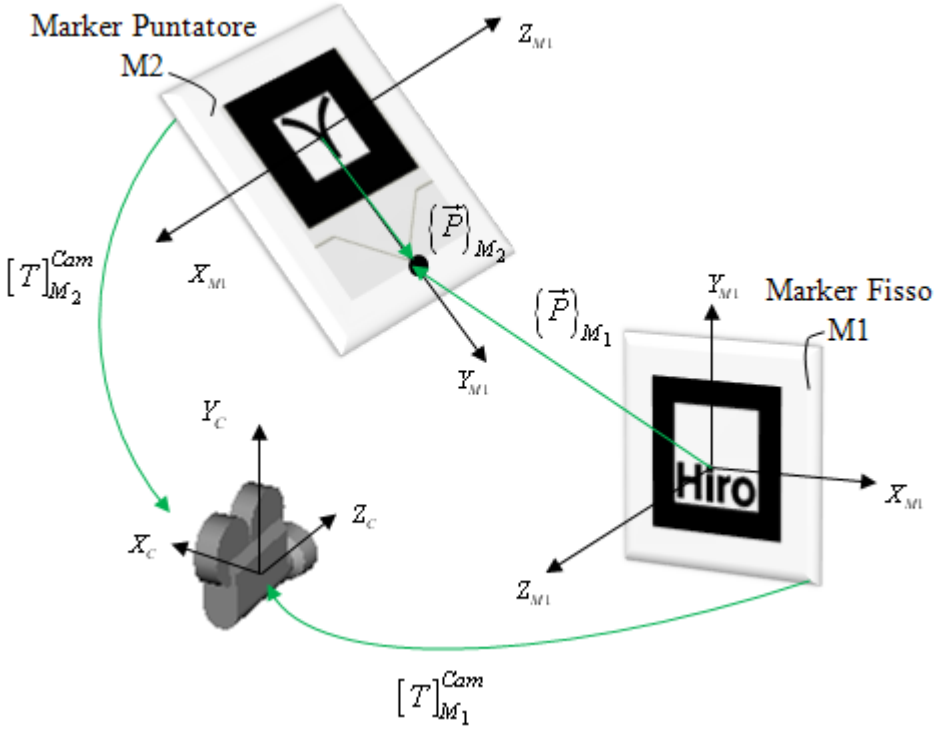


Figura 3.3.1. Sistemi di riferimento e trasformazioni geometriche camera-marker1 e camera-marker2.

Esprimendo le trasformazioni in oggetto con le matrici $[T]_{M_1}^{Cam}$ e $[T]_{M_2}^{Cam}$ e assumendo che $\{P\}_{M_2} = \{0 \quad d \quad 0\}$ sia il vettore posizione espresso nel sistema di riferimento solidale al marker M_2, è possibile calcolare la posizione relativa del marker M_2 rispetto al marker M_1 con la seguente espressione:

$$\{P\}_{M_1} = [T]_{Cam}^{M1} \cdot [T]_{M_2}^{Cam} \cdot \{P\}_{M_2} = [T]_{M_2}^{M_1} \cdot \{P\}_{M_2} \tag{3.5}$$

Mentre le corrispondenti istruzioni da inserire nel codice del programma sono[19]:

```
arUtilMatInv(T_M1_Cam, Inv_T_M1_Cam);
arUtilMatMul(Inv_T_M1_Cam, T_M2_Cam, T_M2_M1);
```

[19] La corrispondenza tra le matrici indicate nel codice e quelle espresse nella (3.5) è del tipo $[T]_i^j = \text{T\_i\_j}$.

3.3.2 Scelta dei marker e stima della precisione

Oltre alla velocità di elaborazione dati, un aspetto fondamentale da considerare nell'implementazione del sistema di acquisizione riguarda la precisione assicurata dai dispositivi adottati.

La ripetibilità della misura dei metodi di rilievo per elaborazione dell'immagine è intrinsecamente più debole di quelli per contatto. Inoltre, diversi fattori influenzano la misura, dalla luminosità della scena alla dimensione dei marker, dalla distanza della camera alla relativa messa a fuoco.

Sia nella documentazione delle Artoolkit [5] che in letteratura [6][40] vengono forniti dati sperimentali sulla precisione assicurata dalle Artoolkit nel calcolo della posizione relativa camera-marker, per configurazioni diverse di inclinazione e posizione dei marker rispetto alle camere. Tali studi evidenziano alcune caratteristiche del metodo implementato dalle artoolkit:

- Sensibilità alle dimensioni del marker;
- Sensibilità all'illuminazione;

Il [40] mette in risalto la relazione dimensione del marker-distanza. Per un marker quadrato della dimensione di 80 mm determina un errore sulla posizione di 2-3 mm, per distanze inferiori a 300 mm, ed un errore d'inclinazione inferiore a 1°. Il [6] utilizzando un marker decisamente più grande (200 mm per lato) rileva errori più contenuti a parità di distanza ed errori uguali per distanze maggiori.

La sensibilità all'illuminazione della scena è un fatto intrinseco dell'algoritmo di elaborazione d'immagine implementato nelle Artoolkit (v. Capitolo 2). Di fatto le Artoolkit processano il singolo fotogramma con un filtro a soglia. Infatti, l'istruzione utilizzata per rilevare i marker (v. §2.5.4):

```
arDetectMarker(dataPtr,thresh,&marker_info,&marker_num)
```

richiede come secondo argomento di input proprio il valore costante per discriminare, nel singolo fotogramma, i pixel luminosi da quelli scuri (v. Figura 3.3.2).

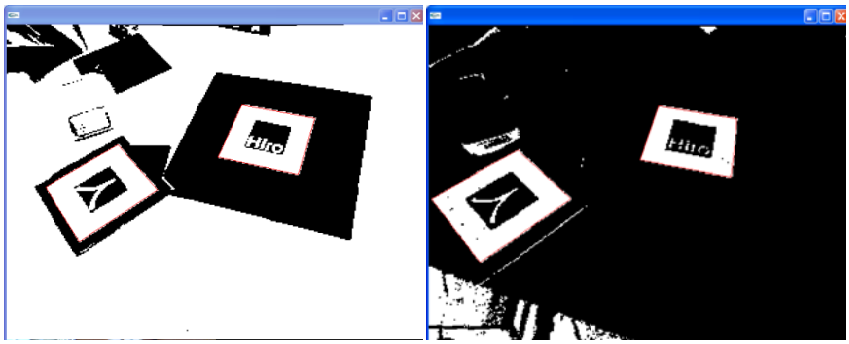


Figura 3.3.2. Elaborazione dell'immagine con impostazioni diverse: valore di soglia 200 (a sinistra) e valore di soglia 70 (a destra).

Sebbene gli studi sperimentali [6] siano accurati e utili per comprendere i fattori che influenzano maggiormente il calcolo della posizione relativa camera marker, l'impossibilità di risalire alle stesse condizioni ambientali ha spinto verso lo sviluppo e la valutazione di un proprio apparato di misura.

A tal fine sono state condotte analisi volte a valutare la configurazione ottimale per il numero e la dimensione dei marker. Infine si è proceduto alla stima del valore di soglia ottimale per rilevare la posizione dei marker nei fotogrammi elaborati.

Dapprima sono stati considerati, per il dispositivo puntatore l'impiego di 1, 2 e 3 marker (v. Figura 3.3.3).

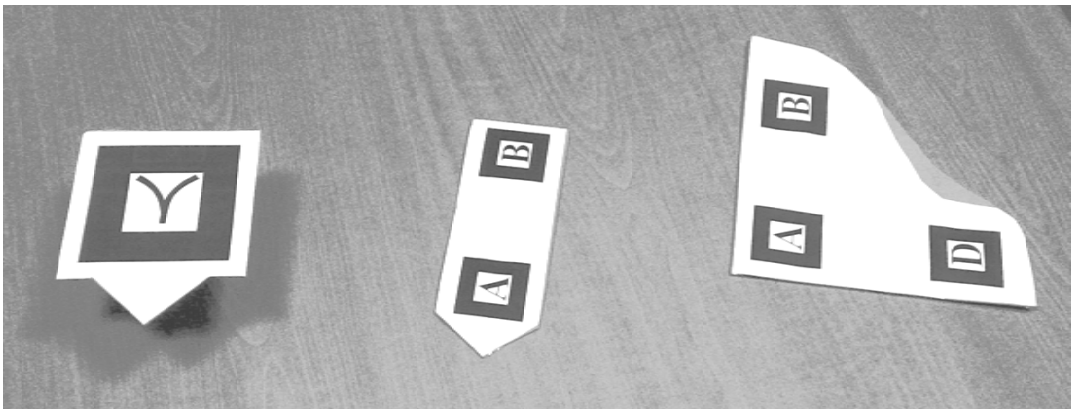


Figura 3.3.3. I puntatori impiegati nella sperimentazione: con un solo marker di dimensioni 80mm, con due marker di dimensioni 40 mm e con tre marker di dimensioni 40 mm.

Dai risultati delle prove sperimentali sulla precisione e sulla ripetibilità della misura, si è arrivati alla conclusione che il marker unico di dimensioni maggiori è quello che assicura la precisione migliore. Inoltre considerando che il fattore visibilità, per quanto si è detto, è un fattore determinante si è deciso di utilizzare per il marker fisso una sagoma di dimensioni 170 x 170 mm.

La fase successiva alla scelta dei marker ha riguardato la valutazione della precisione del sistema adottato. A tal proposito sono state rilevate le misure calcolate secondo la (3.5) di un insieme di punti selezionati con la punta del marker puntatore, per diverse inclinazioni rispetto al piano del marker di riferimento. La webcam è stata posizionata ortogonalmente al marker del sistema di riferimento fisso ad una distanza leggermente superiore a quella massima testa operatore-mano, riproducendo così la condizione ideale di funzionamento del sistema.

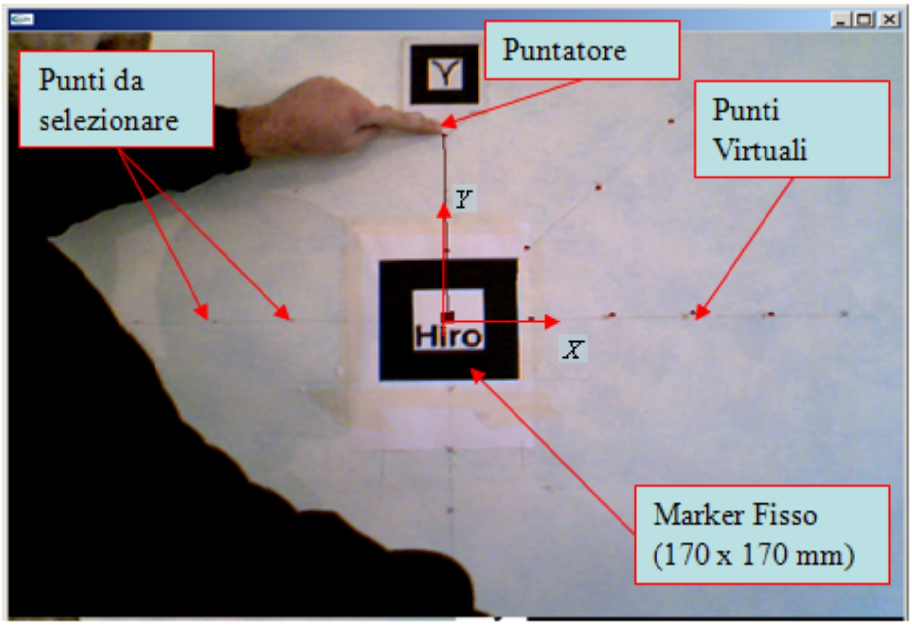


Figura 3.3.4. Procedura sperimentale per la stima dell'errore.

Tabella 3.3.1. Posizione dei punti di misura rispetto al marker.

Punto n°	x	Y	z	
1	90	0	0	Asse 0°
2	180	0	0	
3	270	0	0	
4	360	0	0	
5	90	90	0	Asse 45°
6	174	174	0	
7	258	258	0	
8	0	90	0	Asse 90°
9	0	170	0	
10	0	250	0	
11	-90	90	0	Asse 135°
12	-174	174	0	

13	-258	258	0	
14	-90	0	0	Asse 180°
15	-180	0	0	
16	-270	0	0	
17	-360	0	0	
18	0	-90	0	Asse 270°
19	0	-170	0	
20	0	-250	0	

Sono stati segnati 20 punti da rilevare nell'intorno del marker. In Tabella 3.3.1 si riportano le coordinate cartesiane dei suddetti punti nel riferimento *X-Y* associato al marker (v. Figura 3.3.4).

Come anticipato i punti sono stati rilevati per inclinazioni diverse del marker puntatore rispetto al piano del marker fisso. Si è proceduto quindi con tre campagne di misura, con tre puntatori diversi (v. Figura 3.3.5) aventi inclinazioni di 0°, 30° e 45°.

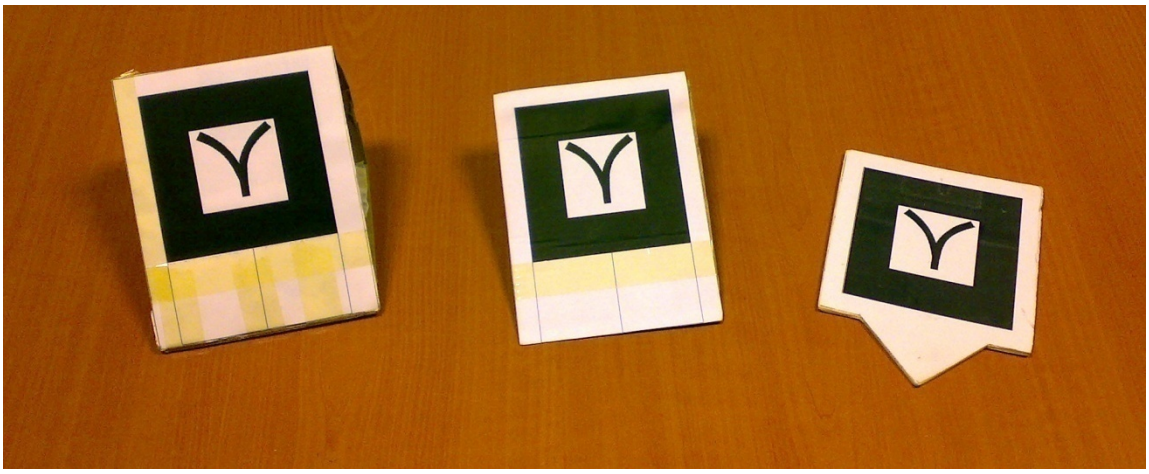


Figura 3.3.5. I marker impiegati nella sperimentazione del sistema: inclinato di 45° (a sinistra), inclinato di 30° al centro e inclinato di 0° a destra.

I risultati ottenuti dalla prima prova sono riportati in Tabella 3.3.2 dove vengono evidenziati in rosso i valori che più si discostano dai dati ideali di Tabella 3.3.1.

Tabella 3.3.2. Risultati sperimentali della prima prova.

0°			30°			45°		
x	y	z	x	Y	z	X	Y	z
94,85	-0,93	8,32	93,38	1,27	5,56	94,83	-2,21	-12,49
187,38	-1,63	3,16	183,81	0,09	12,79	185,79	-0,13	2,15

280,99	-3,01	-15,9	274,41	-0,94	8,28	276,29	-0,7	4,88
384,09	-0,16	-44,93	356,11	-0,32	16,95	365,52	-0,91	7,43
95,89	90,21	4,85	93,89	88	29,06	92,22	91,16	11,51
187,98	177,95	-26,45	179,6	170,29	15,32	179,25	171,61	5,92
282,64	267,65	-41,26	258,9	248,31	26,08	262,54	250,01	-0,61
0,79	90,2	12,6	-0,93	85,13	42,91	-0,12	88,66	30,39
-0,68	172,84	-2,77	-0,9	163,45	20,35	-0,76	166,47	33,8
-3,66	250,83	-71,15	0,1	238,96	14,78	2,73	234,47	45,88
-96,21	88,03	5,35	-90,34	88,36	28,83	-92,24	91,22	24,46
-186,3	177,61	-18,14	-172,7	168,34	31,91	-180,09	170,43	20,1
-280,55	268,28	-52,12	-273,81	259,51	-13,54	-268,36	251,68	6,02
-101,16	0,53	-33,32	-97,39	-0,51	-17,71	-100,31	-2,08	-29,41
-188,13	-1,21	1,23	-194,49	-1,45	-27,47	-189,49	-1,4	-0,66
-290,96	1,95	-34,31	-287,67	-0,62	-28,17	-290,55	-3,61	-40,95
-380,35	-0,51	-26,31	-384,95	-0,11	-42,65	-376,84	-2,23	-20,93
0,95	-89,72	12,71	5,14	-85,84	43,87	0,13	-92,48	5,38
-1,17	-178,29	-31,31	-1,3	-163,58	36,87	-3,24	-168,45	15,56
-0,26	-238,41	22,25	-0,28	-241,39	7,4	-2,25	-240,02	39,83

L’andamento dell’errore per i diversi punti, organizzati secondo l’ordine della Tabella 3.3.1, sono riportati nei grafici di Figura 3.3.6.

Il primo grafico visualizza l’andamento dell’errore, misurato algebricamente, nello spazio mentre il secondo riporta l’andamento dell’errore nel piano. I risultati così ottenuti, incompatibili con le esigenze di misura, hanno stimolato verso una ulteriore fase di analisi ed ottimizzazione del sistema.

Partendo dai dati sperimentali della prima misurazione si osserva come i risultati ottenuti siano affetti da una cattiva valutazione della profondità dei marker rispetto alla camera, infatti il grafico in basso di Figura 3.3.6 evidenzia come valutando l’errore nel piano si ottenga un andamento più regolare: per i punti più distanti dal marker (4, 7, 13, 17, 20) si ottiene un errore maggiore, che invece diminuisce costantemente passando dai punti più lontani a quelli più vicini. Inoltre gli errori rilevati per ciascun punto risultano mediamente inferiori con marker puntatore inclinato rispetto al caso di marker giacente nel piano *XY*.

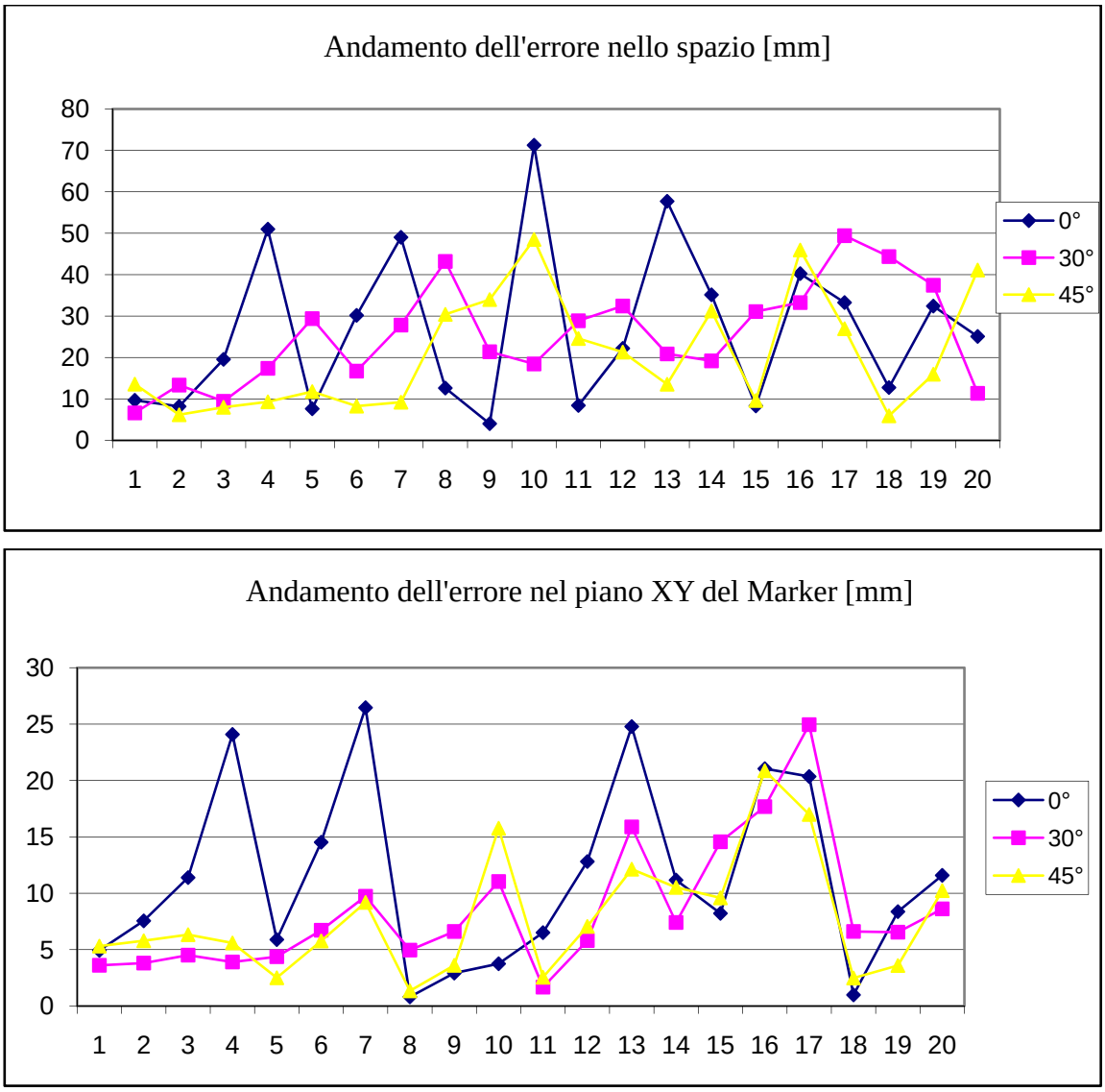


Figura 3.3.6. Risultati della prova sperimentale.

Un'altra anomalia rilevata, risultata poi determinate per la fase di ottimizzazione, riguarda l' asimmetria dell'errore rispetto al piano *ZY* del marker di riferimento. Mentre l'errore sulla misura dei punti 8, 9 e 10 , giacenti sull'asse *Y*, è allineata con i rispettivi 18, 19 e 20 giacenti sull'asse *–Y*, il rilievo dei punti appartenenti all'asse *X* ha fornito valori discordanti rispetto ai rispettivi punti appartenenti all'asse *–X*.

La prima ottimizzazione eseguita sul sistema ha riguardato la disposizione delle luci. Data l'asimmetria dell'errore rilevato rispetto al piano destro e sinistro del marker e osservando l'illuminazione, anch'essa asimmetrica sulla scena, sono state modificate le condizioni di luminosità ed eliminate le eventuali ombre sia degli oggetti presenti nella scena sia dell'operatore durante la fase di acquisizione. In accordo con le indicazioni trovate in letteratura [57], riscontrate misurando in condizioni diverse di illuminazione lo

stesso punto, si è arrivati alla condizione ottimale di luminosità diffusa in grado di eliminare l'asimmetria sulla misura.

Il secondo passo ha riguardato l'impostazione del valore di soglia nella fase di elaborazione. A tal fine è stato modificato il programma sviluppato per l'acquisizione aggiungendo altre due finestre (v. Figura 3.3.7) e un cubo virtuale per guidare l'operatore nella fase di registrazione del sistema.

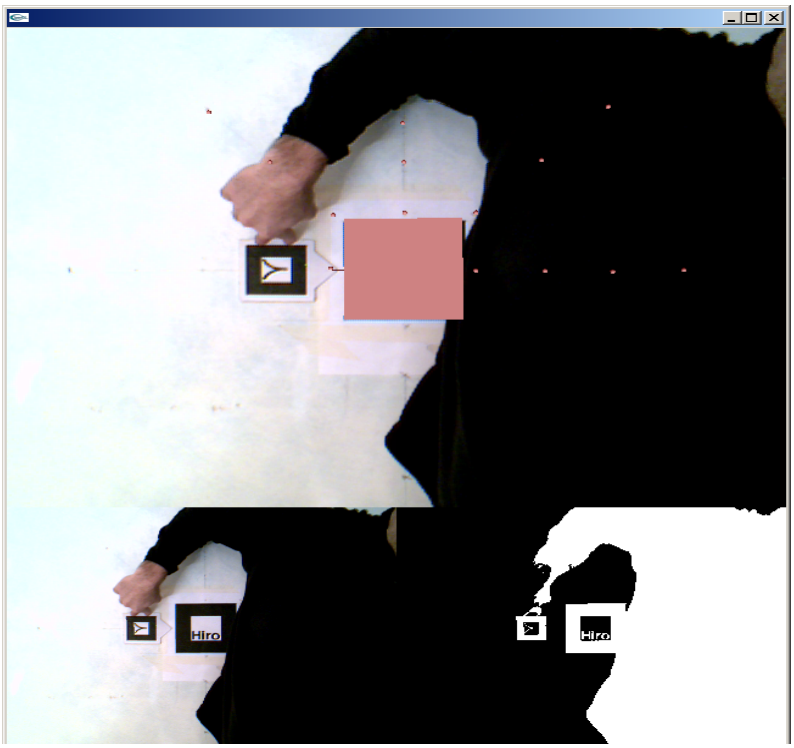


Figura 3.3.7. Ottimizzazione del sistema di acquisizione, variazioni sull'illuminazione e sul programma per la registrare la scena.

In Figura 3.3.7 si riporta una schermata dell'applicazione, in alto viene proiettata la scena aumentata, nelle finestre in basso il fotogramma da processare (a sinistra) e quello processato (a destra). La scena aumentata proietta i punti acquisiti come sferette virtuali, mentre il cubo sul marker fisso consente all'operatore di valutare il corretto allineamento del punto di vista della webcam con l'ambiente virtuale delle OpenGL. Inoltre la funzione associata agli eventi da tastiera è stata fornita della funzionalità per la modifica del valore di soglia con cui elaborare l'immagine. In questo modo l'utente è in grado di valutare e cambiare in maniera dinamica il valore di soglia dell'applicazione, valutando di volta in volta gli effetti dei valori inseriti sull'estrazione dei marker. Tanto più i marker vengono rilevati nitidamente e tanto più il valore inserito è vicino a quello ideale. Successivamente alla fase di ottimizzazione, si è proseguito con la fase di

sperimentazione del sistema. Dopo diversi tentavi, volti a trovare i giusti valori del valore di soglia, si è riscontrato un notevole miglioramento sulla precisione di acquisizione. I risultati migliori sono riportati in Figura 3.3.8 e evidenziano un errore contenuto entro i 2 mm nei punti più vicini al marker di riferimento e con il marker puntatore inclinato rispetto al piano del marker fisso.

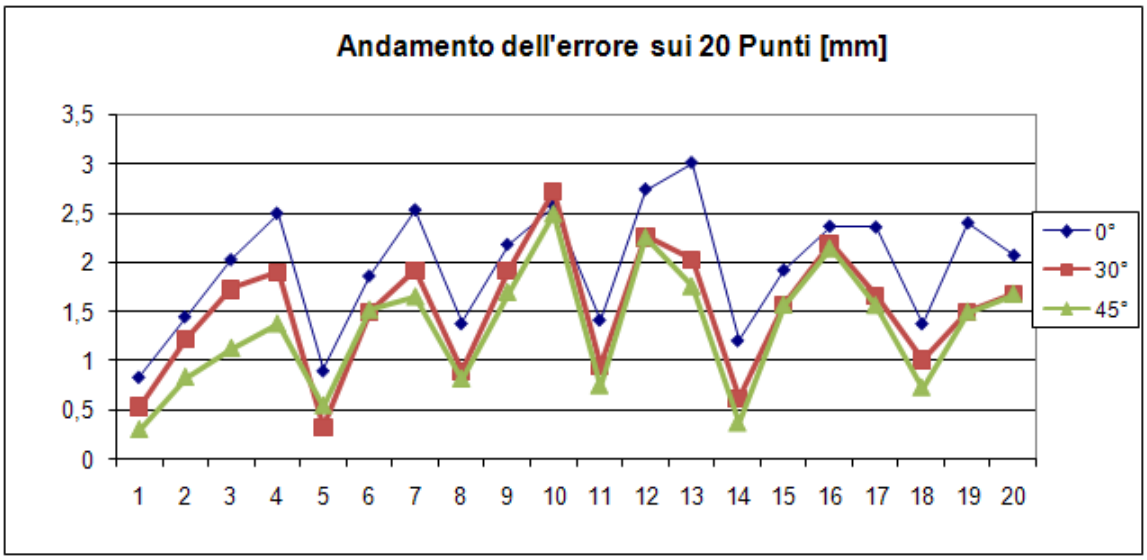


Figura 3.3.8. Andamento dell'errore nello spazio dopo l'ottimizzazione.

Ulteriori ottimizzazioni possono essere apportate al sistema, sia migliorando l'ambiente di acquisizione sia utilizzando attrezzature più sofisticate. Tuttavia una precisione sul rilievo di 2 mm risulta accettabile, in prima istanza, con gli obiettivi delle applicazioni sviluppate in questo lavoro.

La caratteristica principale di questo sistema è quello di fornire all'ingegnere progettista un sistema di acquisizione facile da utilizzare, economico e mirato a velocizzare la fase di elaborazione dati nel processo di duplicazione virtuale dell'oggetto acquisito.

3.3.3 Reverse Engineering per simulazioni fluidodinamiche

In campo motociclistico uno dei vincoli progettuali riguarda il comfort aerodinamico assicurato dalle sovrastrutture del motoveicolo. Maggiore è la velocità e maggiore è la necessità di avere una protezione aerodinamica adeguata per garantire il giusto benessere di guida al conducente.

Figura 3.3.9. Moto a confronto: protezioni aerodinamiche differenti tra granturismo (sinistra) e *naked* destra.

Per contro la scelta di una carrozzeria ad elevata protezione aerodinamica fa spesso lievitare gli ingombri della moto, generando effetti estetici negativi. In Figura 3.3.9 si confrontano due tipi di moto agli antipodi per ciò che riguarda la protezione aerodinamica, nella parte sinistra una moto da granturismo è dotata di una carenatura avvolgente, che protegge efficacemente il pilota, ma che appesantisce la linea della moto stessa. Al contrario la moto riportata sulla destra è più snella, ma sicuramente meno confortevole dal punto di vista aerodinamico.

Figura 3.3.10. Sovrastrutture a confronto: parabrezza originale (sinistra) e parabrezza modificato (a destra) per la Ducati Multistrada 620.

In tale ambito è evidente che uno studio volto ad ottimizzare gli ingombri e massimizzare l'efficacia aerodinamica, delle sovrastrutture di una moto, risulti vantaggioso per la competitività del prodotto finale. Recentemente le case produttrici di componenti after-market hanno messo in commercio parabrezza in grado di migliorare l'efficacia aerodinamica della moto a fronte di una riduzione degli ingombri. Sebbene la sperimentazione di questi apparati risulti efficace per la loro messa a punto, una simulazione virtuale del comportamento aerodinamico di queste strutture è auspicabile per valutarne l'efficacia e per guidare la sperimentazione ed ottimizzazione dei componenti stessi.

In Figura 3.3.10 si riportano le foto del modello oggetto dello studio fluidodinamico e del parabrezza installato sul cupolino originale.

La possibilità di selezionare solo i punti di maggiore interesse e la necessità di ripetere più volte il processo di misurazione ha reso necessaria una fase di preparazione, volta a determinare e contrassegnare i punti da acquisire. A tal fine sono stati utilizzati un pennarello e del nastro carta adesivo (v. Figura 3.3.11).

Figura 3.3.11. Immagini della fase di preparazione: applicazione del nastro adesivo (in alto a sinistra), selezione dei punti appartenenti ai bordi (in alto a destra), risultato finale (in basso).

Il criterio con il quale è stato applicato il nastro e sono stati segnati i punti è dipeso dalla scelta preliminare delle tecniche di modellazione utilizzate nella fase di riproduzione virtuale delle superfici esterne di carena e cupolino. Compatibilmente con le esigenze della simulazione si è scelto di modellare la carrozzeria utilizzando i comandi per la creazione di curve e superfici parametriche (*loft*, *sweep*, *patch*)[20]. A tal fine sono stati dapprima selezionati i contorni delle superfici che compongono idealmente la carrozzeria, individuandoli nelle zone limite e nelle zone con singolarità di forma, e poi sono stati selezionati i punti ritenuti fondamentali per la modellazione (v. Figura 3.3.12).

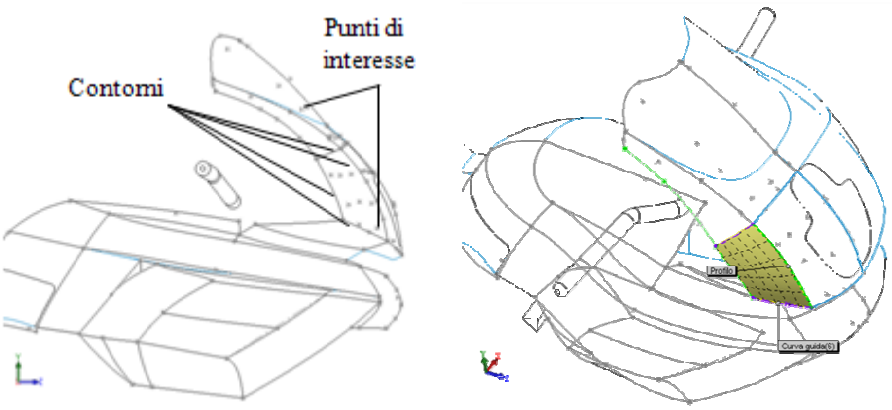


Figura 3.3.12. Dettagli della strategia di modellazione: modellazione delle curve di contorno per i punti di interesse (a sinistra), modellazione di una delle superfici con il comando *loft* (a destra).

Tra questi si distinguono i punti appartenenti alle curve di contorno delle diverse parti e i punti appartenenti alle curve intermedie, necessarie per riprodurre le convessità di forma delle superfici da acquisire (v. Figura 3.3.13). Un'ulteriore distinzione riguarda i punti intermedi e i punti finali.

La suddetta classificazione evidenzia la differenza sostanziale tra i punti finali e le curve di contorno rispetto alle curve intermedie e ai punti intermedi: mentre i primi devono essere selezionati obbligatoriamente, le seconde possono essere localizzate ed inserite in funzione delle esigenze di modellazione. A tal proposito è necessario precisare che le curve utilizzate per la riproduzione di contorni e forme sono di tipo *B-Spline* pertanto sia la scelta del numero di punti che la loro disposizione è non banale.

[20] Ulteriori dettagli sui comandi di modellazione utilizzati sono forniti nel manuale [59] della *SolidWorks*.

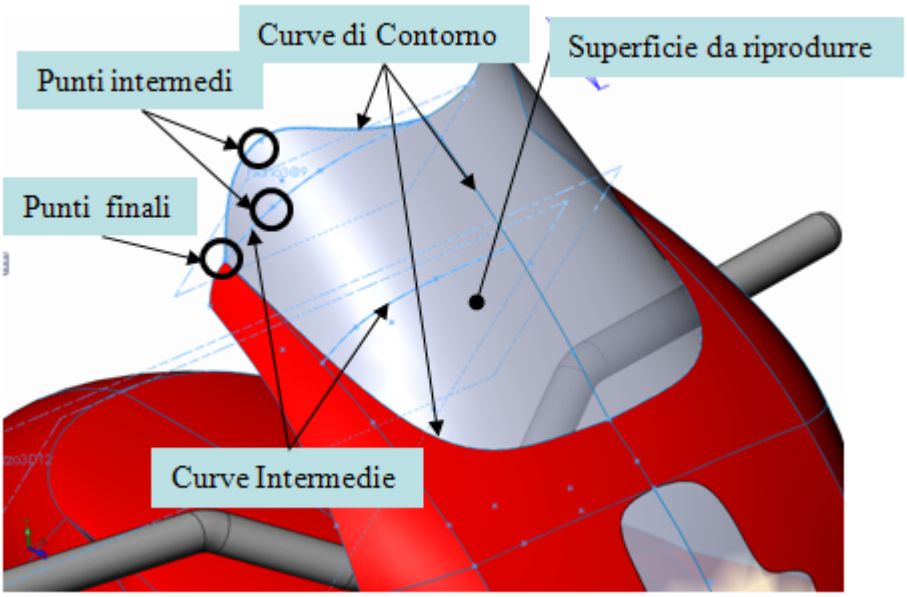


Figura 3.3.13. Classificazione dei punti acquisiti in funzione delle curve di controllo delle superfici.

In Figura 3.3.14 si confrontano gli effetti, su una curva parametrica di costruzione (v. Figura 3.3.13), della selezione di gruppi diversi di punti d'interpolazione. Si osservi come la selezione di soli due punti (Figura 3.3.14) generi una curva, la C_2^3, di forma poco attinente alla sezione del cupolino reale. Allo stesso modo con la selezione di 3 punti alla volta, curve $C_{3,i}^3$, non si riesca ad approssimare correttamente la curva di sezione da riprodurre: la curva $C_{3,1}^3$ affetta il volume di ingombro del cupolino, la curva $C_{3,2}^3$ lo ingloba sovrastimandone le dimensioni. La curva C_4^3 (evidenziata in celeste) è quella che approssima con maggiore regolarità e fedeltà la sezione di cupolino da modellare. Di fatto tale curva passa per 4 punti, ovvero il numero di punti minimo per definire una curva polinomiale di grado 3 come le *B-Spline* senza dover imporre delle condizioni fittizie.

Un altro fattore da tenere in considerazione è la disposizione dei punti da selezionare lungo le curve. Si noti dall'esempio di Figura 3.3.14 la posizione dei punti P_0, P_1, P_2 e P_3. A tal riguardo i punti P_2 e P_1 hanno distanze diverse tra loro dai punti P_0 e P_3. Infatti la scelta di avvicinare il punto P_1 al punto P_0 risponde alla necessità di avere in quella zona un tratto a curvatura maggiore (v. Figura 3.3.14 destra) rispetto al tratto finale, individuato sulla curva dai punti P_2 e P_3, che invece ha un andamento più morbido.

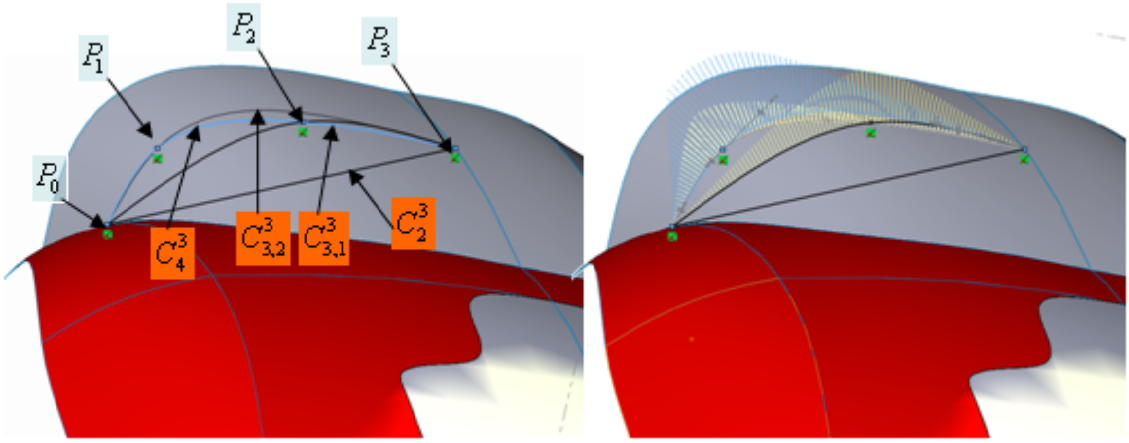


Figura 3.3.14. Gli effetti sulle curve d'interpolazione dei punti selezionati: curve per 2, 3, 4 punti (a sinistra), valutazione delle curvature (a destra).

Dopo aver marcato i punti da selezionare si è passati alla fase di registrazione del sistema. Attraverso un procedura iterativa di modifica e valutazione è stato trovato il corretto valore di soglia da utilizzare per rilevare efficacemente i marker presenti nella scena Figura 3.3.15.

Figura 3.3.15. La fasi di registrazione e di acquisizione dei punti.

Il sistema nella sua configurazione ottimale, ottenuta dopo diversi tentativi e in grado di garantire una precisione sufficiente, si è rilevata particolarmente efficace dal punto di vista operativo.

Di fatto l'impiego di marker ha consentito l'auspicata libertà e velocità di selezione e l'eliminazione della fase di elaborazione dati ha permesso di effettuare velocemente diverse campagne di misura per valutare l'affidabilità del sistema.

Il risultato finale è riportato in Figura 3.3.16, nella quale si confrontano i modelli virtuali con il cupolino originale e con quello variotouring.

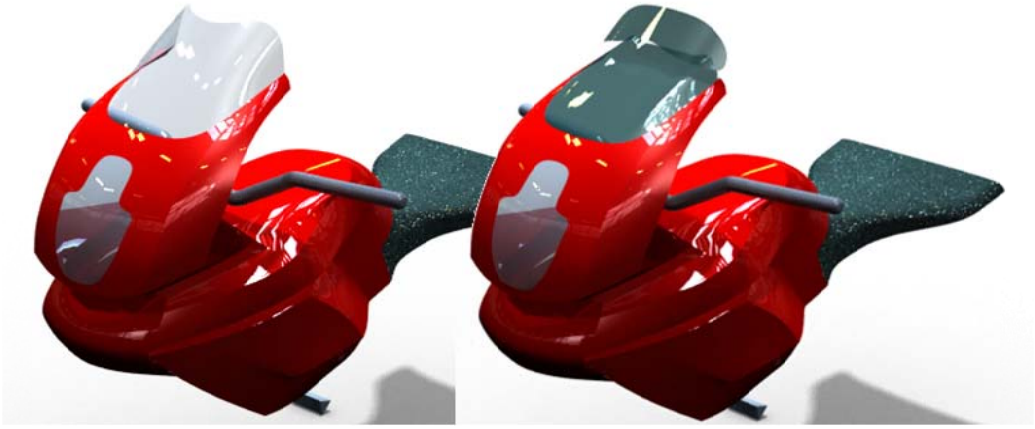


Figura 3.3.16. La fase di rendering del modello 3D: modello con cupolino originale (a sinistra) e modello con variotouring (a destra).

In Figura 3.3.17 si riportano il modello e la relativa mesh utilizzata per l'analisi fluidodinamica: alla carenatura e cupolino sono stati aggiunti il modello di un manichino antropometrico riconfigurabile e un casco, ottenuto dall'acquisizione dei punti appartenenti ad un modello reale con il sistema di acquisizione adottato per la carenatura.

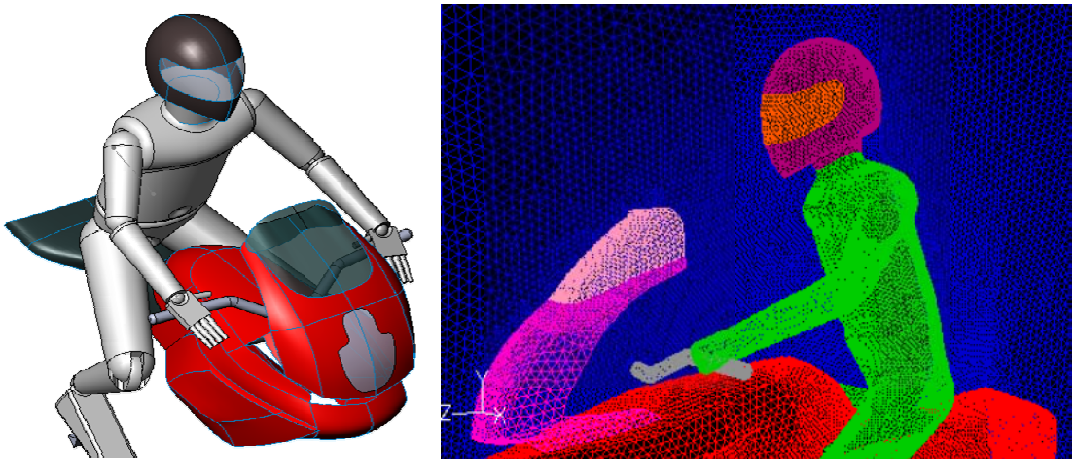
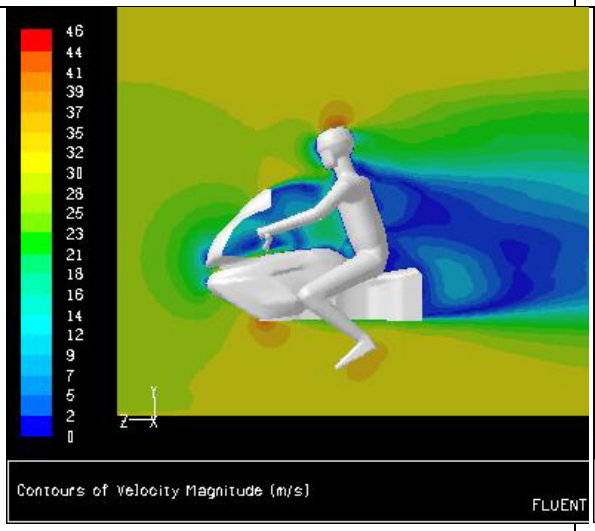
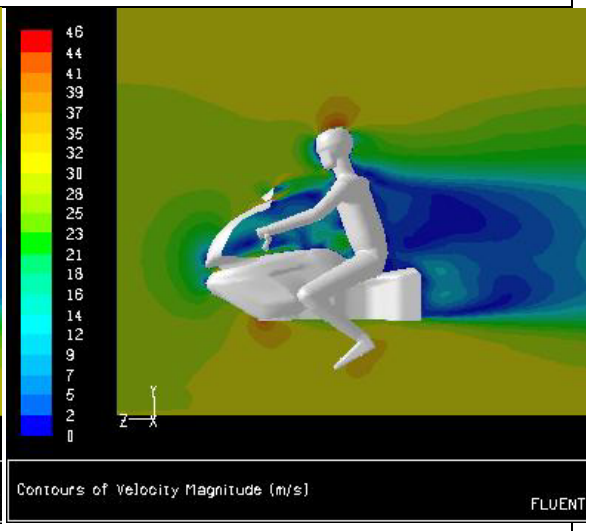
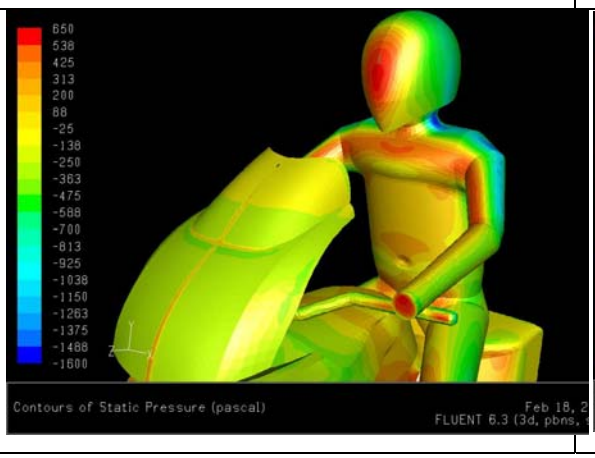
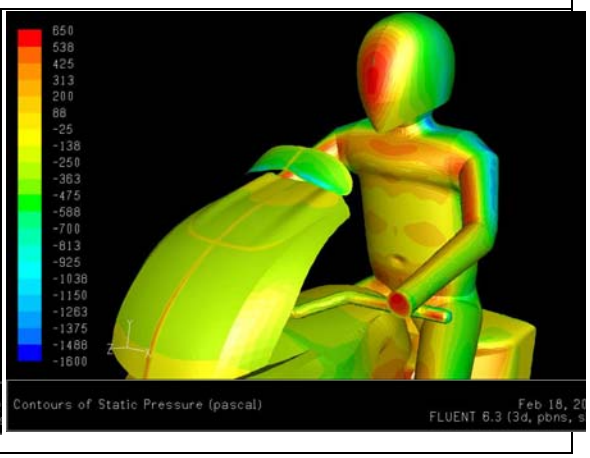


Figura 3.3.17. Il modello virtuale e mesh della simulazione fluidodinamica.

L'analisi fluidodinamica ha riguardato lo studio dei campi di moto e della pressione esercitata sul pilota per valutare e confrontare il comfort aerodinamico nelle due configurazioni, cupolino originale e cupolino variotouring.

In Tabella 3.3.3 si riportano i risultati ottenuti dalle analisi fluidodinamiche per una velocità di circa 30 m/s: nella colonna di sinistra si riportano i risultati ottenuti sul modello originale mentre nella colonna di destra i risultati ottenuti sul modello con cupolino variotouring. La prima riga riporta i diagrammi a falsi colori della velocità, mentre la seconda i diagrammi delle pressioni.

Tabella 3.3.3 Risultati delle analisi fluidodinamiche

Cupolino Originale	**Cupolino Variotouring**
	
	

3.3.4 Conclusioni e sviluppi futuri

Il sistema di ingegneria inversa sviluppato coniuga la velocità di acquisizione degli apparecchi senza contatto, con la duttilità di selezione dei metodi con contatto, a costi irrisori. Sebbene sia soggetto ai problemi fisiologici dei sistemi basati sul riconoscimento visivo di immagini, grazie all'utilizzo di un sistema di AR, fornisce all'utente gli strumenti per l'impostazione dei parametri che maggiormente condizionano l'acquisizione. Tra questi rientrano il valore di soglia, utilizzato per l'elaborazione dell'immagine, la luminosità della scena e eventuali disturbi dovuti ad occlusioni e interferenze.

Oltre ad assistere l'operatore nella fase di impostazione, l'applicazione coadiuva l'operatore anche nella fase di acquisizione. Di fatto la finestra grafica proietta informazioni virtuali, aggiornate in tempo reale, sui punti acquisiti e sul corretto allineamento del punto di vista.

Tra gli sviluppi futuri del sistema c'è l'implementazione di una procedura di calibrazione in grado di velocizzare l'impostazione delle condizioni al contorno (illuminazione della scena, visibilità dei marker, etc.) e la sostituzione della tastiera con un dispositivo interattivo per la gestione dei comandi di acquisizione.

3.4 Manutenzione in Realtà Aumenta

Come accennato nella fase introduttiva del Capitolo 1, la AR nasce come applicazione di assistenza nelle operazioni di manutenzioni di aeromobili.

In questa sezione si riporta un'applicazione di AR, sviluppata con le librerie Artoolkit a scopo dimostrativo, per lo smontaggio del componente di una stampante laser. La configurazione del sistema è quella riportata in Figura 3.4.1, con un dispositivo di tipo *HMD* installato sulla testa dell'operatore, una webcam allineata con il punto di vista dell'operatore stesso e un marker, utilizzato per la collimazione dell'ambiente virtuale con l'ambiente reale.

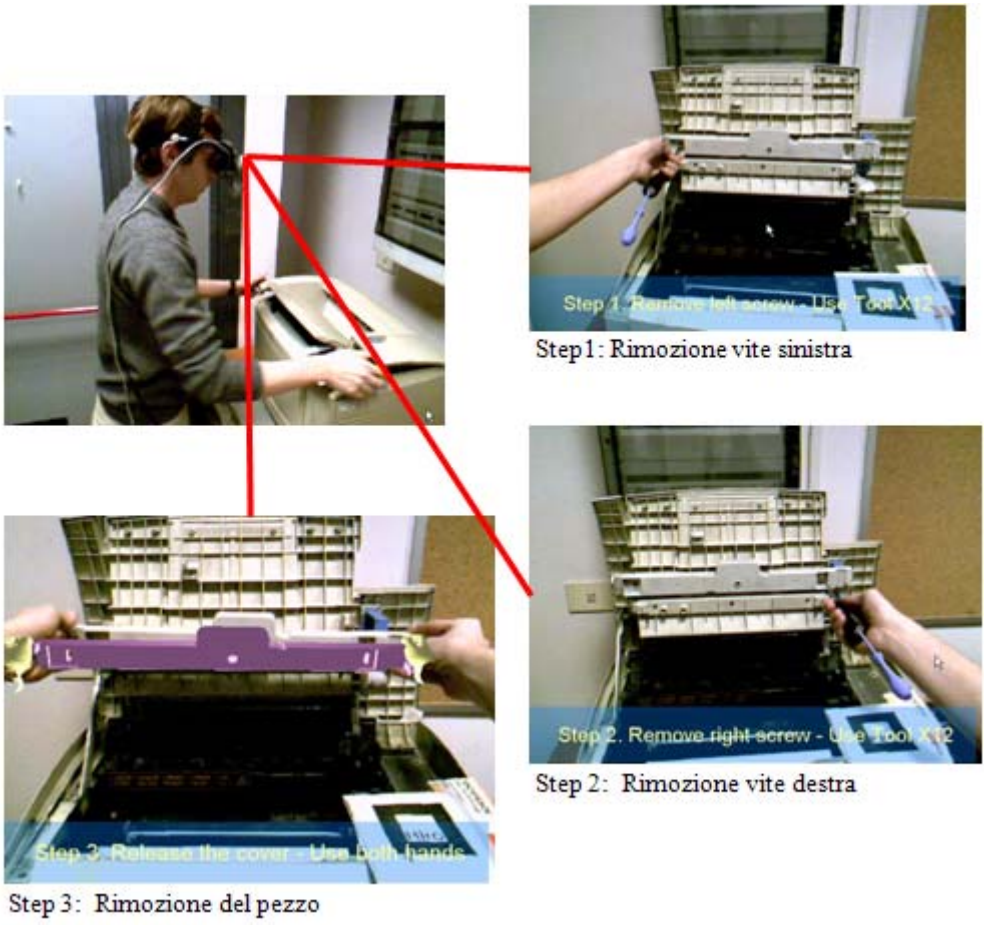


Figura 3.4.1. La sequenza delle operazioni di manutenzione.

Da un punto di vista funzionale, l'applicazione elabora i comandi vocali dell'operatore e visualizza la corrispondente operazione di montaggio o smontaggio e le informazioni relative all'utensile da adoperare (v. Figura 3.4.1 *Step1, Step2, Step3*).

La strategia da adottare nello sviluppo di questi sistemi si può riassumere nei seguenti passi:

- Riproduzione virtuale dei componenti da movimentare e degli utensili nel formato *VRML*;
- Implementazione dei comandi per il caricamento e il disegno del pezzo;
- Definizione degli spostamenti, per simulare le operazioni di movimentazione dei componenti, in funzione della sequenza di operazioni di smontaggio o montaggio da eseguire sull'assieme;

I modelli in formato *VRML* dei componenti si possono ottenere per esportazione a partire dai modelli CAD, ottenuti tramite procedure di ingegneri inversa o come risultato del processo di progettazione.

La funzionalità di caricamento dei modelli, per una applicazione sviluppata con le Artoolkit, si può effettuare utilizzando le funzioni della libreria `arvrml.h`. La funzione utilizzata per caricare il file *.wrl del componente i-esimo è `arVrmlLoadFile()`, mentre la funzione utilizzata per disegnare il componente è `arVrmlDraw()`. Il posizionamento e la movimentazione dei pezzi si ottiene attivando la modalità di proiezione `glMatrixMode(GL_PROJECTION)` e specificando separatamente traslazioni e rotazioni con i comandi OpenGL dedicati (v. Appendice A).

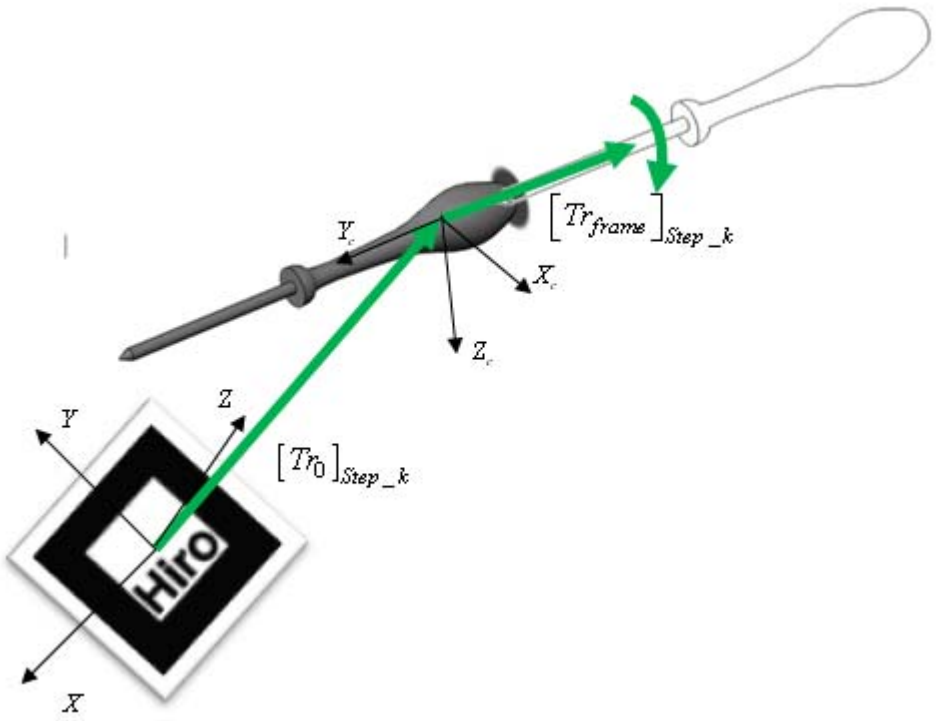


Figura 3.4.2. Animazione degli oggetti nelle operazioni di manutenzione.

Le animazioni dei componenti sono state implementate definendo posizioni diverse dei componenti nella scena rispetto al tempo. In particolare, sfruttando le proprietà delle trasformazioni OpenGL, il posizionamento del pezzo nello spazio virtuale è stato scomposto in due aliquote dove la prima, $[Tr_0]_{Step\_k}$, è costante e rappresenta la posizione del componente rispetto al marker (v. Figura 3.4.2), mentre la seconda $[Tr_{frame}]_{Step\_k}$, dipende dal tempo e quindi dal fotogramma visualizzato. In Figura 3.4.2 si riporta l'esempio del cacciavite, movimentato al *k-esimo* step mediante traslazione, per il posizionamento e l'animazione, e rototraslazione lungo l'asse Y_c per l'animazione.
In funzione del posizionamento iniziale e della movimentazione dei componenti stessi si possono utilizzare i comandi semplificati `glTranslatef()` e `glRotatef()` oppure caricare la matrice che definisce la trasformazione *Tr* con la funzione `glMatrixd()`.
Infine le informazioni circa le operazioni di manutenzione, lo step di manutenzione attivo e l'utensile da utilizzare si possono proiettare mediante un riquadro virtuale (v. Figura 3.4.1).

Tabella 3.4.1. Funzione per la proiezione di scritte virtuali.

```
void print_string( char *string )
{
int     i;
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
/* Posizionamento del riquadro */
glTranslatef(-0.95, -0.20, 0.0);

/* Disegno del riquadro */
glColor3f(1.0, 1.0, 1.0);
glBegin(GL_POLYGON);
glVertex2f(1.50, 0.10);
glVertex2f(1.50, -0.12);
glVertex2f(0.001, -0.12);
glVertex2f(0.001, 0.10);
glEnd();

/* Proiezione del testo string */
glColor3f(0.75, 0.0, 0.0);
glRasterPos2i(0.0, 0.0);
for (i=0; i<(int)strlen(string); i++) {
if(string[i] != '\n' ) {
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, string[i]);
}
else {
glTranslatef(0.0, -0.07, 0.0);
glRasterPos2i(0.0, 0.0);
}
}
```

```
return;
}
```

Le istruzioni per la visualizzazione di una scritta mediante le OpenGl sono contenute nella funzione `print_string` di Tabella 3.4.1. Dove dapprima vengono azzerate le trasformazioni di proiezione e inseguito viene disegnato il riquadro contente il testo. Infine, l'ultima operazione riguarda la visualizzazione con la funzione `glutBitmapCharacter()` della stringa di caratteri `string` ricevuta come variabile di input.

Un'applicazione possibile del sistema considerato consiste nella visualizzazione delle operazioni di assemblaggio per assiemi meccanici complessi a scopo didattico.

3.5 Multibody in Realtà Aumentata

Il multibody [34] è un'applicazione dell'ingegneria virtuale che si rivolge allo studio dinamico di strutture meccaniche labili e flessibili.

In questa sezione si riportano alcuni esempi di simulazioni dinamiche sviluppate in ambiente di AR. Le applicazioni riguardano la simulazione dinamica di una sfera, di un braccio robotizzato e di un manovellismo di spinta.

L'obiettivo è quello di valutare i vantaggi, in termini di interattività ed efficacia, connessi all'impiego della AR per applicazioni multibody.

3.5.1 Caduta libera di un grave in AR

La prima applicazione riportata in questa sezione riguarda la simulazione del moto di una sfera che cade da una posizione iniziale e rimbalza su un piano. La simulazione richiede due marker, di cui il primo rappresenta il sistema di riferimento assoluto e viene impiegato per simulare la presenza di un piano rigido di rimbalzo, mentre il secondo marker rappresenta il sensore di posizione applicato alla sfera virtuale e viene utilizzato per rilevare posizione e velocità di un punto *P* appartenente al marker (v. Figura 3.5.1).

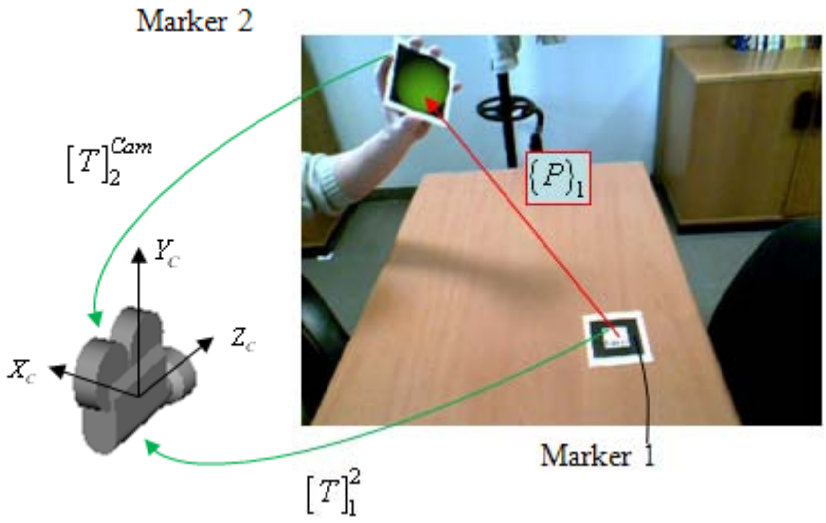


Figura 3.5.1. Descrizione dei parametri geometrici della simulazione.

La posizione del punto *P* può essere stimata attraverso le artoolkit per elaborazione dell'immagine (v. Capitolo 2). In particolare utilizzando il metodo *arGetTransMat* è possibile risalire alle matrici omogenee di trasformazione dei marker rispetto alla

camera, dalla quale si ricava la posizione relativa tra i due marker con la seguente espressione :

$$\{P\}_1 = [T]^1_{\text{camera}} \cdot [T]^{\text{camera}}_2 \cdot \{P\}_2 \tag{3.6}$$

dove:

- $\{P\}_i$ è il vettore posizione del punto *P*, espresso nel riferimento del marker *i*;
- $[T]^{\text{camera}}_i$ è la matrice di trasformazione tra il marker *i* e la camera;
- $[T]^i_{camera}$ è la matrice inversa di $[T]^{\text{camera}}_i$;

La velocità del punto *P*, è l'altro parametro cinematico che è necessario calcolare ai fini della simulazione e si può stimare con la seguente espressione:

$$\begin{aligned} \{\dot{P}\}_1 &= [T]^1_{\text{camera}} \cdot [T]^{camera}_2 \cdot \{\dot{P}\}_2 = \\ &= [T]^1_{camera} \cdot [T]^{camera}_2 \cdot \left(\frac{\{P\}_2^{frame\ \text{i}} - \{P\}_2^{frame\ \text{i-1}}}{1 / framerate} \right) \end{aligned} \tag{3.7}$$

dove:

- $\{\dot{P}\}_j$ rappresenta la velocità vettoriale del punto *P* nel sistema di riferimento del marker *j*;
- $\{P\}_j^{frame\ \text{i}}$ rappresenta il vettore posizione del punto *P*, nel riferimento del marker *j* e riferita all'*i*-esimo fotogramma elaborato;
- *framerate* rappresenta la velocità di elaborazione dei fotogrammi espressa in fotogrammi/s.

Infine per simulare la caduta libera della sfera ad un certo istante iniziale occorre inserire le condizioni cinematiche iniziali del marker 2 in termini di velocità e posizione. Poiché nel caso specifico la simulazione deve avvenire contemporaneamente alla visualizzazione occorre esplicitare le variabili di posizione della sfera, durante il moto, a partire dalle velocità e posizioni iniziali.

Le posizioni dei corpi devono essere visualizzate e quindi calcolate rispetto al tempo, secondo incrementi di tempo inversamente proporzionali alla frequenza di acquisizione.

Di conseguenza le equazioni relative alle velocità e posizione, riferite al generico *frame* i, si possono calcolare secondo le seguenti integrazioni:

$$\begin{Bmatrix} v_x \\ v_y \\ v_z \end{Bmatrix}^{\text{frame } i} = \begin{Bmatrix} v_x \\ v_y \\ v_z - g \cdot \Delta t \end{Bmatrix}^{\text{frame } i-1} \tag{3.8}$$

$$\begin{Bmatrix} p_x \\ p_y \\ p_z \end{Bmatrix}^{\text{frame } i} = \begin{Bmatrix} p_x + v_x \cdot \Delta t \\ p_y + v_y \cdot \Delta t \\ p_z + v_z \cdot \Delta t - g \cdot \Delta t^2 \big/ 2 \end{Bmatrix}^{\text{frame } i-1} \tag{3.9}$$

Infine la condizione di contatto può essere simulata utilizzando una condizione di prossimità sulla distanza relativa tra sfera (marker 2) e piano (marker 1) del tipo:

$$f := \left|d\right|_{\text{sfera-piano}} - R_{\text{sfera}} \leq \varepsilon \quad (10)$$

dove:

- R_{sphere} rappresenta il raggio della sfera;
- $\left|d\right|_{\text{sphere-plane}}$ rappresenta il modulo della distanza tra sfera e piano;
- ε è un fattore di tolleranza;

quando si verifica la condizione $f \leq \varepsilon$ il sistema rileva l'impatto ed utilizza la (3.11), di seguito riportata, per calcolare di nuovo le condizioni iniziali di moto:

$$\{v\}^{\text{frame } i} \cdot \{n\}_{\text{piano}} = -e \cdot \{v\}^{\text{frame } i-1} \cdot \{n\}_{\text{piano}} \tag{3.11}$$

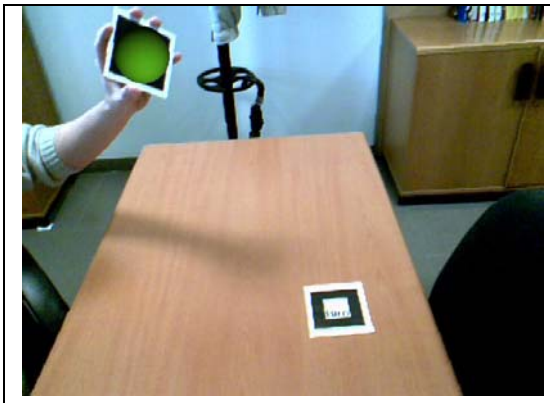
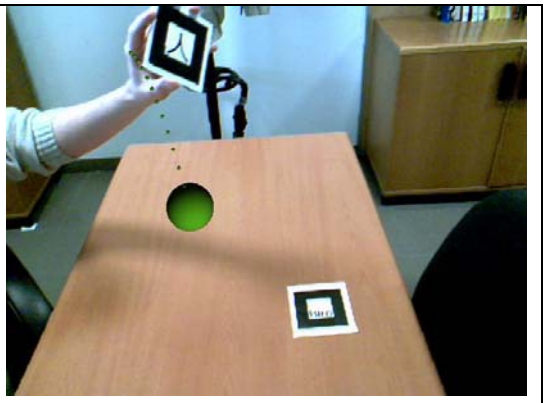
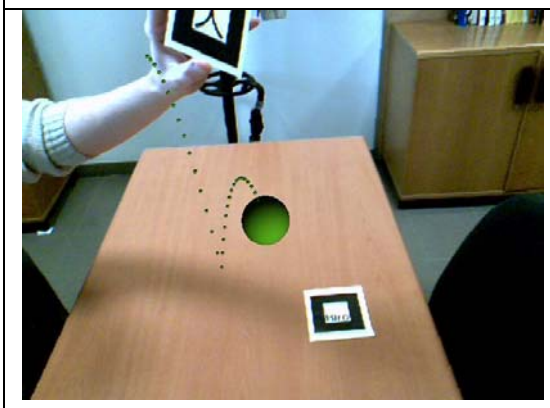
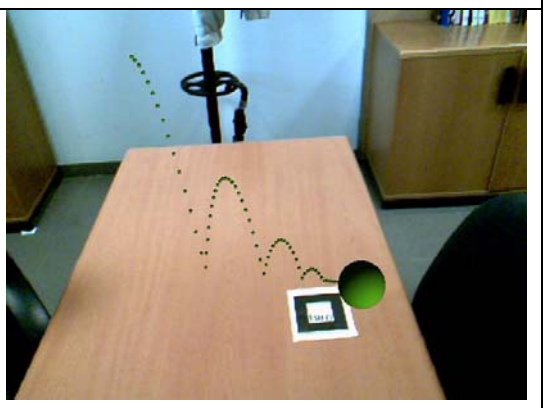
dove:

- $\{n\}_{\text{piano}}$ è il vettore normale al piano;
- e è uno scalare che rappresenta il coefficiente di impatto;

Mentre la simulazione è interamente gestita dall'applicazione, le condizioni iniziali di moto sono definite dall'utente in maniera interattiva. Infatti, premendo un tasto della tastiera o del mouse l'utente può avviare l'applicazione, mentre movimentando il marker

definisce la posizione della sfera nello spazio rispetto alla frequenza di acquisizione e quindi direzione e modulo della velocità iniziale .

Tabella 3.5.1. Visualizzazione della simulazione.

	
1-Condizioni cinematiche iniziali	2-Simulazione avviata
	
3-Impatto e nuovo avvio della simulazione	4-Traiettoria finale

In Tabella 3.5.1 si riportano alcune schermate della simulazione nelle quali, registrando le posizioni assunte ogni 50 fotogrammi, si visualizza l'intera traiettoria seguita dalla sfera.

3.5.2 Simulazione cinematica di un braccio robotizzato

L'applicazione di questa sezione concerne la simulazione cinematica di un braccio robotizzato, dove la posizione dell'effettore viene controllata in maniera interattiva dall'utente.

La tecnica utilizzata per la simulazione prevede il calcolo degli angoli di giunto e la posizione di tutti gli elementi, per configurazioni congruenti con la posizione dell'effettore e delle coppie cinematiche. Questa applicazione, grazie all'utilizzo della tecnologia della realtà aumentata, è supportata visivamente dai componenti virtuali che

vengono collegati al mondo reale mediante due marker, di cui uno agisce da base e l'altro definisce la posizione dell'effettore.

La procedura utilizzata per lo sviluppo dell'applicazione, può essere scomposta nelle seguenti fasi:

1. Preparazione dei modelli virtuali dei componenti appartenenti al braccio in un formato compatibile a quello dell'applicazione.
2. Preparazione della scena. In questa fase si devono scegliere tipologia e dimensioni dei marker utilizzati nella simulazione e la strumentazione necessaria ad acquisire un flusso video. Inoltre occorre impostare le luci ed eliminare eventuali fonti di disturbo;
3. Definizione di una procedura di calcolo per eseguire la simulazione cinematica in tempo reale;

Le prime due fasi sono state sviluppate seguendo la strategia già vista per le applicazioni di ingegneria inversa e manutenzione, mentre la fase di calcolo cinematico richiede istruzioni computazionali specifiche. Tale procedura deve consentire di calcolare esplicitamente la posizione e l'orientamento di ciascun componente compatibilmente con i vincoli.

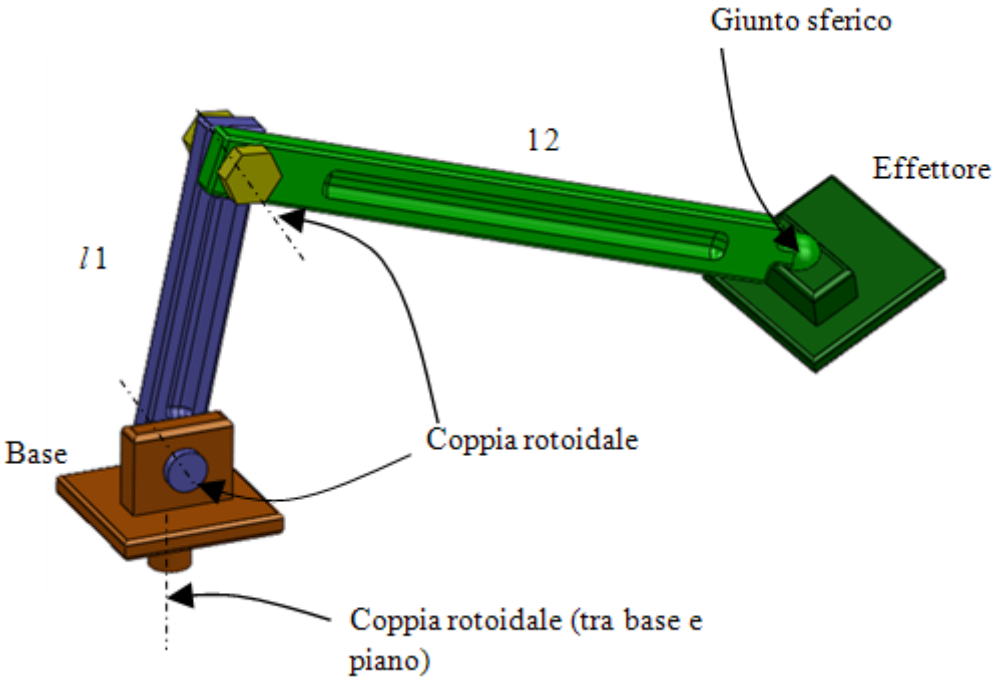


Figura 3.5.2. Modello virtuale del robot e vincoli cinematici.

In Figura 3.5.2, si riporta il modello virtuale del robot, costituito da 4 componenti connessi mediante tre coppie rotoidali e una coppia sferica. Secondo la formula di Grubler il sitema possiede 6 gradi di libertà:

$$dof = 6 \cdot n_{\text{link}} - \sum_{i=1}^{joints} (6 - f_i) = 6$$

dove n_{link} è il numero di parti in movimento, *joints* rappresenta il numero di coppie cinematiche ed f_i è il numero di vincoli semplice per la i-esima coppia. Quindi occorre scegliere 6 parametri indipendenti per definire una configurazione ammissibile del sistema. Al fine di garantire l'interattività si possono utilizzare i parametri cinematici di posizione e orientamento dell'effettore, rilevati attraverso i marker.

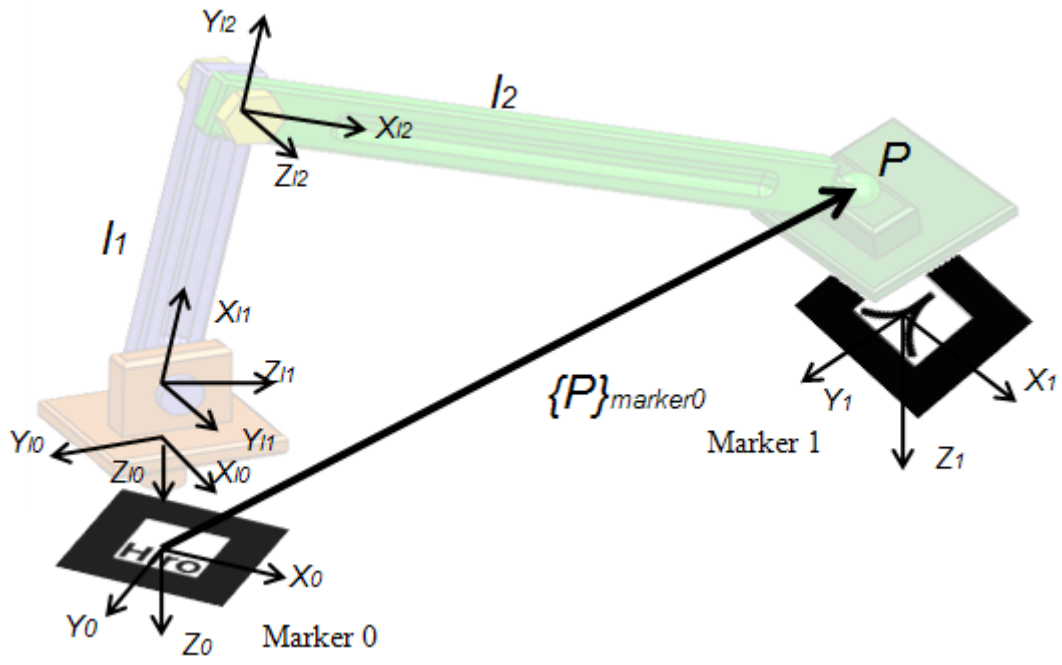


Figura 3.5.3. Descrizione dei sistemi di riferimento assegnati al robot.

Il primo passo consiste nell'implementazione dei comandi per il disegno e la visualizzazione dei componenti virtuali del braccio. Al solito i componenti si possono esportare da un software CAD e caricare mediante i comandi (v. §3.4) della libreria *arVrml.h*, oppure attraverso i comandi di base delle OpenGL (v. Appendice).

Il secondo passo riguarda l'impostazione della scena di simulazione. A tal fine si è scelto di impiegare due marker: uno per definire la posizione e l'orientamento del

manipolatore nella scena e l'altro per definire la posizione dell'effettore, manovrato dall'utente Figura 3.5.2.

Il terzo passo riguarda l'implementazione della procedura di calcolo, per la quale si possono utilizzare le equazioni di chiusura tra i componenti del braccio (v. Figura 3.5.3). In particolare considerando la punta del componente *l2*, corrispondente al centro del giunto sferico, si può calcolarne la posizione rispetto al marker 0 come:

$$\{P\}_{\text{marker0}} = [T]_{l0}^{0} \cdot [T]_{l1}^{l0} \cdot [T]_{l2}^{l1} \cdot \{P\}_{\text{link2}} \qquad (3.12)$$

dove:

- $\{P\}_{\text{marker0}}$ è il vettore posizione che individua il punto *P* dell'effettore nel sistema di riferimento associato al marker 0;
- $\{P\}_{\text{l2}}$ è il vettore posizione che individua il punto *P* nel riferimento locale del secondo componente *l2*;
- $[T]_{0}^{l0}$ è la matrice di trasformazione omogenea relatica al componente *l0* e al marker 0. Tale matrice è funzione del parametro α che definisce la rotazione relativa rispetto alla coppia rotoidale che lega la base al sistema di riferimento globale associato al marker 0;
- $[T]_{l1}^{l0}$ è la matrice di trasformazione omogenea tra i componenti *l1* e *l0* ed è funzione del parametro angolare β, che descrive la rotazione relativa tra i due componenti rispetto alla coppia rotoidale che li vincola;
- $[T]_{l2}^{l1}$ è la matrice di trasformazione che lega i componenti *l1* e *l2* ed è funzione del parametro γ, che descrive la rotazione relativa tra i due componenti rispetto alla coppia rotoidale che li vincola;

D'altra parte definendo la posizione relativa del punto *P* dell'effettore, che appartiene al marker 2, rispetto al marker 0, la relazione (3.12) si può riscrivere come:

$$\{P\}_{\text{marker0}} = [T]_{\text{M0}}^{M1} \cdot \{P\}_{\text{slider}} \qquad (3.13)$$

dove:

- $\{P\}_{\text{slider}}$ rappresenta la posizione del punto *P* nel riferimento del marker 1, coincidente con la punta dell'effettore;

- $[T]_{\mathrm{M0}}^{M1}$ è la matrice di trasformazione omogenea ed è funzione di 6 parametri indipendenti che individuano la posizione e l'orientamento relativo tra i due marker. Tali parametri si possono considerare come l'input della simulazione cinematica poichè sono definiti interattivamente dall'utente attraverso la movimentazione del marker associato all'effettore.

La matrice di trasformazione $[T]_{\mathrm{M0}}^{M1}$ si può calcolare a partire dalla relazione (3.5) mediante il metodo `arGetTransMat`.

Quindi le equazioni di chiusura, per il meccanismo considerato, assumono la seguente espressione:

$$[T]_{l0}^{0}\cdot[T]_{l1}^{l0}\cdot[T]_{l2}^{l1}\cdot\{P\}_{\mathrm{link}2}\cdot\{P\}_{\mathrm{link}2}-[T]_{\mathrm{M0}}^{M1}\{P\}_{\mathrm{slider}}=\begin{Bmatrix}0\\0\\0\\1\end{Bmatrix} \tag{3.14}$$

Risolvendo il sistema di equazioni riportate in (3.14), a partire dai valori di inclinazione e posizione del marker 1 rispetto al marker 0, si ricavano i parametri cinematici α,β,γ, per ogni configurazione assunta dall'effettore.

Partendo dai risultati della simulazione, il passo successivo riguarda il posizionamento dei corpi nella finestra grafica. Poiché le Artoolkit utilizzano l'ambiente grafico OpenGl occorre definire sia le trasformazioni di vista che quelle di proiezione per gli oggetti da visualizzare. Mentre le trasformazioni di proiezione si ricavano dalla conoscenza dei parametri intrinseci della camera adottata per l'acquisizione e dalla posizione relativa del marker 0 rispetto alla camera, le trasformazioni per il posizionamento degli oggetti nella scena devono essere aggiornate ad ogni passo della simulazione. Il problema si riconduce al calcolo delle matrici di trasformazione da applicare alle primitive grafiche o agli oggetti importati in formato *VRML*[21]. A tal fine si possono adottare due strategie:

- calcolare le trasformazioni da applicare rispetto al riferimento assoluto;

[21] La posizione dei sistemi di riferimento associati agli oggetti VRML può essere definita variando le condizioni di un file di testo.

- calcolare le trasformazioni relative di ogni oggetto in funzione della posizione dell'oggetto che lo precede nelle istruzioni di disegno;

In entrambi i casi si possono utilizzare le funzioni semplificate `glRotated` e `glTranslated` oppure caricare la matrice di trasformazione omogenea mediante l'istruzione `glLoadMatrixd(m)` per ogni corpo.
Nel caso si voglia procedere con il primo metodo occorre inserire le istruzioni di posizionamento per ogni oggetto tra le istruzioni `glPushMatrix()` e `glPopMatrix()`,con il primo comando che carica le trasformazioni nello stack delle matrici e il secondo che torna allo stato precedente alle trasformazioni applicate (v Appendice A).

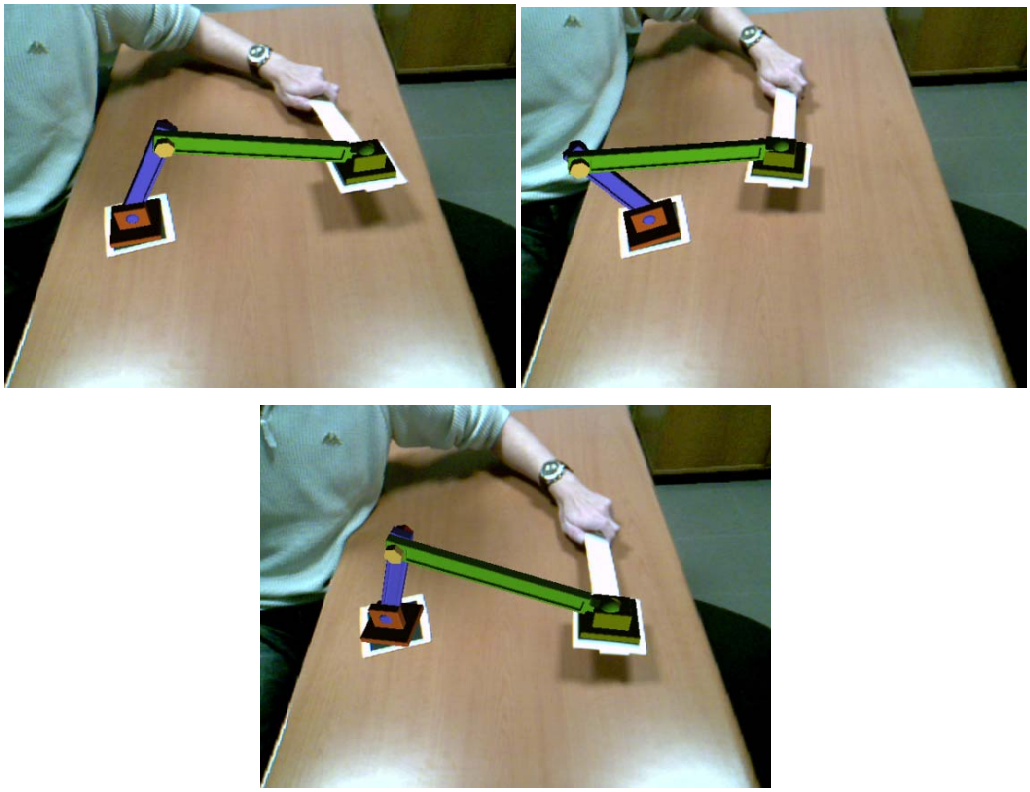


Figura 3.5.4. Alcune schermata dell'applicazione: il robot viene movimentato attraverso un marker che è virtualmente attaccato all'effettore.

Data la relativa semplicità dell'applicazione sviluppata si è ritenuto opportuno posizionare gli elementi seguendo tale metodo. Di seguito si riportano in dettaglio le istruzioni utilizzate per ciascun corpo.
Per la base è sufficiente applicare una rotazione rispetto all'asse *Z* del marker 0, l'istruzione corrispondente è la seguente:

```
glRotated(alpha,0.0,0.0,1.0);
```

Per il primo componente *l0* occorre definire tre istruzioni elementari: una traslazione *a1* lungo l'asse *z* del marker 0, una rotazione α attorno al medesimo asse e una rotazione, pari a β intorno all'asse di rotazione definito dal primo giunto. Le istruzioni corrispondenti sono le seguenti:

```
glTranslated(0.0,0.0,a1);
glRotated(alpha,0.0,0.0,1.0);
glRotated(beta,0.0,-1.0,0.0);
```

Per il secondo componente occorrono tre trasformazioni: una rotazione di un angolo α intorno all'asse *Z* del marker 0, una traslazione verso il centro dell'asse di rotazione tra *l0* e *l1* e infine una rotazione, pari a γ, tra i componenti *l0* e *l1*. Le istruzioni corrispondenti sono le seguenti:

```
glRotated(alpha,0.0,0.0,1.0);
glTranslated(l1*cos(beta),0.0,l1*sin(beta)+a1);
glRotated(gamma,0.0,-1.0,0.0);
```

Infine il componente dell'effettore, poichè è attaccato al marker 1, può essere posizionato utilizzando direttamente la matrice di proiezione del marker stesso.

Un altro aspetto da considerare nello sviluppo di applicazioni multibody in AR è quello di prevedere elementi grafici addizionali inerenti ai dati della simulazione, secondo quanto previsto dalla pratica ingegneristica. Di fatto nei software multibody attualmente in commercio i risultati riguardanti le posizioni, le velocità, le accelerazioni e i risultati riguardanti forze, momenti e reazioni generate in prossimità delle coppie cinematiche, vengono registrate in file e visualizzate mediante grafici.
D'altra parte la AR offre la possibilità di proiettare tali valori direttamente sulla scena aumentata, aggiungendo entità vettoriali, rappresentati a mezzo di frecce tridimensionali, corredate di scritte che riportano i valori numerici direttamente sul componente oggetto dello studio. Esempi di indicatori e annotazioni virtuali sono riportati nelle Figura 3.5.5 e Figura 3.5.6.

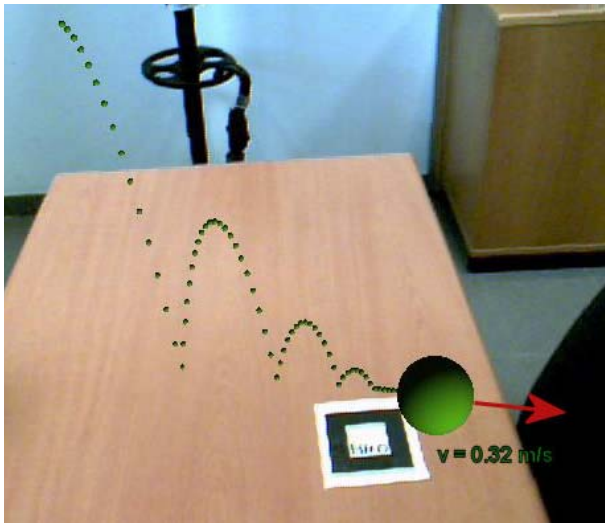


Figura 3.5.5. La simulazione della caduta di un grave corredata da informazioni virtuali.

In Figura 3.5.5 oltre alla sfera e alla sua traiettoria, si riporta una freccia virtuale, che ne rappresenta la velocità durante la simulazione. Posizione, orientamento e lunghezza della freccia vengono aggiornate in tempo reale per comunicare più efficacemente i risultati della simulazione.

In Figura 3.5.6 lo scenario del braccio robotizzato viene completato con l'aggiunta di due contatori, che visualizzano la posizione angolare della base e dell'effettore del braccio durante la simulazione.

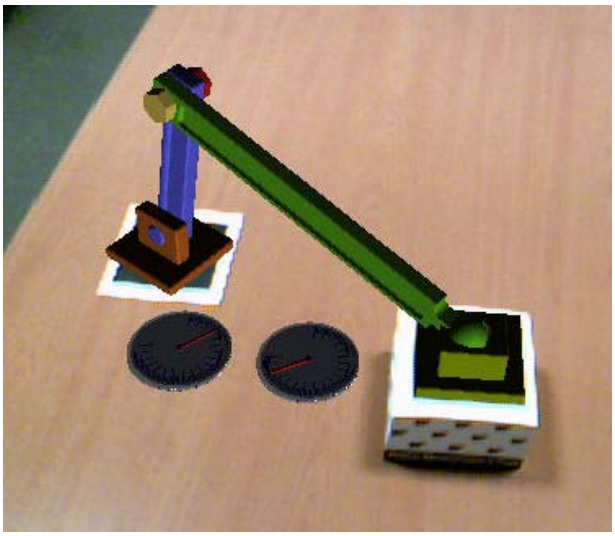


Figura 3.5.6. Simualzione del robot e indicatori angolari virtuali.

3.5.3 Simulazione dinamica di un manovellismo in AR

L'ultimo esempio di applicazione multibody in AR riguarda la simulazione dinamica di un manovellismo di spinta composto dai componenti riportati in Figura 3.5.7: base, manovella, biella e pattino, con la base connessa al pattino mediante molla e smorzatore. Lo scopo dell'applicazione è quello di illustrare come la AR consenta di definire le condizioni iniziali della simulazione e visualizzare efficacemente il comportamento dinamico del meccanismo.

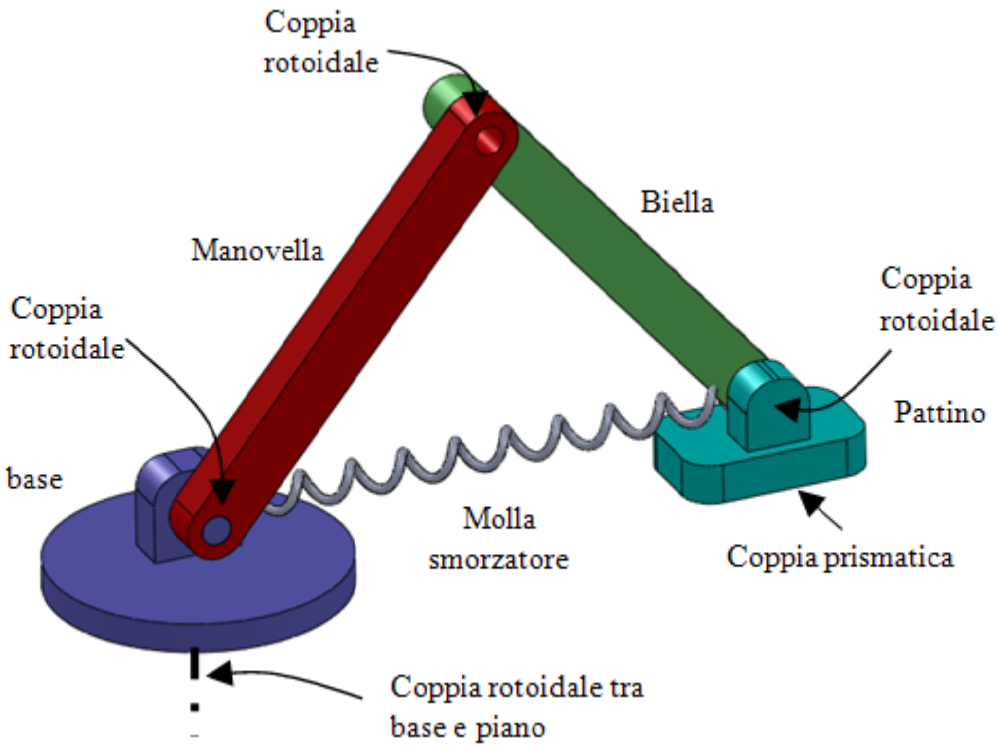


Figura 3.5.7. Descrizione dei componenti e dei vincoli cinematici del meccanismo studiato.

Anche in questo caso occorre implementare il sistema prevedendo le tre fasi di impostazione riguardanti:

- preparazione dei modelli virtuali dei componenti;
- preparazione degli elementi utilizzati come sensori nella scena;
- implementazione del codice necessario alla simulazione.

Mentre i modelli virtuali dei componenti si possono importare in formato *VRML*, con le procedure presentate in precedenza, per la collimazione degli scenari, reale e virtuale, occorrono due marker, di cui uno attaccato al componente base e l'altro al pattino.

Inoltre occorre prevedere una procedura di calcolo per risolvere il problema dinamico e una di visualizzazione per proiettare i risultati della simulazione.

Per risolvere le equazioni del moto in maniera iterativa, occorre scrivere le equazioni del sistema algebrico differenziale del meccanismo

$$\begin{bmatrix} M & \left[\Psi_q\right]^t \\ \left[\Psi_q\right] & 0 \end{bmatrix} \cdot \begin{Bmatrix} \{\ddot{q}\} \\ \{\lambda\} \end{Bmatrix} = \begin{Bmatrix} \{F\} \\ \{\gamma\} \end{Bmatrix} \tag{3.15}$$

dove:

- $\{\ddot{q}\}$ è il vettore generalizzato delle accelerazioni ;
- $\{q\}$ è il vettore generalizzato delle posizioni;
- $\{\dot{q}\}$ è il vettore generalizzato delle velocità;
- $\{\lambda\}$ è il moltiplicatore lagrangiano;
- $\{F\}$ è il vettore generalizzato delle forze che contiene i contributi dovuti alla molla e allo smorzatore;
- Ψ_q è lo jacobiano delle equazioni di vincolo;
- γ può essere esplicitato, in funzione dei parametri cinematici con la seguente espressione:

$$\{\gamma\} = -\left(\left[\Psi_q\right]\{\dot{q}\}\right)_q \{\dot{q}\} - 2\left[\Psi_{qt}\right]\{\dot{q}\} - \{\Psi_{tt}\}$$

- *M* è la matrice inerziale;

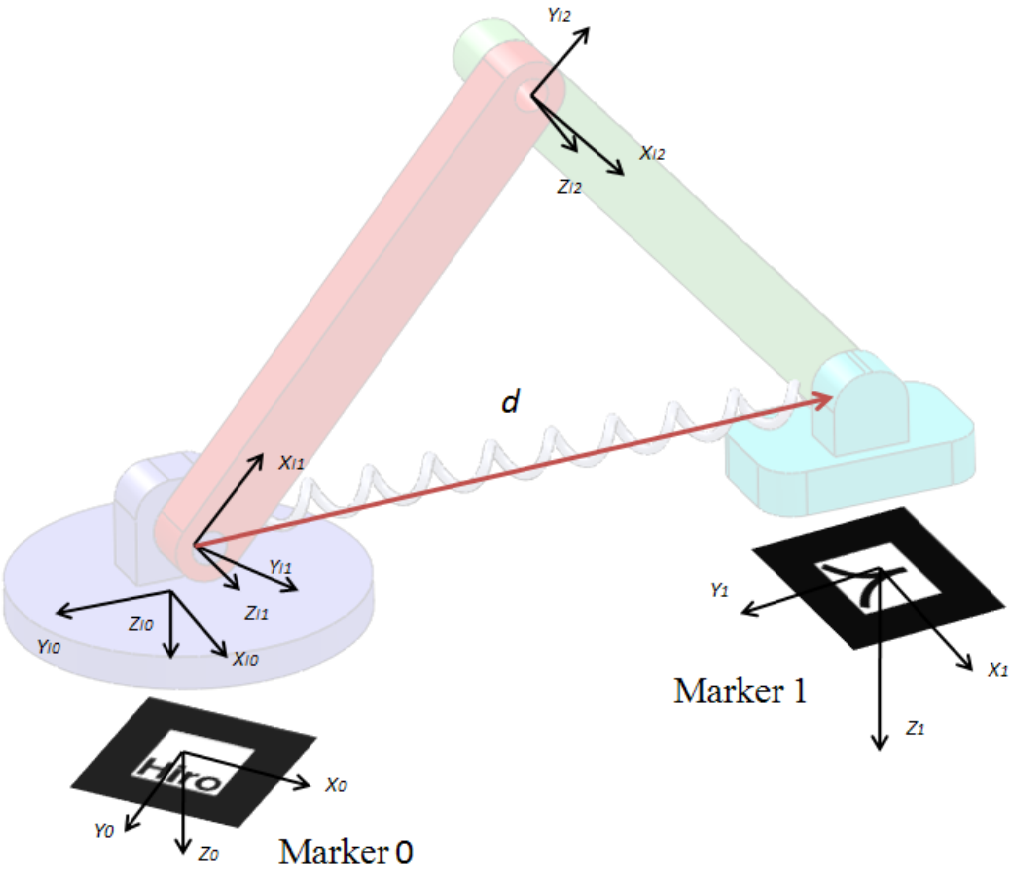


Figura 3.5.8. Sistemi di riferimento assegnati ai giunti e ai marker.

Per l'esempio in oggetto si hanno 23 equazioni di vincolo, di cui 5 per le coppie cinematiche rotoidali e 3 per quelle prismatiche, e il meccanismo possiede 1 grado di libertà.

Tabella 3.5.2. Procedura di impostazione interattiva della simulazione.

Figura 3.5.9 Impostazione della lunghezza della molla a riposo

Figura 3.5.10 Inserimento della molla

Figura 3.5.11 Deformazione iniziale

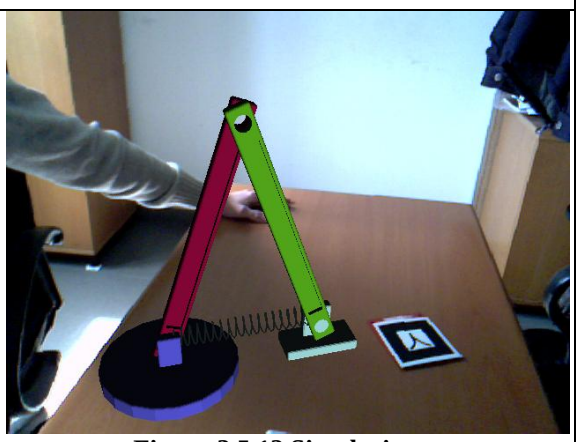


Figura 3.5.12 Simulazione

In Tabella 3.5.2 si riportano alcune schermate relative alla procedura di impostazione interattiva della simulazione e alla simulazione stessa. La Figura 3.5.9 mostra il primo passo della simulazione, dove l'utente sposta il marker 1 per definire la configurazione a riposo del meccanismo. La Figura 3.5.10 riporta l'operazione successiva, nella quale l'utente, premendo un tasto sulla tastiera o sul mouse, inserisce la molla e avvia la fase successiva, rappresentata in Figura 3.5.11. In questa fase l'utente definisce la lunghezza e quindi la deformazione iniziale della molla. Premendo nuovamente un tasto si avvia la simulazione (v. Figura 3.5.12). Le condizioni cinematiche iniziali si possono determinare con un approccio simile a quello riportato nel § 3.5.1 per la sfera.

Infine, in funzione dei risultati ottenuti dalla simulazione gli oggetti possono essere posizionati nello spazio utilizzando i comandi di proiezione delle OpenGl mostrati nella sezione precedente.

3.5.4 Conclusioni e sviluppi futuri

Nei sistemi di AR, sfruttando la frequenza di elaborazione del flusso video proveniente dalla camera, si riesce a risalire alla posizione di un corpo virtuale, associato ad un sensore nello spazio, rispetto al tempo. Tale potenzialità estende l'utilizzo di queste applicazioni alle simulazioni multibody.

Le applicazioni riportate in questo capitolo evidenziano alcuni vantaggi nell'impiego della AR per la simulazione dinamica di strutture meccaniche. Il primo riguarda la possibilità di visualizzare le strutture studiate nel loro contesto di utilizzo e valutare possibili interferenze o malfunzionamenti rispetto agli oggetti reali.

Un altro aspetto da considerare è la maggiore interattività assicurata dall'ambiente di simulazione aumentato. L'utilizzo dei sensori (marker) consente un'interazione più intuitiva di quella assicurata dal mouse, solitamente utilizzato nei software CAE per puntare ai componenti.

Invece i possibili sviluppi del lavoro svolto interessano principalmente due aspetti. Il primo concerne la proiezione dei parametri cinematici e dinamici della simulazione mediante elementi e scritte virtuali. Il secondo riguarda la valutazione, finalizzata all'ottimizzazione delle applicazioni, dei limiti della simulazione real time in funzione dell'accuratezza del risultato e della complessità delle strutture elaborate.

3.6 Post-processing in Realtà Aumentata

La AR riesce ad aumentare la percezione visiva del mondo. Tale caratteristica viene spesso utilizzata per proiettare informazioni aggiuntive sulle proprietà e caratteristiche degli oggetti presenti nel campo visivo dell'utente (v. Capitolo 1). In questa sezione vengono presentate applicazioni di Post Processing in AR, dove le informazioni aggiuntive riguardano le proprietà fluidodinamiche di un cilindro investito da un flusso d'aria e le proprietà strutturali di una barra. Lo schema di implementazione delle applicazioni presentate è quello riportato in Figura 3.6.1, dove si distinguono due gruppi di attività da svolgere.

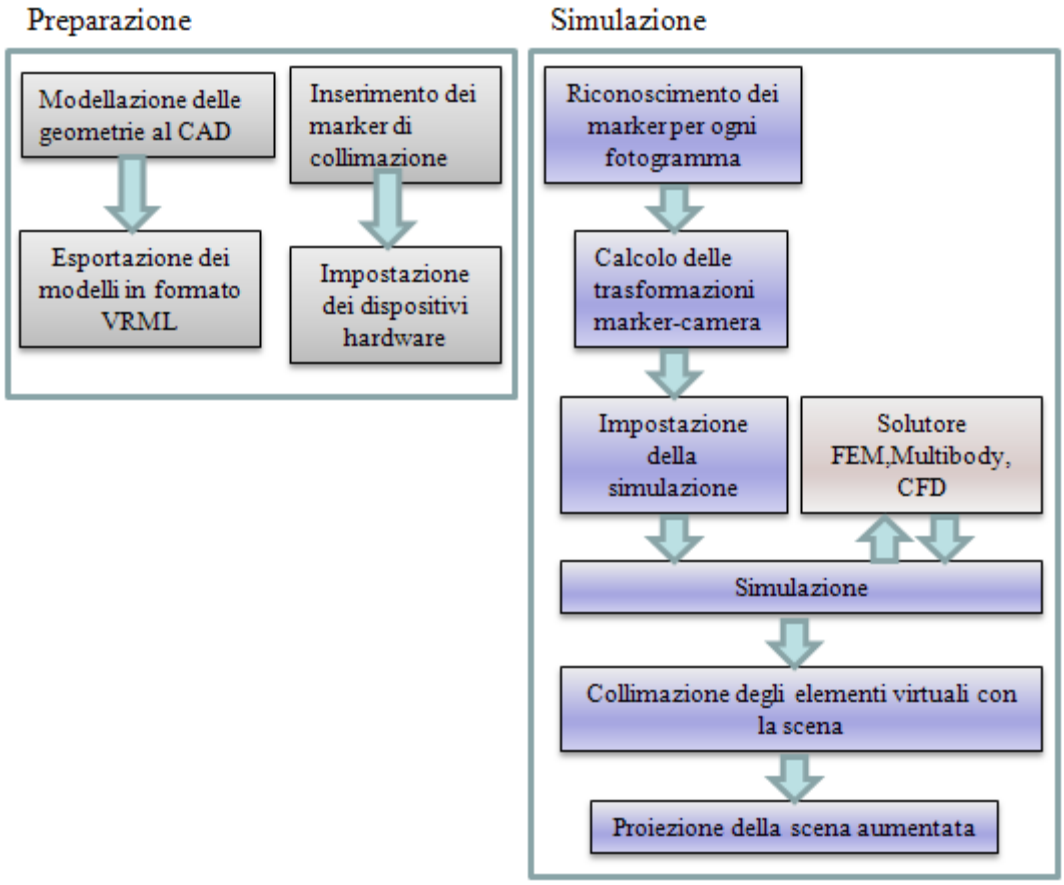


Figura 3.6.1. Attività per implementare simulazioni ingegneristiche in ambiente di AR.

Le attività da svolgere prima della simulazione sono riportate nella parte sinistra della Figura 3.6.1 e come verrà evidenziato nelle sezioni seguenti riguardano la preparazione dei modelli virtuali, dei marker da applicare alla scena e dei dispositivi utilizzati per l'acquisizione e l'elaborazione dati.

Le attività da svolgere dopo la simulazione sono riportate nel riquadro destro di Figura 3.6.1 riguardano la collimazione e la proiezione del flusso video aumentato e vengono eseguite mediante le funzioni implementate nelle Artoolkit (v. Capitolo 2).

3.6.1 Simulazione fluidodinamica in Realtà Aumentata

Le simulazioni fluidodinamiche consentono di studiare il comportamento, in termini di parametri cinematici e termodinamici, di un flusso di gas o liquidi all'interno o all'esterno di componenti (v. Figura 3.6.2).

Figura 3.6.2. Risultati della simulazione fluidodinamica di un condotto.

In questa sezione si riporta il caso dello studio fluidodinamico di un flusso d'aria che investe un cilindro appoggiato su un piano, dove cilindro e supporto sono reali mentre i risultati dell'analisi sono proiettati mediante tecniche di realtà aumentata. Il primo passo, nell'implementazione dell'applicazione, è quello di riprodurre le geometrie in gioco (piano e cilindro) necessarie per la simulazione virtuale. Successivamente, dato il numero consistente di parametri da considerare, occorre impostare le condizioni al contorno del problema ed eseguire l'analisi con un solutore fluidodinamico esterno all'applicazione. L'ultimo passo riguarda la sovrapposizione dei risultati numerici alla geometria di base.

Le simulazioni fluidodinamiche (v. Figura 3.6.3 in alto a destra) vengono di solito visualizzate a mezzo di linee di flusso virtuali, che rappresentano la traiettoria del fluido e vengono colorate in funzione del valore del parametro cinematico o termodinamico

caratterizzante lo stato del fluido nello spazio della simulazione. Questo tipo di visualizzazione consente di rilevare in maniera qualitativa ed intuitiva il comportamento del fluido per il caso studiato.

La corretta proiezione dei risultati sulla scena richiede al solito la collimazione dei sistemi di riferimento dell'ambiente reale e virtuale (*comunication reference frame* in Figura 3.6.3). Poiché l'applicazione è stata sviluppata con le Artoolkit (v. Capitolo 2) la registrazione avviene per riconoscimento visivo di un marker posizionato in prossimità del cilindro oggetto dello studio (v. Figura 3.6.3).

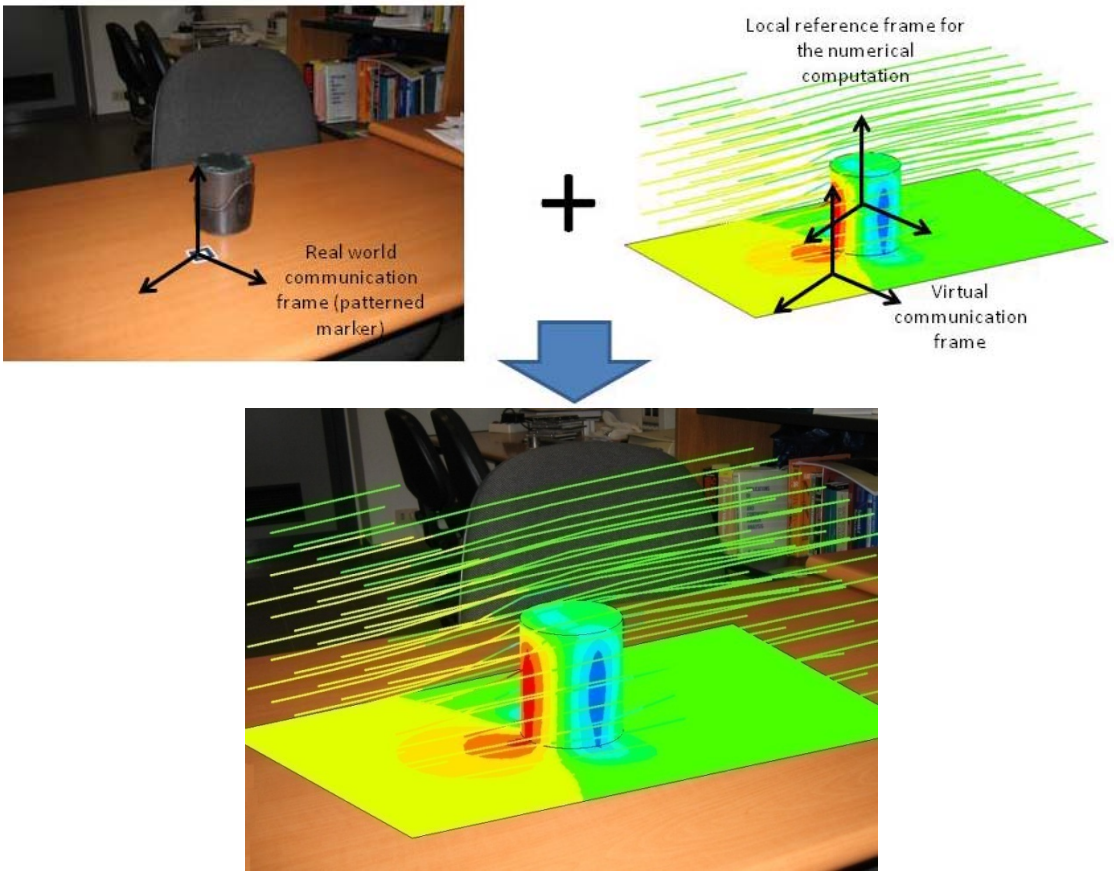


Figura 3.6.3. Simulazione fluidodinamica del flusso d'aria intorno ad un cilindro mediante tecniche di AR. Scenario reale e scenario virtuale della simulazione (in alto), scenario aumentato (in basso).

Infine i contenuti virtuali si possono importare in formato *VRML* o modellare con i comandi dedicati al disegno delle OpenGL. Inoltre, mediante le funzioni OpenGL per la disposizione degli oggetti nella scena (`glRotated` e `glTranslated`), è possibile registrare la posizione delle entità virtuali nella scena.

La scena finale, visualizzata in Figura 3.6.3 in basso, contiene, in sovrapposizione alla scena iniziale, le superfici virtuali del cilindro e del piano investite dal flusso di linee che rappresenta l'andamento del flusso d'aria.

Riassumendo il processo di sviluppo dell'applicazione è composto da 6 fasi principali::

1. Definizione di un sistema di collimazione per contatto o per riconoscimento visivo, per registrare la posizione degli oggetti virtuali rispetto alla scena reale;
2. Acquisizione e riproduzione del problema reale, in termini di geometrie e condizioni fluidodinamiche e termodinamiche, necessarie per impostare la simulazione;
3. Calcolo dei campi di moto e pressione con un solutore FEM esterno;
4. Esportazione e/o ricostruzione dei risultati in oggetti solidi corredati di informazioni grafiche (diagrammi a falsi colori).
5. Inserimento nella scena aumentata e visualizzazione dei risultati nel contesto di pre-processing.

Infine si possono inserire informazioni aggiuntive riguardanti valori numerici delle caratteristiche analizzate (velocità, temperatura, pressione).

3.6.2 Simulazioni strutturali in Realtà Aumentata

La tecnica di simulazione agli elementi finiti, consente di risalire allo stato tensionale interno e alle deformazioni presenti in una struttura sottoposta a vincoli e carichi esterni.

In questa sezione si riporta il caso di una barra d'acciaio caricata con una forza in mezzeria e vincolata alle estremità mediante appoggi. Mentre la barra e i supporti sono realmente presenti nella scena (v. Figura 3.6.4 in alto a sinistra), la forza e i risultati della simulazione devono essere aggiunti utilizzando la tecnologia della AR.

Come per lo studio fluidodinamico presentato nella sezione precedente, il primo passo consiste nell'acquisire, mediante tecniche per contatto o senza contatto, e riprodurre al CAD la topologia del caso studiato. Il passo successivo consiste nel definire la posizione e l'entità della forza (v. Figura 3.6.4 in alto a destra), utilizzando puntatori tridimensionali o inserendo manualmente i parametri della simulazione.

Dopo aver riprodotto virtualmente le geometrie e definito le condizioni al contorno, per simulare il comportamento della struttura, si devono esportare i parametri della simulazione in un solutore FEM esterno all'applicazione.

L'ultima operazione riguarda l'importazione e la proiezione dei risultati della simulazione. In genere i risultati delle simulazioni strutturali vengono visualizzati con diagrammi a falsi colori, dove si possono confrontare la struttura a riposo con la struttura deformata dai carichi e valutare qualitativamente l'andamento delle tensioni agenti.

Affinché l'applicazione risulti efficace occorre riprodurre lo stesso effetto grafico dei programmi FEM. Anche in questo caso occorre implementare la procedura di collimazione per sovrapporre correttamente i risultati alla scena.

La corretta proiezione dei risultati sulla scena richiede al solito la collimazione dei sistemi di riferimento dell'ambiente reale e virtuale (*comunication reference frame* di Figura 3.6.4). Poiché l'applicazione è stata sviluppata con le Artoolkit (v. Capitolo 2), la registrazione avviene per riconoscimento visivo di un marker posizionato in prossimità di uno dei due appoggi, allineato orizzontalmente con la barra.

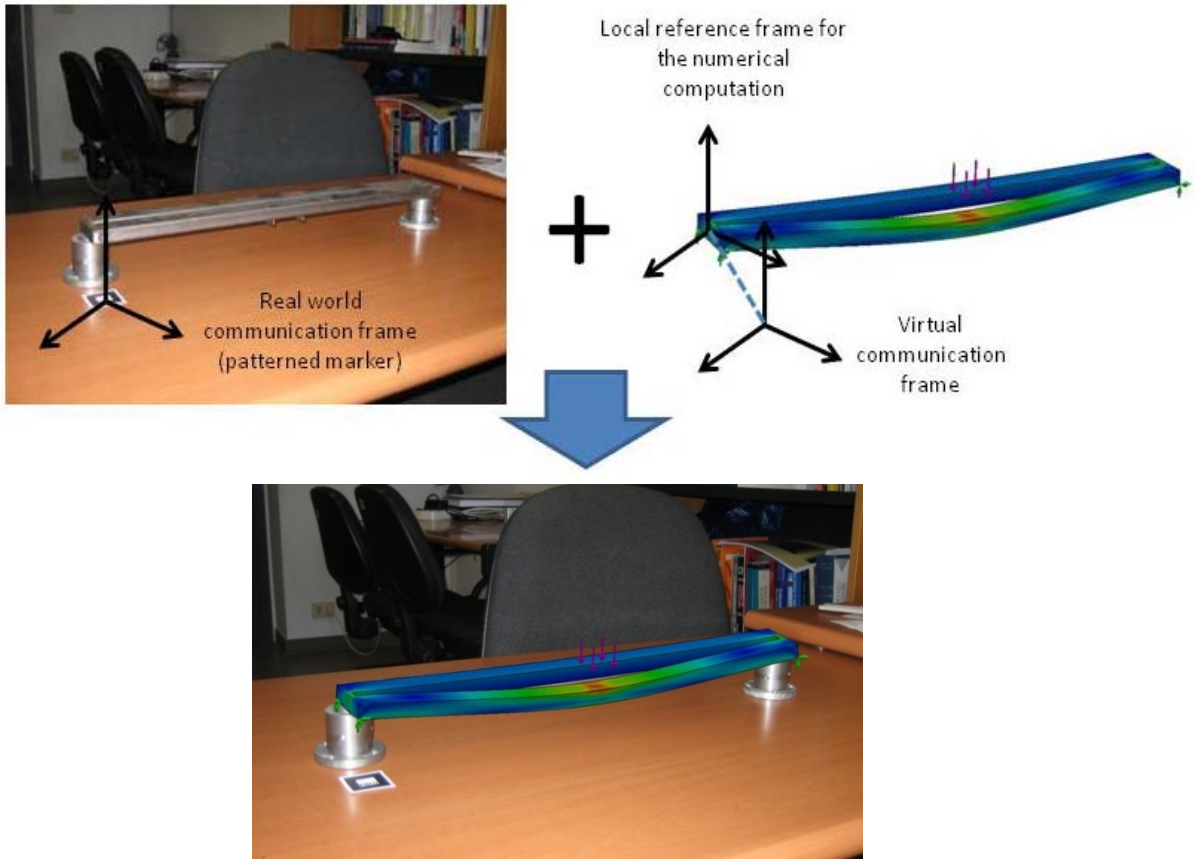


Figura 3.6.4. Visualizzazione della deformata di una trave. Immagini reale e virtuale (in alto), immagine aumentata in basso.

Una schermata della scena aumentata con i risultati della simulazione FEM è riporta in Figura 3.6.4 in basso. Grazie alla tecnologia della AR l'utente può osservare la deformata e lo stato tensionale nella struttura secondo diversi punti di vista. Inoltre si può includere nella scena aumentata l'animazione della deformazione al posto della deformata statica della struttura. Tale funzionalità risulta particolarmente utile per le

analisi modali dove è spesso richiesta la visualizzazione dei modi di vibrare delle strutture.

Riassumendo il processo di sviluppo dell'applicazione è composto da 6 fasi principali:

1. Inserimento di marker per la collimazione tra scena reale e scena virtuale;
2. Preparazione delle geometrie virtuali oggetto della simulazione;
3. Impostazione delle condizioni al contorno per l'analisi da condurre;
4. Generazione del file di input da inviare al solutore FEM;
5. Esportazione dei risultati in formato *vrml* per la visualizzazione grafica;
6. Visualizzazione dei risultati nella scena aumentata;

Un possibile sviluppo dell'applicazione riguarda l'introduzione di scritte virtuali e legende, posizionate opportunamente nella scena, per comunicare i valori numerici di stress e deformazione raggiunti dalla struttura.

3.6.3 Conclusioni e sviluppi futuri

La AR, utilizzata per il post processing di simulazioni fluidodinamiche e strutturali, consente di proiettare efficacemente i risultati delle simulazioni e di migliorare l'interattività rispetto agli oggetti reali, presenti nella scena, e a quelli virtuali coinvolti nella simulazione.

Gli strumenti e le tecniche presentate in questa sezione, si possono applicare proficuamente al design collaborativo, dove è necessario condividere, tra più utenti, le informazioni relative ad un prototipo virtuale. In questo caso si pensa ad una applicazione server, in grado di gestire il flusso dati del prototipo e di smistarlo tra diversi reparti. In ciascun reparto gruppi di progettisti che indossano dispositivi AR client possono analizzare il modello tridimensionale del prodotto e studiarne caratteristiche meccaniche e proprietà fluidodinamiche.

Capitolo 4 **Conclusioni e sviluppi futuri**

Un limite sempre più evidente delle applicazioni di Realtà Virtuale riguarda i costi connessi ai dispositivi di interazione. Nonostante i tentativi di importare le tecnologie più evolute nelle moderne applicazioni ingegneristiche [54] la tradizionale interfaccia desktop rimane tuttora la più diffusa.

D'altra parte la Realtà Aumentata è una tecnologia informatica emergente, proprio in virtù della sua economicità di implementazione. Ragionando in termini di sovrapposizione di oggetti virtuali, piuttosto che sostituzione dell'ambiente che ci circonda, questa disciplina ha trovato applicazione in diversi settori di ricerca che fino agli anni 90 erano completo appannaggio delle applicazioni di Realtà Virtuale (addestramento, medicina, architettura etc.).

L'obiettivo di questo lavoro di ricerca, partendo dalle diversità concettuali tra Realtà Virtuale e Realtà Aumenta e guardando alle tecnologie sviluppate nell'ultimo ventennio in ambito informatico, è stato quello di valutare l'applicabilità della Realtà Aumentata alle simulazioni ingegneristiche, impiegate nella progettazione dei sistemi meccanici.

Sebbene risulti prematuro parlare di "ingegneria aumentata", le applicazioni sviluppate evidenziano come la AR sia in grado di rivoluzionare il tradizionale modo di interagire con gli ambienti di simulazione CAE. A tal riguardo le applicazioni realizzate per la visualizzazione delle operazioni di manutenzione e per la proiezione dei risultati di analisi fluidodinamiche e strutturali, consentono al progettista di analizzare e condividere dati in maniera più intuitiva. Inoltre l'applicazione *AR-CAD2.0* è un esempio di come si possa personalizzare un'applicazione di Realtà Aumentata e renderla

funzionale alla progettazione meccanica. A tal riguardo l'intuizione sull'utilizzo di un dispositivo di puntamento magnetico costituisce una alternativa alle interfacce delle applicazioni di disegno assistito dal calcolatore tradizionali e consente di avvicinarsi all'auspicata, ma costosa interattività, dei CAVE.

D'altra parte l'applicazione di Reverse Engineering, proprio grazie all'impiego della Realtà Aumentata, fornisce un metodo ibrido che per talune tipologie di oggetti coniuga la velocità dei metodi di acquisizione senza contatto con la possibilità di filtrare il numero dei punti, già nella fase di acquisizione, dei metodi per contatto. Tale strategia, come evidenziato dall'applicazione pratica, è risultata particolarmente economica, in termini di costi, per la realizzazione del sistema, e in termini di tempo, per la fase di elaborazione dati.

Già si è discusso, nelle relative sezioni del Capitolo 3, dei possibili impieghi ed evoluzioni dei sistemi sviluppati. Tuttavia è necessario osservare che, come è accaduto per la Realtà Virtuale, affinché la AR porti dei reali benefici alla progettazione dei sistemi meccanici, occorre percepirne le potenzialità e trovare, di conseguenza, le giuste motivazioni e le risorse per lo sviluppo e l'impiego di questa tecnologia nelle simulazioni ingegneristiche.

Appendice A **Le librerie grafiche OpenGL e GLUT per applicazioni ingegneristiche**

Le librerie grafiche OpenGL sono spesso utilizzate nello sviluppo di software CAD per migliorare la qualità e la velocità di visualizzazione dei modelli virtuali. Inoltre grazie alla facilità con cui è possibile gestire primitive grafiche, sia per la proiezione che per la loro disposizione nello spazio virtuale, le OpenGL sono incluse nei pacchetti di sviluppo di applicazioni di grafica computazionale e nello specifico di Realtà Aumentata [5].
Pertanto in questa appendice si descrivono, senza alcuna pretesa di esaustività e completezza, i comandi principali e la logica di funzionamento di queste librerie con l'obiettivo di creare un valido riferimento nella spiegazione dei sistemi di Realtà Aumentata, sviluppati con librerie Artoolkit.

A1. Introduzione alle OpenGL

OpenGL è un API (application programming interface) sviluppata da SGI nel 1992, composta di circa 150 comandi che permettono di interfacciarsi con l'hardware di accelerazione grafica del sistema, al fine di produrre applicazioni 3D interattive.
È estremamente portabile, in quanto è utilizzabile su Mac OS, OS/2, UNIX, Windows 95/98/Xp/Vista, Windows 2000, Windows NT, Linux, OPENStep, e BeOS.

Inoltre le sue funzionalità e comandi possono essere richiamati da ambienti di sviluppo basati su differenti linguaggi (Ada, C, C++, Fortran, Python, Perl e Java) con una totale indipendenza dai protocolli e dalle tipologie di rete.

Un programma sviluppato con le OpenGL permette di proiettare animazioni e scenari virtuali sul monitor del PC. Le immagini prodotte da queste librerie grafiche sono visualizzate all'interno di una finestra che le OpenGL non gestiscono direttamente. A tal fine è stata sviluppata una libreria complementare GLUT in grado di controllare la finestra grafica e i relativi eventi generati dall'utente.

A2. OpenGL e Glut

Un programma di grafica che utilizzi le librerie grafiche Glut e OpenGL ha la seguente struttura:

- Funzione principale *main.*
- Funzioni di tipo *Callback* per il controllo degli eventi.

Al *main* vengono affidate le seguenti operazione di inizializzazione e di impostazione:

- Inizializzazione dell'applicazione
- Creazione della finestra
- Registrazione di Callback
- Creazione di Menu
- Innesco del ciclo di eventi grafici (*MainLoop*)

Tabella 3.6.1: Descrizione delle principali funzioni dell'esempio di codice riportato in Tabella 3.

Funzione	Descrizione
`void casa(void)`	Contiene i comandi OpenGL per la modellazione di un solido a forma di casa
`void ProcessMenu(int value)`	Gestisce gli eventi associati al menu contestuale dell'applicazione
`void Disegna(void)`	Contiene i comandi OpenGL per disegnare la scena
`void Inizializzazione(void)`	Inizializza l'applicazione impostando la proiezione della scena, le luci e le proprietà dei materiali
`void Ridimensionamento(int w, int h)`	Controlla e scala la scena da proiettare nella finestra di visualizzazione in caso di ridimensionamenti od occlusioni della finestra stessa
`int main(int argc, char* argv[])`	La funzione principale dell'applicazione, inizializza la finestra grafica, crea i menu contestuali associati all'applicazione e associa le funzioni di callback agli eventi ritenuti importanti

Un esempio di codice[22] per la generazione di una finestra grafica e oggetti tridimensionali è riportato in Tabella 3.6.2[23].

Tabella 3.6.2. Listato di esempio per la generazione di un'applicazione grafica interattiva utilizzando comandi OpenGL e GLUT.

```
#include <windows.h>
#include <gl/gl.h>
#include <gl/glu.h>
#include <gl/glut.h>
// Variabili per le rotazioni degli oggetti
static GLfloat xRot = 0.0f;
static GLfloat yRot = 0.0f;
// Proprietà delle luci
GLfloat  ambientLight[] = { 0.3f, 0.3f, 0.3f, 1.0f };
GLfloat  diffuseLight[] = { 0.7f, 0.7f, 0.7f, 1.0f };
GLfloat  specular[] = { 1.0f, 1.0f, 1.0f, 1.0f};
GLfloat  specref[] =  { 1.0f, 1.0f, 1.0f, 1.0f };
// Variabile di disegno
static int iShape = 19;
```

```
//Funzione di disegno
void casa(void)
{
glScalef(2.0,2.0,2.0);
glColor3f(1.0,0.0,0.0);
 glBegin(GL_POLYGON);
  glVertex3f(0.0,0.0,0.0);
  glVertex3f(0.5,0.0,0.0);
  glVertex3f(0.5,0.5,0.0);
  glVertex3f(0.25,0.6,0.0);
  glVertex3f(0.0,0.5,0.0);
 glEnd();
glPushMatrix();
glColor3f(1.0,0.0,0.0);
 glTranslatef(0.0,0.0,0.5);
  glBegin(GL_POLYGON);
  glVertex3f(0.0,0.0,0.0);
  glVertex3f(0.5,0.0,0.0);
  glVertex3f(0.5,0.5,0.0);
  glVertex3f(0.25,0.6,0.0);
  glVertex3f(0.0,0.5,0.0);
 glEnd();
glPopMatrix();
glPushMatrix();
glColor3f(1.0,1.0,0.0);
```

[22] Il codice riportato è stato ottenuto modificando il codice di esempio del 3° capitolo dell' *OpenGL SuperBible* ed è stato implementato su piattaforma Microsoft Visual Studio 2005, puntando alle librerie di riferimento *odbc32.lib odbccp32.lib* e *glut32.lib.*

[23] Le parti di codice precedute dai simboli // commentano le funzioni che precedono e ne spiegano le utilità.

```
 glRotatef(-90.0,0.0,1.0,0.0);
  glBegin(GL_POLYGON);
  glVertex3f(0.0,0.0,0.0);
  glVertex3f(0.5,0.0,0.0);
  glVertex3f(0.5,0.5,0.0);
  glVertex3f(0.0,0.5,0.0);
 glEnd();
glPopMatrix();
glPushMatrix();
glColor3f(1.0,1.0,0.0);
 glRotatef(-90.0,0.0,1.0,0.0);
 glTranslatef(0.0,0.0,-0.5);
  glBegin(GL_POLYGON);
  glVertex3f(0.0,0.0,0.0);
  glVertex3f(0.5,0.0,0.0);
  glVertex3f(0.5,0.5,0.0);
  glVertex3f(0.0,0.5,0.0);
 glEnd();
glPopMatrix();
glColor3f(0.0,1.0,0.0);
 glBegin(GL_POLYGON);
  glVertex3f(0.5,0.5,0.0);
  glVertex3f(0.5,0.5,0.5);
  glVertex3f(0.25,0.6,0.5);
  glVertex3f(0.25,0.6,0.0);
 glEnd();
 glBegin(GL_POLYGON);
  glVertex3f(-0.5,0.5,0.0);
  glVertex3f(-0.5,0.5,0.5);
  glVertex3f(-0.25,0.6,0.5);
  glVertex3f(-0.25,0.6,0.0);
 glEnd();
glColor3f(0.0,0.0,1.0);
 return;
}
```

```
//Funzione assegnata ai menu di selezione
void ProcessMenu(int value)
 {
    iShape = value;
 glutPostRedisplay();
 }
```

```
// Disegna la scena
void Disegna(void)
 {
 // Pulisce la finestra prima di disegnare
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
 // Applica le trasformazioni geometriche (rotazioni)
 // al sistema di riferimento in cui disegna l'oggetto
 glPushMatrix();
 glRotatef(xRot, 1.0f, 0.0f, 0.0f);
 glRotatef(yRot, 0.0f, 1.0f, 0.0f);
    switch(iShape)
        {
        case 1: glutWireSphere(1.0f, 25, 25);break;
        case 2: glutWireCube(1.0f);break;
        case 3: glutWireCone(0.30f, 1.1f, 20, 20);break;
        case 4: glutWireTorus(0.3f, 1.0f, 10, 25);break;
        case 5: glutWireDodecahedron(); break;
        case 6: glutWireOctahedron(); break;
        case 7: glutWireTetrahedron(); break;
        case 8: glutWireIcosahedron(); break;
        case 9: glutWireTeapot(1.0f);  break;
```

```
        case 11:glutSolidSphere(1.0f, 25, 25); break;
        case 12:glutSolidCube(1.0f); break;
        case 13:glutSolidCone(0.30, 1.1f, 20, 20);break;
        case 14:glutSolidTorus(0.3f, 1.0f, 10, 25);break;
        case 15:glutSolidDodecahedron(); break;
        case 16:glutSolidOctahedron(); break;
        case 17:glutSolidTetrahedron(); break;
        case 18:glutSolidIcosahedron(); break;
        case 19: casa();break;
        default:
            glutSolidTeapot(1.0f);
            break;
        }
 // Restituisce le trasformazioni
 glPopMatrix();
 //Visualizza il buffer disegnato scambiandolo con quello corrente
 glutSwapBuffers();
 }
```

```
//Inizializza il contesto di rendering
void Inizializzazione()
 {
 // Background bianco
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f );
 // Attiva il Depth Testing
    glEnable(GL_DEPTH_TEST);
 // Attiva la luce lighting
    glEnable(GL_LIGHTING);
 // Imposta e attiva light 0
 glLightfv(GL_LIGHT0,GL_AMBIENT,ambientLight);
 glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuseLight);
 glLightfv(GL_LIGHT0,GL_SPECULAR,specular);
 glEnable(GL_LIGHT0);
 // Attiva il color tracking
 glEnable(GL_COLOR_MATERIAL);
 // Imposta le proprietà del materiale
 glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
 // All materials hereafter have full specular reflectivity
 // with a high shine
 glMaterialfv(GL_FRONT, GL_SPECULAR,specref);
 glMateriali(GL_FRONT,GL_SHININESS,128);
 // Imposta il colore di disegno sul blue
 glColor3ub(0, 0, 255);
 }
```

```
void SpecialKeys(int key, int x, int y)
 {
 if(key == GLUT_KEY_UP)  xRot-= 5.0f;
 if(key == GLUT_KEY_DOWN) xRot += 5.0f;
 if(key == GLUT_KEY_LEFT) yRot -= 5.0f;
 if(key == GLUT_KEY_RIGHT) yRot += 5.0f;
 if(key > 356.0f)  xRot = 0.0f;
 if(key < -1.0f)   xRot = 355.0f;
 if(key > 356.0f)  yRot = 0.0f;
 if(key < -1.0f)   yRot = 355.0f;
 // Aggiorna la finestra
 glutPostRedisplay();
 }
```

```
void Ridimensionamento(int w, int h)
 {
 GLfloat  lightPos[] = { -50.f, 50.0f, 100.0f, 1.0f };
 GLfloat nRange = 1.9f;
 // Prevent a divide by zero
 if(h == 0)
```

```
  h = 1;
 // Imposta la Viewport alle dimensioni della finestra
    glViewport(0, 0, w, h);
 // Azzera lo stack della matrice di proiezione
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 // Definisce il volume di clipping (Sx, Dx, bottom, top, near,
far)
    if (w <= h)
  glOrtho (-nRange, nRange, -nRange*h/w, nRange*h/w, -nRange,
nRange);
    else
  glOrtho (-nRange*w/h, nRange*w/h, -nRange, nRange, -nRange,
nRange);
 //Azzera lo stack della matrice ModelView
 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();
 glLightfv(GL_LIGHT0,GL_POSITION,lightPos);
 }
```

```
int main(int argc, char* argv[])
 {
 int nSolidMenu;
 int nWireMenu;
 int nMainMenu;
//INIZIALIZZAZIONE DELLA FINESTRA CON I COMANDI GLUT
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
 glutInitWindowSize(640,480);
 glutInitWindowPosition(50,50);
 glutCreateWindow("GLUT");

//CREAZIONE DI MENU E REGISTRAZIONE DELLA FUNZIONE DI
//ELABORAZIONE DEI MENU
 nWireMenu = glutCreateMenu(ProcessMenu);
 glutAddMenuEntry("SFERA",1);
 glutAddMenuEntry("CUBO",2);
 glutAddMenuEntry("CONO",3);
 glutAddMenuEntry("TORO",4);
 glutAddMenuEntry("DODECAEDRO",5);
 glutAddMenuEntry("OTTAEDRO",6);
 glutAddMenuEntry("TETRAEDRO",7);
 glutAddMenuEntry("ICOSAEDRO",8);
 glutAddMenuEntry("Teapot",9);
 nSolidMenu = glutCreateMenu(ProcessMenu);
 glutAddMenuEntry("SFERA",11);
 glutAddMenuEntry("CUBO",12);
 glutAddMenuEntry("CONO",13);
 glutAddMenuEntry("TORO",14);
 glutAddMenuEntry("DODECAEDRO",15);
 glutAddMenuEntry("OTTAEDRO",16);
 glutAddMenuEntry("TETRAEDRO",17);
 glutAddMenuEntry("ICOSAEDRO",18);
 glutAddMenuEntry("Casa",19);
 glutAddMenuEntry("Teapot",20);
 nMainMenu = glutCreateMenu(ProcessMenu);
 glutAddSubMenu("WireFrame", nWireMenu);
    glutAddSubMenu("Solido", nSolidMenu);
 glutAttachMenu(GLUT_RIGHT_BUTTON);
 glutReshapeFunc(Ridimensionamento);
 glutSpecialFunc(SpecialKeys);
 glutDisplayFunc(Disegna);
 Inizializzazione();
```

```
glutMainLoop();
return 0;
}
```

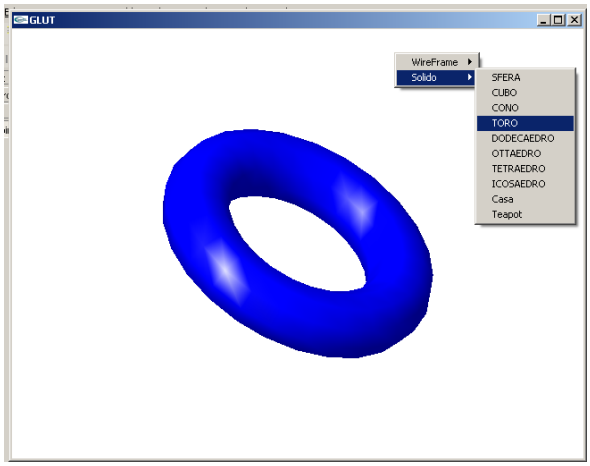


Figura A. 1: Esempio di grafica tridimensionale utilizzando librerie grafiche OpenGL e GLUT.

Nel codice riportato, nella funzione *main*, si distinguono quattro sottosezioni principali: la prima riguarda l'inizializzazione e la creazione della finestra, la seconda registra le funzioni di tipo *callback*[24] per la gestione degli eventi, la terza definisce un menu di tipo a tendina attivabile con il tasto destro del mouse e l'ultima sezione innesca il ciclo principale attraverso la funzione `glutMainLoop`. Tale comando rende visibile la finestra sullo schermo ed esegue in automatico il seguente ciclo:

- Attende un evento al quale sia stata associata una *callback*;
- Attiva la *callback* corrispondente;

Tra tutte le funzioni di gestione degli eventi, la registrazione della funzione di disegno della scena è obbligatoria. A tal fine si utilizza nel *main* il comando:

```
glutDisplayFunc(f);
```

dove `f` è la funzione di disegno della finestra che viene richiamata ad ogni ciclo o tutte le volte che la finestra subisce un ridimensionamento od una parziale occlusione da altre finestre.

Le altre funzioni per la gestione degli eventi sono facoltative e si può scegliere tra le seguenti:

- **Reshape**: si attiva quando la finestra cambia dimensioni, tipicamente per intervento dell'utente.

[24] Le *Callback* sono funzioni eseguite al generarsi di un evento specifico. Sono assegnate all'evento stesso mediante registrazione nella funzione principale del programma (*main*).

- **Keyboard**: si attiva quando l'utente preme un carattere della tastiera avendo il focus sulla finestra (solo i caratteri "scrivibili", sono esclusi i tasti speciali).
- **Special**: si attiva quando l'utente preme un tasto speciale (tasti funzione, frecce, ecc.) avendo il focus sulla finestra.
- **Mouse**: si attiva quando l'utente preme o rilascia un bottone del mouse con il puntatore dentro la finestra.
- **Motion**: si attiva quando l'utente muove il mouse all'interno della finestra, con uno dei bottoni premuto.
- **PassiveMotion**: si attiva quando l'utente muove il mouse all'interno della finestra, senza nessun bottone premuto.
- **Entry**: si attiva quando il mouse entra od esce dalla finestra.
- **Idle**: si attiva "continuamente", cioè ad ogni ciclo del main loop, serve per creare animazioni. Per esempio: sposta l'oggetto e lo ridisegna (essendo chiamata continuamente, produce l'immagine di un oggetto in movimento).
- **Visibility**: si attiva quando lo stato della finestra passa da visibile a non visibile o viceversa, tipicamente usata per disattivare l'animazione quando la finestra non è visibile.
- **Timer**: si attiva dopo un certo numero di millisecondi.
- **MenuStatus**: si attiva quando un menu' pop-up è attivato o disattivato, tipicamente usata per sospendere l'animazione mentre l'utente sta interagendo con il menu.

Le funzioni da utilizzare come *Callback* devono essere scritte dal programmatore prevedendo specifici parametri come argomenti. Il numero, tipo e significato degli argomenti dipendono dal tipo di evento e sono generati automaticamente, in fase di esecuzione del programma, e passati alla funzione al verificarsi dell'evento ad essa associato[25].

Al fine di avere un quadro completo sull'utilizzo delle principali funzioni di *callback* si analizzano di seguito le funzioni principali.

A3. Le callback relative al disegno

Come è stato anticipato nella sezione precedente la funzione di disegno è indispensabile per la visualizzazione della finestra grafica. La sintassi della funzione è la seguente:

```
void Disegna(void);
```

[25] Se X è l'evento da controllare con la funzione *F* allora il comando `glutXFunc(F)` registra la funzione *F* come funzione di controllo per quell'evento mentre `glutXFunc(NULL)` cancella l'associazione.

e contiene il codice OpenGL che genera le primitive da proiettare sulla finestra grafica. Tale funzione può essere richiamata anche da programma, mediante istruzione apposita, per ridisegnare la scena dopo averne cambiato i contenuti:

- `glutDisplayFunc(Disegna);` per associare la funzione *Disegna* all'evento di visualizzazione;
- `glutPostReDisplay();` per richiamarla da programma, dopo averla associata all'evento di visualizzazione.

Tutto il codice OpenGL trova posto nel blocco della funzione ed è consigliato impostare i comandi di disegno e scenografia della scena solo in questa sezione del programma. Tipicamente il corpo della display callback contiene:

- **Abilitazioni delle funzionalita' OpenGL** (`glEnable`) Alcune funzioni OpenGL computazionalmente costose non sono abilitate, le abilita il programmatore solo se servono per la scena che vuole disegnare. Per esempio:
  `glEnable(GL_DEPTH_TEST);` abilita l'eliminazione delle superfici nascoste e viene utilizzato soprattutto per disegnare scene tridimensionali.
  `glEnable(GL_LIGHTING);` abilita l'illuminazione nel caso si voglia produrre una scena illuminata.
  `glEnable(GL_LIGHTi);` con i=0,1,2... abilita l'i-esima luce (dipende da quante luci si vogliono attivare)
- **Pulizia del foglio** (`glClear`)
  Cancella il contenuto dei buffer dove si andra' a disegnare. L'argomento è una lista di costanti intere messe in OR bit-a-bit. Per esempio:
  `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BIT);` pulisce color buffer e depth buffer
- **Trasformazione di proiezione**
  Stabilisce quale parte della scena viene inquadrata per produrre l'immagine (volume di vista). In OpenGL è gestita da una matrice 4x4 chiamata projection matrix. La trasformazione di default, rappresentata dalla matrice identita', corrisponde ad una proiezione ortografica che ha come volume di vista il cubo con diagonale (-1,-1,-1)-(1,1,1). La trasformazione di default viene caricata con le seguenti istruzioni:
  ```
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  ```
- **Trasformazione di vista**
  Stabilisce la posizione del punto di vista da cui guardiamo la scena. In OpenGL è gestita da una matrice 4x4 chiamata modelview. La trasformazione di default, rappresentata dalla matrice identita', mette il punto di vista in (0,0,0) e rivolto verso la direzione negativa dell'asse z.
  ```
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  ```
- **Assegnazione degli attributi e chiamata delle primitive**.
  Prima di chiamare una primitiva occorre impostarne le proprietà assegnando il valore desiderato a tutti gli attributi rilevanti.
  `glPointSize(float);` stabilisce la grandezza del marchio usato per disegnare i

punti, per default 1.0
`glLineWidth(float);` stabilisce lo spessore del tratto usato per disegnare le linee, per default 1.0
`glPolygonMode(unsigned int, unsigned int);` stabilisce il modo di tracciamento dei poligoni. Il primo argomento è la faccia: `GL_FRONT`, `GL_BACK`, oppure `GL_FRONT_AND_BACK`.
Il secondo argomento è il modo: `GL_FILL` = pieno (default), `GL_LINE` = solo i lati, oppure `GL_POINT` = solo i vertici.
`glColor` in assenza di illuminazione, stabilisce il colore. Ha varie forme a seconda di come si vogliono passare i parametri.

Dopo aver impostato le proprietà della scena si passa alla fase di definizione delle primitive geometriche da proiettare:

```
glBegin(int tipo_primitiva);
...
glEnd();
```

dove tra la coppia `glBegin`, `glEnd` trovano posto i vertici. Per specificare un vertice si usa `glVertex` seguito da un numero e una lettera che specificano il formato con il quale vengono passati i parametri. In alternativa si possono disegnare entità geometriche implementate nelle GLUT, ad esempio si può creare una sfera con il comando `glutSolidSphere(…)` o un cubo con il comando `glutSolidCube(…)`.

- **Scambio di front e back buffer**
  Essendo la finestra double buffer, il disegno tracciato nel back buffer, viene reso visibile sullo schermo (front buffer), con le istruzioni:

```
glFlush();
glutSwapBuffers();
```

A4. Le callback per il ridimensionamento delle finestra

Attraverso il comando:

```
glutReshapeFunc(ChangeSize);
```

si associa all'evento di variazione delle dimensioni della finestra la funzione `ChangeSize`, che deve avere una sintassi del tipo:

```
void ChangeSize(int w,int h);
```

la quale specifica con i parametri *w* e *h* le dimensioni della finestra grafica, denominata *Viewport*. Il corpo della funzione *Reshape* contiene il comando:

```
glViewport(x0,y0,w,h);
```

dove i parametri definiscono la posizione e le dimensioni in pixels della finestra di disegno.

A5. Le callback associate a tastiera e mouse

Con il comando `glutKeyboardFunc(Tastiera)` si registra la funzione *Tastiera* che gestisce l'evento generato dall'utente quando preme un tasto stampabile della tastiera del pc. La funzione tastiera deve avere la seguente sintassi:

```
void Tastiera(unsigned char c,int x, int y);
```

dove l'argomento *c* rappresenta il carattere premuto mentre *x* e *y* la posizione del mouse in pixel nel momento in cui è stato premuto.
Ai tasti funzione, che non sono stampabili, è riservata una specifica funzione per la registrazione: `glutSpecialFunc(f),` con la funzione *f* che deve avere la seguente sintassi:

```
void f(int k,int x, int y);
```

dove in *k* si registra il codice ASCII del tasto premuto ed *x*, *y* memorizzano la posizione del mouse sullo schermo.
Per gli eventi generati dal mouse le GLUT restituiscono quattro valori: il primo è associato all'evento di pressione dei tasti del mouse e può assumere tre valori: `GLUT_LEFT_BUTTON,GLUT_MIDDLE_BUTTON e GLUT_RIGHT_BUTTON,` in funzione del tasto premuto. Il secondo parametro, della funzione di controllo del mouse, restituisce lo stato dei tasti: premuto o rilasciato, con i valori rispettivamente `GLUT_DOWN, GLUT_UP.`
Infine la funzione di controllo contiene le informazioni relative alla posizione del mouse sullo schermo attraverso le variabili *x* e *y*.
Tale funzione ha la seguente sintassi:

```
void mouse(int b,int s,int x, int y);
```

e deve essere registrata nel main con il comando `glutMouseFunc(mouse)`.
Esistono infine altre tre funzioni di controllo per gli eventi connessi all'utilizzo del mouse:

- `glutMotionFunc(f)` : associa all'applicazione la funzione `f(x,y)` a due argomenti di tipo intero *x* e *y* che registrano la posizione (in pixel) del puntatore del mouse sullo schermo, quando nessun tasto è premuto;

- `glutPassiveMotionFunc(f)` : associa all'applicazione la funzione `f(x,y)` a due argomenti di tipo intero `x` e `y` che registrano la posizione (in pixel) del puntatore del mouse sullo schermo, quando almeno un tasto è premuto;
- `glutEntryFunc(f)`: registra una funzione `f` ad un solo argomento di tipo intero *s*, che può assumere i valori `GLUT_ENTERED` e `GLUT_LEFT` rispettivamente quando il puntatore del mouse è all'interno o all'esterno della finestra grafica.

A6. Callback relative all'animazione, al tempo e ai menu

Con il comando `glutIdleFunc(f)` è possibile associare una funzione `f` richiamata ciclicamente in fase di esecuzione, che restituisce un valore intero. Tale valore memorizza lo stato di visibilità della finestra.

Spesso i contenuti di animazione di una scena sono vincolati al tempo. Le GLUT consentono di registrare una funzione associata al trascorrere del tempo che viene chiamata ad intervalli costanti.

Il comando utilizzato per la registrazione è `glutTimerFunc(k,f,v)`, dove `k` è l'intervallo in millisecondi e `v` il valore passato alla funzione `f`.

Le GLUT permettono inoltre di registrare un menu pop-up[26] gerarchico (anche detto "a cascata") associato ad bottone del mouse in una finestra. Una variante del menu semplice è il menu a cascata che contiene più livelli di sottomenu associati alle voci del menu principale.

La gerarchia di menu è rappresentata da un albero. La radice è il menu principale che appare cliccando con il mouse sulla finestra (con il bottone prestabilito). Se una voce di menu chiama un sottomenu, il sottomenu è figlio del menu che lo chiama (il quale può essere sottomenu di un altro). Le foglie sono menu che non ne chiamano altri.

La gerarchia si costruisce dal livello più basso iniziando dalle foglie con i seguenti comandi:

- `glutCreateMenu` crea un menu (senza voci e indipendente dalla gerarchia) e restituisce un valore intero associato alla voce di menu;
- `glutAddMenuEntry` aggiunge una voce a un menu;

[26] Il pop-up è il menù che appare sovrapposto alla finestra quando si preme il tasto del mouse e scompare al rilascio dello stesso. Permette l'interazione con l'applicazione attraverso la selezione delle voci in elenco.

- `glutAddSubMenu` aggiunge una voce che invoca un sottomenu (collega un menu alla gerarchia rendendolo sottomenu di un altro);
- `glutAttachMenu` collega il menu (quello a radice della gerarchia) ad un tasto del mouse;

Per una finestra si possono avere fino a tre gerarchie di menu, ciascuna con la radice collegata a un diverso bottone del mouse. La funzione relativa alla creazione del menu è:

```
int glutCreateMenu(f);
```

attraverso la quale si crea un menu avente la funzione `f` come callback, che ritorna un identificatore per il menu. Il menu appena creato diventa il menu corrente, tutte le successive istruzioni di manipolazione di menu si riferiscono a quello corrente.
La funzione `f` è incaricata di eseguire le azioni corrispondenti alle varie voci.
È una funzione a un argomento del tipo:

```
void f (int i);
```

Quando l'utente seleziona la voce i-esima del menu, Glut chiama automaticamente la funzione `f` con argomento `i`.
Tipicamente il corpo di `f` contiene uno switch a seconda del valore di `i`.
Per l’aggiunta di voci ordinarie al menu si utilizza il comando:

```
void glutAddMenuEntry(char * label, int v);
```

che consente di aggiungere una voce al menu corrente, dove `label` è la stringa che appare sulla corrispondente voce di menu e `v` è il valore che Glut passa alla callback a seguito della selezione della voce di menu.
Il collegamento di voci di sottomenu al menu principale si effettua con il comando:

```
void glutAddSubMenu(char * label, int m);
```

la quale aggiunge una voce che invoca un altro menu (sottomenu). Il sottomenu appare quando l'utente scorre con il mouse su tale voce. La variabile `label,` della funzione è la stringa che apparirà sulla voce di menu mentre la variabile `m` è l'identificatore ritornato da `glutCreateMenu` quando è stato creato il sottomenu:

Il menu radice:

```
void glutAttachMenu(int b);
```

collega il menu corrente al bottone del mouse passato in argomento, `b` è `GLUT_LEFT_BUTTON`,`GLUT_MIDDLE_BUTTON`, oppure `GLUT_RIGHT_BUTTON`. Il menu pop-up appare premendo tale bottone. La Menu Status callback è chiamata da Glut quando attiva o disattiva un menu pop-up.
La menu callback è una funzione a tre argomenti, di tipo:

```
void f(int s, int x, int y)
```

dove `s` è lo stato attuale: `GLUT_MENU_IN_USE` se qualche menu è popped-up, `GLUT_MENU_NOT_IN_USE` altrimenti. Invece le variabili x,y rappresentano la posizione del mouse quando il menu è stato invocato o rilasciato e viene associata con la funzione:

```
glutMenuStatusFunc(f);
```

Altre funzioni utili per l'impostazione e la gestione dei menu sono le funzioni:
`int glutGetMenu(void);` ritorna il menu corrente.
`glutSetMenu(int m);` assegna il menu corrente.
`glutDetachMenu(int b);` sgancia il menu attaccato al bottone `b` del mouse.

A7. Trasformazioni di coordinate in OpenGL

Una caratteristica importante delle applicazioni OpenGL riguarda la sequenza delle trasformazioni geometriche (v. Figura A. 2 e Figura A. 3), applicate ai vertici delle geometrie disegnate, per visualizzare la scena virtuale secondo la prospettiva desiderata. La logica di proiezione è assimilabile a quella di una macchina fotografica (o telecamera nel caso di animazioni) che punta sulla scena disegnata.

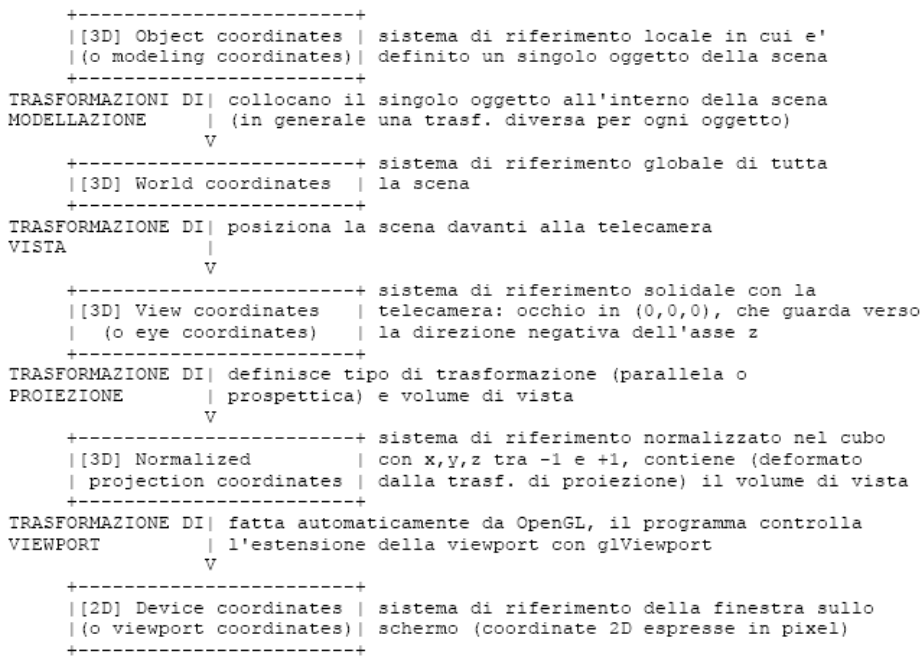


Figura A. 2. Sequenza di trasformazioni geometriche applicata dalle OpenGL alle primitive geometriche.

Una scena OpenGL è paragonabile ad un contenitore di oggetti, ciascuno dei quali può essere aggregato o contenere altri oggetti. La gerarchia di aggregazione o appartenenza ad un gruppo è determinata dalla sequenza di trasformazioni geometriche e dall'ordine di inserimento dei singoli oggetti nella scena.

Gli oggetti elementari sono a loro volta costituiti da altri elementi: le primitive geometriche. Ciascuna primitiva è definita attraverso i suoi vertici nel sistema di coordinate locale. È appena il caso di far notare come i sistemi di riferimento locali di primitive diverse siano indipendenti tra loro e vengano introdotti grazie alle trasformazioni di modellazione.

La proiezione prospettica, degli oggetti da disegnare, viene praticata dalle OpenGL mediante una sequenza definita di trasformazioni geometriche applicate ai vertici delle primitive.

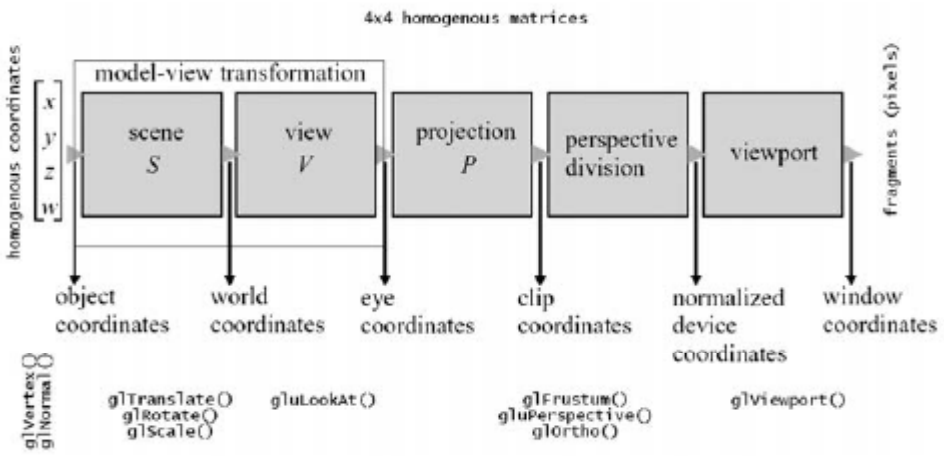


Figura A. 3. Schematizzazione delle trasformazioni geometriche applicate ai vertici di una scena OpenGL (OpenGL rendering transformation pipeline).

La sequenza di trasformazioni è schematizzata in Figura A. 3. Le OpenGL, partendo dal codice utilizzato per definire gli oggetti da disegnare (v. funzione *Disegna* della Tabella 3.6.2), attraverso la prima sequenza di trasformazioni (*Trasformazioni di Modellazione*) inseriscono l'oggetto nel sistema di riferimento assoluto della scena (*coordinate del mondo*), che è unico. Con la dicitura trasformazioni di modellazione ci si riferisce a:

1. Traslazioni
2. Rotazioni
3. Scalature

Una volta collocati gli oggetti all'interno della scena si passa a creare un'immagine della scena secondo un punto di vista ben definito, a tal fine occorre eseguire le seguenti operazioni:

- Posizionare il punto di vista rispetto alla scena;
- Impostare la focale;
- Scegliere la dimensione in pixel dell'immagine;

Alla macchina fotografica virtuale viene assegnato un sistema di coordinate (*coordinate di vista*) con l'origine nella posizione del punto di vista, l'asse *z* opposto alla direzione verso cui la macchina è puntata, e asse *y* coincidente in direzione e verso con la verticale dell'osservatore (Figura A. 4).

Le *Trasformazioni di Vista* portano la scena da coordinate del mondo a coordinate di vista e compiono traslazioni e rotazioni.

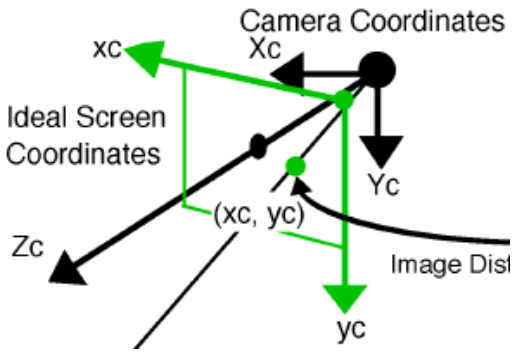


Figura A. 4: Il sistema di coordinate associato al punto di vista della camera.

Le *Trasformazioni di Proiezione* definiscono, allo stesso modo di una lente fotografica, l'estensione del campo visivo e la distorsione di proiezione. Tali effetti vengono applicati nelle OpenGL attraverso il volume di vista. La parte di scena inquadrata è quella dentro il volume di vista e la forma di tale volume ne definisce la distorsione.

In altri termini le trasformazioni di proiezione portano il volume di vista a coincidere con il cubo di diagonale (-1,-1,-1) –(1,1,1). Il sistema di coordinate interno al cubo è denominato sistema di *coordinate normalizzate di proiezione* o *NPC*. Per ottenere l'effetto prospettico desiderato si applicano deformazioni al volume di vista della scena. Esistono due tipi di trasformazione la trasformazione di tipo *ortografica*, che lascia immutate le proporzioni tra le diverse parti della scena, e la trasformazione *prospettica* che definiscono un tronco di piramide con base minore verso l'osservatore, dove le parti distanti della scena appaiono scalate rispetto a quelle vicine.

Per le trasformazioni di proiezione si utilizzano i comandi:

- `glOrtho(left,right,bottom,top,near,far);`Specifica una matrice ortografica.Il volume di vista è il parallelepipedo di diagonale (left,bottom,-near) - (right,top,-far). Per avere un volume di vista giacente davanti all'occhio, near e far devono essere positivi.
- `glFrustum(left,right,bottom,top,near,far);` Specifica matrice prospettica. Il volume di vista è un tronco di piramide con l'apice in (0,0,0) (che rappresenta il punto di vista sulla scena), base minore il rettangolo di diagonale (left,bottom) - (right,top) giacente sul piano z=-near, e base maggiore sul piano z=-far (ottenuta effettuando una proiezione da (0,0,0) della base minore sul piano z=-far).
- `gluPerspective(fovy,aspect,znear,zfar);` è un'alternativa a `glFrustum`, e accetta parametri più intuitivi. Per questa funzione si definisce la trasformazione

prospettica, dove fov (*field of view*) è l'angolo di apertura della piramide in gradi, aspect è il rapporto largezza/altezza della base del tronco di piramide, che tiene conto della deformazione di vista provocato dalle dimensioni non omogenee della finestra (in genere coincide con il rapporto larghezza/altezza della finestra). Infine i parametri znear e zfar che rappresentano la distanza delle due basi del tronco di piramide dalpunto di vista.

La *Trasformazione di ViewPort* definisce il numero di righe e colonne del frame buffer corrispondente alle dimensioni in pixel della finestra sullo schermo e genera l'immagine proiettando la scena nello spazio bidimensionale del monitor. L'operazione prevede:

- La scalatura delle coordinate xy del cubo *NPC* per farle coincidere con le dimensioni del frame buffer e la discretizzazione in pixel;
- Lo schiacciamento delle coordinate z del cubo *NPC*;

Se sullo stesso pixel vengono proiettate più primitive, di default vince quella disegnata per ultima nel codice OpenGL, se invece è abilitata l'eliminazione delle parti nascoste (glEnable(GL_DEPTH_BUFFER)) vince quella con z più vicina all'osservatore. Il confronto considera le z discretizzate su tanti bit quanta la profondità del depth buffer.

Le trasformazioni possono essere specificate dal programmatore attraverso due matrici:

- Matrice di modellazione *ModelView*
- Matrice di proiezione *Projection*

Tali matrici sono espresse in coordinate omogenee (4x4) e moltiplicano tutti i vertici delle primitive da visualizzare nella scena.

Per modificare una delle matrici occorre specificare su quale matrice si vuole agire, mediante il comando `glMatrixMode()`passando come argomenti:

- `GL_MODELVIEW,` se si vuole agire sulle trasformazioni di modellazione;
- `GL_PROJECTION,` se si vuole agire sulle trasformazioni di proiezione;

OpenGL fornisce inoltre funzioni dedicate alla modifica delle matrici di proiezione e modellazione. Nella fase di programmazione e in funzione degli scopi è possibile impostare per valori le matrici di trasformazione e caricarle nello stack delle trasformazioni oppure utilizzare una serie di comandi elementari.

Nel caso si voglia impostare manualmente la trasformazione[27] occorre definire una matrice 4x4 del tipo:

[27] Tale procedura è spesso utilizzata per visualizzare simulazioni cinematiche degli oggetti virtuali [34].

$$[M] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (1.1)$$

dove la sotto-matrice 3x3 (con elementi r_{ij}) è una matrice di rotazione e i primi tre elementi della quarta colonna rappresentano i componenti del vettore traslazione. I valori della matrice *M* vanno ordinati su un vettore *v* nel modo seguente:

$$\{m\} = \{r_{11} \quad r_{21} \quad r_{31} \quad 0 \quad r_{12} \quad r_{22} \quad \ldots \quad 0 \quad 0 \quad 0 \quad 1\} \qquad (1.2)$$

e passati come argomento alle funzioni `glMultMatrixd(m)`, `glMultMatrixf(m)` o alla funzione `glLoadMatrix(m)`.

Tabella 3.6.3. Trasformazioni geometriche elementari in OpenGL.

`glTranslatef(dx,dy,dz)`	$[M_{Trasl.}] = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$
`glRotatef(ang,rx,ry,rz)`	$[M_x] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(ang) & -\sin(ang) & 0 \\ 0 & \sin(ang) & \cos(ang) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, $[M_y] = \begin{bmatrix} \cos(ang) & 0 & \sin(ang) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(ang) & 0 & \cos(ang) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, $[M_z] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(ang) & -\sin(ang) & 0 \\ 0 & \sin(ang) & \cos(ang) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

`glScalef(sx,sy,sz)`	$[M_{scala}] = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Nel primo caso la matrice viene unita alla coda di matrici e le trasformazione finale sarà il risultato della composizione delle trasformazioni. Nel secondo caso la matrice viene caricata nella sequenza delle trasformazioni al posto della matrice di modellazione o di proiezione.

Tuttavia nel caso in cui la disposizione degli oggetti nella scena non richieda trasformazioni particolari, è possibile posizionare le primitive utilizzando i comandi riportati nella Tabella 3.6.3.

Anche per le trasformazioni elementari, sopraelencate, esiste la possibilità di applicare trasformazioni indipendenti per ciascun oggetto. La procedura, equivalente al metodo `glLoadMatrix`, prevede la definizione di gruppi di matrici che memorizzano le trasformazioni per il singolo oggetto. Dal punto di vista dell'implementazione è necessario inserire il codice relativo al disegno e al posizionamento di un oggetto *i* nella scena tra le funzioni `glPushMatrix()` e `glPopMatrix()`, le quali isolano le trasformazioni applicate all'oggetto i-esimo. Un esempio di tale procedura è riportato nella Tabella 3.6.2 per la funzione `casa()` utilizzata per la costruzione semplificata a superfici della sagoma tridimensionale di una casa.

Anche la trasformazione di vista rientra nelle trasformazioni di modellazione e si riferisce al posizionamento del punto di vista rispetto alla scena da proiettare. Sono applicabili a questo tipo di trasformazione tutte le funzioni appena viste per il posizionamento degli oggetti. Tenendo conto della posizione relativa degli oggetti rispetto alla scena, si spostano gli oggetti e non il punto di vista. In alternativa esiste un comando `gluLookAt(…)` che consente di impostare velocemente il punto di vista della telecamera virtuale delle OpenGL rispetto alla scena da visualizzare.

L'ordine seguito dalle OpenGL per l'applicazione delle trasformazioni è inverso rispetto all'ordine con il quale le trasformazioni vengono implementate. Pertanto l'ordine delle trasformazioni riportato in Figura A. 3 viene definito in senso inverso nel listato dell'applicazione. A titolo si riporta, in Tabella 3.6.4, il layout della sequenza di istruzioni da scrivere per impostare correttamente la scena.

Tabella 3.6.4. Listato esempio per l'impostazione delle trasformazioni di modellazione e proiezione.

```
…
// definzione della finestra
glViewport(...);
// attivazione della matrice di proiezione
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
// trasformazioni di proiezione
glFrustum(...); // o  gluPerspective(...); o
glOrtho(...);
// attivazione della matrice di modellazione
// trasformazioni di proiezione
glBegin(…) …
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
// impostazione del punto di vista
gluLookAt(...);
// trasformazioni applicate alla scena
glTranslate(...);
glRotate(...);
glScale(...);
//comandi di disegno
…
```

A8. Trasformazioni inverse

La corrispondenza tra i punti nello spazio virtuale della scena e i pixel sullo schermo è una relazione fondamentale per consentire l'interazione dell'utente con la scena. In tale ambito il mouse è un puntatore bidimensionale che consente di selezionare punti all'interno della finestra grafica, mentre si affida alle funzioni OpenGL il compito di trovare la relazione tra il punto selezionato e il corrispondente punto nello spazio.

Il problema quindi si riconduce alla determinazione della trasformazione inversa che partendo dalle coordinate in pixel del punto restituisce le coordinate di modellazione $\{x \quad y \quad z\}^t$.

OpenGL consente di ricavare i valori delle trasformazioni perchè memorizza le trasformazioni di modellazione, di vista, di proiezione e visualizzazione in tre variabili di stato:

- Viewport, che controlla la trasformazione di visualizzazione applicata con glViewPort;
- Projection Matrix, che controlla la trasformazione di proiezione assegnata in modalità GL_PROJECTION;
- Modelview Matrix: che controlla le trasformazioni di proiezione e di vista assegnate in modalità GL_MODELVIEW;

Per la trasformazione inversa OpenGL mette a disposizione la funzione `gluUnProject`, che esegue la traduzione inversa da coordinate della finestra in coordinate di modellazione.
Tale funzione riceve in input la viewport e le due matrici projection e modelview che sono state usate nella traduzione diretta rispettando la seguente sintassi:

```
gluUnProject(x, y, z,
             modelview_mat,
             projection_mat,
             viewport,
             &xx, &yy, &zz);
```

dove:

- `(x,y)` sono le coordinate del punto nella finestra;
- `z` è la profondità se l'obiettivo è quello di disegnare una scena 3D in una finestra con z-buffer. `x,y,z` sono espresse come GLdouble e rappresentano le coordinate del punto nello spazio;
- `modelview_mat` e `projection_mat` sono matrici 4x4 di elementi GLdouble che contengono le due matrici omonime;
- `viewport` è un array di 4 elementi GLint che descrivono la viewport, nell'ordine (x0,y0,w,h);

- `xx,yy,zz`, di tipo GLdouble, rappresentano le coordinate del punto nello spazio che restituiscono le coordinate di modellazione[28]

le due matrici Projection e ModelView sono parte dello stato OpenGL e si possono ricavare con l'istruzione `glGet,` che esiste in forme diverse a seconda del tipo dei parametri di stato richiesti.
Per le variabili di stato a valori interi si utilizza la sintassi:

```
glGetIntegerv(nome, ptr);
```

dove:

- `nome` è la costante OpenGL che identifica il parametro di stato cercato;
- `ptr` è un array di tipo GLint di dimensione adeguata per contenere il risultato;

Un'applicazione tipica dei metodi appena riportati riguarda il calcolo dei parametri della viewport corrente. Definendo una variabile del tipo:

```
GLint viewport[4];
```

i valori cercati si ottengono con la seguente istruzione:

```
glGetIntegerv(GL_VIEWPORT, viewport);
```

Per variabili di stato a valori double si utilizza la funzione `glGetDoublev` rispettando la seguente sintassi:

```
glGetDoublev(nome, ptr);
```

dove:

[28] Le coordinate della finestra hanno y=0 in alto, mentre OpenGL le considera con y=0 in basso. Quindi se la posizione del cursore è `(wx,wy)` abbiamo `x=wx` e `y=viewport[3]-wy.`

- `nome` è una costante OpenGL che identifica il parametro di stato cercato;
- `ptr` è un array di GLdouble della giusta dimensione per registrare il risultato;

Per risalire alla matrice di proiezione, si richiama il metodo `glGetDoublev` con le seguenti istruzioni:

```
GLdouble projection_mat[16];
glGetDoublev(GL_PROJECTION_MATRIX, projection_mat);
```

Il metodo che esegue la trasformazione inversa ovvero spazio-finestra è il metodo `gluProject`, esegue la trasformazione opposta di `gluUnProject` ma ha gli stessi argomenti:

```
gluProject(x, y, z,
            modelview_mat,
            projection_mat,
            viewport,
            &xx, &yy, &zz);
```

Nota la funzionalità del metodo `gluUnproject`, che permette di risalire alle coodinate del punto nello spazio virtuale corrispondente a quello selezionato nella finestra grafica, il problema dell'interazione con la scena si riconduce all'implementazione di una procedura per la gestione dei vertici che fanno parte della primitiva geometrica che si vuole creare con l'applicazione.
Un possibile iter per tale procedura è il seguente:

- Prevedere un vettore di punti allocabile nel quale memorizzare i punti immessi;
- Prevedere una struttura dati per gli oggetti virtuali in grado di leggere il vettore dei punti del passo precedente;
- In corrispondenza delle istruzioni per il disegno delle primitive salvare la viewport e le matrici di proiezione e di vista in variabili globali, in modo da memorizzare la posizione relativa del punto di vista rispetto al riferimento globale.
- Definire una funzione di tipo mouse callback che a partire dalle coordinate in pixel del punto selezionato, passando al metodo `gluUnProject` le matrici

caricate al punto precedente, calcoli le coordinate del punto corrispondente nello spazio, aggiorni la primitiva attiva e ridisegni la finestra grafica con il metodo `glutPostRedisplay`.

A9. Interazione con la scena OpenGL

Una delle operazioni più frequenti eseguita in applicazioni di ingegneria virtuale è il puntamento (*picking*) agli oggetti disegnati o alle entità geometriche che li costituiscono. Questa funzionalità richiede l'implementazione di una procedura per la selezione di entità nell'ambiente OpenGL. Da un punto di vista operativo, l'interazione dell'utente con la scena avviene attraverso il puntamento del mouse su un pixel dell'immagine della scena.

Il problema si riconduce quindi ad una procedura di calcolo inverso, rispetto alla proiezione degli oggetti sulla finestra grafica, che il motore OpenGL dovrà eseguire per trovare la relazione tra il pixel e l'oggetto disegnato.

A tal riguardo è necessario osservare che OpenGL possiede due modalità di rendering:

- Il *render mode*, utilizzato per proiettare le primitive sulla finestra grafica (*frame buffer*).
- Il *selection mode*, utilizzato per la selezione. Non produce immagini, ma memorizza i nomi assegnati alle primitive da disegnare nel *selection buffer*.

Il selection mode serve per selezionare oggetti dell'immagine visualizzata (ad esempio gli oggetti sotto la posizione del cursore). La procedura di selezione prevede:

- Il disegno in selection mode della stessa scena disegnata in render mode;
- La restrizione dell'area da elaborare a quella nell'intorno del cursore;
- L'assegnazione di nomi alle primitive o agli oggetti;
- Gestione dell'evento di *hit* che OpenGL genera quando una primitiva ricade nel volume di selezione.

La procedura deve pertanto essere implementata rispettando i seguenti passi:

1. Creazione del selection buffer;
2. Inizio della modalità di selezione;
3. Restrizione dell'area di elaborazione
4. Assegnazione di nomi alle primitive da disegnare;
5. Fine della modalità di selezione;
6. Interrogazione del selection buffer per leggere i nomi degli oggetti selezionati;

Il comando associato alla creazione del Selection Buffer è:

```
glSelectBuffer(size,buffer);
```

dove `buffer` è un vettore di interi di tipo `GLuint`, che funziona da buffer e deve avere una dimensione sufficiente a contenere tutti gli eventi di selezione. Invece `size` rappresenta la dimesione di `buffer`.

Per entrare nella modalità di selezione si utilizza il comando `glRenderModel(GL_SELECT)` mentre per restringere il campo di selezione c'è la funzione `gluPickMatrix(x,y,w,h,viewport)`. Quest'ultima funzione restringe l'area di selezione (ovvero l'area della finestra in cui le primitive da disegnare generano l'evento di selezione). Se non si specifica il volume di selezione OpenGL assume, per default, un volume pari all'intera finestra grafica.

La funzione `gluPickMatrix` deve essere chiamata subito dopo le istruzioni:

```
glMatrixMode(GL_PROJECTION) ;
glLoadIdentity();
```

ovvero prima di applicare altre trasformazioni sulla projection matrix.

Gli argomenti della funzione gluPickMatrix hanno il seguente significato:

- x,y rappresentano il centro dell'area di selezione sulla finestra grafica e coincidono spesso con la posizione del cursore;
- w,h rappresentano la larghezza e la lunghezza dell'area ristretta;
- viewport è un array di quattro interi che descrive la finestra corrente[29].

Per disegnare la scena si attivano le stesse trasformazioni e attributi usati nella modalità `RENDERING_MODE`.

Nella fase di disegno si assegnano i nomi alle primitive o a gruppi di primitive per consentirne il riconoscimento in fase di selezione. La logica di assegnazione rispecchia quella di aggregazione. Di fatto più primitive hanno lo stesso nome se fanno parte di un oggetto pensato come unico. Una primitiva ha più nomi se è pensata come facente parte

[29] Per ricevere informazioni sulle dimensioni della finestra corrente esiste la funzione `glGetIntegerv`, è necessario precisare che le coordinate della finestra hanno y=0 in alto, mentre per convenzione OpenGL le considera con y=0 in basso. Quindi date le coordinate (`wx,wy`), che definiscono la posizione del mouse nella finestra grafica, in OpenGL si avrà `x=wx` e `y=viewport[3]-wy`.

di oggetti diversi a seconda del livello di astrazione considerato. Quindi un oggetto (figlio) inglobato in un oggetto più grande (padre) ha due nomi, quello che gli viene assegnato e quello dell'oggetto padre.
In OpenGL i nomi sono in realtà numeri interi che vengono assegnati alle primitive da disegnare e vengono memorizzati in modo sequenziale (*stack*).
Le istruzioni per l'assegnazione dei nomi sono le seguenti.

```
glInitNames();
glPushName(int nome);
glPopName();
glLoadName(int nome);
```

L'istruzione `glInitNames` inizializza lo stack dei nomi a vuoto mentre `glPushName` riceve per argomento l'intero della primitiva da aggiungere nello stack. `glPopName` toglie il nome dallo stack e `glLoadName` sostituisce l'ultimo nome inserito nella sequenza dei nomi con il nome dell'oggetto che riceve per argomento. In pratica l'istruzione associata a `glLoadName` è equivalente alla sequenza di istruzioni `glPopName` e `glPushName`. Un modo di procedere efficace per inizializzare i nomi è quello di assegnare un nome fittizio con il comando `glPushName` e poi utilizzare il comando glLoadName per caricare le altre entità.

Una volta attivato la modalità di selezione e memorizzato gli eventi di selezione, si esce dalla modalità di selezione con l'istruzione:

```
N=glRenderMode(GL_RENDER);
```

la quale fa ritornare il programma in rendering mode e scrive su N il numero di *hit* generati.

Il passo successivo è quello di leggere il selection buffer per risalire agli oggetti selezionati. Gli eventi di selezione vengono memorizzati in sequenza nel selection buffer e ciascuno di essi occupa un numero variabile di posti per un minimo di quattro.

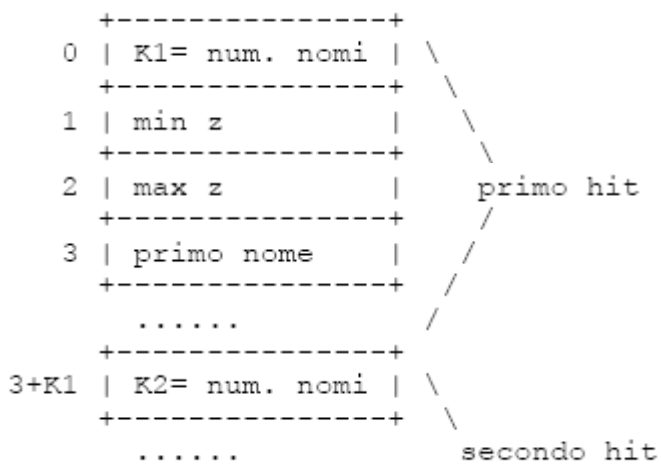


Figura A. 5: Struttura del *Selection Buffer* per memorizzare gli oggetti selezionati.

In Figura A. 5 è riportata schematicamente la struttura dati del vettore di selezione memorizzato nel selection buffer.

Per ogni hit il selection buffer riporta:

1. Il numero di nomi della primitiva selezionata (k_i);
2. Il valore di z minimo nello z-buffer della finestra, corrispondente ad un numero reale tra 0 e 1, che viene moltiplicato per $\left(2^{32}-1\right)$ e arrotondato all'intero più vicino;
3. Il valore di z massimo nello z-buffer;

Dal punto di vista operativo quello che interessa conoscere è il nome della primitiva selezionata. Se `N` è il numero di hit registrati e `buffer` è il vettore usato come buffer, tutte le primitive hanno un solo nome e il nome dell'i-esimo oggetto selezionato sarà registrato nell'elemento `buffer[i*4+3]`. Mentre nel caso che più primitive si trovino in corrispondenza del punto cliccato, tutte generano un evento di selezione e per capire qual è la primitiva che è più vicina al punto di osservazione occorre interrogare il selection buffer. Una possibile implementazione di questa ricerca è riportata nel codice seguente:

```
InizioHit = 0;
HitBuono = -1; /* hit da cercare */
ZetaBuona = 0xFFFFFFFF; /* la sua zeta */
for (i=0; i < N; i++)
{
zmin = buffer[InizioHit+1];
if (zmin < ZetaBuona)
{ ZetaBuona = zmin; HitBuono = i; }
InizioHit += (3 + buffer[InizioHit]);
/* inizio del prossimo hit record  */
}
```

Si noti come ad ogni ciclo sia necessario incrementare il contatore di inizio selezione a `3 + buffer[InizioHit],` poiché ad ogni selezione corrispondono 3 elementi (`k1, min z` e `max z`) più `k1` nomi.

A10. Struttura dati VRML per il disegno di oggetti solidi

La rappresentazione grafica di un solido in OpenGL richiede una struttura dati per memorizzarlo e funzioni di disegno per la resa grafica.

Un solido è assimilabile ad un insieme di facce poligonali, ciascuna delle quali può avere un numero arbitrario di vertici ed ogni vertice è una terna di coordinate cartesiane del tipo $\{x \quad y \quad z\}^t$. Poiché un vertice appare su più facce è opportuno memorizzare le sue coordinate in formato indicizzato. Ovvero tenendo tutti i vertici in una lista globale che compare una sola volta e i vertici di ciascuna faccia richiamati nella lista globale dei vertici.

Per convenzione se il solido ha N vertici, il primo vertice della lista ha indice 0 mentre l'ultimo N-1.

La struttura dati corrispondente al solido può essere organizzata in questo modo:

- Numero di vertici del solido (variabile di tipo intero);
- Numero di facce del solido (variabile di tipo intero),
- 3 vettori contenenti ciascuno una delle coordinate cartesiane dei punti (variabili di tipo float)
- Vettore contenente tutte le facce (struttura dati)

La struttura dati può essere inglobata in una classe C++ del tipo:

```
class Solid
{
public:
int vert_num, face_num;
float * x;
float * y;
float * z;
struct OneFace * faces;
...
};
```

mentre la struttura dati per la faccia deve contenere:

- v_count: la variabile che registra il numero di vertici della faccia
- v_array[]: il vettore che contiene gli indici dei vertici della faccia. Ciascuno è un indice nell'array dei vertici del solido

La struttura dati per le facce può essere del tipo:

```
struct OneFace
{
unsigned int v_count;
unsigned int * v_array;
};
```

Ragionando in questo modo i vertici dell'i-esima faccia del solido si trovano in `faces[i].v_array[k]` per `k=0..faces[i].v_count-1`. Mentre le coordinate di questi vertici (es: la x) si trovano in `x[faces[i].v_array[k]]` per `k=0..faces[i].v_count-1`.

La classe *Solid* così implementata contiene la struttura dati necessaria per memorizzare le informazioni relative ad un solido poliedrico. Tuttavia deve essere completata con funzioni per la lettura da file o per l'inizializzazione delle variabile e per l'allocazione dinamica di memoria dei vettori dei vertici e del vettore delle facce.

Un esempio completo di implementazione è riportato in Tabella 3.6.5, mentre in Tabella 3.6.6 sono sintetizzati i metodi della classe.

Tabella 3.6.5. Listato di esempio per la gestione di un solido con struttura dati VRML. In solid.c è contenuta la dichiarazione della classe Solid mentre in solid.h la sua implementazione.

//SOLID.C

```
#pragma once
#ifndef SOLID_H
#define SOLID_H
#include <stdio.h>
//Struttura dati OneFace
struct OneFace
{
  unsigned int v_count;
  unsigned int * v_array;
};
//Dichiarazione della classe Solid
class Solid
{
  public:
  //numero di face e vertici  unsigned int vert_num;
  unsigned int face_num;
 //vettori delle componenti dei vertici
 float * x;
  float * y;
  float * z;
  //vettore delle facce
  struct OneFace * faces;
  protected:
  //metodo per l'allocazione dei vertici
 void make_vertices(int n);
  //vettore per l'allocazione del vettore delle facce
  void make_faces(int n);
  //rilascio della memoria nell'eliminazione della classe
```

```
  void kill(void);
  public:
  /* metodi per la costruzione della classe */
  inline Solid()
  {  vert_num = face_num = 0;
     x = y = z = NULL;
     faces = NULL;
  }
  /* distruzione */
  inline ~Solid() {  kill();  }
  /* Metodi per la lettura dati */
  int readSolid(FILE * fd);
  int readSolid(char *filename);
  /* Metodi per il salvataggio dei dati */
  void writeSolid(FILE * fd);
  void writeSolid(char *filename);
  protected:
  /* Metodo per la lettura dell'i-esimo vertice */
  int read_vertex(FILE * fd, int i);

  /* Metodo per la lettura dell'i-esima faccia */
  int read_face(FILE * fd, int i);

};
#endif
```

//SOLID.H

```
#include "StdAfx.h"
#include ".\solid.h"
#include <stdlib.h>
#include "solid.h"
// Numero massimo di vertici per una faccia
#define MAX_FACE_VERT_NUM 50

void Solid :: make_vertices(int n)
{
  x = (float *) calloc(n,sizeof(float));
  y = (float *) calloc(n,sizeof(float));
  z = (float *) calloc(n,sizeof(float));
  if (x && y && z)  vert_num = n;
  else
  { if (x) free(x); x = NULL;
    if (y) free(y); y = NULL;
    if (z) free(z); z = NULL;
    vert_num = 0;
  }
}

void Solid :: make_faces(int n)
{
  faces = (struct OneFace *) calloc(n,sizeof(struct OneFace));
  if (faces)  face_num = n;
  else face_num = 0;
}

void Solid :: kill(void)
{  if (x) delete x;
   if (y) delete y;
   if (z) delete z;
   x = y = NULL;
   vert_num = 0;
   if (faces) delete faces;
```

```
   faces = NULL;

   face_num = 0;
}

int Solid :: read_vertex(FILE * fd, int i)
{  if ( (i<0) || (i>=vert_num) ) return 0;
   if (fscanf(fd, "%f %f %f", &x[i], &y[i], &z[i]) != -1)  return 1;
   return 0;
}

int Solid :: read_face(FILE * fd, int i)
{  unsigned int aux[MAX_FACE_VERT_NUM];
   int n = 0;
   int next_vert = 0;
   if ( (i<0) || (i>=face_num) ) return 0;
   while (next_vert != -1)
   {  if (fscanf(fd, "%d", &next_vert) == -1) return 0;
      if (next_vert>=0)  aux[n++] = next_vert;
   }
   faces[i].v_array = (unsigned int *) calloc(n,sizeof(unsigned int));
   if ( !faces[i].v_array ) return 0;
   faces[i].v_count = n;
   for (n=0; n<faces[i].v_count; n++)  faces[i].v_array[n] = aux[n];
   return 1;
}

int Solid :: readSolid(FILE * fd)
{
  int n;
  int i;
  if (fscanf(fd, "%d", &n)== -1) return 0;
  make_vertices(n);
  for (i=0; i<n; i++)  read_vertex(fd,i);
  if (fscanf(fd, "%d", &n)== -1) return 0;
  make_faces(n);
  for (i=0; i<n; i++)  read_face(fd,i);
  return 1;
}

int Solid :: readSolid(char *filename)
{
  FILE * fd;
  fd = fopen(filename,"r");
  if (!fd) return 0;
  return readSolid(fd);
}

void Solid :: writeSolid(FILE * fd)
{
  unsigned int i, j;
  fprintf(fd, "%d\n", vert_num);
  for (i=0; i<vert_num; i++)  fprintf(fd, "%f %f %f\n",
x[i],y[i],z[i]);
  fprintf(fd, "%d\n", face_num);
  for (i=0; i<face_num; i++)
  {  for (j=0; j<faces[i].v_count; j++)
          fprintf(fd, "%d ",faces[i].v_array[j]);
     fprintf(fd, "-1\n");
  }
}
```

```
void Solid :: writeSolid(char *filename)
{
  FILE * fd;
  fd = fopen(filename,"w");
  if (fd) writeSolid(fd);
}
```

Tabella 3.6.6: Descrizione dei metodi della classe *Solid*.

Metodo	Descrizione

`int readSolid(FILE * fd);`
`int readSolid(char *filename);`
`int read_vertex(FILE *fd,int i);`
`int read_face(FILE * fd,int i);`
	Metodi della classe per la lettura dei dati da file, relativi a vertici e facce del solido da disegnare

`void writeSolid(FILE * fd);` `void writeSolid(char filename);`	Metodo per la scrittura dei dati relativi ad un solido su file
`Solid()`	Metodo per l'inizializzazione della classe
`void Inizializzazione(void)`	Inizializza l'applicazione impostando la proiezione della scena, le luci e le proprietà dei materiali
`~Solid()` `void kill(void);`	Metodi per il rilascio di memoria
`void make_vertices(int n);` `void make_faces(int n);`	Metodi per l'allocazione dinamica di memoria dei vettori delle coordinate dei vertici e dell'indicizzazione delle facce

Come applicazione pratica si pensi alla struttura dati relativa ad un solido di forma tetraedrica. Rispettando la sintassi della struttura dati VRML, il file di testo corrispondete dovrebbe contenere le seguenti dichiarazioni:

```
4
-0.866025 -0.5 -0.25
0.866025 -0.5 -0.25
0 1 -0.25
0 0 0.75
4
0 1 3 -1
1 2 3 -1
0 3 2 -1
0 2 1 -1
```

Dove la prima riga definisce il numero dei vertici (4) ed è seguita da quattro righe che riportano le coordinate *x*, *y*, *z* per i vertici. Di seguito si definisce il numero di facce (4) a cui seguono quattro righe che ne definiscono l'indicizzazione rispetto ai vertici.

Per provare la classe *Solid* si potrebbe modificare la funzione `casa()` del Tabella 3.6.2 con il seguente frammento di codice:

```
void casa(void)
```

```
{
int i,j,k;
Solid * P;
P=new Solid;
P->readSolid("c:/tetraedro.txt");
for(k=0;k<P->face_num;k++)
{
 glBegin(GL_POLYGON);
  for(i=0;i<P->faces[k].v_count;i++)
  {
   glColor3f(0.2*i,0.0,0.2);
   glVertex3f( P->x[P->faces[k].v_array[i]],
      P->y[P->faces[k].v_array[i]],
      P->z[P->faces[k].v_array[i]]);
  }
 glEnd();
P->kill;
};
```

dove si esegue in ordine: l'inizializzazione di un'istanza della classe *Solid*, si assegnano i valori alle variabili relative ai vertici e alle facce, utilizzando il metodo di lettura della classe (`P->readSolid(…)`), e si utilizzano in maniera ricorsiva i comandi di disegno OpenGL sfruttando la struttura dati *Solid*.

Il risultato è riportato in Figura A.6 e si ottiene lanciando il programma e selezionando dal menu contestuale la voce "Casa".

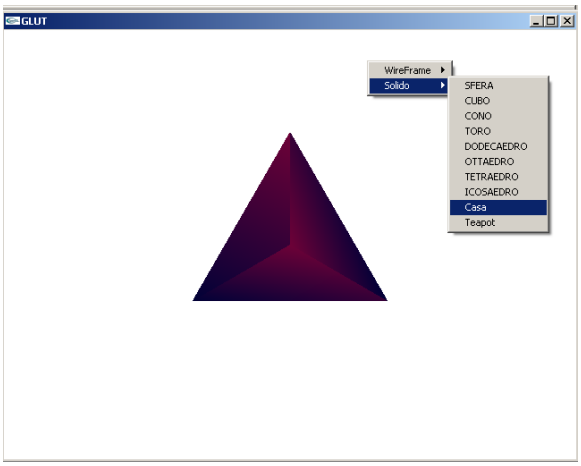


Figura A.6: Modellazione di un tetraedro con la struttura dati VRML.

Appendice B **Tecniche di elaborazione dati nei processi di ingegneria inversa**

In questa sezione si riportano due applicazioni di ingegneria inversa sviluppate dall'autore [63][66]. La prima riguarda l'elaborazione dati nel processo di ricostruzione di reperti archeologici (vasi), mentre la seconda riguarda la riproduzione virtuale del braccio umano, per configurazioni diverse dell'avambraccio rispetto al braccio.
Lo scopo è quello di discutere aspetti connessi all'elaborazione dati nei processi di ingegneria inversa.

B1. **Il processo di Reverse Engineering**

L'ingegneria inversa (dall'inglese *Reverse Engineering*[30]) [62] è una tecnica progettuale utilizzata per duplicare un oggetto esistente. Dove per duplicazione si intende la riproduzione virtuale dell'oggetto nelle sue forme, dimensioni e funzionalità.
Le fasi tipiche del processo di ingegneria inversa sono, in ordine:

- La fase di acquisizione dei punti;
- La fase di preprocessing;
- La fase di segmentazione;

[30] Secondo il manuale militare americano *MIL-HDBK-115 (ME)* la Reverse Engineering è "*...il processo di duplicazione di un oggetto nelle sue funzioni e nelle sue dimensioni attraverso un'analisi fisica e la misura delle sue parti, per ottenere i dati tecnici richiesti per la lavorazione.*"

- La fase di costruzione del modello 3D;

La scelta dei dispositivi di acquisizione e delle tecniche di elaborazione condiziona notevolmente la fedeltà e la laboriosità del processo di riproduzione.

La prima operazione da compiere è la scelta del sistema di acquisizione: per contatto[31]o senza contatto[32]. La scelta ricade sulla strumentazione che meglio si adatta ai vincoli di costo e alle esigenze di rilievo. Forme con sottosquadri importanti, ad esempio, precludono l'utilizzo di bracci robotizzati poco flessibili o strumenti di riflessione che non sono in grado di rilevarne le profondità e la forma. D'altra parte per la riproduzione di forme complesse, come ad esempio i profili delle facce delle monete metalliche, i metodi senza contatto sono senz'altro i più vantaggiosi, poiché assicurano un'elevata velocità di acquisizione. In altri casi, la necessità di elaborare un modello parametrico della geometria da riprodurre, spinge verso l'impiego dei metodi per contatto con la selezione dei punti più importanti.

Le due fasi intermedie del processo di Reverse Engineering, preprocessing e segmentazione, si occupano dell'elaborazione dei dati di acquisizione via software. In funzione del modello e delle applicazioni finali, tali fasi possono subire ottimizzazioni e miglioramenti volti a migliorarne la velocità e/o l'efficienza. Di seguito vengono presentati due casi di ottimizzazione, il primo riguarda l'implementazione di una procedura di calcolo automatico attraverso le interfacce di programmazione [63]. La seconda riguarda la riproduzione dei moti del gomito umano per diverse configurazioni di braccio e ulna rispetto al braccio [66].

B2. Procedure automantiche per l'elaborazione dati

Le applicazioni destinate alla ricostruzione di reperti archeologici richiedono un modello virtuale parametrico facilmente manipolabile con i comandi CAD. Infatti, l'applicazione che si occupa della ricostruzione deve trovare la corrispondenza tra i bordi di due frammenti diversi, partendo dalla conoscenza dell'asse geometrico di rivoluzione dei frammenti stessi. Le fasi del processo di ricostruzione sono schematizzate in Figura B1.

[31] Gli strumenti di misura includono gli strumenti manuali, i bracci robotizzati, le macchine di misura a coordinate o *CMM* (*Coordinate Measuring Machine*).

[32] Per sistemi di acquisizione senza contatto si fa riferimento agli strumenti di misura che si basano sul principio di riflessione delle onde magnetiche, acustiche o ottiche.

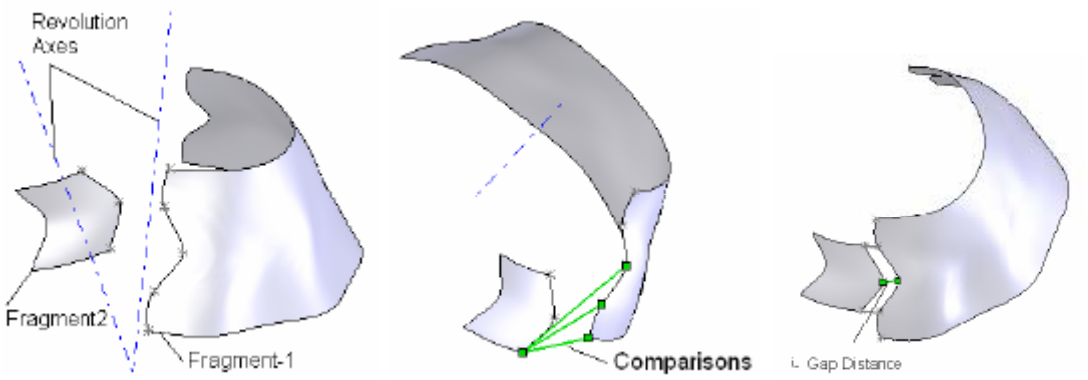


Figura B1 La procedura di ricostruzione dei reperti archeologici: unione degli assi (a sinistra), ricerca delle corrispondenze (al centro), posizionamento.

Pertanto, in questo tipo di applicazioni, la fase di ingegneria inversa, al fine di agevolare la successiva fase di ricostruzione, deve restituire non solo il modello parametrico ma anche l'asse geometrico di rivoluzione del frammento rilevato.

In tale ambito l'implementazione di una procedura di calcolo automatico, sfruttando le interfacce di programmazione dei moderni software CAD [65] (v. Figura B2), consente di snellire notevolmente il processo di ricostruzione virtuale.

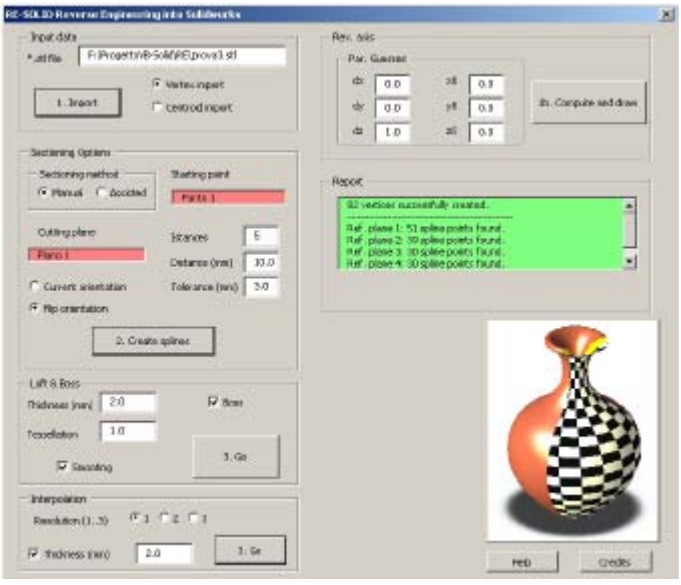


Figura B2. Interfaccia dell'applicazione per la ricostruzione di frammenti archeologici.

L'algoritmo da implementare può essere scomposto nei seguenti passi:

1. Acquisizione dei punti mediante scanner laser;
2. Ricerca di un ipotetico asse di rivoluzione con il metodo di *Halir*;
3. Estrazione dei punti appartenenti ai piani ortogonali agli assi;

4. Creazione dei profili appartenenti ai piani;
5. Modellazione delle superfici mediante *loft*;

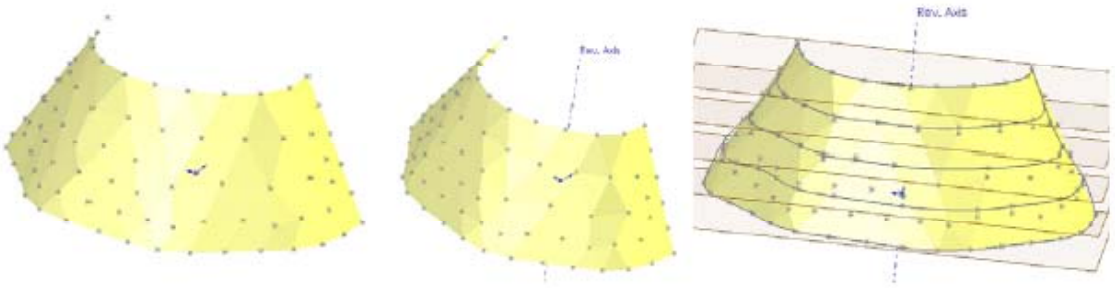


Figura B3. Ricostruzione del modello parametrico di un frammento archeologico;

In Figura B2 si riporta la finestra virtuale dell'applicazione implementata per la ricostruzione dei frammenti di vaso, mentre in Figura B3 sono rappresentate la fasi di modellazione del frammento e dell'asse geometrico di rivoluzione; si noti come per la selezione dei punti sia stato necessario prevedere un criterio di differenziazione tra i punti appartenenti alle sezioni di loft, necessari alla modellazione, e i punti distanti dai piani delle sezioni, inutili ai fini della modellazione.

B3. La scelta dei punti nella fase di elaborazione dati

Uno dei problemi, recentemente affrontati nel settore biomedico, riguarda la riproduzione degli apparati articolari del corpo umano per simulazioni capaci di assistere il chirurgo nell'installazione di impianti protesici.

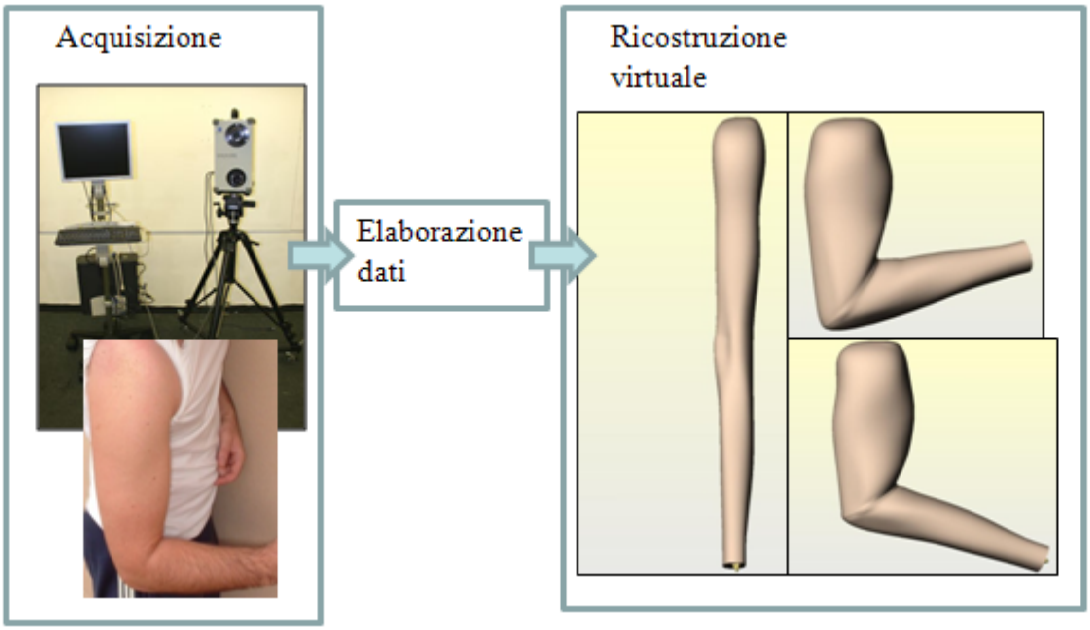


Figura B4. Acquisizione e riproduzione virtuale del braccio.

Un esempio rappresentativo delle problematiche e delle strategie utilizzate per affrontare questa classe di problemi è la riproduzione delle funzionalità e della cinematica del gomito umano, per impianti protesici sostitutivi dell'articolazione del gomito stesso. Le problematiche affrontate nella simulazione riguardano la corretta riproduzione dei vincoli cinematici presenti tra le ossa delle articolazioni e la ricostruzione della superficie esterna del braccio per posizioni relative diverse dell'avambraccio rispetto al braccio.

Secondo l'approccio tradizionale la ricostruzione virtuale per la generica posizione si otterrebbe grazie ad una fase di acquisizione ed elaborazione dati (v. Figura B4). Tuttavia la necessità di ricostruire il braccio per diverse configurazione dei moti di prono-supinazione e flesso-estensione [67] ha spinto verso l'implementazione di una procedura di ricostruzione basata sull'interpolazione dei parametri cinematici e geometrici per un numero finito di configurazioni.

Anche in questo ambito le interfacce di programmazione sono un valido strumento per automatizzare le operazioni di ricostruzione e simulazione (Figura B5).

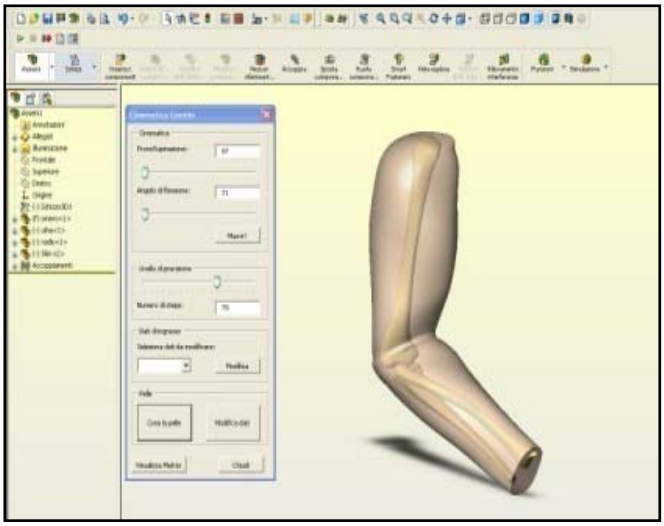


Figura B5. Interfaccia dell'applicazione per la ricostruzione del braccio.

La posizione relativa di omero, ulna e radio può essere stimata interpolando i valori noti sulle posizioni e orientazioni dell'asse di flessione (*IAFE*) e del moto di roto-traslazione tra ulna e radio (*UAD* e *swaying angle*). Lo strato superficiale della pelle, che circonda i tessuti del braccio invece, può essere modellato mediante interpolazione (*loft*) di alcune sezioni del braccio (v. Figura B 6).

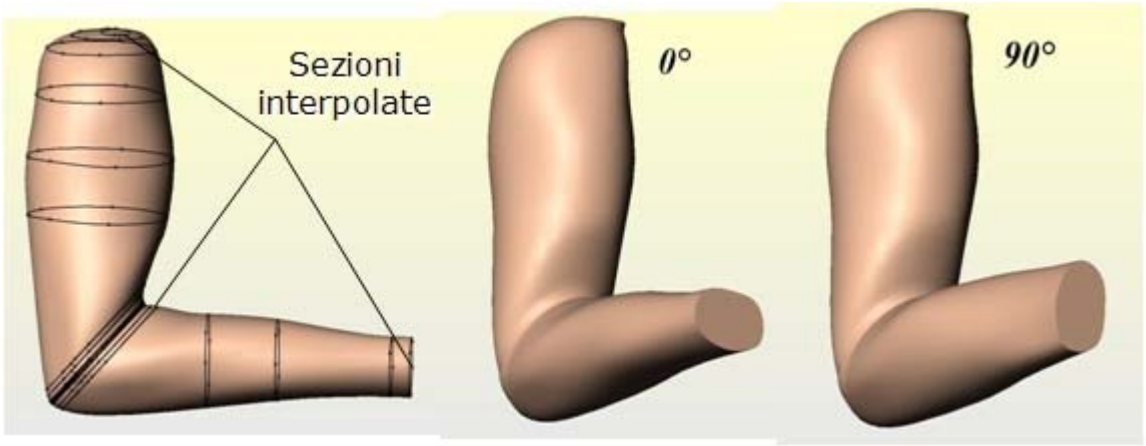


Figura B 6. La modellazione per interpolazione: sezioni interpolate a sinistra, effetto della prono-supinazione per 0° e 90° a destra.

I risultati sperimentali dimostrano che è sufficiente utilizzare 16 sezioni, opportunamente disposte, ciascuna ottenuta per interpolazione di 7 punti, per riprodurre con sufficiente precisione la superficie esterne del braccio. Modificando la disposizione delle sezione si riproduce l'effetto della flessione sulla forma del braccio, mentre modificando la disposizione dei punti di sezione l'effetto della prono-supinazione.

Infine sono sufficienti 4 configurazioni per la flessione e, per ciascuna di queste, 3 per la prono-supinazione.

In definitiva, mediante tecniche di interpolazione e criteri di selezione finalizzati alla modellazione, è possibile riprodurre il modello virtuale del braccio senza dover acquisire ed elaborare i punti per ciascuna configurazione.

B4. Conclusioni

Le applicazioni di ingegneria inversa riportate in questa sezione evidenziano come l'elaborazione dati sia una fase fortemente dipendente dall'applicazione finale. In entrambi i casi i dati acquisiti devono essere filtrati per considerare solo i punti che effettivamente interessano al modello virtuale impiegato per la simulazione. In realtà, per la ricostruzione dei vasi si selezionano solo i punti che appartengono ai piani ortogonali all'asse geometrico di rivoluzione, mentre per la ricostruzione del braccio si arriva ad una doppia selezione, distinguendo tra i punti che appartengono alle sezioni d'interesse per l'operazione di loft e selezionando, tra questi ultimi, solo 7 punti per creare i profili delle sezioni di interpolazione.

Pertanto l'esigenza comune è quella di trovare procedure di elaborazione dati in grado di ricavare, in maniera veloce ed efficace, i punti di interesse, ovvero i punti strettamente necessari alla ricostruzione degli oggetti misurati.

Elenco delle Figure

Bibliografia

[1] Bimber Oliver. and Raskar Ramesh. *Spatial Augmented Reality Merging Real and Virtual Worlds.* A K Peters, Ltd, 2005.

[2]Bimber, O., Combining Optical Holograms with Interactive Computer Graphics, IEEE Computer (cover feature), pp. 85-91, 2004

[3]Sutherland, I., The Ultimate Display, In Proceedings of International Federation of Information Processing, Spartan Books, pp. 506-508, 1965.

[4]Sutherland, I., A Head-Mounted Three Dimensional Display, In Proceedings of Fall Joint Computer Conference, pp 757-764, Washington, DC, 1968.

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

[6]Pierre Malbezin, Wayne Piekarski and Bruce H. Thomas. Measuring ARToolKit Accuracy in Long Distance Tracking Experiments.

[7]Brunner, S., Bimber, O., Mader, S.. *Report on Tracking Technology*, 2003

[8]Vallino James. *Interactive augmented reality*. PhD thesis, Department of Computer Science, University of Rochester, USA (1998).

[9]Milgram, P.; Kishino, F, 1994, A Taxonomy of Mixed Reality Visual Displays, IECE Trans. On Information and Systems (Special Issue on Networked Reality), Vol. E77-D, N. 12, pp. 1321-1329.

[10]P. Milgram, H. Takemura et al. *Augmented Reality: A Class of Displays on the Reality-Virutality Continuum.* In Proceedings of SPIE: Telemanipulatorand Telepresence Technologies 2351 (1994), 282–292.

[11]*The Augmented Reality in Surgery* http://www.ariser.info/ .

[12] E. Trucco, A. Verri. Introductory Techniques for 3-D Computer Vision. Prentice Hall, 1998.

[13] Stilman M., Michel P., Chestnutt J., Nishiwaki K., Kagami S., Kuffner J.J. (2005). *Augmented Reality for Robot Development and Experimentation*. Tech. Report CMU-RI-TR-05-55, Robotics Institute, Carnegie Mellon University.

[14] Webster A., Feiner S., MacIntyre B., Massie W., Krueger T.. *Augmented reality in architectural construction, inspection, and renovation*. Int. Proc. Of Third Congress on Computing in Civil Engineering ASCE 3, Anaheim, CA, pp. 913-919, 1996.

[15] Bimber, O., Raskar, R.. *Modern Approaches to Augmented Reality*. 25th Annual Conference of the European Association for Computer Graphics, Interacting with Virtual Worlds, Tutorial 8, pp. 1-86, 2004

[16] Samset E., Talsma A., Elle O., Aurdal L., Hirschberg H., Fosse E. (2002). *A virtual environment for surgical image guidance in intraoperative MRI*, Computer Aided Surgery 7(4), pp.187– 196

[17] Bell, B., Feiner, S., Höllerer, T.. *Information at a Glance.* IEEE Computer Graphics and Applications, vol. 22, n. 4, Jul/Aug, pp. 6-9, 2002.

[18] Bernard A. *Proceedings of the Institution of Mechanical Engineer.*, Part B: Journal of Engineering Manufacture Vol. 219 (5), pp. 413-422, 2005.

[19] Silke Geisen. *Augmented Reality in Surgery*. Universitat Paderborn 2005.

[20] Azuma R.. *A Survey of Augmented Reality*. Presence: Teleoperators and Virtual Environments, vol. 6, n. 4, August, pp. 355 – 385, 1997.

[21] Ong S.K., Pang Y., Nee, A.Y.C.. *Augmented Reality Aided Assembly Design and Planning*. Annals of the CIRP Vol. 56/1 pp.49-52, 2007.

[22] Pang Y., Nee A.Y.C., Ong S.K., Yuan M.L. Youcef-Toumi K. (2006). *Assembly Feature Design in an Augmented Reality Environment*, Assembly Automation, 26/1, pp. 34-43.

[23] Sharma R. and Molineros J. (1995). *Computer vision-based augmented reality for guiding manual assembly*. PRESENCE: Teleoperators and Virtual Environments, n. 3, pp. 292-317.

[24] Liarokapis, Petridis P., Lister P.F., White M. (2002). *Multimedia augmented reality interface for E-learning (MARIE)*. World TransEng Technol Educ 1(2), pp.173–176.

[25] Pan Z., Cheok A.D., Yang H., Zhu J.,_, Shi J. (2006). *Virtual reality and mixed reality for virtual learning environments*. Computers & Graphics 30, pp. 20–28.

[26] Kaufmann H., Schmalstieg D., Wagner, M. (2000). *Construct3D: A Virtual Reality Application for Mathematics and Geometry Education*. Education and Information Technologies 5(4), pp. 263-276

[27] Dangelmaier W., Fischer M., Gausemeier J., Grafe M., Matysczok C., Mueck B. (2005). Virtual and augmented reality support for discrete manufacturing system simulation. Computers in Industry 56, pp. 371–383.

[28] Ong S.K., Nee A.Y.C. (2004). Virtual and Augmented Reality Applications in Manufacturing. Springer, London, UK.

[29] Reif R., Walch D. (2008). Augmented & Virtual Reality applications in the field of logistics. *Visual Computing* 24, pp. 987–994.

[30] Friedrich W. (2004). *ARVIKA—Augmented Reality for Development, Production and Service*. Publicis Corporate Publishing, Erlangen.

[31] Liarokapis F., Sylaiou S., et al. (2004). *An interactive visualization interface for virtual museum*. Proceedings of the 5th international symposium on Virtual Reality, Archaeology- Cultural Heritage, Brussels and Oudenaarde, pp 47–56.

[32] Narzt W., Pomberger G., Ferscha A., Kolb D., Muller R, Wieghardt J., Hortner H., Lindinger C. (2006). Augmented reality navigation systems. *Univ Access Inf Soc* 4, pp. 177–187.

[33] Pinz, A., Brandner, M., Ganster, H., Kusej, A., Lang, P., Ribo, M., *Hybrid Tracking in Augmented Reality*, ÖGAI Journal, vol. 21, n. 1, pp. 17-24, 2000.

[34] Ettore Pennestrì. *Dinamica Tecnica e Computazionale*. Casa editrice Ambrosiana, Vol. 2 Aprile 2002.

[35] M. Fischler and R. Bolles, Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications ACM, vol. 24, no. 6, pp. 381–395, 1981.

[36] I. Sutherland. *Sketchpad: A Man Machine Graphical Communications System.* Technical Report 296, MIT Lincoln Laboratories, 1963.

[37] R.Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[38] Klinker G., Stricker D., Reiners D. (1999). Optically based direct manipulation for augmented reality, *Computers & Graphics* 23, pp. 827-830.

[39] Koller, G. Klinker, E. Rose, D. Breen, R. Whitaker, and M. Tuceryan. *Realtime vision-based camera tracking for augmented reality applications*. ACM Symposium

on Virtual Reality Software and Technology (Lausanne, Switzerland), pp. 87–94, September 1997.

[40] H. Kato and M. Billinghurst. *Marker tracking and HMD Calibration for a video-based augmented reality conferencing system*. IEEE and ACM International Workshop on Augmented Reality, October 1999.

[41] H. Kato, M. Billinghurst, I. Poupyrev, K. Imamoto, and K. Tachibana. *Virtual object manipulation on a table-top AR environment*. International Symposium on Augmented Reality, pp. 111–119, 2000.

[42] J. Rekimoto. Matrix: A realtime object identification and registration method for augmented reality. Asia Pacific Computer Human Interaction, 1998.

[43] Bowman D.A. (1996). Conceptual Design Space: Beyond Walk-through to Immersive Design. In Bertol D., *Designing Digital Space*, John Wiley & Sons, New York.

[44] Vincent Lepetit and Pascal Fua. *Monocular Model-Based 3D Tracking of Rigid Objects: A Survey*. Foundation and trends in computer graphics and vision Vol. 1, no. 1 pp. 1- 89, 2005.

[45] Bowman D.A. (1999). Interaction Techniques for Common Tasks in Immersive Virtual Environments: Design, Evaluation, and Application. Ph.D. thesis, Virginia Polytechnic & State University.

[46] Liang J. and Green M. (1994). JDCAD: a highly interactive 3D modeling system. *Computers & Graphics* 18(4), pp. 499-506.

[47] Raskar R., Welch G., Cutts M., Lake A., Stesin L., Fuchs H. (1998). *The office of the future: a unified approach to image-Based modeling and spatially immersive displays*. Proceedings of SIGGRAPH 98, Orlando, FL, July 19–24

[48] Klinker G., Ahlers K.H. et al.. Confluence of computer vision and interactive graphics for augmented reality, *PRESENCE: teleoperations and virtual environments*. Special issue on Augmented Reality, 6(4), pp. 433–451, 1997 ------- [7] in ch NOVA.

[49] Fotis Liarokapis (2007). An augmented reality interface for visualizing and interacting with virtual content, *Virtual Reality* 11, pp. 23–43

[50] Muller A., Conrad S., Kruijff E. (2003). *Multifaceted interaction with a virtual engineering environment using a scenegraph-oriented approach*. Proceedings of the 11th Int. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision, Czech Republic.

[51] Ong S.K., Nee A.Y.C. (2004). Virtual and Augmented Reality Applications in Manufacturing. Springer, London, UK.

[52] Reif R., Walch D. (2008). Augmented & Virtual Reality applications in the field of logistics. *Visual Computing* 24, pp. 987–994.

[53] Friedrich W. (2004). *ARVIKA—Augmented Reality for Development, Production and Service*. Publicis Corporate Publishing, Erlangen.

[54] Fiorentino M. *Design estetico in Mixed Reality: studio e implementazione*. Tesi di Dottorato – Politecnico di Bari, (2003)

[55] Canfield, T., Diachin, D., Disz, T., Freitag, L., Heath, D., Ku, J., Papka, M.,Szymanski, M., Huang, M., Chen, J., Taylor, V. "Visualizing Engineering Applications in the CAVE." Proceedings Symposium on Virtual Reality in Manufacturing Research and Education. October 1996.

[56] W. Buxton, G. Fitzmaurice, R. Balkrishnan, and G. Kurtenbach. *Large displays in automotive design*. IEEE Computer Graphics and Applications, pages 68-75,July/August 2000.

[57] Wayne Piekarski. *Interactive 3d modelling in outdoor augmented reality worlds*. Bachelor of Engineering in Computer Systems Engineering (Hons), University of South Australia, Adelaide 2004.

[58] Richard S. Wright Jr, Banjamin Lipchak. *OpenGL SuperBible Third Edition.* Sams Publishing june 30, 2004

[59] SolidWorks Advanced Part Modeling. SolidWorks Corporation 2006.

[60] Ascension Technology Corporation. *Motion trackers for Computer Graphics Application – Flock of Birds*. Installation and Operation Guide. www.ascension-tech.com Ascension Technology Corporation 2002.

[61] Ascension Technology Corporation. *Api documentation Bird.dll*. www.ascension-tech.com.

[62] Wills L.M., Newcomb P., *ReverseEngineering*, Springer, 1996.

Pubblicazioni

In questa sezione è riportato l'elenco delle pubblicazioni, di cui l'autore della tesi è coautore, inerenti ai contenuti tecnici e teorici affrontati nel presente lavoro di ricerca.

[63] P. P. Valentini, Davide Gattamelata, Eugenio Pezzuti. *Using Application Programming Interface to integrate Reverse Engineering Methodologies into SolidWorks*. Congresso Ingegraf, giugno 2006, Barcellona (Spagna).

[64] D. Gattamelata, E. Pezzuti, P.P. Valentini. *Sensibilità agli errori di OFF-DESIGN nella modellazione estetica di superfici.* Giornate di Studio ADM su "Metodi di Progettazione Concettuale per L'innovazione", settembre 2006,Forlì

[65] D. Gattamelata, E. Pezzuti, P. P. Valentini. *Personalizzazione dei programmi CAD attraverso le interfacce di programmazione*. Rivista Il Progettista industriale, tecniche nuove, febbraio 2009.

[66] D. Gattamelata, P.P. Valentini, E. Pezzuti. *Computer-aided simulation of human upper limb movements*. Congresso ADM-INGEGRAF di Perugia, maggio 2007.

[67] P.P. Valentini, D. Gattamelata, E. Pezzuti. *Accurate geometrical constraints for the computer aided modelling of the human upper limb*. Rivista Computer Aided Design, 2007 volume 39, n°7 pagg. 540-547.

[68] D.Gattamelata, P.P. Valentini, E. Pezzuti. *A CAD system in Augmented Reality application.* 20th European Modeling and Simulation Symposium, track on Virtual Reality and Visualization, Briatico (CS), September 17-19, 2008.

[69] P.P. Valentini, D. Gattamelata, E. Pezzuti. *Using Augmented Reality for Interactive Engineering Simulations of motion*. Congresso congiunto ADM-Ingegraf, Lugo (Spagna) 2009.

[70] P.P. Valentini, D. Gattamelata, E. Pezzuti. *Virtual Engineering in Augmented Reality*. Capitolo all'interno di Computer Animation, NOVAPublishers (in stampa).

[71] Pier Paolo Valentini, Eugenio Pezzuti, Davide Gattamelata. *Interactive Multibody Simulation in Augmented Reality*. Multibody Dynamics 2009, Eccomas Warsaw, Polland. 29 June – 2 July 2009.

[72] Biancolini C., Biancolini M.E., Costa E., Gattamelata D., Valentini P.P., *Industrial Application of the Meshless Morpher rbf-Morph to a Motorbike Windshield Optimisation.* European Automotive Simulation Conference (EASC 2009) in Munich, 6-7 July 2009.