

ALMA MATER STUDIORUM – UNIVERSITA' DI BOLOGNA
SEDE DI CESENA
FACOLTA' DI SCIENZE MATEMATICHE, FISICHE E NATURALI
CORSO DI LAUREA IN SCIENZE DELL'INFORMAZIONE

REALIZZAZIONE DI UNA INTERFACCIA GRAFICA MULTI-PIATTAFORMA IN 3D PER L'IMAGING MEDICO

Tesi di Laurea in
Fisica Numerica

Relatore
Prof. Renato Campanini

Presentata da
Federico Brozzetti

Co-relatore
Dott. Matteo Roffilli

Sessione III
Anno Accademico 2003-2004

Introduzione	V
 Capitolo 1 - Computer Graphics	1
1.1 Introduzione alla Computer Graphics	2
1.2 Lo sviluppo della Computer Graphics	3
1.3 I sistemi grafici interattivi	7
1.4 Sistema grafico a livello hardware	8
1.5 Sistema grafico a livello software	9
 Capitolo 2 – Application Program Interface multi-piattaforma	15
2.1 Definizione	16
2.2 API Open Source	18
2.3 L’Open Source e il multi-piattaforma	20
2.3.1 I vantaggi del multi-piattaforma	22
2.4 Simple DirectMedia Layer	26
2.4.1 Inizializzazione	27
2.4.2 Un contesto grafico	28
2.4.3 I modi video disponibili	30
 Capitolo 3 - Open Graphic Library (OpenGL)	33
3.1 Le librerie grafiche	34
3.2 La libreria Open Graphics Library (OpenGL)	37
3.3 Primitive e attributi	42
3.3.1 Vertici e segmenti	43
3.3.2 Poligoni	46
3.3.3 Caratteri di testo	51
3.3.4 Oggetti curvilinei	52
3.3.5 Attributi	52
3.3.6 Colore	54
3.3.7 Visualizzazione	56
3.3.8 Matrix mode	59

3.4 Funzioni di controllo.....	59
3.4.1 Interazioni con il sistema window	60
3.4.2 Aspect-ratio e ViewPort	61
3.4.3 L'event-loop	63
Capitolo 4 - Open Scene Graph.....	65
4.1 Definizione	66
4.2 Strutture adottate.....	68
4.2.1 Octree.....	69
4.2.2 BSP-tree.....	70
4.2.3 Gerarchia di un Bounding Volume.....	71
4.3 Tecniche di grafica	72
4.3.1 Tecniche di culling	73
4.3.1.1 Viewfrustum Culling.....	73
4.3.1.2 BackFace Culling	75
4.3.1.3 Occlusion Culling.....	76
4.3.1.4 Potentially Visible Sets.....	77
4.3.2 Tecniche di semplificazione	78
4.3.2.1 Level of detail.....	78
4.3.2.2 Image Based Rendering.....	80
4.3.2.3 Image Caching.....	82
4.4 Caratteristiche tecniche della Scene Graph	83
4.5 La libreria Open Scene Graph	85
4.5.1 Open Producer	86
4.5.2 Open Threads.....	88
4.5.3 Open Scene Graph	89
4.5.4 Vantaggi dell'OpenSceneGraph	90
Capitolo 5 - L'informatica nel campo medico.....	95
5.1 Tecnologie informatiche in medicina	96
5.2 Ruolo dell'informatica nella medicina	97

5.3 Sistemi informativi in medicina.....	99
5.4 Dati, segnali e immagini biomediche.....	100
5.5 Classificazione dimensionale.....	103
5.6 Eidologia informatica.....	104
Capitolo 6 - Realizzazione dell'applicazione	107
6.1 Setup di Microsoft Visual Studio .Net 2003	108
6.1.1 Librerie utilizzate	111
6.2 Creazione di un contesto grafico con SDL	113
6.3 Impostazioni per creare un contesto OpenGL.....	114
6.4 Inserimento di una console	116
6.5 Realizzazione dell'interfaccia e caricamento di un'immagine	118
6.6 Unione degli elementi creati	120
6.7 Inserimento dell'Open Scene Graph	122
6.8 GUI integrata con 2D e 3D	124
6.9 Applicazione di dati medici	125
6.10 Conclusioni e sviluppi futuri.....	130
Conclusioni	VII
Bibliografia	IX
Ringraziamenti	XIII

Soltanto descrivere un universo così variegato come quello dell'Informatica e della Medicina è impresa veramente ardua. Non c'è tecnologia informatica o approccio informatico alla risoluzione di problemi, anche apparentemente astratti, che non abbia avuto una ricaduta notevole nel campo medico.

L'elenco è senza fine, dall'image processing alla grafica virtuale, dall'elaborazione in tempo reale di segnali al riconoscimento automatico di patologie, dalla ricostruzione di immagini in 3 dimensioni (3D) ai problemi di archiviazione di dati ed immagini, dai problemi di sicurezza e riservatezza agli algoritmi di analisi, visualizzazione e compressione di dati fino ai metodi di classificazione.

L'obiettivo di questa tesi è la realizzazione di un'interfaccia grafica utente (Graphics User Interface - GUI) implementata con librerie multi-piattaforma e open source per la visualizzazione 3D di dati medici.

Il lavoro svolto diventerà perciò parte di un sistema CAD (Computer Aided Detection) progettato e sviluppato dal gruppo di ricerca Medical Imaging Group (MIG) dell'Università degli Studi di Bologna, costituito da ricercatori del Corso di Laurea di Scienze dell'Informazione (sede di Cesena) e del Dipartimento di Fisica.

Un argomento simile, infatti, è già stato affrontato da un mio collega come tesi; il suo obiettivo è stato quello di creare un'interfaccia grafica utente in 2D per analizzare immagini radiografiche.

L'obiettivo di questa tesi è di estendere l'analisi all'elaborazione in 3D, e quindi sviluppare una GUI che permette di integrare le funzioni su dati bidimensionali a quelle su dati tridimensionali.

La tesi sarà così organizzata inizialmente trattando il concetto di computer graphics, ed in particolare le applicazioni che può avere questo settore dell'informatica. Successivamente l'attenzione verterà sulle librerie utilizzate:

- *una libreria 2D di basso livello (Simple Directmedia Layer) che permette di sfruttare le caratteristiche hardware e software dell'elaboratore;*
- *una libreria 3D di basso livello (OpenGL) che consente di sfruttare le accelerazioni hardware delle schede grafiche;*
- *una libreria 3D di alto livello (Open Scene Graph) che permette di gestire e creare oggetti tridimensionali.*

Infine verranno mostrate le fasi di sviluppo dell'applicazione, dalla gestione di un visualizzatore 3D alla generazione di una GUI che si integri con essa; dopodichè verranno presentati alcuni esempi del programma applicato a dati medici, e le possibilità di elaborazione grafica consentite.

Capitolo 1

Computer Graphics

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

1.1 Introduzione alla Computer Graphics

In seguito all'incredibile sviluppo della tecnologia hardware e software, siamo in grado di generare, raccogliere, e in seguito elaborare, informazioni in un campo di vastità prima impensabili. Con l'aumento del volume delle informazioni, sorge il problema di trasferirle dalla macchina all'uomo in modo al tempo stesso efficiente ed efficace. La computer graphics è orientata direttamente alla soluzione di questo problema: la grafica consente di comunicare tramite figure, schemi e diagrammi e offre un'alternativa stimolante alle stringhe di simboli della tastiera; un grafico può sostituire un'enorme tabella di numeri e permette, a chi lo legge, di notarne gli elementi fondamentali e le caratteristiche in modo immediato. Dare al calcolatore la possibilità di esprimere i suoi dati in forma grafica significa dunque aumentare enormemente la capacità di fornire informazioni all'utente.

La computer graphics costituisce uno dei campi dell'informatica moderna più ricchi di applicazioni, tra i quali:

- 1) **Interfacce utente:** Interazione con l'elaboratore dominata da un paradigma che includa finestre, icone, bottoni, cursori, oggetti vari, etc.. Al giorno d'oggi è normale parlare di tali parametri ma fino al 1983/84 (primi sistemi Apple Lisa¹ e Mac²) tutte le *interfacce utente* erano esclusivamente *command line*. Invece ora tutti gli elementi che compongono un'interfaccia grafica devono essere "disegnati" quindi necessitano di software grafico (2D).
- 2) **Progettazione a produzione:** Progettazione assistita da elaboratore (CAD³) di parti meccaniche, architettura, oggetti di design, etc.. praticamente tutti gli oggetti di fabbricazione industriale; inoltre interessa la manifattura assistita

¹ Uno dei primi personal computer sviluppato dalla Apple Computer Inc. [A02] negli anni Ottanta.

² Macintosh, abbreviato come Mac, è una popolare famiglia di personal computer costruiti a partire dal 1984 dalla Apple Computer Inc.

³ CAD: Computer Aided Design.

da elaboratore (CAM⁴) come macchine a controllo numerico (NC) o catene di montaggio robotizzate;

- 3) **Visualizzazione di informazioni:** Importanza della comunicazione visuale rispetto al linguaggio parlato o scritto. Esempi di tale settore sono la cartografia (mappe, plastici virtuali), dati statistici (grafici di vario tipo), dati medici (tomografie (TAC, RM, PET, SPECT), ecografie, Doppler, etc..), visualizzazione scientifica (resa grafica di dati generati per simulazione di fenomeni fisici come analisi ad elementi finiti, fluidodinamica, biologia molecolare, etc..)
- 4) **Simulazione:** Nasce dalla possibilità dei moderni sistemi CG di generare immagini realistiche in tempo reale. Esempi rilevanti sono (spesso in abbinamento con dispositivi I/O specializzati, Realtà virtuale):
 - Volo;
 - Guida e teleguida di veicoli speciali;
 - Operazioni mediche e chirurgiche;
 - Catastrofi naturali (terremoti, inondazioni, esplosioni, crolli, ...);
 - Operazioni di destrezza in ambiente ostile.

Tale tesi comprende questi quattro campi di applicazione dato che:

- 1) Si avvale di un'interfaccia grafica utente per rendere l'utilizzo il più semplice possibile;
- 2) Come già accennato nell'introduzione, è parte di un progetto CAD⁵;
- 3) Visualizza in una grafica sia bidimensionale che tridimensionale dati medici;
- 4) E' di fatto una simulazione di una diagnosi medica.

⁴ CAM: Computer Aided Manufacture.

⁵ CAD: Computer Aided Detection.

1.2 Lo sviluppo della Computer graphics

E' necessario dare spazio alla trattazione dello sviluppo della computer graphics. Ciò consentirà di introdurre alcuni degli aspetti e delle problematiche principali di questo settore di ricerca, che saranno poi ripresi e affrontati più in dettaglio nel seguito.

La computer graphics è un settore piuttosto giovane dell'informatica, soprattutto se ne consideriamo la diffusione. Infatti, fino ai primi anni 80, essa costituiva un piccolo campo, estremamente specializzato. Le ragioni di questo fatto sono da ricercarsi sia nel costo elevato delle tecnologie hardware, che nella scarsa disponibilità di programmi applicativi. Solo grazie al notevole progresso tecnologico degli ultimi anni, che ha consentito l'immissione sul mercato di terminali grafici e di programmi applicativi di costo accessibile, la computer graphics si è progressivamente diffusa, al punto da diventare il mezzo più comune di interazione tra uomo e macchina.

Le origini della computer graphics si potrebbero far coincidere con i primi tentativi di utilizzare i tubi a raggi catodici (in breve CRT⁶), utilizzati nei comuni apparecchi televisivi, come dispositivi di output grafico. Informalmente, un tubo a raggi catodici è un dispositivo in grado di trasformare segnali elettrici in immagini: esso utilizza i campi elettrici per generare fasci di elettroni ad alta velocità, che, opportunamente indirizzati e deviati, vanno a colpire uno schermo rivestito di materiale fosforescente provocando emissione di luce.

Il primo calcolatore dotato di un CRT come dispositivo di output grafico è stato Whirlwind I, progettato e divenuto operativo nel 1950 presso il Massachusetts Institute of Technology (MIT, [A03]). Successivamente, nel corso degli anni sessanta, i dispositivi di output grafico sono apparsi anche sul mercato. Da allora il CRT ha mantenuto un ruolo dominante come tecnologia di output (più per il costo relativamente basso e la disponibilità che per una effettiva superiorità tecnologica) e le sue proprietà e limitazioni hanno avuto un effetto profondo

⁶ CRT: Cathode Ray Tube.

sugli sviluppi della computer graphics. Solo il recente sviluppo delle tecnologie dello stato solido potrebbe ridurre, a lungo termine, il ruolo dominante del CRT.

I primi dispositivi di output grafico entrati in commercio negli anni sessanta, e rimasti in uso fino alla metà degli anni ottanta, erano basati sul concetto di grafica vettoriale (dove il termine vettoriale è da intendersi come sinonimo di lineare). La caratteristica principale di questi dispositivi è che il fascio di elettroni, che va a colpire il rivestimento fosforescente del CRT, può muoversi direttamente da una posizione all'altra, secondo l'ordine arbitrario dei comandi di display. Chiaramente, annullando l'intensità del fascio, questo può essere spostato in una nuova posizione, senza modificare l'immagine visibile. Questa tecnica, chiamata random scan (scansione casuale), è rimasta in uso fino agli anni settanta, quando hanno cominciato a diffondersi i sistemi di grafica raster, basati sulla tecnologia televisiva.

Nella grafica raster, ogni immagine è rappresentata tramite una matrice, chiamata appunto raster, di elementi, o pixel⁷, ciascuno dei quali corrisponde ad una piccola area dell'immagine. Anziché trattare con linee e punti, posizionati casualmente sulla superficie di visualizzazione del CRT, l'elaborazione delle immagini è dunque basata su matrici di pixel che rappresentano l'intera area dello schermo. Un CRT a scansione raster percorre, con il suo fascio di elettroni, la matrice di pixel; la scansione dell'immagine viene fatta sequenzialmente, e l'intensità del fascio di elettroni viene regolata in modo da riflettere l'intensità di ciascun pixel.

La tecnologia raster ha consentito l'introduzione del colore, realizzato controllando tre fasci di elettroni, relativi ai tre colori primari rosso, verde e blu, così come specificato in corrispondenza di ciascun pixel.

E' evidente che la tecnologia raster richiede la disponibilità di memorie di capacità elevata: intere griglie, diciamo di 1024 linee di 1024 pixel ciascuna,

⁷ Pixel: picture element.

devono infatti essere memorizzate esplicitamente. Nella grafica vettoriale vi è invece una minore necessità di memoria: per visualizzare una linea è sufficiente memorizzarne gli estremi.

Negli anni ottanta lo spettacolare progresso della tecnologia a semiconduttori, che ha reso disponibili multiprocessori e memorie a basso costo, ha consentito la creazione e la diffusione di interfacce grafiche per personal computer basate sulla tecnologia raster. La progressiva diminuzione dei costi, ha dunque contribuito fortemente alla diffusione dei sistemi raster, al punto che oggi essi rappresentano la tecnologia hardware dominante. Più in generale, il progresso tecnologico degli ultimi vent'anni, ha contribuito moltissimo anche alla crescita e allo sviluppo della computer graphics, che da disciplina altamente specializzata e costosa, è diventata un mezzo standard di interazione con il calcolatore, accessibile a milioni di utenti. Oggi sono in circolazione sottosistemi di pochi chip in grado di visualizzare in tempo reale animazioni tridimensionali, con immagini a colori di oggetti complessi, tipicamente descritti da migliaia di poligoni. Questi sottosistemi possono essere aggiunti non solo alle workstation, ma anche ai personal computer. Inoltre, anche applicazioni quali il photorealistic rendering di oggetti su display raster, considerate fino a poco tempo fa irrealizzabili, fanno oggi parte dello stato dell'arte di questa disciplina.

Anche le tecnologie di input si sono fortemente sviluppate nel corso degli ultimi anni. Oggi, la quasi totalità dei sistemi grafici affianca alla tastiera almeno un altro dispositivo di input grafico: mouse, joystick o data tablet (tavoletta magnetica), solo per citare i più comuni. Questi dispositivi forniscono informazioni di tipo posizionale al sistema, e consentono all'utente di specificare operazioni e comunicare informazioni con un uso della tastiera ridotto al minimo.

Il progresso dei dispositivi hardware, ha infine determinato, in modo del tutto naturale, un'evoluzione delle applicazioni software. Siamo così passati dai pacchetti software di basso livello, forniti dalle industrie per i propri dispositivi

di display, agli attuali pacchetti applicativi di alto livello, progettati per essere indipendenti da qualsiasi periferica grafica di input e output, e dunque con una portabilità simile a quella dei linguaggi di programmazione di alto livello (quali FORTRAN, Pascal, o C).

La necessità di sviluppare degli standard grafici, emersa per la prima volta verso la metà degli anni settanta, ha portato nel 1977 alla specificazione del primo sistema grafico (s. g.) tridimensionale, il cosiddetto Core Graphic System (in breve, CORE), prodotto da un gruppo di professionisti afferenti alla Association for Computing Machinery (ACM, [A04]), poi successivamente ridefinito nel 1979. Il primo pacchetto grafico ufficialmente standardizzato è stato il Graphical Kernel System (GKS, [A05]), una versione ripulita ed elaborata di CORE, però ristretta alla grafica bidimensionale, standardizzato dall'ANSI [A06] nel 1985. Successivamente, nel 1988, un'estensione tridimensionale di GKS, GKS-3D, è diventata uno standard ufficiale, così come un altro sistema grafico ancora più complesso, il PHIGS⁸, trasformato nel 1992 nello standard PHIGS PLUS, un sistema in grado di gestire lo pseudorealistic rendering di oggetti sui display raster.

Oltre agli standard ufficiali, promulgati dalle associazioni internazionali o professionali, sono oggi in circolazione anche standard grafici industriali, sviluppati e commercializzati da compagnie industriali o consorzi di università ed industrie. Tra i più noti, ricordiamo Adobe's PostScript [A06], Silicon Graphics' OpenGL, Ithaca Software's HOOPS, ed il MIT-led X-Consortium's X Window System. Gli standard industriali sono quelli sicuramente più importanti dal punto di vista commerciale, poiché possono essere aggiornati molto più rapidamente rispetto agli standard ufficiali.

1.3 I sistemi grafici interattivi

Il settore della computer graphics che riguarda in modo diretto la progettazione di sistemi grafici (s. g.), che consentono all'utente di interagire con il computer,

⁸ PHIGS: Programmer's Hierarchical Interactive Graphics System.

è chiamato computer graphics interattiva. Con il termine interattiva si vuole distinguere questo settore da altri rami della computer graphics in cui le immagini sono invece generate tramite plotter digitali, film recorder, stampanti, o comunque dispositivi che generano immagini permanenti. Certamente anche questi dispositivi risultano estremamente utili; tuttavia essi non possono essere realmente utilizzati per comunicare con un computer. Al contrario, un computer provvisto di un display è in grado di modificare velocemente le immagini generate, e dunque può rispondere in modo tempestivo alle sollecitazioni dell'utente; in questo modo utente e computer possono effettivamente comunicare tra loro. Le immagini non sono statiche, bensì dinamiche: gli oggetti possono essere mossi, ruotati, ingranditi, possono cambiare forma e colore. Il display di un computer presenta inoltre il vantaggio di permettere di elaborare e costruire immagini di oggetti non solo del mondo reale, ma anche di oggetti astratti, sintetici e di dati che non presentano un'inerente geometria.

1.4 Sistema grafico a livello hardware

Un computer riceve dati da un dispositivo di input, e invia delle immagini ad un display.

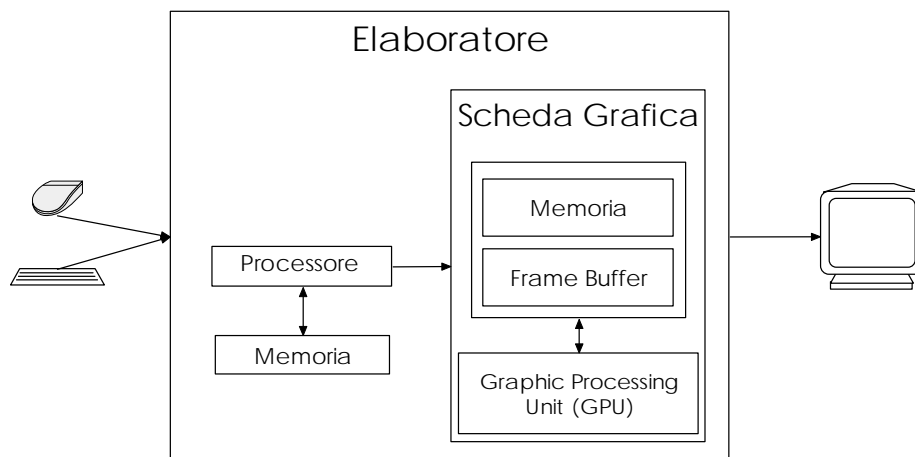


Figura 1.1: schema di un s. g. a livello hardware

Il *Frame Buffer* è una porzione di memoria dedicata alla memorizzazione dell'immagine come insieme di pixel da mostrare a video; la struttura a matrice

del frame buffer riproduce la struttura a maschera del CRT raster o della matrice LCD, in base al dispositivo.

La *Scheda Grafica* (S.G.) è un particolare dispositivo hardware atto ad elaborare dati grafici e a pilotare il display raster; la tecnologia moderna ha portato la creazione di schede con una memoria ed un processore interni indipendenti dal resto del sistema: il *frame buffer* della S.G. è utilizzato per immagazzinare l'immagine da visualizzare sul display; il *Graphic Processing Unit* (GPU) compie elaborazioni specifiche necessarie per formare l'immagine; la *Graphic Memory* è una memoria ausiliaria per compiere tale operazioni.

1.5 Sistema grafico a livello software

Dal punto di vista software, si considerano tre componenti.

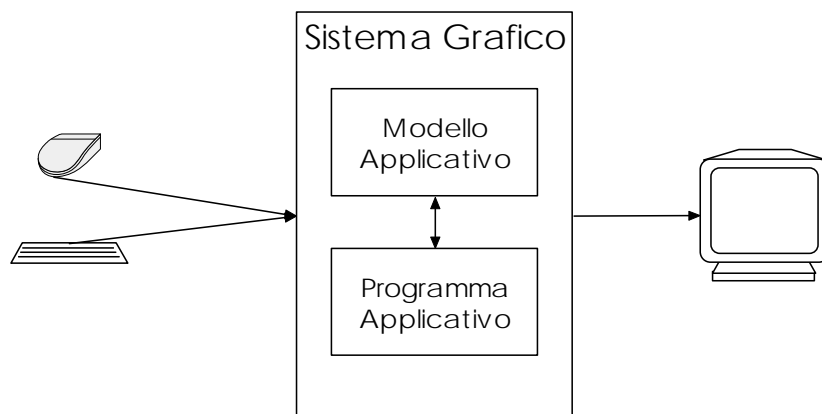


Figura 1.2: schema concettuale di un s. g. a livello software

La prima, il programma applicativo, è responsabile della creazione e dell'aggiornamento, sulla base delle interazioni con l'utente, dei dati o degli oggetti da raffigurare sullo schermo. Il modello applicativo produce le immagini inviando al sistema grafico una serie di comandi grafici di output che contengono sia una descrizione geometrica dettagliata di cosa deve essere visualizzato, sia gli attributi che specificano come gli oggetti dovranno apparire.

Infine, il sistema grafico produce le immagini, e trasferisce l'input dell'utente al programma applicativo.

Il sistema grafico agisce dunque da intermediario tra il programma applicativo ed il display hardware. Esso non solo trasforma gli oggetti descritti nel modello in immagini, ma anche le azioni dell'utente in input per il programma che, di conseguenza, modificherà il modello.

Il compito principale di chi progetta un programma grafico interattivo è quello di specificare quali classi di dati e oggetti sono da generare e rappresentare graficamente, ed il modo col quale l'utente ed il programma applicativo possono interagire per creare e modificare il modello e la sua rappresentazione visuale. Il maggior carico del lavoro di programmazione consiste dunque nella creazione del modello e nella gestione delle interazioni con l'utente, piuttosto che nell'effettiva creazione di immagini, che è invece un compito svolto dal sistema grafico.

Un modello di dati può essere rudimentale come un vettore di dati puntuali, o sofisticato quanto una lista con puntatori che rappresenta un database relazionale per la memorizzazione di un insieme di relazioni. Esso memorizza tipicamente le descrizioni di alcune primitive grafiche (punti, linee, curve e poligoni in 2 o 3 dimensioni, poliedri e superfici in 3 dimensioni) che definiscono la forma delle componenti dell'oggetto, insieme ai suoi attributi, ovvero stile grafico delle linee, colore, struttura delle superfici, e alle relazioni di connettività e posizionamento dei dati che descrivono come mettere insieme le varie componenti.

L'utente può chiedere in qualsiasi momento al programma applicativo di mostrare una rappresentazione visuale del modello creato fino a quel punto. Il programma deve pertanto convertire una descrizione della porzione del modello da visualizzare, dalla rappresentazione interna della sua geometria in procedure o comandi utilizzabili dal sistema grafico per creare l'immagine.

Questo processo di conversione avviene in due fasi. Per prima cosa il programma estrae dal database che memorizza il modello, la porzione da visualizzare, utilizzando qualche criterio di selezione. Quindi, i dati estratti ed i

loro attributi sono convertiti in un formato inviabile al sistema grafico. I dati estratti dal database devono essere di tipo geometrico, oppure devono essere convertiti in dati geometrici; inoltre essi possono essere descritti al sistema grafico in termini sia di primitive che il sistema può immediatamente visualizzare, sia di attributi che controllano l'aspetto delle primitive.

Le primitive visualizzate solitamente coincidono con quelle memorizzate nel modello. Il sistema grafico consiste tipicamente di un insieme di subroutine di output, corrispondenti a primitive, attributi ed altri elementi, raccolte in una libreria grafica che può essere chiamata dai linguaggi di programmazione di alto livello quali C, C++, Java, etc..

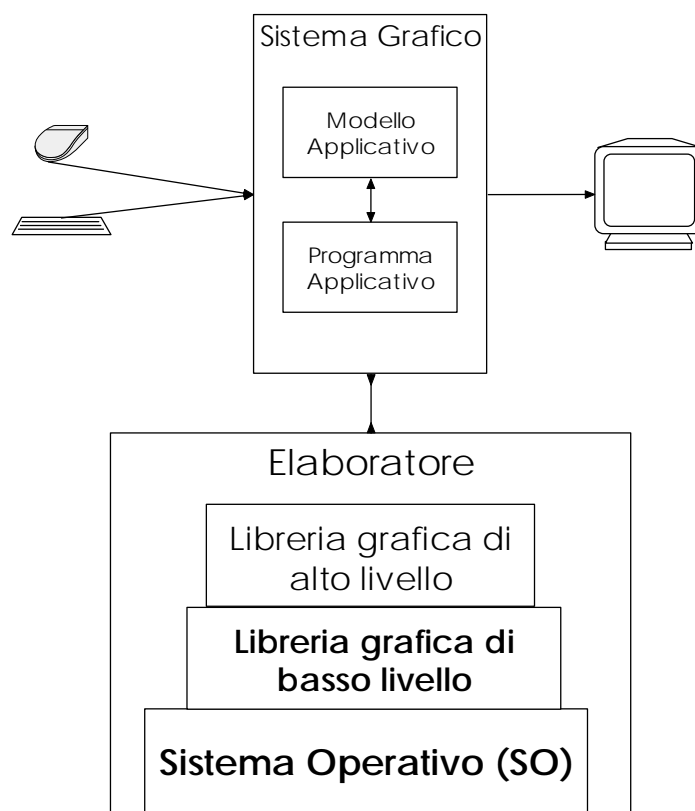


Figura 1.3: progettazione di un s. g. a livello software

La libreria grafica costituisce dunque l'interfaccia tra programma applicativo e sistema grafico: il programma specifica le primitive geometriche e gli attributi alle subroutine, e quindi le subroutine guidano il dispositivo di display, e

producono l'immagine. In questo modo, il programma rimane separato dai dettagli sia hardware che software dell'implementazione della libreria grafica.

Le librerie grafiche di basso livello creano oggetti grafici a partire dal singolo pixel fino a generare forme geometriche in base a procedimenti matematici, cioè le primitive.

Le librerie grafiche di alto livello permettono, invece, di generare qualsiasi oggetto grafico evitando di richiamare in ogni momento le primitive che lo generano.

In maniera più dettagliata verrà affrontato l'argomento delle librerie grafiche. Innanzitutto è importante mantenere l'attenzione sul fatto che per creare qualsiasi applicazione di CG è necessario fare uso sia di librerie grafiche di basso livello che di alto livello.

Nel caso della nostra applicazione si inserisce inoltre un altro intermezzo allo schema sopra mostrato, che consente di gestire contesti grafici indipendentemente dal particolare sistema operativo (SO) presente nella macchina in uso.



Figura 1.4: struttura della progettazione dell'applicazione

Gli argomenti dei prossimi capitoli seguono tale schema; ad ogni gradino di questa scala è stata associata una libreria utilizzata nel progetto. La progettazione dell'applicazione viene così schematizzata:

- 1) Per la creazione di una finestra per l'interfaccia grafica utente: *Simple DirectMedia Layer (SDL)*;
- 2) Per l'elaborazione di immagini in 2 o 3 dimensioni: *Open Graphic Library (OpenGL)*;
- 3) Per l'elaborazione di strutture tridimensionali: *Open Scene Graph (OSG)*.

E lo schema risultante sarà:

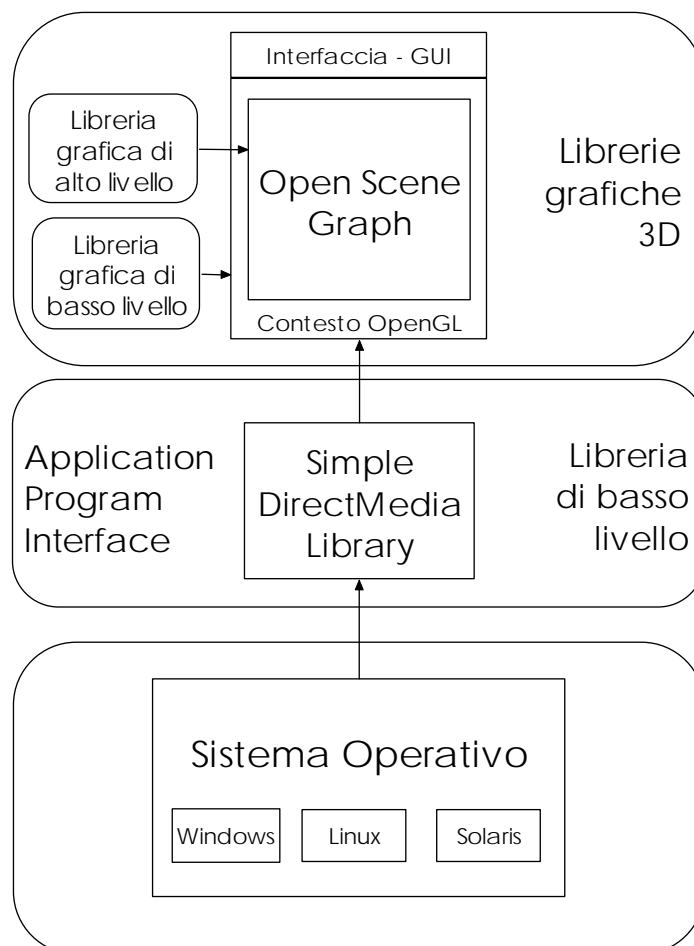


Figura 1.5: librerie utilizzate ai vari livelli di programmazione

Capitolo 2

Application Program Interface

In questo capitolo verranno date alcune definizioni di API (*Application Program Interface*) che permettono di utilizzare gli strumenti di un computer, dalle caratteristiche hardware a quelle software. L'argomentazione verterà soprattutto su quelle librerie che consentono di gestire le proprietà grafiche di un computer e del suo sistema operativo. A tale proposito verranno illustrate le principali caratteristiche di *Simple DirectMedia Layer* (SDL), un API di basso livello, che fornisce un ottimo esempio di libreria *multi-piattaforma* e *open source*. Questa è in grado di girare nei principali sistemi operativi (Windows [B01], Machintosh, Linux [B02], Unix [B03], Solaris [B04]) ed è completamente gratuita.

2.1 Definizione

Una *Application Program (Programming) Interface* (API) è un insieme di definizioni delle modalità grazie alle quali un software può mettere in comunicazione ogni parte di un elaboratore con un'altra. È un metodo per raggiungere un'astrazione, di solito (ma non necessariamente) tra software di basso-livello e software di alto-livello.

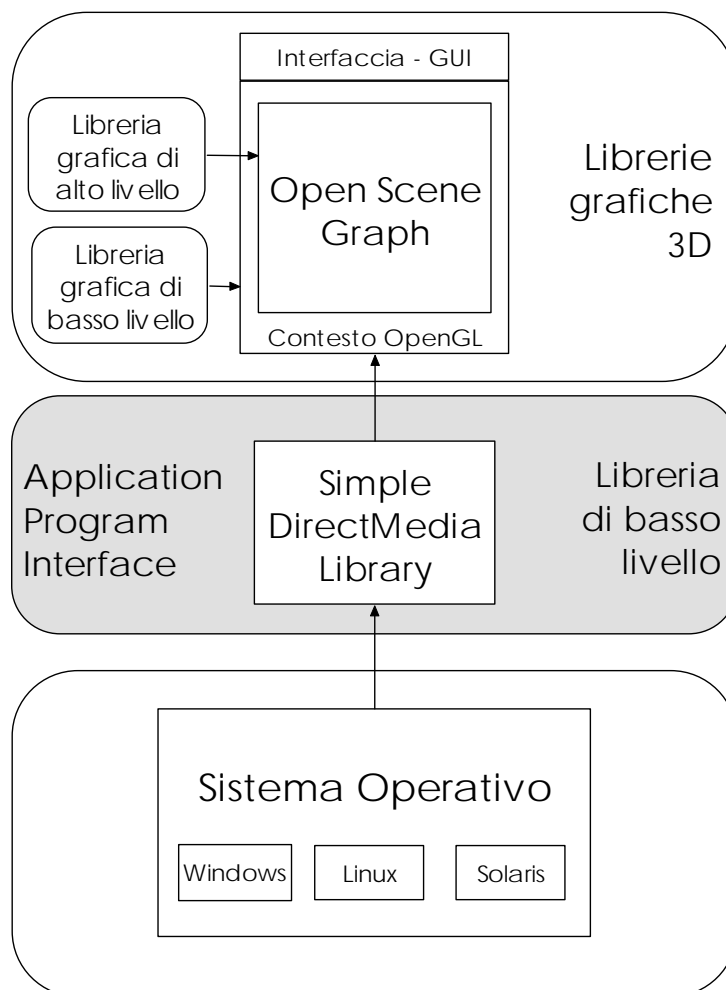


Figura 2.1: struttura del progetto di tesi - API

Le API sono essenziali ai computer, per esempio, tanto quanto lo sono le prese elettriche in una casa. Ognuno può collegare un toaster in una presa a muro di casa sua o di un suo vicino, poiché entrambi le case sono conformi alla stessa

interfaccia standard di presa elettrica. Se non fosse esistita un'interfaccia standard, ognuno avrebbe dovuto portarsi dietro un generatore di corrente per farsi un toast! Si può notare implicitamente che non c'è niente che vieti che qualcun altro presenti un altro standard: infatti un toaster costruito in Europa non funzionerà negli Stati Uniti senza un giusto trasformatore, proprio come un programma scritto per Windows non funzionerà direttamente su Unix senza un API intermedia che funge da adattatore.

Uno dei principali scopi di un'API è di fornire un insieme di funzioni usate comunemente, per esempio per disegnare finestre o icone sullo schermo. I programmatori possono così prenderne i vantaggi, per creare funzioni per programmare qualsiasi cosa in un attimo. Le API stesse sono astratte: il software che fornisce una determinata API è spesso chiamata *implementazione* di quella API.

Per esempio, si può considerare l'operazione di scrivere "Hello, World" su schermo a vari livelli di astrazione:

1. Eseguire tale lavoro senza comunicare con il sistema operativo:
 - 1.1. Disegnare, su di un documento multimediale grafico, le forme delle lettere H, e, l, l, o, W, o, r, l, d;
 - 1.2. Elaborare una matrice di piccoli quadrati bianchi e neri che ha la forma di quelle lettere;
 - 1.3. Escogitare un modo per programmare la CPU per mettere questa matrice nel frame buffer del display;
 - 1.4. Impostare la scheda grafica per aggiornare il proprio frame buffer in modo da generare il corretto segnale;
2. Utilizzare il SO per compiere alcune parti del lavoro:
 - 2.1. Caricare una struttura dati detta "font" fornita dal sistema operativo;
 - 2.2. Ottenere dal SO la creazione di una finestra vuota;
 - 2.3. Ottenere dal SO il disegno della scritta "Hello, World" sulla finestra precedentemente creata;

3. Utilizzare un programma-applicazione (che usa il sistema operativo) per svolgere l'intero lavoro:
 - 3.1. Scrivere un documento HTML contenente la scritta "Hello, World";
 - 3.2. Aprire tale documento in un browser come Mozilla o Internet Explorer

Sicuramente la prima opzione richiede più passi, ognuno dei quali più complicato del passo precedente. Possiamo perciò dire che è impraticabile utilizzare il primo approccio per presentare a schermo un ingente numero di informazioni, mentre il secondo approccio rende tale compito più semplice di un passo, e infine il terzo richiede solo che sia digitata la scritta "Hello, World".

Tuttavia le API di alto livello possono perdere flessibilità; per esempio sarebbe molto complicato ruotare, in un browser, un testo attorno ad un punto, cosa che invece potrebbe essere svolta facilmente a un livello basso.

Ci sono vari modelli di progettazione per le API. Le interfacce realizzate per le esecuzioni più veloci spesso consistono di insiemi di funzioni, procedure, variabili e strutture dati. Una buona API fornisce una "black box" o strato astratto, che evita al programmatore di conoscere le relazioni tra le funzioni delle API e i più bassi di astrazione. Ciò dà la possibilità di ridisegnare o migliorare le funzioni al di sopra delle API senza dover decifrare il codice che fa affidamento ad esso.

2.2 API Open Source

Esistono universalmente due politiche generali che riguardano la divulgazione delle API:

1. Alcune compagnie proteggono le proprie API premurosamente. Per esempio, la Sony rende disponibili le sue API per PlayStation solo agli sviluppatori ufficiali. Questo perché la Sony vuole restringere il numero

di programmatori di giochi, e vuole trarne profitto il più possibile. Ciò è tipico delle compagnie che non guadagnano dalla vendita di API;

2. Altre compagnie propagano le loro API liberamente. Per esempio, la Microsoft rende pubbliche le proprie API, cosicché un software sarà scritto per la piattaforma Windows. La vendita di software realizzata da terzi incrementa la diffusione di copie di Windows. Questo è tipico di compagnie che traggono profitto dalla vendita di API.

Alcune API, come quelle standard per un dato sistema operativo, sono implementate come librerie con codice separato che sono così distribuite con il sistema operativo. Altre richiedono software di pubblicazione per integrare le API direttamente nell'applicazione. Ciò costituisce un'altra distinzione dagli esempi sopra visti. Le API di Microsoft Windows sono presenti insieme al sistema operativo per l'uso di ognuno. I software per i sistemi integrati, come le console di video game, generalmente si distinguono dalla categoria di applicazione integrate. Se da una parte un documento ufficiale di un'API PlayStation può essere interessante da leggere, dall'altra è di poco uso senza la sua corrispondente implementazione, cioè presa come una libreria particolare o kit di sviluppo.

Un API che non richiede particolari privilegi per l'uso e l'accesso è detta "open". Le API prodotte da un Free software (come i software distribuiti sotto la GNU General Public License [B05]) sono "open" per definizione, dal momento che ognuno può ottenere i sorgenti del software e riuscire a interpretare le API. Anche se di solito esistono implementazioni autorevoli di riferimento per le API (come Microsoft Windows per le API Win32), niente impedisce la creazione di implementazioni aggiuntive. Per esempio, più di un'API può essere convertita per sistema Unix usando un software chiamato WINE [B06].

2.3 L'Open Source e il multi-piattaforma

L'argomento relativo alle due politiche sopra analizzate coinvolge l'intero campo delle librerie grafiche. Tale discussione è uno dei punti cardine che ha portato allo sviluppo della tesi dal punto di vista della scelta accurata delle API.

Con l'avvento dei personal computer, l'informatica si è diffusa in un decennio fino a condizionare la vita quotidiana delle persone e delle loro attività professionali. Tale velocità di sviluppo tuttavia non le ha consentito di avere le stesse caratteristiche delle altre industrie produttive, avere cioè numerosi fornitori in concorrenza fra di loro. Il software è oggi nella mani di pochi grandi gruppi, che impongono i loro standard proprietari e guidano l'evoluzione più in base ai loro interessi che alla convenienza degli utenti.

Come alternativa a tutto ciò è nato il movimento *Open Source* [B07], divenuto popolare soprattutto grazie alla diffusione di Internet, che è stato ed è tuttora uno dei motori propulsivi dello sviluppo del web.

Gran parte dell'infrastruttura tecnologica di Internet è basata su standard aperti e software *Open Source*, rilasciato con licenze non restrittive sia nella distribuzione che nell'uso, come GPL⁹, LGPL¹⁰, etc. Il più delle volte è utilizzabile gratuitamente, ma la cosa ancor più entusiasmante, soprattutto per i programmatori è il codice, che può essere studiato, analizzato, modificato, ampliato ed esteso. Viene sviluppato in progetti collaborativi, che spesso coinvolgono sviluppatori da tutto il mondo, provenienti dalle esperienze più diverse. In molti casi questo tipo di software risulta essere più affidabile, e funzionale dell'analogo software proprietario e commerciale.

Questa modalità di sviluppo degli applicativi favorisce la competizione, in quanto induce i produttori di software ad accordarsi sulle tecnologie, che pertanto devono essere aperte, basate su standard e soddisfare le necessità del

⁹ GPL: General Public License, garantisce la libertà di condividere e modificare il software libero.

¹⁰ LGPL: Lesser General Public License, una licenza pubblica meno generale.

maggior numero di utenti. Molti sono gli esempi di successo di questo modello: prodotti come OpenOffice [B08] o Linux si propongono come seria alternativa a prodotti e sistemi operativi proprietari.

Una caratteristica oramai standard dei progetti *Open Source* di maggior successo, e ora anche di molti altri prodotti, è di essere multi-piattaforma. Consideriamo il caso di una piattaforma hardware e prendiamo per esempio il sistema operativo Linux; inizialmente si compilava solamente su processori Intel, dal 386 in su. Nato come progetto per un'architettura hardware come il PC, Linux si è evoluto ed ora è in grado di funzionare su molte altre piattaforme hardware. Oggi Linux si compila e funziona anche su processori Alpha, Arm, PowerPC, etc.

C'è da aggiungere inoltre che è compito del linguaggio di programmazione di astrarre l'applicazione del sistema operativo sottostante, come il compito del sistema operativo è di astrarre dall'hardware sottostante. Con il tempo, i sistemi di programmazione multi-piattaforma si sono fatti sempre più sofisticati e oggi si preferiscono ad altri [B09].

Per esempio, fino a qualche anno fa si programmava per Windows o X11 [B10], chiamando direttamente il sistema operativo perché non esistevano modalità alternative; oggi è possibile usare C/C++ con librerie portabili tra Windows e X11 come Qt, wxWindows, Gtk, SDL oppure è possibile utilizzare linguaggi interpretati come Java o Python, che supportano lo sviluppo di applicazioni funzionanti in maniera pressoché identica su più sistemi operativi.

Quale sia la piattaforma verso la quale gli utenti si indirizzeranno nei prossimi anni è una questione aperta. Al giorno d'oggi la maggior parte degli utenti possiedono prodotti Microsoft; allo stesso tempo però c'è la tendenza, oramai irreversibile, a sfruttare le applicazioni multi-piattaforma *Open Source*, soprattutto per ragioni di costi. Questa tendenza porterà probabilmente ad accantonare l'idea di un sistema operativo proprietario.

Si può affermare quindi che il multi-piattaforma nasce come conseguenza dei software *Open Source* e che l'avvento di nuove piattaforme ha richiesto sempre più al software di essere multi-piattaforma.

2.3.1 I vantaggi del multi-piattaforma

In ambienti di sviluppo non multi-piattaforma, le informazioni possono essere universalmente comprese e trasferite, ma i programmi rimangono sempre intimamente legati alla piattaforma software per la quale sono realizzati indipendentemente dall' hardware su cui girano. Esistono comunque dei linguaggi, come il C, che possono essere impiegati su più sistemi operativi. Questo perché attraverso uno strumento chiamato compilatore, il codice scritto dal programmatore viene tradotto nel linguaggio macchina specifico della piattaforma utilizzata. Spostandoci da un sistema all'altro, e utilizzando il compilatore adeguato all'architettura, si possono ottenere più versioni dello stesso programma. Tutto questo però avviene solo in linea teorica. Per compiere operazioni basilari come mostrare una semplice finestra, ciascun software deve agganciarsi alle funzionalità offerte dal SO in uso e dal sistema grafico. Ogni piattaforma in base alla propria architettura hardware e software, fornisce questi servizi in maniera diversa. Da ciò si deduce che il porting di un programma prevede sempre due fasi:

1. la riscrittura di tutte le parti che si occupano dell'interazione con il sistema operativo;
2. la ricompilazione del sorgente sulla macchina.

La prima operazione può risultare ardua e sconveniente ed è per questo motivo che molti programmi sono disponibili solo per alcuni sistemi operativi.

Un programma in esecuzione (processo) è inizialmente presente in un file chiamato binario salvato su un supporto di memorizzazione adeguato; tali file

presentano una struttura particolare che indica alla macchina come essi devono essere eseguiti.

Ogni architettura HW presenta un formato particolare di questi file che dipende dal codice macchina della stessa, ma non solo: ogni sistema operativo presenta un formato binario che dipende dalla struttura e dalle scelte progettuali con cui è stato realizzato.

I formati binari indicano come il programma deve essere eseguito sul sistema operativo: come caricare il codice in memoria, stack, gestione dei registri ecc.

Utilizzare formati binari comuni tra varie architetture e SO permette una gestione coerente del sistema su cui girano i programmi. Poiché i vari formati non sono tra di loro compatibili non è possibile utilizzare ad esempio eseguibili Windows in un ambiente operativo Unix, anche se la macchina per cui sono progettati i relativi applicativi è la stessa fisicamente.

I programmi multi-piattaforma sono realizzati per essere eseguiti su sistemi operativi differenti, necessitano solo della fase di ricompilazione, senza quindi la riscrittura di alcune delle loro parti. Nello sviluppare un'applicazione per un'unica piattaforma ci si preoccupa solo dei problemi relativi ad essa. Rendere in seguito quella applicazione multi-piattaforma può mettere in evidenza bugs che si pensava non esistessero. È pertanto preferibile spendere più tempo in fase di progettazione del software, in quanto se un programma è ben sviluppato, i costi per trasferirlo su una nuova piattaforma saranno minori. I vantaggi di utilizzare un approccio multi-piattaforma sono evidenti:

1. non c'è bisogno di programmatori specializzati in ciascun sistema operativo per ottenere le conversioni richieste;
2. non bisogna distribuire più pacchetti differenti di un medesimo programma, con una conseguente riduzione dei costi di manutenzione;

3. non si deve mai escludere una specifica piattaforma dal proprio target semplicemente perché si è valutata sconveniente un'ipotetica conversione.

Il motto dei programmatori multi-piattaforma è *Write Once, Run Anywhere*¹¹.

Sviluppare un' applicazione multi-piattaforma richiede comunque costi aggiuntivi da tenere in considerazione:

1. Il tempo aggiuntivo e lo sforzo umano per creare codice multi-piattaforma astratto;
2. Un aspetto riguarda l'adattamento di almeno una parte del codice per differenti piattaforme, che è quasi sempre necessario;
3. Una parte che riguarda il testing e il debugging per assicurare che il prodotto possa funzionare correttamente su diverse piattaforme.

A volte apportare anche semplici e piccole modifiche al codice per una particolare piattaforma, può creare grossi problemi al flusso logico. Tener traccia di tutte le variazioni in fase di test e apportare i dovuti accorgimenti non è un'operazione tanto facile, soprattutto se, chi lavora sopra ad una specifica applicazione, è più di una persona. Sviluppare prodotti multi-piattaforma funzionanti obbliga i programmatori a scrivere codice che non incorpori alcuna interfaccia o trucco di un particolare sistema operativo o piattaforma hardware. Si devono usare interfacce relativamente semplici e di basso livello, affinché siano comuni alle differenti piattaforme.

Per poter progettare un'applicazione multi-piattaforma qualsiasi, software o gioco che sia, bisogna procedere con logica e seguire alcune regole fondamentali:

¹¹ Traduzione: Scrivi una volta, esegui ovunque.

1. Pensare prima di scrivere il codice. Sembra banale, ma in realtà molti programmatori, soprattutto nell'industria del gioco, iniziano a scrivere codice senza nemmeno programmare la struttura del software. Così facendo si crea, molto facilmente, un codice disordinato e difficile alla comprensione e sarà ancora più problematico renderlo portabile. Il programmatore per sviluppare un buon progetto deve avere, come punto di riferimento, un modello analizzato a priori ed essere anche pronto alle critiche delle persone e preparato a modificare parti intere di codice. Fare uno sforzo iniziale, nel progettare il software, rende il programma più durevole nel tempo.
2. Fare astrazioni. Se si sta scrivendo un gioco per Windows, prima o poi, si andrà a usare una specifica funzione di sistema. Occorrerà quindi tempo per modificare la struttura del codice ed adattarlo alla nuova piattaforma. Fare una buona astrazione porta a sviluppare un buon progetto con tutti i suoi benefici.
3. Fare attenzione all'ordinamento e al pacchettamento dei byte (endianess). Nel caso di giochi online, può capitare di imbattersi in un messaggio di errore come *sizeof(myStructure)* nella connessione di rete. Bisogna utilizzare con accuratezza i numeri a virgola mobile: in certi casi, infatti, molte CPU hanno una diversa precisione, e pertanto nella comunicazione dei dati si può avere un errore di trasmissione.
4. Scrivere ciò che è necessario e trovare il resto in rete. Il modo più semplice per sviluppare un gioco o un software è quello di reperire materiale in Internet. Infatti non ha senso scrivere da zero librerie per la codifica delle immagini o dell'audio, quando possono essere semplicemente reperiti nella comunità Open Source. Quindi è utile sfruttare appieno tutte le risorse possibili che si possono reperire in rete.

Rimane sempre da analizzare una delle questioni chiave nello sviluppare in multi-piattaforma, cioè quella di rendere portabile la GUI. Una soluzione sarebbe quella di sviluppare GUI front-end indipendenti, garantendo una grande flessibilità e permettendo un certo adattamento all'interfaccia di sistema, ma questo sarebbe come aggirare il problema e non affrontarlo. Non c'è da meravigliarsi se non siamo in grado di astrarre alcuni concetti come la creazione di bottoni o disegnare un particolare oggetto. Fortunatamente esistono dei toolkit *GUI* in grado di risolvere questo tipo di problema. Così invece di inventare il programma partendo da zero si ha la possibilità di usare un particolare toolkit già esistente.

2.4 Simple DirectMedia Layer

Simple DirectMedia Layer (SDL) è una API gratuita multi-piattaforma per lo sviluppo multi-media, usata per la creazione di giochi, sistemi di sviluppo (*Software Development Kit*, SDK) di giochi, demo, emulatori, MPEG players¹² e tante altre applicazioni. Permette un accesso di basso livello alle periferiche multimediali, ovvero all'audio, al video 2D, al video 3D tramite OpenGL, alla tastiera, al mouse, al joystick ed al cdrom, inoltre fornisce funzioni per la gestione del tempo e dei thread. Ovviamente SDL è utilizzata per lo sviluppo di applicazioni multimediali, ed in particolar modo di giochi, come esempi possiamo indicare Cube , The battle for Wesnorth , Never Winter Night e svariati porting della Loki Games .

La libreria può essere scaricata dal sito ufficiale insieme alla documentazione in formato HTML.

SDL permette all'utente di scrivere codice senza dover preoccuparsi di problemi legati al copyright e cosa ancor più interessante è libera, l'utente quindi non è soggetto ad eventuali pagamenti per l'uso commerciale o privato. L'utente che cerca una libreria che permetta di scrivere codice senza preoccuparsi in seguito

¹² Applicazioni di elaborazione su file di tipo MPEG, Moving Picture Experts Group.

di dover modificare parti di esso, trova in SDL ciò che fa al caso suo, in quanto questa libreria gli dà la possibilità di scrivere codice per un ampio settore di dispositivi e sistemi operativi. Bisogna tener presente di questa potenzialità soprattutto in questi ultimi anni che hanno visto una crescita esponenziale di dispositivi quali: palmari, computer portatili e telefoni cellulari di ultima generazione. Non avrebbe senso infatti legarsi ad un solo SO o ad una sola architettura.

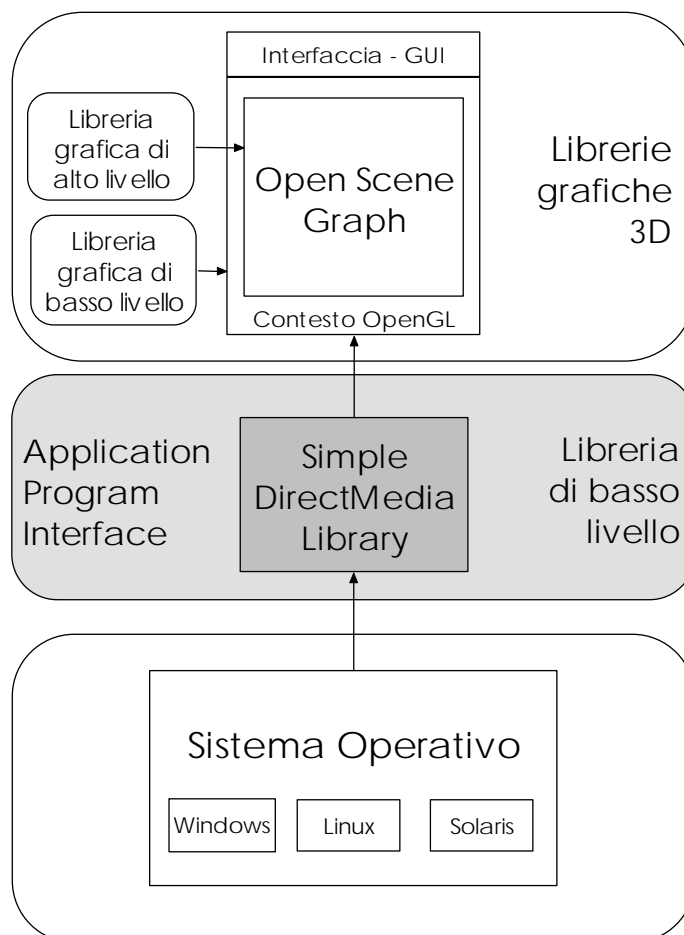


Figura 2.2: struttura del progetto di tesi - SDL

2.4.1 Inizializzazione

Dopo una breve descrizione è utile scendere nel dettaglio e vedere quali sono i particolari dell'architettura interna. Il set di funzionamento che offre SDL può essere approssimativamente diviso in sette sezioni principali:

1. Video;
2. Gestione finestre;
3. Gestione eventi;
4. Audio;
5. Interfaccia CD;
6. Multi-Threading;
7. Gestione timer.

Per attivarli è necessario fare una esplicita richiesta al momento dell'inizializzazione della libreria con la chiamata ad una funzione `int SDL_Init (Uint32 flags)` che viene chiamata all'inizio del programma e inizializza il modulo richiesto. Il parametro `flags` è ottenuto mediante la combinazione, tramite l'operatore OR, delle costanti corrispondenti ai moduli della libreria desiderati. Si ricorda di seguito quelle principali:

- `SDL_INIT_TIMER;`
- `SDL_INIT_AUDIO;`
- `SDL_INIT_VIDEO;`
- `SDL_INIT_CDROM;`
- `SDL_INIT_JOYSTICK.`

È possibile in qualsiasi momento attivare e disattivare i singoli sottoinsiemi facendo uso delle funzioni `SDL_InitSubSystem(Uint32 flags)` e `SDL_QuitSubSystem(Uint32 flags)` e controllare quali sono attivi con `SDL_WasInit(Uint32 flags)`, impostando e interpretando correttamente le maschere di bit secondo la loro definizione in libreria.

2.4.2 Un contesto grafico

La prima variabile dichiarata è un puntatore a `SDL_Surface`, la struttura chiave del sistema video di SDL. Si tratta di un tipo di dati che contiene informazioni

- 1) buffer immagine: quindi qualsiasi area di memoria nella quale è possibile disegnare;
- 2) dimensioni in pixel di questa;

- 3) eventuale area di clipping, all'interno della quale è prevista la possibilità di scrittura;
- 4) formato del pixel e altre cose ancora.

Molte funzioni SDL restituiscono un valore significativo sul risultato dell'operazione che compete a loro; in questo caso un valore negativo ritornato da `SDL_Init()` informerà il programmatore che l'operazione non è andata a buon fine e, tramite `SDL_GetError()`, otterrà una stringa descrittiva sull'ultimo errore riscontrato; è possibile naturalmente stamparla a video, oppure salvarla in un file di log, etc., per dare così informazioni importanti al programmatore su cosa è andato storto.

Una volta inizializzato il sottosistema video, è necessario creare una superficie grafica tramite la funzione `SDL_Surface *SDL_SetVideoMode(int width, int height, int bpp, Uint32 flags)` la quale ritorna in caso di successo un puntatore a `SDL_Surface`. L'ultimo argomento `flags` permette di impostare alcuni parametri aggiuntivi quali:

- `SDL_SWSURFACE`
- `SDL_HWSURFACE` – Alloca la superficie nella memoria della scheda video;
- `SDL_DOUBLEBUF` – Abilita il double buffering (valido solo con `SDL_HWSURFACE`);
- `SDL_FULLSCREEN` – Modalità fullscreen;
- `SDL_RESIZABLE` – Finestra ridimensionabile;
- `SDL_NOFRAME` – Bordi della finestra assenti, automatico se la `SDL_FULLSCREEN` è specificata;
- `SDL_OPENGL` – Sulla superficie è possibile usare OpenGL.

Naturalmente al programmatore è permesso usare combinazioni, qualora abbiano senso, con l'operatore binario *OR* (`|`); la lista completa è consultabile sulla documentazione online SDL. Si deve comunque fare attenzione in quanto alcuni attributi possono essere impostati solo per la surface video, come ad esempio `SDL_FULLSCREEN`, `SDL_DOUBLEBUF`, etc. Da aggiungere che con il flag `SDL_RESIZABLE`, attivo sebbene specifichi che la superficie è ridimensionabile,

la libreria non effettua il ridisegno di quanto contenuto in essa, ma si limita a generare un evento `SDL_VIDEORESIZE`, a seguito del quale l'applicazione deve interessarsi di ricavare le nuove dimensioni e gestire l'output grafico di conseguenza. Di seguito è presente un frammento di codice per la gestione degli eventi; in particolare si esce dal ciclo quando il programma riceve l'evento `SDL_QUIT` generato dalla pressione del tasto di chiusura della finestra del window manager.

2.4.3 I modi video disponibili

Dopo aver inizializzato la libreria è possibile utilizzare alcune funzioni per ottenere delle informazioni sulle capacità dell' hardware e conoscere quali modalità video sono utilizzabili.

Typedef struct {	
Uint32 hw_available:1;.....	Possibilità di creare una superficie video hardware
Uint32 ww_available:1;.....	Possibilità di interfacciarsi ad un window manager
Uint32 blit_hw:1;.....	Accelerazione per blit tra superfici video hardware
Uint32 blit_hw_CC:1;.....	Accelerazione per blit tra superfici video hardware con color key;
Uint32 blit_hw_A:1;.....	Accelerazione per blit tra superfici video hardware con canale alfa
Uint32 blit_sw:1;.....	Accelerazione per blit per superfici software a hardware
Uint32 blit_sw_CC:1;.....	Accelerazione per blit da superfici video software a hardware con colorkey
Uint32 blit_sw_A:1;.....	Accelerazione per blit da superfici video software a hardware con canale alfa
Uint32 blit_fill;.....	Accelerazione per riempimento a colore uniforme
Uint32 video_mem;.....	Quantità di memoria video disponibile in Kilobytes
SDL_PixelFormat *vfmt;.....	Pixel Format della superficie video
} SDL_VideoInfo;	

Tabella 2.1: struttura `SDL_VideoInfo`

Con la funzione `SDL_VideoDriverName()` è possibile ottenere una stringa che descrive il driver in uso, ad esempio X11 o DirectX [B13]. Le informazioni riguardanti l'hardware video si ricavano con `SDL_GetVideoinfo()`, che ritorna

un puntatore a una struttura `SDL_VideoInfo` (vedi tabella 2.1); è da notare che chiamando questa funzione prima di `SDL_SetVideoMode()`, ovvero dopo aver inizializzato la libreria ma non la superficie video primaria, verranno restituite all'interno del membro `wfmt` le informazioni relative al miglior Pixel Format disponibile. Una applicazione che debba girare su una gran varietà di hardware non può fare assunzioni di nessun tipo a proposito delle funzionalità che potrà sfruttare; è quindi molto utile avere a disposizione delle funzioni che permettano di capire quali modalità video siano disponibili e con quali caratteristiche. La prima funzione adatta a questo scopo è `SDL_VideoModeOK()`: nel caso in cui la risoluzione specificata non sia disponibile ritornerà 0; se invece è supportata ma ad essa non è applicabile la profondità di colore richiesta, la funzione restituirà la più simile utilizzabile. A volte possiamo avere necessità di applicare criteri diversi per la scelta della modalità video, tra quelle disponibili, o presentare all'utente le possibili alternative. Possiamo chiamare `SDL_ListModes()`, specificando il Pixel Format desiderato e i flag per la superficie video: la funzione ritorna un puntatore ad un array di strutture `SDL_Rect` che descrivono le risoluzioni disponibili o il valore 0 se non è disponibile alcuna risoluzione compatibile con i parametri specificati.

Capitolo 3

Open Graphics Library (*OpenGL*)

Una volta creata una *Surface* con l'SDL è possibile iniziare ad inserire elementi grafici. A questo punto si introduce l'OpenGL, come libreria che rispetta appieno le esigenze di multi piattaforma e open source. Il livello di programmazione di tale libreria permette di generare immagini tridimensionali partendo da funzioni base che creano punti e linee; lo sviluppo di queste tecniche porterà alla realizzazione di strutture dati ancor più complesse, come quelli presenti nell'*OpenSceneGraph*.

3.1 Le librerie grafiche

Il progresso dei dispositivi hardware di output grafico ha determinato, in modo del tutto naturale, un'evoluzione delle applicazioni software e ha portato alla realizzazione di librerie grafiche di alto livello, indipendenti da qualsiasi periferica grafica di input e output e con una portabilità simile a quella dei linguaggi di programmazione di alto livello (quali FORTRAN, Pascal, o C).

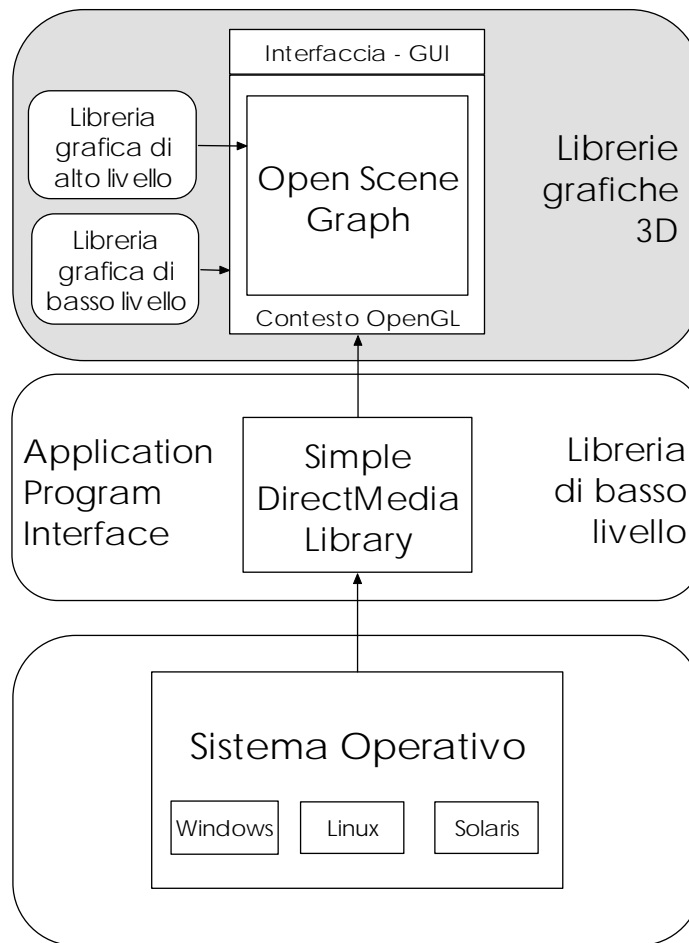


Figura 3.1: struttura del progetto di tesi – librerie grafiche

Il modello concettuale alla base delle prime librerie grafiche (bidimensionali) realizzate è quello ora definito modello *pen plotter*, con chiaro riferimento al dispositivo output allora disponibile. Un pen plotter produce immagini

muovendo una penna lungo due direzioni sulla carta; la penna può essere alzata e abbassata come richiesto per creare l'immagine desiderata. Diverse librerie grafiche (come LOGO, GKS, e PostScript) sebbene diverse l'una dall'altra, hanno in comune proprio il fatto di considerare il processo di creazione delle immagini simile al processo di disegnare una figura su un pezzo di carta: l'utente ha a disposizione una superficie bidimensionale, e vi muove sopra una penna lasciando un'immagine.

Questi sistemi grafici possono essere descritti con due funzioni di *drawing*:

```
moveto(x, y);  
lineto(x, y);
```

L'esecuzione della funzione `moveto` muove la penna alla locazione (x, y) sul foglio, senza lasciare segni. La funzione `lineto` muove la penna fino alla posizione (x, y) e disegna una linea dalla vecchia alla nuova locazione della penna. Aggiungendo qualche procedura di inizializzazione e terminazione, la possibilità di cambiare penna per variare il colore e lo spessore delle linee, abbiamo dunque un sistema grafico, semplice ma completo.

Per alcune applicazioni, quali il *layout* di pagina, i sistemi costruiti su questo modello sono sufficienti. Questo è il caso, ad esempio, del linguaggio PostScript, un'estensione sofisticata di queste idee di base oggi ampiamente utilizzato per la descrizione di pagine, in quanto rende relativamente facili gli scambi elettronici di documenti contenenti sia testo che immagini grafiche in due dimensioni.

Il sistema X window, ormai uno standard per le workstation UNIX, è usato sia per ottenere una finestra su un display grafico, in cui possono essere visualizzati, sia testi che grafica 2D, sia come mezzo standard per ottenere input da dispositivi quali tastiera e mouse. L'adozione di X dalla maggior parte di compagnie che realizzano workstation, ha avuto il seguente effetto: un singolo programma può produrre grafica 2D su un'ampia varietà di workstation, semplicemente ricompilando il programma. Questa integrazione funziona anche

sulle reti: il programma può essere eseguito su una workstation, ma può visualizzare l'output o ricevere l'input da un'altra, anche se le workstation in rete sono state costruite da compagnie diverse.

Il modello pen plotter non si estende bene ai sistemi grafici 3D. Ad esempio, se vogliamo usare un modello pen plotter per produrre un'immagine di un oggetto tridimensionale su una superficie bidimensionale, è necessario proiettare sulla nostra superficie punti nello spazio tridimensionale. Tuttavia, è preferibile utilizzare un'interfaccia grafica che consenta all'utente di lavorare direttamente nel dominio del problema, e di utilizzare il computer per eseguire i dettagli del processo di proiezione in modo automatico, senza che l'utente debba eseguire calcoli trigonometrici.

Per la grafica tridimensionale sono stati proposti diversi standard, ma ancora nessuno ha ottenuto un'ampia accettazione. Una libreria relativamente ben nota è il pacchetto PHIGS (dall'inglese *Programmer's Hierarchical Interactive Graphics System*), così come il suo discendente PHIGS+. Basata su GKS (*Graphics Kernel System*), PHIGS è uno standard ANSI (*American National Standard Institute*) che consente di manipolare e realizzare immagini di oggetti in tre dimensioni, incapsulando la descrizione degli oggetti e dei loro attributi in una *display list* cui si fa riferimento ogni volta che si visualizza o si deve manipolare l'oggetto. Il vantaggio delle *display list* è che è sufficiente una sola descrizione degli oggetti, anche se questi vengono poi visualizzati diverse volte, mentre un possibile svantaggio è dovuto allo sforzo considerevole che le *display list* richiedono per specificare nuovamente un oggetto quando esso viene modificato in seguito alle interazioni con l'utente.

La libreria PEX (acronimo per PHIGS *Extension to X*) è un'estensione di X che consente di manipolare e realizzare oggetti 3D, e di ottenere un *rendering* in modo immediato, ossia di visualizzare gli oggetti così come sono descritti senza dover prima compilare una *display list*. Una difficoltà legata all'uso della libreria PEX è dovuta al fatto che i fornitori di interfacce PEX hanno scelto di consentire caratteristiche diverse, rendendo la portabilità dei programmi

problematica. Inoltre, la libreria PEX è priva di caratteristiche di rendering avanzate, ed è disponibile solo agli utenti X.

OpenGL è una libreria grafica piuttosto recente, che consente di realizzare e manipolare immagini in due e tre dimensioni, e utilizza inoltre le tecniche più avanzate di rendering, sia in modo immediato, che attraverso display list. E' molto simile, sia nella sua funzionalità che interfaccia, a IRIS GL della Silicon Graphics¹³.

3.2 La libreria Open Graphics Library (*OpenGL*)

Come le librerie menzionate, OpenGL è un'interfaccia software per hardware grafico. L'interfaccia consiste di un insieme di diverse centinaia di funzioni e procedure che consentono al programmatore di specificare gli oggetti e le operazioni coinvolte nella produzione di immagine grafiche di alta qualità, quali immagini a colori di oggetti tridimensionali. Come il pacchetto PEX, OpenGL integra la grafica 3D nel sistema X, ma può essere integrato in altri sistemi window, o utilizzato senza alcun sistema window. Rispetto ad altre librerie, OpenGL è facile da imparare ed è anche piuttosto potente.

OpenGL è stata derivata dall'interfaccia GL, sviluppata per le workstation Silicon Graphics, e progettata per il rendering in tempo reale, ad alta velocità. OpenGL è il risultato dei tentativi di trasferire i vantaggi di GL ad altre piattaforme hardware: sono state rimosse funzioni input e windowing e ci si è concentrati sugli aspetti di rendering dell'interfaccia, realizzando così una libreria estremamente portabile, e conservando allo stesso tempo le caratteristiche che hanno fatto di GL un'interfaccia molto potente per i programmi di applicazione.

¹³ [Http://www.sgi.com/products/software/opengl](http://www.sgi.com/products/software/opengl).

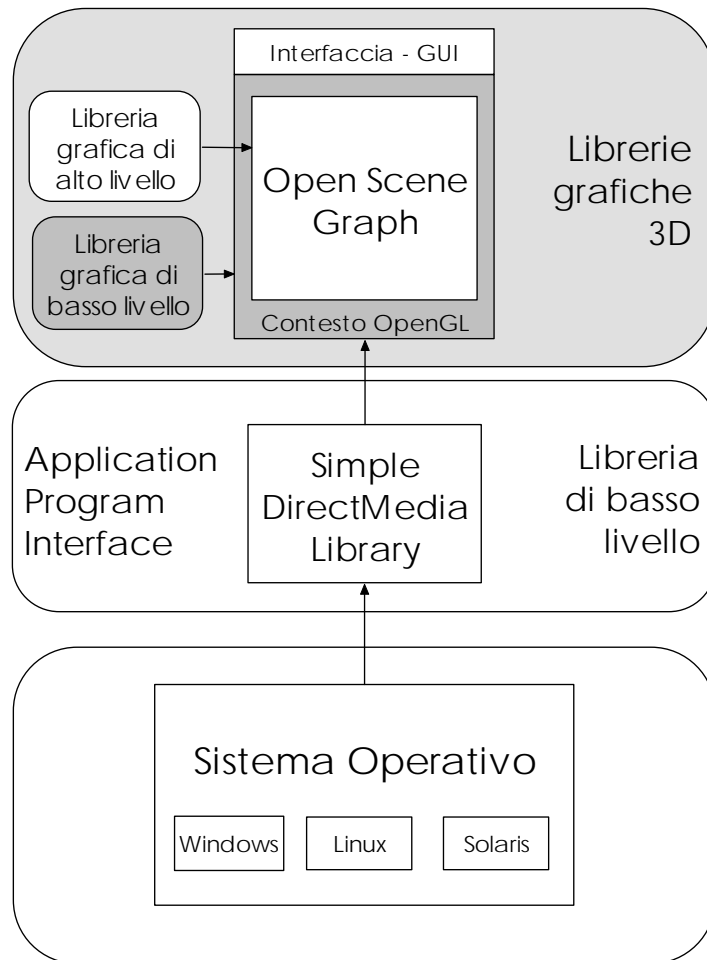


Figura 3.2: struttura del progetto di tesi – OpenGL

OpenGL fornisce un controllo diretto sulle operazioni fondamentali di grafica in due e tre dimensioni. Ciò include la specificazione di parametri quali matrici di trasformazione, o operazioni di aggiornamento dei pixel. Inoltre, OpenGL non impone un particolare metodo per la descrizione degli oggetti geometrici complessi, ma fornisce piuttosto i mezzi di base per mezzo dei quali gli oggetti, indipendentemente da come siano stati descritti, possono essere ottenuti.

Analizziamo ora, in modo schematico, il funzionamento di OpenGL. Per prima cosa, possiamo pensare al nostro pacchetto grafico come ad una scatola nera, ossia un sistema le cui proprietà sono descritte solo per mezzo di input e output, senza che nulla sia noto sul suo funzionamento interno. In particolare, gli input

sono costituiti dalle funzioni chiamate dal programma applicativo, dagli input provenienti da altri dispositivi quali mouse e tastiera, ed infine dai messaggi del sistema operativo. Gli output sono per la maggior parte di tipo grafico, sono infatti prevalentemente costituiti da primitive geometriche da visualizzare sullo schermo. Questo stesso schema caratterizza anche molte altre librerie grafiche.

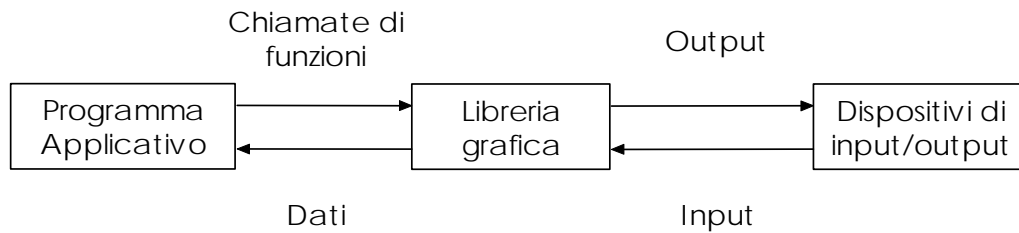


Figura 3.3: posizione della libreria grafica a livello concettuale

Le primitive geometriche sono disegnate in un *frame buffer* sulla base di alcuni attributi. Ogni primitiva geometrica, che può essere un punto, un segmento, un poligono, un pixel o una bitmap¹⁴, è definita da gruppi di uno o più *vertici*. Un vertice definisce un punto, l'estremo di un lato, il vertice di un poligono dove due lati si incontrano. Gli *attributi* possono essere cambiati indipendentemente, e il setting degli attributi relativi ad una primitiva non influenza il setting degli attributi di altre. I dati (*coordinate* e attributi) sono associati ai vertici, ed ogni vertice è processato indipendentemente.

La specificazione di attributi e primitive e la descrizione delle altre operazioni avviene inviando comandi sotto forma di chiamate di funzioni o procedure. Le funzioni contenute in una libreria grafica possono essere classificate a seconda della loro funzionalità:

1. Le **funzioni primitive** definiscono gli oggetti di basso livello che il sistema grafico può visualizzare. A seconda della libreria, le primitive possono includere punti, segmenti lineari, poligoni, pixel, testi e vari tipi di curve e superfici.

2. Le **funzioni attributo** governano il modo con cui le primitive appaiono sullo schermo. Gli attributi consentono infatti di specificare colore, pattern, tipo di caratteri, etc.
3. Le **funzioni di visualizzazione** consentono di descrivere la posizione e l'orientazione, e di fissare la visualizzazione delle immagini, in modo tale che le primitive appaiano entro una regione specifica dello schermo.
4. Le **funzioni di trasformazione** permettono all'utente di eseguire trasformazioni di oggetti quali traslazioni, rotazioni e trasformazioni di scala.
5. Le **funzioni di input** consentono all'utente di interagire con le diverse forme di input che caratterizzano i sistemi grafici moderni, quali gli input da tastiera, mouse, e data tablet.
6. Le **funzioni di controllo** permettono di comunicare con i sistemi window, per inizializzare i programmi, e per gestire eventuali errori che possono verificarsi nel corso dell'esecuzione del programma applicativo. Spesso infatti ci si deve preoccupare della complessità che nasce dal lavorare in ambienti in cui si è connessi ad una rete a cui sono collegati anche altri utenti.

I nomi delle funzioni di OpenGL iniziano con le lettere `gl` e sono memorizzate in una libreria, usualmente detta GL. L'utente può inoltre avvalersi di altre librerie collegate. Una è la libreria GLU¹⁵, che usa solo funzioni GL, e contiene il codice per oggetti comuni, quali ad esempio sfere, che l'utente preferisce non dover descrivere ripetutamente. Questa libreria è disponibile in tutte le implementazioni OpenGL. Una seconda libreria, GLUT¹⁶ si occupa della gestione delle interfacce con il sistema window. Fornisce la funzionalità minima

¹⁴ Bitmap è sinonimo di immagine digitale o raster graphics image.

¹⁵ GLU: OpenGL Utility.

¹⁶ GLUT: GL Utility Toolkit.

aspettata da qualsiasi sistema moderno di windowing. Per l'integrazione con i sistemi X Window è inoltre disponibile la libreria GLX¹⁷.

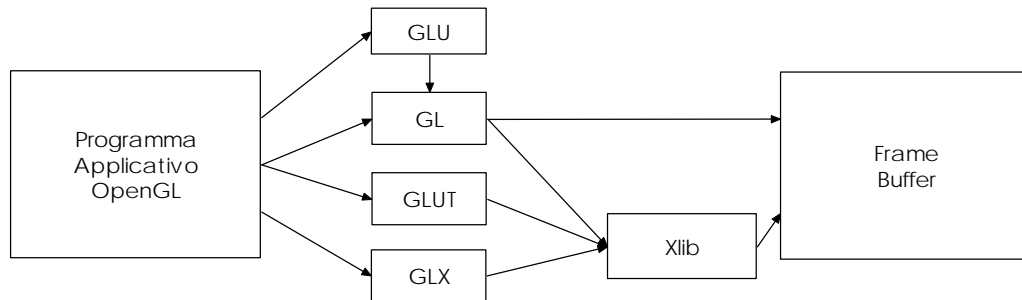


Figura 3.4: organizzazione delle librerie sul sistema operativo Linux.

Nella Figura 3.5 è illustrato un diagramma schematico del funzionamento di OpenGL. La maggior parte dei comandi inviati a OpenGL può venire accumulata in una display list, che viene processata successivamente. Altrimenti i comandi sono inviati attraverso un *processing pipeline*¹⁸. Durante il primo stadio vengono approssimate curve e superfici geometriche valutando funzioni polinomiali sui valori di input. Nel secondo stadio si opera sulle primitive geometriche descritte dai vertici: i vertici sono trasformati e illuminati, e le primitive sono tagliate al volume di osservazione, per essere inviate allo stadio successivo, la rasterizzazione. La rasterizzazione produce una serie di indirizzi e valori per il frame buffer, utilizzando una descrizione bidimensionale di punti, segmenti e poligoni. Ogni frammento così prodotto viene quindi inviato allo stadio successivo in cui si eseguono delle operazioni sui frammenti individuali, prima che essi finalmente modifichino il frame buffer. Queste operazioni includono, oltre alle operazioni logiche sui valori dei frammenti, l'aggiornamento del frame buffer in base ai dati in entrata memorizzati precedentemente, e la colorazione dei frammenti entranti con i colori

¹⁷ GLX: OpenGL Extension to the X Window System.

¹⁸ Nel gergo informatico, una *pipeline* è un insieme di elementi, atti a processare dati connessi in serie, cosicché l'output di un elemento è l'input del successivo.

memorizzati. Infine, i rettangoli di pixel e le bitmap possono bypassare lo stadio relativo all'elaborazione dei vertici, ed inviare, attraverso la rasterizzazione, blocchi di frammenti alle operazioni sui frammenti individuali.

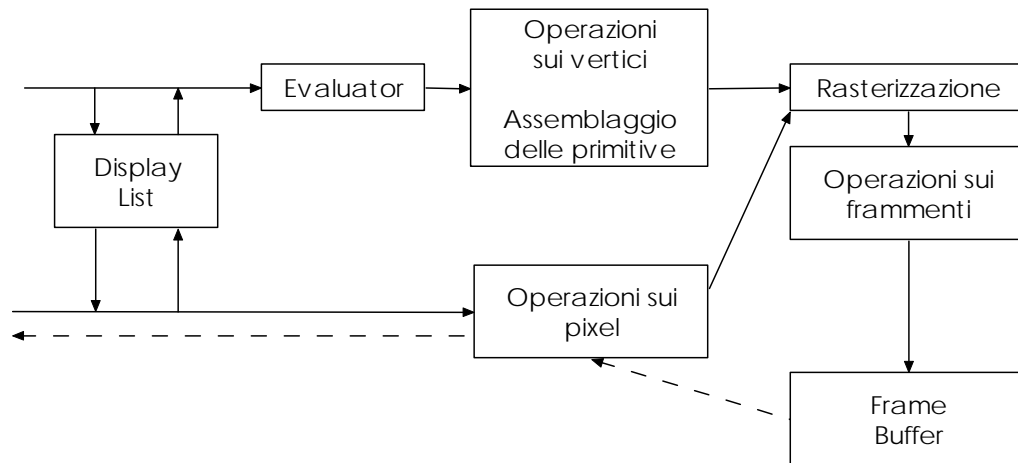


Figura 3.5 Diagramma a blocchi di OpenGL

3.3 Primitive e attributi

All'interno della comunità grafica c'è stato un ampio dibattito riguardante quali e quante primitive dovrebbero essere mantenute in una libreria grafica, non ancora del tutto risolto.

Da un lato c'è chi sostiene che le librerie dovrebbero contenere un insieme ristretto di primitive, quelle che tutti i sistemi hardware sono in grado di gestire. Inoltre queste primitive dovrebbero essere ortogonali, nel senso che ciascuna dovrebbe garantire all'utente delle possibilità non ottenibili dalle altre. Questi sistemi minimali di solito mantengono primitive quali linee, poligoni e qualche forma di testo (stringhe di caratteri), tutte generabili dall'hardware in modo efficiente.

Dall'altro lato ci sono le librerie che possono mantenere una grossa varietà di primitive, tra cui anche cerchi, curve, superfici e solidi. L'idea è quella di rendere disponibili primitive più complesse, al fine di rendere l'utente in grado di realizzare applicazioni più sofisticate. Tuttavia, poiché pochi sistemi hardware sono in grado di sostenere questa varietà di primitive, i programmi basati su questi pacchetti grafici risultano poco portabili.

OpenGL sta in una posizione intermedia tra le due tendenze. La libreria di base contiene un insieme limitato di primitive, mentre la libreria GLU contiene un insieme molto ricco di oggetti derivati dalle primitive di base.

3.3.1 Vertici e segmenti

Le primitive di base di OpenGL sono specificate attraverso una serie di vertici. Il programmatore definisce quindi un oggetto attraverso una sequenza di comandi della forma:

```
glBegin(type);  
glVertex*(...);  
..  
glVertex*(...);  
glEnd();
```

Il comando `glVertex*()` si usa per specificare un singolo vertice. OpenGL mette a disposizione dell'utente diverse forme per descrivere un vertice, in modo che l'utente possa selezionare quella più adatta al problema. Il carattere `*` può essere interpretato come due o tre caratteri della forma `nt` o `ntv`, dove `n` indica il numero di dimensioni (2, 3, o 4); `t` denota il tipo di dati (interi (`i`), virgola mobile (`f`), doppia precisione (`d`); infine la presenza del carattere `v` indica che le variabili sono specificate per mezzo di un puntatore ad un vettore o ad una lista. Ad esempio, se si vuole lavorare in due dimensioni, usando gli interi, allora il comando più appropriato è:

```
glVertex2i(GLint x, GLint y);
```

mentre il comando

```
glVertex3f(GLfloat x, GLfloat y, GLfloat z);
```

descrive un punto in tre dimensioni usando i numeri in virgola mobile. Se invece l'informazione è memorizzata in un vettore:

```
GLfloat vertex[3]
```

possiamo usare il comando

```
glVertex3fv(vertex);
```

I vertici possono definire un'ampia varietà di oggetti geometrici, e un diverso numero di vertici risulta necessario, a seconda dell'oggetto geometrico da rappresentare. Possiamo raggruppare quanti vertici vogliamo usando le funzioni `glBegin` e `glEnd`. L'argomento di `glBegin` specifica la figura geometrica che vogliamo che i nostri vertici definiscano. Ad esempio, per specificare un triangolo con vertici nei punti di coordinate (0, 0, 0), (0, 1, 0) e (1, 0, 1) si potrebbe scrivere:

```
glBegin(GL_POLYGON);  
glVertex3i(0, 0, 0);  
glVertex3i(0, 1, 0);  
glVertex3i(1, 0, 1);  
glEnd();
```

Tra la coppia di comandi `glBegin` e `glEnd` possono naturalmente trovarsi altre istruzioni o chiamate di funzioni. Ad esempio si possono modificare gli attributi oppure eseguire dei calcoli per determinare il vertice successivo.

I possibili oggetti geometrici, tutti definibili in termini di vertici o segmenti di linea, messi a disposizione da OpenGL sono riassunti nella tabella 3.1. Presi complessivamente, questi tipi di oggetto soddisfano le necessità di quasi tutte le applicazioni grafiche. Naturalmente anche un segmento di linea è specificato da una coppia di vertici, ma esso risulta talmente importante da essere considerato quale un'entità geometrica di base. I segmenti possono essere usati per approssimare curve, per connettere valori di un grafico, come lati di poligoni.

Primitiva	Interpretazione dei vertici
GL_POINTS	Ogni vertice descrive la locazione di un punto.
GL_LINES	Ogni coppia di vertici descrive un segmento di linea.
GL_LINE_STRIP	Serie di segmenti di linea connessi: ogni vertice dopo il primo è un estremo del segmento successivo.
GL_LINE_LOOP	E' come il line strip, ma un segmento viene aggiunto tra il vertice finale ed il vertice iniziale.
GL_POLYGON	Line loop formato da vertici che descrive il contorno di un poligono convesso.
GL_TRIANGLE	Ogni triade di vertici consecutivi descrive un triangolo.
GL_QUAD	Ogni gruppo consecutivo di quattro vertici descrive un quadrilatero.
GL_TRIANGLE_STRIP	Ogni vertice, ad eccezione dei primi due, descrive un triangolo formato da quel vertice e dai due precedenti.
GL_QUAD_STRIP	Ogni coppia di vertici, ad eccezione dei primi due, descrive un quadrilatero formato da quella coppia e dalla coppia precedente.
GL_TRIANGLE_FAN	Ogni vertice, ad eccezione dei primi due, descrive un triangolo formato da quel vertice e dal vertice precedente e dal primo vertice.

Tabella 3.1: primitive geometriche di OpenGL

Come risulta da questa tabella, ci sono più scelte per la visualizzazione delle primitive geometriche. Ad esempio per i segmenti di linea ci sono il tipo `GL_LINES`, che interpreta ogni coppia di vertici come estremi di un segmento; il tipo `GL_LINE_STRIP` che consente di connettere i segmenti successivi; ed infine il tipo `GL_LINE_LOOP` che aggiunge un segmento di linea tra il primo e l'ultimo vertice.

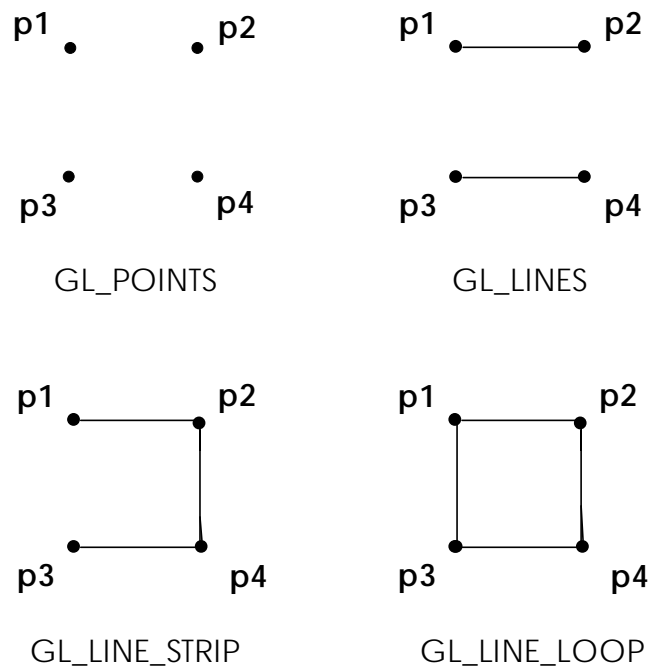


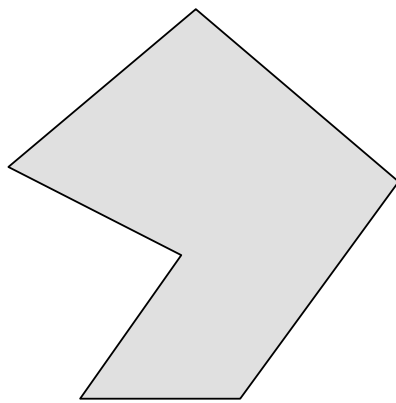
Figura 3.6: tipi di segmenti di linea

3.3.2 Poligoni

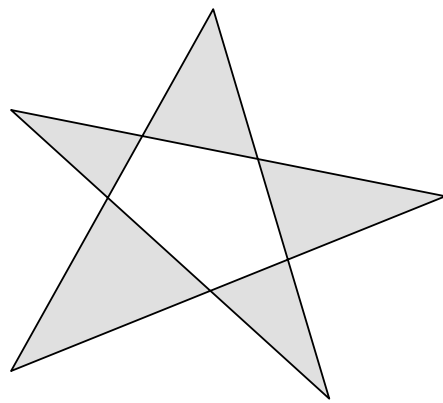
Una delle differenze concettuali più importanti tra i tipi di oggetto è se essi abbiano o meno una regione interna. Usualmente il termine poligono è riservato agli oggetti chiusi, come quelli ottenibili con il tipo `LINE_LOOP`, ma dotati di regione interna. E' possibile visualizzare un poligono in diversi modi. Ad esempio si possono visualizzare solo i suoi lati. Oppure possiamo riempire la regione interna con un colore, o con un pattern, e i lati possono essere visualizzati oppure no. Nonostante i lati di un poligono siano facilmente definibili tramite una lista di vertici, se la regione interna non è ben definita, il rendering del poligono può risultare scorretto. Di conseguenza è necessario

stabilire come poter definire la regione interna di un poligono. In due dimensioni, fino a quando nessuna coppia di lati si interseca, abbiamo un poligono semplice, con una regione interna chiaramente definita. Dunque le locazioni dei vertici determinano se il poligono è semplice oppure no.

E' possibile definire una regione interna anche per i poligoni non semplici, in modo che l'algoritmo di rendering possa riempire qualsiasi tipo di poligono. Gli algoritmi di riempimento dei poligoni sono basati sull'elaborazione di punti all'interno del poligono, a ciascuno dei quali viene assegnato un particolare colore.



Poligono semplice



Poligono non semplice

Figura 3.7: poligoni semplici e non semplici.

Ci sono diversi test che possono essere utilizzati per determinare se un punto sia da considerare interno o esterno al poligono. Uno dei più applicati è il cosiddetto odd-even (pari-dispari) test. L'idea di base è fondata sulla considerazione che se un punto è nella regione interna del poligono, allora un raggio che si emana da esso in direzione dell'infinito, deve attraversare un numero dispari di lati del poligono, mentre per i punti all'esterno, i raggi emanati devono attraversare un numero pari di lati per raggiungere l'infinito. Questo test è facile da implementare e si integra bene con i processi standard di rendering.

Sebbene il test pari-dispari sia semplice da implementare e si integri bene nel processo di rendering standard, a volte si desidera un algoritmo che riempia interamente un poligono a stella come quello della figura 3.7. L'algoritmo di winding considera di percorrere i lati del poligono da un vertice qualunque in una direzione qualunque fino a raggiungere il punto di partenza. Per ciascun punto si considera una linea arbitraria infinita e si calcola il numero di winding per quel punto come il numero di lati che tagliano la linea in direzione verso il basso meno il numero di lati che tagliano la linea verso l'alto. Se il numero di winding non è zero, il punto è interno al poligono. Il linguaggio PostScript utilizza questo algoritmo di riempimento.

Una ragione per preferire il test pari-dispari è che l'effetto di riempimento non cambia a seconda di quali vertici sono usati per indicare il poligono, ad esempio:

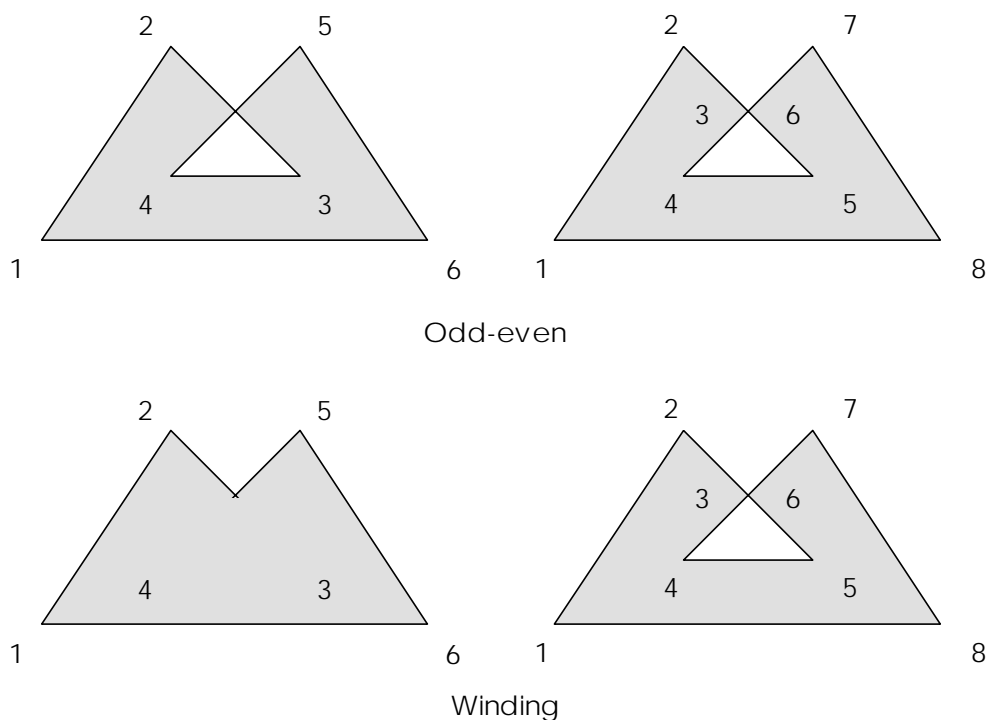


Figura 3.8: riempimento di poligoni non semplici

Un oggetto si definisce convesso se tutti i punti su un segmento di linea che congiunge due punti al suo interno, o sul suo bordo, si trovano all'interno

dell'oggetto. Anche per la convessità ci sono appositi test. Nel caso bidimensionale (la definizione di convessità si estende a qualunque dimensione) se un punto è all'interno del poligono, e ne tracciamo il contorno in senso orario, il punto deve rimanere sulla destra di ciascun lato del contorno.

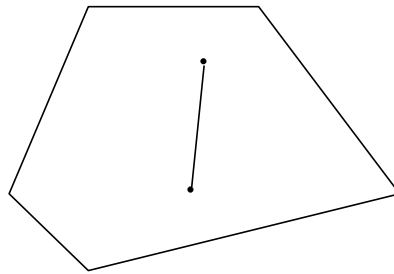


Figura 3.9: poligono convesso.

Molte librerie grafiche garantiscono un corretto riempimento delle figure geometriche solo quando queste sono convesse. In particolare OpenGL richiede che i poligoni siano semplici, convessi e senza buchi: nel caso queste condizioni non siano rispettate, non è assicurata una visualizzazione corretta. Figure complesse che richiedono poligoni con queste proprietà si ottengono formando l'unione di poligoni semplici convessi, come fanno alcune delle funzioni della libreria GLUT.

In tre dimensioni si presenta qualche difficoltà in più poiché le figure possono non essere piane. Se un poligono non è planare, per effetto di trasformazioni, ad esempio una proiezione, si può ottenere un poligono non semplice. Molti sistemi grafici sfruttano la proprietà che tre vertici non allineati determinano sia un triangolo che il piano in cui giace il triangolo. Quindi, usando sempre i triangoli è possibile garantire un rendering corretto degli oggetti.

Tornando ai diversi tipi di OpenGL illustrati in tabella 3.1, per le figure con regione interna abbiamo queste possibilità. Il tipo `GL_POLYGON` produce la stessa figura che si ottiene usando `LINE_LOOP`: i vertici successivi definiscono i segmenti di linea, e un segmento connette il primo e l'ultimo vertice. Il riempimento della regione interna del poligono è determinato dagli attributi, che

consentono inoltre di specificare se visualizzare o meno i lati. I tipi `GL_TRIANGLES` e `GL_QUADS` sono casi speciali di poligoni: gruppi successivi di tre e quattro vertici sono interpretati come triangoli e quadrilateri, rispettivamente. L'uso di questi tipi consente un rendering più efficiente di quello ottenibile con `GL_POLYGON`.

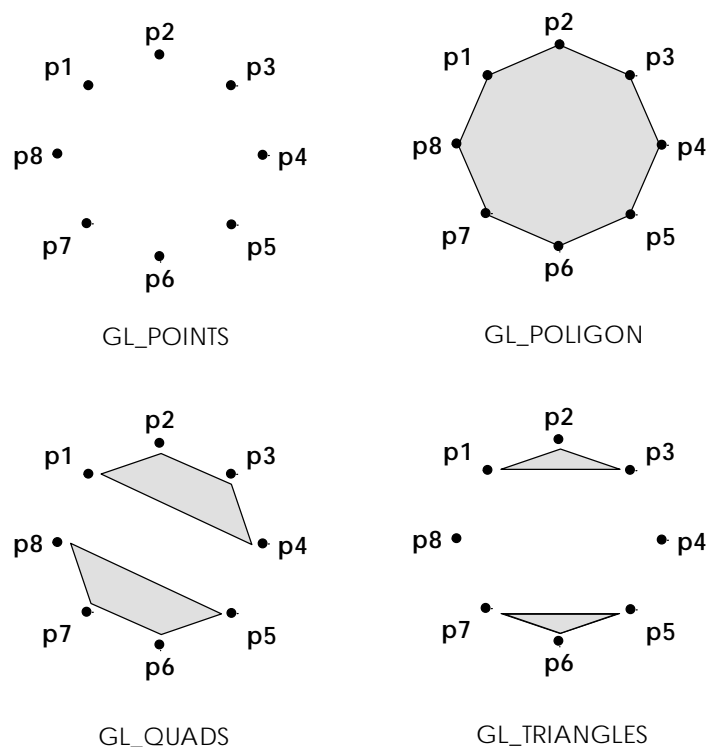


Figura 3.10: tipi di poligoni

I tipi `GL_TRIANGLE_STRIP`, `GL_QUAD_STRIP`, `GL_TRIANGLE_FAN` sono invece basati su gruppi di triangoli e quadrilateri che condividono vertici ed archi. Col tipo `TRIANGLE_STRIP`, ad esempio, ogni vertice aggiuntivo aggiunge un nuovo triangolo (figura 3.11).



Figura 3.11: tipi Triangle_strip (a sinistra) e Quad_strip

3.3.3 Caratteri di testo

In diverse applicazioni gli output grafici sono accompagnati da annotazioni testuali. Nonostante gli output testuali siano la norma nei programmi non grafici, i testi in computer graphics risultano più problematici da gestire. Infatti, mentre nelle applicazioni non grafiche ci si accontenta di un solo insieme di caratteri, visualizzati sempre nella stesso modo, nella computer graphics si desidera controllare stile, font, dimensione, colore ed altri parametri.

Ci sono due forme di testo: stroke e raster. I testi stroke sono costruiti esattamente come le altre primitive grafiche. Si usano vertici, segmenti di linee e curve per descrivere il contorno dei caratteri. Il vantaggio principale dei testi stroke è una conseguenza diretta della loro definizione: dato che i caratteri sono descritti nello stesso modo degli altri oggetti geometrici, è possibile raggiungere un notevole livello di dettaglio nella loro rappresentazione. E' inoltre possibile manipolarli attraverso le trasformazioni standard, e visualizzarli come qualsiasi altra primitiva grafica. Un altro vantaggio dei testi stroke è che rendendo un carattere più grosso o più piccolo, esso mantiene dettagli e aspetto di qualità elevata; di conseguenza è necessario definire i caratteri una sola volta e usare le trasformazioni per generarli con la dimensione e l'orientazione desiderate. La definizione di una famiglia di 128 o 256 caratteri di un determinato stile (font) può tuttavia risultare complessa e richiedere una porzione di memoria ed un tempo di elaborazione significativi.

I testi di tipo raster sono più semplici e veloci. I caratteri sono definiti come rettangoli di bit, chiamati *bitblock*, e ogni blocco definisce un singolo carattere attraverso un opportuno pattern di 0 e di 1. Un carattere raster può essere posto nel frame buffer attraverso un'operazione di trasferimento che muove ogni bitblock con una singola istruzione. Alcune trasformazioni dei caratteri raster, ad esempio le rotazioni, possono tuttavia perdere significato in quanto i bit che definiscono il carattere possono finire in locazioni che non corrispondono alle locazioni dei pixel nel frame buffer.

3.3.4 Oggetti curvilinei

Tutte le primitive del nostro set di base sono definite attraverso i vertici. Ad eccezione del tipo `point`, tutti gli altri tipi consistono di segmenti di linea, o usano i segmenti di linea per definire contorni di altre figure geometriche. E' comunque possibile arricchire ulteriormente il set di oggetti.

Per prima cosa, si possono usare le primitive disponibili per approssimare curve e superfici. Ad esempio un cerchio può essere approssimato attraverso un poligono regolare di n lati; nello stesso modo, una sfera può essere approssimata con un poliedro.

Un approccio alternativo è quello di partire dalle definizioni matematiche degli oggetti curvilinei, e quindi di costruire funzioni grafiche per implementarle. Oggetti quali superfici quadriche e curve polinomiali parametriche hanno una chiara definizione matematica, e possono essere specificati attraverso opportuni insiemi di vertici. Ad esempio, una sfera può essere definita dal suo centro e da un punto sulla sua superficie.

Quasi tutti i sistemi grafici consentono di seguire entrambi gli approcci. In OpenGL possiamo usare la libreria GLU che mette a disposizione una collezione di approssimazioni di curve e superfici comuni, e possiamo anche scrivere nuove funzioni per definire altre figure.

3.3.5 Attributi

In un sistema grafico moderno vi è una distinzione tra il tipo di primitiva e la sua visualizzazione. Una linea tratteggiata, ad esempio, ed una linea continua sono dello stesso tipo geometrico, ma sono visualizzate diversamente. Si definisce attributo qualsiasi proprietà che determina come una primitiva geometrica deve essere visualizzata. Il colore è un attributo ovvio, così come lo spessore delle linee ed il pattern usato per riempire i poligoni.

Ad ogni attributo è associato un valore corrente, il quale può essere modificato con apposite funzioni di modifica. Il valore corrente di un attributo si applica a tutte le operazioni che seguono, fino alla successiva modifica. Inizialmente gli attributi hanno ciascuno un proprio valore di default.



Figura 3.12: alcuni attributi per linee e poligoni.

Gli attributi possono essere associati alle primitive in diversi punti del processing pipeline. Nel modo immediato, le primitive non sono memorizzate nel sistema, ma vengono visualizzate non appena sono state definite. I valori attuali degli attributi sono parte dello stato del sistema grafico. Nel sistema non rimane dunque memoria della primitiva; solo l'immagine appare sul display, e una volta cancellata dal display, la primitiva è persa. Le display list consentono invece di mantenere oggetti nella memoria, in modo che essi possano essere visualizzati nuovamente.

Ad ogni tipo geometrico è associato un certo insieme di attributi. Un punto prevede attributi per il colore e la dimensione. Ad esempio, la dimensione può essere posta uguale a due pixel usando il comando

```
glPointSize(2.0);
```

Un segmento lineare può avere uno spessore che si imposta con:

```
void glLineWidth(Glfloat size)
```

ed un tipo di tratto (continuo, tratteggiato, punteggiato), che si imposta con:

```
void glLineStipple(Glint factor, Glushort pattern)
```

dove `pattern` è costituito da 16 bit che rappresentano il tratto, e `factor` il fattore di scala.

Le primitive dei poligoni hanno più attributi, poiché si deve specificare il riempimento delle regioni interne: possiamo usare una tinta unita, o un pattern, possiamo decidere se riempire o meno il poligono, se visualizzare o meno il suo contorno. Il tipo di rendering dei poligoni si determina con la primitiva:

```
void glPolygonMode(Glenum face, Glenum mode)
```

dove `face` può essere `GL_FRONT` (solo la faccia di fronte), `GL_BACK` (solo la faccia posteriore), `GL_FRONT_AND_BACK` (entrambe le facce), mentre `mode` può essere `GL_POINT` (solo i vertici), `GL_LINE` (solo i lati) o `GL_FILL` (riempimento della parte interna).

Nei sistemi in cui si utilizzano i testi stroke come primitive, esistono attributi anche per la direzione della stringa di testo, l'altezza e la larghezza dei caratteri, il font e lo stile (italico, neretto, sottolineato).

Si osservi infine che gli attributi relativi alla dimensione dei punti e alla larghezza delle linee sono specificati in termini di dimensione dei pixel. Pertanto, se due display hanno pixel di dimensione differente, l'immagine visualizzata può apparire leggermente diversa. Alcune librerie grafiche, nel tentativo di assicurare che immagini identiche siano prodotte su tutti i sistemi, specificano gli attributi in modo indipendente dal dispositivo di output. Sfortunatamente, assicurare che due sistemi producano lo stesso output grafico costituisce un problema di difficile risoluzione.

3.3.6 Colore

Ci sono diverse tecniche per la visualizzazione del colore. Nel sistema RGB, ogni pixel ha componenti separate per i tre colori, un byte per ciascuno. Dato che la libreria grafica deve essere il più possibile indipendente dal sistema hardware, è importante avere la possibilità di specificare il colore

indipendentemente del numero di bit nel frame buffer, e lasciare all'hardware del sistema il compito di approssimare il colore richiesto nel miglior modo possibile, compatibilmente con il display disponibile.

Una tecnica molto diffusa è quella basata sul cosiddetto color cube: le componenti di colore vengono specificate tramite i numeri compresi tra 0.0 e 1.0, dove 1.0 denota il valore massimo del corrispondente colore primario, e 0.0 il valore nullo. In OpenGL il color cube si implementa nel modo seguente. Per disegnare usando, ad esempio, il colore rosso si chiama la funzione

```
glColor3f(1.0, 0.0, 0.0);
```

Poiché il colore fa parte dello stato, si continuerà a disegnare in rosso fino a quando il colore viene cambiato. La stringa `3f` è usata per indicare che il colore è specificato in accordo al modello RGB¹⁹ a tre colori, e che i valori delle componenti di colore sono in virgola mobile. Se si fa riferimento al sistema RGBA, si utilizza il valore alpha come indice per l'opacità o la trasparenza (un oggetto opaco è un oggetto attraverso cui non passa la luce, mentre un oggetto è trasparente se lascia passare la luce).

Uno dei primi compiti che devono essere eseguiti in un programma è quello di ripulire la drawing window²⁰ destinata alla visualizzazione dell'output; e questa operazione deve essere ripetuta ogni volta che si deve visualizzare una nuova immagine. Usando il sistema di colorazione a quattro dimensioni (RGBA) si possono creare effetti in cui le drawing window possono intersecarsi con altre finestre, manipolando il valore dell'opacità. La chiamata della funzione

```
glClearColor(1.0, 1.0, 1.0, 0.0);
```

definisce un *clearing* bianco, poiché le prime tre componenti sono uguali a 1.0, ed opaco, poiché la componente alpha è pari a 0.0. Quando invece si utilizza il

¹⁹ RGB: Red, Blue e Green.

²⁰ L'area dello schermo.

sistema di colorazione basato sulle tabelle look-up, i colori vengono selezionati attraverso la funzione

```
glIndexi(element);
```

che seleziona un particolare colore dalla tabella.

Per quanto riguarda il setting degli attributi di colore, quando si usa il sistema RGB, ci sono tre attributi da definire. Il primo è il colore di *clear*, che è definito dal comando

```
glClearColor(1.0, 1.0, 1.0, 0.0);
```

Il colore di rendering per i punti che visualizzeremo può essere selezionato attraverso la chiamata della funzione

```
glColor3f(1.0, 0.0, 0.0);
```

che, in questo esempio, è relativa al colore rosso.

3.3.7 Visualizzazione

Siamo ora in grado di disporre un'ampia varietà di informazioni grafiche sul nostro schermo bidimensionale, e siamo anche in grado di descrivere come vorremmo che questi oggetti vengano visualizzati. Tuttavia ancora non abbiamo un metodo per specificare esattamente quali di questi oggetti devono apparire sullo schermo.

La visualizzazione in due dimensioni consiste nel prendere un'area rettangolare sul nostro mondo bidimensionale e nel trasferire il suo contenuto sul display. L'area del mondo che visualizziamo è chiamata rettangolo di visualizzazione o rettangolo di clipping. Gli oggetti all'interno del rettangolo saranno nell'immagine visualizzata; gli oggetti al di fuori saranno tagliati fuori; gli oggetti a cavallo dei lati del rettangolo risulteranno parzialmente visibili (figura 3.13). La dimensione della finestra e la posizione in cui la finestra deve apparire

sullo schermo sono due decisioni indipendenti, che vengono gestite attraverso le funzioni di controllo.

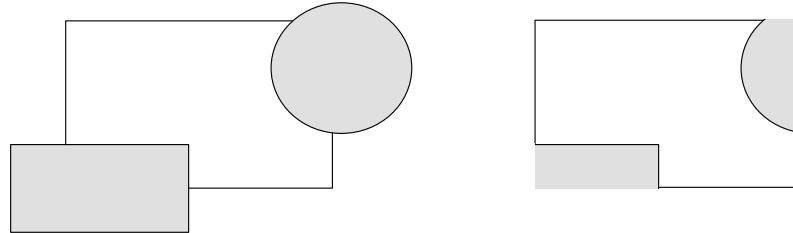


Figura 3.13: visualizzazione bidimensionale

Poiché la grafica bidimensionale è un caso particolare della grafica tridimensionale, possiamo considerare il nostro rettangolo di visualizzazione disposto sul piano $z=0$ all'interno di un volume tridimensionale di visualizzazione. Se non specifichiamo un volume di visualizzazione, OpenGL usa il suo cubo di default, con origine nel centro. In termini del nostro piano bidimensionale, il vertice in basso a sinistra giace nel punto di coordinate $(1.0, 1.0)$, e il vertice in alto a destra nel punto $(1.0, 1.0)$.

La visualizzazione bidimensionale descritta rappresenta un caso speciale di proiezione ortografica. Una proiezione ortografica consiste nel proiettare il punto (x, y, z) sul punto $(x, y, 0)$. Dato che il nostro mondo bidimensionale consiste del solo piano $z=0$, la proiezione non ha alcun effetto; tuttavia le proiezioni consentono di impiegare le tecniche dei sistemi grafici tridimensionali per produrre le immagini. Nella libreria OpenGL, una proiezione ortografica con un volume di visualizzazione formato da un parallelepipedo retto è specificata tramite il comando

```
void glOrtho (GLdouble left, GLdouble right, GLdouble  
bottom, GLdouble top, GLdouble near, GLdouble far)
```

Fino a quando il piano $z=0$ è posizionato tra *near* e *far*, il piano bidimensionale interseca il volume di visualizzazione. Se invece il fatto di fare

riferimento ad un volume tridimensionale in un'applicazione bidimensionale può sembrare strano, si può utilizzare la funzione

```
void glOrtho2D(GLdouble left, GLdouble right, GLdouble  
bottom, GLdouble top)
```

della libreria GLU, che consente di rendere il programma più leggibile. Questa funzione è equivalente a `glOrtho`, dove a `near` e `far` sono attribuiti i valori 1.0 e 1.0, rispettivamente.

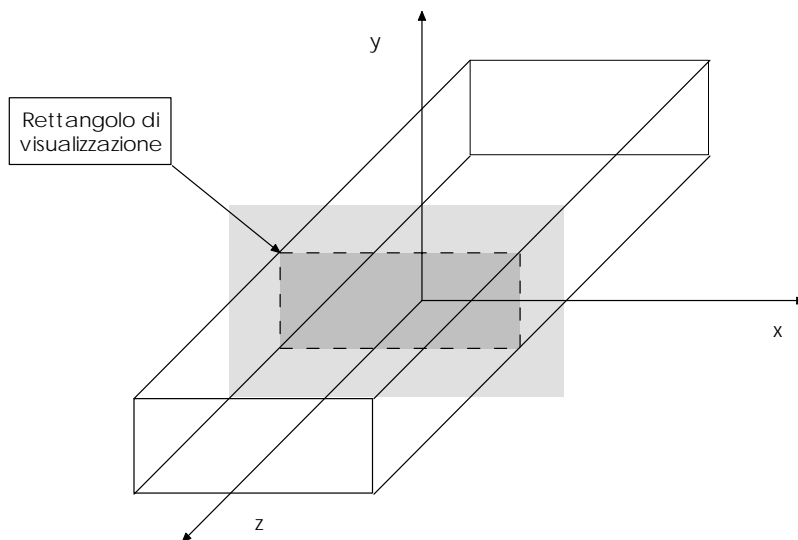


Figura 3.14: volume di visualizzazione

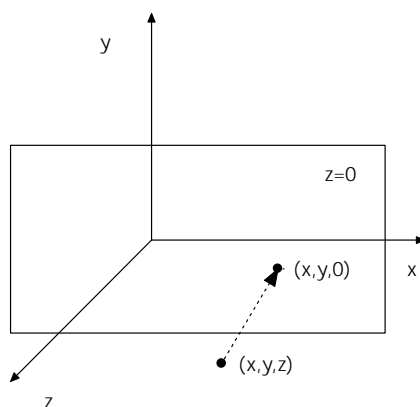


Figura 3.15: proiezione ortografica

3.3.8 Matrix mode

Nei sistemi grafici l'immagine desiderata di una primitiva si ottiene moltiplicando, o concatenando, un certo numero di matrici di trasformazione. Come la maggior parte delle variabili di OpenGL, i valori di queste matrici fanno parte dello stato del sistema, e rimangono validi fino a quando non sono cambiati. Le due matrici più importanti sono la matrice model-view²¹ e la matrice proiezione. OpenGL contiene delle funzioni che hanno proprio il compito di manipolare queste matrici. Tipicamente si parte da una matrice identica che viene successivamente modificata applicando una sequenza di trasformazioni. La funzione `matrix mode` si usa per indicare la matrice a cui le operazioni devono essere applicate. Per default, le operazioni vengono applicate alla matrice model-view. La seguente sequenza di istruzioni è comunemente utilizzata per predisporre un rettangolo di visualizzazione bidimensionale:

```
glMatrixMode(GL_PROJECTION);glLoadIdentity();  
glOrtho2D(0.0, 500.0, 0.0, 500.0);  
glMatrixMode(GL_MODELVIEW);
```

Questa sequenza definisce un rettangolo di visualizzazione 500 500, con il vertice in basso a sinistra posto nell'origine del sistema bidimensionale. Nei programmi complessi, è sempre utile restituire il `matrix mode`, nell'esempio model-view, per evitare i problemi che si potrebbero verificare perdendo traccia del `matrix mode` in cui il programma si trova ad un certo istante.

3.4 Funzioni di controllo

Le funzioni di controllo gestiscono le interazioni del sistema grafico con il sistema window ed il sistema operativo. La libreria GLUT fornisce una semplice interfaccia tra il sistema grafico e i sistemi window e operativo.

²¹ Usata per trasformare oggetti 3D e per cambiare le coordinate da uno spazio standard locale a uno spazio associato alla camera.

3.4.1 Interazioni con il sistema window

Verrà utilizzato il termine finestra (window) per indicare un'area rettangolare del display che, per default, sarà lo schermo di un CRT. Dato che nei display di tipo raster si visualizza il contenuto del frame buffer, ampiezza e l'altezza della finestra sono misurate in pixel. Si noti che in OpenGL le coordinate della finestra sono tridimensionali, mentre quelle dello schermo sono bidimensionali.

Verrà utilizzato invece il termine sistema window per fare riferimento agli ambienti multiwindow, forniti da sistemi come X Window e Microsoft Windows, che consentono di aprire più finestre contemporaneamente sullo schermo di un CRT. Ogni finestra può avere finalità differenti, dall'editing di un file al monitoraggio del sistema.

Per il sistema window, la finestra grafica è un particolare tipo di finestra (una in cui la grafica può essere visualizzata) gestito dal sistema. Le posizioni nella finestra grafica sono misurate rispetto ad uno dei suoi angoli. Di solito si sceglie l'angolo in basso a sinistra come origine. Tuttavia, tutti i sistemi grafici visualizzano lo schermo così come i sistemi televisivi, dall'alto al basso e da sinistra verso destra. In questo caso l'angolo in alto a sinistra dovrebbe essere l'origine. Nei comandi OpenGL si assume che l'origine sia in basso a sinistra, ma l'informazione restituita al sistema di windowing, ad esempio la posizione del mouse, ha di solito come origine l'angolo in alto a sinistra.

Prima di aprire una finestra, ci deve essere interazione tra il sistema window e OpenGL. Nella libreria GLUT, questa interazione è avviata dalla chiamata della funzione

```
glutInit(int *argcp, char **argv);
```

i cui argomenti sono simili a quelli della funzione main del linguaggio C. Possiamo quindi aprire una finestra chiamando la funzione GLUT

```
glutCreateWindow(char *name)
```

dove il titolo della finestra è dato dalla stringa `name`. La finestra creata avrà di default una dimensione, una posizione sullo schermo e alcune caratteristiche quali l'uso del sistema di colorazione RGB. Possiamo comunque usare le funzioni della libreria GLUT per specificare questi parametri prima di creare la finestra. Ad esempio il codice

```
glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);  
glutInitWindowSize(640, 480);  
glutInitWindowPosition(0, 0);
```

specifica una finestra di dimensione 640 480 nell'angolo in alto a sinistra del display, in cui si usa il sistema di colorazione RGB anziché i colori indicizzati (`GLUT_INDEX`); un *depth buffer* per la rimozione delle superfici nascoste, e un *double buffering* (`GLUT_DOUBLE`) piuttosto che *single* (`GLUT_SINGLE`). Non è necessario specificare queste opzioni in modo esplicito, ma la loro indicazione rende il codice più chiaro. Si osservi infine che nella chiamata di `glutInitDisplayMode`, si effettua un OR logico dei parametri.

3.4.2 Aspect – Ratio e ViewPort

L'aspect-ratio di un rettangolo è definito dal rapporto tra la sua altezza e la sua larghezza. Se l'aspect-ratio del rettangolo di visualizzazione, specificato nel comando `glOrtho`, non è uguale a quello della finestra di output grafico specificata nel comando `glutInitWindowSize`, si possono verificare effetti indesiderati: gli oggetti possono infatti risultare distorti sullo schermo. Infatti, per trasferire tutto il contenuto del rettangolo di visualizzazione sulla finestra di display si è costretti a distorcerne il contenuto. La distorsione può invece essere evitata facendo in modo che rettangolo di visualizzazione e finestra di display abbiano lo stesso aspect-ratio.

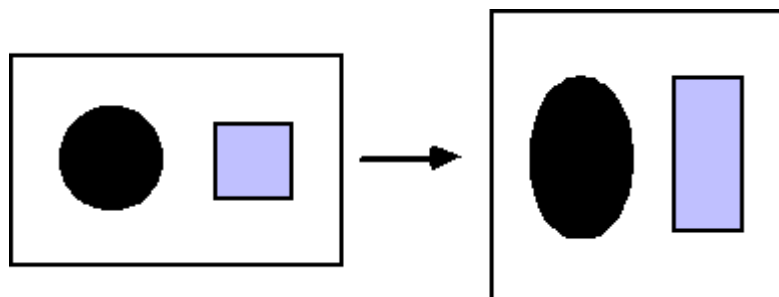


Figura 3.16: distorsione delle immagini dovuta a aspect-ratio differenti

Un altro metodo più flessibile è quello di usare il viewport (spioncino). Un viewport è un'area rettangolare della finestra di display. Per default è l'intera finestra, ma si può scegliere una qualsiasi dimensione utilizzando il comando

```
void glViewport(GLint x, GLint y, GLsizei w, GLsizei h)
```

dove x e y è l'angolo in basso a sinistra del viewport (misurato rispetto all'angolo in basso a sinistra della finestra), e w e h definiscono larghezza e altezza del viewport. I dati, rappresentati da numeri interi, permettono di specificare posizioni e distanze in pixel. Le primitive sono visualizzate nel viewport, e l'altezza e la larghezza del viewport possono essere definite in modo da ottenere lo stesso aspect-ratio del rettangolo di visualizzazione, per evitare il problema della distorsione delle immagini. E' inoltre possibile utilizzare più viewport per disporre immagini differenti in parti differenti dello schermo.

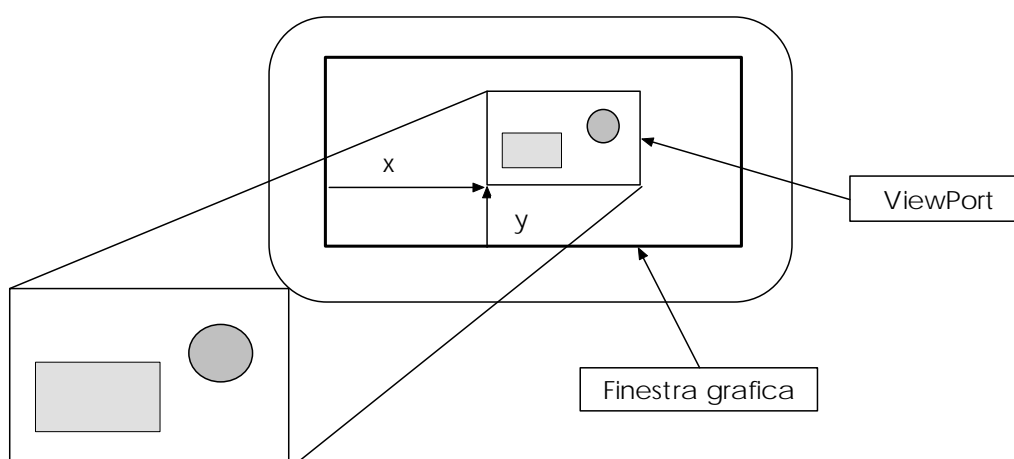


Figura 3.17: mapping del rettangolo di visualizzazione sul viewport

3.4.3 L'event - loop

Potremmo pensare che il nostro programma, una volta effettuata l'inizializzazione di GLUT e degli attributi, potrebbe passare a disegnare direttamente l'immagine che ci interessa. Quando si lavora in modalità grafica immediata, le primitive infatti vengono rese sullo schermo non appena invocate. Tuttavia, se usiamo un sistema a finestre, questo non basta, perchè la finestra può venire nascosta da altre finestre e quando verrà riesposta l'immagine dovrà essere ridisegnata. Quindi occorre che il disegno del contenuto della finestra venga svolto ogni volta che accade un evento nel sistema a finestre che lo rende necessario. Questo si ottiene passando il controllo ad un event-loop che esamina gli eventi del sistema a finestre e li gestisce, realizzato dalla procedura:

```
void glutMainLoop(void)
```

Per informare l'event-loop di come deve essere ridisegnato il contenuto di una finestra OpenGL, dobbiamo passargli la funzione che dovrà utilizzare, tramite la procedura GLUT:

```
void glutDisplayFunc(void (*func)(void))
```

Naturalmente la chiamata a questa procedura va fatta prima di eseguire

```
glutMainLoop.
```

Poichè un'applicazione non interattiva non deve far altro che passare il controllo all'event-loop, molto spesso può essere realizzata secondo lo schema di programma che segue. All'inizio sono inseriti i comandi per l'inclusione degli header file per OpenGL e GLUT.

Per predisporre le variabili di stato relative alla visualizzazione e agli attributi (parametri che si vogliono scegliere in modo indipendente della funzioni di visualizzazione), si definisce una procedura a parte: `myinit()` (esempio nel codice 3.1 della pagina seguente).

```

#include <GL/gl.h>
#include <GL/glut.h>

void main(int argc, char** argv)
{
    void myinit(), display();
    glutInit(&argc,argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("simple OpenGL example");
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
}

```

Codice 3.1: inizializzazione con myinit().

Capitolo 4

Scene Graph

Nel precedente capitolo è stata introdotta la metodologia per arrivare a costruire oggetti tridimensionali a partire dalla singola forma geometrica in 2D; ma esempi lampanti, come la maggior parte dei videogames in commercio, mostrano quanto sia ancor più avanzato il livello di costruzione grafica applicata alla realizzazione di mondi virtuali.

L'uso delle primitive descritte precedentemente non può risolvere tale problema: vi è la necessità di utilizzare strutture dati più complesse che evitino di richiamare le funzioni a basso livello.

Una delle strutture dati più utilizzate a tale scopo dalle librerie grafiche 3D è lo Scene Graph (rappresentazione della scena tramite grafo di scena).

4.1 Definizione

Un requisito molto importante nella grafica 3D in tempo reale è la capacità di processare molto velocemente scene complesse, in modo da alleggerire il più possibile il carico del chipset grafico, che solitamente rappresenta il collo di bottiglia delle applicazioni di realtà virtuale. Ciò si traduce in un'efficiente strategia di controllo della visibilità e di gestione delle collisioni.

E' importante, dunque, che la scena sia organizzata in maniera da poter effettuare i calcoli relativi a queste operazioni nella maniera più veloce possibile, e questo può avvenire solo fornendo alla scena una struttura che ne permetta una gestione efficace.

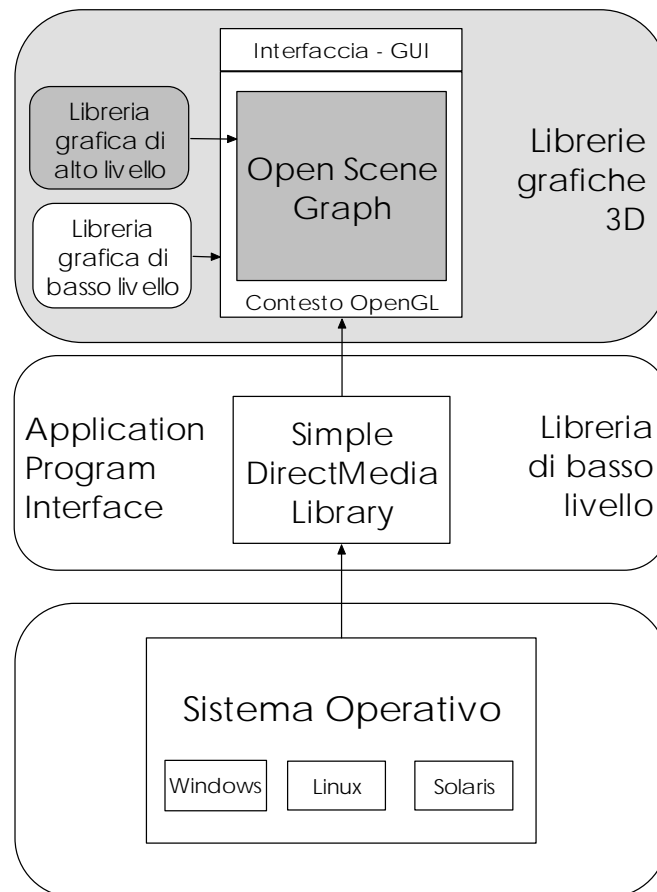


Figura 4.1: struttura del progetto di tesi – Open Scene Graph

Tale struttura è chiamata Scene Graph (SG), e presenta solitamente una configurazione gerarchica che ne massimizza l'efficienza. E' possibile che la scena sia partizionata secondo una struttura non gerarchica (si immagini di suddividerla mediante un semplice reticolo tridimensionale equispaziato, una sorta di griglia 3D i cui elementi sono detti *voxel*) ma solo in rari e sporadici casi in cui questa è la scelta migliore.

La struttura di uno Scene Graph può essere quella di un albero o quella più generica di un DAG²². Nel primo caso ogni componente dello SG (nodo) può essere diretto discendente di un solo altro nodo, nel secondo caso è possibile che un nodo possa discendere direttamente da più di un nodo. In entrambi i casi, comunque, le relazioni fra nodi non permettono la creazione di cicli per cui, comunque ci si muova fra i rami, non si può tornare indietro al nodo di partenza.

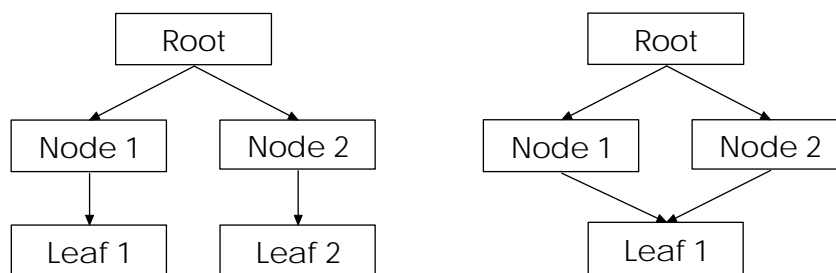


Figura 4.2: a sinistra l'esempio di un albero; a destra di un DAG.

Ogni nodo di uno SG incapsula informazioni relative a forme geometriche (meshes o semplici collezioni di poligoni), telecamere, suoni etc. Genericamente faremo riferimento a tali entità come ad oggetti.

Il motivo per il quale una struttura gerarchica possa migliorare l'efficienza dell'organizzazione della scena è evidente. Si immagini di voler processare la scena per verificare la visibilità dei suoi elementi; se la gerarchia è stata impostata su criteri di occupazione spaziale (ogni nodo "padre" occupa uno spazio maggiore o uguale all'insieme dello spazio occupato dai nodi "figli"), la

²² DAG: Direct Acyclic Graph (Grafo Diretto Aciclico).

non visibilità di un nodo implica la non visibilità di tutto il sottoramo che parte da quel nodo. In questo modo è possibile evitare di processare tutti i nodi di quel sottoramo, risparmiando preziose risorse computazionali.

Nella grafica 3D, infatti, solitamente la gerarchia è usata per decomporre una scena in oggetti, ogni oggetto in parti, ogni parte in poligoni. Un sistema di riferimento viene associato al nodo radice (root), e le informazioni geometriche che descrivono ogni oggetto vengono immagazzinate in una lista di poligoni relativa ad ogni nodo. Una possibile limitazione è che i poligoni possano essere associati solo a nodi foglia (leaf); in tal modo i nodi dei livelli superiori sono relativi solo a trasformazioni geometriche e ad organizzazioni spaziali (ad esempio collezioni di oggetti, o cluster), ma non a informazioni geometrico-descrittive.

La costruzione di uno SG implica la presenza di una fase di pre-processing nella quale vengono eseguite le operazioni di suddivisione una volta per tutte. La complessità di tali operazioni non è limitata in linea di principio, dal momento che questa fase non viene eseguita in tempo reale e dunque non costituisce un carico aggiuntivo alle elaborazioni grafiche.

4.2 Strutture adottate

Le strutture gerarchiche tipicamente usate nella grafica 3D sono i BSP²³ tree, di cui gli octree sono un caso particolare, e i Bounding Volume (BV) tree. I due tipi di gerarchia si differenziano essenzialmente per la modalità di suddivisione dello spazio. Nel caso dei BSP tree tutto lo spazio viene ripartito e le partizioni possono contenere o meno oggetti mentre nel caso dei BV tree gli oggetti, o gruppi (cluster) di oggetti, vengono racchiusi in apposite strutture (bounding volume) che li racchiudano. I bounding volume sono dunque volumi di spazio, solitamente sfere o box (scatola), che danno un'informazione circa il limite

²³ BSP: Binary Space Partition.

superiore dello spazio occupato dagli oggetti in esso racchiusi; se, pertanto, un bounding volume risulta escluso dalla visibilità, lo sono anche tutti gli oggetti in esso contenuti.

Una volta costruito l'albero, la fase di visualizzazione in tempo reale consiste nell'attraversare l'albero e verificare le proprietà dei nodi, a seconda dello scopo che si vuole raggiungere: se, ad esempio, si vuole verificare la collisione tra due oggetti si può, in prima approssimazione, verificare se occupano lo stesso spazio; se, invece, si vogliono effettuare controlli sulla visibilità degli oggetti, si può controllare se il nodo in esame è incluso nel viewfrustum (volume di vista, spiegato in seguito) e così via.

Si noti che la stessa suddivisione gerarchica può essere applicata non solo all'intera scena, ma anche ai singoli oggetti: in questo caso i nodi conterranno sub-oggetti o direttamente primitive geometriche. Nel caso invece la gerarchia sia applicata all'intera scena, ogni nodo conterrà un puntatore alle strutture dati relative agli oggetti facenti parti della regione di spazio relativa al nodo.

4.2.1 Octree

L'octree è la più semplice forma di organizzazione gerarchica dello spazio. Si immagini di circoscrivere con un cubo lo spazio occupato da una scena. Questo cubo, che rappresenta il nodo radice dell'albero, viene suddiviso in otto sottocubi, dividendo ogni faccia in quattro parti e raccordando i vertici così creati. Ognuno di questi sottocubi è dunque un nodo dell'albero che può essere a sua volta suddiviso.

Ci sono due criteri di terminazione della suddivisione. Il primo è quello di stabilire a priori la profondità massima dell'albero per cui, una volta raggiunto l'ultimo livello (che corrisponde alla cella di dimensioni minime) la suddivisione si arresta. L'altro consiste nello stabilire un numero minimo di oggetti (o di poligoni, se la suddivisione è applicata ad un oggetto) contenuti all'interno del nodo: quando al nodo è associato un numero minore o uguale a

tale minimo, la suddivisione si arresta. Da questo discende, banalmente, che un nodo vuoto non viene ulteriormente suddiviso, come è logico attendersi.

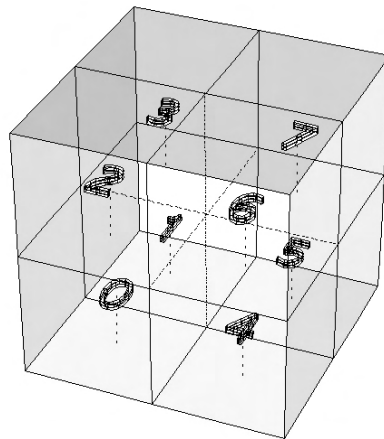


Figura 4.3: esempio grafico di un Octree

4.2.2 BSP-tree

Un BSP tree può essere considerato una generalizzazione dell'octree. In questo tipo di albero, la suddivisione viene effettuata dividendo ogni livello in due parti mediante un piano di separazione; se tale suddivisione viene effettuata sempre su tutte e tre le dimensioni si ottiene un octree, altrimenti un BSP di tipo più generale. Qualora i piani di separazione siano sempre orientati secondo gli assi principali si parla di KD tree (K-Dimensional tree).

Per quanto detto, un BSP tree con elementi cubici è sostanzialmente un octree, dunque vale la pena di esaminare quale tipo di elementi può far ritenere vantaggioso l'uso di questo tipo di alberi. Innanzitutto è facile notare che un octree rappresenta una soluzione ottimale quando la distribuzione degli oggetti nello spazio (o dei poligoni nell'oggetto) è uniforme, mentre può rappresentare uno spreco di risorse qualora questa uniformità non vi sia. In questo caso, allora, è possibile procedere con la suddivisione solo sulle dimensioni sulle quali ci sia un'elevata densità di oggetti, e lasciare indiviso lo spazio in cui la densità sia minore. In figura 4.4 e 4.5 è mostrata, in due dimensioni, la differenza tra una suddivisione uniforme (tipo octree – nel caso bidimensionale è più appropriato parlare di quadtree) e una suddivisione BSP dello stesso spazio, in cui la scelta del piano di divisione è compiuta cercando di dividere ogni volta uno spazio in

modo che ci sia lo stesso numero di oggetti nelle due parti, così da avere un albero più bilanciato.

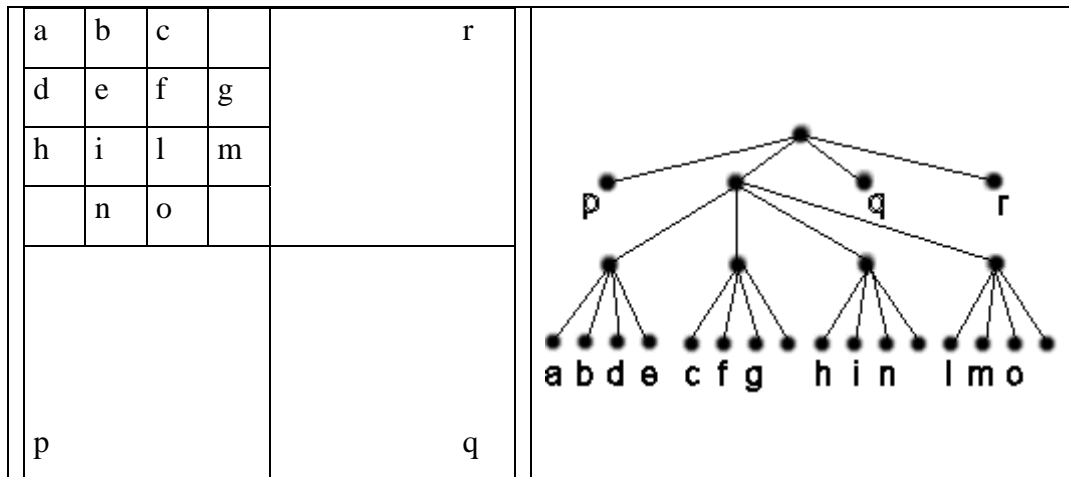


Figura 4.4: suddivisione spaziale con Octree

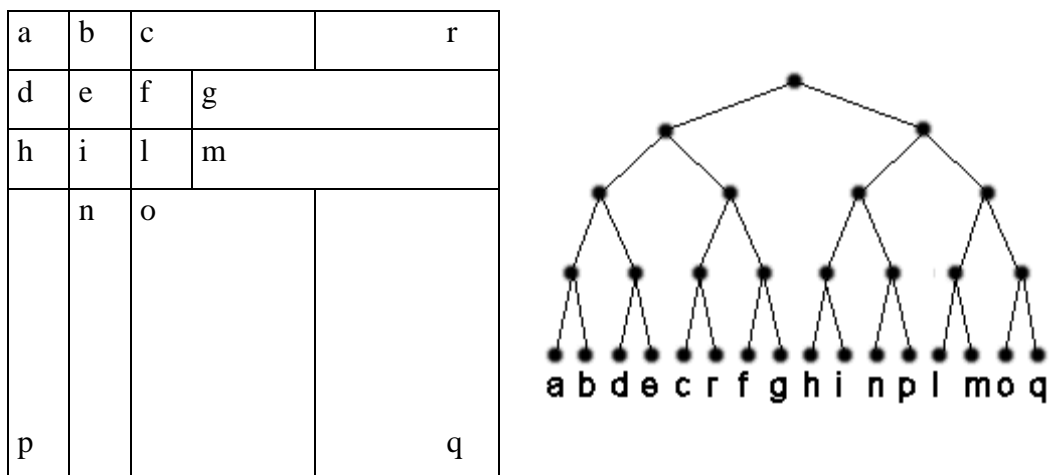


Figura 4.5: suddivisione spaziale con BSP

4.2.3 Gerarchia di un Bounding Volume

L'approccio gerarchico mediante i Bounding Volume potrebbe essere definito di tipo bottom-up, rispetto a quello tipicamente top-down degli alberi; in questo caso, infatti, si parte dagli oggetti, o da cluster di oggetti vicini, e successivamente si individuano i bounding volumes minimi in grado di racchiuderli completamente. Tali BV possono essere a loro volta clusterizzati e

racchiusi in BV di livello superiore. La scelta del tipo di BV dipende dal tipo di ottimizzazione che si vuole ottenere. Solitamente si usano sfere, box con lati paralleli agli assi principali²⁴, o box con orientazione qualsiasi²⁵

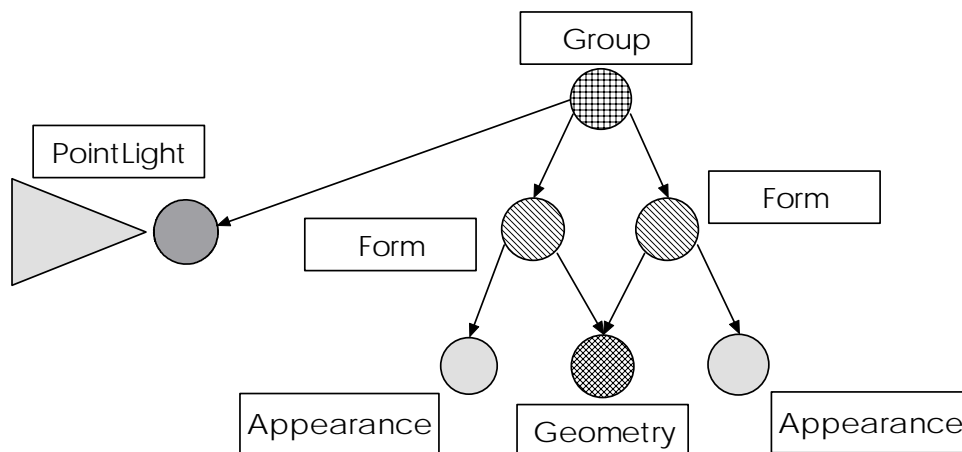


Figura 4.6: esempio di un tipico Scene Graph che segue l'approccio gerarchico mediante Bounding Volume.

Più semplice è il volume, più rapido è il test che però rischia di non essere efficiente dal punto di vista della rappresentazione. Si consideri una sfera, ad esempio, destinata a racchiudere un oggetto alto e stretto: in questo caso il BV non è affatto rappresentativo dell'oggetto in esso contenuto, e gli eventuali test (di visibilità, per esempio) fatti sul BV sono inefficienti.

4.3 Tecniche di grafica

Negli ultimi anni lo sviluppo dell'hardware grafico ha raggiunto livelli elevati; la crescita di prestazioni fa sì che ci si spinga nella realizzazione di modelli sempre più dettagliati e complessi. La complessità di un modello geometrico può però risultare troppo elevata anche per l'odierno hardware, tanto da poter pregiudicare l'interattività della rappresentazione. Sono perciò sempre presenti

²⁴ Axis Aligned Bounding Boxes, o AABB.

²⁵ Oriented Bounding Boxes, o OBB.

tecniche software che agiscono sulle proprietà geometriche o visive del modello per semplificarlo in maniera tale da raggiungere i migliori risultati di visualizzazione nel rispetto dei forti vincoli temporali imposti.

Tali tecniche di ottimizzazione possono essere suddivise in base al tipo di semplificazione che adottano; in particolare si parla di:

- tecniche di *culling*, se agiscono sulla visibilità del modello;
- tecniche di *semplificazione*, se intervengono sulla struttura del modello.

Va rilevato che entrambe le tipologie possono essere adottate contemporaneamente e anzi sono complementari.

4.3.1 Tecniche di culling

Le tecniche di culling determinano un sottoinsieme di primitive del modello che non sono visibili dall'attuale punto di vista e che, perciò, non devono essere passate alla pipeline di renderizzazione. Fra di esse si riconoscono quelle di *Viewfrustum culling*, di *Backface culling* e di *Occlusion culling*.

4.3.1.1 Viewfrustum Culling

Il Viewfrustum Culling è basato sull'individuazione delle primitive che non rientrano nel volume di vista e che pertanto non sono visibili dall'osservatore (Figura. 4.7); i risultati raggiunti da tale algoritmo sono tanto migliori quanto più piccola è la porzione del modello che risulta visualizzata.

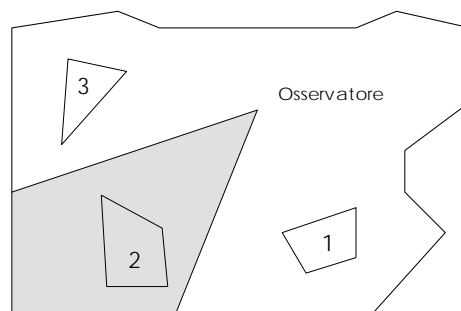


Figura 4.7: i poligoni che non rientrano nel volume di vista (1 e 3) non vengono renderizzati.

Come anticipato in precedenza, il volume di vista (*ViewFrustum*) è realizzato come un tronco di piramide a base quadrata, approssimazione del reale cono visivo, avente il vertice nell'osservatore e le basi in due piani paralleli. La geometria può essere divisa in una porzione visibile, quella all'interno del viewfrustum, e in una porzione esterna ad esso, e quindi non visibile; per tale motivo essa può essere rimossa dalla scena (*culled*).

Nel caso di scene molto complesse il calcolo della visibilità, se svolto direttamente sulle primitive geometriche, risulta inefficiente; diventa perciò necessario adottare delle tecniche di partizionamento del modello, come proposto da Clark nel 1976. Il partizionamento viene effettuato in una fase di pre-processing, e perciò il carico computazionale di questa elaborazione non va ad appesantire la visualizzazione interattiva.

Una volta realizzato il partizionamento del modello, i controlli sulla visibilità vengono effettuati sui bounding volumes invece che direttamente sulle singole primitive, con evidente risparmio di tempo ed aumento dell'efficienza.

Nel caso che il bounding volume non sia banalmente accettato perché completamente incluso nel viewfrustum, o rifiutato perché completamente escluso, sono possibili varie alternative. Una possibile soluzione è quella di procedere al test sulle singole primitive incluse nella bounding box che interseca il viewfrustum, ma se queste primitive sono numerose si può verificare un calo dell'efficacia. Più efficiente è la soluzione di renderizzare in ogni caso tutte le primitive contenute, anche se alcune di esse non sono incluse nel volume visibile. Questa soluzione più tollerante ha infatti il grosso vantaggio di essere velocissima riuscendo così a compensare efficacemente il dispendio di risorse introdotto dalla renderizzazione di queste primitive.

Uno dei test più veloci è quello svolto sulle bounding spheres (Figura. 4.8), perché la particolare forma geometrica di questi volumi consente di semplificare le elaborazioni e quindi di abbreviare i tempi di calcolo.

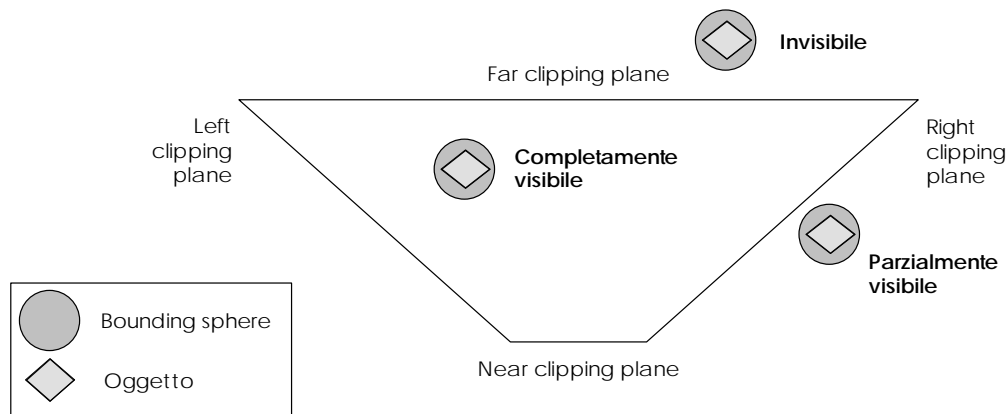


Figura 4.8: esempio di test effettuati sulle bounding-sphere

I bounding volumes possono essere organizzati in maniera gerarchica partendo da un unico volume che racchiude l'intera scena e suddividendo ricorsivamente la geometria in base a criteri di partizionamento spaziale, come ad esempio l'octree.

4.3.1.2 Backface Culling

Ogni poligono inviato alla pipeline grafica ha associato un vettore normale che ne descrive l'orientamento nello spazio. Se la normale è rivolta verso l'osservatore il poligono è detto *frontfacing*, altrimenti viene detto *backfacing*. Nella visualizzazione di oggetti solidi racchiusi da una superficie poligonale chiusa, i poligoni backfacing risultano invisibili all'osservatore perché nascosti da quelli frontfacing.

Il principio su cui è basato il Backface Culling è che, rispetto ad un determinato punto di vista, solo una parte dei poligoni della scena risulta frontfacing; evitando di inviare alla pipeline i poligoni backfacing, si otterrebbe in media un dimezzamento dei tempi di renderizzazione.

Nella pratica il miglioramento effettivo è al di sotto di tale soglia dal momento che in realtà è possibile conoscere l'orientamento di un poligono solo dopo le trasformazioni geometriche.

Molte delle architetture hardware moderne effettuano un efficiente test di backface culling che però può avere luogo solo dopo i calcoli di roto-traslazione,

consentendo di scaricare i poligoni backfacing prima degli stadi di illuminazione e rasterizzazione. E' possibile affiancare a questo test una più efficiente procedura software che sia in grado di effettuare il test prima ancora dei calcoli geometrici.

Per effettuare il culling bisogna che ogni poligono sia sottoposto a un test, detto *backface test*, costituito dal calcolo di un prodotto scalare fra la normale al poligono e la direzione di vista. Se il risultato è negativo, il poligono è rivolto verso l'osservatore, altrimenti è rivolto dall'altra parte e può essere considerato *backfacing* (Figura 4.9).

Per effettuare il culling bisogna che ogni poligono sia sottoposto a un test, detto *backface test*, costituito dal calcolo di un prodotto scalare fra la normale al poligono e la direzione di vista. Se il risultato è negativo, il poligono è rivolto verso l'osservatore, altrimenti è rivolto dall'altra parte e può essere considerato *backfacing* (Figura 4.9).

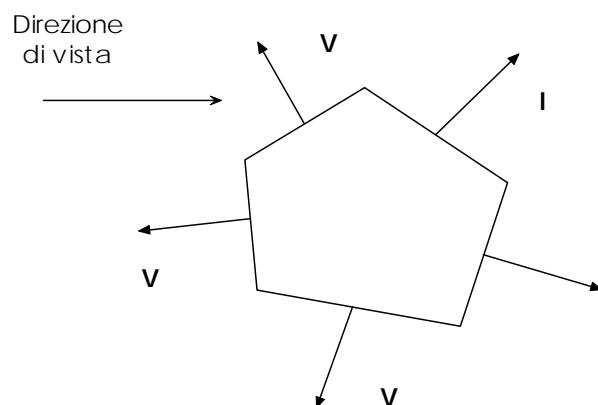


Figura 4.9: le facce indicate con V sono visibili; quelle con I sono invisibili.

Come visto per il viewfrustum culling, anche il backface culling può essere reso più efficiente se i poligoni vengono raggruppati per affinità e sottoposti al culling in gruppo (cluster) piuttosto che uno per volta.

4.3.1.3 Occlusion Culling

Lo scopo dell'Occlusion culling è quello di escludere dalla renderizzazione le primitive che, pur rientrando nel volume di vista, risultano nascoste da altre primitive situate in posizioni più vicine all'osservatore.

Una delle strategie più comunemente usate per l'occlusion culling è la tecnica dello “Z-buffering”²⁶; nel caso ci siano più oggetti in fila uno dietro l'altro dei quali, ad esempio, solo il primo sia visibile, si calcola il valore delle coordinate z per ognuno di questi oggetti e lo si compara con quello presente nello z-buffer, in maniera da disegnare sullo schermo solo quello più vicino all'osservatore. Ma ciò implica ancora una volta che tutti i calcoli, le trasformazioni e le elaborazioni geometriche siano comunque effettuate su tutti gli oggetti.

Le tecniche di occlusion culling, invece, fanno in modo che gli oggetti nascosti non arrivino affatto alla pipeline grafica.

4.3.1.4 Potentially Visible Sets

Una tipologia di Visibility Culling che ingloba molti aspetti delle tecniche finora analizzate, ma che agisce ad un livello più alto, è la tecnica dei *Potentially Visible Sets* (PVS). Tale tecnica, ideale per modelli architettonici, consiste nella suddivisione dell'ambiente in *celle* e *portali*. La cella è definita come un volume di spazio, solitamente un parallelepipedo; il portale è una regione 2D trasparente, situata al confine di una cella, che unisce celle adiacenti. Nei modelli architettonici una stanza è rappresentata da una cella, mentre una porta o una finestra costituiscono un portale attraverso il quale celle adiacenti possono vedersi l'un l'altra.

Il Potentially Visible Set è l'insieme delle celle che sono potenzialmente visibili da un dato punto di vista; la cella attiva, quella in cui si trova l'osservatore, fa parte del PVS come pure le celle adiacenti connesse alla cella attiva da un portale, e via via tutte le celle visibili dai portali della cella attiva (Figura.4.10).

²⁶ E' il processo che permette di stabilire in ogni momento quali sono gli oggetti in primo piano e di rimuovere dalla scena le linee nascoste.

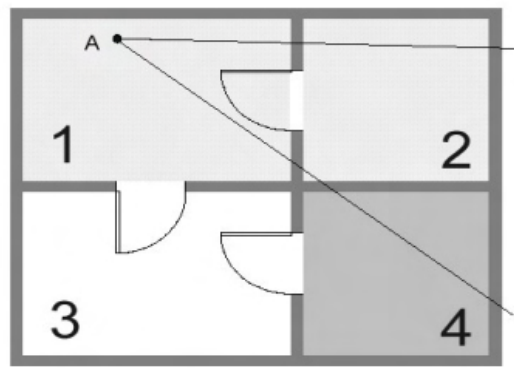


Figura 4.10: dal punto di vista A, sono visibili le celle 1 e 2. La cella 3 non fa parte del Viewfrustum. La 4, pur rientrando nel Viewfrustum, non è visibile da A, per cui non fa parte del PVS.

4.3.2 Tecniche di semplificazione

Un nuovo campo di studio ancora aperto a nuove idee e sperimentazioni, è quello relativo alle tecniche di semplificazione del modello. Sono stati proposti due differenti approcci per raggiungere tale obiettivo, entrambi fondati sulla considerazione che, a seconda della posizione dell'osservatore, non tutte le porzioni visibili del modello necessitano dello stesso grado di fedeltà della rappresentazione, ma possono essere semplificate in base alla loro angolazione o alla loro distanza dall'osservatore.

4.3.2.1 Level of Detail

Il *Level of Detail* (LOD) *modeling* è basato sul principio che gli oggetti lontani dall'osservatore non necessitano di una rappresentazione estremamente particolareggiata, dal momento che i piccoli dettagli risultano indistinguibili oltre una certa distanza. Con il LOD modeling si forniscono rappresentazioni progressivamente meno dettagliate di un modello man mano che aumenta la distanza fra questo e l'osservatore.

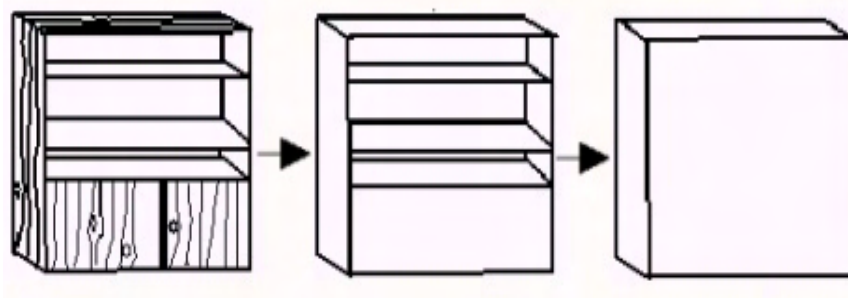


Figura 4.11: oggetto rappresentato da tre LOD con livello di dettaglio decrescente.

La tecnica del LOD è ben conosciuta e sfruttata già da molto tempo soprattutto in campi ben specifici, come i simulatori di volo, ma già nel '93 è stata implementata nella rappresentazione di modelli ad elevato grado di complessità. Esistono varie versioni del LOD modelling, distinguibili per la modalità di creazione dei livelli di dettaglio. In particolare tali livelli possono essere creati off-line, precostituendo rappresentazioni alternative ben determinate, oppure dinamicamente. Uno dei metodi per la costruzione di LOD dinamici consiste nel far collassare più vertici adiacenti in un unico vertice *rappresentativo*.

Il primo approccio offre una maggiore garanzia sulla rappresentazione nel senso che, essendo i LOD precostituiti, si ha sempre la certezza del modo con il quale viene rappresentato il modello, ma al costo di un certo tempo di pre-computazione e di un maggior carico di memoria impiegata per conservare tutti i modelli.

Il secondo approccio, al prezzo di una minore fedeltà o comunque del non totale controllo su di essa, offre minor dispendio di memoria e la riduzione dei tempi di pre-processing.

Va considerato che un sistema di LOD precostituiti ha per sua natura un numero prefissato e dunque limitato di LOD; se invece i LOD vengono generati dinamicamente è possibile un continuo raffinamento del modello aggiungendogli piccole quantità di dettagli locali.

Hoppe ha proposto una struttura incrementale (*mesh progressiva*) in cui i vari LOD sono creati dinamicamente, ordinati a partire da quello più grossolano per

arrivare via via al LOD corrispondente al modello originale. Ogni LOD può essere rappresentato come differenza dal LOD immediatamente precedente, in maniera che sia possibile riutilizzare i dati dei LOD meno dettagliati anche per quelli più definiti; in questo modo si riesce a contenere le strutture dati entro dimensioni ragionevoli. Un approccio di tale tipo risulta particolarmente conveniente se i dati devono essere trasmessi verso reti, come ad esempio nelle applicazioni distribuite o nelle applicazioni Internet.

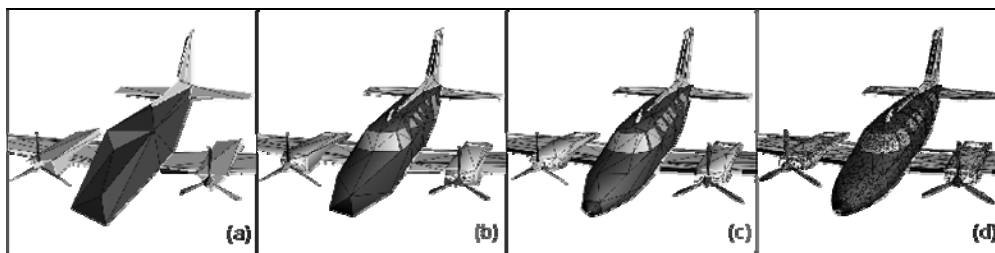


Figura 4.12: esempio di mesh progressiva. Il modello originale (d) è composto da 13546 poligoni; gli altri, da (c) ad (a), rispettivamente da 1000, 500 e 150

In questo caso vengono trasmessi dapprima i LOD più semplici; man mano che ci si avvicina all'oggetto vengono trasmessi i dati che, per aggiunta, vanno a costituire i LOD più dettagliati. Il vantaggio in questo caso sta nel fatto che le trasmissioni riguardano i soli incrementi, e dunque sono di dimensioni contenute; inoltre, poiché possono essere rappresentati soltanto i LOD arrivati completamente a destinazione, se al momento della renderizzazione non sono stati ricevuti tutti i dati relativi al successivo LOD, è sempre possibile usare la rappresentazione precedente.

4.3.2.2 Image based Rendering

Le tecniche di culling e di level of detail hanno in comune il fatto di essere basate su considerazioni geometriche; il loro obiettivo consiste, infatti, nel ridurre il numero totale di poligoni che devono essere renderizzati.

Un approccio totalmente diverso è quello dell'image-based rendering nel quale si rappresenta, dinamicamente, geometria più o meno complessa tramite l'uso di texture. Purtroppo le texture hanno molte limitazioni; sono, infatti, utilizzabili

solo come complemento della geometria per cui, per descrivere la forma di un oggetto, devono essere *mappate* su un modello geometrico già esistente; ne possono però ridurre, anche notevolmente, la complessità simulando effetti visuali complessi come nel caso del *bump mapping* (Figura 4.13), una tecnica che simula depressioni e rilievi su una superficie senza la necessità che tali rilievi siano modellati geometricamente. A questo scopo si agisce sulle normali alla superficie introducendo su di esse delle perturbazioni che ingannano il modello di riflessione della luce.



Figura 4.13: esempi di bump mapping

Il texture mapping permette di dare una maggiore ricchezza di dettaglio al modello, ma può essere sfruttato anche per scopi di semplificazione dello stesso. Ad esempio nelle tecniche di *image morphing* si renderizza solo una porzione dei frame richiesti dai metodi convenzionali, e si ricavano gli altri mediante il morphing dai frame già renderizzati; con questa tecnica si ottiene, assumendo che il morphing sia efficiente, un notevole aumento di velocità, anche se l'immagine ricavata dal morphing è solo un'approssimazione dell'immagine reale.

L'image morphing non è tuttavia un tipo di image based rendering in senso stretto ma, oltre al modo appena descritto, può essere utilizzato in maniera più "locale".

Un'altra metodologia, comunemente chiamata *Database Approach*, consiste nel pre-renderizzare tutte le possibili viste di un oggetto, immagazzinarle e

visualizzare dinamicamente ogni volta la vista giusta. Questo approccio presenta lo svantaggio di richiedere un'ingente quantità di memoria, ma è applicabile comunque a piccoli oggetti, pre-renderizzando solo un numero finito di viste e considerando accettabile l'approssimazione introdotta dalla non disponibilità delle viste intermedie.

Un semplice esempio di Image Based Rendering è quello dei *billboards* in cui oggetti molto complessi, come ad esempio gli alberi, sono disegnati mediante geometria semplice con l'ausilio del texture mapping; la geometria viene trasformata affinché, frame per frame, sia sempre rivolta verso il punto di vista.

4.3.2.3 Image caching

Come detto in precedenza, le tecniche di culling e di LOD per migliorare il frame rate mettono l'accento sulla riduzione della quantità di geometria da elaborare, in maniera da alleggerire il carico computazionale. Però nessuna di loro sfrutta la coerenza tra i frame consecutivi in una sequenza grafica (*frame-to-frame coherence*). Negli attuali sistemi grafici il costo della renderizzazione dell'intera geometria, dovendo ripartire da zero ad ogni frame, è molto alto. In questo modo l'immagine rasterizzata in un frame non viene riutilizzata nel frame successivo; questo sebbene vi sia un'elevata continuità nella sequenza di fotogrammi generata dall'hardware grafico, in quanto il moto dell'osservatore è solitamente di velocità moderata e il moto relativo degli oggetti lontani sullo schermo è di entità tanto minore quanto più tali oggetti sono lontani dal punto di vista. Ogni frame presenta quindi, a meno di cambi di visuale, una notevole percentuale di punti in comune con il frame precedente e con quello successivo. Una tecnica interessante è quella di riutilizzare porzioni dello schermo che, essendo relative ad oggetti lontani, verosimilmente nel frame successivo sarebbero renderizzati esattamente allo stesso modo, in quanto è plausibile che i pixel di tali porzioni non cambino il loro stato.

Un modo per sfruttare la *frame-to-frame coherence* va sotto il nome di *tecnica degli impostori*; in questo caso le immagini rasterizzate di questi oggetti non

vengono scartate ma vengono memorizzate come texture per poter essere riutilizzate nei frame successivi; tali texture sono poi applicate su poligoni che prendono il nome di *impostori*.

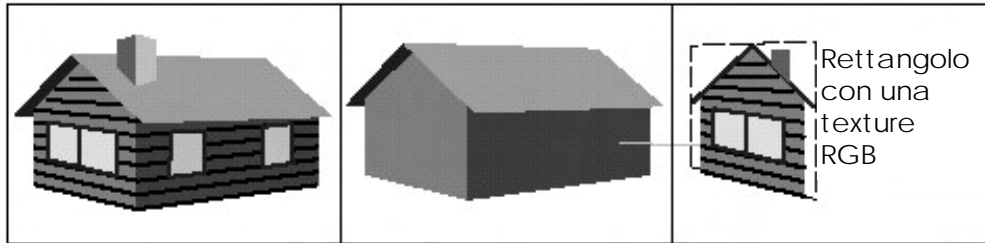


Figura 4.14: tre diverse rappresentazioni per lo stesso oggetto: a destra la texture da inserire, al centro l'oggetto 3D di destinazione e a sinistra il risultato della fusione.

4.4 Caratteristiche tecniche della Scene Graph

Le ragioni chiave che portano tanti sviluppatori grafici ad usare lo Scene Graph sono determinate in termini di Performance, Produttività, Portabilità e Scalabilità:

- *Performance*

Lo Scene Graph mette a disposizione un eccellente motore per massimizzare le performance grafiche. Un buon Scene Graph impiega due tecniche chiave: scartare velocemente gli oggetti che non sono visibili nella scena, ed avere la capacità di ordinare gli oggetti per le loro proprietà, come texture e materiali, in modo da permettere che tutti gli oggetti simili vengano disegnati insieme. Senza la fase di scarto la CPU, i bus e la GPU sarebbero travolte da un enorme carico di dati insignificanti per la rappresentazione della scena. La struttura gerarchica dello Scene Graph rende questa fase di scarto veramente efficiente. Inoltre, senza la fase di ordinamento, i bus e la GPU sarebbero costretti a passare da uno stato all'altro causando uno stallo della pipeline grafica e distruggendo il throughput grafico. Invece, con questa fase, la GPU

diventa sempre più veloce e lo stallo della pipeline grafica diventa sempre meno probabile.

- *Produttività*

Lo Scene Graph possiede molti strumenti per lo sviluppo di applicazioni grafiche performanti. Esso gestisce al posto dell'utente tutta la parte grafica, riducendo a sole poche funzioni quello che con OpenGL si farebbe con tantissime righe di codice. Inoltre, uno dei concetti più importanti della programmazione Object Oriented è la composizione di oggetti, custodito perfettamente nel Composite Design Pattern, che adatta perfettamente la struttura ad albero dello Scene Graph e lo rende altamente flessibile e con un design riutilizzabile (con questo si intende che può essere facilmente adattato per risolvere ogni tipo di problema). Scene Graph può anche diventare una libreria aggiuntiva per aiutare gli utenti a impostare e controllare la finestra grafica e importare facilmente immagini e modelli 3D. Tutto questo permette all'utente di ottenere un grande progetto con un piccolo codice (un dozzina di linee di codice bastano per aprire dei dati e creare una visualizzazione interattiva).

- *Portabilità*

Scene Graph incapsula al suo interno molti programmi di basso livello per il rendering grafico, la lettura e la scrittura di dati, riducendo o annullando il codice specifico di una piattaforma che serve nella propria applicazione. Se le linee base dello Scene Graph sono portabili così da essere portate da piattaforma a piattaforma, di conseguenza sarà altrettanto semplice ricompilare il proprio codice sorgente ed eseguirlo in ogni tipo di piattaforma.

- *Scalabilità*

La grafica 3D si sposa bene con configurazioni hardware complesse e avanzate come clusters di macchine grafiche, o sistemi multiprocessore o

multipipe come SGI's Onyx, e lo Scene Graph rende decisamente facile gestire questi tipi di sistemi. Infatti, aiuterà gli sviluppatori a concentrarsi nel programmare le proprie applicazioni senza preoccuparsi di gestire l'hardware, mentre il motore del rendering dello Scene Graph si occuperà delle diverse configurazione hardware.

4.5 La libreria Open Scene Graph

OpenSceneGraph (OSG) è una libreria grafica 3D portabile, open source, realizzata per progetti a prestazioni elevate e quindi usata per applicazioni come simulazioni di volo, giochi, realtà virtuale, modellazione e visualizzazione scientifica, ecc... [D01] [D02].

Questa libreria è scritta interamente in C++ (Object Oriented) e OpenGL e può essere portata in tutte le piattaforme e sistemi operativi Windows, OSX, Linux, IRIX, Solaris e FreeBSD [D03]. La sua grande portabilità è uno dei punti chiave della sua diffusione.

Il progetto partì come hobby di Don Burns nel 1998, come mezzo per rendere portabile un simulatore di caduta planare scritta al massimo delle performance grafiche eseguite su un IRIX [D04]. Nel 1999, Robert Osfield iniziò ad aiutarlo col simulatore concentrandosi sull'implementazione degli elementi grafici per Windows. Nel settembre del 1999 il codice sorgente divenne open source, e nacque il sito Internet²⁷.

Nell'aprile del 2001, in risposta alla grande crescita dell'interesse nazionale, Robert si impegnò a tempo pieno sul progetto, creando un supporto commerciale chiamato OpenSceneGraph Professional Service che comprendeva un servizio di consulenza per l'apprendimento. Alla fine del 2001 Don Burns formò una sua compagnia chiamata Andes Computer Engineering e partecipò alla progettazione e al supporto di OpenSceneGraph ma con un progetto complementare chiamato OpenProducer e BluMarbleViewer [D05].

²⁷ <http://www.openscenegraph.org>.



Figura 4.15: esempi di applicazioni con OSG.

OSG integra al suo interno il sistema di finestre `osgProducer` che sfrutta `OpenProducer`, ma ci sono anche esempi di come interfacciare correttamente OSG con altre librerie come `GLUT`, `Qt`, `MFC`, `WxWindows` e `SDL`. Alcuni utenti hanno integrato OSG anche con `Motif` e `X`.

Tutt'ora il progetto `OpenSceneGraph` è in completa evoluzione ed è arrivato alla versione 0.9.8 comprendente 3 pacchetti: `OpenProducer`, `OpenThreads` e `OpenSceneGraph`, descritti nei paragrafi seguenti.

4.5.1 Open Producer

`OpenProducer` (o semplicemente `Producer`) è una libreria C++ ideata per gestire i processi di rendering di `OpenGL` in un sistema a finestre indipendenti. Il `Producer` permette un semplice e potente approccio scalabile per applicazioni 3D real-time che volessero essere eseguite con un'unica grande finestra in un sistema multi-display.

`OpenProducer` è molto portabile ed è stato testato su sistemi `Linux`, `Windows`, `Mac OSX`, `Solaris` e `IRIX`. Il `Producer` lavora su tutti i sistemi operativi basati su `Unix` attraverso l'`X11 Windowing System` e su tutti i sistemi `Windows` attraverso il `Win32` nativo.

Inoltre, l'`OpenProducer` è altamente produttivo, con grandi performance e facilmente scalabile, per sistemi grafici complessi come clusters.

La libreria `OpenProducer` offre al programmatore tutto ciò che serve per visualizzare e renderizzare la scena 3D, e più precisamente:

- *Camera*

Camera è un oggetto che cattura la scena con un frame rate fissato. Essa ha come attributi la sua posizione (Position) e la direzione in cui punta (LookAt), entrambi definiti nel mondo a tre dimensioni. La camera ha una lente (Lens), e una RenderSurface.

- *Lens*

Lens è un oggetto che definisce i parametri di proiezione per proiettare il mondo a tre dimensioni in un rettangolo a 2 dimensioni. Internamente, essa è una matrice di proiezione che può essere modificata direttamente o tramite alcuni metodi messi a disposizione per facilitarne la programmazione.

- *RenderSurface*

RenderSurface è un oggetto che definisce il buffer in cui la scena verrà renderizzata. A primo avviso sembra appropriato definirla come analogo del sistema a finestre di Windows, tuttavia differisce da esso in quanto offre componenti utili ai programmatori di grafica 3D come la qualità del formato dei pixel. Inoltre gestisce internamente gli eventi di configurazione cosicché il programmatore non dovrà manipolarli quando la finestra verrà ridimensionata, spostata, ridotta a icona o cambiata esternamente.

- *CameraGroup*

Come il nome spiega, CameraGroup è un gruppo di camere sincronizzate in modo da far iniziare i loro frames allo stesso tempo e renderizzarle contemporaneamente. Molti metodi che CameraGroup mette a disposizione sono gli stessi della Camera singola, con la differenza che verranno applicate a tutte le camere.

- *CameraConfig*

CameraConfig è una classe che descrive la configurazione di un CameraGroup. Esso deve essere esplicitamente espresso tramite un API o configurato direttamente in un file di configurazione.

- *KeyboardMouse*

La classe KeyboardMouse fornisce al programmatore un modo di intercettare e gestire gli eventi sollevati dalla tastiera e dal mouse tramite una classe Callback. Essa può intercettare questi eventi da una singola finestra, o da finestre multiple tramite una InputArea.

- *InputArea*

InputArea è una configurazione di rettangoli di input che, una volta combinati, descrivono una area di input per gli eventi della tastiera e del mouse. Gli eventi della tastiera verranno gestiti nel modo standard, mentre quelli del mouse descriveranno la posizione del puntatore nello spazio delle coordinate descritto in esso o la lista di rettangoli dell'InputArea.

4.5.2 Open Threads

È una libreria pensata per fornire una minima ma completa interfaccia Thread Object Oriented per i programmatori di C++. OpenThreads è modellata liberamente sulle Java threadAPI e sugli standard POSIX Threads. L'architettura della libreria è disegnata attorno ad un modello thread "swappabile" definito come oggetto condiviso dalla libreria a tempo di compilazione.

Questa libreria non è direttamente utilizzata dal programmatore, ma serve per l'utilizzo e la compilazione delle librerie OpenProducer e OpenSceneGraph.

OpenThreads mette a disposizione dei vari oggetti i quattro tipi fondamentali della programmazione threads, ovvero Thread, Mutex, Barrier e Condition.

4.5.3 Open Scene Graph

Il pacchetto OpenSceneGraph contiene al suo interno tutti i principali core (letteralmente tradotto: cuore) della libreria indispensabili per il suo corretto funzionamento.

Ecco la lista dei core di OpenSceneGraph:

- *osg*: è il cuore dello Scene Graph, include anche il supporto per l'apertura di file .osg e formati di file RGB.
- *osgUtil*: è la classe che contiene al suo interno numerose utility come il setup o il rendering/draw.
- *osgDB*: è la libreria delle utility per la gestione di letture e scritture di database 3D e files immagine.
- *osgParticle*: è il pacchetto di gestione dell'effetto particelle.
- *osgText*: è il pacchetto per la gestione del testo; contiene al suo interno le classi per il rendering di font true type.
- *osgSim*: è una libreria per le simulazioni visuali.
- *osgGA*: è la libreria per la gestione della Graphic User Interface (GUI) astratta.
- *osgProducer*: è la classe che contiene le basi per la visualizzazione della scena. Implementa anche un semplice demo del Producer per la costruzione di una finestra e l'interazione keyboard-mouse.

Come accennato precedentemente, OSG include numerosi Plug-in che semplificano le azioni di importazione ed esportazione di formati di file grafici nei diversi formati di ultime generazioni. Eccone alcuni esempi:

- *osgPlugin*: per la gestione di file nativi (.osg).
- *fltPlugin*: per la gestione di file Open Flight (.flt).
- *lib3dsPlugin*: per la gestione di file Auto Studio Max (.3ds).
- *lwoPlugin*: per la gestione di file Light Wave binari (.lwo, .lw, .geo).
- *rgbPlugin*: per la gestione di file RGB.

- pngPlugin: per la gestione di file PNG.
- gifPlugin: per la gestione di file GIF.
- jpegPlugin: per la gestione di file JPEG.
- tgaPlugin: per la gestione di file TGA.
- tiffPlugin: per la gestione di file TIF.

Per facilitare l'apprendimento del programmatore inesperto, il pacchetto OSG contiene al suo interno numerosi demo implementati che danno un piccolo assaggio della potenza e della facilità d'uso di OpenSceneGraph.

4.5.4 Vantaggi dell'OSG

Lo scopo dell'OSG è di rendere i vantaggi della tecnologia delle Scene Graph gratuitamente disponibili a tutti, per un utilizzo commerciale e non. Nonostante tale libreria sia ancora in via di sviluppo, ha già ottenuto una notevole fama nel mondo della grafica 3D per le sue qualità di performance, portabilità e purezza nel disegno. Scritta interamente in un linguaggio Standard C++ e OpenGL, fa pieno uso di STL (*Standard Template Library*) e di Design Patterns, e influenza il modello di sviluppo open source nel provvedere di una libreria liberamente accessibile atta alle necessità di qualsiasi programmatore. L'Open Scene Graph trasmette i suoi vantaggi, in base ai quattro punti di vista già elencati nella trattazione generale delle Scene Graph, come segue:

- *Performance*

La libreria supporta il view frustum culling, occlusion culling e small feature culling, nodi LOD, state sorting, vertex array e liste di display come parte del nucleo di una scene graph. Tutte queste caratteristiche messe assieme fanno dell'OSG una delle librerie con le più elevate prestazioni. Di conseguenza si ha che le performance uguagliano o sorpassano tutto ciò che era stato ottenuto con altre librerie di scene graph come Open Performer, VTree, Vega Scene Graph, e Java3D. L'OSG fornisce infatti una facile personalizzazione del processo di disegno, che ha permesso così l'implementazione delle

strutture con Livello Continuo di Dettaglio (*Continuous Level of Detail*, CLOD) nell'utilizzo più profondo delle scene graph. Queste consentono la visualizzazione di database di ampi spazi di terreno, ed esempi di questa metodologia possono essere consultati nel sito "VTerrain.org" e "TerrainEngine.com", entrambi integrati con OSG.

- *Produttività*

Alla base dell'OSG si provvede di incapsulare le maggior parte delle funzionalità di OpenGL, incluse le più recenti estensioni; vengono fornite inoltre ottimizzazioni a livello di rendering come il culling e il sorting, e un intero insieme di librerie add-on che permettono di sviluppare applicazioni grafiche con alte performance a livello di velocità.

Combinando le nozioni trasmesse dallo studio di librerie come Performer e Open Inventor, con i nuovi modelli di software engineering come Design Patterns, è stato possibile creare un'altra libreria che è pulita ed estensibile. Questo rende ancor più semplice adattarsi all'OSG e integrarla con le altre applicazioni.

Per leggere e scrivere database, una sotto libreria (osgDB) inserisce supporti per un'ampia varietà di formati tramite un meccanismo estensibile e dinamico – la distribuzione infatti include ora 33 separati plugins per caricare svariati formati di dati 3D e immagini.

Tali estensioni includono OpenFlight (.flt), TerraPage (.txp) che ingloba il supporto per il multi-threading, LightWave (.lwo), Alias Wavefront (.obj), Carbon Graphics GEO (.geo), 3D Studio MAX (.3ds), Peformer (.pfb), Quake Character Models (.md2). Direct X (.x), e Inventor Ascii 2.0 (.iv)/VRML 1.0 (.wrl), Designer Workshop (.dw) and AC3D (.ac) e il formato ASCII nativo .osg. I caricatori delle immagini comprendono .rgb, .gif, .jpg, .png, .tiff, .pic, .bmp, .dds (incluso il compressed mip mapped imagery), .tga

e quicktime (sotto OSX). Un intero insieme di alta qualità e font anti-alias possono essere così caricati tramite plugin freetype.

La scene graph ha così un insieme di *Node Kits* che sono librerie separate, che possono essere compilate insieme alle applicazioni o caricate a runtime, che aggiungono un supporto per sistemi Particle (osgParticle), testo di alta qualità anti-alias e punti di luce regolabili (osgSim).

La società di sviluppo ha così ampliato il numero di *Node Kits* addizionali come osgNV (che include la compatibilità con i dati nVidia e il linguaggio NVidia's Cg), Demeter (terreni CLOD più l'integrazione con OSG), osgCal (che incapsula Cal3D e l'OSG), osgVortex (che comprende il motore fisico CM-Labs Vortex con OSG) e una grande varietà di librerie che integrano le API di Windowing.

Il progetto è stato inoltre integrato con i sistemi di realtà virtuale “VR Juggler” e “Vess”.

- *Portabilità*

La struttura della scene graph è stata così ridisegnata per avere una minima dipendenza su alcune specifiche piattaforme, richiedendo poco più che lo Standard C++ e OpenGL. Ciò ha consentito alla scene graph di essere rapidamente portabile su una vasta quantità di piattaforme – originariamente sviluppata su IRIX, poi su Linux, e infine su Window, FreeBSD, Mac OSX, Solaris, HP-UX e, negli ultimi mesi, su PlayStation2. Il fatto di essere completamente indipendente e compatibile allo stesso tempo da sistemi windowing, rende ancor più facile agli utenti aggiungere le proprie librerie (specifiche per window) e applicazioni alla struttura dell'OSG. Nel pacchetto OSG viene incluso infatti la libreria osgProducer che si integra con l'“OpenProducer”, e sono già stati implementati con successo applicazioni in combinazione con GLUT, Qt, MFC, WxWindows e SDL (come d'altronde è

stata sviluppato tale progetto di tesi). I programmatori hanno così integrato l'OSG con Motif e X.

- *Scalabilità*

L'OSG non solo gira su quasi la totalità dei sistemi operativi, ma integra i molteplici sottosistemi grafici inclusi su macchine come un Onyx multipipe. Questo è possibile poiché l'OSG supporta svariati contesti grafici per entrambi OpenGL Display Lists e oggetti texture, e le operazioni di cull e draw sono state predisposte per il cache rendering data localmente e usano le scene graph quasi interamente come un'operazione di read-only. Questo permette a molteplici coppie di istruzioni cull-draw di girare su diverse CPU che sono legate a diverse sottosistemi grafici. Il supporto per differenti contesti grafici e il multi threading è pienamente disponibile fuori dal pacchetto OSG tramite l'osgProducer – tutti gli esempi infatti della distribuzione possono essere eseguiti in multi – pipe solo modificando un file di configurazione.

Tutti i sorgenti dell'OSG sono pubblicati sotto l'Open Scene Graph Public License (una versione di LGPL) che permette l'uso, la modifica e la diffusione per entrambi i tipi di progetto open source e no. Il progetto è stato sviluppato inizialmente 4 anni fa da Don Burns, e poi preso in consegna da Robert Osfield che continua a migliorare il progetto tutt'ora. Entrambi i programmatori Robert e Don stanno ora lavorando sull'OSG in termini più professionali per garantire una maggiore consulenza, e completando la scrittura del relativo libro.

Il progetto è al momento in una versione beta, che sta a significare che le principali caratteristiche del sistema scene graph sono presenti, ma si attende una versione release 1.0 nel Febbraio 2005. A dispetto dello stato della versione beta, il progetto ha già guadagnato una certa reputazione come guida open source alla struttura delle scene graph, e si sta affermando come una valida alternativa a librerie commerciali già in uso. Molte compagnie, ricerche

universitarie e appassionati di grafica stanno già utilizzando l'OSG per i loro lavori in tutto il mondo. Esempi di una vasta varietà di applicazioni già sviluppati sulla base dell'OSG sono Blue Marble Viewer, Virtual Terrain Project, Combat Simulator Project, OSG-Edit. Questa è solo una piccola parte dei progetti, ma altri esempi possono essere trovati nel sito ufficiale della libreria.

Capitolo 5

L'informatica nel campo medico

L'informatica ha dato un contributo fondamentale allo sviluppo della medicina. Le tecnologie e le innovazioni usate hanno fornito degli strumenti di indagine clinica completamente nuovi. “3D reconstruction”, “multimodal and image matching” sono soltanto alcuni esempi dei campi di applicazione dell'informatica ai problemi medici.

5.1 Tecnologie informatiche in Medicina

Soltanto descrivere un universo così variegato come quello dell'Informatica e della Medicina è impresa veramente ardua. Non c'è tecnologia informatica o approccio informatico alla risoluzione di problemi, anche apparentemente astratti, che non abbia avuto una ricaduta notevole nel campo medico. L'elenco è senza fine, dall'immagine processing alla grafica virtuale, dall'elaborazione in tempo reale di segnali [E01] al riconoscimento automatico di patologie, dalla ricostruzione di immagini 3D (Figura 5.1) ai problemi di archiviazione di dati ed immagini, dai problemi di sicurezza e riservatezza agli algoritmi di analisi, visualizzazione e compressione di dati fino ai metodi di classificazione e discriminazione, per arrivare all'intelligenza artificiale, ai sistemi di decision supporting system ed al data mining.

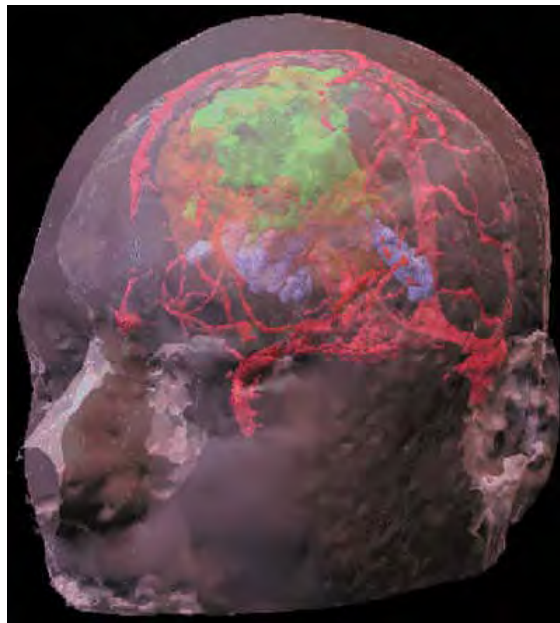


Fig 5.1: esempio di ricostruzione 3D

Molti programmatori finiscono per far parte del grande universo dell'informatica applicata alla medicina. In questo capitolo verranno illustrati i campi di applicazione dell'informatica medica, partendo dalla tipologia dei dati usati e dal loro formato, facendo il punto sulle tecnologie informatiche che vengono

correntemente utilizzate e su alcune promesse dell'intelligenza artificiale (per il momento) mancate, fino a spiegare quali vantaggi ha portato il software di tale tesi.

Mettere insieme una scienza sperimentale come la medicina e le metodiche tecnologie razionali derivanti dall'informatica è stata una delle più grandi rivoluzioni silenziose dei nostri tempi.

5.2 Ruolo dell'informatica nella medicina

La base di partenza di tutti i sistemi informatici applicati al campo medico sono i dati che vengono trattati. I dati o segnali biomedici sono un mezzo per convogliare delle informazioni provenienti da diversi tipi di sorgenti, collegate in modo diretto o indiretto alla fisiologia dei pazienti. Su tali informazioni vi sono poi due livelli distinti di trattamento, uno prettamente di elaborazione e l'altro, a più alto livello, di gestione, comunicazione e memorizzazione. Si possono quindi distinguere due grandi settori di applicazione dell'informatica alla medicina, ovvero:

- 1) Elaborazione;
- 2) Sistemi informativi.

In genere, i settori dediti all'elaborazione trattano i metodi per l'acquisizione, l'analisi e l'interpretazione diretta delle informazioni contenute nei dati biomedici. Questo tipo di analisi viene di norma eseguita immediatamente dopo il momento dell'acquisizione e generalmente nello stesso luogo in cui avviene l'acquisizione.

I settori legati ai sistemi informativi in medicina sono quelli dediti al trattamento, dall'acquisizione al trasferimento fino alla memorizzazione, della grande mole di informazioni che un qualunque reparto, clinica, istituto, ospedale

o ASL (Azienda Sanitaria Locale) produce quotidianamente. Tali informazioni vengono alla fine memorizzate ed utilizzate in fasi successive a quella diagnostica, sia per seguire l'iter terapeutico di un paziente nel tempo che per eseguire delle statistiche o ricerche su gruppi di pazienti con patologie simili.

Per esempio, negli Stati Uniti la quantità di memoria occupata dalle immagini digitali catturate in media ogni anno è dell'ordine di diversi petabyte (2^{50} byte) e, considerando che attualmente soltanto il 30% circa dei sistemi di imaging diagnostico sono di tipo digitale e che questo valore sta aumentando drasticamente, ci si può rendere conto dell'enorme flusso di informazioni che occorre trattare, memorizzare ed ottimizzare. Un altro esempio del volume di dati da trattare proviene dal Visible Human Project della NLM (National Library of Medicine). Questo progetto, partito nel 1986 prevede la creazione di un database completo, anatomicamente dettagliato e tridimensionale del corpo umano, sia maschile che femminile. Lo stato corrente del progetto è il seguente: sono state acquisite da un maschio 1871 sezioni intervallate da 1 millimetro attraverso 3 modalità di imaging:

- risonanza magnetica (MRI);
- tomografia computerizzata (TC);
- criosezioni a colori;

per un totale di 15 Gb di immagini; per la donna sono stati utilizzati gli stessi strumenti ma con sezioni intervallate di 1/3 di millimetro (per un totale di 40 GB di immagini). Inoltre questi dati sono attualmente analizzati e rielaborati in oltre 1000 centri di ricerca medica: si pensi alle problematiche che nascono nella trasmissione e nella memorizzazione di questi 55 Gb di immagini.

5.3 Sistemi informativi in medicina

In linea di principio, i sistemi informativi dei diversi istituti o cliniche presenti in un ospedale, fanno capo al livello più alto ad un sistema ospedaliero (HIS, Hospital Information System) dedito al trattamento dei dati di tutti i pazienti, incluse le informazioni anagrafiche, demografiche, prenotazioni e statistiche. All'interno di ciascun istituto esistono poi dei PACS (Picture Archiving and Communication System), sistemi in grado di interfacciarsi con le diverse strumentazioni diagnostiche e degli ISA (Information Storage and Archive), sistemi in grado di memorizzare le informazioni utili su supporti quali dischi ottici, magneto-ottici o CD-ROM. Quasi sempre le diverse fasi del trattamento del flusso informativo avvengono, tramite reti locali (LAN, Local Area Network) o geografiche (WAN, Wide Area Network), in luoghi diversi. Ad esempio, la visualizzazione e la diagnosi avvengono generalmente in clinica mentre la memorizzazione e l'archiviazione viene eseguita in luoghi a tal scopo preposti.

Uno dei problemi aperti e delicati di molti HIS sono gli standard di comunicazione e la compatibilità fra strumentazione di diversi costruttori. I PACS dovrebbero risolvere il problema del dialogo tra sistemi diversi ma non è difficile comprendere come ogni produttore di sistemi medicali tenda a far utilizzare i propri standard (e, di conseguenza, le proprie macchine). Di fatto il problema dell'interfacciamento e della comunicazione tra diversi sistemi non è stato ancora risolto e l'unico tentativo non interessato è quello dello standard DICOM (Digital Imaging and Communication in Medicine), giunto alla versione 3. DICOM è un protocollo di comunicazione in grado di supportare contemporaneamente diverse tipologie di dati (immagini, filmati, suoni, referti, dati anagrafici ecc.) e si basa sul protocollo di rete TCP/IP [E02] [E03].

La sempre maggiore attenzione ai sistemi di health care, soprattutto dal punto di vista economico e di ottimizzazione delle risorse, sta favorendo la nascita di molte sperimentazioni di telemedicina. Internet, multimedialità e tecnologie

affini sono in questi casi strumenti essenziali per garantire servizi qualitativamente ed economicamente molto appetibili; ma, a differenza di tanti altri campi di applicazione dell'informatica e a discapito dei molti entusiasmi iniziali, in questi settori prima di mettere a regime un sistema medicale integrato con tecnologie di rete avanzate, occorre valutarne la piena affidabilità e sicurezza di tutte le componenti ed in tutte le condizioni operative. Anche per questo motivo in questi settori si preferisce spesso affidarsi alle grosse industrie di elettromedicali (HP, Siemens, GE, Philips ed altri) ai quali si affida la progettazione, la realizzazione e la gestione dei sistemi informativi complessi, comprese le tecnologie informatiche avanzate.

Particolare attenzione va infine dedicata alla riservatezza dei dati immagazzinati: in questo settore, più di altri, la Lg. 675/96 sulla privacy ha imposto delle regole ferree sulla riservatezza dei dati personali di ciascun paziente ed i sistemi informativi sanitari devono poter garantire degli alti livelli di sicurezza.

5.4 Dati, segnali ed immagini biomediche

Nel settore dell'elaborazione le aree di interesse comune fra informatica e medicina sono tante e possono essere divise in diversi modi. In particolare, suddividendo queste aree secondo un criterio legato al tipo di dati che vengono trattati, si possono individuare tre filoni principali:

- 1) elaborazione dati;
- 2) elaborazione biosegnali;
- 3) elaborazione immagini.

Nel primo caso si ha a che fare con tante misure fisiologiche (si pensi per esempio all'insieme dei valori numerici che vengono fuori da molte analisi cliniche), si costruiscono delle matrici di dati, su singoli pazienti o su

popolazioni di pazienti, e quindi si elaborano queste matrici. Gli scopi possono essere diversi, ad esempio classificazione tra soggetti normali e patologici, ricerca di valori fuori norma, creazione di modelli esplicativi, fino alle analisi statistiche più complesse.

Generalmente si applicano dei metodi classici ormai ampiamente testati e per questo si usano pacchetti software affermati tipo SPSS, Statistica, Systat, SAS o simili. Le metodiche di analisi spaziano dalle statistiche di base alle misure di variabilità, dalla regressione alla correlazione, dal confronto fra gruppi ai diversi test statistici, fino alla regressione multipla ed all'analisi multivariata. Oramai su questo punto non si programma più niente, perché praticamente si trova già tutto fatto. Esistono dei centri di ricerca dove si continuano a creare applicazioni specifiche per questo tipo di analisi, ma sono delle realtà isolate rispetto ai tanti centri pubblici e privati in cui si fa elaborazione dati di tipo biomedico.

Diverso è il discorso sui punti 2) e 3), nei quali l'attività di programmazione e di ricerca è molto intensa. In particolare, nel secondo caso si analizzano direttamente i segnali fisiologici acquisiti nel loro andamento temporale, tipo ECG (elettrocardiogramma), EEG (elettroencefalogramma), pressione, temperatura ecc.. In questo settore mediamente bisogna programarsi tutto, dalla progettazione allo sviluppo di software, dalla realizzazione dei moduli di raccolta, all'analisi, alla visualizzazione ed archiviazione in rete locale (LAN) dei segnali fisiologici. I linguaggi più usati sono: C, C++, Visual Basic, ambienti integrati come Matlab, Mathematica e non è raro trovare gruppi di programmazione in Fortran. Le tecniche di biosignal processing più in voga ed utilizzate sono: varie tecniche di filtraggio, rivelazione e discriminazione di eventi, analisi dello spettro in frequenza dei segnali, analisi frattale, wavelet, logica fuzzy, reti neurali e metodi della teoria del caos.

Di norma, tutti i sistemi hardware biomedicali vengono forniti del necessario software di supporto per la gestione della macchina e di software base per l'elaborazione standard dei segnali acquisiti. Ma, per diversi motivi, la tipologia

di analisi che si può effettuare non è esaustiva, soprattutto per gli specialisti del settore.

Ad esempio un buon elettrocardiografo è fornito anche dei supporti necessari per interfacciarsi con un PC e del software base per lo studio della variabilità cardiaca e delle relative patologie. Per compiere delle analisi più sofisticate occorre acquisire il segnale ECG ed analizzarlo con applicativi scritti appositamente. Esistono comunque casi particolari di società di elettromedicali che, attraverso l'interazione dei propri team di programmatori e di collaudati centri di ricerca, sviluppano algoritmi di analisi innovativi che prontamente vengono messi a disposizione sulla propria strumentazione, minimizzando i tempi di implementazione e testing e soddisfacendo anche gli specialisti più esigenti.

Nel caso della elaborazione di immagini, a seconda della tipologia di sorgenti usate e del loro utilizzo clinico, si applicano diverse metodiche di analisi [E04]. Le immagini ed i loro formati sono molteplici quali: ecografiche, color e laser doppler, TC, TAC, risonanza magnetica, risonanza funzionale, PET ecc.

Anche gli algoritmi che si usano sono i più diversi: riconoscimento di contorni, valutazione delle dimensioni di oggetti all'interno di una immagine, videodensitometria, analisi dei livelli di grigio, matching tra immagini diverse per aumentare le possibilità di visione, ricostruzione 2D, 3D e 4D (usando anche la coordinata temporale), ecc. Ovviamente il tipo di elaborazione dipende direttamente dal tipo di sorgente dei dati e dal trattamento che questi subiscono a monte; una tipica sequenza di trattamento è: *sorgente - segnale - acquisizione - filtraggio - memorizzazione - elaborazione - visualizzazione*. Per esempio nei sistemi per l'imaging medico quali la TAC, l'informazione sulla struttura dell'organo in esame viene prima convogliata su una successione di onde elettromagnetiche, le quali sono successivamente campionate per generare una matrice di numeri. Tali numeri sono prima filtrati e poi elaborati ed i risultati sono presentati su uno schermo sotto forma di immagine, oppure possono essere

archiviati per usi successivi. In ognuno di questi passi la ricerca informatica propone continuamente strumenti e metodiche nuove che, per poter essere utilizzate, ogni centro deve implementare di volta in volta. Esistono comunque pacchetti software specifici per campi molto ristretti, ma non fanno mai quello che serve, soprattutto per la continua evoluzione delle tecniche da applicare.

5.5 Classificazione dimensionale

Possono essere adottati diversi criteri di suddivisione di dati, segnali ed immagini biomediche. Il criterio più utile dal punto di vista informatico è quello della dimensionalità della misura, riportata nella Tabella 5.1. Nel caso più semplice il dato può essere monodimensionale singolo (1-D singolo, un singolo valore numerico). La misura può arricchirsi di più valori (1-D vettore), come nella valutazione minima e massima della pressione del sangue, audiometria, ecc.

	STATICO	DINAMICO
1-D SINGOLO	Esami biochimici, temperatura, ecc.	EMG, curva glicemica, dosaggi radioimmunologici, fonoca- rdiogramma, potenziali d'azione, ecc.
1-D VETTORE	Pressione sangue, audiometria, acuità visiva, ecc.	ECG, EEG, ecc.
2-D	Radiografia, scintigrafia, termografia, ecografia statica, TAC statica, RM	Scintigrafia dinamica, elettromappe cardiache, ecocardiografia
3-D	Stereoradiografia, ecografia spaziale, TAC spaziale, RM spaziale, PET	Cineangiografia, RM tipo 'echo planar', TAC ultrarapida.

Tabella 5.1: classificazione dimensionale dei dati medici

Quando i dati sono associati in modo diretto alla dimensione temporale, cioè le misure precedenti vengono eseguite nel tempo, si passa da valori statici a valori

dinamici o tracciati; ad es. nel caso dell'ECG o EEG ci sono più tracciati acquisiti contemporaneamente. Si possono poi avere mappe bidimensionali e tridimensionali. In queste situazioni si passa da un 2-D semplice (come nel caso di una lastra radiografica) ad un 2-D dinamico (come nel caso delle ecocardiografie). Infine il dato tridimensionale è costituito dalla identificazione spaziale di volumi ottenuti mediante tecniche di imaging quali la Tomografia ad Emissione di Positroni (PET), la Tomografia Assiale Computerizzata (TAC), la risonanza magnetica (RM), l'ultrasonografia 3-D. Con l'introduzione sul mercato di sistemi di imaging veloci (TAC, RM) in cui la risoluzione temporale è sufficiente per ricostruire organi in movimento quali il cuore, è possibile la rappresentazione 3-D dinamica.

5.6 Eidologia informatica

La rappresentazione del corpo umano e delle sue funzioni sotto forma di immagine rappresenta un importante strumento di indagine diagnostica. L'eidologia informatica in medicina (*eidos* significa immagine), avvalendosi delle tecniche di acquisizione, trattamento e presentazione di strutture biologiche tramite ultrasuoni, medicina nucleare, risonanza magnetica e tomografia a raggi X, ha fatto in modo che le immagini risultanti da queste tecniche abbiano caratteristiche simili a prescindere dal principio fisico utilizzato.

Le tecniche di ricostruzione di immagini 3D (come per esempio mostra la figura 5.1) permettono poi di creare viste precise e realistiche degli organi in modo automatico ed obiettivo, fungendo da valido (ed ormai insostituibile) supporto tecnologico alla diagnosi medica. Inoltre è possibile produrre, attraverso tecniche di *multimodality imaging*, immagini integrate sia di informazioni morfologiche che funzionali, provenienti da diverse modalità di imaging, diversi supporti di visualizzazione e spesso anche da diversi centri diagnostici.

Le tecniche di eidologia informatica in medicina possono concettualmente essere suddivise in tre aree fondamentali:

- elaborazione a basso livello;
- elaborazione a livello intermedio;
- elaborazione ad alto livello.

Secondo tale schema l'immagine viene considerata non solo per il processo che è alla base della sua visualizzazione, ma anche per quei processi e conoscenze che sono presenti nella mente e nell'esperienza di colui che effettua l'elaborazione delle immagini.

L'elaborazione a basso livello non richiede alcuna forma di intelligenza da parte del sistema che realizza l'elaborazione. Esso comprende le attività di formazione dell'immagine in forma digitale, la riduzione del rumore, il filtraggio, ecc. L'elaborazione a livello intermedio comporta l'estrazione e la regionalizzazione delle componenti utili dell'immagine e comprende le operazioni di segmentazione, rappresentazione e descrizione. In questo livello si richiede una qualche forma di procedura intelligente per realizzare l'operazione in modo efficace. Il terzo livello di elaborazione comprende le operazioni di riconoscimento e interpretazione, le quali sono basate sulla conoscenza pregressa dei componenti dell'immagine e del loro significato.

A questo livello, troviamo i sistemi CAD: questa metodologia ha rivoluzionato le tecniche classiche sul campo dell'analisi tomografica. I vantaggi del CAD sono chiaramente innumerevoli, basti pensare alla possibilità di modificare esami radiografici in seguito al prelievo delle informazioni in forma digitale. Questo permette di applicare fasi di image processing, quali filtraggi e aumenti del contrasto e luminosità, per migliorare l'immagine, in maniera assolutamente automatica. Un altro beneficio apportato è che, idealmente, non ci sono vincoli alla risoluzione spaziale dell'immagine, per cui in generale si ha una qualità

diagnostica nettamente migliore. Tuttavia le tecniche di elaborazione delle radiografie richiedono un notevole costo computazionale, sia a causa dell'alta risoluzione delle immagini trattate, che per la complessità degli algoritmi utilizzati. Per questo motivo si impiegano calcolatori multiprocessore molto potenti e tecniche di parallelizzazione avanzata [E05] [E06] [E07].

Capitolo 6

Implementazione del software

In questo capitolo la trattazione verte esclusivamente sull'implementazione del software di tesi. Il progetto è stato realizzato su un sistema operativo Microsoft Windows XP, ed è stato sviluppato e compilato con Microsoft Visual Studio .NET 2003.

Come è già stato detto, il programma ha la possibilità di girare su altri sistemi operativi per cui il capitolo seguente spiega parti di codice sorgente indipendentemente dal compilatore che genererà l'eseguibile e soprattutto dalla piattaforma sulla quale sarà avviato.

Solamente nel primo paragrafo verrà in dettaglio mostrato il settaggio per la compilazione con Visual Studio, dato che il progetto è stato sviluppato con tale software.

6.1 Setup di Microsoft Visual Studio .Net 2003

Per poter compilare correttamente il progetto di tesi vanno considerati alcuni aspetti:

- *Inclusione delle librerie*: al momento della compilazione il programma deve sapere quali librerie utilizzare e dove cercarle;
- *Compatibilità tra di esse*: a partire da Windows 95, la Microsoft dispone del supporto multi-thread, ossia l'accesso comune ai dati da parte di varie applicazioni. Tale sistema viene chiamato sistema OLE (*Object Linking and Embedding*, ossia collegamento e impressione di oggetti);

Per rendere possibile ciò, Visual Studio .NET 2003²⁸ dispone di un'interfaccia semplice e pratica.

Per quanto riguarda il primo punto, è necessario inserire le cartelle contenenti le librerie nel menu “Tools” alla voce “Options..”.

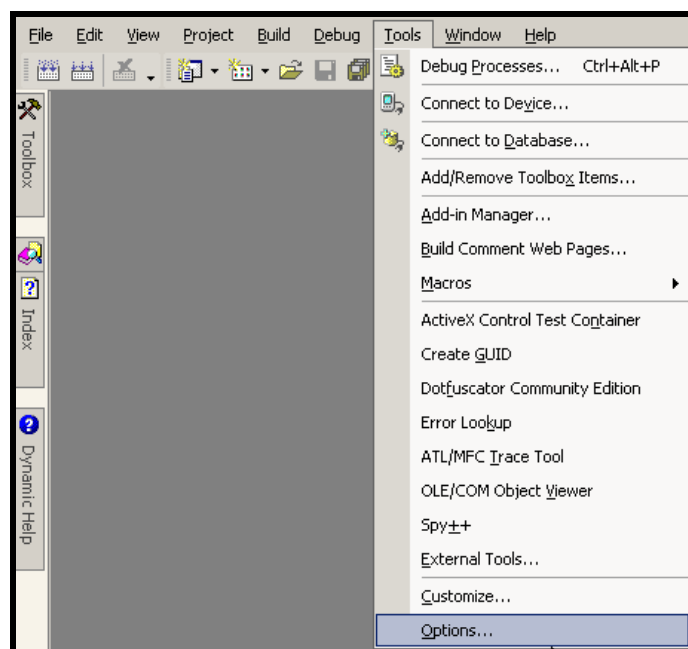


Figura 6.1: settaggio delle librerie incluse

²⁸ Al momento della stesura, Microsoft ha rilasciato la versione beta di .NET 2005.

Una volta fatto ciò, basta inserire nella sezione “VC++ Directories” della cartella “Projects” le directory che ospitano le librerie in base a:

- I. La posizione dei file di inclusione da inserire nella categoria “Include Files”, come segue nella figura;

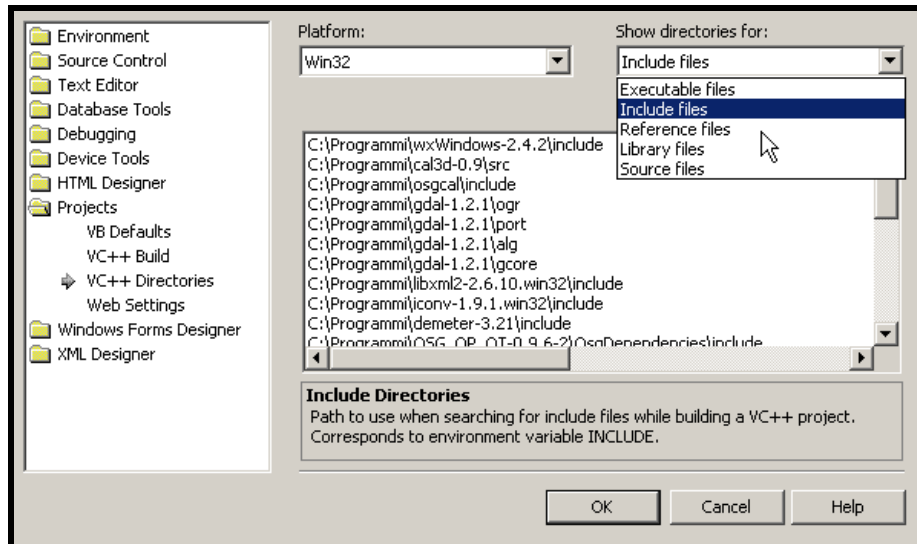


Figura 6.2: inclusione dei file

- II. La posizione dei file “.lib” che sono le librerie vere e proprie da inserire nella opzione “Library Files”.

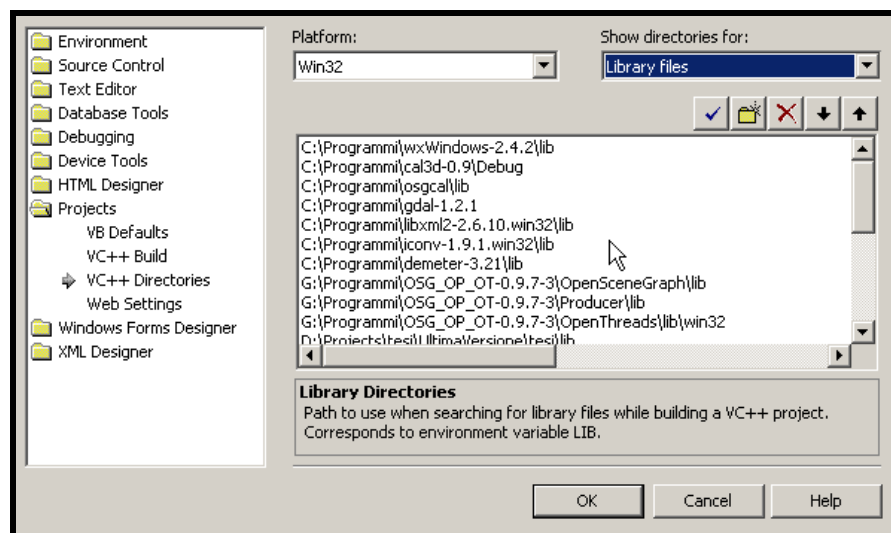


Figura 6.3: inclusione delle directory delle librerie

La terza operazione da compiere è l'inclusione delle stesse librerie sopra citate: è necessario digitare la lista di quelle che dovranno essere presenti all'esecuzione del programma. Per questo si deve accedere al menu principale "Projects" e cliccare sulla voce "<nome progetto> Properties ..".

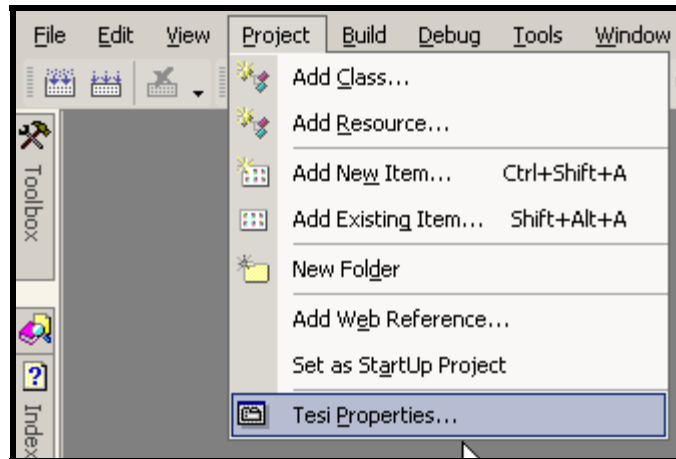


Figura 6.4: inserimento delle librerie a livello dinamico

Nella finestra che si apre è sufficiente spostarsi nella cartella Linker e scrivere correttamente le librerie necessarie per l'esecuzione del progetto.

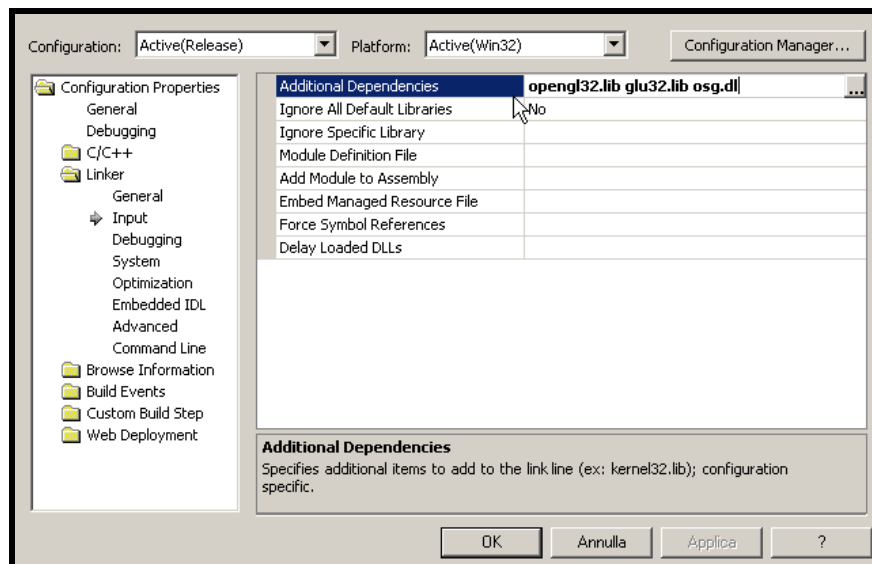


Figura 6.5: dipendenze del progetto dalle librerie dinamiche

A questo punto resta da trattare il problema del multi-thread che viene risolto da MVS all'interno dell'ultima finestra aperta; basta entrare nella cartella "C/C++" e selezionare "Multi-Threaded Dll (/Md)" alla voce "Code Generation".

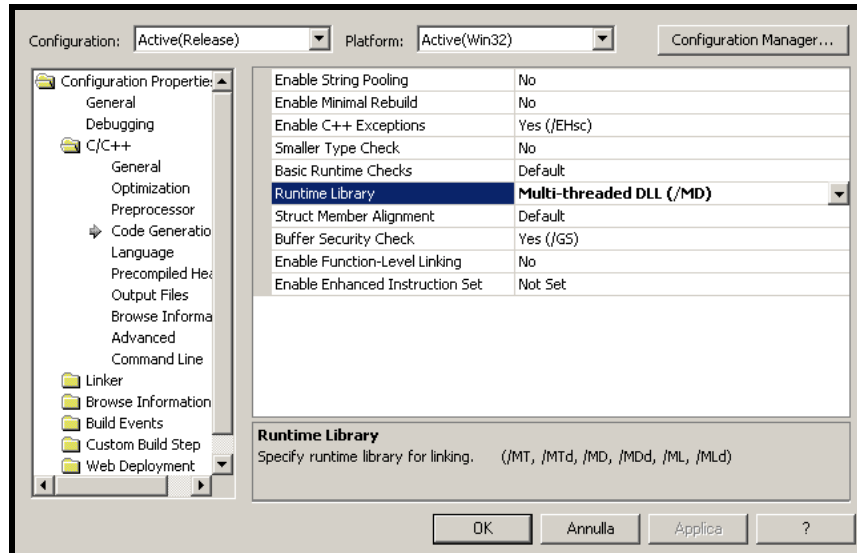


Figura 6.6: supporto per il multi-thread

Infine per quanto riguarda l'esecuzione del programma, è sottointeso che dovranno essere reperibili a livello di path, oppure nella stessa directory dell'eseguibile, le librerie stesse a livello dinamico, cioè le Dll (Dinamic Link Library).

6.1.1 Librerie utilizzate

E' ora il momento di specificare quali librerie inserire nelle impostazioni; finora è stato detto dove andare a modificare i parametri, ma non è stato fatto un riferimento esplicito su quali file aggiungere.

Di seguito è riportata la lista dei pacchetti ottenibili tutti gratuitamente da Internet:

- 1) SDL 1.2.7;
- 2) SDL_Gfx 2.0.8;
- 3) SDL_Image 1.2.3;
- 4) SDL_Console 2.1;

- 5) OpenSceneGraph 0.9.8 (che contiene al suo interno, come già detto, OpenThreads e Producer);

Ogni pacchetto va compilato singolarmente e ognuno di essi genera principalmente le librerie vere e proprie con estensione “.lib” (a livello statico) e “.dll” (a livello dinamico).

Inoltre, per quanto riguarda l’inserimento nella parte del “<nome progetto> Properties ..”, sezione “Linker”, vanno aggiunte le librerie dell’OpenGL, che sono già in dotazione col sistema operativo Windows XP.

In conclusione la lista delle librerie sarà:

- 1) opengl32.lib
- 2) glu32.lib
- 3) glut32.lib
- 4) sdl.lib
- 5) sdlmain.lib
- 6) sdl_gfx.lib
- 7) sdl_image.lib
- 8) sdl_console.lib
- 9) OpenThreadsWin32.lib
- 10) Producer.lib
- 11) osg.lib
- 12) osgdb.lib
- 13) osgutil.lib
- 14) osgFX.lib

E’ ora possibile cominciare a trattare il codice generato. Gli argomenti verranno suddivisi in base all’ordine col quale si è arrivati al risultato finale, e cioè:

- 1) Creazione di un contesto grafico con SDL;
- 2) Settaggi per inserire un contesto grafico OpenGL su di una finestra creata da SDL;

- 3) Inserimento di una console;
- 4) Realizzazione dell'interfaccia e caricamento di un'immagine;
- 5) Combinazione di ogni elemento ottenuto: console, immagine in 2D e visuale 3D;
- 6) Introduzione dell'OSG su contesto di SDL: impostazioni;
- 7) Inserimento dell'OSG sull'interfaccia precedentemente creata;
- 8) Creazione di una nuova interfaccia con una nuova gestione delle console;
- 9) Creazione della versione finale con la possibilità di avviare l'applicazione in ogni modalità video a finestra e in fullscreen.

6.2 Creazione di un contesto grafico con SDL

La libreria grafica SDL, già vista nel Capitolo 2, permette di creare una finestra con pochissime righe di codice:

```
#include <stdio.h>
#include <stdlib.h>
#include <SDL.h>

int main(int argc, char *argv[])
{
    int done=0;
    SDL_Surface *screen;
    if ( SDL_Init(SDL_INIT_AUDIO|SDL_INIT_VIDEO) < 0 ) {
        printf("Unable to init SDL: %s\n", SDL_GetError());
        exit(1); }
    atexit(SDL_Quit);
    screen=SDL_SetVideoMode(640,480,32,SDL_HWSURFACE|SDL_DOUBLEBUF);
    while(done == 0) {
        SDL_Event event;
        while ( SDL_PollEvent(&event) ) {
            if ( event.type == SDL_KEYDOWN ) {
                if ( event.key.keysym.sym == SDLK_ESCAPE ) {
                    done = 1; }
            }
        }
    }
    return 0;
}
```

Codice 6.1: Inizializzazione SDL

Tale scritto genera una finestra gestita totalmente da SDL: con questo si intende che, per esempio, non ha più effetto cliccare sull'icona "x" per chiudere la

window, ma, come recita il codice, è necessario digitare il tasto “ESC” per uscire.

A questo punto è stata generata una finestra nella quale è possibile inserire qualsiasi oggetto SDL o compatibile con esso: come prima verifica verrà inserita una visuale tridimensionale generata con OpenGL.

6.3 Impostazioni per creare un contesto OpenGL

Come già discusso nel Capitolo 2, l’SDL ha la possibilità di creare una `SDL_Surface` che supporti le funzioni OpenGL. Per poter disegnare qualsiasi cosa è necessaria un’impostazione standard dopo la quale è possibile cominciare a scrivere liberamente codice OpenGL.

In base al codice generato possiamo aggiungere una funzione di questo tipo:

```
void InitMy3D(SDL_Surface* MainSur)
{
    SDL_GL_SetAttribute(SDL_GL_RED_SIZE, 5);
    SDL_GL_SetAttribute(SDL_GL_GREEN_SIZE, 6);
    SDL_GL_SetAttribute(SDL_GL_BLUE_SIZE, 5);
    SDL_GL_SetAttribute(SDL_GL_DEPTH_SIZE, 16);
    SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glViewport(0, 0, 800, 600);
    glFrustum(-0.5f, 0.5f, -0.5f, 0.5f, 0.65f, MAX_VIEW_DISTANCE);
    glClearColor(0.5f, 0.75f, 1.0f, 0.0f);
    glDisable(GL_NORMALIZE);
    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_SMOOTH);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
}
```

Codice 6.2: Inizializzazione OpenGL

Questi parametri contribuiscono a far comunicare SDL con OpenGL: dato che la finestra è stata generata da SDL, questi deve dar accesso a OpenGL alle risorse grafiche del computer, come per esempio la definizione di quanti bit siano assegnati ai 3 colori base, cioè rosso, verde e blu. In un secondo momento possiamo cominciare ad abilitare la ViewPort e altri parametri già visti nel Capitolo 3.

Il risultato dato dalla chiamata di tale funzione all'interno del main prima generato inizializza un contesto OpenGL entro il quale è possibile cominciare a lavorare con il mondo 3D.

Per vedere all'interno della finestra un visione tridimensionale è necessario prima di tutto dar vita a una camera che gestisca la visuale, inoltre è basilare che all'interno del main tale camera sia aggiornata dei cambiamenti sia del paesaggio sia del punto di vista. Tale variabile è stata creata in appunto file “camera.cpp” che gestisce ogni movimento della camera e il suo aggiornamento.

Come esempio, è stato utilizzata un'ulteriore libreria grafica, Demeter, atta creazione di vasti terreni, con la quale è stata generata un piccolo terreno per vedere i risultati dell'aggiunta di un contesto OpenGL all'interno di una contesto 2D generato con SDL.

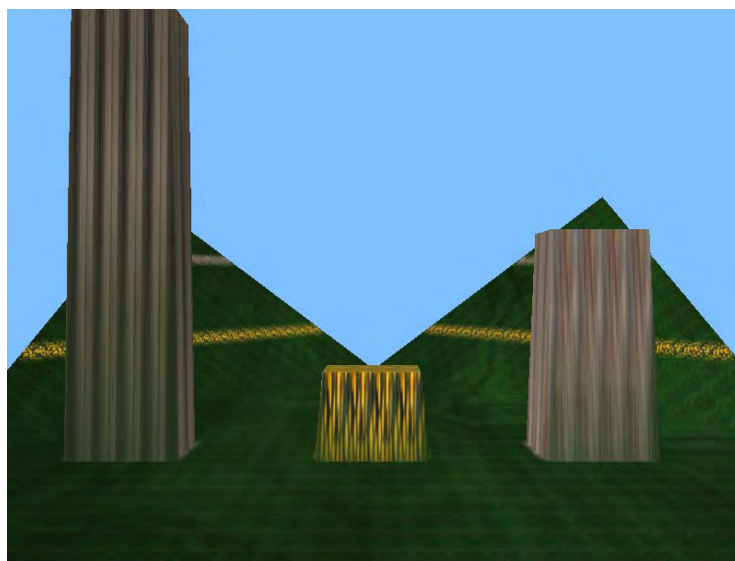


Figura 6.7: Esempio di OpenGL su un contesto SDL

La scelta di inserire il 3D su tutta la finestra non è obbligata: come già discusso nel Capitolo 3, la funzione ViewPort mi permette di impostare i parametri del contesto OpenGL.



Figura 6.8: la visuale 3D occupa solo una parte della finestra

6.4 Inserimento di una console

Per poter modificare i dati della visuale 3D senza dover in ogni momento cambiare il codice e ricompilarlo ogni volta, è stata inserita una console come aiuto all'utente. La console generata proviene da una libreria già collaudata, che permette di processare le informazioni a runtime come la più nota console Microsoft MS-DOS, il primo sistema operativo creato.

Tale console proviene anch'essa da una libreria, per l'appunto `SDL_Console` 1.2. Il pacchetto che la comprende dà alcuni esempi di utilizzo, tramite i quali è stato possibile arrivare ad inserire questo tool all'interno del progetto.

Infatti l'inizializzazione avviene nel main (esempio nel codice 6.1), ma le funzioni chiamate provengono dal file `"console.cpp"` che gestisce ogni operazione valida della console.

Per generare tale tool vengono di base impostate delle variabili riguardanti la posizione di destinazione e le dimensioni, per cui il codice è del tipo:

```

Con_rect.x = 10;
Con_rect.y = 0;
Con_rect.w = 240;
Con_rect.h = 120;

if((Consoles[0] = CON_Init("ConsoleFont.bmp", pScreen, 100,
Con_rect)) == NULL) return 1;
CON_SetExecuteFunction(Consoles[0], Command_Handler);
ListCommands(Consoles[0]);
CON_Show(Consoles[0]);

```

Codice 6.3: creazione di una console

La variabile `Con_rect` è propria dell'SDL e dà informazioni sulla posizione di creazione a livello di coordinate `x` e `y` con, rispettivamente, `Con_Rect.x` e `Con_Rect.y`, ed inoltre le sue dimensioni, `Con_Rect.w` relativo all'ampiezza e `Con_Rect.h` sull'altezza. Le coordinate a cui si fa riferimento sono relative al punto (0,0) rappresentato dal vertice in basso a sinistra della finestra creata.

L'introduzione di tale variabile, `SDL_Rect`, è fondamentale per gestire ogni parte dell'interfaccia: più avanti saranno viste le sue molteplici applicazioni. A questo punto possiamo inserire la console all'interno della finestra precedentemente generata e ottenere:

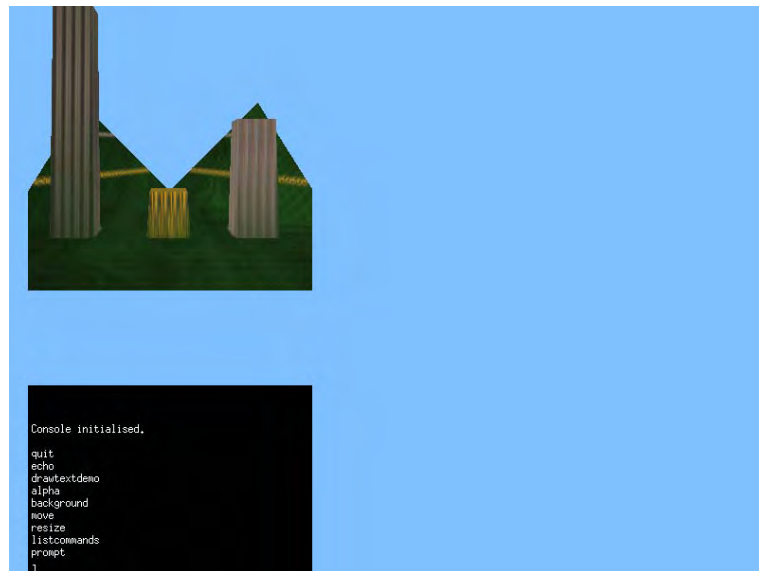


Figura 6.9: visuale 3D e console

6.5 Realizzazione dell'interfaccia e caricamento di un'immagine

E' così diventato necessario inserire gli elementi sopra creati all'interno di un'interfaccia grafica. La creazione dell'interfaccia è un dovere prettamente dell'SDL, che dispone di funzioni di grafica bidimensionale.

A tale scopo è stata utilizzata la struttura dati `SDL_Rect` che permette di definire una porzione rettangolare della finestra da poter colorare o nella quale inserire oggetti (come una console).

Diventa così facilmente realizzabile un disegno organico per dividere la window in zone diverse dove inserire immagini, visuali 3D, console, etc..

La possibilità di disegnare qualcosa è data dalla sovrapposizione di una `SDL_Surface` sopra quella principale, generata nel codice 6.1; l'esempio qui riportato mostra come settare la variabile per generare l'interfaccia nella figura 6.10, dove alla funzione `InitMySurface` è passata la variabile `screen`:

```
SDL_Surface* InitMyInterface(SDL_Surface* MainSur)
{
    int SVF = SDL_HWSURFACE | SDL_DOUBLEBUF;
    int width = 800, height = 600, depth = 32;
    SDL_Rect rect;
    SDL_Surface* MySurface=NULL;
    SDL_Surface* Temp=NULL;

    rect.w=800;
    rect.h=600;
    rect.x=0;
    rect.y=0;

    Temp = SDL_CreateRGBSurface(SVF, rect.w, rect.h,
    MainSur->format->BitsPerPixel, 0, 0, 0, 0);

    if(Temp == NULL) {
        fprintf(stderr, "Non posso ionizializzare la Surface Temp!!\n");
        return NULL;
    }
    MySurface = SDL_DisplayFormat(Temp);
    SDL_FreeSurface(Temp);

    return (MySurface);
}
```

Codice 6.4: creazione di un'interfaccia

Queste considerazioni portano all'introduzione del prossimo passaggio per la realizzazione del software: inserire un'immagine. Gli argomenti relativi a `SDL_Rect` e `SDL_Surface` ora visti tornano molto utili: aggiungere una figura al nostro contesto grafico non significa altro che ripetere le operazioni esaminate. A rendere più immediata la manipolazione di qualsiasi immagine (in termini di formato, e cioè jpg, tiff, bmp, etc..) è la libreria `SDL_Image` atta proprio allo scopo da raggiungere. Le operazioni da compiere sono perciò:

- 1) inizializzare una variabile `SDL_Surface`, `MySurface`;
- 2) caricare l'immagine su di una variabile temporanea;
- 3) adattare in base alle misure volute la variabile temporanea alla variabile `MySurface`.

Questo si traduce, in termini di codice, con le istruzioni:

```
SDL_Surface* InitMySprite(SDL_Surface* MainSur,int zoom)
{
    int SVF = SDL_HWSURFACE | SDL_DOUBLEBUF;
    int width = 800, height = 600, depth = 32;
    SDL_Rect rect;
    SDL_Surface* MySurface=NULL;
    SDL_Surface* Temp=NULL;

    rect.w=800;
    rect.h=600;
    rect.x=0;
    rect.y=0;

    int    width_layer=400;
    int    height_leyer=400;

    Temp = SDL_CreateRGBSurface(SVF, rect.w, rect.h,
    MainSur->format->BitsPerPixel, 0, 0, 0, 0);
    Temp=SDL_LoadBMP(background);
    Temp = zoomSurface(Temp,(width_layer/(float)
    Temp->w)/zoom,(height_leyer/(float)Temp->h)/zoom,0);

    if(Temp == NULL) {
        fprintf(stderr,"Non posso inizializzare la Surface!!\n");
        return NULL; }

    MySurface = SDL_DisplayFormat(Temp);
    SDL_FreeSurface(Temp);

    return (MySurface);
}
```

Codice 6.5: caricamento di un'immagine

La funzione `SDL_CreateRGBSurface` crea una `Surface` simile a quella principale; la seconda operazione è l'immagazzinamento della figura, e la terza è il ridimensionamento. Una volta sicuri che ogni istruzione è andata a buon fine, viene ottimizzata con l'istruzione `SDL_DisplayFormat` e immagazzinata nella variabile che verrà poi utilizzata nell'interfaccia per la visualizzazione dell'immagine.

Una volta caricata, è possibile inserirla nel contesto SDL, insieme alle console, e avere:

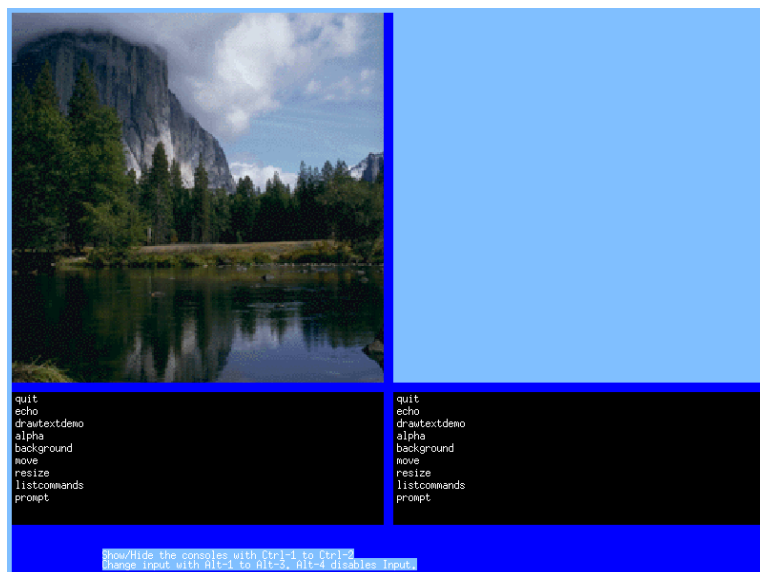


Figura 6.10: interfaccia grafica con immagine caricata

6.6 Unione degli elementi creati

Una volta disegnata l'interfaccia e caricata l'immagine 2D, posso aggiungere finalmente la mia visuale 3D per completare l'interfaccia.

A questo livello è importante preoccuparsi del codice interno alla procedura `while` dove vengono ogni secondo aggiornate tutte le singole parti dell'interfaccia. E' innanzitutto necessario fare l'update del disegno dell'interfaccia, poi l'immagine e infine la parte 3D.

In base al codice finora scritto, il risultato sarà:

```
while(!done) {
ProcessEvents();
DrawInterface();
DrawSprite();
Draw3D(); }
```

Codice 6.6: ciclo while per il refresh

La prima funzione è necessaria per processare l'input da tastiera e da mouse; le funzioni elencate sono le stesse presenti nel codice della versione finale del progetto, con un'eccezione per la parte 3D sostituita dall'OSG.

L'interfaccia risultante dal codice finora analizzata sarà:

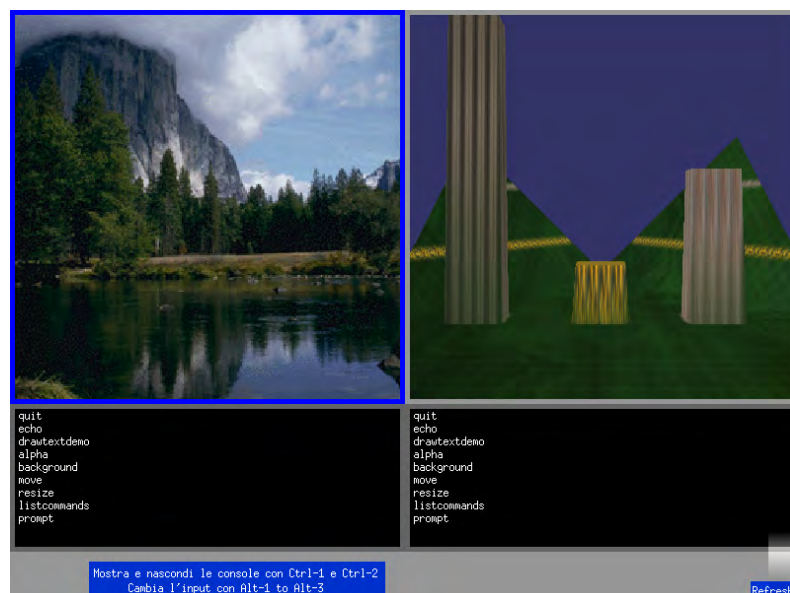


Figura 6.11: interfaccia grafica con l'immagine caricata e la visuale 3D

A questo punto è sorto il problema di poter visualizzare la scena tridimensionale da due punti di vista diversi, ossia generare un'altra variabile “camera” che si proietti sullo stesso mondo virtuale.

In base alle considerazioni fatte, per raggiungere l'obiettivo sarebbe bastato ripetere le istruzioni OpenGL viste in precedenza (Codice 6.2) per creare un'altra viewport: il problema di base è che l'OpenGL, una volta inizializzata la prima viewport, non permette di gestirne un'altra indipendente.

Nel caso che venga ripetuto il codice già scritto con altri parametri posizionali, viene solamente spostata la visuale 3D già creata.

Il problema ha occupato una buona parte del lavoro totale, ma è stato risolto con l'introduzione della libreria grafica analizzata nel Capitolo 4, cioè con l'Open Scene Graph.

6.7 Inserimento dell'Open Scene Graph

L'introduzione di tale libreria ha spostato il lavoro al punto iniziale: creare cioè una finestra e inserire qui dentro una visuale 3D costruita sulla base della teoria delle scene graph.

La libreria OSG è ottenibile, come già detto nel capitolo precedente, gratuitamente in un pacchetto che comprende pure l'OpenThreads e l'OpenProducer. Quest'ultima libreria è utilizzata per visualizzare degli esempi contenuti, nel senso che ogni scena e tool applicato su essa viene renderizzato su di una finestra inizializzata con il Producer.

Il compito da svolgere è perciò rendere possibile lo stesso procedimento tramite però l'SDL. A questo proposito è stata creata una libreria, sempre disponibile gratuitamente in rete, chiamata "osgSDL"; questa libreria rende compatibili le operazioni di inizializzazione di scena da parte dell'OSG con le operazioni di visualizzazione da parte dell'SDL.

La caratteristica dell'OSG è senz'altro la struttura ad albero: all'interno di un nodo principale è possibile inserire varie variabili; tra queste anche una immagine 3D realizzata con la libreria (presa d'esempio) Demeter può rappresentare un nodo, ed essere così aggiunta come qualsiasi altro oggetto.

L'inizializzazione dell'OSG avviene tramite una variabile SceneView che contiene le informazioni sugli oggetti presenti nella scena. Il codice seguente genera ciò che è rappresentato in figura 6.12:

```

void InitMyOSG3D(Terrain* MyT,SDL_Surface* MainSur)
{
char szMediaPath[20]="data\\";
SceneView *pView = new SceneView();

int pViewP_W=390;
int pViewP_H=390;

pView->setDefaults();
pView->setViewport(5, 185, pViewP_W, pViewP_H);

Settings::GetInstance()->SetScreenWidth(pViewP_W);
Settings::GetInstance()->SetScreenHeight(pViewP_H);

pView-
>setComputeNearFarMode(CullVisitor::DO_NOT_COMPUTE_NEAR_FAR);
Group *pRootNode = new Group;
Node *pTerrainNode = CreateTerrainNode(MyT);
pRootNode->addChild(pTerrainNode);
Group *aTree= CreateTree(MyT);
pRootNode->addChild(aTree);
pView->setSceneData(pRootNode);
}

```

Codice 6.7: esempio di creazione di una scena

Per rendere più marcato l’inserimento della struttura della scene graph è stato aggiunto un albero all’immagine 3D finora creata con la funzione `CreateTree`. La variabile `pRootNode` è il contenitore degli oggetti presenti: l’ultima istruzione serve proprio ad impostare i dati della scena.

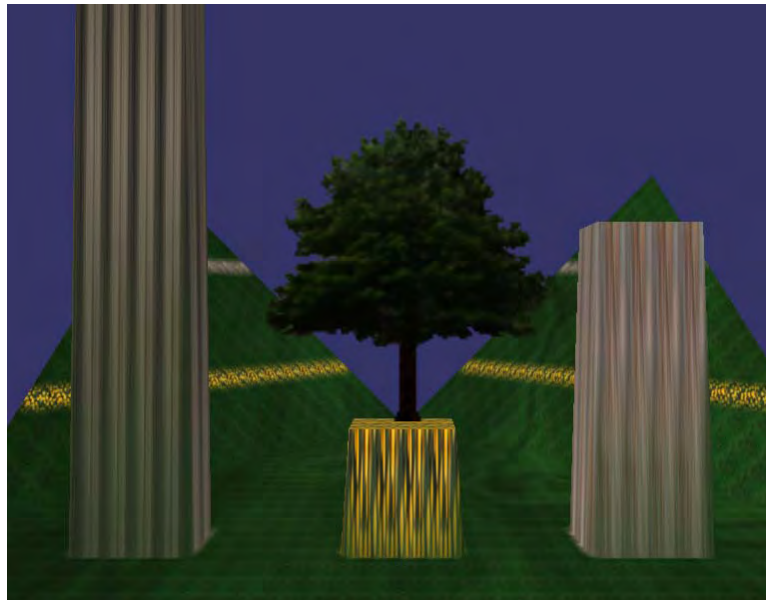


Figura 6.12: creazione di una scena 3D con OSG integrando altre librerie

6.8 GUI integrata con 2D e 3D

E' stata creata una nuova interfaccia che comprende tutte le componenti:

- 1) inizializzazione del contesto grafico SDL;
- 2) inserimento di due console che gestiscano i contesti grafici;
- 3) caricamento di un'immagine 2D;
- 4) caricamento di una visuale tridimensionale con OSG.

Inoltre è stata ottimamente realizzata la possibilità di gestire due indipendenti camere che seguono la stessa scena oppure due scene differenti (esempio in figura 6.13).



Figura 6.13: nuova interfaccia con due camere su differenti scene; inoltre con lo spazio per lo zoom sull'immagine

La funzione che permette lo zoom sull'immagine sfrutta parametri e procedure dell'SDL stesso e cattura una zona dell'interfaccia in cui ho figura 2D visualizzandola nell'angolo in basso a sinistra.

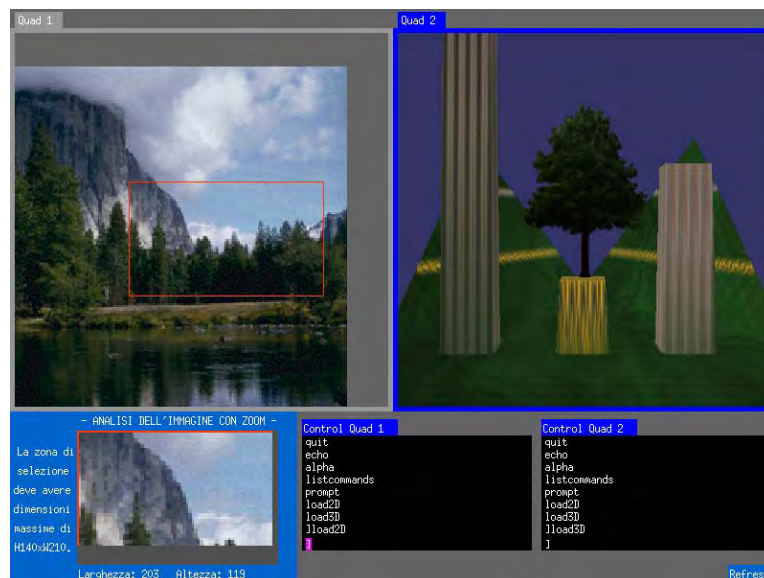


Figura 6.14: nuova interfaccia con lo zoom sull'immagine e il 3D

6.9 Applicazione di dati medici

Il caricamento di un'immagine, lo zoom su di essa e il visualizzatore 3D sono stati creati: a questo punto è il momento di aggiungere i dati medici.

L'ultima versione implementata è avviabile in molteplici risoluzioni : 640x480, 800x600, 1024x768, etc.. adattando la suddivisione dei vari tool negli spazi appositi in base alla grandezza della finestra.

Inoltre quest'ultima applicazione ha un'impostazione grafica molto più semplicistica, dato che è opportuno che l'elaborazione computazionale si rivolga più al caricamento di ingenti quantità di informazioni piuttosto che al caricamento di un'interfaccia più curata esteticamente.

Gli esempi seguenti mostrano come le funzioni finora implementate non rappresentino solamente un lavoro interessante per conoscere il mondo della grafica 3D, ma che all'interno di esso vi siano le applicazioni più disparate che si traducano per l'uomo come strumenti per la sua evoluzione in campo medico.

Nella parte del visualizzatore tridimensionale è stato caricato un modello volumetrico 3D di un polmone (Figura 6.15).

Nella zona in alto a destra è lasciato uno spazio vuoto per il caricamento di un'immagine.

Lo scopo di questo progetto è quello di poter sezionare il polmone raffigurato in 3D e caricare, in base alla posizione del piano di taglio, la corrispondente immagine in 2D (slices) proveniente da un esame radiografico. Questa realizzazione porta ad un'analisi rapida e funzionale con strumenti visivi di ottima qualità per un esame ulteriormente accurato dal punto di vista medico.

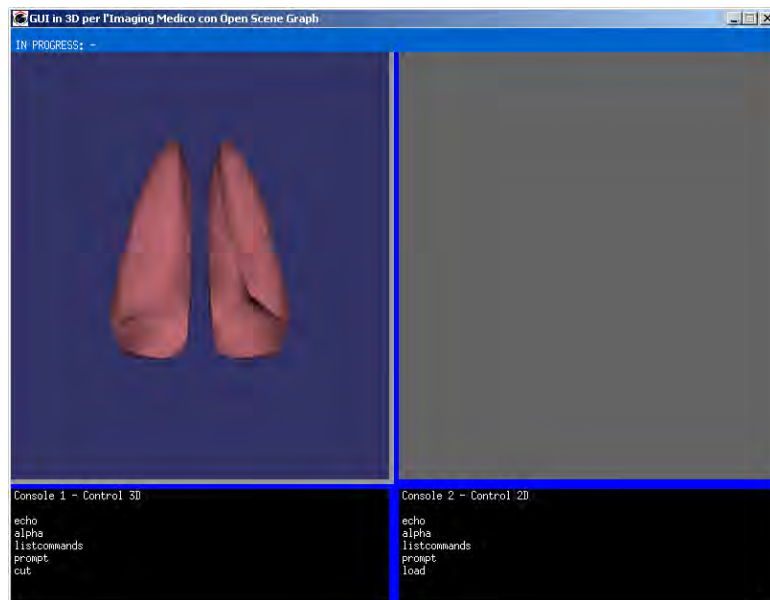


Figura 6.15: nuova interfaccia con l'immagine 3D di un polmone

Per quanto riguarda l'ultima interfaccia creata, al suo interno è inoltre presente (in alto a sinistra) uno strumento di verifica (status) dell'elaborazione in atto da parte dell'applicazione. Infatti tramite la console 1 (in basso a sinistra) è possibile eseguire l'operazione "cut" che genera all'interno della visualizzazione 3D un piano che divide virtualmente il polmone in 2 parti; inoltre lo status informerà dell'operazione in corso.

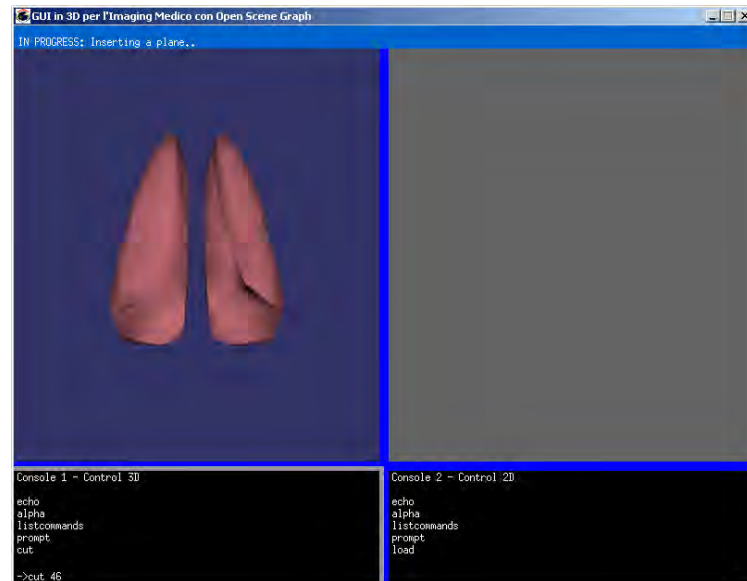


Figura 6.16: l'operazione di cut è monitorata dallo status in alto a destra

A questo punto verrà visualizzato il piano che taglia l'oggetto 3D.

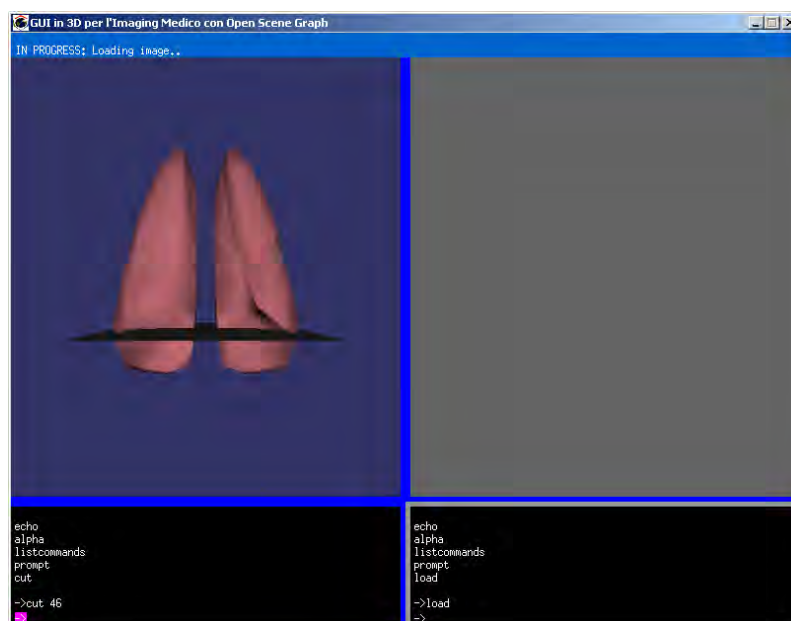


Figura 6.17: il piano di taglio è stato inserito nella scena 3D

Per vedere il corrispettivo risultato dal punto di vista bidimensionale è sufficiente spostarsi nella console 2 (a destra) e digitare il comando “load”; per rendere attivo uno dei quattro quadranti è sufficiente cliccare con il tasto sinistro del mouse all'interno della zona interessata.

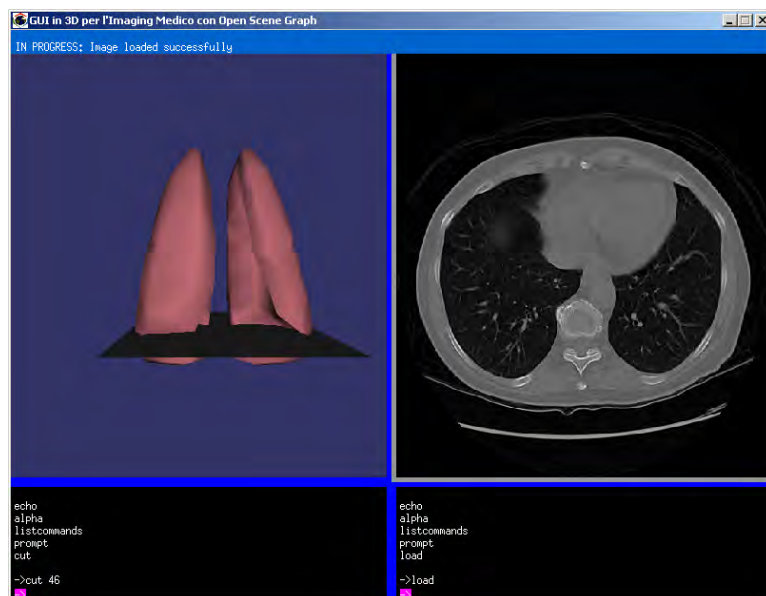


Figura 6.18: l'immagine 2D della sezione tagliata è visualizzata a destra

Le possibilità concesse dal gestore grafico tridimensionale permettono di creare più piani di taglio e analizzare il disegno da più punti di vista.

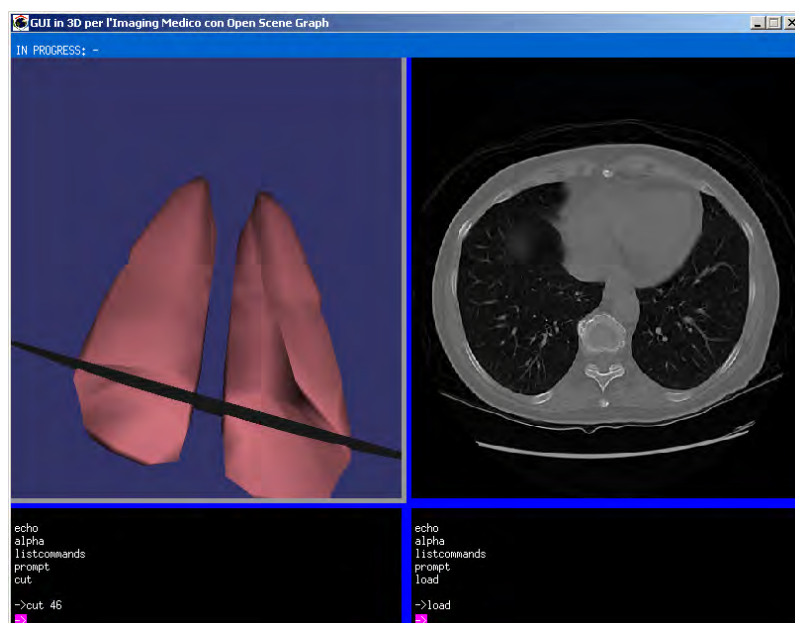


Figura 6.19: è possibile ruotare il polmone per analizzare da più punti di vista

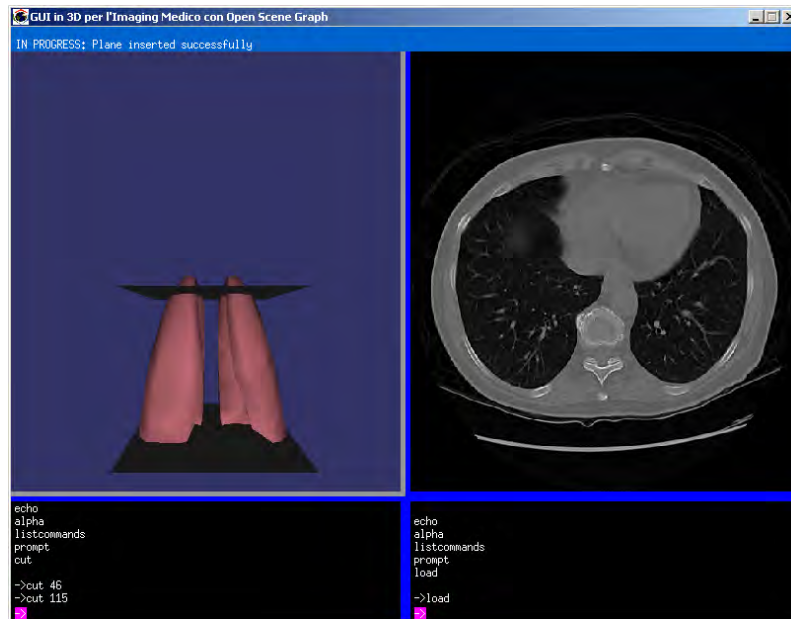


Figura 6.20: altri piani di taglio possono essere aggiunti o tolti

Inoltre è sempre possibile in ogni momenti effettuare il “load” dell’immagine riferito all’ultimo piano di taglio creato.

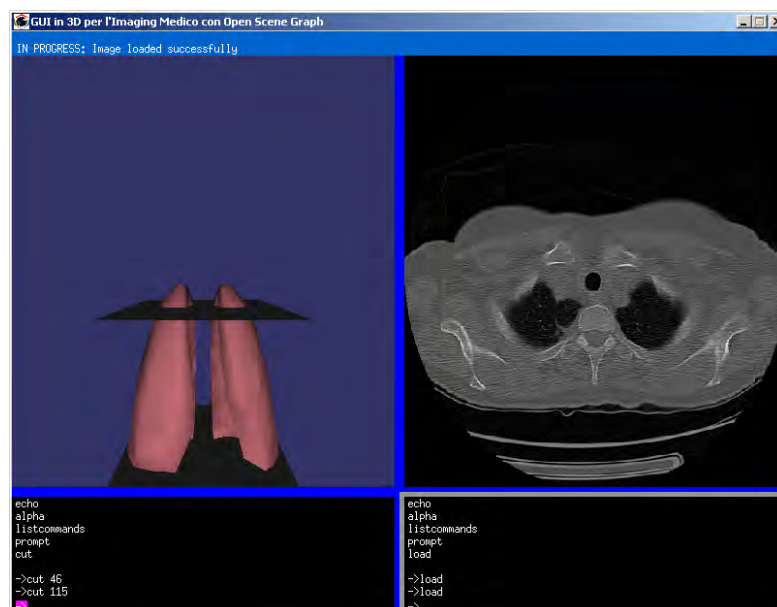


Figura 6.21: corrispondente immagine 2D al nuovo piano di taglio

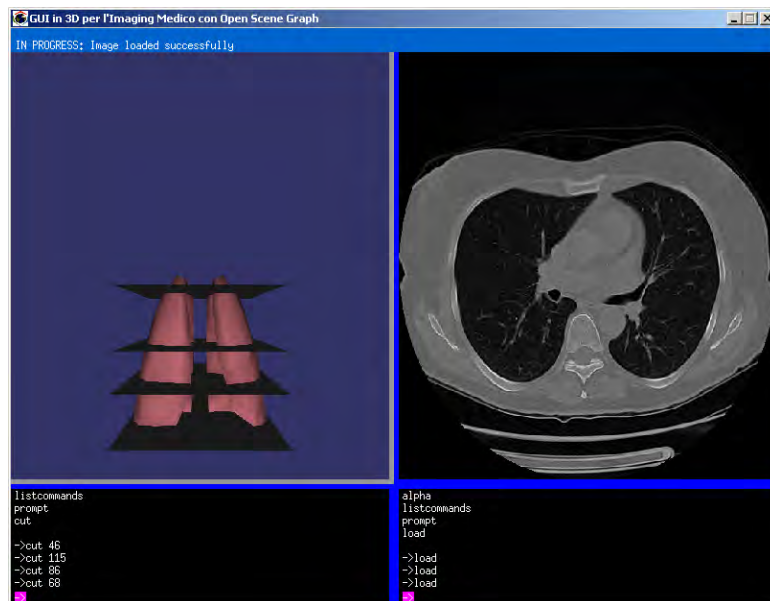


Figura 6.22: quattro piani di taglio creati; con l'ultimo piano viene caricata la corrispondente immagine 2D

6.10 Conclusioni e sviluppi futuri

E' ora visibile come sia diventato semplice costruire tale applicazione una volta conosciuti a fondo i mezzi; ancor più intuitiva è l'utilità che ne consegue dal punto di vista strettamente umano.

L'utilizzo dei widget grafici portabili apre la possibilità di un loro utilizzo al di fuori dei comuni PC o workstation, in particolare rende possibile sviluppare soluzioni multi-piattaforma eseguibili su dispositivi mobili quali palmari, smartphone, tablet e dispositivi embedded per usi specifici.

A titolo di esempio sarebbe anche possibile fornire la diagnosi in tempo reale su dispositivi palmari mediante l'invio della stessa tramite Bluetooth²⁹, WLAN³⁰,

²⁹ La tecnologia Bluetooth è specificatamente progettata per realizzare la comunicazione senza fili per apparecchi di piccole dimensioni

GPRS³¹ o UMTS³² e successivamente avviare il viewer con la visualizzazione finale dell'analisi.

Ciò permetterebbe una ancor più rapida e fluida comunicazione tra i vari settori del campo medico per ottenere in tempi altrettanto veloci risposte mediche importanti per la vita dell'uomo.

³⁰ Wireless LAN: per WLAN si intende una rete di comunicazione senza fili.

³¹ GPRS: General Packet Radio Service.

³² UMTS: Universal Mobile Telecommunications System.

L'idea base della tesi è stata l'applicazione dell'informatica in campo medico: questo tema è stato inizialmente affrontato con un lavoro di ricerca di quali mezzi implementativi fare uso.

La scelta è ricaduta sull'utilizzo di particolari librerie grafiche: SDL, OpenGL e Open Scene Graph hanno una caratteristica comune: sono multi-piattaforma e open source.

La ricerca di documentazione è stata così favorita dal fatto che risultano virtualmente accessibili e disponibili a tutti gli sviluppatori; inoltre la loro compatibilità ha reso l'obiettivo finale ancor più facilmente raggiungibile.

Le funzioni implementate hanno avuto applicazione inizialmente su esempi di sintesi, ma non è stato difficile renderle molto più professionali con l'introduzione di dati scientifici medici reali.

La realizzazione di un visualizzatore 3D generato da zero solo con OpenGL, e poi integrato con l'Open Scene Graph, ha mostrato quanto possano essere le variabili da considerare per ottenere un buon programma 3D; d'altro canto, le molteplici difficoltà affrontate nell'implementazione si equivalgono alle altrettante molteplici operazioni che si possono compiere su oggetti tridimensionali.

Disegnare un'interfaccia, che integra pienamente due visuali 2D e 3D, ha reso l'elaborazione ancor più interessante, ma soprattutto ha evidenziato le sue caratteristiche di grande utilità come mostrato dagli esempi visivi di applicazione in campo medico. Inoltre l'inserimento di console che gestiscono in pieno ogni istruzione a run-time ha portato a realizzare una GUI completa dal punto di vista funzionale.

E' necessario ricordare che lo sviluppo di questi tool singolarmente non rappresenta un'innovazione nel settore; ciò che caratterizza questa applicazione è l'originalità ed l'innovatività data dall'uso integrato di queste librerie, studiate per essere multi-piattaforma.

Lo sviluppo di tale progetto per una sua possibile evoluzione prevedrebbe la programmazione dello stesso su computer più complessi di quelli in cui è stata implementata, con la possibilità di gestire ingenti quantità d'informazioni a livello di ricostruzione tridimensionale, e di elaborazione computazionale ulteriormente ottimizzata; la possibilità di far girare il prodotto su molteplici sistemi operativi permetterebbe inoltre di analizzare i risultati in relazione al sistema operativo in uso, ma inoltre espande universalmente il campo di verifica a qualsiasi tipo di programmatore.

Bibliografia

- [A01] R. Campanini, E. Angelini, D. Dongiovanni, E. Iampieri, N. Lanconelli, C. Mair-Noack, M. Masotti, G. Palermo, M. Roffilli, G. Saguatti, O. Schiaratura,
“Preliminary results of a featureless CAD system on FFDM images”,
7h International Workshop on Digital Mammography, June 18-21 2004,
Durham, North Carolina, USA
- [A02] Apple Computer Inc.,
<http://www.apple.com/>
- [A03] Massachusetts Institute of Technology,
<http://web.mit.edu/>
- [A04] Association for Computing Machinery,
<http://www.acm.org/>
- [A05] Graphical Kernel System Availability,
<http://members.aol.com/pde2d/gks.htm>
- [A06] American National Standards Institute,
<http://www.ansi.org/>
- [B01] Microsoft Windows,
<http://www.microsoft.com/>
- [B02] Linux.com: The Enterprise Linux Resource, <http://www.linux.com/>
- [B03] The UNIX System, UNIX System,
<http://www.unix.com/>

- [B04] Solaris Operating System,
<http://www.sun.com/solaris>
- [B05] GNU Operating System - Free Software Foundation,
<http://www.gnu.org/>
- [B06] Wine HQ,
<http://www.winehq.com/>
- [B07] Open Source Initiative OSI - Welcome,
<http://www.opensource.org/>
- [B08] Open Office,
<http://www.openoffice.nl>
- [B09] “SWAR - MMX,SSE,SSE2 - Programmazione Multi-piattaforma”,
Talk, Linux Day, November 22 2002, Cesena, Italy
- [B10] Open Standard X11,
<http://www.x11.org/>
- [B11] “GCC 3.3.x Vs .NET 2003”,
Talk, Linux Day, November 28 2003, Cesena, Italy
- [B12] “The multi-platform philosophy: how to develop under Linux and to
sell for Windows”,
Talk, Security Day & Linux Day, November 26-27 2004, Cesena, Italy,
- [B13] Microsoft DirectX: Home Page,
<http://www.microsoft.com/windows/directx/default.aspx>
- [D01] M. Nori, Tesi di Laurea, CdL in Scienze dell'Informazione
“Visualizzazione e controllo remoto di agenti in ambienti Virtuali
tramite la libreria Open Scene Graph”, 2004

- [D02] S. Cacciaguerra, A. Lomi, M. Roccetti, M. Roffilli,
“*A Wireless Software Architecture for Fast 3D Rendering of Agent-Based Multimedia Simulations on Portable Devices*”,
IEEE Consumer Communications and Networking Conference 2004,
Caesars Palace, Las Vegas, Nevada, USA, 5-8 January 2004
- [D03] The FreeBSD Project,
<http://www.freebsd.org/it>
- [D04] SGI - Products: Software: IRIX,
<http://www.sgi.com/products/software/irix/>
- [D05] Blue Marble Viewer on Open Scene Graph,
<http://www.andesengineering.com/BlueMarbleViewer>.
- [E01] A. Bevilacqua, M. Roffilli,
“*Robust denoising and moving shadows detection in traffic scenes*”,
IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Kauai Marriott, Hawaii, Dec 9-14,2001
- [E02] G.Garavini, Tesi di Laurea, CdL in Scienze dell’Informazione
“Gestione, tramite protocollo DICOM, dei risultati di un programma di Computer Aided Detection”, 2003
- [E03] M.Gieri, Tesi di Laurea, CdL in Scienze dell’Informazione
“Realizzazione di una applicazione per il trasferimento di immagini mediche tramite protocollo DICOM”, 2002
- [E04] R. Campanini, E. Angelini, E. Iampieri, N. Lanconelli, M. Masotti, M. Roffilli, O. Schiaratura, M. Zanoni,
“*A fast algorithm for intra-breast segmentation of digital mammograms for CAD systems*”,

7th International Workshop on Digital Mammography, June 18-21
2004, Durham, North Carolina, USA

- [E05] A. Bazzani, A. Bevilacqua, D. Bollini, R. Campanini, D. Dongiovanni, E. Iampieri, N. Lanconelli, A. Riccardi, M. Roffilli, R. Tazzoli,
"A novel approach to mass detection in digital mammography based on Support Vector Machines (SVM)",
Proceedings of the 6th International Workshop on Digital Mammography, June 22-25 2002, Bremen Germany
- [E06] R. Campanini, D. Dongiovanni, E. Iampieri, N. Lanconelli, M. Masotti, G. Palermo, A. Riccardi and M. Roffilli,
"A novel featureless approach to mass detection in digital mammograms based on Support Vector Machines",
Physics in Medicine and Biology, volume 49, issue 6, pages 961 - 975
- [E07] R. Campanini, E. Angelini, D. Dongiovanni, E. Iampieri, N. Lanconelli, C. Mair-Noack, M. Masotti, G. Palermo, M. Roffilli, G. Saguatti, O. Schiaratura,
"Preliminary results of a featureless CAD system on FFDM images",
7th International Workshop on Digital Mammography, June 18-21 2004, Durham, North Carolina, USA

RINGRAZIAMENTI

Ringrazio il prof. Renato Campanini per avermi dato la possibilità di partecipare a questo progetto, per avermi fornito i mezzi necessari e per avermi guidato nella sua realizzazione.

Ringrazio il dott. Matteo Roffilli per avermi fornito tutta l'assistenza e l'attenzione possibile, gli incoraggiamenti e i consigli che hanno portato allo sviluppo del progetto più importante che abbia realizzato.

Ringrazio e dedico dal più profondo del cuore questa tesi, ma soprattutto questa avventura universitaria alla mia famiglia: grazie a Massimo, Elsa, Massimiliano, Gina e Luisa ogni momento più difficile e amaro di questi sei anni è stato addolcito dal loro sostegno e dalla loro piena fiducia in me.

Non ci parole che possano descrivere cosa significhi questo per me, perciò mi servo di un “semplice” grazie mille.

Ringrazio tutti quelli a cui voglio bene e che me ne vogliono altrettanto, a tutte quelle persone che mi sono state vicine dall'inizio alla fine, e anche a quelle che in questo periodo sono entrate e uscite dalla mia vita: ognuno ha portato a modificare il mio carattere per rendermi conto sempre più di quanto sia importante l'amicizia e l'amore per riuscire ad ottenere quello che si vuole.

Voglio perciò ringraziare le persone più importanti che ho conosciuto qua a Cesena e che hanno reso la mia vita qua in Emilia Romagna un piacevole soggiorno: “Cesena”, Lalla, Chiara, Marta, Luana, Matteo, Manuela, Bruna, “Colla”, Alessandra, Romina, Diletta, Elisa, Vanessa, Graziana, Filippo, Franco, il Signor. Medri, LucaG, Rocco, Roberto, Erich, Mattia, Marco, Berna, Manuele, Gianni, Max, Giovanni, Vito, Mauro, Nunzio, Riccardo, Stefano AN, Angelo e il gruppo del calcetto.

Le persone che non ho mai perso di vista quando avevo residenza a Cesena, sono gli amici di Città di Castello, la mia città natale: Maurizio, Pao, Polmone, Tarpi, Alberto, Chess, Stefania, Laura, il gruppo della Domauto, Elisa, Federica, Tamara, Lucia, Francesca, Trella, Ricciolino, Muscio, Rigo, Grillo, l'Arci Riosecco, Checco, Annarita,

All'amore della mia vita, Claudia, per ogni istante vissuto con lei, per avermi sostenuto nei momenti più difficili e per essere stata la mia guida in questi cinque anni.