

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

**PROGETTAZIONE E REALIZZAZIONE DI
UNA GUI MULTI-PIATTAFORMA PER
APPLICAZIONI MEDICHE IN 2D**

Tesi di laurea in

Fisica Numerica

Relatore

Chiar.mo Prof. Renato Campanini

Presentata da

Erich Zangheri

Co-relatore

Dott. Matteo Roffilli

Sessione I

Anno Accademico 2003/2004

INTRODUZIONE.....	5
 CAPITOLO PRIMO	 7
INTERFACCIA GRAFICA.....	7
1.1 INTRODUZIONE.....	8
1.2 PROGETTARE UN'INTERFACCIA GRAFICA UTENTE	13
1.2.1 Focalizzare l'interfaccia sull'utente finale	15
1.2.2 GUI procedurali e ad oggetti.....	16
1.2.3 Modelli, metafore e analogie	22
1.2.4 Alcune regole per sviluppare una buona interfaccia utente	23
 CAPITOLO 2.....	 33
MULTI-PIATTAFORMA.....	33
2.1 I VANTAGGI DEL MULTI-PIATTAFORMA	37
2.2 WRAPPER VS EMULATED.....	42
2.3 INTRODUZIONE ALLE GUI PER APPLICAZIONI MEDICHE	44
2.3.1 Fornire le indicazioni e le opzioni di navigazione.....	45
2.3.2 Ricorrere all'utilizzo di una griglia.	46
2.3.3 Uso di semplici caratteri.....	47
2.3.4 Attenzione a etichette ridondanti	48
2.3.5 Utilizzo di un linguaggio semplice.....	48
2.3.6 Rifinire e armonizzare le icone	48
2.3.7 Eliminare le inconsistenze.....	50
 CAPITOLO TERZO	 53
GLI STRUMENTI PER LO SVILUPPO MULTI-PIATTAFORMA	53
3.1 JAVA.....	54
3.1.1 Il bytecode e la Java Virtual Machine	55
3.1.2 I vantaggi e gli svantaggi di Java	56

3.2 Qt.....	62
3.3 GTK	65
3.4 WxWINDOWS	67
3.5 LE LIBRERIE MESSE A CONFRONTO	71
3.6 RUNTIME-EFFICIENCY	73
3.7 MEMORY-EFFICIENCY	74
 CAPITOLO 4	 77
SIMPLE DIRECTMEDIA LAYER.....	77
4.1 PERCHÉ SCEGLIERE SDL.....	78
4.2 BREVE STORIA	79
4.3 LA SITUAZIONE OGGI	80
4.4 LA COMUNITÀ	81
4.5 PANORAMICA.....	82
4.6 INIZIALIZZAZIONE	83
4.6.1 <i>Eventi</i>	84
4.6.2 <i>Un contesto grafico</i>	85
4.6.3 <i>SDL_KEYDOWN, SDL_KEYUP</i>	90
4.6.4 <i>SDL_ACTIVEEVENT</i>	91
4.6.5 <i>Gli eventi del mouse</i>	92
4.7 VIDEO	94
4.7.1 <i>SDL_VIDEORESIZE</i>	95
4.7.2 <i>I modi video disponibili</i>	96
4.8 PIXEL FORMAT	97
4.9 BLITTING	102
4.9.1 <i>Double Buffering</i>	103
4.10 FUNZIONI AVANZATE	103
4.10.1 <i>Bump Mapping</i>	103
4.10.2 <i>Lock Surface</i>	104
4.11 L'AUDIO.....	104
4.12 CD-ROM.....	108

4.13 JOYSTICK.....	111
4.14 MULTITHREADING.....	113
4.14.1 Alcune particolarità	114
4.15 TIMER	119
 CAPITOLO QUINTO	121
IMPLEMENTAZIONE DELL'INTERFACCIA GRAFICA.....	121
5.1 SETUP DI MICROSOFT VISUAL STUDIO .NET PER SDL.....	122
5.2 SEGMENTAZIONE, PRE-DETECTION, DETECTION.	123
5.2.1 La segmentazione.....	124
5.2.2 La Pre-detection.....	125
5.2.3 Detection	127
5.3 IL PROGETTO	128
5.4 IL SOTTO SISTEMA VIEWER.	129
5.5 IL SOTTO SISTEMA LAYER	133
5.5.1 Creare una Color Key (la trasparenza).....	134
5.5.2 Alpha Blending.....	137
5.5.3 Per-surface alpha.....	138
5.5.4 Per-Pixel Alpha.....	138
5.5.5 Le palette.....	139
5.5.6 L'aggiornamento del monitor	141
5.6 LO ZOOM.....	142
5.7 LA CONSOLE.....	144
 CAPITOLO SESTO.....	147
SVILUPPI FUTURI.....	147
 CAPITOLO SETTIMO.....	149
CONCLUSIONI.....	149

BIBLIOGRAFIA	151
---------------------------	------------

RINGRAZIAMENTI	157
-----------------------------	------------

INTRODUZIONE

Il tumore al seno è statisticamente la forma tumorale più diffusa fra le donne del mondo con una età compresa tra i 40 ed i 55 anni¹. Si stima che nel corso della sua vita, una donna, ha il 12% di probabilità di soffrire di questa malattia. Sono diversi i fattori che possono influire sull'aumento del rischio di sviluppo di un carcinoma mammario, quali la menopausa tardiva, fattori dietetici, terapie ormonali, etc [ROE90]. L'unico che è dimostrato assumere un ruolo fondamentale sulla probabilità di contrazione del disturbo è l'età. Si stima che ogni anno vengano diagnosticati un milione di nuovi casi, con un tasso di mortalità superiore alle trecento mila vittime.

Il modo più efficace per poter diagnosticare questo tipo di patologie e di conseguenza intervenire tempestivamente è l'esame mammografico preventivo effettuato con apparecchiature radiodiagnostiche.

Purtroppo, il 25% di queste patologie sfugge al controllo del radiologo, mentre una percentuale del 17% viene erroneamente diagnosticata da questi. Le cause vanno cercate nella notevole difficoltà della sua individuazione ad occhio nudo, dovuto alla somiglianza con il tessuto ospitante. Ciò accade soprattutto nella fase embrionale dello sviluppo della patologia, in quanto caratterizzata da dimensioni molto ridotte. È la diagnosi precoce che fornisce una possibilità abbastanza elevata di guarigione, garantendo anche una buona possibilità di intervento sul carcinoma con tecniche non invasive. Per aumentare

¹ Greenlee, Hill-Hammon, Murray, Thun 2001

sensibilmente l'accuratezza delle letture sarebbe necessaria la visione delle lastre da parte di un secondo radiologo o in alternativa l'utilizzo di un sistema computerizzato specifico per questo tipo di problemi.

Questa tesi vuole essere un contributo alla realizzazione di un sistema CAD, (Computer Aided Detection) progettato e sviluppato da un gruppo di ricerca 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. Scopo dell'elaborato è valutare le soluzioni più efficienti per la realizzazione di un framework grafico multi-piattaforma. In particolare modo sono stati sviluppati quattro tools per il sistema CAD al fine di ottenere una visualizzazione grafica della diagnosi effettuata da quest'ultimo.

Capitolo Primo

Interfaccia grafica

In questo capitolo verranno date prima alcune definizioni di interfaccia utente, ed in particolar modo di interfaccia grafica, e poi verranno spiegate le tecniche e le regole da seguire nella progettazione e realizzazione di una valida interfaccia grafica. Verrà mostrato quanto risulti fondamentale porre l'utente finale al centro dell'attenzione sia durante lo sviluppo dell'interfaccia utente, sia durante le ripetute fasi di test per valutare il gradimento del progetto sviluppato [DRA86]. Verrà spiegato brevemente la differenza fra interfacce progettate con linguaggi procedurali e interfacce progettate con linguaggi ad oggetti. Forniremo un esempio pratico che permetta di coglierne la differenza fondamentale. Infine, saranno proposte le dieci regole indicate in letteratura [MCT04] come criteri di base per progettare interfacce grafiche ben strutturate, estremamente efficienti e apprezzate dagli utenti finali.

1.1 Introduzione

Prima degli anni Settanta i computer erano progettati per elaborare calcoli matematici e scientifici, in grado cioè di processare solo input e output di tipo numerico e le informazioni che restituivano erano prevalentemente dei risultati di numerici. Negli anni Ottanta, con l'avvento dei primi sistemi di elaborazione di tipo aziendale, oltre all'elaborazione dei numeri comparve anche la gestione dell'alfabeto. Si oltrepassò quindi per necessità (come stampare gli indirizzi dei clienti o articoli) una visione puramente matematica.

Però la relazione uomo-macchina non era ancora matura; i sistemi erano concepiti per ricevere e svolgere un determinato calcolo e fornire un risultato. L'uomo si limitava ad interagire, spesso usando dei mezzi (schede perforate, nastri, interruttori, etc.) per dare dei comandi iniziali e di termine di un programma. I risultati venivano poi visualizzati tramite schede perforate o nastri. I messaggi di errore e ogni altra comunicazione erano forniti dal calcolatore su display di tipo meccanico, o per mezzo di lampadine, o ancora tramite stampante. La visualizzazione su monitor era agli albori e non rappresentava comunque un'interfaccia vera e propria.

La necessità di vendere computer ad acquirenti privati, privi di qualsiasi conoscenza tecnica, portò alla sostituzione dell'uso di sole parole, per rappresentare l'input e l'output di un programma, con l'uso di immagini. Nacque così l'*interfaccia grafica utente*, meglio conosciuta come *Graphical User Interface – GUI*, proprio allo scopo di rendere più familiari i comandi da impartire alla macchina. Infatti, l'uso delle finestre, del puntatore, dei menù e delle icone rendono più facile il lavoro all'utente e più intuitivo l'apprendimento.

La Apple [APP04] fu la prima società a vendere computer che possedevano una *interfaccia utente grafica*. Questa è la parte di un programma informatico in grado di visualizzare graficamente le operazioni compiute dall'utente e di interagire con il calcolo in modalità visuale. Una buona interfaccia utente rende semplice, a chi l'utilizza, fare ciò che desidera [SPI04].

I sistemi attuali Windows Xp, Mac OS X e Linux, utilizzano interfacce grafiche denominate rispettivamente Luna, Appleacqua, Kde, che sono variazioni più o meno fedeli all'originale di quella prima tecnologia Apple.

Progettare un'efficace interfaccia utente per un programma non è molto diverso dal progettare le componenti di controllo di uno stereo o di una televisione; vi sono delle considerazioni in termini di immediatezza, usabilità, chiarezza che vanno attentamente studiate se si desidera ottenere il miglior risultato. Proprio per questa ragione sarebbe utile che il programmatore di interfacce estendesse la propria conoscenza che riguarda l'iterazione uomo-oggetti anche al di fuori del settore informatico.

Il progetto dell'interfaccia utente dovrebbe essere ottenuto come la combinazione di due modelli, molte volte contrapposti:

- Il *modello concettuale dell'utente*
- Il *modello del programmatore*

ciò che ne risulta prende il nome di *modello del progettista*.

Il primo è un concetto molto informale, formato da un insieme di relazioni tra un insieme di elementi. Tali relazioni sono basate sull'esperienza quotidiana dell'utente e sulla somiglianza con altre applicazioni. Infatti per le nuove esperienze l'uomo crea delle analogie e somiglianze basandosi sulle situazioni passate. Ad esempio l'icona di un fax verrà associata all'idea di un programma o di un dispositivo per l'invio di fax. È importante cercare di avvicinarsi il più possibile allo schema mentale dell'utente, altrimenti il rischio è quello di vedere la propria applicazione considerata come troppo complessa. Purtroppo capita che la visione del programmatore sia troppo tecnica per l'utente finale, ed è proprio per questo che nasce il bisogno di un modello che cerchi di mediare tra le esigenze della visione dell'utente e del programmatore, spesso contrastanti. Tale mediazione viene svolta dal modello del progettista, i cui elementi sono:

1. Gli oggetti a livello di interfaccia e le loro relazioni
2. La rappresentazione visiva degli oggetti che prende il nome di *look*
3. Le tecniche ed i meccanismi di interazione che prendono il nome di *feel*

Scopo del progettista è quello di scegliere le azioni e gli oggetti più rappresentativi, capire cosa va inserito e cosa escluso, per esempio per la difficoltà eccessiva di programmazione o tempi di sviluppo troppo lunghi.

Esistono due modelli fondamentali per l'interazione uomo-macchina (vedi figura 1.1) che spesso vengono utilizzati all'interno della stessa applicazione in momenti diversi:

- Action-object
- Object-action

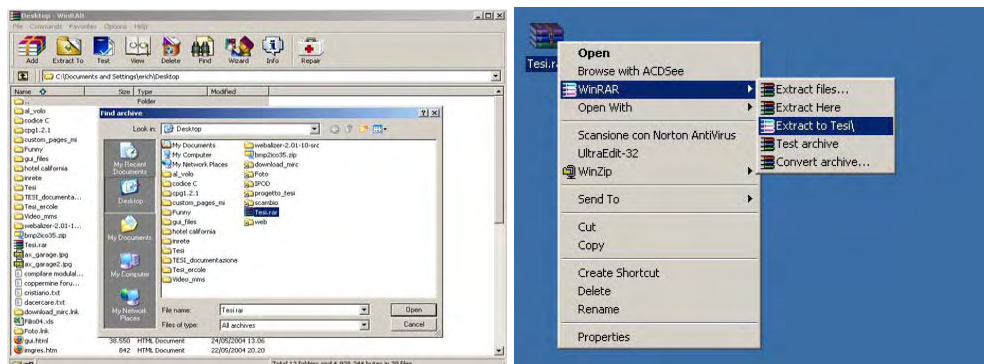


Figura 1.1: a sinistra un esempio di action-object, a destra di object-action

Il metodo action-object consiste nello scegliere l'azione da compiere, e poi l'oggetto sul quale compierla, ad esempio l'apertura di un file all'interno di un'applicazione: si seleziona prima l'azione file poi open e in seguito l'oggetto da aprire. In questo caso il sistema guida l'utente posizionandosi nella cartella predefinita per quel tipo di file.

In generale il metodo action-object guida il più possibile l'utente, il quale in seguito ad una azione si trova di fronte ad una serie di scelte, quasi tutte

obbligate, sulle quali operare come la compilazione di un form, una selezione, etc. È ovvio che il progettista obbliga l'utente ad operare secondo il proprio senso logico.

Al contrario il metodo object-action consiste nello scegliere l'oggetto ed in seguito l'azione da compiere come ad esempio il browser o i programmi vettoriali come il Flash [FLA04]. Nel primo caso possiamo navigare all'interno del disco fisso, scegliendo in ogni momento l'azione da applicare agli oggetti selezionati come aprire, cancellare, spostare etc. Nel secondo caso possiamo selezionare un qualunque oggetto grafico ed applicare qualche azione come ingrandire, ruotare, cambiare colore, etc.

L'utente è completamente libero da costrizioni, naturalmente nei limiti previsti da quelle del sistema; non vi sono imposizioni sulla sequenza delle operazioni da utilizzare per il completamento di una attività. L'utente esperto è in grado muoversi secondo le proprie abitudini, anziché essere influenzato dall'applicazione; al contrario l'utente meno esperto si troverà intimidito e bloccato.

Il primo è meno flessibile, però è il più indicato per gli utenti inesperti od occasionali.

Possiamo pensare ad un'interfaccia come l'elemento di contatto o di intermediazione fra entità, sistemi, cose o persone diverse e un ambiente. Anche se non ce ne rendiamo conto, nella vita di tutti i giorni utilizziamo un gran numero di interfacce (vedi figura 1.2), come per esempio:

- stereo
- lavatrice
- telecomando TV
- sistema di guida dell'automobile (cruscotto + comandi)
- computer (sistema operativo e programmi)

Tra le caratteristiche principali di un'interfaccia che funzioni bene ci sono due elementi fondamentali a garanzia di efficacia comunicativa:

- 1) chiarezza;
- 2) semplicità.



Figura 1.2: Alcuni esempi di interfaccia utente da sinistra: telecomando, cruscotto auto, lavatrice, Apple ibook

Un esempio di un'interfaccia ben progettata è il sistema dei comandi di un'automobile. Se si sa guidare, si possono utilizzare anche macchine che non si sono mai usate prima, seguendo comportamenti e azioni che fanno parte ormai dell'abitudine di qualunque conducente [HTM04].

Progettare *l'Interfaccia Utente* significa comporre in un unico disegno:

1. *metafore di interazione;*
2. *immagini;*
3. *concetti.*

usati per veicolare *funzioni* e *contenuto informativo* sullo schermo. Proprio per questo risulta essere di fondamentale importanza un'accurata progettazione che ponga l'attenzione sull'utente e i suoi compiti piuttosto che sugli aspetti tecnologici.

L'interfaccia permette l'utilizzo di un sistema tecnologico complesso e dipende dal livello di esperienza culturale di chi utilizza l'oggetto tecnologico, cioè dall'esperienza e dal know how di quel'ambiente specifico posseduti dall'utilizzatore.

Ad esempio è possibile utilizzare un telefonino con le istruzioni in cinese, se si ha una buona esperienza di telefonini perché si è in grado di interpretare la struttura semiotica del telefonino anche senza capire i singoli idiomi. Nello stesso modo si può utilizzare una macchina con cui si ha minore dimestichezza se le istruzioni (cioè l'interfaccia) sono sufficientemente chiare.

Sintetizzando, per utilizzare agevolmente un'interfaccia due sono le opzioni possibili:

1. capire il linguaggio;
2. conoscere l'utilizzo.

Un'interfaccia fa sempre riferimento ad un ipotetico livello medio cognitivo dell'utilizzatore, che si identifica con l'ambito culturale e la familiarità tecnologica col sistema. È cosa nota che gli utenti di molte macchine, dai videoregistratori al computer, utilizzino solo in parte l'insieme delle operazioni consentite dalla macchina. Proprio per questo si identificano tre categorie di operazioni, che qui sintetizziamo:

1. le operazioni che un oggetto permette di compiere;
2. le operazioni che l'utilizzatore percepisce come consentite dall'oggetto;
3. le operazioni che l'utilizzatore effettivamente compie con l'oggetto;

Una buona interfaccia (o potremmo dire un buon design) deve ridurre le distanze tra le operazioni che l'oggetto permette di compiere e quelle che effettivamente compie.

1.2 Progettare un'interfaccia grafica utente

Qualsiasi realizzazione di un software, sia di grande potenzialità o più semplicemente per uso personale o per l'ufficio, ha bisogno di una efficace interfaccia utente per rendere appieno la propria potenzialità. Il progetto di una

tale interfaccia prevede disciplina (seguire le convenzioni della piattaforma e i principi di una buona progettazione), richiede tecnica (effettuare test di usabilità) e necessita conoscenza d'arte grafica (creare schemi grafici che siano esplicativi, intuitivi e visivamente piacevoli).

La preparazione di software a pagamento mostra i principi che stanno dietro alle valide interfacce utente, meglio di qualsiasi altro tipo di applicazioni. Questo serie di programmi sono in sostanza nuovi a tutti i futuri utenti e nessuno di essi avrà la voglia e/o il tempo necessari per mettersi a leggere la relativa documentazione.

Nonostante questo, milioni di acquirenti, compresi molti utenti alle prime armi, hanno intenzione di vedere subito i frutti del loro investimento.

La soluzione è semplice: funzionalità e interfaccia. Se funzionalità indica quello che un programma effettivamente fa, l'interfaccia indica come l'utente interagisce con il programma e ne percepisce il funzionamento. Nascondere i dettagli di una complessa tecnologia dietro a una valida interfaccia utente, indirizzata ad un semplice utilizzo, non è un concetto nuovo. Una persona non deve certo sapere come funziona l'instradamento dei pacchetti GPRS per poter telefonare.

Anche se le interfacce puramente testuali possono essere ancora appropriate per alcune applicazioni, oggi la maggior parte dei programmatori Windows [MIC04g] , Linux [LIN04], Macintosh utilizzano la più popolare e versatile *Graphic User Interface*. I segni distintivi della programmazione di una *GUI* risiedono nell'utilizzo di caratteristici controlli grafici, come possono essere le barre degli strumenti, le icone o i bottoni. A differenza delle interfacce puramente testuali che accettano input solo mediante la pressione di tasti sulla tastiera, le *GUI* consentono una grande varietà di dispositivi di input, tra i quali il più noto è il mouse, a disposizione degli utenti per manipolare il testo e le immagini proprio come le vedono disegnate.

Visual Basic [MIC04b], *Delphi* [BOR04], *MFC* [MIC04c], *.NET 2003* [MIC04d], *QT* [QT04], *SDL* [SDL04a], sono strumenti comunemente usati per lo sviluppo di applicazioni grafiche.

1.2.1 Focalizzare l'interfaccia sull'utente finale

Nei primi anni della programmazione software intorno al 1950 un programma era ben sviluppato se era in grado di funzionare e utilizzare un basso quantitativo di risorse. I programmi di un tempo, sviluppati mediante un duro e complicato lavoro, erano destinati ad esperti di computer e non ad utenti comuni, una generica interfaccia utente che permettesse un rapido inserimento dati da parte dell'utente era la scelta più frequente. In quel periodo il punto di incontro tra le persone comuni e i computer non era il software sviluppato, ma piuttosto i resoconti bancari, i tabulati telefonici e i risultati elaborati dai laboratori di ricerca/analisi [ERC04].

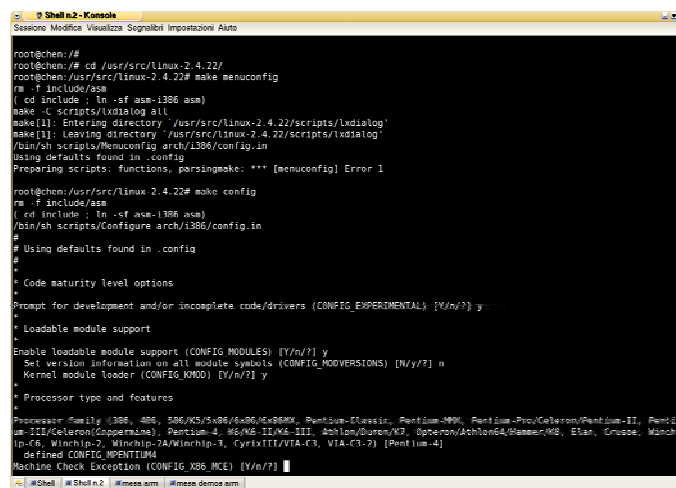
Nel mondo di oggi, in cui le risorse informatiche sono abbondanti e gli utilizzatori di computer sono in buona parte neofiti del settore, un buon programma è quello che non solo funziona ma che è anche semplice da capire. E un grande programma è quello che funziona ed è talmente intuitivo da non aver bisogno di essere capito. Con il passaggio ad un'ampia utenza non specializzata nel settore informatico, che dispone di computer altamente performanti e in grado di svolgere molte funzioni, l'attenzione nello sviluppo dei software si è estesa ben oltre dall'obiettivo centrale prefissosi, ma si avvicina sempre più all'esigenze di semplicità e chiarezza richieste dall'utente.

Dal punto di vista dell'utente, il segreto di un software è colpire l'attenzione. Il programma non è più sviluppato pensando che la soluzione di un singolo problema principale sia al centro dell'attenzione del programmatore, ma pensando che deve essere l'utente al quale il programma è destinato ad essere al centro dell'attenzione degli sviluppatori. Ragionando in questa prospettiva si nota come sia fondamentale coinvolgere l'utente stesso nel processo di sviluppo dell'applicazione. Pertanto è necessario analizzare il comportamento di un

utente generico, osservare il suo atteggiamento con i software a sua disposizione e i relativi manuali d'uso, quali strumenti gli sono più familiari e quali operazioni svolge in automatico. Lo scopo del programmatore è capire come l'utente si pone davanti al programma, per poi offrire un'interfaccia in cui l'utilizzatore finale si trovi a proprio agio. Una parte essenziale di questo processo è la fase di testing, in cui si coinvolgono gli utenti per vedere le loro reazioni di fronte alle versioni beta dei programmi in fase di sviluppo. Più l'applicazione si evolve e si avvicina alla sua forma finale, maggiore dovrebbero essere le operazioni di testing.

1.2.2 GUI procedurali e ad oggetti

Nella programmazione procedurale guidata, le istruzioni sono elaborate in un ordine prefissato, un passo dopo l'altro. Per esempio, in un software per la gestione dei contributi, un programma procedurale farebbe avanzare l'utente da una linea successiva all'altra, attraverso tutti i moduli da riempire mediante un ordine prestabilito.



```
root@chen:/#  
root@chen:/# cd /usr/src/linux-2.4.22/  
root@chen:/usr/src/linux-2.4.22# make menuconfig  
rm -f include/asm  
( cd include ; ln -sf asm-i386 asm )  
make -C scripts/xdialog all  
make[1]: Entering directory /usr/src/linux-2.4.22/scripts/xdialog  
make[1]: Leaving directory /usr/src/linux-2.4.22/scripts/xdialog  
/bin/sh scripts/Menuconfig arch/i386/config.in  
Using defaults found in .config  
Preparing scripts: functions, parsingmake: *** [menuconfig] Error 1  
root@chen:/usr/src/linux-2.4.22# make config  
rm -f include/asm  
( cd include ; ln -sf asm-i386 asm )  
/bin/sh scripts/Configure arch/i386/config.in  
#  
# Using defaults found in .config  
#  
# Code maturity level options  
#  
Prompt for development and/or incomplete code/drivers (CONFIG_EXPERIMENTAL) {Y/n/?} y  
#  
# Loadable module support  
#  
Enable loadable module support (CONFIG_MODULES) {Y/n/?} y  
Set version information on all module symbols (CONFIG_MODVERSIONS) {N/y/?} n  
Kernel module loader (CONFIG_KMOD) {Y/n/?} y  
#  
# Processor type and features  
#  
Processor family (386, 486, 586/K5/586/6x86/6x86MMX, Pentium-Classix, Pentium-MMM, Pentium-Pro/Celeron/Pentium-II, Pentium-III/Celeron/Coppermine), Pentium-4, 66/K6-II/K6-III, Athlon/Duron/K7, Opteron/Athlon64/Itanium2, Elan, Crusoe, Winchip-C6, Winchip-2, Winchip-2A/Winchip-3, CyrixIII/VIA-C3, VIA-C3-2) [Pentium-4]  
defined CONFIG_MPENTIUM4  
Machine Check Exception (CONFIG_X86_MCE) {Y/n/?}   
#
```

Figura 1.3: ricompilazione Kernel Linux shell console

L'utente è obbligato ad inserire i dati del modulo corrente e non è in grado di passare ad un altro se non a quello successivo, a meno che ciò non sia previsto e gestito dal programmatore mediante l'implementazione di un apposito menu.

La shell di linux per esempio (figura 1.3), consente la comunicazione tra l'utente e il sistema operativo, ed interpreta i comandi inseriti dall'utente in modo sequenziale e li trasferisce al SO

Al contrario le *GUI* permettono una programmazione orientata agli oggetti, ed in questa facoltà risiede buona parte della loro forza.

Nella programmazione agli oggetti, l'utente controlla l'ambiente di lavoro mediante eventi (cliccando col mouse, attraverso comandi vocali, muovendo un joystick, o semplicemente premendo sulla tastiera, ecc) inviati dalla *GUI* al core dell'applicazione.

La molteplicità degli eventi che un utente può generare è definita dall'hardware e dal sistema operativo. Chiaramente, se l'utente possiede un mouse con un singolo tasto, di certo non potrà accedere all'evento associato alla pressione del tasto destro del mouse, anche nel caso esso sia previsto e gestito dal software.

Sia la programmazione procedurale che quella ad oggetti (figura 1.4) restringono il campo delle possibili scelte a disposizione dell'utente, ma solo quest'ultima gli permette un pieno

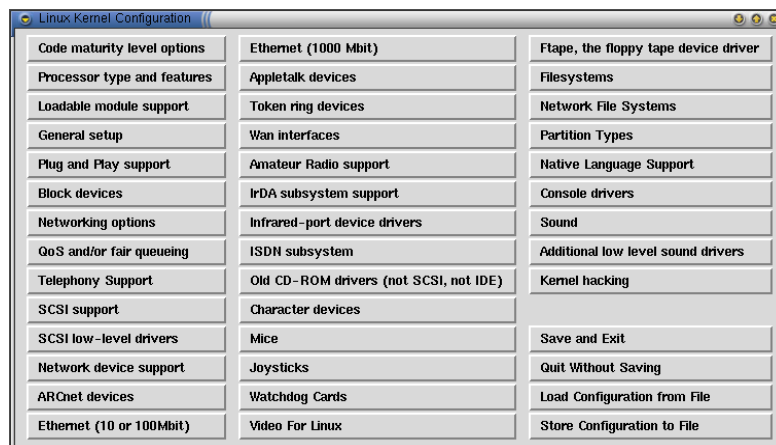


Figura 1.4: ricompilazione Kernel Linux conGUI

controllo sulle fasi di lavoro. Un programma per la gestione dei contributi, oggi sarebbe implementato mediante programmazione orientata agli oggetti; in questo modo l'utente avrebbe la facoltà di decidere di compilare i moduli nell'ordine che ritiene più opportuno, richiamare wizard in ogni momento, importare informazioni da altre fonti, lasciare campi vuoti, chiedere l'esecuzione di calcoli sui moduli ancora non completamente compilati e creare diversi tipi di output, stampando i risultati a video, su carta o inviandoli mediante la rete ad un server centrale. È importante sottolineare che sia i programmi realizzati con programmazione procedurale che orientata agli oggetti sono in grado di raggiungere gli stessi risultati finali, ma la differenza consiste nel come l'utente interagisce con il programma.

Il miglior programma implementato con un linguaggio orientati agli oggetti ha al centro della sua attenzione le esigenze dell'utente che lo dovrà utilizzare. Un buon programma eccellerà se lo sviluppatore riuscirà prima a capire i modelli mentali dell'utente, poi a creare un modello che li rispecchi ed infine rendere questo modello usufruibile da parte dell'utente. Nel caso del programma per i contributi, l'utente alle prese con la compilazione del familiare modello 740, se si trovasse in un lato le tabelle di calcolo per le fasce di reddito sicuramente apprezzerrebbe questa scelta dello sviluppatore in quanto il programma gli risulterebbe più intuitivo del previsto e particolarmente semplice da usare. In questo caso avremmo realizzato un'interfaccia grafica non solo pienamente compatibile con i modelli mentali dell'utente, ma in grado di stupirlo grazie ad un'estrema semplicità di utilizzo. Ha senso quindi chiedersi come i comandi esposti tramite menu e bottoni debbano essere organizzati, strutturati e raggruppati.

Un'interfaccia grafica 2D non rappresenta lettere e numeri avvalendosi di matrici fisse o di uno o più set di caratteri, ma opera costruendo un layout di pixel indipendenti l'uno dall'altro. In altre parole dal computer al monitor non

viaggiano byte che rappresentano ciascuno un carattere che viene poi interpretato e visualizzato, ma semplici punti da visualizzare.

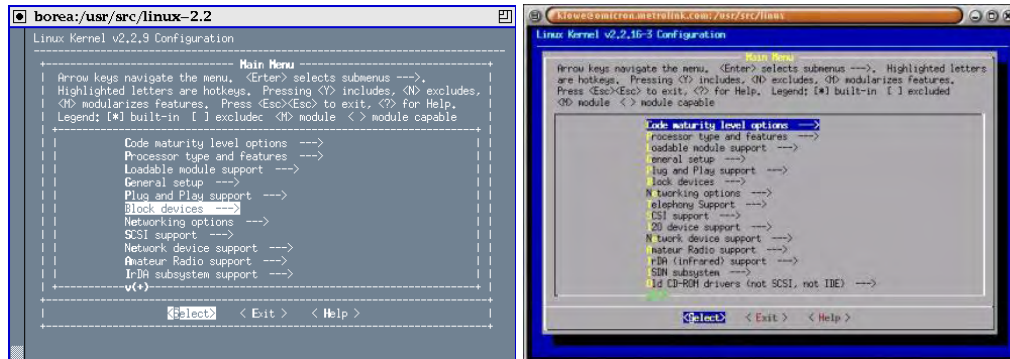


Figura 1.5: esempio di interfaccia grafica front-end

Ogni punto avrà poi le sue coordinate di visualizzazione e i suoi attributi, che possono essere di luminosità, di lampeggiamento, di colore etc. I punti disegnano dunque qualsiasi traccia, da una lettera dell'alfabeto ad una serie di linee, di fondi, di immagini, ecc. È chiaro che l'impegno per disegnare su uno schermo è ben diverso da quello di comunicare dei dati che già rappresentino dei caratteri.

Ciò fa capire subito una cosa: perchè ci siano voluti molti anni di evoluzione prima di arrivare ad una vera e propria interfaccia grafica.

Prima che compaiano le reali interfacce grafiche nasceranno degli ingegnosi compromessi: ovvero interfacce composte non di singoli pixel, ma che sfruttando un set di caratteri particolarmente ampio e contenente anche alcuni piccoli simboli grafici, come rette, angoli, frecce, faccine, ecc. che saranno in grado di dare l'illusione della grafica. Queste interfacce prendono il nome di interfacce grafiche front-end, ma si tratta pur sempre di interfacce di solo testo e non d'interfacce grafiche nel vero senso del termine (vedi figura 1.5).

In una videata come in figura 1.4, ad esempio, una semplice icona non è visualizzabile. È comunque un primo passo verso la facilitazione operativa.

La caratteristica più rilevante di una GUI è lo scambio di dati immediato fra diverse applicazioni. Avendo un primo programma in esecuzione in una finestra ed un secondo programma in esecuzione in un'altra finestra (vedi figura 1.6), è

possibile copiare un insieme di dati dal primo ed importarlo nel secondo (copia&incolla).

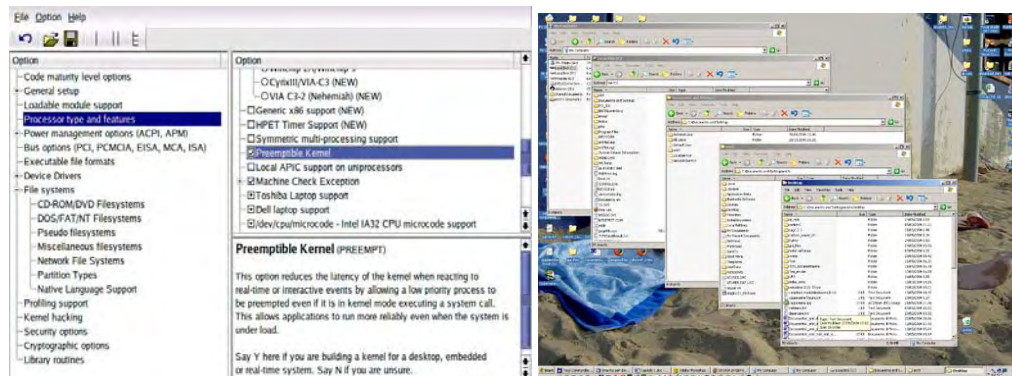


Figura 1.6: Esempio di interfaccia grafica 2-D: a sinistra GUI a finestra singola, a destra GUI con più finestre

Lo scambio di dati fra programmi è possibile anche con i sistemi operativi a riga di comando, ma con programmi GUI l'operazione è immediata e, soprattutto, permette l'importazione di immagini grafiche oltre che di testi. È anche possibile avere un collegamento dinamico fra i documenti, nel quale se i dati cambiano in documento, cambiano anche nelle copie incollate in altri documenti.

Un ambiente grafico non può presentare una sola finestra alla volta, deve comprendere la possibilità di ridimensionare una finestra, spostarla, aprire altre finestre e via di seguito. Ciò significa che oltre ad avere la massima elasticità d'esposizione dei dati, ogni finestra può realmente rappresentare un diverso programma; siamo quindi nella necessità di disporre contemporaneamente di un sistema operativo multitasking.

Le finestre devono presentare una barra con dei comandi essenziali e nel loro interno deve essere possibile visualizzare qualsiasi cosa.

Altra caratteristica implicita nel concetto di finestra è la possibilità di scorrerne il contenuto in verticale e in orizzontale. È implicito nel termine stesso di finestra, un modo per guardare ai propri dati, qualunque sia la loro ampiezza, dunque anche se superiore alle reali dimensioni dello schermo.

L'ultima evoluzione è la realizzazione di una *GUI* in 3D come il file manager della Innolab [INN04] (vedi figura 1.7).

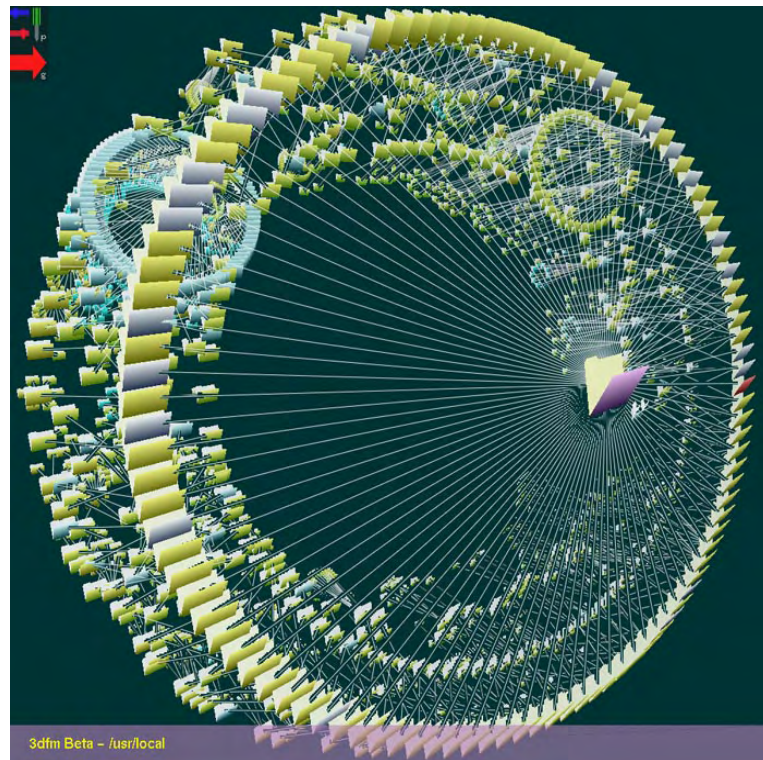


Figura 1.7: file manager 3-D

Si pensi ad un'interfaccia grafica in 3D che organizza i contenuti in base alla loro relazione anziché in base alla loro posizione fisica sul drive. Ogni filo di questa ragnatela lega cartelle che hanno contenuti correlati alla cartella di file aperta (in violetto). Il colore mostra come siano correlate le altre cartelle: la cartella rossa è la generatrice, le cartelle blu sono subdirectory, quelle gialle e grigie sono situate altrove, ma legate in qualche modo alla cartella centrale. Il programma mostra relazioni che non sarebbero chiare in un normale albero bidimensionale. L'utente può ruotare, zoomare, fare panoramiche dell'albero di file per scorgere tutte le possibili correlazioni con criteri variabili.

Le tendenze per i prossimi anni futuri sono la realizzazioni di prototipi di *GUI* a realtà virtuale (figura 1.8); per avere un'idea di come possa essere realizzata una struttura di questo tipo basti pensare al film *Minority Report* [MIN02].



Figura 1.8: tratto dal film Minority Report

1.2.3 Modelli, metafore e analogie

Nel mondo reale gli oggetti hanno caratteristiche fisiche definite, e alcune di queste determinano come l'uomo può utilizzare tale oggetto. Il modo in cui un particolare oggetto può essere usato dipende dallo scopo per cui è nato. Un mouse ad esempio, ha uno o più bottoni, essi forniscono una sola possibilità di utilizzo: possono essere premuti con l'intento di attivare l'evento associato al click; ciò è ovvio, ed è intrinseco nella forma e composizione del mouse stesso, non avendo alette laterali o simili espedienti per far pensare alla possibilità di sollevare un bottone non ci sono molte possibilità che ad un utente venga voglia di provarci. Il mouse è un oggetto che ci offre diverse possibilità di utilizzo, ora ci sembrano scontate, ma questo solo perché a suo tempo qualcuno le ha illustrate o ne siamo entrati a conoscenza prima intuendone il funzionamento e poi verificando con prove pratiche.

Sviluppatori di successo sfruttano la conoscenza che l'utente ha degli oggetti del mondo reale e del loro possibile utilizzo per creare metafore e analogie visive che richi amino ambienti conosciuti e oggetti noti. Il cestino di Windows (figura 1.9) è un classico esempio; nella vita reale per eliminare i documenti li gettiamo dentro il cestino. Se volessimo recuperarli li cercheremmo all'interno del cestino per poi tirarli fuori e svuotando il cestino ci libereremmo in maniera definitiva del suo contenuto.

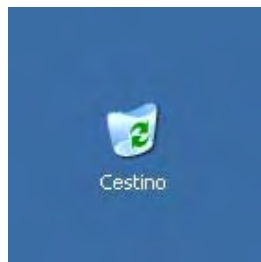


Figura 1.9: cestino di windows.

Naturalmente, nel mondo reale agiamo per manipolazione diretta degli oggetti e di certo non diciamo al cestino di svuotarsi e ci aspettiamo che lo faccia da solo, ma lo prendiamo su e lo andiamo a svuotare da qualche parte. Simili ambienti a manipolazione diretta possono essere riprodotti mediante interfacce grafiche, utilizzando apposite metafore ed analogie.

È fondamentale capire chi sono gli utenti finali del nostro programma; creare collegamenti con oggetti da essi conosciuti nel mondo reale è molto utile per rendere l'interfaccia utente del programma facilmente comprensibile e quindi pienamente sfruttabile da parte dei suoi utenti.

1.2.4 Alcune regole per sviluppare una buona interfaccia utente

- 1) L'utente deve essere in grado di anticipare il comportamento di un simbolo grafico dalle sue proprietà visuali. Con simboli grafici, in questo contesto, si intendono i controlli visuali come bottoni, menu, checkbox,

barre di scorrimento e righelli. Questa prima regola può essere definita come

- *Principio di consistenza a livello grafico.*

Questo principio richiede che i controlli grafici di un interfaccia si comportino con consistenza. Se un bottone risponde al singolo click da parte del tasto sinistro del mouse ogni altro bottone presente deve rispondere allo stesso tipo di evento. I moderni ambienti di sviluppo, come Delphi o Visual Studio .NET, consentono sia l'utilizzo di un set predefinito di controlli sia di poterne creare di propri, e in entrambi i casi la tipologia di eventi a cui possono rispondere è completamente personalizzabile.

- 2) L'utente deve essere in grado di anticipare il comportamento di un programma mediante le conoscenze che ha acquisito utilizzando altri programmi. Questa regola possiamo chiamarla il

- *Principio di consistenza a livello piattaforma.*

La consistenza è importante non solo a livello dei controlli grafici, ma anche ad un livello di astrazione più alto, come i movimenti del mouse, tasti acceleratori, posizionamento di menu, icone e barre degli strumenti. Per ottenere una buona *GUI* è consigliabile seguire i modelli più collaudati della piattaforma per cui si sta sviluppando un'applicazione. Nel caso si sviluppino applicazioni multi-piattaforma è buona regola mantenere consistenza con la piattaforma usata per lo sviluppo.

- 3) Evitare l'apparizione di warning e finestre di dialogo modali con errori gestite dal sistema operativo. Le GUI ben progettate raramente generano warning (figura 1.10) o errori, con eccezione per quelli generati da

problemi hardware, ad esempio errori di lettura del disco rigido o persa connessione dal modem o warning che richiedono il permesso dell'utente per eseguire operazioni potenzialmente pericolose e/o irreversibili. Possibili generici errori, derivati da un non corretto utilizzo dell'interfaccia, come testo inserito nel formato non corretto, omissione di dati richiesti o operazioni eseguite in ordine non consentito, devono essere previsti e gestiti dall'interfaccia stessa. Una buona interfaccia va sviluppata in modo da far capire all'utente come inserire i dati in modo corretto. Warning e finestre di dialogo tipiche del sistema operativo generano sempre sensazioni di disagio nell'utente, il quale si trova di fronte a messaggi d'errori non interpretabili e non sa come comportarsi.

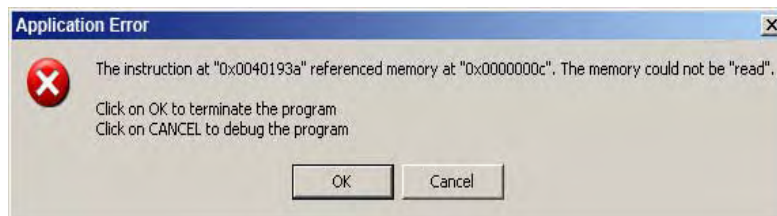


Figura 1.10: esempio di warning

- 4) Fornire un adeguato *feedback*. L'utente deve potersi rendere conto delle conseguenze delle azioni che sta compiendo. Come i controlli di default, quali bottoni o check box, una volta premuti cambiano il proprio stato in modo da indicare all'utente di aver recepito un comando, così qualsiasi altro elemento, compresi quelli creati da zero dal programmatore, che rispondono ad un qualsiasi evento, deve essere realizzato in modo da mostrare all'utente il cambiamento avvenuto. Sono esempi comuni di feedback: le barre di progresso che ci mostrano gli avanzamenti di processi in esecuzione e i mutamenti grafici di bottoni e icone atti ad indicare se sono stati premuti o meno, o anche semplicemente se è possibile premerli e se ci troviamo con il mouse sulla loro area sensibile,



Figura 1.11: esempio di Feedback

la freccia del mouse che diventa una clessidra per farci capire che il programma sta elaborando delle informazioni e dobbiamo intendere (figura 1.11). E' importante progettare propriamente l'interfaccia grafica in modo che anche l'utente meno esperto sia in grado di capire cosa stia facendo il programma in ogni istante, se sta aspettando dati, se li sta elaborando o ha concluso il suo compito.

- 5) Creare un ambiente sicuro per l'esplorazione. Buone interfacce invitano e ricompensano l'esplorazione da parte degli utenti, ed offrono ai novizi l'entusiasmo della ricerca e la soddisfazione delle scoperte effettuate senza aiuto. Alcune interfacce (figura 1.12) stimolano l'utente ad esplorare da soli le funzionalità a disposizione,

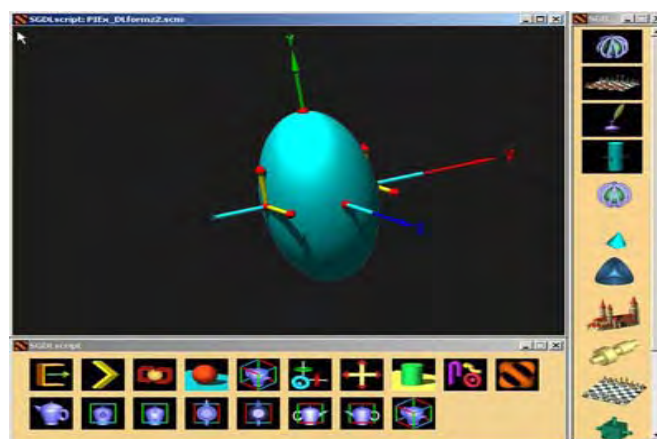


Figura 1.12: esempio di interfaccia grafica che stimola l'utente all'esplorazione.

a volte vengono visualizzati piccoli messaggi con consigli e poi si lascia all'utente il compito di approfondire l'argomento. Con l'implementazione della funzione undo/redo, ovvero la possibilità di annullare un'operazione

ed eventualmente ripristinarla, si invita l'utente ad effettuare tutte le prove che desidera senza doversi preoccupare per le possibili conseguenze negative. Una buona interfaccia fa sì che l'utente si senta competente, cioè in grado di usare tutti gli strumenti a sua disposizione.

- 6) Cercare di rendere l'applicazione comprensibile in maniera autonoma. Le buone applicazioni hanno esaurienti manuali e aiuti in linea in grado di spiegare il funzionamento del programma e le sue caratteristiche nel modo più semplice possibile, spesso utilizzano problemi presi dal mondo reale che l'utente dovrebbe conoscere e ne illustrano la possibile soluzioni. Applicazioni ben sviluppate sono quelle in cui anche l'utente meno esperto raramente ha bisogno di rivolgersi al manuale o all'aiuto in linea. Una buona scelta delle etichette di testo, delle icone e degli altri elementi grafici visualizzati, uniti alla loro disposizione sullo schermo aiutano notevolmente l'utente a capirne il funzionamento in fase di testing, prima di rilasciare le applicazioni sul mercato, permettono di avere valide indicazioni sulla comprensione del funzionamento dell'interfaccia e quindi dell'applicazione, da parte degli utenti alla quale essa è destinata.

Il massimo risultato possibile è progettare un'interfaccia utente che non abbia bisogno di alcuna spiegazione, ma che sia talmente ben organizzata da risultare familiare anche a chi non l'ha mai vista. Ovviamente l'interfaccia utente di un'applicazione specifica, per un particolare genere di utenza, non dovrà essere necessariamente semplice anche per gli utenti comuni.

Se stiamo realizzando un'applicazione destinata all'uso ospedaliero (Figura 1.13) non ci preoccupiamo certo di sviluppare un'interfaccia utente che sia intuitiva per un architetto, quello che ci interessa è che lo sia per un medico.

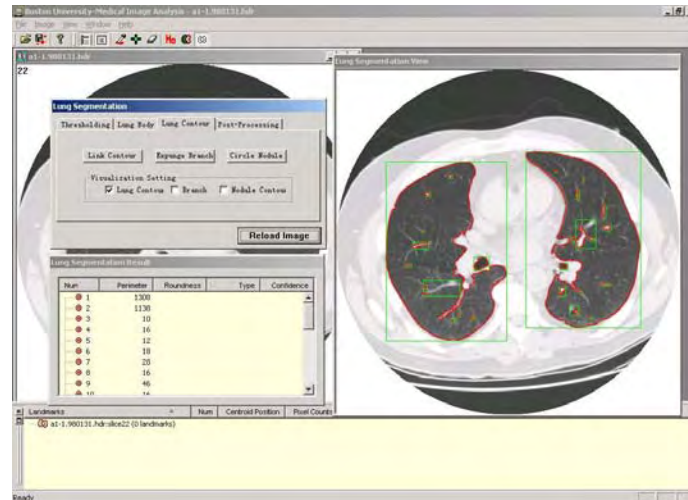


Figura 1.13: esempio di interfaccia grafica ben organizzata.

- 7) Usare moderatamente suoni, colori, animazioni e clip multimediali. Suoni, colori, animazioni e presentazioni multimediali sono appropriate per applicazioni dedicate all'intrattenimento o all'educazione, ma è difficile farne un efficace uso in altri contesti. E' bene seguire tali indicazioni, e ricordare di non usare elementi audio o grafici (basati sulle differenze cromatiche) come *unico* mezzo per comunicare concetti, infatti tra gli utenti è facile che vi siano persone con problemi nella distinzione dei colori e/o nella percezione dei suoni. Se noi basiamo la comunicazione con l'utente solo su elementi non percepibili in ugual modo da tutta l'utenza, rischiamo di escluderne una buona parte; quindi, è consigliato utilizzare effetti sonori e particolari animazioni con il solo scopo di sottolineare concetti già espressi con i tipici mezzi a disposizione.

- 8) Aiutare l'utente a personalizzare e preservare il proprio ambiente di lavoro, prende il nome di customizzazione. Se l'applicazione è diretta a singoli utenti, per memorizzare le impostazioni d'ambiente personalizzate può essere sufficiente un registro in cui andare a salvare la posizione e la dimensione della finestra dell'applicazione così come l'aveva impostata l'utente prima che la chiudesse. Diverso è se le applicazioni sono pensate

per supportare più utenti o installazioni su più macchine con impostazioni differenti, pertanto devono prendere in considerazione le molteplici informazioni da memorizzare; ad esempio un'applicazione può avere comportamenti differenti in base alle diverse specifiche video dei computer su cui è installata, o più semplicemente, deve permettere l'immagazzinamento delle singole impostazioni di ogni utente che la utilizzano. È bene sapere che l'aspetto di un'applicazione può cambiare notevolmente in base alle specifiche dello schermo su cui è visualizzata, per questo sarebbe bene elaborare una versione ridotta dell'interfaccia, in modo che possa essere correttamente visualizzabile sulla quasi totalità degli schermi a disposizione degli utenti (Figura 1.14). Infine, è consigliato, nonché utile e apprezzato, dare all'utente la facoltà di scegliere tra più di una impostazione grafica di base. Fornendo quelle che vengono chiamate *skin* (traduzione letterale: pelle) gli utenti hanno la possibilità di personalizzare l'aspetto grafico della proprio applicazione, e quelli più esperti possono anche crearsi skins completamente personalizzate.

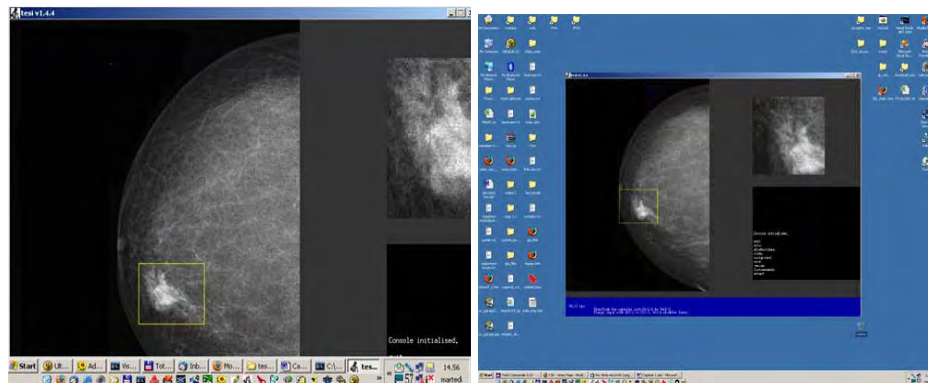


Figura 1.14: visualizzazione di console su un monitor a 1600x1200 di risoluzione utilizzando una risoluzione a sinistra 800x600, a destra 1600x1200

- 9) Evitare l'uso di finestre con comportamenti modali. Finestre di questo tipo obbligano l'utente a eseguire le operazioni richieste in un ordine specifico non modificabile, questo diminuisce l'intuitività dell'interfaccia

e ne modifica il naturale comportamento a cui l'utente è abituato. Vi sono comunque casi in cui è utile, se non necessario, l'impiego di finestre modali, ad esempio per implementare un wizard (Figura 1.15) che guidi l'utente ad una prima configurazione della applicazione. In questi casi mentre vengono modificate le opzioni relative all'applicazione non è possibile far navigare liberamente l'utente fra le altre finestre aperte in precedenza, esse potrebbero essere soggette a cambiamenti, nell'aspetto e nei contenuti, dovuti alle nuove preferenze impostate. Warning e messaggi d'errore sono tipicamente modali, forzano l'utente a leggere un importante messaggio e poi a premere un bottone, per confermare di aver letto il messaggio, prima di poter tornare al consueto utilizzo dell'applicazione. I migliori comportamenti modali sono quelli che risultano impercettibili, cioè quelli che vengono fuori come naturale conseguenza dell'impiego di metafore col mondo reale.

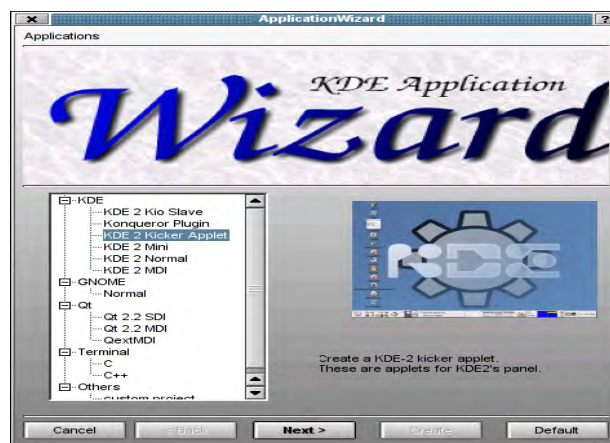


Figura 1.15: esempio di wizard.

Per fare un esempio preso dal mondo reale, se stiamo cambiando una gomma alla nostra autovettura non ci aspettiamo certo di poterla usare mentre stiamo ancora completando questa operazione; il comportamento delle finestre modali per l'impostazione delle preferenze di sistema funziona allo stesso modo, nel momento in cui stiamo modificando il comportamento della nostra applicazione non ci aspettiamo di poterla

usare, ma attenderemo di aver finito di definire la nuova configurazione. In definitiva, se la nostra interfaccia richiede assolutamente controlli modali è importante cercare di legarli a forti metafore col mondo reale, in modo che l'utente capisca da solo il motivo di questo comportamento da parte dell'applicazione e che si ritrovi in un ambiente naturale governato da azioni e reazioni per lui ovvie e prevedibili.

10) Sviluppare l'interfaccia in modo che l'utente possa dedicarsi al proprio lavoro senza avere l'attenzione distolta dall'interfaccia che sta utilizzando: questo lo possiamo chiamare il *Principio della trasparenza*. Un'interfaccia è detta trasparente quando l'utente pone la propria attenzione oltre l'interfaccia stessa, in quanto è giustamente diretta allo scopo di fondo per cui l'applicazione è stata creata. Ciò è il risultato di diversi fattori,

- a) primo fra tutti un layout grafico che disponga gli strumenti e le informazioni esattamente dove l'utente si aspetta di trovarli;
- b) icone ed etichette di testo che siano chiare e al tempo stesso significative;
- c) metafore che siano facili da riconoscere, imparare, memorizzare, e utilizzare.

Quando l'utente non si deve preoccupare più del come inserire i dati in suo possesso, e come aspettarsi dei risultati da parte dell'applicazione, in quanto si muove in maniera disinvolta in un ambiente a lui familiare, significa che l'interfaccia sviluppata ha raggiunto il suo scopo.

Scegliere buone metafore e seguire il più possibile i dieci principi appena elencati è un ottimo punto di partenza per lo sviluppo di una valida interfaccia utente, ma il modo migliore per essere certi di realizzare una buona interfaccia è coinvolgere il più possibile gli utenti durante la fase di sviluppo. Effettuare test sui futuri utenti ci permette di evitare la costruzione di interfacce che ci

potrebbero sembrare estremamente efficienti, ma che all'utente potrebbe risultare poco pratiche e di difficile apprendimento (figura 1.16).



Figura 1.16: esempio di come non dovrebbe essere realizzata l'interfacciagrica.

Capitolo 2

Multi-piattaforma

In questo capitolo verrà mostrato quali siano i vantaggi del multi-piattaforma, verranno introdotte le GUI per applicazioni mediche, e verranno descritte quali siano le indicazioni per rendere un'applicazione funzionale.

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* [OPS04], divenuto popolare soprattutto grazie alla diffusione di Internet, che è stato ed è tutt'ora 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 GPL², LGPL³, 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 maggior numero di utenti. Molti sono gli esempi di successo di questo modello: prodotti come OpenOffice [OPO04] 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.

² GPL: *General Public License*, garantisce la libertà di condividere e modificare il software libero.

³ LGPL: *Lesser General Public License*, una licenza pubblica meno generale.

Consideriamo il caso di una piattaforma hardware e prendiamo per esempio il sistema operativo Linux; inizialmente si compilava solamente su processori Intel [INT04], 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 [ALP02], Arm [ARM04], PowerPC [PPC04], etc.

Consideriamo ora un altro caso di software e prendiamo come esempio Apache [APC99].

Inizialmente era un progetto solamente per sistemi Unix-like, il che comprendeva una grande quantità di sistemi: non solo Linux (e di conseguenza tutte le piattaforme hardware su cui gira), ma anche altri sistemi operativi Unix-like come Solaris [SOL04], AIX [AIX04], FreeBSD [BSD04], etc. Ma in seguito si è evoluto ed ha conquistato anche sistemi operativi non Unix-like, come Windows, AS/400 e OpenVMS. Nulla di sorprendente se non viene citato il MacOS in quanto oggi è una variante di Unix e quindi va contato tra le piattaforme Unix-like native di Apache.

Altro esempio di grande successo è Mozilla [MOZ04]: è stato sviluppato per funzionare fin dall'inizio sulle piattaforme client principali come Windows, MacOS e Linux, ma esistono port⁴ ad altri sistemi oramai scomparsi come OS/2 e BeOS; ovviamente non ha alcun senso portarlo su sistemi senza interfaccia grafica come l' AS/400.

Quando un software viene sviluppato, nasce in una piattaforma, solitamente quella usata dalla sviluppatore. Ma il software ha degli utenti, e non è detto che anche loro utilizzino la stessa piattaforma del programmatore. Data la presenza del sorgente e la grande flessibilità del software, è naturale che l'atteggiamento degli utenti, a loro volta programmatori, sia di tentare di modificare il software affinché possa girare anche su altre piattaforme.

⁴ Port: termine che indica la conversione di un sorgente nato per un sistema in un altro.

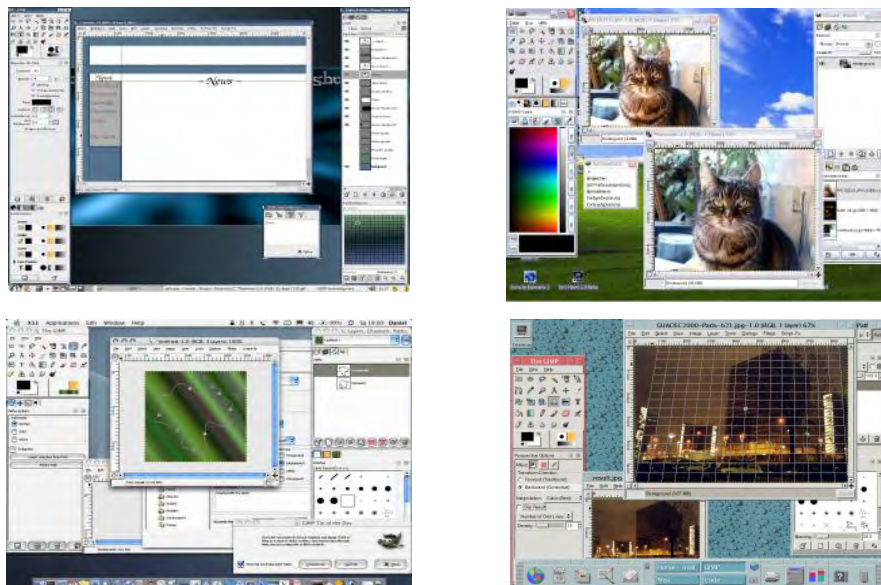


Figura 2.1: In senso orario, programma Gimp per le piattaforme Linux, Windows, Machintosh, Solaris

Per esempio il programma grafico *Gimp* [GIM04] (figura 2.1) e il programma per il disegno di diagrammi *Dia* [DIA04] nati per sistemi Unix-like, ora sono disponibili anche per ambiente Windows.

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.

Per esempio, fino a qualche anno fa si programmava per Windows o X11, 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 [PYT04], che supportano lo sviluppo di applicazioni funzionanti in maniera pressoché identica su più sistemi operativi (figura2.2).



Figura 2.2: Gerarchia di astrazione logica per GUI multi-piattaforma

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.

2.1 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. Infatti nessun programma concepito per un particolare sistema operativo può essere avviato su una macchina di tipo differente: ad esempio un file *.exe* di Windows, per esempio, è un file privo di significato su Linux. 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.

Tra i formati più conosciuti vi sono il DOS⁵, Win32⁶, Elf⁷. Di questi ad esempio si consideri il dos, nell'header del formato sono inserite le informazioni sui segmenti dati, stack, programma. Il formato Elf invece è quello standard degli Unix moderni.

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;

⁵ DOS: Disk Operating System

⁶ Win32: Windows 32 Bit

⁷ Elf: Executable and Linking Format

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* (*scrivi una volta, esegui ovunque*) [WRA04].

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

1. Il tempo addizionale 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:

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 (viste nel capitolo precedente) 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.

2.2 Wrapper vs Emulated

La maggior parte delle interfacce utenti sono sviluppate utilizzando delle componenti visuali, conosciute meglio come *widgets* (o controlli), che sono raccolti insieme nelle librerie che fanno riferimento alle *user interface toolkits*. La toolkit più conosciuta è la MFC [MFC04] (Microsoft Foundation Classes), usata, come si intuisce dal nome, per sviluppare applicazioni Windows, ma ne esistono altre come la JFC [JFC04] (Java Foundation Classes)

Due sono gli approcci che rendono una piattaforma *GUI* dal punto di vista funzionale indipendente (Figura 2.3):

1. l'approccio wrapper;
2. l'approccio emulato.

L'approccio wrapper sposta i widgets di un sistema nativo in un layer astratto che fornisce una funzionalità comune tra i sistemi diversi.

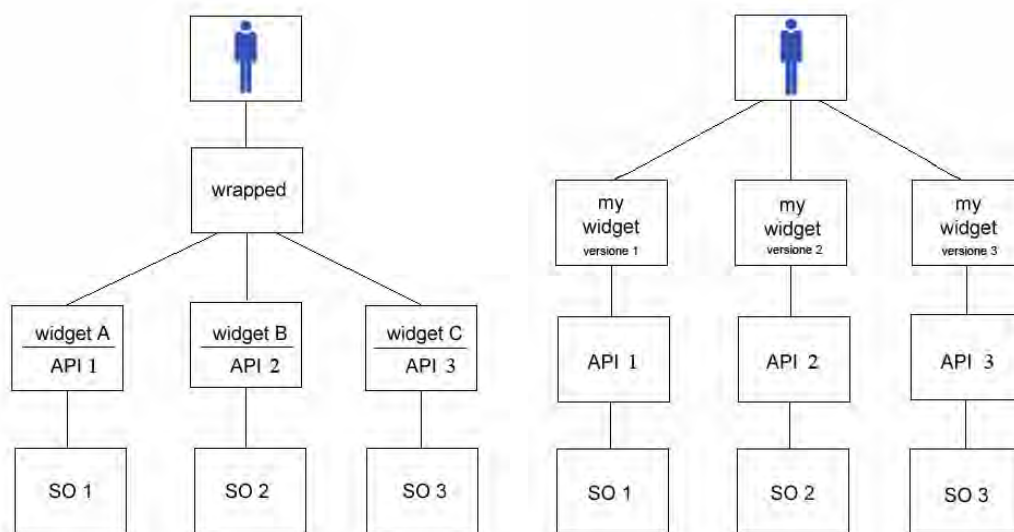


Figura 2.3: Esempio wrapped a sinistra, e di emulato a destra

L'approccio emulato, o anche chiamato *puro*, intercetta semplicemente le così dette *drawing calls* e in seguito usa quelle chiamate per implementare i propri widgets.

I wrappers sono più semplici da programmare in quanto non c'è bisogno di scrivere i widgets.

Comunque i wrappers perdono flessibilità poiché possono solo fornire un minimo comune denominatore dei widgets disponibili nei vari SO, e questo significa fare in modo che l'applicazione non si avvantaggi di speciali caratteristiche della piattaforma su cui gira. Questa metodologia di lavoro, porta ad avere una applicazione che si comporta sempre allo stesso modo, indipendentemente dalla piattaforma. Spesso applicazioni di questo tipo hanno comportamenti simili alle applicazioni dell'ambiente su cui sono state sviluppate, ma possono rivelarsi difficili da comprendere su altre piattaforme. In più non possono essere estesi e non permettono di sfruttare l'intera potenza della piattaforma

Sta al programmatore scegliere quale approccio seguire. Prima ancora di affrontare la questione di portabilità questi deve scegliere una libreria per la *GUI* adatta allo scopo.

2.3 Introduzione alle GUI per applicazioni mediche

La multimedialità intrinseca delle informazioni utilizzabili nel contesto medico quali immagini 2D, 3D, monocromatiche, a colori, video documenti, etc., e le consolidate attività operative del personale sanitario, costituiscono il punto di partenza per la progettazione di una *GUI* per applicazioni mediche. L'analisi di questi requisiti è essenziale se si considera che i vantaggi operativi introdotti dalle tecnologie vengono spesso limitati dal loro mancato o parziale utilizzo dovuto all'inerzia mentale da parte dell'utenza.

Una volta assimilati i principi di creazione delle *GUI*, si tratta di adattarli alla sensibilità medica: a tal fine risulta particolarmente utile, durante la progettazione, la partecipazione di personale medico. Questo al fine di individuare le eventuali limitazioni della *GUI* proposta e sviluppare delle linee guida da seguire nella progettazione di applicazioni per uso specificatamente medico.

Le aziende produttrici di software medicale impiegano mesi per sviluppare una preliminare interfaccia utente e spesso proprio la progettazione di questa è considerata come il tallone di Achille del prodotto, soprattutto se la concorrenza ha sviluppato e prodotto sul mercato un'applicazione più *user-friendly*.

La caratteristica del software percepita è, quindi, l'interfaccia con cui si presenta all'utente. Questa considerazione è ancora più vera se applicata al contesto medicale. Basta semplicemente pensare che l'utente non è un professionista informatico ma un medico il quale deve considerare l'applicazione come una estensione delle sue capacità ed è indispensabile che questa sia logica e semplice nello stesso tempo, per permettere al medico di svolgere al meglio il proprio lavoro.

Per realizzare una *GUI* in questo ambito è necessario conoscere a priori l'obiettivo, individuando le linee guida ottimali per la realizzazione di interfacce

in campo medico, e questo studiando le modalità di presentazione dei dati e le interazioni uomo-macchina.

La maggior parte dei display medici sono talmente pieni di informazioni e comandi da lasciare pochissimo spazio rimanente. Tale spazio è importante in una interfaccia utente, in quanto contribuisce a separare le informazioni in gruppi logici e offre una sensazione di riposo all'occhio dell'utente, altrimenti soffocato dalla presenza di immagini visive. Un'eccessiva presenza di immagini può addirittura mettere a disagio il personale come infermieri, tecnici e medici rendendo difficile il reperimento delle informazioni.

Per tali motivi il progettista deve far attenzione a cosa visualizzare sullo schermo, eliminando tutte le informazioni estranee, pertanto può:

- Aggiungere le informazioni secondarie a richiesta attraverso l'utilizzo di pop-up o riallocandole in altri schermi;
- Ridurre il formato dei grafici ;
- Usare grafici più semplici (sostituendo le icone 3-D con uno stile più sobrio);
- Usare lo spazio vuoto piuttosto che le linee per separare i contenuti;
- Ridurre la quantità di testo dichiarando le cose più semplici;
- Utilizzare semplici caratteri;
- Fare attenzione alle etichette ridondanti;
- Utilizzare un linguaggio semplice
- Progettare icone più armoniose
- Fare attenzione alle ridondanze

Di seguito vengono proposti in dettaglio alcuni di questi aspetti.

2.3.1 Fornire le indicazioni e le opzioni di navigazione.

Può capitare che spostandosi da una schermata all'altra in un'applicazione medica, l'utente possa addirittura perdersi, come qualcuno che, viaggiando in una città straniera ha i segnali stradali in un'altra lingua o non ha la strada che gli serve per arrivare in una certa direzione. A volte questa problema è causato dal fatto che l'utente non sa in quale schermata si trovi. Si pensi al medico che

deve fissare dei livelli di allarme per il monitoraggio, per esempio, della pressione sanguigna arteriosa: si può presentare la situazione in cui ci si perda in una gerarchia complessa di opzioni come quelle di settaggio che variano dal controllo respiratorio al controllo dei globuli rossi. Altre volte può capitare che l'utente non comprenda come muoversi all'interno dell'interfaccia, perché i metodi di controllo non sono chiari. Un' infermiera per esempio può trovare il modo di configurare una schermata, ma non è in grado di identificare quali siano i pulsanti per tornare indietro o come lasciare una schermata, dopo aver fatto un giusto settaggio. Utile è disporre di titoli significativi, come le opzioni di navigazioni e di controllo quali *vai al menù principale*, *vai indietro*, *precedente*, *avanti*, *aiuto*, che dovrebbero essere raggruppati insieme sempre nella stessa posizione, in modo che l'utente riesca facilmente ad individuare il comando voluto in ogni schermata, senza aver timore di creare un danno irreparabile.

2.3.2 Ricorrere all'utilizzo di una griglia.

Alcune interfacce mediche assomigliano ad una scacchiera da gioco in una partita di scacchi; dopo aver fatto una decina di mosse, i pezzi si trovano tutti mescolati senza una disposizione ordinata. Invece risulta fondamentale applicare una certa armonia agli elementi visivi.

Gli elementi allineati conferiscono una idea di minor complessità alla struttura, inoltre l'occhio umano riesce a trovare con maggiore facilità le informazioni quando esamina un percorso dritto piuttosto che irregolare o discontinuo.

Centred	Left justified
This is a sample of text that illustrates the appearance of centered text. This is a sample of text used to illustrate the appearance of centered text.	This is a sample of text that illustrates the appearance of left-justified text. This is a sample of text that illustrates the appearance of left-justified text.

Figura 2.4: Il testo a sinistra disturba la lettura, mentre il testo a destra è più armonioso

È per questo motivo che programmatori di una certa esperienza adottano il testo a sinistra o lo giustificano invece di formattarlo al centro. Il testo centrato genera

un effetto di *riverbank*⁸ (figura 2.4) che rende il lavoro dell'occhio più difficile nel localizzare la linea successiva di inizio riga.

Lo sforzo iniziale di inserimento in modo ordinato degli elementi viene ripagato in termini di richiamo visuale e di semplicità.

È altresì importante allineare gli oggetti ad una distanza fissa rispetto la griglia.

La simmetria degli oggetti deve essere creata a partire dall'asse verticale dell'interfaccia grafica, in modo da generare un'armonia visiva all'utente. Vale la pena notare che una simmetria troppo perfetta può sembrare noiosa e stancante allo stesso tempo.

Inoltre c'è un altro aspetto importante da tener presente: il colore.

I programmatori suggeriscono di limitare la gamma di colori usati. Lo sfondo e gli elementi principali dello schermo dovrebbero avere una varietà di colore che varia dai tre ai cinque colori, compresa la tonalità di grigio; non solo, è altresì apprezzabile selezionare con attenzione i colori appropriati, in modo tale che siano coerenti con le convenzioni mediche.

2.3.3 Uso di semplici caratteri

L'interfaccia utente deve basarsi su alcune regole tipografiche, regole necessarie affinché l'utente possa visualizzare sullo schermo contenuti facili da leggere. Per questo motivo è indicato utilizzare un solo tipo di carattere con alcune diverse grandezze come il size 12, 18, e 24 (figura 2.5).

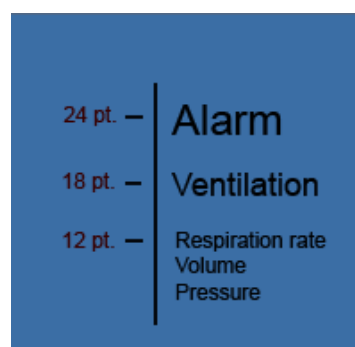



Figura 2.5: Alcune size dello stesso carattere

⁸ Riverbank: Il suo uso indica un effetto frastagliato, di disordine.

Un altro modo per semplificare il testo è eliminare l'eccessivo uso di testo sottolineato, testo in grassetto o testo italico, questo perché i diversi tools ora in commercio rendono facile l'utilizzo di queste particolari enfasi e ne può risultare quindi un eccessivo uso.

2.3.4 Attenzione a etichette ridondanti

Etichette ridondanti possono congestionare lo schermo, aumentando così le informazioni inutili e rallentando lo scorrimento della videata.



HR	Low	40	High	160
RR	Low	4	High	20
PA	Low	10	High	60
Art	Low	90	High	180

	Low	High
HR	40	160
RR	4	20
PA	10	60
Art	90	180

Figura 2.6: A sinistra un esempio da evitare, a destra un corretto uso delle label

Non solo, ma una eccessiva visualizzazione di immagini può mascherare i dettagli più salienti della schermata (figura 2.6). L'utilizzo regolare di etichette gerarchiche può generare spazio e velocità di visualizzazione.

2.3.5 Utilizzo di un linguaggio semplice

Le interfacce utente dei dispositivi medici sono spesso caratterizzate da un eccessivo utilizzo di parole e frasi complesse, in quanto si fa spesso uso di parole tecniche. I programmatori di oggi sviluppano i loro prodotti tenendo presente che semplificare la comunicazione è utile non solo da un punto di vista professionale, ma anche per aumentare il numero di pazienti e di personale ad usare particolari dispositivi a casa propria, pur non avendo una ampia educazione o preparazione.

2.3.6 Rifinire e armonizzare le icone

Ultimamente, gran parte dei dispositivi medici, in particolare quelli con uso di display ad alta risoluzione (per esempio VGA o superiori), presentano opzioni

funzionali, come la calibrazione del device, settaggio degli alarm o guide utenti, in forma di icone (figura 2.7).



Figura 2.7: Icone di Windows Xp a sinistra, Windows 95 a destra.

Questo porta le industrie che producono software medico ad avere, per l'interfaccia utente, una propria iconografia in modo tale da avere una certa assomiglianza di icone.

Sfortunatamente la produzione di dispositivi medici può trovare difficile combinare lo sviluppo di software in termini di investimenti in qualità di icone. Le icone che appaiono nelle diverse applicazioni sono molto raffinate e testate da industrie come Microsoft, Apple, Oracle le quali utilizzano ingegneri e disegnatori grafici professionali.

Per le compagnie nel settore medico non è così, hanno meno tempo e soldi da dedicare al disegno di icone. I produttori di dispositivi medici dovrebbero essere consapevoli, tuttavia, che un investimento limitato nella qualità delle icone può portare all'abbandono del software da parte dell'utente, sempre più motivato a spostarsi verso grafiche più accattivanti.

I seguenti passaggi potrebbero essere presi per massimizzare la comprensione delle icone e dare una certa familiarità ad esse:

- Sviluppare un set limitato di icone che rappresentano i nomi (ad esempio oggetti come siringhe, pazienti...), eliminare i casi dove gli elementi rappresentano la stessa cosa;

- Semplificare le icone eliminando i dettagli che potrebbero portare alla incomprensione e alla confusione e accentuare gli aspetti più significativi dell'oggetto o dell'azione, come se si stesse creando una caricatura;
- Costruire icone della stessa grandezza come 40x40 pixel;
- Far eseguire all'utente dei test in modo da assicurare che due icone non siano così simili da essere confuse fra di loro;
- Impiegare lo stesso stile per le icone che hanno uno scopo simile;
- Rinforzare le icone con etichette di testo o con i cosiddetti tool tips che appaiono, per esempio, quando l'icona è selezionata o evidenziata.

2.3.7 Eliminare le inconsistenze

Le creazioni di inconsistenze sono spesso dannose per l'aspetto dell'interfaccia utente di alcuni dispositivi medici e la loro usabilità e possono compromettere la sicurezza. Per esempio gli utenti-finali potrebbero essere confusi, o annoiati dall'utilizzo del colore rosso per comunicare sia dati critici e non. Per prevenire le inconsistenze, potrebbe essere creata una guida allo stile. Guide standard alle *GUI* come Microsoft's User Interface Design Guidelines per Windows 95⁹ sono pratiche guide da utilizzare come prototipi alle guide di specifiche applicazioni mediche. Queste guide non dovrebbero essere intese come documenti raffinati, ma piuttosto come una collezione organizzata di note e regole che assicurano consistenza pratica, nelle fasi di un primo progetto. Come una funzione di analisi e testing, la produzione di una interfaccia utente può eventualmente divenire in seguito più raffinata e aggiornata. Infine la guida può essere integrata nelle specifiche di progettazione finali.

Il successo di molti dispositivi medici, è assai legato alla qualità dell'interfaccia utente. Questa è particolarmente vero nei casi in cui c'è una forte competizione e la tecnologia associata è più o meno matura, facendo dell'interfaccia utente

⁹ Microsoft's User Interface Design Guidelines: Guida di riferimento Microsoft per le interfacce grafiche

una essenziale fattore di differenziazione del prodotto (un valore aggiunto). Anche il fattore sicurezza è legato alla qualità dell'interfaccia utente, in quanto una sua erronea progettazione e implementazione può indurre direttamente o indirettamente ad errori, con conseguenze a volte anche severe che possono portare a danni per il paziente.

Nel capitolo seguente verranno analizzate alcune delle librerie per *GUI* multi-piattaforma più utilizzate.

Capitolo Terzo

Gli strumenti per lo sviluppo multi-piattaforma

In questo capitolo verranno introdotti alcuni strumenti di sviluppo per la multi-piattaforma: da strumenti noti a tutti gli utenti come Java a strumenti conosciuti solo da programmatori esperti come GTK, Qt, wxWindows e SDL. Particolare attenzione verrà posta su quest'ultima libreria e verrà spiegato perché sia così importante il suo utilizzo per lo sviluppo di applicazioni multi-piattaforma.

3.1 Java

Java è un linguaggio molto usato in ambiente Windows e Unix, ma non è un linguaggio Open Source. È stato annunciato che lo sarebbe diventato, ma attualmente non lo è. Le ragioni di questa scelta da parte della casa produttrice Sun [SUN04] sono molte e condivisibili (legate tra l'altro alla difesa dalla Microsoft).

Per Java è disponibile, oltre alla versione per Windows, il port per Linux dell'intero ambiente di sviluppo: *Java Development Kit – JDK* sviluppato dal team blackdown [BLD04]. Questo port è presente nelle versioni principali usate in sistemi di produzione, correntemente fino alla 1.4.1. c'è da precisare che Java per Linux non significa per altri sistemi, come per esempio FreeBSD, anche se ci sono sforzi in tale direzione. Java è oggi un software molto complesso, che richiede numerose ottimizzazioni per funzionare bene in uno specifico sistema operativo.

Il JDK per Linux ha avuto l'approvazione di Sun che ha fornito i test di conformità per consentire al JDK per Linux di fregiarsi del logo di Java Compatibile. Inoltre esiste anche una versione ufficiale pubblicata da Sun che usa parti di codice prese proprio dal porting per Java del gruppo Blackdown.

Il JDK, comunque, non è il solo Java per Linux in circolazione, ne esistono versioni completamente Open Source, la più completa delle quali è la versione chiamata Kaffe¹⁰, che è una riscrittura completa da zero di Java. L'implementazione raggiunge, però, a malapena il supporto della libreria della versione 1.1: pressappoco un decimo dell'attuale versione 1.4.x.

Java è composto da due parti:

1. virtual machine
2. librerie

La virtual machine è relativamente facile da realizzare, e ne esistono già diverse. Il vero problema, almeno nell'ottica di avere un sistema completamente Open Source, è avere le enormi librerie standard di Java come Open Source. Un pò

¹⁰ Kaffe: è un compilatore di sorgenti Java e un interprete di compilati in formato Java bytecode.

come Wine¹¹, il cui compito è quello di ricostruire tutte le API di Windows. Per far fronte a ciò è nato un progetto GNU. Il progetto è conosciuto come Classpath, per la realizzazione ex-novo di una libreria di classi free software compatibile con quella Sun, che vada oltre il limite delle librerie, compatibili solo con Java 1.1 di Kaffe.

Il progetto Classpath è già andato al di là del JDK 1.1; correntemente ha implementato in parte molte librerie del JDK 1.2, 1.3 e 1.4.

La situazione attuale è incerta, a volte si possono usare certe virtual machine, a volte altre, tipico di un programma in sviluppo che non ha ancora finalizzato i suoi obiettivi. Per esempio al momento il sito ufficiale dichiara che Classpath non funziona con Kaffe.

C'è da ricordare le *servlet*. Sun ha definito le *servlet* come lo standard per l'interfacciamento di Java ai server Web. A fronte di ciò sono nati vari progetti per lo sviluppo Open Source di queste specifiche; il più conosciuto è Tomcat, un servlet engine specializzato per il collegamento con Apache.

Tecnicamente Tomcat è un server autonomo interamente sviluppato in Java con il quale Apache comunica utilizzando un suo protocollo, implementato come modulo per Apache.

3.1.1 Il bytecode e la Java Virtual Machine

Quando un programma Java viene compilato, il risultato dell'operazione non è, come in C/C++, un codice eseguibile, ma ciò che si ottiene è un *bytecode*, ossia una codifica intermedia. Il bytecode a differenza del codice eseguibile, è indipendente. Preso così com'è è del tutto inutile. Ad eseguire il bytecode ci pensa un apposito ambiente di runtime, chiamato *Java Virtual Machine – JVM*. Questo componente deve essere presente nel sistema operativo sul quale si vuole eseguire del software Java (vedi figura 3.1).

¹¹ Wine: è un'implementazione Open Source delle Windows API al di sopra di X e Unix. Fornisce sia un toolkit di sviluppo per il porting dei sorgenti di Windows, sia un program loader, consentendo a diversi file binari di windows di funzionare sotto sistemi Unix x86-based.

Java Virtual Machine

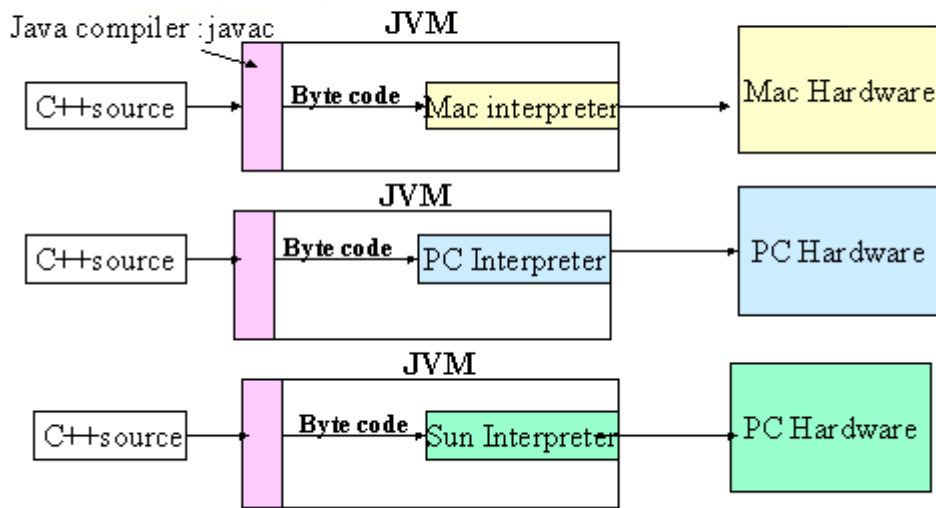


Figura 3.1: Esempio di funzionamento della JVM

La macchina virtuale, oltre ad eseguire e supervisionare il bytecode, offre tutti i servizi di cui un programma può aver bisogno. Un codice eseguibile deve necessariamente interfacciarsi con il sistema operativo. Un programma Java, invece, richiede i servizi direttamente alla macchina virtuale. Mentre ciascuna piattaforma dispone di funzionamenti differenti. Tutte le Java Virtual Machine offrono al programmatore gli stessi identici strumenti: il codice necessario per aprire una finestra passando attraverso la Java Virtual Machine di Windows, per esempio, è esattamente lo stesso richiesto alla macchina virtuale di Linux.

3.1.2 I vantaggi e gli svantaggi di Java

Oltre alle caratteristiche di portabilità che facilitano lo sviluppo del software compatibile con più piattaforme, Java è:

1. *Sicuro*. L'esistenza di un ambiente di runtime coeso come la Java Virtual Machine garantisce una sicurezza del codice mai avuta prima. La

macchina virtuale, frapponendosi tra i programmi ed il sistema operativo, controlla ogni operazione effettuata. Con Java non è possibile utilizzare la memoria in maniera illecita; ad ogni programma poi possono essere associati differenti profili di sicurezza, in maniera esplicita o implicita. Un software untrusted non può né leggere né scrivere sul disco fisso dell'utente, così come non può accedere alle risorse di rete.

2. *Robusto.* I software robusti sono stabili ed affidabili. Java fornisce dei modelli che aiutano lo sviluppatore nel gestire le situazioni inattese, fronteggiandole per tempo ed evitando ogni spiacevole inconveniente. La memoria inoltre è amministrata dalla macchina virtuale, che rimuove automaticamente le risorse non più utilizzate (garbage collector).
3. *Pensato per la rete.* Questo linguaggio è nato per rispondere alle esigenze della rete. Esiste un connubio molto forte tra Java e il Web.
4. *Dinamico.* Il bytecode prodotto dal compilatore non è un blocco monolitico, ma è suddiviso in tante altre parti dinamiche che possono essere trasferite, scambiate e recuperate con grande semplicità, anche durante l'esecuzione. La riusabilità del codice già compilato è massima. Le installazioni e gli aggiornamenti del software sono semplici e veloci, poiché si tratta spesso della copia di file.
5. *Multithreaded.* Ossia supporta nativamente la programmazione concorrente. Un programma Java può fare più cose contemporaneamente. Il modello offerto al programmatore consente di scrivere software multithreaded senza dover badare agli aspetti più strettamente tecnici della concorrenza, dovuti al dialogo del sistema operativo. La sincronizzazione è gestita in maniera stabile ed elegante.

6. *Concepito dai programmatori per i programmatori.* È un aspetto molto importante, lo stesso che ha decretato il successo di C/C++ ed il declino di altri linguaggi pensati per ambiti troppo specifici e realizzati con mentalità e strutture logiche non proprie del programmatore.

Naturalmente Java non è privo di svantaggi. Altrimenti non si spiegherebbe come mai tutto il software non venga distribuito sotto forma di bytecode. Ci sono delle limitazioni che bisogna tener presente.

Java non è adatto a scrivere componenti che debbano interfacciarsi direttamente con il sistema operativo o con l'hardware. Sarebbe impossibile per esempio scrivere un driver per dispositivi come tastiera, mouse, etc. Infatti per questo genere di operazioni devono essere impiegati linguaggi di più basso livello: questi, a fronte di una maggiore difficoltà di impiego, consentono un uso diretto di tutte le caratteristiche hardware e software necessarie. Invece Java resta dietro il muro della sua macchina virtuale, oltre il quale non può spingersi.

La *Java Virtual Machine* interpretando ed amministrando il bytecode, si frappone fra il software e il sistema operativo. Benché questa caratteristica favorisca la portabilità, la sicurezza, la robustezza, la dinamicità e la semplicità, bisogna temere il suo più grande difetto: il calo delle prestazioni. Il bytecode prima di essere eseguito, deve attraversare dei processi che impiegano tempo e spazio per essere portati a compimento. Al contrario, un programma compilato in codice eseguibile non è soggetto a manovre intermedie e può essere elaborato all'istante dal sistema operativo. Quando l'efficienza è una caratteristica prioritaria, bisogna accantonare Java in favore di altri linguaggi di più basso livello come il C/C++. In casi particolari è necessario addirittura ricorrere a routine ottimizzate mediante un assembler. Poi affinché il bytecode possa essere seguito, è necessario che una JVM sia presente nel sistema. Infatti se non si dispone di una macchina virtuale e non si ha la possibilità di installarla, Java è inutile.

C'è da aggiungere comunque che le più recenti versioni di Java dispongono di funzionalità utili per interagire a basso livello con l'hardware. Ad esempio la Java Communications API sono un insieme di strumenti che consentono un controllo diretto sulle porte seriali e parallele dei PC, mentre la Java Sound API si interfacciano con i driver delle schede sonore in maniera molto più avanzata rispetto di quanto si potesse fare prima della loro emissione. Inoltre è possibile utilizzare parti di codice nativo all'interno di un software Java. In questa maniera, le routine che necessitano di accessi di basso livello possono essere efficacemente realizzate con linguaggi adatti allo scopo, che compilino in linguaggio macchina, per essere poi richiamate dall'interno di una più vasta e portabile sovrastruttura Java. Gli sforzi del software grazie a questo espediente, sono minimi e le limitazioni di Java possono essere scavalcate persino in casi estremi.

Gli ultimi anni hanno portato inoltre stupefacenti progressi dal punto di vista delle prestazioni. In principio, la macchina virtuale interpretava il bytecode comando per comando, a tutto svantaggio dell'efficienza. Poi sia Sun che altri marchi hanno incrementato le prestazioni includendo la possibilità di una compilazione *Just in Time JIT* del bytecode. La tecnica consiste nel dotare la macchina virtuale di un secondo compilatore integrato, capace di trasformare le parti più pesanti del bytecode in codice nativo, prima della loro esecuzione.

In questo modo si raggiungono delle prestazioni che non si distaccano di molto da quelle ottenibili con C/C++, senza sacrificare la portabilità. Inoltre bisogna anche tener presente la crescita tecnologica che permea costantemente il mondo dell'informatica. Un complesso software Java avrà sicuramente dei problemi di efficienza se eseguito da un obsoleto 200Mhz con poca memoria a disposizione. Il calo delle prestazioni comunque non è altrettanto evidente sulle configurazioni messe in commercio più recentemente. Inoltre l'efficienza di Java è un problema che andrà a risolversi con l'emissione di nuove macchine virtuali quanto con l'incedere della crescita tecnologica. Già si sente parlare di complessi videogiochi in 3D completamente sviluppati in Java.

Attualmente la JVM è presente in ogni sistema; non c'è piattaforma sulla quale non sia teoricamente possibile eseguire il software Java. Alcuni sistemi operativi, come Mac OS X, includono la macchina virtuale direttamente nella loro installazione base, considerandola un tutt'uno con altri componenti. Diversi software in uso comune come i browser, installano automaticamente una Java Virtual Machine su quei sistemi che ne sono sprovvisti nella loro configurazione originaria.

Ci sono ulteriori vantaggi nell'adozione di Java. Per prima cosa è una tecnologia dalle specifiche aperte. Benché, come già scritto sopra, solo Sun Microsystems abbia diritto di vita e di morte sulla propria proposta, diversi produttori contribuiscono alla causa di Java nei modi più disparati. Le specifiche aperte contribuiscono alla libera implementazione del linguaggio e della sua macchina virtuale a chiunque se la senta di imbarcarsi in un progetto di tale portata. Quello di Java è uno dei pochi casi in cui il controllo esclusivo sull'evolversi di una tecnologia da parte di un'unica azienda è stato più un bene che un male. In poche parole un tempo sono sempre state impedito le estensioni proprietarie sviluppate da terzi, mentre sono state favorite le implementazioni rispettose della formulazione originaria. Per la prima volta da tanti anni a questa parte con Java arriva una tecnologia che sia allo stesso tempo aperta ed omogenea. Non possono sussistere differenze eclatanti fra le implementazioni di un produttore e quelle di un suo concorrente.

I principali strumenti di sviluppo di software Java, sono gratuiti e liberamente adoperabili senza alcuna limitazione tecnica o legale. Chi sceglie il linguaggio per uso personale, didattico o no-profit, non si ritroverà inutili bastoni tra le ruote. Anche chi vuole applicare Java ad un settore business non è soggetto a limitazioni. Data la natura più vasta del software d'impresa, ad ogni modo, in casi come questo sarà utile avvantaggiarsi di alcuni ambienti di sviluppo commerciali.

Per concludere Java e Open Source sono due mondi che spesso si toccano; l'uno favorisce la causa dell'altro. Numerosi strumenti di sviluppo e tantissimi

software Java vengono distribuiti insieme al loro codice sorgente. I vantaggi di questa caratteristica sono una maggiore sicurezza, una più veloce risoluzione dei problemi, la presenza di una comunità attiva che collabora alla crescita e all'arricchimento delle caratteristiche disponibili, la possibilità di consultare i sorgenti per scopi didattici e la facoltà di modificarli per meglio renderli consoni alle proprie esigenze.

Preso atto dei vantaggi e degli svantaggi di Java è possibile stilare una lista degli ambiti ideali per una corretta e coerente applicazione del linguaggio:

- Grandi progetti. Java per propria natura è uno strumento ideale per la gestione di grandi progetti, ai quali possono lavorare più persone simultaneamente. La programmazione orientata agli oggetti favorisce l'organizzazione, la manutenzione ed il riutilizzo del codice.
- Applicazioni per la rete. Tutto il software che punta sulla rete e sull'interoperabilità tra le macchine distinte si presta benissimo ad essere sviluppato con Java, anche per via della portabilità del bytecode. Il linguaggio di Sun è adatto tanto alle applicazioni server quanto alle loro controparti client. La sicurezza inoltre è un elemento preponderante nello sviluppo di software per la rete, e Java fa della sicurezza uno dei maggiori punti forza.
- Applicazioni per il Web. Tra i software che lavorano in rete hanno assunto particolar risalto le applicazioni per il Web. Java è ideale per il loro sviluppo, soprattutto per la programmazione lato-server. Le tecnologie *Servlet* e *JavaServer Pages – JSP* costituiscono la punta di diamante di un vasto insieme di strumenti adatti o adattabili al Web. Dal punto di vista lato-client, Java getta in tavola due prestigiose carte: le

applet e la tecnologia *Java Web Start – JWS*, che trovano significative applicazioni soprattutto nelle reti aziendali.

- Applicazioni enterprise. Le caratteristiche precedenti elencate, insieme con altre, confluiscono tutte in una naturale predisposizione di Java per lo sviluppo di software per le imprese. In Java sono incorporati diversi meccanismi all'avanguardia, che consentono un facile accesso alle basi di dati e la possibilità di distribuire il calcolo in maniera efficiente e sicura.
- Software per i dispositivi portatili. Java è la tecnologia più all'avanguardia in questo settore. Grazie alle sue doti di portabilità, si presta molto bene allo sbarco di apparecchi mobili di ogni tipo e natura. Già diversi produttori hanno adottato Java per lo sviluppo di software destinato ai dispositivi wireless. Sun Microsystem incoraggia il settore rilasciando continuamente nuove soluzioni disegnate per facilitare l'integrazione di Java in ogni genere di dispositivo portatile.

3.2 Qt

Qt è un C++ toolkit designato per lo sviluppo di applicazioni GUI multi-piattaforma. Qt supporta lo sviluppo di applicazioni GUI multi-piattaforma con l'approccio *write once, compile anywhere*. Usando un semplice albero sorgente e una normale ricompilazione, le applicazioni possono eseguire su Windows 95, per XP, per Mac OS X, Linux, Solaris, HP-UX e molte altre versioni di Unix. Le applicazioni Qt possono anche essere compilate per funzionare in modalità Qt/embedded. Inoltre introduce un unico meccanismo di comunicazione inter-object chiamato *signal and slots*. Ha un ottimo supporto per molti programmi come quelli per il 2D o il 3D grafico, XML, etc.



Figura3.2: Esempio di ambiente grafico in 3D, sviluppato in Qt

Le applicazioni possono anche essere sviluppate in ambiente visuale attraverso l'utilizzo del *Qt Designer*(figura 3.2).

Il toolkit Qt C++ ha cominciato il suo sviluppo principale dal 1995, usato da diverse compagnie come Adobe, IBM, Motorola, NASA e Volvo e da molte altre diverse compagnie e organizzazioni. Dalla versione 3.3 le Qt sono facile all'uso e nello stesso tempo molto potenti, e questo grazie all'aggiunta di significanti funzionalità e all'introduzione di nuove classi. Quest'ultime sono caratterizzate dal fatto che riducono di molto il carico di lavoro degli sviluppatori e forniscono dell'interfacce grafiche facile alla comprensione. Sono sempre state completamente object-oriented.

Qt include un ricco set di widgets, chiamati nella terminologia di Windows *controlli*, che forniscono funzionalità GUI standard. La innovativa comunicazione inter-object chiamata *signals and slots* sostituisce la vecchia e non sicura tecnica del callback¹². Le applicazioni Qt multi-piattaforma GUI possono usare tutta la funzionalità dell'interfaccia utente richiesta dalle moderne applicazioni come menu, context menu, drag and drop and dockable toolbar.

¹² Callback: puntatori a funzioni.

Qt è ideale per creare applicazioni per database di piattaforme indipendenti, usando database standard; include driver nativi come per Oracle, MicrosoftSql Server, Sybase Adaptive Server, IBM DB, PostgreSQL, MySQL, Borland Interbase, SQLite and ODBC-compliant database.

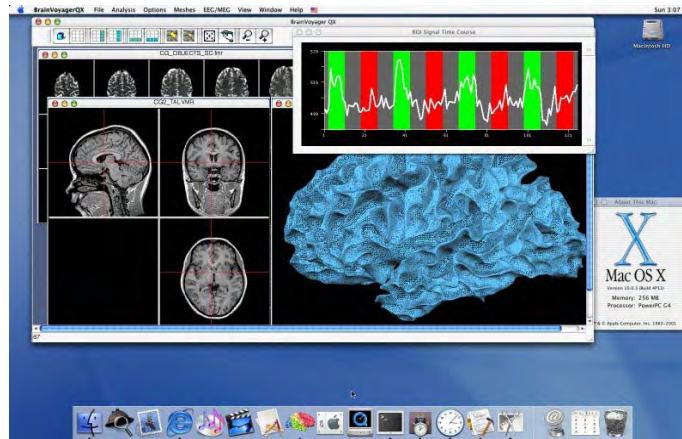


Figura 3.3: Esempio di applicazione sviluppata con Qt

Le funzionalità del database sono completamente integrate con il Qt Designer, (strumento di sviluppo per la realizzazione delle interfacce grafiche) offrendo una preview in tempo reale dei dati presenti nel database (Vedi figura 3.3). Le funzionalità delle Qt possono essere estese con l'utilizzo dei plugins e di alcune librerie dinamiche.

Si può definire come un toolkit maturo scritto in un C++ che è ampiamente usato in tutto il mondo. In aggiunta ai diversi usi commerciali, la libera edizione delle Qt è la fondazione del KDE [KDE04], il desktop environment di Linux. Qt provvede a tutte le classi e le funzioni necessarie a creare le moderne applicazioni GUI. Le Qt possono essere usate per creare

- Applicazioni con stile *main window*
- Applicazioni con stile *dialog*

Per stile *main window* si intendono le applicazioni con un menu bar, toolbar e status bar le quali circondano l'area centrale. Per stile *dialog* invece si intendono le applicazioni che usano bottoni e possibili tabs per presentare le opzioni e le

informazioni. Oltre a supportare l'uso di drag and drop e di clipboard, le Qt supportano anche i modelli chiamati *SDI* – *single document interface* e i *MDI* – *multiple document interface* (vedi figura 3.4). Le prime sono a interfaccia a documento singolo, supportano un singolo documento aperto per volta, mentre le applicazioni interfaccia a documenti multipli consentono di aprire due o più documenti contemporaneamente e supportano anche viste multiple di un documento.

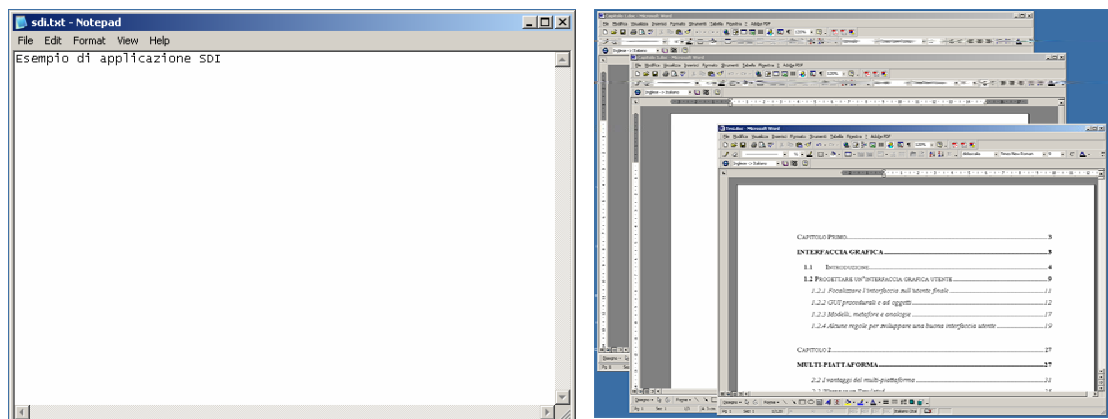


Figura 3.4: Esempio di SDI a sinistra e di MDI a destra

Le toolbar possono essere mosse intorno all'area toolbar, spostate in altre aree o posizionate come tool di palette. Queste funzionalità non richiedono codice aggiuntivo e sono intrinseche nelle Qt, sebbene i programmatori possono modificare gli aspetti delle toolbar se lo ritenessero necessario.

3.3 GTK

GTK è il toolkit di GIMP¹³. È una libreria per la creazione di interfacce utente grafiche, disponibile sotto licenza GPL¹⁴, che organizza le funzioni offerte da

¹³ GIMP: Programma GNU per la gestione delle immagini.

¹⁴ GPL: General Public License.

GDK¹⁵ in oggetti. Usando questa libreria si possono creare programmi Open Source, liberi o commerciali. La maggior parte di tali oggetti sono oggetti grafici che possono essere disegnati a schermo quali label, pulsanti, finestre, etc. Tali oggetti, chiamati widgets, facilitano la costruzione di un interfaccia grafica poiché permettono di accedere a tutte le caratteristiche di ogni singola componente grafica tramite un' interfaccia standard. GTK è costruito sulla base del kit di disegno di GIMP, il quale è sviluppato a sua volta attorno alle funzioni della Xlib. È chiamato toolkit di GIMP poichè era inizialmente scritto per sviluppare GIMP (vedi figura 3.5), ma ora viene utilizzato nello sviluppo di molti altri progetti software.



Figura 3.5: Esempio di applicazione Gimp

GTK è essenzialmente una API¹⁶ orientata agli oggetti. Anche se scritto completamente in C, è implementato usando l'idea delle classi e delle funzioni di callback. C'è anche una terza componente chiamata *glib* che fornisce una serie di implementazioni differenti di alcune chiamate di funzioni standard e anche alcune funzioni aggiuntive, per esempio per la manipolazione delle liste collegate, pronte all'uso quando si creano applicazioni GDK e GTK. Tali librerie sono definite all'interno del progetto GNU, ma sono portabili anche su piattaforme Windows.

¹⁵ GIMP Drawing Kit.

¹⁶ API: application programmers interface.

3.4 WxWindows

WxWindows [WXW04] è un framework per C++ che offre accesso alla GUI e ad altre funzioni comuni a diversi SO . La versione attuale supporta diversi desktop quali MS Windows (16-bit, Windows 9.x e Windows NT), Unix con GTK+¹⁷ o con Motif¹⁸ e MacOS, ed è in fase di sviluppo un port per l' OS/2. Originariamente wxWindows è stato creato dall'istituto di intelligenze artificiali dell'Università di Edinburgo, per uso interno e poi reso pubblico dal 1992. È stato sviluppato per garantire un economico e flessibile modo per massimizzare l'investimento sostenuto nello sviluppo delle *GUI*, mentre altre librerie commerciali già esistenti per lo sviluppo del multi-piattaforma non hanno fatto nulla per avvicinarsi a criteri come:

- Basso prezzo;
- Disponibilità di sorgenti;
- Semplicità di programmazione;
- Supporto per un numero ampio di compilatori.

Da quando è comparso wxWindows, sono nati altri diversi frameworks di questo tipo. Comunque nessuno ha le stesse caratteristiche, la stessa flessibilità, la stessa documentazione, e un valido team di sviluppo come il suo. Come software libero, wxWindows ha tratto beneficio non solo dalle idee e dalle osservazioni degli utenti, ma anche dallo stesso entusiasmo che questi mettevano nella programmazione. Ciò ha dato a questo toolkit un particolare vantaggio rispetto ad altri nel suo genere.

Molte sono le funzionalità di wxWindows, e riproporli tutti sarebbe impossibile, ma di seguito vengono proposti alcuni suoi benefici:

- Basso costo;

¹⁷ GTK+: GTK orientato agli oggetti

¹⁸

- Disponibilità dei sorgenti;
- Disponibile su una varietà di piattaforme popolari;
- Binding con quasi tutti i compilatori popolari;
- Oltre 50 programmi di esempio;
- Oltre 1000 pagine di documentazione stampabile ed in linea;
- Include Tex2RTF, per permettere lo sviluppo di una propria documentazione;
- API simple-to-use e orientate agli oggetti;
- Sistema di eventi flessibile;
- Le chiamate dei grafici includono le linee, i rettangoli arrotondati, etc;
- Architetture di document/view e di Print/preview;
- Toolbar, taccuino, controllo ad albero, controllo avanzato dei file;
- Supporto di MDI (interfaccia multipla del documento);
- Può essere usato per generare DLLs sotto Windows, librerie dinamiche su UNIX;
- Sotto MS WINDOWS, supporto per la generazione dei metafiles e la loro copiatura nella clipboard;
- Un API per l'invocazione dell'aiuto dalle applicazioni;
- Finestra pronta per l'uso del HTML;
- Elaborazione di immagini indipendente della piattaforma;
- Supporto di molti formati grafici come bmp, png, jpeg, gif, xpm, pnm, pcx.

Quindi wxWindows fornisce un ricco set di API per scrivere applicazioni GUI su diverse piattaforme (vedi figura 3.6). Linkando la libreria appropriata per la piattaforma in uso (figura) e compilando (supporta quasi tutti i più diffusi compilatori), l'applicazione generata assumerà l'aspetto appropriato per quella piattaforma. wxWindows fornisce API per: help in linea, programmazione di rete, gestione dei files, clipboard e drag and drop, multithreading, caricamento e salvataggio di vari formati grafici, supporto database, visualizzazione e stampa in formato HTML , e molto altro ancora [KON03].

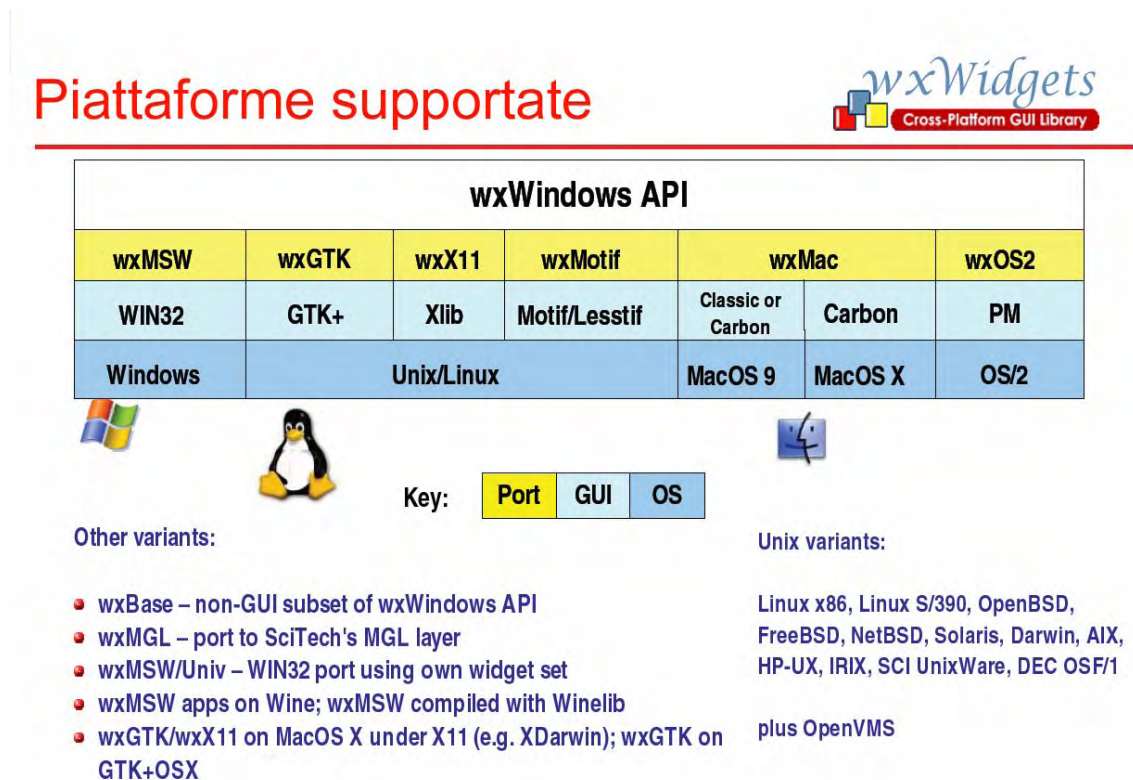


Figura 3.6: wxWindows - Piattaforme supportate

Nonostante fossero già disponibili un discreto numero di classi per lo sviluppo di applicazioni *cross-platform*, nessuna fornisce tutte le caratteristiche che possiamo trovare in wxWindows [SMA04]. Data la sua natura open source, wxWindows ha beneficiato dei commenti, idee, bug fix, esempi pratici e il grande entusiasmo degli utenti. Questo dà a wxWindows un certo vantaggio

rispetto ai suoi concorrenti di natura commerciale (e anche rispetto alle librerie gratuite ma con un team di sviluppo indipendente), più una struttura robusta contro la transitorietà di molti progetti individuali o commerciali, che spesso vengono dimenticati e non costantemente aggiornati. Questa apertura e disponibilità di codice sorgente è, in particolar modo, molto importante quando il futuro di migliaia di applicazioni dipende dalla longevità delle classi sottostanti.

La seconda versione delle librerie wxWindows ha introdotto notevoli migliorie sia in termini di potenzialità che di portabilità, permettendo lo sviluppo di applicazioni (vedi figura 3.7) che spesso sono indistinguibili da quelle realizzate con i più avanzati strumenti di sviluppo in ambienti dedicati; come per esempio l'uso delle librerie MFC dedicate al solo ambiente Windows. WxWindows piace ai programmatori in quanto si ha un facile apprendimento, un facile utilizzo e non è proprietario. Si riesce a migrare da MFC con molta semplicità. In più ha una soluzione particolarmente adatta per chi già utilizza Win32:

- Costruttori di classi simili a MFC;
- Possibilità di utilizzare codice misto wxWindows e MFC (su Win32);
- Ottima soluzione per migrazione incrementale verso linux

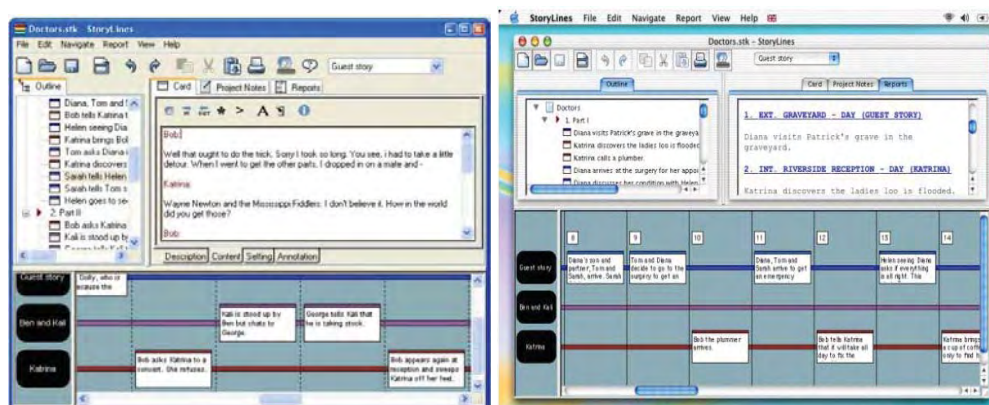


Figura 3.7: Un'applicazione wxWindows a sinistra con XP, a destra con Mac OS X

3.5 Le librerie messe a confronto

Consideriamo ora le Qt scritte in C++ e mettiamole a confronto con Java.

La scelta del linguaggio di programmazione è una decisione che condiziona notevolmente lo sviluppo del progetto. In più la sua scelta impone un considerevole impatto sulle altre opzioni disponibili. In aggiunta c'è da considerare un altro fattore, quello che si basa sull'esperienza e sulla preferenza personale del programmatore piuttosto che su criteri oggettivi come

1. il programmer-efficiency;
2. il runtime-efficiency;
3. la memory-efficiency.

Il programmer-efficiency descrive quanto efficacemente un programmatore con una certa esperienza e conoscenza può implementare un insieme di richieste (debug, setup, etc) per un particolare linguaggio di programmazione. Si consideri Java: la fortuna di questo linguaggio è legata in maniera inequivocabile all'espansione della rete Internet che ne ha valorizzato le caratteristiche e gli ha offerto un vasto campo di sviluppo. Allo stesso modo anche la rete ha ottenuto innumerevoli vantaggi da questo linguaggio che ha portato ad una notevole semplificazione in fase di implementazione di molti servizi, alla semplificazione delle operazioni, ad un miglioramento sostanziale dell'interfaccia utente ed alla introduzione di nuove potenzialità. Java è un linguaggio object-oriented *puro*, cioè che rispetta i principi di questo paradigma. Questo, non è vero per il C++, che perde alcune caratteristiche del paradigma OOP¹⁹ per mantenere la compatibilità con il C. Java è un linguaggio di programmazione che non privilegia le prestazioni, ma favorisce in generale la semplicità nella gestione di gran parte delle risorse ricorrendo spesso all'automazione di queste. Tale limite alle prestazioni è dato dalla presenza (come già citato nei paragrafi precedenti) di una macchina virtuale che

¹⁹ OOP: Object Oriented Programming

interpretando il metacodice (bytecode) realizza l'indipendenza dall'architettura. L'obiettivo di Java è di aumentare la produttività della programmazione (*programmer-efficiency*) rispetto ad altri linguaggi general-purpose come il C++, piuttosto che un'efficienza nell'utilizzo della memoria o nella velocità di esecuzione (*memory o il runtime-efficiency*). Per ottenere tale vantaggio, il programmatore non deve ricordarsi di liberare esplicitamente la memoria, perché se ne occuperà automaticamente un garbage collector operante in background, a discapito della *memory* e della *runtime-efficiency*. Ma c'è da considerare che per chi programma ad alto livello, questo è influente. Studi fatti mostrano che in pratica il garbage collection e le altre caratteristiche di Java non influiscono di molto sul *programmer-efficiency*, visto che nello sviluppo di un software le ore richieste dipendono dall'esperienza del programmatore, dall'esperienza con il linguaggio di programmazione in questione e da quanto il programma deve essere affidabile, mentre sono altamente indipendenti dal livello del linguaggio, cioè da quanto il linguaggio si astrae dalla architettura del sistema. Uno dei classici software che sono in grado di fare una stima sul costo e sullo schedule di un progetto software usando variabili come l'esperienza di un programmatore, l'affidabilità del programma, etc. è il Barry Boehm's COCOMO²⁰. Boehm scrive che << *the amount of effort per source statement was highly independent of the language level*>>. Altri studi come per esempio "A method of programming measurement and estimation" by C.E. Walston and C.P. Felix of IBM²¹ portano alle stesse considerazioni. C'è poi una recente ricerca, dell'Università di Karlsruhe, dove vengono empiricamente comparati linguaggi come C, C++, Java Perl, Python, Rexx and Tcl²², descrive un esperimento fatto sugli studenti, ai quali è stato dato da implementare un progetto nel linguaggio che essi preferivano: C, C++, Java, etc. I dati raccolti

²⁰ Software Engineering Economics, by Barry Boehm, Prentice Hall.

²¹ A method of programming measurement and estimation by C.E. Walston and C.P. Felix, IBM.

²² An empirical comparison of C, C++, Java, Perl, Python, Rexx and Tcl by Lutz Prechelt, University of Karlsruhe.

mostrano gli stessi risultati sia per il C che per il Java. Questo ha dimostrato che i programmatori di un certo livello, che scelgono di sviluppare un software nel loro linguaggio preferito che sia Java o C, raggiungono lo stesso livello di programmer-efficiency. Questo tipo di ricerca contraddice chi sostiene che i programmatori Java raggiungono un più alto livello di efficienza dei programmatori di C.

3.6 Runtime-efficiency

I programmi Java tendono ad andare 2 o 3 volte più lenti rispetto ai programmi scritti in C per lo stesso compito. Questo perché Java perde molto quando i task sono CPU-bound. Studi fatti sull'usabilità mostrano che gli utenti non fanno caso al tempo che impiega un programma a svolgere il proprio lavoro, che possa essere anche di due o tre minuti, ma fanno caso quando un programma non mostra immediatamente una reazione alla loro interazione, per esempio quando premono un pulsante. Questi studi mostrano che i limiti di risposta per i quali un programma è considerato lento devono essere poco più di 0,7 secondi. Ritornando al confronto con il C e il Java, i programmi scritti in C sono compilati in un formato binario che può essere eseguito direttamente dalla CPU, mentre il Java, come visto nel paragrafo precedente, compila il codice sorgente in un bytecode, che non è eseguito direttamente dalla CPU, ma dalla JVM. Un'altra considerazione importante da fare è a discapito del motto dei programmatori di Java. Compila una volta ed esegui ovunque, è un approccio piuttosto ideale. Una volta scritto il programma in Java, questo è compilato nel bytecode e può essere eseguito su ogni piattaforma avente la JVM. In pratica questo non è sempre vero, in quanto esistono diverse implementazioni di differenti JVM e anche perché a volte è necessario usare un codice nativo non Java, scritto di solito in C, insieme a programmi in Java. Invece nel caso venisse usato un toolkit multi-piattaforma come Qt e un affidabile compilatore sulle varie piattaforme potrebbe aver senso dire *scrivi una volta, compila ovunque*.

3.7 Memory-efficiency

Java e il C hanno un differente approccio di controllo della memoria. In C tutti i controlli della memoria devono essere fatti dal programmatore, questi deve allocare e deallocare la memoria secondo circostanza. Nel caso si dimenticasse di deallocarla, si andrà a creare la cosiddetta *memory leak*. Se il *memory leak* si presenta una sola volta durante l'esecuzione di una applicazione, non ci sono problemi, in quanto il sistema operativo rilascia la memoria una sola volta al termine dell'applicazione. Ma se il *memory leak* avviene più volte, la memoria di un programma in esecuzione aumenterà ogni volta che l'utente invocherà una certa funzione, ciò facendo si andrà a consumare tutta la memoria disponibile del computer, con un possibile crash della macchina. Java invece dealloca automaticamente la memoria non utilizzata e la JVM tiene traccia di tutti i *blocchi di memoria* e di ogni riferimento ad essa. Quando i blocchi di memoria non sono più referenziati, possono essere eliminati. Questo avviene in un processo chiamato *garbage collection*, nel quale la JVM periodicamente tiene traccia dei blocchi di memoria allocati e li rimuove quando non sono più referenziati. Il *garbage collection* è molto conveniente, ma il compromesso di ciò è un grande consumo di memoria e una lenta velocità di esecuzione. In C/C++ i blocchi di memoria possono essere cancellati dal programmatore ogni volta che lo ritiene necessario, in Java invece questi blocchi non sono cancellati fino a che non va in esecuzione il *garbage collection*, dipendente dall'implementazione della JVM. In un programma scritto in C/C++, il programmatore conosce dove sono i blocchi di memoria (grazie a puntatori) e sa quando questi non sono più necessari. In un programma in Java questa informazione non è disponibile. Ritornando al toolkit Qt, questi ha un approccio efficiente per facilitare il controllo della memoria ad ogni programmatore: quando un oggetto è cancellato, tutte le dipendenze dell'oggetto sono cancellate a loro volta. Questo tipo di approccio non interferisce con la libertà del programmatore nel cancellare manualmente gli oggetti quando lo ritiene opportuno. Come è facile intuire, il linguaggio C++ è un'estensione del

linguaggio C. In particolare, il C++ conserva tutti i punti di forza del C, come la potenza e la flessibilità di gestione dell'interfaccia hardware e software, la possibilità di programmare a basso livello e l'efficienza, l'economia e le espressioni, tipiche del C. Ma, in più, il C++ introduce il dinamico mondo della programmazione orientata agli oggetti che rende tale linguaggio una piattaforma ideale per l'astrazione di problemi di alto livello.

Il C++ fonde, quindi, i costrutti tipici dei linguaggi procedurali standard, familiari per molti programmatori, con il modello di programmazione orientata agli oggetti, che può essere pienamente sfruttato per produrre soluzioni completamente orientate agli oggetti di un determinato problema. In pratica, un'applicazione C++ riflette questa dualità incorporando sia il modello di programmazione procedurale che il modello di programmazione orientato agli oggetti.

La scelta di un particolare toolkit dipenderà dalla scelta del linguaggio preferito dal programmatore tenendo presente due cose: in primo luogo perché alcuni linguaggi nascono per un particolare toolkit, in secondo luogo perché alcuni toolkit hanno un supporto limitato solo ad alcuni linguaggi. I toolkit e i linguaggi visti in questo capitolo sono ottimi strumenti di sviluppo, sta poi al programmatore sceglierne uno, in base alle proprie esigenze e al programma da sviluppare.

Per lo sviluppo del progetto di questa tesi si è preferito usare una libreria che fosse in grado di avere aspetti comuni alle librerie sopra descritte e soprattutto fosse Open Source. Una libreria che avesse nello stesso tempo caratteristiche come semplicità, flessibilità e soprattutto altamente performante e che fosse in grado di essere compilata sulle piattaforme principali ora in commercio, senza preoccuparsi di riscrivere e riadattare il codice. Questa libreria si chiama *SDL - Simple Directmedia Layer* e verrà descritta nel capitolo seguente.

Capitolo 4

Simple DirectMedia Layer

In questo capitolo verrà spiegato che cos'è SDL, quali sono le sue potenzialità, chi sono gli utenti che utilizzano questa libreria e quali sono i vantaggi nell'utilizzarla.

Simple DirectMedia Layer - SDL [SDL04] è una API gratuita multi-piattaforma per lo sviluppo multi-media, usata per la creazione di giochi, sistemi di sviluppo (SDK) di giochi, demo, emulatori, MPEG players e tante altre applicazioni. Permette un accesso di basso livello alle periferiche multimediali, ovvero all'audio, al video 2D, al video 3D tramite OpenGL [OGL04], 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*²³, *The battle for Wesnorth*²⁴, *Never Winter Night*²⁵ e svariati porting della Loki Games²⁶.

La libreria può essere scaricata dal sito ufficiale²⁷ insieme alla documentazione in formato HTML.

4.1 Perché scegliere SDL

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 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.

²³ Cube: <http://cube.sourceforge.net>

²⁴ The battle for Wesnorth: <http://www.wesnoth.org>

²⁵ Never Winter Night : <http://nwn.bioware.com>

²⁶ Loki Games Entertainment: <http://www.lokigames.com/products>

²⁷ Sito ufficiale: <http://www.libsdl.org/index.php>

4.2 Breve storia

Lo sviluppo SDL è cominciato nel 1997 quando Sam Lantinga stava lavorando alla conversione, in ambiente Windows di un emulatore Macintosh chiamato Executor. Nel sviluppare il codice, si rese conto come il sistema che stava emulando non fosse, a livello di implementazione, troppo diverso rispetto a quello su cui stava lavorandoole applicazioni grafiche richiedono al sistema operativo la possibilità di lavorare sul framebuffer, di conoscere gli eventi di input, quali tastiera, mouse, joystick, etc. Sarebbe stato molto conveniente se, per scrivere un'applicazione, fosse stata disponibile una libreria che facesse da tramite con più sistemi operativi, lasciando invariata la API. In questo modo, una volta completato il sorgente, ci sarebbe stato il vantaggio di poterlo compilare su tutti i tipi di macchina supportati dalla libreria, senza apportare cambiamenti al codice. Lantinga si interessò a questa idea e iniziò a documentarsi sulle problematiche coinvolte: era necessario offrire una grande possibilità di interazione con il sistema in uso e, nello stesso tempo, mantenere la libreria snella e semplice, in modo da non gravare sulle applicazioni che si sarebbero appoggiate ad essa. Nel 1998 fu completata la prima release veramente stabile di SDL; per testare le potenzialità e pianificare nuovi miglioramenti, Lantinga la utilizzò per terminare il porting di Executor. Quasi in contemporanea fu rilasciato il codice sorgente di Doom e le chiamate grafiche del gioco furono presto redirette verso SDL e, dopo pochi giorni di test, SDL Doom girava correttamente su Linux, su Windows e su BeOS; le tre piattaforme che SDL supportava a quel tempo. Questo catturò immediatamente l'attenzione di molte persone, programmatori e non. Lantinga stesso afferma che uno dei punti cruciali durante lo sviluppo di SDL fu proprio la sua integrazione in Doom e il fatto che il codice risultante fosse a disposizione di chi volesse provarlo. In breve tempo egli si vide sommerso da commenti e idee di persone che avevano guardato il codice e cominciavano ad usare la sua libreria.

La comunità di utilizzatori crebbe velocemente; la semplicità d'uso di SDL, rispetto alle comuni librerie legate ai vari sistemi come X11 sotto Linux e

DirectX sotto Windows, la rendeva una attraente alternativa anche per chi non fosse interessato a problemi di portabilità. Con il crescere della libreria, crebbero anche le attenzioni che orbitavano attorno ad essa; SDL venne usata per integrare molte applicazioni, anche commerciali.

4.3 La situazione oggi

Al momento SDL ha raggiunto la stabilità nella versione 1.2.7 e supporta diverse piattaforme: Linux, Windows, BeOS, MacOS classic, MacOS X, Solaris, IRIX, QNX e FreeBSD. È usata principalmente per lo sviluppo di applicazioni grafiche, quali videogiochi, player di file video, benchmark, screensaver, plugin per XMMS, etc. La licenza con cui la libreria viene distribuita è la LGPL²⁸. In pratica questo consente agli sviluppatori di poter scrivere software commerciale facendo uso di SDL; l'unica restrizione è il linking, che deve essere di tipo dinamico. Questo particolare oltre a far sì che il programmatore renda conto dell'utilizzo di SDL nel proprio programma, comporta numerosi vantaggi. Ad esempio il fatto che più programmi condividano gli stessi file di libreria, riducendo la quantità di spazio su disco e la memoria allocata; oppure aggiornando la libreria, i programmi installati che la usano beneficino dei bugfix e miglioramenti presenti nella nuova versione, senza dover essere ricompilati. Una delle nuove innovazioni introdotte dalla versione 1.2.1, riguarda il supporto del sistema grafico di PlayStation 2. Sony ha, infatti, prestato un esemplare della sua console, insieme al kit Linux per PSX2 (ai tempi ancora in beta), al team di sviluppo di SDL, al fine di far produrre una patch apposita alla libreria. Inizialmente doveva rimanere un esperimento destinato a rimanere nei server CVS del progetto per molto tempo, ma visto l'ottima qualità del codice, che supportava sia il sistema grafico che l'interfacciamento con i controller, si è deciso di importare tutto nel progetto ufficiale. Lo stesso Sam Lantinga ha inoltre completato la conversione di un suo

²⁸ LGPL: Lesser General Public License

videogioco su PlayStation2. Oltre a dichiararsi soddisfatto della qualità del supporto PSX2 di SDL, ha aggiunto che il codice non ha ancora raggiunto il 100% di sviluppo visto che manca un test su larga scala, ma si tratta di una situazione momentanea e con il tempo i pochi problemi saranno risolti e nel frattempo saranno aggiunte nuove funzionalità. La filosofia alla base di SDL ne guida lo sviluppo verso le principali caratteristiche di semplicità e leggerezza. Infatti SDL è una libreria che interagisce con il sistema a basso livello, ponendosi come un layer tra esso e il programmatore. Questo però fornendo una API essenziale, ad esempio in termini pratici, qualora ci fosse bisogno di disegnare sulla superficie grafica, l'utente dovrà scriversi le varie routine del caso, dato che SDL offre semplicemente l'accesso al framebuffer e poco più per quanto riguarda il video.

4.4 La comunità

SDL è una libreria progettata e scritta da un gruppo di programmatori di giochi, competenti e altamente professionali. Senza togliere nessun merito agli autori delle altre librerie, SDL risulta semplicemente essere sostenuta da un ampio numero di programmatori; scritta da programmatori di giochi per i programmatori di giochi. Ti permette di prendere un programma scritto per Windows, di ricompilarlo e di eseguirlo sotto Linux, Mac OS, e addirittura su palmari e viceversa. Se si è sotto Windows, SDL è stratificato al di sopra delle DirectX, mentre sotto Linux SDL si trova sopra X e diverse altre librerie. Un'altra potenzialità delle SDL sta nel fatto che l'utente è in grado di trovare migliaia e migliaia di esempi e di risposte a domande di ogni genere e questo grazie non tanto alla documentazione ufficiale, ma alla comunità che sta dietro a questa libreria. Guidata da Sam Lantinga con l'aiuto di un gruppo di volontari ben affiatati, la comunità crea sempre nuovi progetti, migliora il codice già sviluppato e rende portabile l'utilizzo di SDL per nuove piattaforme. La SDL

mailing list²⁹ è una sorgente costante di informazioni. La mailing list è un archivio con una vasta gamma di informazioni su come utilizzare SDL per risolvere problemi nella programmazione di giochi per computer.

4.5 Panoramica

SDL gestisce gli eventi della tastiera, del mouse e del joystick, usando un modello di gestione degli eventi non complesso, in modo tale da risultare familiare ad ogni programmatore che ha scritto codice sotto X11, Windows o MacOS. La questione sta nel fatto che SDL gestisce gli eventi indipendentemente dal sistema operativo che l'utente usa, questo perché non usa particolari eventi specifici di un SO che potrebbero complicare il lavoro di un programmatore.

Ogni programma di gioco deve memorizzare il tempo e deve essere in grado di notificarlo quando un'azione deve essere eseguita. SDL fa uso di una semplice, chiara e sicura API che è comune ad ogni macchina e indipendente dal SO. L'SDL timer API consente la creazione di centinaia di contattori (timers) e possono essere usati senza limitazioni e preoccupazioni di altri timer usati in precedenza.

SDL fa uso di una semplice API per la lettura e la riproduzione delle tracce sui CD e per verificare il funzionamento della scheda sonora e tutto ciò che serve per produrre una bassa latenza sonora per ogni dispositivo.

SDL include un' API a basso livello di gestione della rete, che permette all'utente di controllare sia il socket TCP/IP sia quello UDP/IP. L' API è simile al networking API dei sistemi operativi Windows e UNIX; la differenza sta nel fatto che permette di usare solo le caratteristiche della rete che sono comuni a tutti i sistemi operativi supportati.

SDL gestisce un thread API che assomiglia alla versione semplificata di pthreads [PTH04]. I threads SDL forniscono tutte le funzionalità base che serve

²⁹ Mailing List: <http://www.libsdl.org/mailman/listinfo/>

a un programmatore in un thread package, mentre maschera i dettagli a basso livello che confondono tante persone e fanno il codice thread così difficile. SDL threads sono supportati da tutti i sistemi operativi che supportano sia i thread che SDL.

4.6 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 alla 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.
```

Nel caso si volessero usare e le funzioni video e quelle joystick, la formula da usare sarebbe:

```
SDL_Init ( SDDL_INIT_VIDEO | SDL_INIT_JOYSTICK );
```

È possibile in qualsiasi momento attivare e disattivare i singoli sottoinsiemi facendo uso delle funzioni:

- `int SDL_InitSubSystem(Uint32 flags)`
- `void SDL_QuitSubSystem(Uint32 flags)`

e controllare quali sono attivi con:

- `Uint32 SDL_WasInit(Uint32 flags)`

Impostando e interpretando correttamente le maschere di bit secondo la loro definizione in libreria.

4.6.1 Eventi

In un'applicazione si avrà bisogno di gestire l'input generato dall'utente e, di conseguenza, aggiornare lo stato interno dell'applicazione. Tastiera, mouse e joystick sono le vie di interazione più usate: ma resta da vedere come sia possibile utilizzare gli eventi da essi generati. Per avere a disposizione i dati che servono all'utente, in SDL è implementata una coda degli eventi. Questo significa che ogni volta che un tasto viene premuto oppure ogni volta che viene ricevuta una sequenza di uscita, come il pulsante di chiusura della finestra o Ctrl+C, oppure viene mosso il joystick, etc, alla coda è aggiunto un

identificativo che individua univocamente l'evento. Quando l'utente preleva l'evento dalla coda con la funzione:

- `int SDL_PollEvent (SDL_Event *event);`

attraverso la struttura *event* l'utente è in grado di ricavarne il tipo ed altre informazioni. La funzione restituisce 1 finchè ci sono eventi da processare, 0 in caso contrario. Questo sistema è estremamente conveniente per il programmatore; infatti, in un'applicazione il loop principale consisterà essenzialmente di tre passi:

- 1) la gestione dell'input;
- 2) l'aggiornamento di eventuali stati interni;
- 3) il disegno del frame corrente.

Interpretare gli eventi tramite una coda permette, quindi, al programmatore di processarli correttamente.

4.6.2 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) le dimensioni in pixel di questa;
- 3) l'eventuale area di clipping, all'interno della quale è prevista la possibilità di scrittura;
- 4) il 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 su 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:

Per comprendere meglio come sono gestibili gli eventi, è necessario conoscere come è strutturata `SDL_Event`, in cui si ha a disposizione il tipo dell'evento, come la pressione di un tasto, il ridimensionamento della finestra, il movimento del mouse, etc.) e le informazioni relative ad esso, di che tasto si tratta, di quando si è mosso il mouse, quali sono le nuove dimensioni della finestra, etc. `SDL_Event` è definita nel seguente modo:

```
typedef union{
    Uint8 type;
    SDL_ActiveEvent active;
    SDL_KeyboardEvent key;
    SDL_MouseMotionEvent motion;
    SDL_MouseButtonEvent button;
    SDL_JoyAxisEvent jaxis;
    SDL_JoyBallEvent jball;
    SDL_JoyHatEvent jhat;
    SDL_JoyButtonEvent jbutton;
    SDL_ResizeEvent resize;
    SDL_ExposeEvent expose;
    SDL_QuitEvent quit;
    SDL_UserEvent user;
    SDL_SysWMEvent syswm;
} SDL_Event;
```


Si tratta di una struttura che descrive un evento generico. Il suo primo membro *type* può contenere i valori enumerati nella tabella 4.1

Valori per il membro Type	Struttura corrispondente
SDL_ACTIVEEVENT.....	SDL_ActiveEvent
SDL_KEYDOWN/UP.....	SDL_KeyboardEvent
SDL_MOUSEMOTION.....	SDL_MouseMotionEvent
SDL_MOUSEBUTTONDOWN/UP.....	SDL_MouseButtonEvent
SDL_JOYAXISMOTION.....	SDL_JoyAxisEvent
SDL_JOYBALLMOTION.....	SDL_JoyBallEvent
SDL_JOYHATMOTION.....	SDL_JoyHatEvent
SDL_JOYBUTTONMOTION.....	SDL_JoyButtonEvent
SDL_QUIT.....	SDL_QuitEvent
SDL_SYSWMEVENT.....	SDL_SysWMEvent
SDL_VIDEORESIZE.....	SDL_ResizeEvent
SDL_VIDEOEXPOSE.....	SDL_ExposeEvent
SDL_USEREVENT.....	SDL_UserEvent

Tabella 4.1: I vari tipi di evento

grazie ai quali si può stabilire una corrispondenza tra il tipo dell'evento e la sua struttura dati ad esso associata con la quale interpretare i dati contenuti al suo interno. Il programmatore quindi può utilizzare il costrutto *case* come nel seguente frammento di codice differenziando il campo di lettura a seconda del tipo di evento:

```
switch(event.type)
{
    case SDL_VIDEORESIZE:
        ...
        break;
    case SDL_KEYDOWN:
    case SDL_KEYUP:
```

```

        ...
    break;
    case SDL_ACTIVEEVENT:
        ...
    break;
    case SDL_MOUSEBUTTONDOWN:
    case SDL_MOUSEBUTTONUP:
        ...
    break;
    case SDL_MOUSEMOTION:
        ...
    break;
    case SDL_QUIT:
        ...
    break;
}

```

I nomi dei tipi di evento sono nella maggior parte dei casi autoesplicativi; ognuno di questi viene automaticamente inserito nella coda da SDL, a seguito di un certo input da parte dell'utente vedi tabella 4.2.

Event Type	Generato da:
SDL_QUIT.....	Tasto di chiusura della finestra
SDL_VIDEORESIZE.....	Ridimensionamento della finestra
SDL_KEYUP.....	Pressione di un tasto
SDL_KEYDOWN.....	Rilascio di un tasto
SDL_ACTIVEEVENT.....	Acquisto o perdita di focus per la finestra. Uscita o entrata del puntatore del mouse sulla finestra. Finestra ridotta a icona o massimizzata.
SDL_MOUSEBUTTONDOWN.....	Pressione di un pulsante del mouse
SDL_MOUSEBUTTONUP.....	Rilascio di un pulsante del mouse
SDL_MOUSEMOTION.....	Movimento del mouse

Tabella 4.2: Eventi e tipi di input

Una cosa molto importante da analizzare è che la coda funziona ovviamente in lettura, ma anche in scrittura; ciò significa che è possibile inserire eventi nella coda a piacimento del programmatore. Questo tornerà utile quando si avrà bisogno di generare eventi per associazioni non standard, ad esempio in modo che la pressione del tasto *Escape* generi un evento *SDL_QUIT* e quindi provochi la chiusura dell'applicazione. Questo serve per avere la possibilità di simulare un qualsiasi tipo di evento.

4.6.3 SDL_KEYDOWN, SDL_KEYUP

A questi due tipi corrisponde una sola struttura (*SDL_KeyboardEvent*); una scelta molto comune, utilizzata dai programmatori, è quindi quella di gestire pressione e rilascio di tasti con la stessa funzione e discriminare i due casi successivamente, controllando il membro *state* della struttura. Il membro *keysym*, invece, consente di risalire al tasto in questione. La funzione *SDL_GetKeyName()* permette anche di ottenere una stringa contenente il nome del tasto premuto:

```
printf("%s\n", SDL_GetKeyName(key->keysym.sym) );
```

È possibile anche determinare se, mentre un tasto è stato premuto, erano attivi dei modificatori (CTRL, ALT, SHIFT, ecc); per farlo è sufficiente controllare il sottomembro *mod* usando (*and*, & binario) le maschere che interessano.

Ovviamente, ne possono venire gestiti anche più di uno contemporaneamente (CTRL-ALT-SHIFT-CAPSLOCK-x, etc, per far fronte alle necessità delle vostre applicazioni). Un'altra cosa di cui tenere conto è che i tasti di stato (*num lock*, *caps lock*, *scroll lock*), proprio a causa della loro particolare natura, si comportano diversamente dai tasti normali; se, ad esempio, il *num lock* risulta disattivato, una volta premuto il tasto in questione verrà normalmente passato nella coda un evento *SDL_PRESSED*, ma al suo rilascio non verrà chiamato nessun *SDL_RELEASED*: il *num lock* è, infatti, attivo e verrà disattivato solo da un'altra pressione del tasto corrispondente, alla quale seguirà ovviamente l'evento *SDL_RELEASED*, simulando la meccanica delle vecchie macchine da scrivere. Se

si volesse provare a inserire l'evento *SDL_QUIT* nella coda non appena viene rilasciato il tasto *Esc*, questi genera automaticamente la chiusura dell'applicazione senza la scrittura di ulteriore codice. Dunque, bisogna controllare se il codice (*sym*) del tasto premuto corrisponde a *SDLK_ESCAPE* e se l'utente sta rilasciando il tasto:

```
if (( key->keysym.sym == SDLK_ESCAPE) && (key->state ==
SDL_RELEASED) )
```

In questo caso è sufficiente definire un nuovo evento generico *quit*

```
SDL_EVENT quit;
```

assegnargli il tipo desiderato (*SDL_QUIT*)

```
quit.type = SDL_QUIT
```

e, infine, inserirlo nella coda con la funzione *SDL_PushEvent()*:

```
SDL_PushEvent (&quit) ;
```

L'ultima cosa da notare è che la libreria supporta anche la gestione dei caratteri UNICODE³⁰ e che, se questa viene attivata con *SDL_EnableUNICODE()*, all'interno del membro *unicode* di *keysym* si troverà il codice corrispondente. La conversione del codice comporta, comunque, un certo rallentamento delle prestazioni (che è anche il motivo per cui questa funzionalità è disattivata per default).

4.6.4 SDL_ACTIVEEVENT

Questo evento viene inserito in coda in seguito alla modifica del focus dell'applicazione (vedi figura BB); nel sottomembro *state* è innanzitutto salvato il tipo di focus

³⁰ UNICODE: La codifica per rappresentare caratteri latini, greci, ebraici, arabi, cirillici, etc.

al quale si fa riferimento, mentre in *gain* vediamo la modifica dello stato: può trattarsi di una perdita (0) o di un acquisizione (1). I possibili valori che può assumere *state* sono:

- *SDL_APPMOUSEFOCUS* si riferisce al focus del mouse, ovvero se il puntatore si trova o meno all'interno della finestra;
- *SDL_APPINPUTFOCUS* è invece associato al focus per la nostra applicazione, ovvero se quest'ultima è la finestra attiva o meno;
- *SDL_APPACTIVE* infine viene utilizzato nel caso in cui la finestra venga ridotta a icona (0) o ripristinata (1).

Tutti i controlli si effettuano tramite AND (&) binario

4.6.5 Gli eventi del mouse

Similarmente a quanto visto per l'input da tastiera, anche a questi due tipi in *SDL_Event* fa capo una sola struttura. Il membro *state* permette ancora una volta di determinare se si tratta di una pressione o di un rilascio (i valori associati sono ancora *SDL_PRESSED* e *SDL_RELEASE*); il passo successivo è controllare di quale pulsante si tratta. Questa informazione sta nel sottomembro *button* ; i pulsanti sono mappati come visibile qui di seguito nella tabella 4.3:

Pulsante	Valore di button
Sinistra	SDL_BUTTON_LEFT
Destro	SDL_BUTTON_RIGHT
Centrale	SDL_BUTTON_MIDDLE

Tabella 4.3: Funzioni per intercettare gli eventi dei bottoni del mouse.

Quindi, a fronte di un evento di questo tipo, è possibile testare lo stato dei tasti che interessano, ad esempio:

```
if (button->button == SDL_BUTTON_LEFT )  
...
```

Per quanto riguarda i mouse con la rotella, nelle ultime versioni della libreria sono stati definiti:

- *SDL_BUTTON_WHEELUP*
- *SDL_BUTTON_WHEELDOWN*

che identificano il valore attribuito all'uso di essa nei due movimenti possibili; Infine è possibile recuperare anche la posizione del mouse in cui è stato premuto, dai sottomembri *x* e *y*:

```
printf ( " pos (%d, %d ) \n", button->x, button->y );
```

Ogni volta che il mouse viene mosso, un evento di questo tipo viene inserito nella coda, a patto che l'applicazione abbia i tre focus già visti nella discussione di *SDL_ACTIVEEVENT*; questo si traduce nel fatto che l'applicazione deve essere visibile, avere il focus di finestra e il puntatore del mouse deve trovarsi sopra la sua superficie. Dalla struttura è possibile ricavare molti dati interessanti; in *x* e *y*, come ci si poteva "aspettare", si trovano le coordinate della nuova posizione del mouse.

```
Printf("SDL_MOUSEMOTIONEVENT event->pos( %d [%d] , %d [%d] )\n",  
motion->x, motion->xrel, motion->y, motion->yrel );
```

La cosa più interessante da notare è che in *xrel* e *yrel* viene registrata la quantità di spostamento tra la posizione attuale e la precedente; in questo modo è possibile ricavarsi non solo la posizione del mouse, ma anche la sua velocità istantanea (ovviamente conoscendo l'intervallo di tempo tra una misurazione e la successiva). Grazie alla macro *SDL_BUTTON()* è, inoltre, possibile ricavare quali pulsanti del mouse sono premuti mentre si verifica lo spostamento; tutto quello che si deve fare è un *and (&)* binario tra il sottomembro *state* e la macro, scegliendo l'alias del pulsante che si vuole controllare:

```
if (motion->state & SDL_BUTTON (SDL_BUTTON_LEFT) )
```

```
printf ("Pulsante sinistro\n");
```

Questo metodo di controllo è diverso da quello del precedente evento; in questo caso, infatti, sappiamo che il mouse si è mosso, ma a priori potremmo non aver tenuto traccia dello stato dei pulsanti: da qui la necessità di poterli controllare tutti in un colpo solo.

4.7 Video

Se il programmatore volesse impostare un titolo per la finestra e magari una icona rappresentativa, questo è possibile tramite le funzioni:

- `void SDL_WM_SetCaption (const char *title, const char *icon);`
- `void SDL_WM_SetIcon (SDL_Surface *icon, Uint8 *mask);`

La prima, intuitivamente, imposta il nome dell'applicazione e quello dell'icona associata. Bisogna far attenzione in quanto si intende non il percorso del file grafico contenente l'icona, ma il suo nome (la confusione tra queste due cose è un errore molto comune). La seconda, carica effettivamente l'icona in questione e la passa al gestore di finestre. Il file grafico deve essere scelto preferibilmente di dimensioni 16x16 o 32x32 (se si lavora in ambiente Windows, la dimensione deve essere 32x32), caricato con `SDL_LoadBMP()` e la superficie risultante passata a `SDL_WM_SetIcon()`.

Si supponga di avere un'immagine già pronta, e di voler definire come trasparente una regione della sua superficie. È possibile decidere che un qualsiasi colore RGB venga interpretato come *colore trasparente* e che tutti i pixel di quel colore semplicemente non debbano essere disegnati. Questo colore viene detto *color key*. Si tratta, fra l'altro, della stessa tecnica usata in alcune riprese cinematografiche: si fanno recitare gli attori su uno sfondo completamente blu (detto *blue screen*), per poi aggiungere una scenografia al posto del blu (che funge quindi da sfondo trasparente rispetto agli attori). È possibile usare questa tecnica per definire quali pixel della nostra icona non debbano essere disegnati. Per prima cosa si carica la bitmap:

```
img = SDL_LoadBMP ( "image.bmp" );
```

e si supponga che l'immagine sia circondata da un colore rosso che verrà settato come *color key*:

```
SDL_SetColorKey (img, SDL_SRCCOLORKEY, SDL_MapRGB(img->format, 255, 0, 0));
```

Infine, si comunica a SDL che deve essere utilizzata come icona dal window manager:

```
SDL_WM_SetIcon ( img, NULL );
```

4.7.1 SDL_VIDEORESIZE

Questo evento vedi figura BB precedente viene generato in seguito al ridimensionamento della finestra. Bisogna comunque far attenzione in quando questo non include la riduzione a icona né l'eventuale successivo ripristino dell'applicazione.

Come già accennato, quando le dimensioni della finestra cambiano, viene generato questo evento, ma il framebuffer non viene ridimensionato automaticamente; la prima cosa da fare, quindi, sarà liberare la superficie in uso con la funzione

```
SDL_FreeSurface( screen );
```

e ricrearla, con le nuove dimensioni che vengono ricevute dai membri *w* e *h* rispettivamente larghezza e altezza della struttura *resize*

```
screen = SDL_SetVideoMode( resize->w, resize->h, vid_bpp, vid_flag );
```

visto che la superficie in uso fino a quel momento è stata liberata, su quella appena allocata è buona norma forzare un ridisegno del fotogramma corrente senza attendere ulteriori eventi.

4.7.2 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 4.4: struttura *SDL_VideoInfo*

Con la funzione *SDL_VideoDriverName()* è possibile ottenere una stringa che descrive il driver in uso, ad esempio X11 o DirectX. Le informazioni riguardanti l'hardware video si ricavano con *SDL_GetVideoinfo ()*, che ritorna un puntatore a una struttura *SDL_VideoInfo* (vedi figura CC 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 (vedi tabella 4.4). 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.

4.8 Pixel format

L'hardware video determina, secondo la modalità video impostata, un formato di rappresentazione dei pixel in memoria che prende il nome di Pixel Format. SDL permette di accedere ai pixel di una surface tramite il membro *pixels* della struttura *SDL_Surface*; così facendo, è compito del programmatore rispettare il formato per manipolare correttamente la mappa di pixel. Esistono fondamentalmente due *famiglie* di Pixel Format:

1. a 8 bit per pixel o anche *Palettized*
2. a 16, 24 e 32 bit per pixel o anche *TrueColor*.

Se si usa un Pixel Format a 8 bit ad ogni pixel è possibile associare un valore nell'intervallo 0-255 che ne individua univocamente il colore, ricavabile tramite una tabella di look-up detta appunto palette: questa contiene 256 elementi, per ognuno dei quali sono definite le tre componenti *R(rosso)*, *G(verde)*, *B(blu)*.

Questo metodo, retaggio del modo grafico VGA³¹ standard *13h* (320x200x256 colori), presenta ovviamente alcuni svantaggi, ad esempio il numero massimo di colori utilizzabile in una immagine è proprio 256, le cui tonalità possono comunque essere scelte tra un set abbastanza ampio.

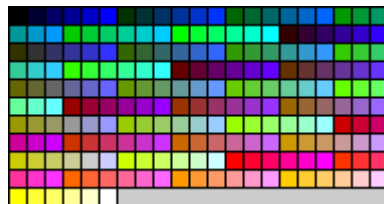


Figura 4.1: Esempio di palette

Il particolare approccio seguito in questo caso presenta però alcune peculiarità, come l'occupazione ridotta di memoria video e la possibilità di effettuare particolari modifiche all'immagine toccando solo la palette (figura 4.1). Se, ad esempio, si volessero oscurare tutti i pixel dell'immagine che hanno un certo colore basterebbe modificare quest' ultimo direttamente sulla palette, senza toccare la memoria video. E' anche possibile realizzare effetti particolari (cascate, suolo a scorrimento parallattico, etc.) semplicemente ruotando i colori contenuti in un certo intervallo della palette. Queste tecniche erano molto utilizzate nei videogiochi di un decennio fa, quando il modo *13h* (e derivati) era lo standard di fatto per le applicazioni di questo tipo. Con i Pixel Format delle modalità TrueColor i valori delle componenti RGB vengono codificati direttamente nella memoria assegnata al pixel, senza bisogno di passare da una tabella ausiliaria; nei modi a 16 bit occupiamo, quindi, due byte per pixel e in quelli a 24 e 32 rispettivamente tre e quattro byte. La codifica a 24 bit è senza dubbio la più intuitiva: ciascuno dei tre byte che descrivono il pixel contiene il valore di una delle tre componenti, mappato in un intervallo 0-255. Questo sistema consente di rappresentare l'intero spettro dei colori visualizzabili su un

³¹ VGA:Video Graphics Adapter, adattatore per la grafica video. È lo standard introdotto da IBM nel 1987, per le schede grafiche, con risoluzione 640x480 pixel a 16 colori.

monitor ($256 \times 256 \times 256 = 16777216$ colori). Nelle modalità a 32 bit possono essere affiancate altre informazioni alle componenti RGB, codificandole negli 8 bit aggiuntivi; una possibilità è l'aggiunta di una componente detta *alpha*, che descrive la trasparenza del pixel: in questo caso si parla di RGBA. Un discorso a parte deve essere fatto per i modi a 16 bit, per i quali si ha a disposizione due *Pixel Format* noti come 555 e 565, in cui ogni cifra rappresenta il numero di bit assegnati rispettivamente alla componente rossa, verde e blu. Nel modo 555, il primo bit del blocco è sempre 0 (zero). Viene naturale da chiedersi il perché di questa complicazione: nei primi modelli di schede video che supportavano i modi TrueColor 16 bit si decise di assegnare un numero uguale di bit per rappresentare le tre componenti, riservando il primo bit per usi interni. Successivamente, considerato che l'occhio umano è più sensibile alle variazioni di tonalità del verde, si cominciarono a dedicare 6 bit alla rispettiva componente e 5 bit alle altre due, mantenendo inalterato l'ordine di memorizzazione (da qui 565), ma nel frattempo esisteva una grande quantità di applicazioni che faceva uso del Pixel Format 555, così i produttori dei chip grafici resero disponibili entrambi i *Pixel Format* all'interno delle loro schede. Successivamente ci si rese conto che per memorizzare una texture in formato compresso era particolarmente comodo il formato 555 (utilizzato nell'algoritmo *FXT1*³² di 3dfx [3DF04] e nel *S3TC*³³ di S3 [S304]).

Il *pixel format* viene reso noto attraverso la struttura SDL che si vede anche nella figura DD, ovvero `SDL_PixelFormat`. I membri raccolgono tutte le informazioni necessarie alla gestione dei casi precedentemente analizzati, come il puntatore alla palette e le maschere attraverso le quali è possibile sapere quanti e quali bit sono dedicati alle varie componenti di colore. Tutte le informazioni sono sufficientemente autoesplicative (vedi tabella 4.5). Le funzioni `SDL_MapRGB()` e `SDL_GetRGB()` facilitano il compito prendendosi carico delle corrette conversioni da valore del pixel a componente di colore e viceversa secondo il Pixel Format in uso.

³² FXT1: Tecnica di compressione delle texture

³³ S3TC: Algoritmo di compressione per le schede grafiche elaborato da S3.

```

typedef struct SDL_PixelFormat {
    SDL_Palette * palette;
    UintB   BitsPerPixel;
    Uint8   BytesPerPixel;
    Uint8   Rloss;
    Uint8   Gloss;
    Uint8   Bloss;
    Uint8   Aloss;
    Uint8   Rshift;
    Uint8   Gshift;
    Uint8   Bshift;
    Uint8   Ashift;
    Uint32   Rmask;
    Uint32   Gmask;
    Uint32   Bmask;
    Uint32   Amask;
    Uint32   colorkey;
    // Valore di opacità globale per la superficie (per-surface
    //alpha biending)
    Uint8   alpha;
} SDL_PixelFormat;

```

Tabella 4.5: SDL_PixelFormat

BitsPerPixel e *BytesPerPixel* informano il programmatore di quanti bit e bytes servono per rappresentare un singolo pixel nel pixel format. Tipicamente i valori per *BitsPerPixel* sono 8, 16, 24, o 32, mentre quelli di *BytesPerPixel* sono 1, 2, 3 e 4. Se il membro *BitsPerPixel* è 8, ci sarà una palette, altrimenti gli altri membri che iniziano con R,G e B avranno un valore diverso da zero. Si possono individuare 3 gruppi che rappresentano:

- la maschera: Rmask, Gmask, Bmask o Amask;
- i valori di shift: Rshift, Gshift Bshift o Ashift;
- i valori di loss: Rloss, Gloss, Bloss o Aloss.

Naturalmente per R, G, B e A si intende rispettivamente il valore Rosso, Verde, Blu, e Alpha di trasparenza. Per capire il significato di maschera, shift e lost consideriamo la tabella 4.6

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	R	R	R	R	G	G	G	G	G	G	B	B	B	B	B

Tabella 4.6: Esempio del formato RGB a 16 colori.

I primi 5 bit a partire da sinistra rappresentano il colore rosso gli ultimi 5 dallo 0 al 4 rappresentano il blu e i sei bit del mezzo rappresentano il verde. Nel caso si attribuisca un numero binario con valore 1 per il colore rosso e 0 per tutti gli altri non rossi, si ottiene una maschera di colore come nella seguente tabella 4.7

1111 1000 0000 0000 b

Tabella 4.7: Rappresentazione della tabella 4.6 in valore binario associando il valore 1 al colore rosso e 0 per tutti gli altri.

Poiché lavorare con un sistema binario è alquanto difficile, cambiando questa rappresentazione in un valore esadimale come F800, si ottiene la maschera per il valore Rmask. Per determinare il valore Rshif si deve traslare la maschera Rmask di un bit fino a quando si trova il valore 1 nel bit 0. Nell'esempio riportato sopra la traslazione avviene 11 volte, quindi Rshift è 11.

Per Rloss si intende la differenza tra il bit 8 di rappresentazione di un canale di colore e la rappresentazione del colore nel formato attuale.

Per poter ottenere significativi miglioramenti di prestazioni nello sviluppo di un'applicazione SDL bisogna tener presente di alcuni punti fondamentali quali:

- il blitting RLE;
- uso del canale alpha;
- double buffer;
- temporizzazione della animazione.

4.9 Blitting

Nel caso del blitting³⁴ bisogna convertire tutte le surface che si utilizzano nel Pixel Format del display e specificare, nel caso di sprite con colorkey³⁵, l'utilizzo dell'accelerazione *RLE - Run Length Encoding*³⁶. La prima si ottiene caricando la superficie su una surface temporanea, da usare come parametro alla successiva chiamata a *SDL_DisplayFormat()* che restituirà in una nuova surface l'immagine di partenza opportunamente convertita; la seconda si può ottenere aggiungendo il parametro *SDL_RLEACCEL* alla chiamata *SDL_SetColorKey()*; la surface verrà ricodificata una volta per tutte alla seguente chiamata di *SDL_DisplayFormat()* o *SDL_BlitSurface()*. A questo punto le superfici sono ottimizzate per il blitting sulla superficie video; RLE, inoltre, elimina gli accessi alla memoria relativi ai pixel considerati trasparenti secondo il colorkey attivato. Spesso è utile andare oltre al concetto di trasparenza e considerare invece l'opacità di una superficie, ovvero la percentuale con la quale l'immagine sovrastante si combina con la regione sottostante alpha blending. SDL tratta l'alpha blending in due modi:

- 1) utilizzando l'aggiuntivo canale alpha proprio della superficie caricata;
- 2) per quelle che non lo hanno, permettendo di assegnare un valore di opacità globale.

Nel primo caso è spesso vantaggioso usare formati grafici che supportano nativamente il canale alpha, come ad esempio il PNG (è necessario l'uso di *SDL_Image* o altre librerie esterne); nel secondo caso si parla di *per-*

³⁴ Blitting: Sovrapporre un'immagine ad un'altra

³⁵ Colorkey: colore definito dal programmatore, corrispondente alla trasparenza assoluta.

³⁶ RLE: è un tipo di compressione che lavora su sequenze di valori uguali.

surface alpha blending³⁷, che è possibile manipolare con la funzione *SDL_SetAlpha()*.

4.9.1 Double Buffering

Questa tecnica consiste nel disegnare la scena da mostrare successivamente su un buffer ausiliario, che verrà mostrato in corrispondenza del prossimo sincronismo verticale del monitor. Per attuarla con SDL è necessario specificare il flag *SDL_DOUBLEBUF* al momento della inizializzazione della modalità video (*SDL_SetVideoMode()*); inoltre, dopo tutte le istruzioni necessarie a disegnare il frame si dovrà scambiare il backbuffer con il frontbuffer chiamando semplicemente la funzione *SDL_Flip()* al posto di *SDL_UpdateRect()*.

4.10 Funzioni Avanzate

4.10.1 Bump Mapping

Il bump mapping è una tecnica che permette di simulare la presenza di rugosità su una superficie: le sue applicazioni spaziano dalle vecchie intro grafiche 2D all'attuale uso nella grafica 3D in combinazione alle normali tecniche di texturing, in modo da aumentare il realismo e il dettaglio dei modelli senza incrementare o addirittura riducendo il numero di poligoni. Il primo passo è definire, per la superficie alla quale verrà applicato l'effetto, una mappa di altitudine: si tratta solitamente di una immagine a 256 toni di grigio con le stesse dimensioni di quella di base, i cui pixel descrivono lo scostamento in altitudine del punto corrispondente in base al loro colore. Il valore intermedio, 127 corrispondente al grigio neutro, indica che in presenza di quel punto non ci sono sporgenze o rientranze; un valore alto indica una sporgenza e al contrario un valore basso una rientranza. Infine, occorre un modello di illuminazione, ovvero la descrizione della forma di luce proiettata dalla sorgente. Durante il loop di

³⁷Per-surface alpha blending: viene specificata una singola componente alpha per l'intera superficie, che risulta essere uniformemente opaca rispetto allo sfondo su cui disegnata.

disegno, per ogni punto verrà calcolata la pendenza nelle direzioni degli assi X e Y sfruttando la mappa di altitudine; il risultato verrà presentato come perturbazione sul modello di illuminazione per scegliere il valore di intensità da associare al punto. Infine, tramite una tabella look-up di colori, verranno recuperate le componenti con cui disegnare il pixel.

4.10.2 Lock Surface

In alcuni casi quando il Pixel Format della superficie video non è supportato dall'hardware e SDL lo emula via software, per accedere alla superficie è necessario eseguirne il lock: questa operazione arresta gli accessi alla superficie da parte della libreria, la converte in un formato utilizzabile (ad esempio, decompime le surface RLE) e la rende disponibile sia in lettura che in scrittura. È consigliabile sbloccare l'immagine non appena si siano terminate le procedure di disegno, visto che tenere una superficie sotto lock consuma risorse, non permette il blitting di questa e può causare problemi in coincidenza di chiamate di sistema. Le due funzioni *SDL_LockSurface()* e *SDL_UnLockSurface()* vengono di solito usate condizionalmente al valore ritornato dalla macro *SDL_MOSTLOCK()* che si occupa di verificare la necessità di effettuare il lock della superficie.

4.11 L'audio.

La riproduzione da parte di un computer di un suono proveniente dall'esterno può avvenire solo se esso è stato precedentemente convertito in una forma comprensibile all'elaboratore, tramite un processo che prende il nome di campionamento: questo consiste nel memorizzare ad intervalli di tempo regolari i valori dell'ampiezza della forma d'onda che rappresenta il suono che interessa. L'accuratezza di questo processo è regolata principalmente da due parametri:

1. frequenza;
2. precisione di campionamento.

La frequenza, misurata in Hz, indica quanto spesso il segnale viene campionato nell'unità di tempo: un valore di 44.100Hz, ad esempio, indica che per ogni secondo di registrazione l'ampiezza del segnale viene misurata e memorizzata 44.100 volte in corrispondenza di altrettanti istanti di tempo equamente distanti tra loro.

La precisione di campionamento regola invece la quantità di memoria utilizzata per memorizzare ogni campione nella sua forma digitale: tipicamente 8 o 16 bit per sistemi non professionali. I campioni ottenuti possono essere immagazzinati, elaborati e successivamente riprodotti tramite una opportuna periferica, esempio la scheda audio. Migliore è il campionamento di partenza e minore sarà, ovviamente, la differenza qualitativa di riproduzione rispetto al segnale originale. La sequenza di campioni ottenuta da un'onda sonora può essere memorizzata in una forma digitale molto semplice detta *PCM*³⁸, che, tra l'altro, è una di quelle utilizzate internamente dai formati *wave* e *au*. I valori dei campioni rappresentano l'ampiezza della forma d'onda originale, pertanto è possibile amplificare o attenuare il segnale semplicemente moltiplicandoli per un valore rispettivamente maggiore o minore di 1; per ottenere la sovrapposizione di due segnali è sufficiente eseguire la media tra i loro campioni e così via. Risulta, quindi, intuitivo lavorare nel dominio del tempo su segnali *PCM*; per operazioni sul dominio della frequenza è necessaria invece l'analisi dei dati usando la trasformata di Fourier. Naturalmente, si ha spesso a che fare con file di campioni audio in forma compressa in modo da risparmiare spazio su disco. La scheda audio, per riprodurre un segnale *PCM*, non fa altro che convertire in analogico i campioni forniti al driver audio dall'applicazione. Supponendo di trattare un suono stereo campionato a 44,1 KHz e precisione a 16 bit ogni secondo dobbiamo inviare $44.100 \times 2 \text{ (16 bit)} \times 2 \text{ (entrambi i canali)} = 176.400$ byte di dati. Il trasferimento avviene solitamente utilizzando il DMA³⁹, un dispositivo hardware in grado di scambiare dati tra la memoria centrale e le

³⁸ PCM : Pulse Code Modulation

³⁹ DMA: Direct Memory Access controller

periferiche svincolando la CPU da questo compito. Al momento opportuno l'attivazione di un segnale di interrupt avvertirà il sistema che sono necessari altri dati per proseguire la riproduzione, quindi l'applicazione che effettua il playback dovrà fornire i nuovi campioni.

Il supporto dell'audio fornito è essenziale, ma efficace; per un semplice riproduttore di file wave (.wav) con interfaccia a riga di comando, per esempio bisogna prima di tutto inizializzare il sottosistema audio utilizzando *SDL_init()*, specificando questa volta il flag *SDL_INIT_AUDIO*. Per poter caricare un file .wav in memoria facendosi carico della conversione in PCM dal formato memorizzato nel file si utilizza *SDL_LoadWAV()*. *SDL_Loadwav()* richiede quattro parametri, uno è il nome del file da caricare; gli altri tre sono usati dalla funzione come indirizzi per memorizzare rispettivamente una struttura di tipo *SDL_AudioSpec*, contenente i dati necessari per la successiva riproduzione, l'indirizzo di memoria a partire dal quale sono memorizzati i campioni PCM e la lunghezza dell'intero suono in byte:

```
SDL_AudioSpec * SDL_LoadWAV
(const char * file, SDL_AudioSpec * spec, Uint8 **audioBuf,
 Uint32 *audio_len) ;
```

La funzione restituisce l'indirizzo della struttura *SDL_AudioSpec* specificata in caso di successo, NULL se il caricamento non è riuscito. E' da notare che il buffer dedicato ai dati audio *audio_buf* viene allocato da *SDL_LoadWAV()* e, al termine del suo utilizzo, dovrà essere deallocato dalla memoria con l'apposito *SDL_FreeWAV()*.

A questo punto, dati i campioni da elaborare, che si andranno a copiare sul buffer audio per poter avviare la riproduzione, si renderà necessaria la preparazione o semplicemente la copia periodica dei campioni nel buffer audio che verrà utilizzato dal driver della periferica. L'opportuna cadenza di questa operazione è essenziale al fine di evitare indesiderate interruzioni nella fase di playback (buffer underrun). Questo meccanismo viene realizzato in SDL tramite

l'uso di un thread dedicato per effettuare le operazioni di copia sul buffer audio: non appena l'hardware è pronto per ricevere nuovi campioni invia una notifica a seguito della quale il thread richiama una funzione detta di *callback* puntata da un membro della struttura *SDL_AudioSpec*. Il suo compito è quello di effettuare le eventuali operazioni sui campioni e di copiarli sul buffer di destinazione il più velocemente possibile. Naturalmente, tutto ciò che è relativo al controllo dell'hardware audio, del DMA e del thread viene svolto da SDL e dal driver audio; si è così svincolati dall'architettura del sistema ed è possibile dedicare tempo completamente alla programmazione della applicazione. I campioni destinati al playback sono, quindi, preparati in blocchi da inviare sequenzialmente. Questo metodo ha un problema intrinseco ovvero quello della latenza: infatti, non è possibile fornire all'hardware audio un nuovo blocco di campioni finché quello precedente è in fase di riproduzione, introducendo quindi un piccolo ritardo tra il momento in cui si decide di emettere un determinato suono e la sua effettiva emissione. Per ridurre la latenza è necessario diminuire la quantità di campioni che vengono copiati ogni volta, ovvero ridurre la dimensione del buffer. Così facendo si accorcerebbe però anche il tempo che intercorre sia tra due accessi consecutivi ai campioni che tra le preparazioni dei dati: questo si traduce in più lavoro per l'hardware e, quindi, in maggiore probabilità di non riuscire a rendere disponibili i nuovi campioni nel tempo previsto. Come si può intuire, quello che viene fatto è cercare un compromesso accettabile. Pertanto è necessario analizzare la struttura *SDL_AudioSpec*. Al suo interno sono presenti i membri *samples* e *size*, che rappresentano rispettivamente il numero di campioni da copiare ogni volta nel buffer e l'occupazione in memoria di quest'ultimo in byte. La funzione *SDL_LoadWAV()* imposta automaticamente *samples* a 4096 e *size* di conseguenza: si vede facilmente che con uno stream registrato a 44.1KHz questo comporta una latenza finale di circa 90ms, un valore ovviamente accettabile nel caso di riproduzione passiva. Per suoni da riprodurre dinamicamente, il tipico esempio è quello di effetti legati ai vari eventi di un videogioco, è invece più indicato un valore di latenza minore, ad esempio

50ms. Per ottenerlo, sempre considerando uno stream a 44.1KHz, bisogna diminuire il valore di *samples* a 2205.

4.12 CD-ROM

Alcuni giochi, oltre ad avere effetti e colonne sonore molto appropriate, permettono al giocatore di utilizzare come sottofondo musicale le tracce audio preferite dei propri CD; al momento in cui un livello di gioco viene superato si passa automaticamente alla traccia successiva. Questa è solo una delle tante occasioni in cui una comoda interfaccia per gestire il lettore CD-ROM è indubbiamente opportuna. Il supporto per il CD-ROM si attiva con il flag `SDL_INIT_CDROM` con la chiamata a `SDL_Init()` O `SDL_InitSubSystem()`, in caso di successo è possibile far uso delle funzioni specifiche del sottosistema. È possibile innanzitutto vedere quante unità sono presenti: questa informazione è ottenuta con la funzione

```
int  SDL_CNumDrives(void);
```

il risultato 0 indica che nessun lettore CD è stato rilevato. In caso contrario, ognuno di questi ha un nome che lo identifica, che è possibile ricavare con:

```
const char *SDL_CDName(int drive);
```

Il parametro `drive` deve essere naturalmente compreso tra 0 e il numero delle unità presenti, meno 1. Secondo il sistema operativo usato, verrà restituita una stringa contenente l'identificativo dell'unità (es. D:) oppure il percorso del device (es. /dev/cdrom). L'accesso ad uno o più di essi è consentito in seguito alla chiamata di:

```
SDL_CD *SDL_CDOpen(int drive);
```

Il valore di ritorno è un puntatore ad una struttura di tipo `SDL_CD` che contiene le informazioni del lettore CD-ROM: la traccia in riproduzione e la posizione all'interno di essa, lo stato del cassetto, le informazioni relative al contenuto del

disco inserito ecc... Queste, in particolare, sono contenute in un array di strutture `SDL_CDTrack`. La dimensione dell'array è fissata come `SDL_MAX_TRACKS` il cui valore è definito come il massimo numero di tracce imposte dal RedBook⁴⁰. Naturalmente, solo le prime `numtracks` posizioni dell'array contengono le informazioni riguardo al disco presente nel lettore.

La struttura `SDL_CD` servirà anche per utilizzare tutte le altre funzioni messe a disposizione dalla libreria per il lettore CD; in questa sono contenute utili informazioni, l'aggiornamento di esse può essere effettuato in qualsiasi momento con la funzione

```
CDstatus SDL_CDStatus(SDL_CD *cdrom);
```

che, oltre a settare opportunamente i membri di `cdrom` restituisce un valore tra quelli enumerati da `CDstatus`, molto utile per avere un feedback immediato dello stato del lettore. Questo può essere utilizzato con la macro `CD_INDRIVE` (status) che facilita il controllo della presenza di un disco nel lettore preso in considerazione. L'utilizzo effettivo del dispositivo si ottiene attraverso le due funzioni di playback con le quali è possibile specificare le tracce o la porzione di CD da riprodurre. Queste, come del resto le strutture precedentemente accennate, fanno uso dell'unità di misura frame. Nel formato audio digitale definito nel RedBook, i dati sono memorizzati in frame, la cui lunghezza è definita come 1/75 di secondo a 44.1 KHz di campionamento, ognuno di essi è quindi costituito da 2352 byte ($44100 \times 2 \times 2 / 75$). In pratica, il frame non è altro che l'unità di dati base di un CD, per cui in un secondo di musica ci sono `CD_FPS` frame (questa costante è appunto definita in SDL come 75).

Questa informazione permette, con alcuni semplici calcoli, di ottenere il corrispondente tempo in minuti e secondi espresso da una certa quantità di frame, ma vista la frequenza con la quale ricorre l'uso di tale conversione sono state scritte appositamente due macro:

⁴⁰ RedBook: Guida ufficiale propria di ogni sistema operativo. Chiamato così dal colore della copertina.

FRAMES_TO_MS⁴¹ () e MSF_TO_FRAMES ()

La prima, dato un numero di frame come primo parametro, imposta il valore contenuto nei tre successivi interi passati per indirizzo, con il valore di tempo convertito in minuti, secondi e settantacinquesimi di secondo (ovvero i frame che rimangono dalla conversione in minuti e secondi). La seconda macro, invece, compie l'operazione inversa: dato un tempo in minuti, secondi e frame lo converte tutto in frame. A questo punto, si passa a considerare i prototipi delle due funzioni di playback di SDL. Una di esse è

```
Int SDL_CDPlay(SDL_CD *cdrom, int start, int lenght);
```

che suona una quantità *length* di frame a partire dal frame *start* relativo all'inizio del disco contenuto nel lettore indicato da *cdrom*. I membri delle strutture contenute nel vettore dei dati delle tracce forniscono le informazioni sufficienti per individuare l'inizio e la durata, in frame, di ogni traccia (membri *offset* e *length*) ma, in alcuni casi, può essere decisamente più comodo utilizzare:

```
int SDL_CDPlayTracks(SDL_CD *cdrom, int start_track, int  
start_frame, int ntracks, int nframes);
```

Per avere un controllo più preciso è possibile specificare dei valori diversi da zero per i due parametri *start_frame* e *nframes*. Il primo indica l'offset in frame, rispetto all'inizio della traccia di partenza, dal quale iniziare il playback; il secondo è il numero del frame, successivo alla fine dell'ultima traccia da suonare, al quale terminare la riproduzione. Durante il playback il flusso del programma continua indisturbato, per avere informazioni sullo stato della riproduzione è sufficiente chiamare *SDL_CDStatus()* con l'handle del CD-ROM desiderato in modo che le informazioni contenute nella struttura *SDL_CD* possano essere correttamente aggiornate. A questo punto, i membri *cur_track*, *cur_frame* e *status* di questa struttura conterranno tutto ciò che serve per

⁴¹ MSF: Acronimo di Mnute Second Frame

mostrare l'avanzamento o per intraprendere azioni particolari. È da notare che il playback viene effettuato dall'hardware del lettore. L'audio prodotto può essere ascoltato utilizzando l'apposito ingresso frontale per le cuffie, oppure attraverso le normali casse nel caso in cui sia presente il cavo di collegamento interno tra lettore e scheda audio: in quest'ultimo caso le impostazioni del mixer della scheda sonora influiscono sul volume di riproduzione. Le altre funzioni sono di uso immediato, il loro unico parametro è, come al solito, un puntatore ad una struttura *SDL_CD* restituito da *SDL_CDOpen()* che indica il CD-ROM desiderato. Il valore da esse restituito per l'operazione richiesta è 0 in caso di successo, -1 in caso di fallimento. I nomi di queste funzioni sono *SDL_CDPause()*, *SDL_CDResume()*, *SDL_CDStop()*, *SDLCDEject()*. Infine, *SDL_CDClose()* libera l'handle creato da *SDL_CDOpen()* e quindi tutte le risorse allocate dalla libreria per la gestione del dispositivo.

4.13 Joystick

Il sottosistema per la gestione del joystick, attivabile con il flag *SDL_INIT_JOYSTICK* nella chiamata a *SDL_init()*, mette a disposizione due differenti approcci all'integrazione di questa periferica di controllo all'interno delle proprie applicazioni. Anche in questo caso, è possibile trovare configurazioni nelle quali sono presenti più dispositivi dello stesso tipo. Per conoscerne il numero è necessario usare:

```
int SDL_NumJosticks(void);
```

Ad ognuno di questi è associato un identificativo che può essere, a seconda dei casi, o il nome e modello della periferica o semplicemente una stringa che ne descrive le caratteristiche:

```
const char *SDL_JoystickName(int index);
```


L'handle che permette di riferirsi ad un particolare joystick presente nel sistema, in modo tale da poterlo utilizzare con le altre funzioni di SDL, viene restituito, sotto forma di puntatore ad una struttura `SDL_Joystick`, dalla funzione:

```
SDL_Joystick *SDL_JoystickOpen(int index);
```

A questo punto, si hanno due alternative:

1. usare il sistema di gestione di eventi di SDL;
2. utilizzare direttamente le funzioni che restituiscono lo stato del controller.

Nel primo caso si deve attivare la gestione degli eventi attraverso la funzione

```
Int SDL_JoystickEventState(int state);
```

state può assumere i valori *SDL_ENABLE*, *SDL_IGNORE*, *SDL_QUERY*.

Nel primo caso si specifica che gli eventi generati dal joystick devono essere immessi nella coda, nel secondo si richiede il contrario. Il terzo permette, invece, di sapere l'impostazione attuale, mediante il ritorno della funzione di uno dei primi due valori appena visti. I campi *axis*, *button*, *hat* e *ball* di queste strutture specificano, per il joystick indicato, quale componente ha generato l'evento fornendo l'indice di esso; i campi rimanenti conterranno i valori ad esso relativi. Nel far uso di questo metodo di rilevamento, è opportuno ricordare che, per ogni singolo frame dell'applicazione verranno gestiti tutti gli eventi messi in coda fino a quel momento. La pressione simultanea di più pulsanti e spostamenti combinati dei vari componenti saranno, quindi, probabilmente processabili nello stesso ciclo di aggiornamento. Inoltre, come intuibile, un evento viene messo in coda nel momento in cui SDL, attraverso la chiamata implicita a *SDL_JoystickUpdate()*, individua una differenza tra i dati riferiti alla lettura precedente del dispositivo e quella attuale.

Può, quindi, essere necessario memorizzare tale informazione in modo da usarla per spostamenti time-based la cui velocità deve variare secondo la posizione sugli assi, oppure quando devono essere eseguite azioni fin tanto che un pulsante o un hat è in un determinato stato. L'altra soluzione consiste nel leggere lo stato del componente desiderato del joystick che prendiamo in esame attraverso le specifiche funzioni di SDL. I dati che esse restituiscono sono aggiornati all'ultima chiamata della funzione `SDL_JoystickUpdate()` che memorizza i valori attuali per ogni joystick in una struttura interna. A differenza di quanto visto precedentemente, la funzione di aggiornamento deve essere chiamata esplicitamente. L'API mette a disposizione:

```
Sint16 SDL_JoystickGetAxis(SDL_Joystick * joystick, int axis);
Uint8  SDL_JoystickGetButton(SDL_Joystick * joystick, int button);
Uint8  SDL_JoystickGetHat (SDL_Joystick * joystick, int hat);
int     SDL_JoystickGetBall(SDL_Joystick * joystick, int ball, int *dx, int *dy);
```

La prima restituisce un valore compreso nel range intero -32768 e 32767 relativo alla posizione sull'asse specificato dall'indice `axis` per il controller individuato con joystick. Questo, analogamente a come si fa nel caso della gestione tramite eventi, potrà essere trattato, in seguito ad opportune conversioni, come velocità di spostamento o posizione a seconda delle necessità. La funzione successiva, come è facilmente intuibile, restituisce lo stato del *i-esimo* pulsante del joystick in esame: i valori possibili sono 1 nel caso sia premuto, 0 altrimenti. La terza funzione fornisce lo stato dell'HAT di indice `hat` attraverso un valore interpretabile come risultato dell'OR logico delle opportune costanti definite dalla libreria SDL. Infine, l'ultima permette di ottenere i valori di incremento sui due assi causati dalla rotazione della *i-esima* trackball, cioè lo spostamento relativo di essa.

4.14 Multithreading

Nel precedente paragrafo si è visto come la gestione dell'audio possa essere implementata facendo uso di un thread apposito, come avviene nel caso di SDL:

in quel caso era richiesto al programmatore di fornire una opportuna funzione di *callback*, che sarebbe stata eseguita in maniera asincrona rispetto al resto del programma, per rifornire di campioni la scheda audio. Verrà ora mostrato come sia possibile fare uso di thread all'interno del codice assegnando loro compiti arbitrari.

Generalmente, su ogni macchina Linux o Windows c'è una certa quantità di processi che vengono eseguiti contemporaneamente. All'inizio dell'esecuzione di ognuno di questi, il sistema operativo associa un opportuno descrittore detto PCB⁴², una struttura che contiene informazioni sulla memoria allocata, l'istruzione e lo stack pointer attuali, un elenco delle risorse in uso dal processo ed altri dati.

Ogni singolo processore può ovviamente eseguire le istruzioni di un solo processo alla volta, l'avanzamento contemporaneo di più di essi viene simulato grazie al Context Switch: ogni volta che un processo viene messo in pausa il sistema operativo ne salva lo stato attuale nella struttura apposita e passa ad eseguire un altro processo, a partire da dove era stato lasciato, usando le informazioni contenute nel relativo descrittore. Questo sistema comporta, però, l'impegno di una quantità di tempo macchina per la copia di tutti i dati del processo nel relativo PCB.

Analogamente, un processo può generare più sottoprocessi detti thread; ognuno di essi condivide i segmenti codice e dati con altri thread e con il processo che li ha generati. Per una operazione di Thread Switch si userà, quindi, un descrittore dedicato dal quale devono essere copiate soltanto le informazioni non in comune tra il thread che va in pausa e quello da eseguire. Per questo motivo, un Thread Switch è di norma più veloce rispetto ad un Context Switch.

4.14.1 Alcune particolarità

Come si è già detto ognuno di essi condivide codice e segmento dati con il processo che lo ha generato. Nella pratica questo si traduce nel fatto che ogni thread può accedere in lettura e scrittura a tutte le variabili globali del

programma ed è, ovviamente, possibile passare informazioni e dati al thread, specificando un puntatore ad un blocco di memoria al momento della sua creazione. È importante notare che ogni thread, grazie all'uso di un descrittore diverso da quello usato per i normali processi, viene eseguito in maniera asincrona rispetto al resto del programma. L'uso dell'approccio multithreading può portare vantaggi in situazioni dove un particolare compito deve essere eseguito in parallelo (l'invio dei campioni alla scheda audio durante l'esecuzione del programma è un ottimo esempio) o quando si debba eseguire una gran quantità di calcoli, i cui risultati siano indipendenti l'uno dall'altro (un rendering 3D per punti o il calcolo di un insieme frattale). Oltre alla maggiore chiarezza del codice, in cui le descrizioni dei compiti da eseguire in parallelo sono effettivamente separate dal programma principale, si ottiene anche una maggiore scalabilità: ad esempio, nel caso in cui si esegua il programma su sistemi che hanno più di una CPU, i thread potranno essere realmente eseguiti in parallelo, massimizzando la velocità di esecuzione complessiva. A fronte di questi vantaggi, si ha ovviamente il rovescio della medaglia: il più importante svantaggio è che la programmazione di tipo multithreading richiede molta più attenzione, da parte di chi scrive il codice, rispetto ad un approccio di tipo sequenziale. Vedremo tra poco quali sono i problemi in cui è possibile incorrere e quali siano le possibili soluzioni. La funzione che SDL mette a disposizione per creare un thread è

```
SDL_Thread SDL_CreateThread(int (*fn) (void *), void *data);
```

questa restituisce un puntatore ad una struttura `SDL_Thread`, che potrà essere usato per riferirsi al thread. Il primo parametro che si deve specificare è un puntatore alla funzione che il thread comincerà ad eseguire non appena creato, mentre il secondo permette di passare un puntatore alla funzione stessa; è così possibile rendere disponibili dei dati al thread, ad esempio, tramite un blocco di memoria così referenziato. Nel caso non si voglia usare questo approccio è

⁴² PCB: Process Control Block

sufficiente specificare NULL. Se non viene eliminato, il thread resterà in vita finché la funzione che sta eseguendo non terminerà la sua esecuzione, ad esempio a seguito di un *return <valore>*: in questo caso è possibile usare

```
SDL_WaitThread (SDL_Thread *thread, int *status);
```

per conoscere quale valore ha restituito la funzione eseguita dal thread. Il parametro thread identifica ovviamente il thread in esame, mentre status, se diverso da NULL, è l'indirizzo di memoria dove sarà posto il valore di ritorno. Se *SDL_WaitThread()* viene chiamata quando il thread specificato non ha ancora terminato la sua esecuzione, il processo chiamante si interromperà finché non sarà disponibile il valore di ritorno desiderato (wait bloccante). L'analisi di questa funzione mostra una prima insidia della programmazione multithreading. Nel caso in cui più thread vogliano conoscere il valore di ritorno di altri thread si crea una potenziale situazione di deadlock, uno scenario in cui ognuno dei due thread sta aspettando che l'altro termini la sua esecuzione; ovviamente il risultato è che entrambi rimarranno bloccati. Questo tipo di problema è solitamente difficile da individuare, visto che un codice corretto sia a livello di sintassi che di semantica può, comunque, essere affetto da deadlock nel caso faccia uso di thread. Anche se sono note in letteratura metodologie per controllare le risorse condivise ed evitare questo fenomeno le librerie SDL non forniscono questa funzionalità, incoraggiando il programmatore ad evitare situazioni di questo tipo con una buona progettazione del codice, fornendo in questo modo anche una soluzione più performante.

Appena creato, un thread comincia ad eseguire la funzione che gli è stata assegnata, quindi è necessario lavorare su quest'ultima per ottenere i risultati che interessano. Si supponga di voler essere in grado di

- mettere in pausa il thread;
- abilitarne il normale funzionamento;
- terminarne l'esecuzione;

e che ognuna di queste condizioni operative sia identificata da un codice di stato (cioè *pause_state*, *running_state* ed *exit_state*), contenuto in una variabile *state* globale o in una locazione di memoria della quale è stato passato l'indirizzo al thread. Il seguente codice rende possibile il controllo del thread con le modalità imposte:

```
while(state != exit_state) {
    while(state == pause_state)
        /*...attende ...*/
    while(state == running_state)
        /*...loop dei thread...*/
}
return ret_code;
```

Per terminare forzatamente l'esecuzione di un thread è disponibile la funzione

```
void SDL_KillThread(SDL_Thread *thread);
```

anche se il suo utilizzo è sconsigliato a vantaggio di meccanismi che utilizzano tecniche analoghe a quella appena discussa. Abbiamo utilizzato la possibilità da parte dei thread di accedere in lettura a variabili globali o a indirizzi di memoria specificati dall'esterno. Come si è già detto, l'accesso è possibile anche in scrittura; questo rende concreto il rischio che si verifichi una situazione di accessi concorrenti: tipico è il caso di due thread, uno dei quali deve aggiornare una struttura in memoria mentre l'altro accede in lettura a uno o più membri della stessa. dato che gli accessi sono asincroni può succedere che i dati vengano letti in parte prima dell'aggiornamento dei membri relativi e in parte dopo. Questo può ovviamente portare a comportamenti indesiderati del programma. Visto che può esistere la necessità di avere dati in comune tra più thread, il problema viene aggirato usando vari tipi di controllori di accesso; Si

consideri il caso dei mutex⁴³. Come il nome suggerisce, essi sono degli interruttori, che vengono usati per fare in modo che una operazione venga compiuta da un solo thread alla volta. Questo sincronizza gli accessi ai dati ed elimina il pericolo degli accessi concorrenti. Per creare un mutex si usa la funzione

```
SDL_mutex *SDL_CreateMutex(void);
```

il valore di ritorno è un puntatore a struttura *SDL_mutex*: questa funzione crea il *mutex* specificato *SDL_MUTEX* deve essere chiamata prima di ogni operazione che possa causare un accesso concorrente: infatti, nel caso il mutex sia già stato bloccato, *SDL_mutex()* non termina finché sullo stesso non viene usata la funzione che sblocca il mutex specificato. Al termine dell'esecuzione del programma i mutex creati devono essere cancellati con

```
int SDL_DestroyMutex(SDL_mutex *mutex);
```

Per chi ne preferisse l'uso, per motivi puramente estetici, sono definite le due macro *SDL_LockMutex* e *SDL_UnlockMutex*:

```
#define SDL_LockMutex(m)    SDL_mutexP(m)
#define SDL_UnlockMutex(m)  SDL_mutexV(m)
```

comunque l'uso incauto di mutex può portare a situazioni di deadlock. Lo scenario è del tutto analogo a quello descritto precedentemente: ognuno di due thread blocca un mutex ed attende che quello bloccato dall'altro si liberi.

⁴³ Mutex: acronimo di MUTual EXclusion

4.15 Timer

Il Timer non è altro che una funzione che viene chiamata dopo un certo periodo di tempo trascorso. Per crearlo si deve usare *SDL_AddTimer*.

```
SDL_TimerID SDL_AddTimer(Uint32 interval, SDL_NewTimerCallback  
callback, void *param);
```

Questa funzione prende tre parametri. Il primo è *interval* che è il numero di millisecondi che devono passare tra le chiamate alla funzione timer. Il secondo è *callback*, che è un puntatore alla funzione timer, e necessita di essere chiamata ad ogni intervallo. L'ultimo è *param* che è un puntatore ad un qualsiasi dato che il timer chiama ogni volta. Questa funzione ritorna un *SDL_TimerID* che il programmatore utilizzerà per rimuovere, in seguito, il timer.

La funzione di callback è così strutturata:

```
Uint32 TimerCallback (Uint32 interval, void *param);
```

La funzione può essere rinominata a discrezione del programmatore; I parametric sono *interval*, che contiene l'intervallo correntemente in uso dal timer, e *param* che è lo stessopuntatore che è passato a *SDL_AddTimer*. La funzione ritorna un Uint32. questo valore sarà il nuovo intervallo.

Per rimuovere il timer occorre chiamare *SDL_RemoveTimer*:

```
SDL_bool SDL_RemoveTimer(SDL_TimerID id);
```

Prende in input il solo parametro ID del timer che si vuole rimuovere. Lo stesso ID che si a con la chiamata a *SDL_AddTimer*. Il valore di ritorno è un *SDL_bool*, e *SDL_TRUE* o *SDL_FALSE* indica se la funzione ha avuto successo o meno.

Le funzioni base di sola disposizione del programmatore sono innumerevoli, come altrettante sono le librerie nate per estendere le sue

funzionalità con compiti specifici come *SDL_net*, *SDL_image*, *SDL_mixer*, alcune di queste, verranno trattate nel prossimo capitolo in riferimento al progetto di tesi.

Capitolo Quinto

Implementazione dell'interfaccia grafica

In questo capitolo verrà spiegata l'implementazione dell'elaborato, in particolare le funzioni necessarie alla realizzazione dell'interfaccia grafica, che è stata sviluppata in C con l'ausilio delle librerie SDL e compilata con Microsoft Visual Studio .NET 2003. Verranno date la definizione fondamentali per la comprensione del progetto.

Verranno presentate alcune immagini del progetto che compongono l'interfaccia grafica, ognuna delle quali arricchita di spiegazione tecnica corrispondente.

5.1 Setup di Microsoft Visual Studio .Net per SDL

Per poter installare correttamente la libreria SDL è necessario scaricare dal sito ufficiale i corretti file in base al tipo di sistema operativo usato dal programmatore. Una volta aperto Microsoft Visual Studio .Net 2003 versione in inglese e estratto la libreria scaricata in un'apposita cartella è necessario impostare le corrette dipendenze alla libreria SDL appena estratta. Per questo motivo bisogna cliccare su *Tools*, e poi su *Options*. Fatto ciò, selezionare dal menù laterale di sinistra la cartella *Projects*, e dal menù a tendina di destra *Include Files*. Infine aggiungere il percorso alla cartella estratta nel passaggio precedente contenente i file *include* (vedi figura 5.1).

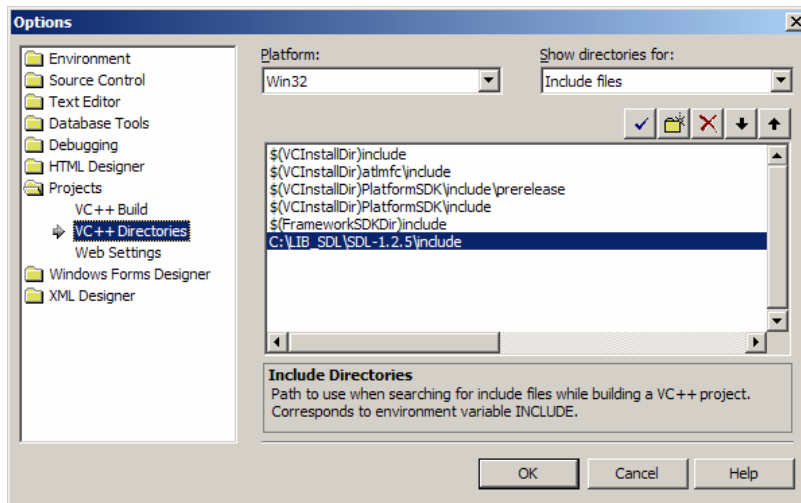


Figura 5.1: Settaggio della cartella include

È inoltre necessario ripetere l'operazione precedente aggiungendo, in questo caso, la cartella relativa alla librerie SDL. Scegliere quindi *Library Files* dal menù di destra a tendina e aggiungere il percorso assoluto delle librerie e degli headers. Una volta fatto questo non rimane altro che creare il proprio progetto, scegliendo dal *menù file* l'opzione *New project* e selezionando *Win32 Project* dalla cartella *Visual C++ Project* del menù di sinistra. Infine in *Application Settings* selezionare *Console Application* e *Empty project* e cliccare sul pulsante *Finish*. A questo punto il progetto è stato creato e non rimane che settare le

ultime proprietà. Nel menù di esplorazione che si è andato a creare cliccare con il tasto destro sul nome del progetto e scegliere *Properties*, e dal menù laterale di sinistra scegliere *Code Generation*, mentre dal menù di destra selezionare *Runtime Library* e infine *Multi-Threaded DLL* (figura 5.2). Ora non rimane altro che creare i file .c aggiungendoli al progetto.

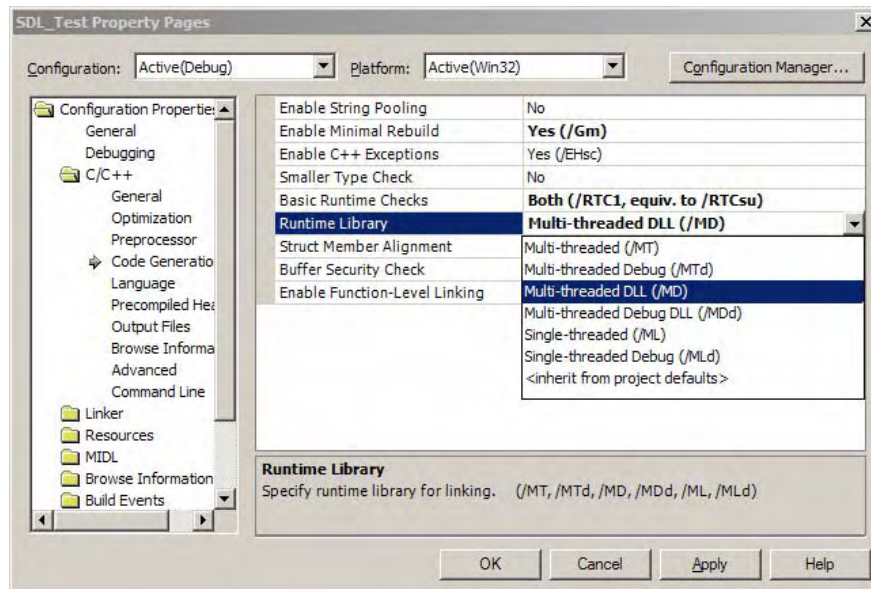


Figura 5.2: Schermata delle opzioni del progetto

Affinché il progetto venga compilato correttamente è necessario includere il file *SDL.dll* della cartella precedentemente settata dentro alla cartella *Debug* e *Release*.

5.2 Segmentazione, Pre-detection, Detection.

Prima di procedere oltre è basilare introdurre i concetti per la comprensione della funzione dell'intero progetto. L'elaborazione si basa su tre fasi principali:

1. Segmentazione;
2. Pre-detection;
3. Detection;

5.2.1 La segmentazione

Il CAD [CAD02] prende in input il mammogramma digitalizzato (che chiameremo immagine originale) ed una immagine relativa al Ground Truth [ZAM03], che è l'indicazione spaziale della presenza di masse. Come si può osservare dagli esempi di mammografie presentati vedi figura 5.3 non tutta l'immagine è occupata dalla mammella, ma vi sono parti, che al il nostro scopo sono assolutamente inutili e che devono quindi essere eliminate. Le motivazioni sono le seguenti:

- La parte complementare alla mammella comprendere informazioni che in un qualche modo potrebbero confondere il classificatore: presentando oggetti, come ad esempio l'etichetta, che si appartengono alla classe dei negativi, ma molto distanti dall'iperpiano di separazione, si condiziona la buona definizione della classe stessa.
- Le immagini mammografiche sono scannerizzate ad altissima risoluzione, si parla di dimensioni che vanno da 4000x2000 a 5000x3000 pixel, per cui ogni minima elaborazione è computazionalmente non trascurabile. Soprattutto vista l'importanza che la velocità di diagnosi riveste come parametro di valutazione del sistema. Quindi, eseguire operazioni su parti non interessanti è un dispendio di tempo assolutamente inutile.

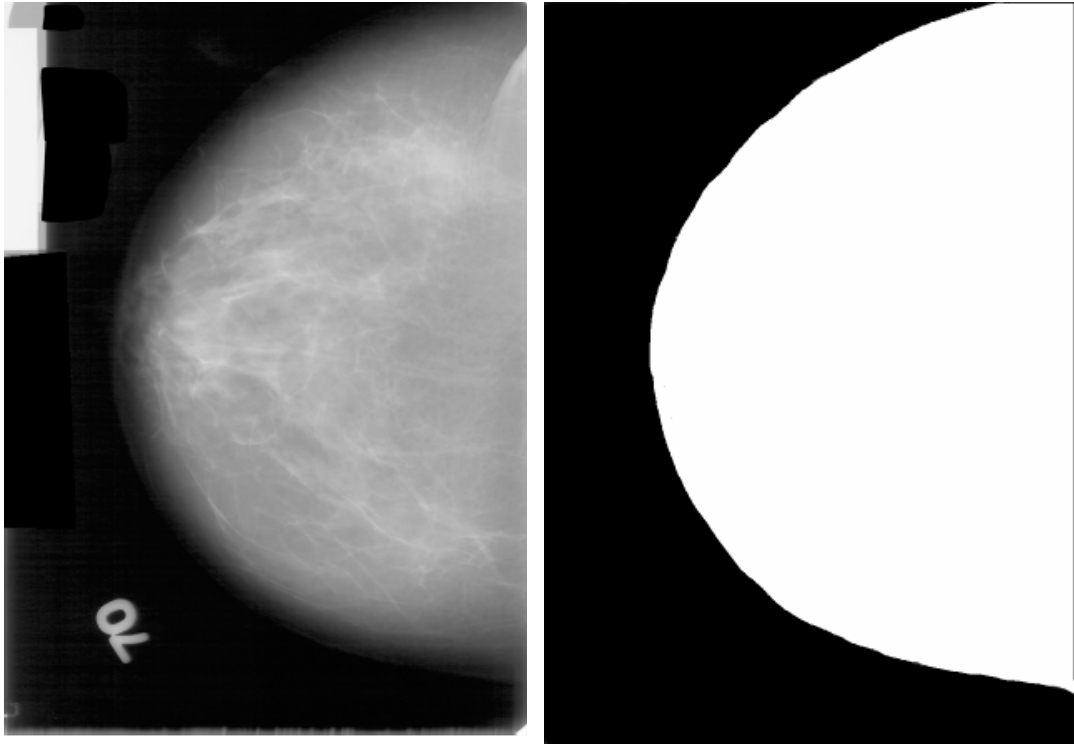


Figura 5.3: A sinistra l'immagine originale a destra l'immagine segmentata

La segmentazione si occupa di tagliare queste zone superflue, generando un file contenente le coordinate della zona nella quale risiede la mammella e sulle quali verranno poi eseguite le elaborazioni successive.

5.2.2 La Pre-detection

Pur essendo di natura molto varia, il tessuto mammario si può dividere in due tipologie fondamentali:

- zone grasse, che sono radio-lucenti;
- zone ghiandolari o fibrose, come i vasi sanguigni, radio-opache.

Queste ultime, in particolare, si contraddistinguono in quanto a grado di luminosità sulla lastra mammografica che risulta decisamente più elevato. Un mammogramma, quindi, da un punto di vista morfologico, si presenta con un fondo abbastanza uniforme su toni scuri e con una parte ben strutturata, più in evidenza, su toni decisamente più elevati. Grazie alla composizione tessutale

generalmente densa del carcinoma mammario, la loro risposta al fascio di raggi X è simile a quella dei corpi radio-opachi, con risultante alta luminosità della zona sul mammogramma. Da quanto detto si deduce che le difficoltà di localizzazione delle lesioni tumorali non sussistono nei casi in cui il tessuto ospitante che le circonda è di composizione grassa. I veri problemi sorgono nel momento in cui, invece, nascono in un tessuto estremamente denso. È in queste situazioni, infatti, che per caratteristiche simili le lesioni possono essere confuse per tessuto sano o viceversa.

Nel precedente capitolo si è descritta la fase di segmentazione, nella quale dal mammogramma vengono asportate le zone esterne alla mammella, le quali non sono di interesse al nostro scopo, in quanto non possono essere lì localizzate masse tumorali.

Basandosi sull'analisi della distribuzione del colore appena fatta, unita al concetto stesso di segmentazione, ci si pone la domanda se esistano zone anche all'interno della parte rappresentante la mammella, nelle quali vi è una bassissima possibilità di trovare lesioni.

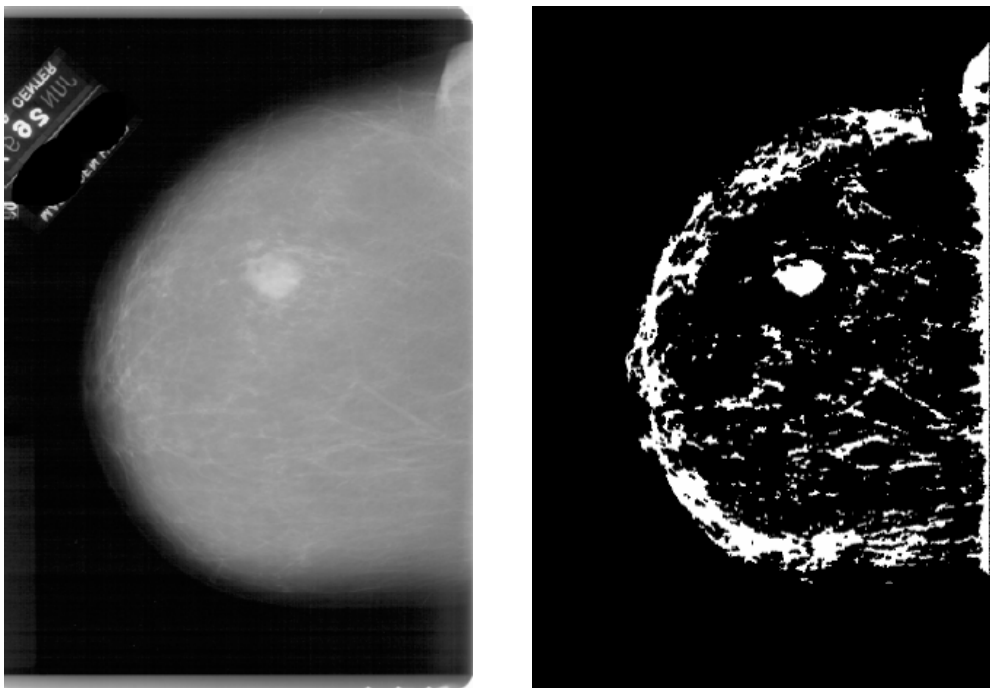


Figura 5.4: A sinistra l'immagine originale, a destra la pre-detection

Dalla caratteristica del carcinoma di essere collocato su frequenze spettrali molto alte, nasce l'idea che sta alla base del modulo di pre-detection: segmentare la parte interna alla mammella, eliminando quelle zone, di tessuto, aventi frequenze spettrali basse, nelle quali vi è una alta probabilità di non trovare masse.

L'output del modulo di pre-detection è dato da una immagine binaria con indicate le Regioni di Interesse (ROI), nelle quali è possibile l'esistenza di una massa (vedi figura 5.4). Grazie a questa, la successiva fase di scanning delle maschere non verrà più eseguita su tutta l'area del seno, ma solo su queste zone. I benefici portati dall'utilizzo di tale tecnica sono principalmente due:

- diminuzione del tempo di calcolo dovuto alla minore area di applicazione dei processi successivi;
- migliore definizione della classe dei negativi

5.2.3 Detection

La fase della detection è molto veloce, in quanto utilizza delle reti neurali, nello specifico si basa su SVM⁴⁴ questa è utilizzata per permettere di rilevare le masse trovate (vedi figura 5.5).

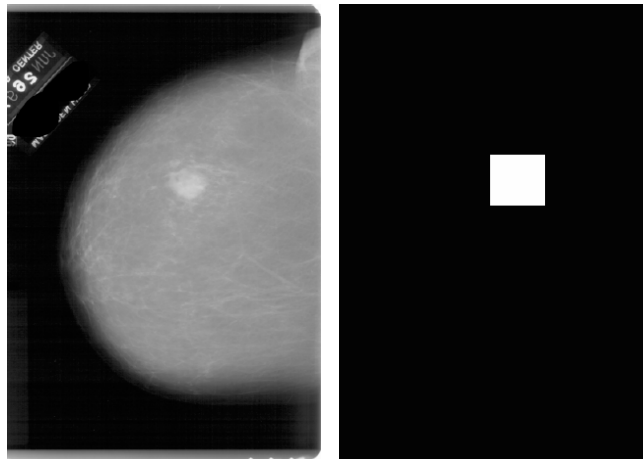


Figura 5.5: A Sinistra immagine originale, a destra maschera ottenuta con la detection

⁴⁴ SVM: Support Vector Machine

5.3 Il progetto

Come DirectX ha le proprie librerie di sotto sistema quali: DirectDraw, Direct3D, DirectSound, DirectPlay, così SDL ha le proprie (vedi figura 5.6), alcune già analizzate nel capitolo precedente.

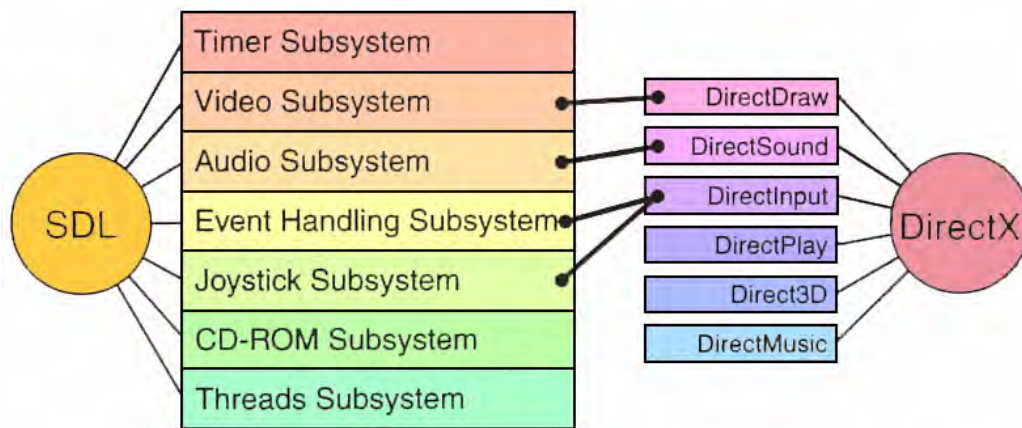


Figura 5.6: SDL e DirectX a confronto

Si è reso però necessario lo sviluppo di un livello intermedio che andasse a interporre fra un linguaggio a basso livello come SDL e un linguaggio ad alto livello per la gestione di quattro sotto sistemi vedi figura 5.7:

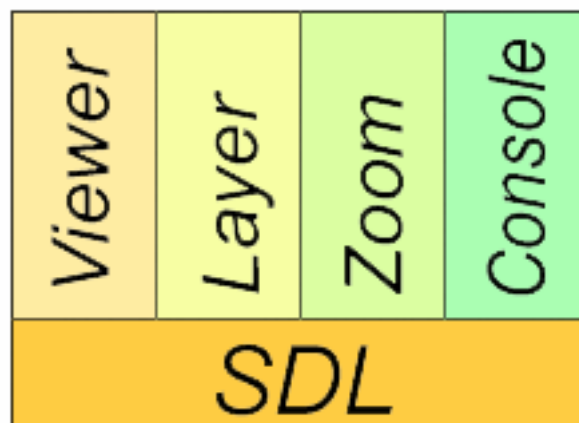


Figura 5.7: I quattro widget implementati

5.4 Il sotto sistema viewer.

Il 70% dei dati è generato dai immagini [SDL04b], così il sotto sistema video SDL è da considerarsi come il più importante e va sviluppato con molta attenzione. Nella prima fase del progetto è nato il problema di come inserire le immagini all'interno della applicazione e di come aggiornarle dopo eventuali modifiche. Proprio per questo si è fatto uso di due particolari funzioni quali:

- `SDL_Rect`;
- `SDL_Surface`.

La maggior parte dei giochi e delle applicazioni funzionano a tutto schermo, o in un'area rettangolare ben definita, così ogni API o libreria usata per creare disegni grafici in 2D ha una speciale struttura che identifica un'area rettangolare. Questo è ciò che fa la funzione *SDL_Rect*. Registra a partire dalla coordinata in alto a sinistra (in pixel) il punto di partenza e poi genera il rettangolo conoscendo la larghezza e l'altezza, mentre *SDL_Surface* memorizza un blocco rettangolare di pixel e lo mantiene in memoria. Con il termine *area rettangolare* si intende che il rettangolo generato segue l'orientamento orizzontale e verticale dello schermo, e l'area in questione non può essere obliqua vedi figura 5.8).

La struttura `SDL_rect` è così definita:

```
typedef struct{
    Sint16 x,y;
    Uint16 w,h;
}SDL_Rect;
```

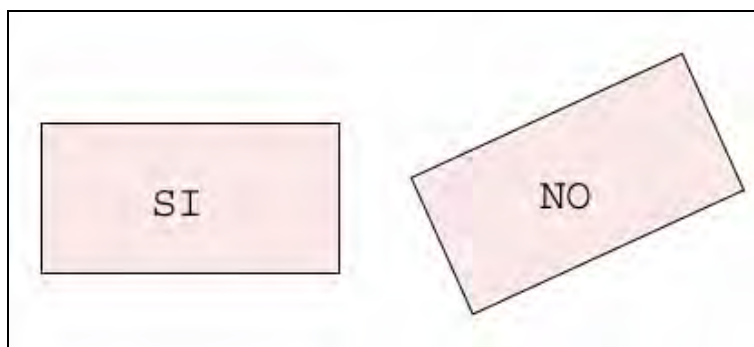


Figura 5.8: a sinistra rettangoli che possono essere rappresentati, a destra rettangoli non permessi.

Poiché il rettangolo è allineato in modo che ogni bordo sia parallelo alla parte alta e sinistra dello schermo (allineamento top-left), sono necessari solo due punti per definirlo: (x,y) e $(x+w,y+h)$ mentre gli altri punti sono $(x+w,y)$ e $(x,y+h)$. I membri x e y di `SDL_Rect` sono *Sint16* (interi con segno a 16 bit) questo significa che hanno un valore compreso tra -32.768 e $+32.768$, più che sufficiente per rappresentare un rettangolo in una schermata video di un monitor. Mentre w e h sono *Uint16* (interi senza segno a 16 bit) quindi hanno un valore che va da 0 a 65.535, valore che è sempre maggiore a quanto è possibile rappresentare attualmente in una schermata video. Bisogna anche ricordare che il sistema di riferimento delle coordinate SDL parte dall'angolo in alto a sinistra $(0,0)$, il valore di x cresce verso destra (figura 5.9), mentre il valore di y parte dall'angolo in alto a sinistra del rettangolo e aumenta scendendo verso il basso.

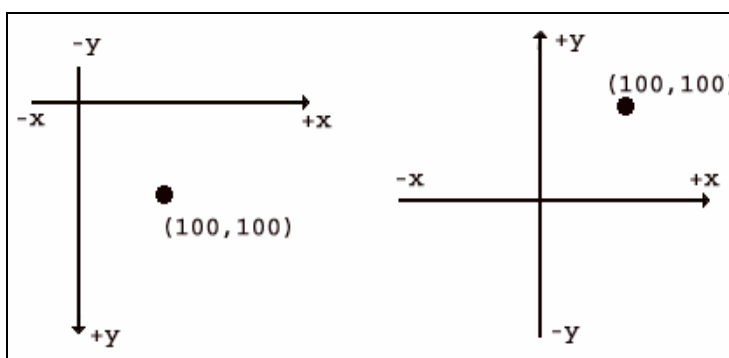


Figura 5.9: Differenza tra coordinate in uno schermo a sinistra e coordinate cartesiane a destra.

Vi sono inoltre due considerazioni importanti da tenere presente. Volendo sapere quali siano i punti all'interno o all'esterno di rettangolo di una data area bisogna considerare i due casi:

- Le linee del rettangolo passano attraverso i centri delle linee e delle colonne;
- Le linee del rettangolo passano tra i centri delle righe e delle colonne (vedi figura 5.8).

Si supponga di avere una struttura che rappresenta un singolo punto 2D come questa:

```
typedef struct{
    Sint16 x,y;
}Point ;
```

Si supponga inoltre di avere un punto *A* e un *SDL_Rect* chiamato *R* come i seguenti :

Point A;

SDL_Rect R;

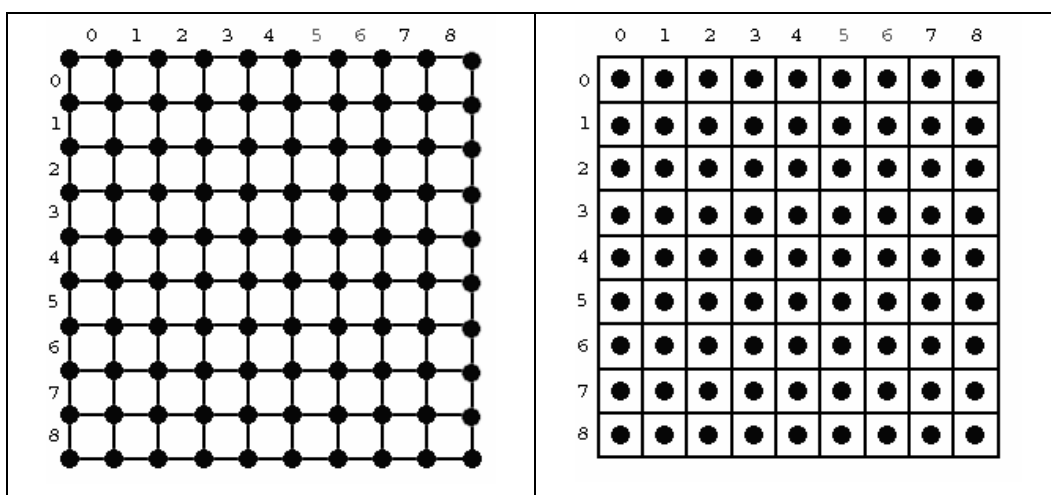


Figura 5.10: A sinistra linee che passano attraverso il centro delle righe e delle colonne, a destra linee passanti tra le righe e le colonne

Entrambi hanno un valore e ad un certo punto si supponga di determinare se A è dentro o fuori a R . Poiché $R.x$ e $R.y$ rappresentano l'angolo in alto a sinistra del rettangolo R , si sa che se $A.x$ è minore di $R.x$ o $A.y$ è minore di $R.y$, allora A deve essere al di fuori di R . Se $A.x$ è più grande o uguale a $R.x$ e $A.y$ è più grande o uguale a $R.y$, allora A si trova in R , ma questo dipende da come vengono considerate le linee dell'area di rettangolo. Per come è strutturata `SDL_Rect`, l'angolo a destra ($R.x + R.w$) e l'angolo in basso ($R.y + R.h$) non sono dentro al rettangolo. Questo è spiegato dal fatto che le posizioni 1,2,3 e così via sono linee passanti attraverso il centro della riga e della colonna (vedi figura 5.8) oppure le linee passano attraverso lo spazio fra un pixel e l'altro. Nel caso si volesse sapere se un punto A sta in R , $A.x$ deve essere più piccolo di $R.x + R.w$ e $A.y$ deve essere minore di $R.y + R.h$. Per sapere se A è dentro a R si deve utilizzare il seguente codice:

```
if(A.x>=R.x && A.y >= R.y && A.x < (R.x+R.w) && A.y < (R.y +
R.h))
{
    //il punto A è dentro a R
}
else{
    //il punto A non è dentro a R
}
```

Un ultima considerazione da fare è che se un rettangolo ha w e h uguali a 0, questo è vuoto.

Ciò che è stato utile nell'applicazione, per la creazione di un'area di rettangolo che individuasse una porzione dell'immagine per poi applicare la funzione di zoom ad essa. Come verrà spiegato nel paragrafo 5.6.

Per visualizzare la schermata dell'applicazione è necessario settare la superficie di visualizzazione e questo avviene con la chiamata alla funzione

```
SDL_Surface *SDL_SetVideoMode(int width,int height,int  
bpp,Uint32 flags)
```

Questa funzione prende tre parametri e ritorna un puntatore a *SDL_Surface*; il valore di ritorno rappresenta la superficie del display. Se il valore di ritorno è uguale a *NULL* la funzione fallisce. Per gli altri parametri non c'è bisogno di alcun commento in quanto sono autoesplicativi. Il parametro *bpp* dice a SDL quanti bit per pixel deve avere la superficie. Il parametro *flag* invece serve per settare alcune opzioni, già viste nel capitolo precedente.

5.5 Il sotto sistema layer

Un altro oggetto fondamentale nel sotto sistema SDL è *SDL_Color*. Definito nel seguente modo:

```
typedef struct {  
    Uint8 r;  
    Uint8 g;  
    Uint8 b ;  
    Uint unused ;  
}SDL_Color ;
```

I membri *r*, *g*, *b* rappresentano rispettivamente l'intensità del rosso, del verde e del blu. Con un'intensità che varia dallo 0, dove c'è mancanza di colore, al 255 che presenta un'intensità massima per il tipo di colore. Il membro *unused* non rappresenta affatto un colore, ma piuttosto immagazzina un'informazione come per esempio l' *alpha*, che come si vedrà in seguito rappresenta l'intensità di trasparenza. Lo spazio del colore RGB è tridimensionale e le dimensioni sono il rosso (R), il verde (G) e il blu (B). Ogni informazione è fissata, in modo che solo i valori tra lo 0.0 e 1.0 sono rappresentati lungo ogni asse. Così 0.0 è rappresentato dal valore 0 e 1.0 è rappresentato dal valore 255, dando così la possibilità di scegliere fra 254 valori intermedi, per un totale di 256 valori. Poiché ci sono tre assi di colore, ci sono $256 \times 256 \times 256$ differenti valori che possono essere rappresentati da *SDL_Color* per un totale di 16.777.216 distinti

colori. Le limitazioni sono date dalle modalità video; infatti non tutti i sistemi video riescono a rappresentare questi colori. Il numero di bits per pixel potrebbe essere 8, 16, 24 o 32. Nel sistema a 24 o 32 bit possono essere rappresentati tutti i colori, mentre in un sistema a 16 bit ognuna delle assi di colore sono troncate tipicamente a 5 bit per il rosso, 5 o 6 bit per il verde e 5 bit per il blu. In questo caso i bit meno significativi sono ignorati. Nel sistema a 8 bit, poiché è possibile rappresentare solo 256 colori, l'immagine risulterà avere poca qualità.

Dato che l'applicazione sviluppata è un modulo che va ad integrare il programma CAD molto più complesso [CAD04], le immagini che questa deve analizzare non sono residenti sull'hard disk, ma vengono passate dalla memoria del programma all'applicazione. Per questo è necessario l'utilizzo della funzione *SDL_CreateRGBSurfaceFrom* così definita:

```
SDL_Surface *SDL_CreateRGBSurfaceFrom(void *pixel, int width,
int height, int depth, int pitch, Uint32 Rmask, Uint32 Gmask,
Uint32 Bmask, Uint32 Amask);
```

Dove Rmask, Gmask e Bmask rappresentano il pixel format che si sta considerando. Il parametro *pixel* è un puntatore al dato del pixel che si usa per la superficie; è responsabilità del programmatore allocare e liberare la memoria da questi utilizzata. E non verrà deallocata quando è chiamata la funzione *SDL_FreeSurface*.

Il *pitch* è il doppio della dimensione di una riga in pixel.

5.5.1 Creare una Color Key (la trasparenza)

La necessità di applicare superfici multiple all'applicazione obbliga l'utilizzo di una funzione che sia in grado di gestire il fattore trasparenza, altrimenti con la sovrapposizione delle immagini si avrebbe la perdita dell'ultima visualizzata. La *SDL_BlitterSurface* è una funzione, molto utilizzata all'interno del programma, in grado di copiare un blocco intero di pixel da una superficie ad un'altra, ma è priva del fattore di controllo della trasparenza, dopo un attento studio delle

funzioni di libreria, si è individuata la funzione *SDL_SetColorKey*, utile a tale scopo (vedi figura 5.11).

Il programma sovrappone tre layer all'immagine originale:

- Un layer per la segmentazione;
- Un layer per predetection;
- Un layer per la detection;

è fondamentale quindi che si riesca a vedere tutti i layer senza problemi di sovrapposizione. La dichiarazione della funzione è la seguente:

```
int SDL_SetColorKey(SDL_Surface *surface, Uint32 flag, Uint32 key);
```

Questa funzione ritorna un valore intero, dove 0 si intende un valore di successo, mentre -1 significa che la funzione non è andata a buon fine. La funzione come si può notare ha tre parametri:

1. Un puntatore alla superficie per il quale viene settato il color key
2. Un insieme di flag;
3. Un valore da usare come colore di trasparenza.

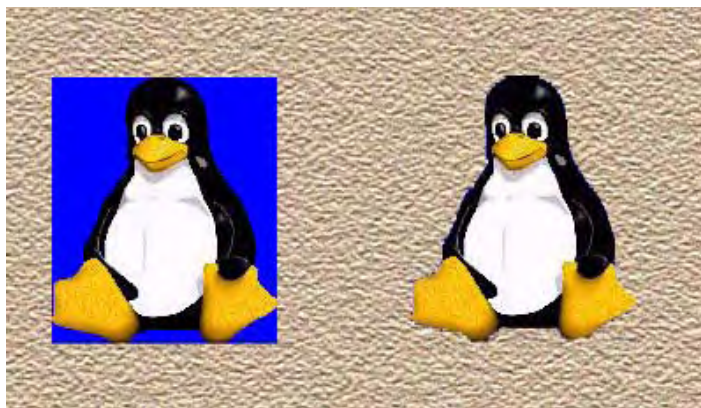


Figura 5.11: A sinistra immagine originale, a destra immagine a cui si è applicata una colorkey RGB (0,0,255)

Ci sono due tipi di parametri che il programmatore può utilizzare come flag per il secondo parametro. Uno di questi è *SDL_SRCCOLORKEY* e l'altro *SDL_RLEACCEL*. Quest'ultimo non può essere utilizzato da solo, quindi in realtà si hanno tre parametri che si possono inserire nel campo flag: 0, *SDL_SRCCOLORKEY* e *SDL_SRCCOLORKEY | SDL_RLEACCEL*. Nel caso venisse passato il valore 0, per qualsiasi *colorkey* inserita, questa viene cancellata. Se si utilizza *SDL_SRCCOLORKEY*, il valore inserito viene riconosciuto come valore di trasparenza.

```
SDL_SetColorKey((*sprite),SDL_SRCCOLORKEY,SDL_MapRGB((*sprite)->format,0,0,0));
```

Il colore deve essere settato con il pixel format, è utile quindi utilizzare *SDL_MapRGB*, come sopra. Se si utilizzasse *SDL_SRCCOLORKEY | SDL_RLEACCEL* si va a settare il *colorkey* e la superficie in modo che utilizzi l'accelerazione *RLE*⁴⁵. In altre parole l'immagine viene codificata così il blit dell'immagine è più veloce.

⁴⁵ RLE Run Length Encoded

5.5.2 Alpha Blending

L'alpha blending non viene gestito dall'hardware, ma è SDL che se ne occupa, emulandolo. Naturalmente qualsiasi cosa emulata risulterà leggermente più lenta, ma di fatto SDL fa in modo che il programmatore non debba implementarlo per conto proprio. L'alpha blending (vedi figura 5.12) è un modo per fare la sovrapposizione delle immagini traslucide.

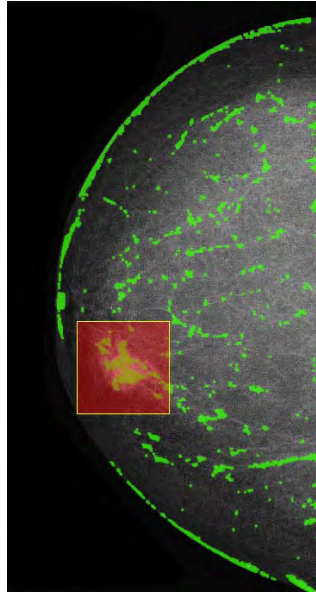


Figura 5.12: Sovrapposizione dei 3 layer

Utile per una varietà di effetti come specchi, immagini di fantasmi, effetti di teletrasporto, dissolvenza delle immagini, etc. Nel progetto è stata utilizzata per dare una dissolvenza, a discrezione dell'utente, ai layer utilizzati in modo tale da non oscurare l'immagine originale con l'aggiunta di nuovi layer. I valori che può assumere l'alpha blending vanno da 0 (0.0), dove l'immagine è completamente trasparente con un livello di percentuale di colore dello 0%, a 255 (1.0) dove l'immagine è completamente opaca e il livello di colore è al 100%.

Il programmatore può specificare il valore dell'alpha blending in due modi:

- specificando un singolo valore di alpha per l'intera superficie, da 0 a 255;
- specificando il valore di ogni pixel della superficie. Da 0.0 a 1.0.

5.5.3 Per-surface alpha

Nel programma per settare il valore di alpha è stata usata la funzione

```
int  SDL_SetAlpha(SDL_Surface  *surface,  Uint32  flag,  Uint8
alpha);
```

La funzione ha lo stesso schema della *SDL_SetColorKey*, il valore di ritorno di questa funzione è un intero, ed è 0 in caso di successo, -1 in caso di insuccesso. Il primo parametro (surface) è la superficie per la quale si va a settare un singolo valore di alpha. Il secondo parametro (flag) è la combinazione di *SDL_SRCALPHA* e *SDL_RLEACCEL*. Se *SDL_SRCALPHA* è presente allora viene settata per quella superficie il valore di alpha. Se non è presente il valore di alpha per la superficie viene cancellato. Se *SDL_RLEACCEL* è messo in OR con *SDL_SRCALPHA*, la superficie viene ottimizzata per funzionare con l'accelerazione *RLE*. Il terzo e ultimo parametro (alpha) è il valore di alpha da utilizzare nella superficie. I valori che può assumere vanno dallo 0 (*SDL_ALPHA_TRANSPARENT*) al 255 (*SDL_ALPHA_OPAQUE*). Il valore intermedio 128 è importante perché è una ottimizzazione degli altri due valori. Per sapere se una superficie ha un valore di alpha o meno bisogna controllare il campo *flag* di *SDL_Surface*. Se è presente *SDL_SRCALPHA* il valore del pixel format di surface sarà il valore di per-surface.

5.5.4 Per-Pixel Alpha

Per un maggior controllo del valore di alpha, il programmatore può creare una superficie che ha un valore di alpha *per-pixel*. Questo avviene attraverso l'utilizzo di due funzioni:

- `SDL_CreateRGBSurface`
- `SDL_CreateRGBSurfaceFrom`

Come già spiegato, è stato necessario l'utilizzo di `SDL_CreateRGBSurfaceFrom`

5.5.5 Le palette

In un mondo dove le schede video hanno una capacità di 32 bit per pixel con una buona velocità di esecuzione, ci si chiede perché bisogna limitarsi a soli 256 colori. Le ragioni sono molteplici. La prima è che, sebbene le schede video migliorino ogni anno, una superficie a 32 bit per pixel occupa memoria quattro volte superiore rispetto a una che ha solo 8 bit per pixel; una superficie a 8 bit inoltre trasferisce i dati più velocemente. La seconda è che ci sono alcuni device come laptops, palmari, ma anche vecchi hardware che funzionano molto meglio con 8 bit per pixel. Se poi l'applicazione in questione è indirizzata ad un particolare settore, è meglio non limitarla. Una terza ragione la si può trovare nel fatto che superfici a 8 bit sono le uniche che ottengono degli effetti particolari. Si pensi infatti all'animazione delle palette, un effetto altrimenti conosciuto come *color cycling*, dove è possibile creare l'illusione del movimento senza sovrapporre altre immagini, ma semplicemente cambiando la palette. Naturalmente ci sono delle limitazioni e la più grande è che si hanno a disposizione solo 256 colori con i quali lavorare. Questo non solo limita il numero di colori sullo schermo, ma rende difficile creare oggetti artistici. In ogni caso la palette in SDL è piuttosto semplice ed è rappresentata dalla struttura seguente:

```
typedef struct{
    int ncolors;
    SDL_color *colors;
} SDL_palette;
```

I due membri sono *ncolors*, che contiene un intero che specifica quanti colori ci sono nella paletta, (tipicamente 256, ma si possono costruire palette più piccole) e *colors* che è un puntatore ad un array di *SDL_Color*. Queste variabili contengono tutti i colori nella paletta. Per creare una superficie che abbia una paletta è necessario specificare 8 bit per pixel durante la creazione di quella superficie. Se si sta lavorando in modalità a tutto schermo, bisogna inoltre utilizzarle il flag *SDL_HWPALETTE*, in quanto garantisce un maggior controllo dei colori della paletta. Questo perché se il desktop è nella modalità a 8 bit, il sistema operativo tipicamente riserva una parte di colori così che possa visualizzare correttamente la videata. Sotto windows questa parte è di 20 colori, i primi e gli ultimi dieci della palette, questo lascia 236 colori che si possono settare in una palette fisica. Per il settaggio della palette SDL ha due funzioni. La prima si chiama *SDL_SetPalette* ed è dichiarata nel seguente modo:

```
int SDL_SetPalette(SDL_Surface *surface, int flags, SDL_Color
*color, int firstcolor, int ncolors);
```

Questa funzione ritorna un intero. Se SDL setta tutti i colori che ha specificato nella chiamata alla funzione, ritornerà valore 1, altrimenti 0. Nel caso ritornasse 0 non significa necessariamente che è un errore, ma piuttosto che SDL non è in grado di settare tutti i colori, e ne ha settati il maggior numero possibile. Nel caso il programmatore specificasse *SDL_HWPALETTE*, questa funzione ritornerà sempre il valore 1, tranne il caso in cui si cercasse di chiamare questa funzione su una superficie che non sia a 8 bit, naturalmente ritornerà un valore 0. Il primo parametro (*surface*) è un puntatore alla superficie per la quale si sta settando la paletta di colori. Il secondo parametro (*flags*) è uno, o entrambi di *SDL_LOGPAL* (logical palette) oppure *SDL_PHYSPAL* (physical palette). Questi due flag possono essere combinati. Il terzo parametro (*color*) è un puntatore ad un array di *SDL_Color*. Il quarto parametro (*firstcolor*) è il primo colore che si vuole settare nella paletta, mentre l'ultimo parametro indica quanti colori si vogliono settare.

La seconda funzione che può essere utilizzata per settare i colori nella paletta è `SDL_SetColors` dichiarata nel seguente modo:

```
int SDL_SetColors(SDL_Surface *surface, SDL_Color *colors, int
firstcolor, int ncolors);
```

Questa funzione è molto simile a `SDL_SetPalette`, con la mancanza del parametro flag ed è utilizzata per settare una parte della mappa dei colori.

5.5.6 L'aggiornamento del monitor

SDL ha due modi per aggiornare il monitor ogni volta che vengono applicati dei cambiamenti alla superficie:

- Aggiornando un' area di rettangolo dello schermo
- Utilizzando il doppio buffer.

Nell'aggiornare un'area di rettangolo si può specificare una o più aree che si devono aggiornare. Questo avviene utilizzando una delle due funzioni:

- *SDL_UpdateRect*
- *SDL_UpdateRects*

Così dichiarate:

```
void SDL_UpdateRect(SDL_Surface *screen, Sint32 x, Sint32 y, Sint32
w, Sint32 h);
void SDL_UpdateRects(SDL_Surface *screen, int numrects, SDL_Rect
*rects);
```

Nella prima ci sono cinque parametri. Il primo è un puntatore alla superficie che si vuole aggiornare (screen), seguita da x,y,w e h che determinano l'area di rettangolo che si vuole aggiornare. Se questi quattro valori sono messi a 0 viene aggiornata l'intera superficie. In *SDL_UpdateRects* invece ci sono solo tre

parametri. Il primo (screen) è un puntatore alla superficie, il secondo (numrects) definisce quanti rettangoli sono presenti nell'array puntato dal terzo parametro (rects). Queste due funzioni sono molto valide e devono essere utilizzate nel caso si voglia cambiare solo una certa parte dello schermo dall'ultimo aggiornamento.

Nell'utilizzo della funzione *SDL_Flip* si intende che si sta usando la modalità tutto schermo con doppio buffer. Comunque nel caso non ci fosse la disponibilità del doppio buffer la chiamata alla funzione *SDL_Flip* è equivalente alla chiamata alla funzione *SDL_UpdateRect* (screen,0,0,0,0). La funzione *SDL_Flip* è così dichiarata:

```
int SDL_Flip(SDL_Surface *screen);
```

Diversamente dalle due funzioni viste precedentemente, questa funzione ritorna un valore che può essere 0 in caso di successo, -1 altrimenti. L'unico parametro è un puntatore alla superficie.

5.6 Lo zoom

L'applicazione sviluppata visualizza immagini che hanno una dimensione molto elevata, per questo motivo è necessario ridimensionarle. Però esiste anche la necessità da parte dell'utente di evidenziare una particolare area dell'immagine, allo scopo di rendere visibile e più chiaramente particolari che altrimenti rimarrebbero nascosti perché troppo piccoli per l'occhio umano (vedi figura 5.13).

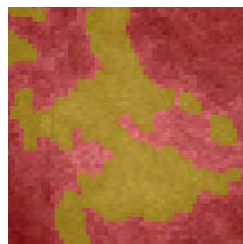


Figura 5.13: zoom del riquadro giallo dell'immagine 5.12

A tal fine è stata creata una funzione che consente all'utente di selezionare una parte dell'immagine con l'utilizzo del mouse e, una volta selezionata, l'immagine che compare alla destra della schermata, può essere spostata, con l'utilizzo dei tasti direzionali o dello stesso mouse e aumentata (e anche rimpicciolita) di un fattore di zoom a discrezione dell'utente. Per la creazione del rettangolo di selezione dell'immagine si è tenuto presente di ciò che è stato detto nel paragrafo 5.3, riferendosi al primo caso in cui le linee del rettangolo passano al centro delle righe e delle colonne dello schermo. In particolar modo si sono create due Macro:

```
#define MAX(a,b)((a<b)?b:a)
#define MIN(a,b)((a>b)?b:a)
```

e si è così proceduto per determinare i quattro punti di coordinate:

```
x=MIN(a,c);
w=MAX(c-a,a-c);
y=MIN(b,d);
h=MAX(d-b,b-d);
```

Invece la funzione di zoom è così definita:

```
SDL_Surface * zoomSurface (SDL_Surface *src, double zoomx,
double zoomy, int smooth);
```

prende in ingresso (src) una superficie a 32bit o a 8 bit e ne restituisce un'altra (dst). Zoomx e zoomy sono i fattori di scala rispettivamente per la larghezza e l'altezza. Il fattore di smooth funziona solo su superfici a 32 bit, se il valore è settato a 1 lo smooth è attivo, a 0 è disattivato. Se smooth è a 1 allora la superficie di destinazione a 32 bit è anti-aliased⁴⁶. Se la superficie non è a 8 bit

⁴⁶ Anti-aliased: Tecnica per limare la scalettatura nelle immagini grafiche.

oppure è a 32 bit RGBA/ABGR sarà convertita in una formato RGBA a 32 bit al volo.

5.7 La console

SDL_Console è una console che può essere aggiunta a qualsiasi applicazione SDL. È simile alla console del gioco Quake [QAK04]. È stata aggiunta all'applicazione per garantire interattività al programma da parte dell'utente (vedi figura 5.14)

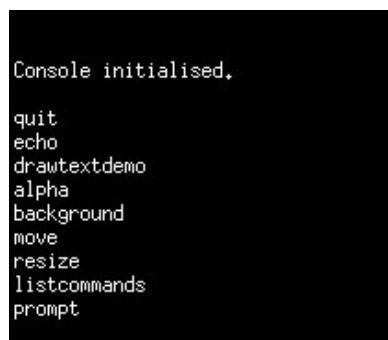


Figura 5.14: Console SDL

All'esecuzione del programma compare sulla destra una prima console, che può essere nascosta o mostrata con la combinazione dei tasti Ctrl+1 da tastiera e volendo l'utente è in grado di far uso di una seconda console con la combinazione Ctrl+2. Per entrare nella modalità di input a seconda della console che desideri utilizzare questi preme Alt+1 oppure Alt+2. Una volta entrato nella modalità di input l'utente è in grado di scrivere sulla console con il comando *echo*, aumentare o diminuire il valore dell'alpha blending, aggiungere un background all'applicazione, spostare o ridimensionare le immagine, etc, o semplicemente uscire dal programma.

A seguire alcuni screenshot dell'applicazione eseguita sulle piattaforme Windows Xp, Linux, Mac OS X.

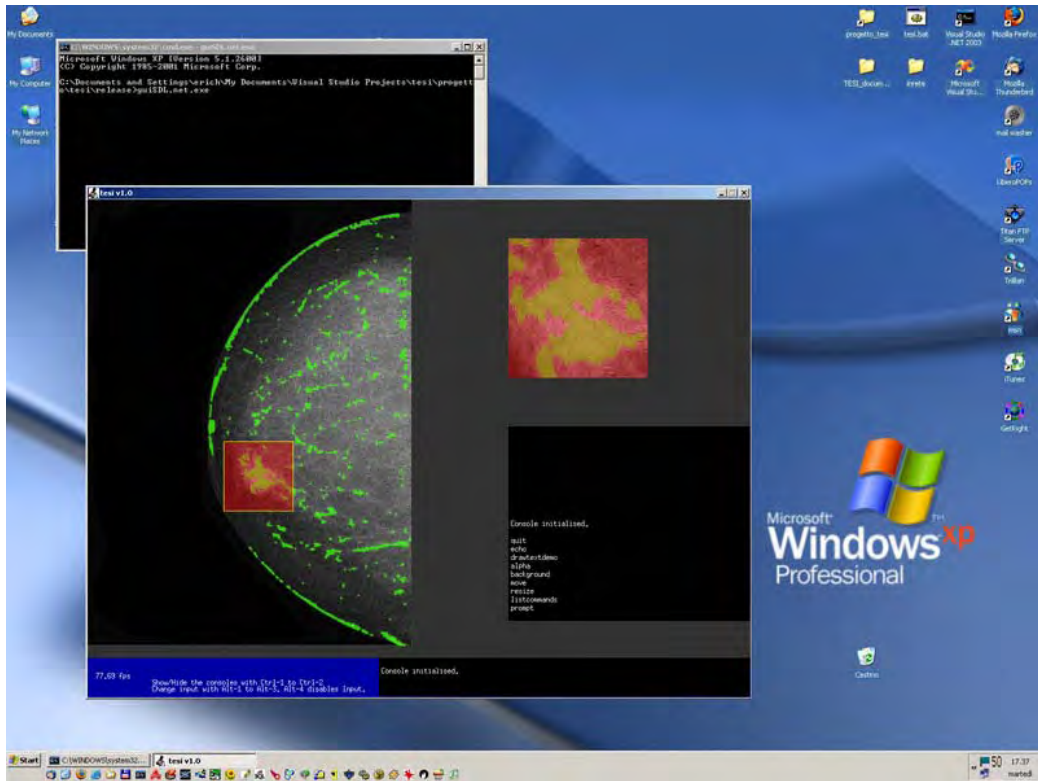


Figura 5.15: Applicazione in esecuzione su Windows Xp

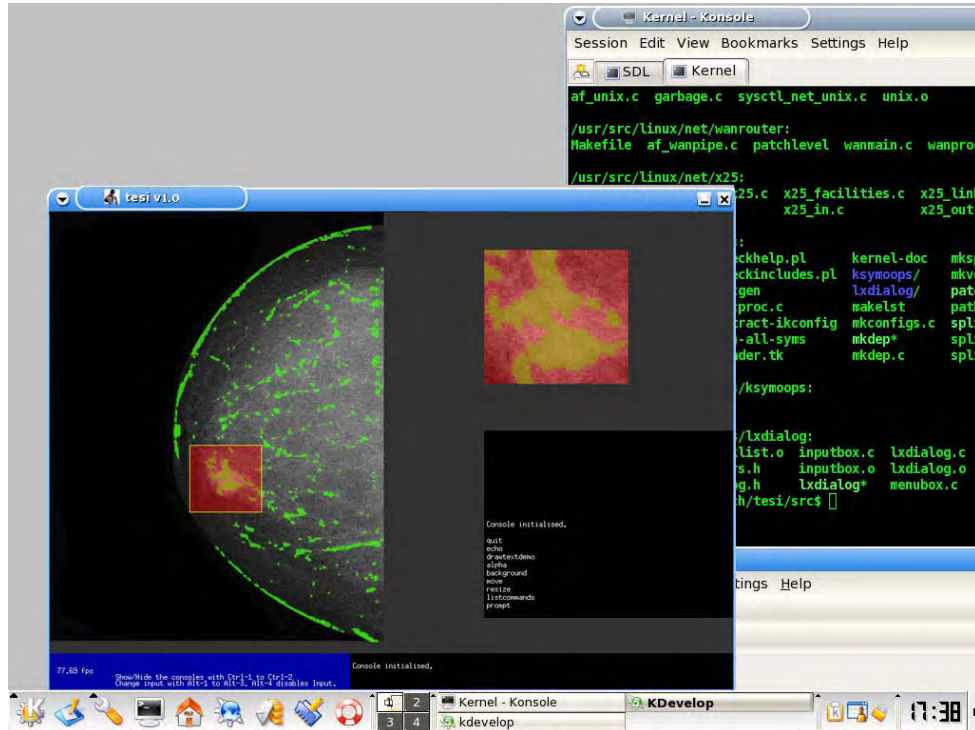


Figura 5.16: Applicazione in esecuzione su Linux Slackware 9.1

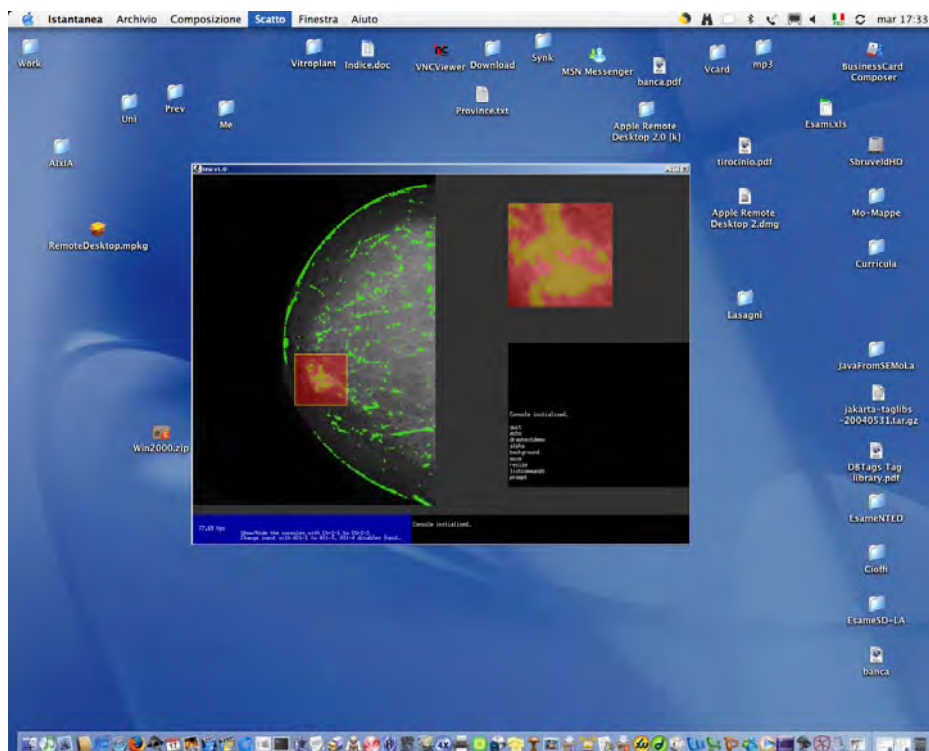


Figura 5.17: Applicazione in esecuzione su Mac OS X 10

Capitolo Sesto

Sviluppi futuri

L'utilizzo di widget grafici multi-piattaforma 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 gprs e successivamente avviare il viewer con la visualizzazione finale dell'analisi.

Una naturale evoluzione dell'applicazione grafica consiste nella gestione di un contesto OpenGL, resa possibile dall'integrazione con la libreria SDL, questo da una parte consentirebbe di espandere le potenzialità del viewer in ambienti grafici 3D e di sfruttare pienamente l'accelerazione HW disponibile sulle schede video native OpenGL

Questo permetterà di estendere i campi di utilizzo dell'applicativo alla TAC⁴⁷ e la ricostruzione tridimensionale di organi ed oggetti in modelli 3D.

⁴⁷ Tomografica Assiale Computerizzata

Capitolo Settimo

Conclusioni

In questo lavoro di tesi si è preso in considerazione le soluzioni più efficaci e computazionalmente efficienti per realizzare un framework grafico multi-piattaforma. Si sono imparati i concetti fondamentali che stanno dietro alla progettazione e lo sviluppo di un'interfaccia utente, si sono viste quali sono le principali regole che consentono di implementare un'interfaccia grafica affidabile e si è visto quanta importanza abbia, nella realizzazione di un progetto, partire dagli schemi e dai modelli utilizzati dalle applicazioni più comuni e diffuse.

Dopo un'analisi approfondita ed esaustive prove sul campo si è scelto di utilizzare una libreria di basso livello altamente performante e multi-piattaforma che sta diventando lo standard nello sviluppo di applicazioni grafiche sotto il sistema operativo Linux. SDL (Simple DirectMedia Layer) si è presentata altamente performante, configurabile e semplice nell'utilizzo, Si sono tuttavia sviluppati widget di medio livello per permettere un'agevole realizzazione di applicativi di più alto livello: in particolare quattro tool con cui è possibile creare agevolmente soluzioni grafiche efficaci e complesse.

Nel dettaglio il viewer rappresenta il componente principale, che permette di gestire, in modo adeguato e veloce, il flusso di immagini che le moderne applicazioni richiedono.

Tramite la gestione dei layer, prerogativa degli applicativi commerciali di alto livello, è possibile utilizzare in modo visuale informazioni complesse che in questo modo diventano di facile fruizione da parte dell'utente. Questo risulta di particolare importanza negli applicativi medici e di interesse soprattutto in fase di sviluppo del software in cui è necessario controllare velocemente lo stato di avanzamento nell'elaborazione delle informazioni.

Tramite la console realizzata è possibile inoltre indirizzare sull'ambiente grafico informazioni testuali.

L'ultimo tool implementato consiste nello zoom di immagini che utilizza l'interpolazione lineare e bilineare implementata a basso livello, sfruttando appieno le caratteristiche dell'hardware sottostante.

Nonostante lo sviluppo di questi widget non rappresentino di per se un'innovazione nel settore, in realtà l'originalità ed innovatività di essi deve ricercarsi nella portabilità di questi tool studiati per essere multi-piattaforma, risultando in questo modo virtualmente accessibili e disponibili a tutti gli sviluppatori esistenti a livello mondiale.

Bibliografia

- [38604] Intel Processors,
<http://www.intel.com/design/intarch/intel386/>
- [3DF04] 3dfx Interactive Inc,
<http://www.3dfx.com/>
- [AIX04] Unix operating system,
<http://www-1.ibm.com/servers/aix/>
- [ALP02] DEC Alpha EV4 21064 150MHz,
<http://cpu-museum.de/?q=alpha>
- [APC99] The apache software foundation,
<http://www.apache.org/>
- [APP04] Apple Computer, “Apple”,
<http://www.apple.com/>, 2004
- [ARM04], Arm processor,
<http://www.altera.com/products/ip/processors/arm/ipm-index.jsp>
- [ASS00] Associated Universities Inc, “AIPS++ Glossary”,
<http://www.astron.nl/aips++/docs/glossary/glossary.html>, 2000
- [BOR04] Borland, “Delphi”, <http://www.borland.com/delphi/>, 2004
- [BSD04] Advanced operating system for x86 compatible,
<http://www.freebsd.org/>

- [CAD02] 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".
In Proc. of LXXXVIII National Congress of the Italian Physics Society, 2002.
- [CAD04] R. Campanini, D. Dongiovanni, E. Iampieri, N. Lanconelli, M. Masotti, G. Palermo, A. Riccardi, 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, 961-975, 2004.
- [DIA04] Diagram creation program,
<http://www.gnome.org/projects/dia/>
- [DRA86] S. Draper, D. A. Norman,
"User Centered System Design: New Perspectives on Human-Computer Interaction", Paperback, 1986
- [ERC04] Ercolani G, Tesi di Laurea, "Progettazione di un sistema p2p: interfaccia grafica", 2004
- [FLA04] Macromedia Flash MX 2004,
<http://www.macromedia.com/>
- [GIM04] The Gimp,
<http://www.gimp.org/>

- [HTM04] HTML.it, “Definizione di interfaccia”,
http://www.html.it/web_design/web_design_06.htm, 2004
- [INN04] Science and Engineering Visualization challenge,
http://www.sciencemag.org/feature/data/vis2003/illus_first.html
- [INT04] Intel Corporation,
<http://www.intel.com/>
- [JFC04] Java Foundation Classes,
<http://java.sun.com/products/jfc/>
- [LIB03] LibraryHQ, “Library Automation/Technology Glossary”,
<http://www.libraryhq.com/glossary.html>, 2003
- [LIN04] What is linux?,
<http://www.linux.org/>
- [MCT04] Designing a Graphical User Interface,
<http://www.medicalcomputingtoday.com/0agui.html#principles>
- [MFC04] Microsoft Foundation Classes,
<http://msdn.microsoft.com/>
- [MIC04b] Microsoft Corporation , “Microsoft Visual Basic Developer Center”,
<http://msdn.microsoft.com/vbasic/>, 2004
- [MIC04c] Microsoft Corporation , “Microsoft Visual C++ Developer Center”,

- <http://msdn.microsoft.com/visualc/>, 2004
- [MIC04d] Microsoft Corporation , “Microsoft Visual Studio Developer Center”
<http://msdn.microsoft.com/vstudio/> 2004
- [MIC04g] Microsoft Corporation, “Microsoft Windows”,
<http://www.microsoft.com/windows/>, 2004
- [MIN02] Minority Report,
<http://www.minorityreport.com/>
- [OGL04] OpenGL - The Industry Standard for High performance Graphics,
<http://www.opengl.org>
- [OPO04] Open Office,
<http://www.openoffice.org/>
- [PYT04] Python language program,
<http://www.python.org/>
- [PPC04] PowerPC processor,
<http://www-306.ibm.com/chips/products/powerpc/>
- [PTH04] Basic use of pthreads, an introduction to POSIX threads,
<http://www-106.ibm.com/developerworks/linux/library>
- [QAK04] Quake,
<http://www.idsoftware.com/games/quake/quake/>

- [QT04] Screenshoots,
<http://www.trolltech.com/screenshots/index.html?cid=11>
- [ROE90] J.Roebuck, “Clinical radiology of the Breast”, Heinemann Medical Books, Oxford, 1990
- [S304] S3 Graphics,
<http://www.s3graphics.com/index.html>
- [SDL04a] Simple Direct Media Layer,
<http://www.libsdl.org/index.php>
- [SDL04b] Focus on SDL, Ernest Pazera, 2004
- [SOL04] Solaris operating system,
<http://www.sun.com/software/solaris/>
- [SPI04] SpiritWorks, “Glossary of Internet Terms”,
<http://about-the-web.com/shtml/glossary.shtml>, 2004
- [SUN04] Sun Microsystem,
<http://www.sun.com/>
- [WRA04] The Java Advanced Imaging application programming interface
<http://www.sun.com/software/imaging/>
- [ZAM03] Zannoni M., Tesi di laurea “Algoritmi avanzati per la rivelazione di masse tumorali in mammografia digitale”

RINGRAZIAMENTI

Si ringrazia 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.

Si ringrazia il dott. Matteo Roffilli per avermi fornito preziose indicazioni su come procedere nello sviluppo di questo progetto, per le informazioni fornite e per la costante e continua assistenza.

Si ringrazia il dott. Omar Schiaratura per i numerosi consigli e il costante impegno riservatomi.

Per la infinita pazienza, per l'appoggio morale e fisico rivoltomi in questi anni di Università dedico i più sentiti ringraziamenti a Mauro senza il quale sarebbe stato tutto più difficile, Andrea e Brigoz per la loro infinita bontà e magnanimità, Gino per essere un buon compagno di beach volley, Don Luigi per essere sempre vicino a me nella preghiera e nel momento del bisogno, Luca, Marinella, i Ragazzi, Ercole, Moro e tutti coloro che mi hanno aiutato in questi anni.

A mia madre Morena, ai miei nonni Liliana e Livio e a tutta la mia Famiglia per aver reso possibile tutto questo, per aver creduto in me, per avermi sostenuto in ogni momento della vita, rendendomi sempre più forte e quello che sono.

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.

Oltre il prossimo orizzonte...

