Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Una Applicazione Open-Source e Cross-Platform per la Gestione Interattiva di Modelli di Grafica 3D

I. Marra, R. Del Gaudio, C. Vanzanella

RT-ICAR-NA-06-02 **03-2006**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) – Sede di Napoli, Via P. Castellino 111, I-80131 Napoli, Tel: +39-0816139508, Fax: +39-0816139531, e-mail: napoli@icar.cnr.it, URL: *www.na.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Una Applicazione Open-Source e Cross-Platform per la Gestione Interattiva di Modelli di Grafica 3D [1]

I. Marra2, R. Del Gaudio[2], C. Vanzanella[2]

Rapporto Tecnico N.: **RT-ICAR-NA-06-02**

Data: **03-2006**

---

[1] sottoposto per pubblicazione

[2] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Napoli, Via P. Castellino 111, 80131 Napoli

Una Applicazione Open-Source e Cross-Platform per la Gestione Interattiva di Modelli di Grafica 3D

R. Del Gaudio, I. Marra, C. Vanzanella

Sommario

In questo lavoro, viene presentata una applicazione interattiva per la visualizzazione di modelli di grafica 3D, che e' stata sviluppata utilizzando la libreria grafica Open Scene Graph (OSG), open-source e cross-platform, in grado di svincolare lo sviluppatore dall'utilizzo diretto delle chiamate OpenGL, fornendo cosi' un valido strumento per un più rapido sviluppo di applicazioni grafiche, ed ottimizzandole attraverso l'impiego della tecnologia degli Scene Graph.
L'applicazione realizzata mette a disposizione dell'utente, attraverso una semplice interfaccia grafica, degli strumenti che consentono un pieno controllo del modello, implementabili con qualsiasi dispositivo di input. Inoltre tale applicazione e' stata integrata in un ambiente di sviluppo per il medical imaging, basato su piattaforma open-source e volto all'implementazione di applicazioni di realta' virtuale altamente immersive.

1 Introduzione

I recenti progressi compiuti nel campo delle tecnologie informatiche e della Computer Graphics ci consentono di disporre oggi di nuovi media digitali che, in quanto estensione del nostro corpo, ci offrono forme di interazione con il mondo che ci circonda sempre più strette, soddisfacendo i nostri bisogni di conoscenza attraverso un approccio di tipo senso-motorio, certamente più naturale per l'essere umano, rispetto a quello di tipo simbolico-ricostruttivo, mediato dalla scrittura.

Scenari virtuali immersivi, realtà aumentata, interfacce tattili in grado di fornire sensazioni di forza all'operatore costituiscono solo alcune delle possibili tipologie tecnologiche utilizzate per offrire simulazioni sempre più vicine all'esperienza reale e caratterizzate da un alto grado di immersività, che sempre più risponde alle necessità di formazione e training dell'attuale società dell'informazione.

Tuttavia, applicazioni di grafica 3D con un buon grado di interattività ed immersività ancora richiedono sistemi altamente specializzati e spesso molto costosi, che necessitano di driver specifici, legando in tal modo lo sviluppo di una particolare applicazione ad un determinato hardware.

Risulta, quindi, molto sentita la necessita' di poter disporre di una architettura aperta e flessibile per lo sviluppo di software applicativo che consenta l'integrazione del più ampio range possibile di device e di componenti software, e che sia funzionale a settori di grande interesse, quale quello dei beni culturali e museali, del training e della formazione, nonché della visualizzazione e della simulazione medica e scientifica.

In questo lavoro viene presentata una applicazione scritta in C++ per la visualizzazione interattiva di modelli di grafica 3D di vario formato, basata sulla struttura degli *Scene Graph* e costruita utilizzando la libreria *Open Scene Graph*.

La scelta di Open Scene Graph e' stata motivata dalla necessita' di avere un ambiente di sviluppo object-oriented, open-source e cross-platform basato sulla libreria OpenGL, divenuta ormai uno standard nell'ambito della grafica 3D, in modo da consentire l'implementazione di applicazioni scalabili, performanti, portabili ed ottimizzate per operazioni di rendering real-time.

L'applicazione sviluppata, attualmente include una semplice interfaccia a partire dalla quale e' possibile richiamare varie funzioni, che consentono vari livelli di interazione con la scena 3D, ed un certo controllo, implementabile con qualsiasi dispositivo di input, degli oggetti presenti in essa.

Il software sviluppato, incapsulando le classi di base della libreria core dell'intero toolkit, osg, consente una corretta gestione dei grafi di scena attraverso l'utilizzo delle varie tecniche di ottimizzazione, quali view frustum culling, occlusion culling, small feature culling, Level Of Detail (LOD) nodes, state sorting, vertex arrays e display lists.

Esso, inoltre, e' stato progettato ed implementato per essere integrato in un ambiente di sviluppo per applicazioni di realta' virtuale altamente immersive, basato su di un'architettura software a componenti (fig.1), altamente scalabile e performante, in grado di connettere dispositivi eterogenei (CAVE, Motion Capture Systems, HMD, Data Glove,etc.) attraverso l'utilizzo di interfacce semplici e portabili.

Requisito essenziale della struttura modulare della architettura in cui integrare la componente realizzata, è l'impiego di librerie e strumenti *open source* e *cross platform,* che facilitino lo sviluppo di applicazioni di realtà virtuale immersiva, svincolando lo sviluppatore dai dettagli implementativi di una particolare piattaforma e determinati device per l'interazione e l'immersivita'.

In particolare, la scelta e' ricaduta sulle seguenti librerie: i) *OpenGL (Open Graphic Library)*, libreria di basso livello per lo sviluppo di applicazioni grafiche 2D e 3D, in grado di sfruttare al meglio l'accelerazione hardware delle schede video; ii)*FLTK (Fast Light ToolKit)* libreria di alto livello per la creazione di interfacce grafiche utente; iii)*VTK* (*Visualization ToolKit)* libreria di alto livello per la visualizzazione grafica e l'image processing; iv)*TK (Insight ToolKit)*, libreria di alto livello che implementa algoritmi di registrazione e segmentazione nel campo del medical imaging; v)*OSG* (*Open Scene Graph)*, libreria di alto livello per applicazioni di grafica

3D ad alte prestazioni; vi)*VRJ (Virtual Juggler)*, libreria di alto livello per lo sviluppo di applicazioni di realtà virtuale orientata all'integrazione di device eterogenei; vii)*Chromium*, Motore di rendering parallelo.

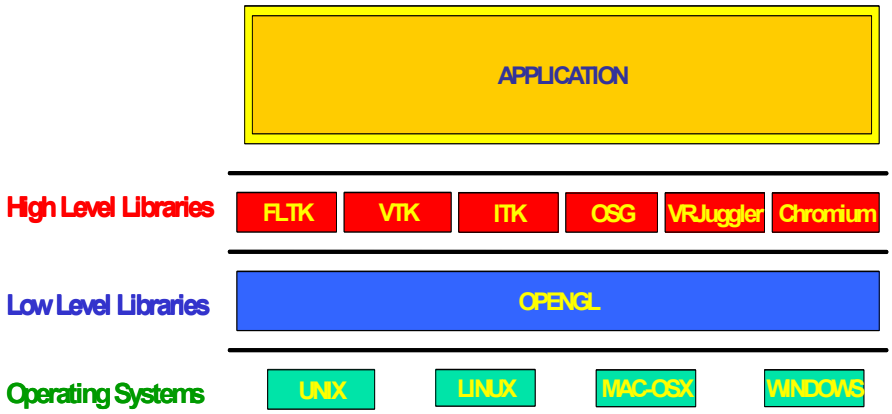


Figura 1 Architettura Software

2 Interattività ed Immersività

L'espressione *real time* ha un'implicazione profonda nella definizione dei problemi di grafica tridimensionale. Il vincolo del tempo reale è forte a tal punto da forzare la scelta di un sistema hardware più costoso di svariati ordini di grandezza rispetto a quello inizialmente previsto. Ma cosa significa esattamente? Tutti noi veniamo continuamente a contatto con prodotti di grafica 3D di eccezionale qualità, basti pensare alle ricostruzioni grafiche usate sempre più frequentemente in alcune trasmissioni televisive, o prodotti legati al campo dei beni culturali, o a quello medicale.

Non si può negare che il livello di realismo raggiunto sia eccellente. Se però si vanno ad osservare applicazioni di grafica in cui sia prevista una certa interattività (si consideri il caso dei video giochi), si può notare come la qualità ed il conseguente realismo della grafica non riesca mai a raggiungere i livelli elevatissimi di film o documentari.

Ciò accade perché la grafica 3D presente nei filmati ha una caratteristica importante: è generata *off line*. Ciascuno dei *fotogrammi*[1] che compongono

[1] un'animazione è composta da una serie di immagini, chiamate appunto fotogrammi o frame, che si susseguono ad alta velocità in modo da dare all'occhio l'illusione del movimento. Dal momento che il tempo di persistenza dello stimolo luminoso sulla rètina è di circa 40 ms, bisogna visualizzare almeno 25 fotogrammi al secondo per dare l'impressione di un'animazione fluida. Solitamente nei software di grafica si punta ad ottenere un frame rate (velocità di aggiornamento dei fotogrammi) di almeno 30 fotogrammi al secondo.

l'animazione è il risultato di un processo di rendering che può durare un tempo variabile, solitamente da alcuni minuti a diverse ore o addirittura giorni nel caso di scene molto ben definite, caratterizzate magari da fenomeni molto complessi il cui calcolo può richiedere molto tempo anche sui calcolatori più veloci. Se il prodotto da realizzare è un film, cioè un'animazione che lo spettatore "subisce" passivamente senza possibilità di interazione, il problema della velocità di rendering non si pone affatto. Infatti, il calcolatore e l'hardware 3D viene usato solo per la "creazione" del filmato (non per la "fruizione" da parte dell'utente). Alla fine, il filmato verrà poi "riprodotto" a velocità naturale.

Nelle applicazioni di grafica 3D interattiva, invece, la velocità del processo di rendering ha un'importanza fondamentale e può fare la differenza tra un buon sistema ed un sistema inefficace. In queste applicazioni bisogna generare la grafica "al volo" in risposta a sollecitazioni esterne. La grafica, cioè, deve essere in *tempo reale*. L'utente si aspetta di vedere una scena virtuale che cambia a seconda delle sue azioni. Affinché questo scopo venga raggiunto è necessario che il processo di rendering non richieda un tempo troppo lungo, perché ciò andrebbe a discapito dell'interattività. Pertanto, la generazione di ogni fotogramma deve essere abbastanza rapida da non far percepire all'utente un eccessivo ritardo tra la causa (il suo comando) e l'effetto (il cambiamento della scena visualizzata).

Questo vincolo impone una semplificazione obbligata dei contenuti grafici e soprattutto richiede l'impiego di pesanti approssimazioni nella riproduzione dei fenomeni in fase di rendering. Il risultato è una grafica più povera e sicuramente meno realistica di quella osservabile negli altri prodotti grafici non interattivi.

Figura 2 Rendering real-time

Figura 3 Rendering off-line

Purtroppo, le semplificazioni necessarie nel rendering in tempo reale hanno sempre determinato un livello di realismo piuttosto basso nelle applicazioni di grafica 3D interattiva. Negli ultimi anni, tuttavia, si è registrato un incremento impressionante nelle prestazioni e nelle capacità di calcolo dei dispositivi grafici di *mainstream*. I microprocessori dedicati all'elaborazione della grafica tridimensionale, *Graphic Processing Units* (GPU), sono oggi in grado di effettuare calcoli in virgola mobile con precisione a 32 bit a velocità elevatissime e, soprattutto, consentono l'elaborazione parallela su insiemi di dati multipli grazie alla loro architettura marcatamente SIMD (Single Instruction Multiple Data). Le schede grafiche odierne offrono una potenza di calcolo sufficientemente elevata da consentire riproduzioni grafiche tridimensionali in tempo reale molto realistiche anche su normalissimi PC.

3 Lo Scene Graph

Uno dei metodi maggiormente utilizzati per descrivere ed organizzare una scena di grafica 3D è quello degli Scene Graph o Grafi di Scena.

Lo Scene Graph è una struttura dati che organizza logicamente la rappresentazione di una scena grafica: una scena viene decomposta in oggetti, ogni oggetto in parti, ogni parte in poligoni.

Non esiste una definizione precisa di *grafo di scena*, questo accade perché i programmatori che l'implementano nelle loro applicazioni, in particolare nell'industria dei video giochi, ne prendono i principi di base adattandoli poi alle specifiche esigenze. Ciò implica che non esiste una regola precisa su cosa uno Scene Graph sia o non sia.

3.1 Implementazione dello Scene Graph

La forma più semplice di Scene Graph usa un vettore o una lista, visualizzando le sue forme attraverso una iterazione lineare sui nodi, ad uno ad uno. Un'altra operazione comune, la determinazione di quali forme intersechino il puntatore del mouse, viene anch'essa effettuata con una ricerca lineare. Questo per piccoli Scene Graph può anche essere accettabile.

Sfortunatamente, tali operazioni lineari, applicate a grafi di scena più grandi (scene più complesse), comportano un sensibile rallentamento della computazione.

La struttura di uno Scene Graph può essere quella più complessa di un albero o quella ancor più generica di un DAG (Direct Acyclic Graph o Grafo Diretto Aciclico). Nel primo caso ogni componente dello Scene Graph (ogni nodo) può essere diretto discendente di un solo altro nodo (Figura 4).

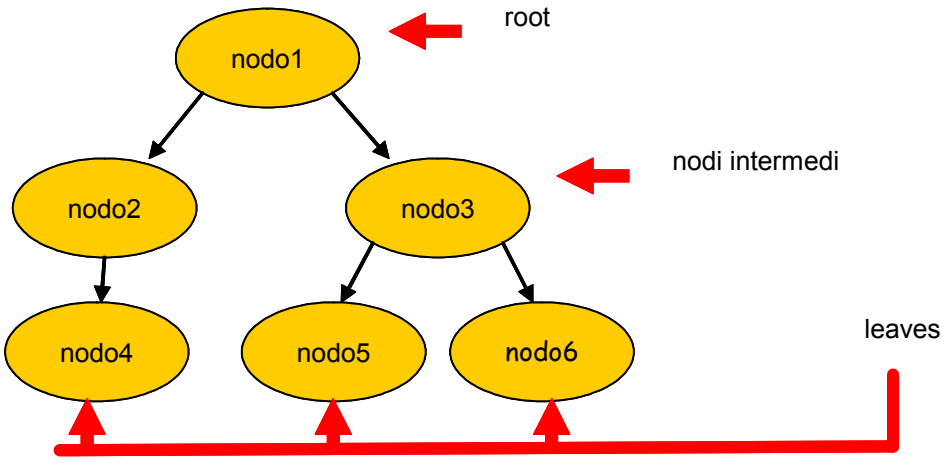


Figura 4 Struttura ad albero di uno Scene Graph

Nel secondo caso è possibile che un nodo possa discendere direttamente da più di un nodo, *node instancing*. Ad esempio (Figura 5), siano date le seguenti trasformazioni:

Trasf1 (ruota di 45°);
Trasf2 (trasla di (10,0));

il risultato visibile saranno due geometrie uguali ma trasformate in modo diverso. Esse sono state ottenute con un notevole risparmio di risorse in quanto si sono evitate inutili duplicazioni e, soprattutto, possibili inconsistenze.

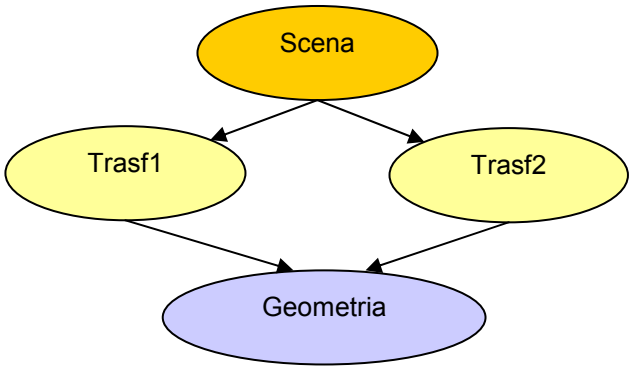


Figura 5 Node Instancing

In entrambi i casi, comunque, le relazioni fra nodi non permettono la creazione di cicli per cui, comunque ci si muova fra i rami, non si può tornare indietro al nodo di partenza.

La struttura ad albero è la più popolare forma di Scene Graph e spesso si assume il *Composite Design Pattern* per crearne la rappresentazione gerarchica di tutti gli elementi, "nodi intermedi" e "nodi foglia", che compongono la scena al di sotto del "nodo radice" che rappresenta l'intero "mondo virtuale" che si desidera rappresentare.

3.1 Vantaggi dello Scene Graph

Le ragioni chiave che portano tanti sviluppatori grafici a preferire lo Scene Graph come struttura dati nelle applicazioni di grafica 3D sono determinate in termini di:

Performance
La struttura gerarchica dello Scene Graph è un eccellente motore per massimizzare le performance grafiche. Permette di rappresentare oggetti strutturati e di gestirli sia come insieme sia come parti separate. Un buon Scene Graph impiega due tecniche chiave: scartare velocemente gli oggetti che non sono visibili nella scena (scartandoli senza la necessità di processarli tutti), ed avere la capacità di ordinare gli oggetti per le loro proprietà, come texture e materiali, in modo da permettere che tutti gli oggetti simili vengano disegnati insieme (rendendo minimi i costosi cambiamenti di stato). Senza la fase di scarto la CPU, i bus e la GPU sarebbero travolte da un enorme carico di dati insignificanti per la rappresentazione della scena.

Produttività
Lo Scene Graph possiede molti strumenti per lo sviluppo di applicazioni grafiche performanti. Esso gestisce al posto dell'utente tutta la parte grafica, riducendo a sole poche funzioni quello che con OpenGL si farebbe con tantissime righe di codice. Il composite design pattern, alla base della struttura ad albero, rende lo Scene Graph altamente flessibile e con un design riutilizzabile (con questo si intende che può essere facilmente adattato per risolvere ogni tipo di problema);

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

Scalabilità
La grafica 3D si sposa bene con configurazioni hardware complesse e avanzate come sistemi multiprocessore, e lo Scene Graph rende decisamente facile gestire questi tipi di sistemi. Infatti, aiuterà gli sviluppatori a concentrarsi nel programmare le proprie applicazioni senza preoccuparsi di gestire l'hardware, mentre il motore di rendering dello Scene Graph si preoccuperà delle diverse configurazioni hardware.
Open Scene Graph (OSG) è una libreria grafica 3D portabile, open source, realizzata per progetti a prestazioni elevate e quindi usata per applicazioni come simulazioni di volo, giochi, realtà virtuale, modellazione e visualizzazione scientifica.
Questa libreria è scritta interamente in C++ (Object Oriented) e OpenGL e può essere portata in tutte le piattaforme e sistemi operativi Windows, OSX, Linux, IRIX, Solaris e FreeBSD. La sua grande portabilità è uno dei punti chiave della sua diffusione.

4 OpenSceneGraph

E' il toolkit utilizzato per l'implementazione della applicazione interattiva di grafica 3D, di seguito descritta.

Si tratta di una libreria interamente scritta in C++ ed OpenGL, cross platform ed open source, particolarmente adatta per lo sviluppo di applicazioni grafiche ad alte prestazioni per simulatori di volo, videogiochi, realta' virtuale e visualizzazione scientifica. Basato sulla struttura dati scene graph, esso offre un framework object oriented al di sopra della libreria OpenGL, cosi' da svincolare lo sviluppatore dall'utilizzo di routine grafiche di basso livello.

Il progetto partì come hobby di Don Burns nel 1998, come mezzo per rendere portabile un simulatore di caduta planare scritta al massimo delle performance grafiche ed eseguito su piattaforma IRIX. Nel 1999, Robert Osfield iniziò ad aiutarlo col simulatore concentrandosi sull'implementazione degli elementi grafici per Windows. Nel settembre del 1999 il codice sorgente divenne open source, e nacque il sito Internet[2].

Nell'aprile del 2001, in risposta alla grande crescita dell'interesse nazionale, Robert Osfield si impegnò a tempo pieno sul progetto, che continua tuttora ad ampliare con la collaborazione di un gruppo di sviluppatori, e creò un supporto commerciale chiamato OpenSceneGraph

[2] www.openscenegraph.org

Professional Service che comprendeva un servizio di consulenza per l'apprendimento. Alla fine del 2001 Don Burns formò una sua compagnia chiamata Andes Computer Engineering e partecipò alla progettazione e al supporto di OpenSceneGraph ma con un progetto complementare chiamato OpenProducer e BluMarbleViewer.

Figura 6 Esempi di applicazioni con OSG

Tutti i sorgenti dell'OSG sono pubblicati sotto l'OpenSceneGraph Public License (una versione di LGPL – Library General Public License) che permette l'uso, la modifica e la diffusione del progetto open source.

La libreria ha già guadagnato una certa reputazione come guida open source alla struttura delle Scene Graph, e si sta affermando come una valida alternativa a librerie commerciali già in uso. Molte compagnie, ricerche universitarie e appassionati di grafica stanno già utilizzando l'OSG per i loro lavori in tutto il mondo.

Tuttora il progetto OpenSceneGraph è in completa evoluzione ed è arrivato alla versione 1.0RC (Release Candidatex) comprendente 3 pacchetti: OpenProducer, OpenThreads e OpenSceneGraph, descritti nei paragrafi seguenti.

4.1 OpenProducer

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

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

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

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

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

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

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

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

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

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

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

4.2 OpenThreads

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

Questa libreria non è direttamente utilizzata dal programmatore, ma serve per l'utilizzo e la compilazione delle librerie OpenProducer e OpenSceneGraph. OpenThreads mette a disposizione dei vari oggetti i quattro tipi fondamentali della programmazione threads, ovvero Thread, Mutex, Barrier e Condition.

4.3 OpenSeneGraph

Il pacchetto OpenSceneGraph contiene al suo interno tutti i principali *core* della libreria indispensabili per il suo corretto funzionamento. Segue la lista dei core di OpenSceneGraph:

osg
cuore della libreria OSG, fornisce le classi di base per i grafi di scena: Node, State e Drawable, include inoltre altre classi di supporto matematico e generale; ad esempio per l'apertura di file nativi ".osg";
osgDB
libreria di supporto per la gestione di letture e scritture di scene graph;
osgFX
libreria che estende la libreria principale osg fornendo strutture per effetti speciali;
osgGA
le lettere 'GA' stanno per GuiAbstraction (GUI astratta); è un namespace che fornisce facilitazioni per aiutare gli sviluppatori di osg a lavorare con diversi sistemi a finestra;
osgIntrospection
libreria di introspezione per evitare interrogazioni e chiamate di metodi e attributi vietati a tempo di esecuzione;
osgParticle
pacchetto di gestione degli effetti particelle;
osgProducer
libreria che contiene le basi per la visualizzazione della scena. Integra OpenProducer e permette la costruzione di una finestra e l'interazione keyboard-mouse;
osgSim
libreria di supporto ad osg per nodi e disegni che specifichino simulazioni visuali;
osgTerrain
libreria di supporto ad osg per la generazione di effetti di terreno geografico;
osgText
pacchetto per la gestione del testo; contiene al suo interno le classi per il rendering di font true type;
osgUtil
classe che contiene al suo interno numerose utility come il setup o il rendering/draw o l'attraversamento/aggiornamento; operatori sui grafi di scena come l'ottimizzazione;
osgUtx
unità di controllo delle strutture.

Come accennato precedentemente, OSG include numerosi Plugin che semplificano le azioni di importazione ed esportazione di file grafici nei diversi formati di ultime generazioni. Alcuni esempi:

osgPlugin file nativi (.osg);
fltPlugin file OpenFlight (.flt);
lib3dsPlugin file AutoStudioMax (.3ds);
loPlugin file LightWave binari (lwo,.lw,.geo);
rgbPlugin file RGB;
pngPlugin file PNG;
gifPlugin file GIF;
jpegPlugin file JPEG;
tgaPlugin file TGA;
tiffPlugin file TIF.

5 Realizzazione della Applicazione

Come anticipato, il programma ha la possibilità di girare su diversi sistemi operativi quindi il codice sorgente può essere studiato indipendentemente dal compilatore che genererà l'eseguibile e, soprattutto, dalla piattaforma scelta.

Il sistema è composto di diversi moduli software che eseguono diverse funzioni ed è stato progettato con il linguaggio UML (Unified Modeling Language). Ciascun componente fisico è costituito da un header, identificato dall'estensione ".h" e da un file contenente l'implementazione, identificato dall'estensione ".cpp", e compilato in ambiente Microsoft Visual Studio .NET 2003 per Windows XP.

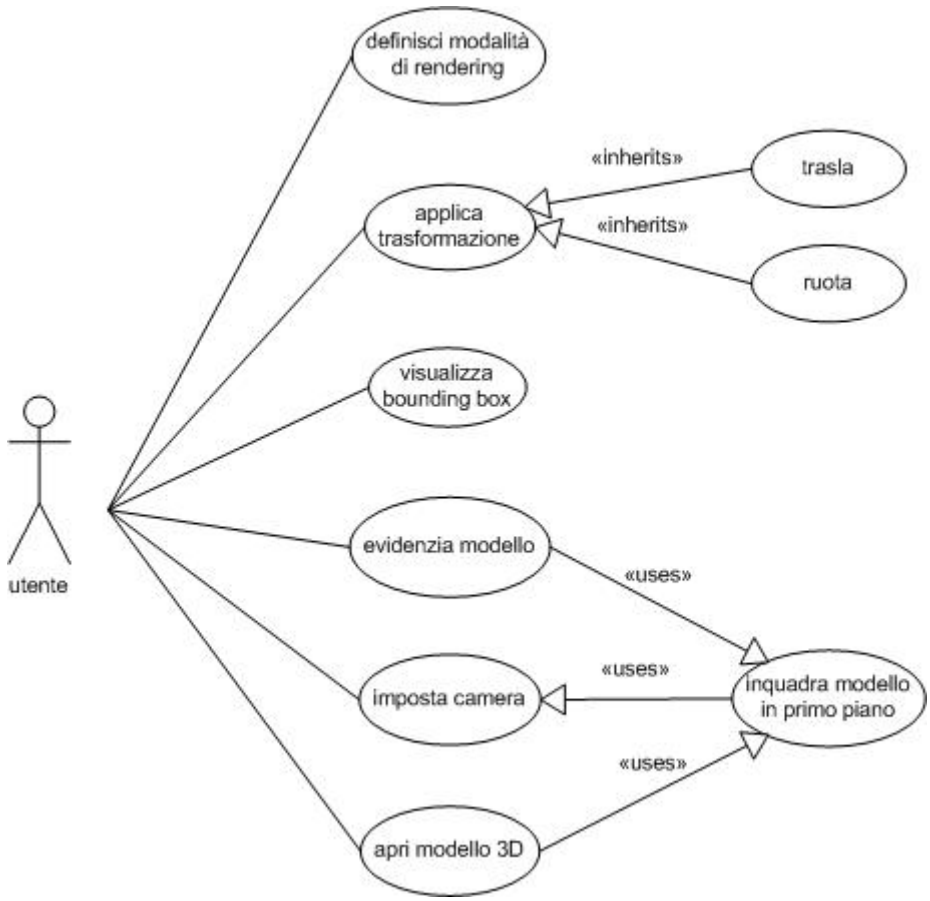
Il codice è commentato seguendo la sintassi del Doxygen per favorire la generazione automatica della documentazione. Infatti Doxygen è un sistema di documentazione per diversi linguaggi di programmazione, potendo generare on-line della documentazione (in HTML) e/o manuale di riferimento off-line da un insieme di archivi sorgente opportunamente commentati. La documentazione viene dunque estratta direttamente dalle fonti, il che rende molto più facile mantenerla consistente con il codice sorgente.

5.1 Diagramma dei casi d'uso

I diagrammi dei casi d'uso sono il punto di partenza quando si progetta un nuovo sistema con UML. Essi vengono utilizzati per enumerare i requisiti del sistema e vengono modellati per fornire una visione di alto livello, illustrando le varie funzionalità che esso offrirà all'utente.

Nel progetto proposto, l'informazione di partenza è costituita da un modello tridimensionale in uno dei formati compatibili già visti in precedenza. Il sistema deve poter fornire un'interfaccia utente che consenta all'operatore di poter visualizzare e manipolare il modello caricato in partenza. Tra le varie operazioni da effettuare sul modello 3D visualizzato le più rilevanti riguardano sicuramente il manipolatore di telecamera, la scelta della modalità di rendering e la visualizzazione dei bounding box.

L’operatore deve poter guardare il modello dall’esterno girandovi attorno (ruotando la telecamera) oppure, più semplicemente, in modalità *panning* (traslando la telecamera). Inoltre, può essere sicuramente utile (soprattutto in campo medico) scegliere una modalità di rendering più dettagliata, che permetta la visualizzazione *wireframe* (lati dei poligoni) o *point* (singoli vertici), piuttosto della modalità di default *fill* (poligoni pieni); ciò potrebbe facilitare, in campo medico, l’individuazione di una patologia. Analogamente, la visualizzazione dei bounding box relativi all’intera scena (fisso) ed ai singoli oggetti che la costituiscono (questi appariranno al passaggio del mouse sull’oggetto), può essere d’aiuto per evidenziare quelle che sono le effettive dimensioni di una parte parzialmente visibile da una certa angolazione.
In Figura 7 è riportato il diagramma dei casi d'uso che determinano i requisiti per tutte le funzionalità necessarie.



5. 2 Gerarchia delle classi

Sono qui trattati gli elementi fondamentali utilizzati per rappresentare staticamente oggetti e funzionalità dell'intero sistema, ovvero, le classi. L'analisi dei requisiti ha portato alla realizzazione di un certo numero di classi, con relativi metodi ed attributi.

L'analisi dei requisiti ha portato alla realizzazione della gerarchia delle classi mostrata in Figura 8. I metodi e gli attributi saranno discussi successivamente per le principali classi progettate.

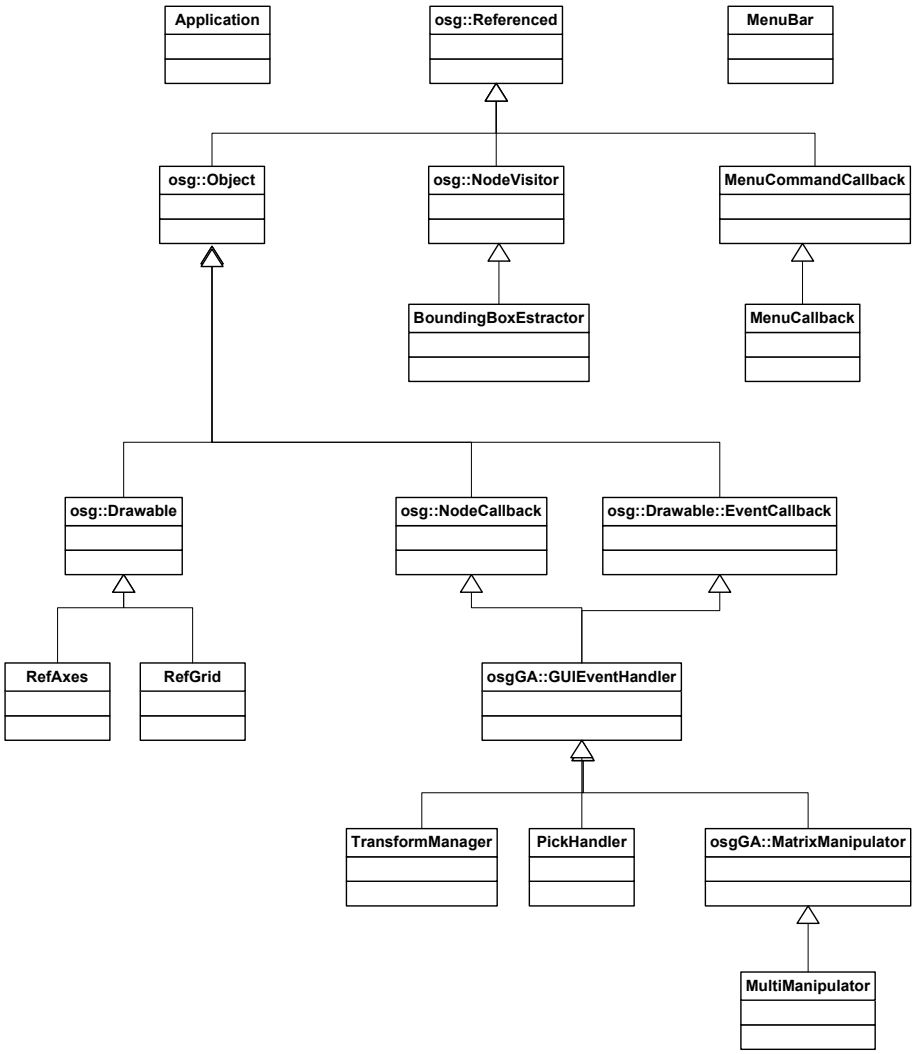


Figura 8 Gerarchia delle classi

La classe **Application** si occupa della gestione dell'intera applicazione. Il programma principale crea un'istanza di questa classe e quindi ne chiama il metodo getRootNode()per ottenere il nodo radice del grafo di scena da comunicare al visualizzatore. Il grafo di scena è creato dal costruttore di questa classe e comprende l'intera scena, inclusi gli elementi di interfaccia (menu e testo scritto) e il modello 3D specificato dall'utente. Quest'ultimo viene comunicato al costruttore di Application e quindi deve essere caricato dal programma principale o comunque dal codice che crea l'istanza di Application. Il grafo di scena non contiene solo i modelli e gli oggetti di supporto ma implementa anche le funzionalità di gestione degli eventi del mouse e della tastiera per quanto riguarda il funzionamento del menu, risultando dunque self-contained. La gestione del *picking* degli oggetti tramite mouse è invece esterna alla classe. Inoltre, è importante evidenziare il fatto che, ad ogni istante, un'istanza della classe Application ha un ed una sola operazione corrente (scelta dall'utente) che definisce il comportamento dell'applicazione. L'operazione di default è NONE, ovvero nessuna.

La classe **MenuBar** viene usata come strumento per generare dei menu grafici a pulsanti. L'utilizzo è il seguente: si crea un'istanza di questa classe, si eseguono una o più chiamate al metodo `addButton()` specificando i dati dei pulsanti da aggiungere, quindi si chiama il metodo `buildSceneGraph()` per generare il grafo di scena del menu. A questo punto l'istanza della classe MenuBar può essere distrutta.

La classe **BoundingBoxExtractor** è un visitatore di nodi (NodeVisitor) il cui scopo è determinare il bounding box di un grafo di scena accumulando i bounding box di tutte le geometrie incontrate durante l'attraversamento del grafo.

La struttura **MenuCommandCallback** funge da base per le classi o strutture definite dall'utente per intercettare i comandi di un menu. L'utente scrive una classe derivata da essa (classe **MenuCallback**) implementando i metodi virtuali `enable()` e `disable()` che vengono chiamati dal gestore del menu quando si verifica un'azione di attivazione/disattivazione di una voce del menu.

La classe **MultiManipulator** implementa un manipolatore di telecamera multi-funzione simile all'originale TrackballManipulator. Si differenzia da quest'ultimo in quanto le funzioni disponibili (spostamento telecamera, zoom in e zoom out) sono separate e mutuamente esclusive. L'utente può in ogni momento selezionare una specifica funzione e quindi limitare il movimento del mouse a quel tipo di operazione. L'interfaccia di questa classe non aggiunge nuove funzionalità rispetto ai tradizionali manipolatori di telecamera, a parte le funzioni per impostare la modalità di funzionamento.

La classe **PickHandler** intercetta gli eventi del mouse per consentire la selezione e l'evidenziazione di nodi visualizzati (parti del modello 3D). Al costruttore della classe deve essere comunicato l'oggetto Application che rappresenta l'applicazione, il quale sarà utilizzato per notificare l'avvenuta selezione dei nodi tramite i metodi `Application::selectNode()` e `Application::highlightNode()`. L'utente deve creare un'istanza di PickHandler ed aggiungerla alla lista di handler di eventi del visualizzatore, operazione solitamente svolta nel programma principale.

5.3 Funzionalità dell'applicazione

Il software realizzato presenta una interfaccia semplice e di immediata comprensione (Figura 9).

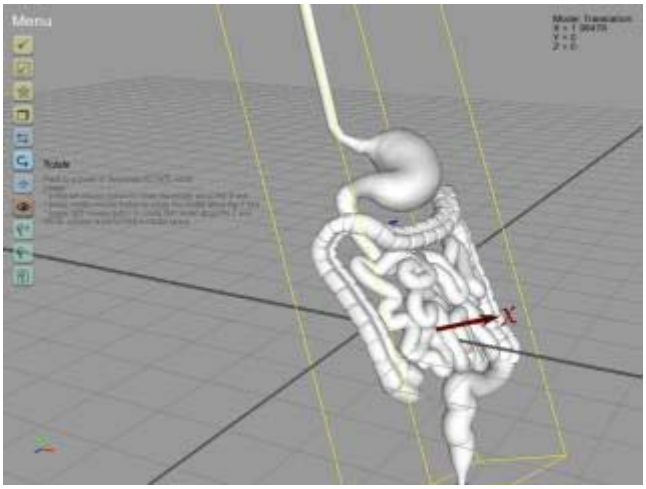


Figura 9 Visualizzazione di un modello 3D

Il menu consente di effettuare sul modello 3D visualizzato diverse operazioni, ciascuna delle quali viene esplicitamente descritta con un breve messaggio che comparirà al passaggio del mouse sulle rispettive icone. Precisamente:

Select

Push to activate or deactivate SELECT mode. When active, click on a part to select it and hide the rest of the model. Press SPACE BAR to restore the default view;

Select All

Push to show all parts of the model;

Render Mode

Toggle between Fill, Wireframe and Point render mode;

Bounding Box

Display/hide bounding boxes;

Move

Push to activate or deactivate MOVE mode. Usage: press left mouse button to move the model along the x axis, press middle mouse button to move the model along the y axis, press right mouse button to move the model along the z axis. NOTE: translation is perfomed in world space;

Rotate

Push to activate or deactivate ROTATE mode. Usage: press left mouse button to rotate the model about the x axis, press middle mouse button to rotate the

model about the y axis, press right mouse button to rotate the model about the z axis. NOTE: rotation is perfomed in model space;

Reset Transform

Push to reset model's translation and rotation to zero;

Camera

Push to activate or deactivate CAMERA mode. Usage: press left mouse to rotate the camera, press right mouse button to pan the camera;

Zoom In

Push and hold to zoom in;

Zoom out

Push and hold to zoom out;

Zoom Extent

Push to zoom to model's extent.

Conclusioni e Sviluppi Futuri

L'applicazione realizzata interamente in C++, utilizza due librerie grafiche, OpenGL e Open Scene Graph, mirate alla realizzazione di applicazioni per la grafica 3D altamente performanti, esse hanno la caratteristica comune di essere multi-piattaforma ed open source, per le quali, quindi, la ricerca di documentazione è stata favorita dal fatto che risultano virtualmente accessibili e disponibili a tutti gli sviluppatori; inoltre la loro totale compatibilità ha facilitato l'implementazione stessa.

Attualmente si sta gia' lavorando all'integrazione dell'applicazione in un ambiente di sviluppo open source e cross platform per la realta' virtuale che possa offrire oltre all'interazione con gli oggetti della scena visualizzata, anche un alto grado di immersivita', per simulazioni sempre più vicine all'esperienza reale, attraverso l'uso di dispositivi in grado di imitare al meglio alcune sensazioni fisiche. In particolare, il componente software realizzato con Open Scene Graph lo si sta integrando in una applicazione che, utilizzando il toolkit VR Juggler, consente di creare l'ambiente di realta' virtuale con il quale l'utente andrà ad interagire

Sviluppare applicazioni di realtà virtuale è un procedimento lungo e complesso, e' necessario preoccuparsi della parte grafica, dei dettagli dell'hardware, delle caratteristiche di rendering, dell'interazione con le periferiche.

Risulta quindi evidente come un ambiente di sviluppo, appositamente strutturato per questi scopi, possa semplificare sensibilmente la creazione di tali applicazioni, potendosi preoccupare solo degli aspetti caratteristici della particolare applicazione, attraverso un elevato livello di astrazione.

Attivita' successive riguarderanno l'integrazione in un siffatto ambiente di sviluppo di un motore di rendering parallelo, Chromium, che, unificando la potenza di calcolo di un insieme pressoché omogeneo di workstation, fornisce una interfaccia virtualizzata all'hardware, attraverso le API OpenGL,

in grado di generare in real time immagini fotorealistiche anche a partire da scene di elevata complessita', spesso necessarie ai fini di una buona simulazione.
.

Bibliografia

[1] "Computer Graphics" second editing Prentice Hall International Editions - Donald Hearn, M. Pauline Baker

[2] "Fondamenti di Computer Graphics" - G. Attardi

[3] "C++ Unleashed" SAMS 1998 - J. Liberty's

[4] "Sviluppo di sistemi informativi con UML" Addison-Wesley prima edizione aprile 2002 - Leszek A. Maciaszek

[5] "Hierarchical Geometric Models for Visibile Surface Algorithms" – James H.Clark – University of California at Santa Cruz

[6] "Designing Virtual Reality Systems – the structured approach" – Gerard Jounghyun Kim – Springer

[7] "VR Juggler: A Framework for Virtual Reality Development", Christopher Just, Allen Bierbaum, Albert Baker, Carolina Cruz-Neira

[8] "R. Robb, Virtual Endoscopy: Evaluation Using the Visible Human Datasets and Comparison with Real Endoscopy in Patients, Medicine Meets Virtual Reality", Studies in Health Technology and Informatics 39 (1997) 195-206.