Facoltà di Scienze Matematiche Fisiche e Naturali
Corso di Laurea Specialistica in Informatica

SL3D: un sistema per la realizzazione di presentazioni tridimensionali

Autore
Antonio Brasile

Relatori:

Dott. Paolo Cignoni

Dott. Fabio Ganovelli

Controrelatore:

Prof. Giuseppe Attardi

Anno Accademico 2005/2006

. . . ai miei genitori e ad Antonia.

Sommario

SL3D è un sistema per la realizzazione di presentazioni tridimensionali.

Per effettuare presentazioni all'interno delle conferenze e seminari che hanno per oggetto argomenti di grafica 3D si utilizzano attualmente gli strumenti classici per presentazioni che tutti conosciamo (PowerPoint, KeyNote, Impress, ...), con risultati poco soddisfacenti, dovuti al fatto che tali software sono pensati per operare in ambito bidimensionale.

Si deve, quindi, ricorrere all'utilizzo di snapshot e video, dove quand'anche, come nel video, si riesca a mantenere l'effetto 3D, si perde sempre e comunque la possibilità di interagire con la scena.

Da qui l'idea di realizzare un software che integri uno strumento per presentazioni, un Viewer 3D, ed un Editor minimo per poter creare una scena 3D in maniera semplice ed intuitiva.

Questo è ciò che si propone di fare SL3D, che cerca nel suo piccolo di colmare la carenza che i software per presentazioni hanno in tema di grafica 3D, senza velleità di realizzare nel corso di una sola tesi uno strumento completo ed esaustivo, che un software di tipo commerciale, in questo ambito, richiederebbe.

La tesi è suddivisa nei seguenti capitoli: nel primo faremo una breve panoramica dello Stato dell'arte e analizzeremo OpenSG, che costituisce il background di conoscenze per poter affrontare il prosieguo della tesi; nel

secondo descriveremo il sistema dal punto di vista dell'utente, descrivendo l'interfaccia e seguendo la costruzione di una slide; nel terzo approfondiremo l'architettura del sistema dal punto di vista più prettamente informatico; infine nel quarto descriveremo un esempio pratico.

Indice

Elenco delle figure

Capitolo 1

Stato dell'Arte

1.1 Introduzione

In questo capitolo effettueremo una panoramica sui prodotti software disponibili sul mercato e su quelle che sono le loro caratteristiche; ciò ci consentirà di meglio definire e collocare SL3D nel panorama esistente.

Analizzeremo i software per presentazioni in 2D ed in particolare PowerPoint; tratteremo brevemente degli editor e dei viewer utilizzati da chi si occupa di grafica 3D; ci soffermeremo su Virtual Inspector, strumento destinato agli utilizzi della grafica 3D nel campo dei beni culturali e termineremo con un'ampia trattazione di OpenSG, che è una libreria per la creazione di SceneGraph, che caratterizzerà in modo profondo la struttura di SL3D.

1.2 I software oggi disponibili

Attualmente ci sono diversi prodotti software disponibili sul mercato che consentono la realizzazione di presentazioni. I più noti sono senza dubbio

PowerPoint del pacchetto di Office per Windows, Impress di OpenOffice per Linux e Keynote per Mac.

Tali software, pur con delle differenze, hanno delle caratteristiche molto simili tra loro. Tra quelle comuni ci sono, senza dubbio, la possibilità di inserire testo formattato e immagini, di visualizzare video, creare effetti audio e la possibilità di costruire dei grafici di vario tipo (come istogrammi, diagrammi, ...), di impostare sfondi mediante immagini o con effetti di colore. Inoltre, tutti mettono a disposizione un sistema di transizioni attivabili mediante click del mouse o input da tastiera, che consente di creare effetti come la comparsa di oggetti o di testo, allo scopo di tenere sempre viva l'attenzione del pubblico.

Ma ciò che più di tutto li accomuna è il fatto di essere stati sviluppati per presentazioni in 2D e che non permettono di editare e manipolare oggetti tridimensionali. Talvolta, questi software cercano di "simulare" effetti 3D con l'intento di creare un più accattivante impatto scenografico, come, ad esempio, fa Keynote nella sua ultima release con i grafici tridimensionali texturizzati (vedi fig. 1.1).

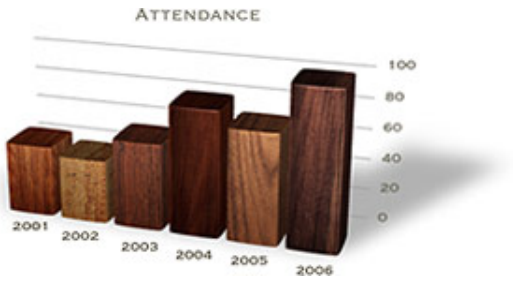


Figura 1.1: *Esempio di istogramma texturizzato in Keynote.*

Tra le ragioni di questa scelta c'è sicuramente quella di raggiungere un

pubblico di utenti più vasto e privo di quelle conoscenze specifiche, che l'ambito del 3D invece richiede.

Per creare effettivamente una scena o un'animazione 3D dobbiamo ricorrere, quindi, a software di editing, come 3DStudio Max o Maya, che sono programmi di grafica vettoriale tridimensionale e di animazione, che trovano le loro principali applicazioni in ambito cinematografico e nello sviluppo di videogames. Proprio per gli utilizzi ai quali sono destinati tali software sono estremamenti complessi, richiedono conoscenze specifiche e sono anche estremamente costosi.

Ci sono poi dei semplici software di visualizzazione, Viewer 3D, che permettono di importare modelli 3D di vari formati e visualizzarli, utilizzati, ad esempio, in campo medico o sempre più in quello dei beni culturali.

Analizzeremo più nel dettaglio il software per presentazioni in 2D maggiormente utilizzato, che è sicuramente PowerPoint di Office, e di seguito ci soffermeremo su Virtual Inspector, che é un Viewer 3D utilizzato nell'ambito dei beni culturali.

1.3 PowerPoint

PowerPoint[3] è un programma di presentazioni grafiche, un software che aiuta a creare una presentazione con "proiezione" di diapositive.

Fornisce strumenti utili per creare gli oggetti che rendono efficace una presentazione: diagrammi, grafici, elenchi puntati e numerati, testo che attrae l'attenzione, filmati e effetti audio; inoltre, facilita la creazione dei necessari sussidi per una presentazione, quali stampati, note per il relatore e lucidi.

Quando si crea una presentazione si può scegliere una modalità di lavoro guidata servendosi di un modello tra i vari che il software mette a disposizione

(vedi fig. 1.2), cioè di una diapositiva completa di sfondo, testo ed immagine che consente di mantenere una coerenza di stile per tutta la durata della presentazione.

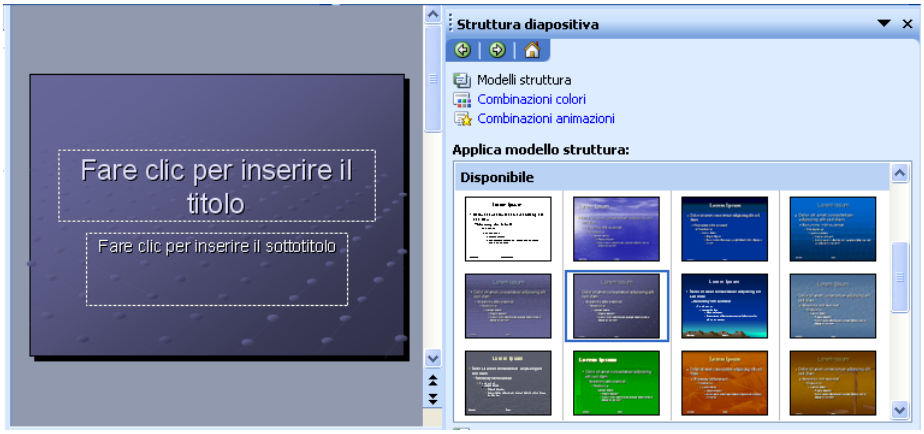


Figura 1.2: *Finestra per la scelta di un modello in PowerPoint.*

Figura 1.3: *Finestra di autocomposizione guidata di PowerPoint.*

Oppure si può ricorrere all'autocomposizione guidata (vedi fig. 1.3), che guida l'utente nell'organizzazione generale della presentazione, proponendogli

tipi (presentazioni di piani commerciali, risultati finanziari, piani di marketing, rapporti sull'evoluzione di un progetto,...), stili (presentazioni sul web, lucidi a colori, lucidi in bianco e nero,...) ed opzioni diverse.

In alternativa l'utente può scegliere di non appoggiarsi a modelli già predisposti e di impostare autonomamente la propria diapositiva/presentazione.

Per aiutare l'utente nelle varie fasi di sviluppo di una presentazione PowerPoint offre cinque diverse modalità di visualizzazione:

Visualizzazione Diapositive : visualizza una diapositiva per volta;

Visualizzazione Struttura : mostra un elenco dei titoli delle diapositive ed i loro contenuti in formato struttura;

Visualizzazione Sequenza diapositive : mostra un'immagine ridotta di ciascuna diapositiva nella stessa finestra; è comoda per organizzare la presentazione ed aggiungere gli effetti di passaggio tra le diapositive, dette transizioni;

Visualizzazione Pagina Note : mostra un'immagine ridotta di una singola diapositiva, con un'area nella quale si possono immettere le note che il relatore può utilizzare durante la presentazione;

Presentazione diapositive : in questa modalità ha luogo la presentazione in sequenza delle diapositive in full screen: si può fare avanzare la presentazione o con il click del mouse o da tastiera.

Gli oggetti costituiscono gli elementi base di una diapositiva. Una diapositiva di presentazione spesso contiene oggetti di testo (titoli, sottotitoli ed elenchi), oggetti visivi (figure, ClipArt, grafici e diagrammi), oggetti multi-

mediali (sequenze audio e video, collegamenti ipertestuali). Questi oggetti si possono manipolare con un insieme comune di operazioni base:

- selezione e deselezione
- ridimensionamento e spostamento
- eliminazione

Perché una presentazione risulti ben comprensibile PowerPoint mette a disposizione la funzione Layout, che aiuta a disporre gli oggetti in modo omogeneo.

Quando si crea una nuova diapositiva si applica un layout automatico (vedi fig. 1.4), che contiene segnaposto per i testi ed altri oggetti.

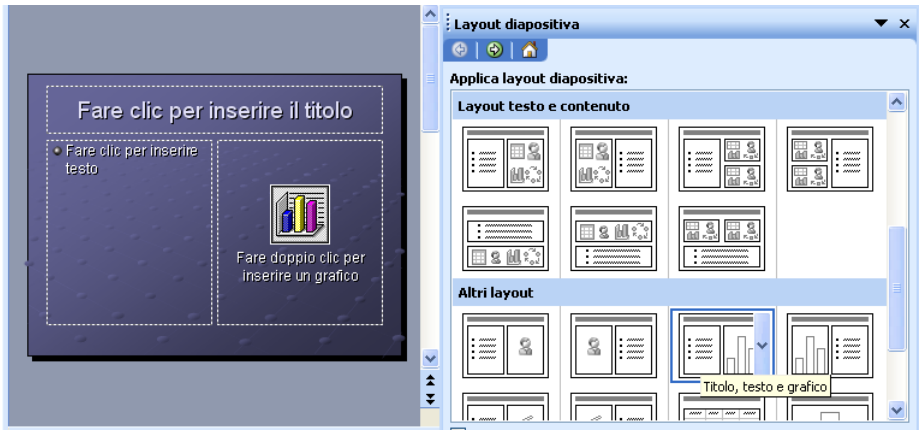


Figura 1.4: *Finestra per la scelta del layout di una diapositiva PowerPoint.*

Una caratteristica importante di PowerPoint è quella di poter "animare" le presentazioni: è possibile creare degli effetti di animazione, come per esempio far comparire i testi uno per volta dall'alto e dal basso o con dissolvenze, avviare l'esecuzione di filmati, animare un grafico,....

Si possono usare Animazioni "Preimpostate" o "Personalizzate". Quando si personalizza un'animazione si possono scegliere:

- l'ordine di animazione: scegliere in quale ordine far comparire i diversi oggetti presenti nella diapositiva;
- l'intervallo: il tempo tra un effetto e l'altro, che può essere su imput dell'utente o temporizzato;
- gli effetti: si può scegliere sia il tipo di animazione che il tipo di suono.

Anche nell'inserimento delle transizioni tra le diapositive è possibile scegliere come passare da una slide all'altra (se con un click o dopo un certo numero di secondi), con che velocità, se o meno con un effetto sonoro.

1.4 Virtual Inspector

Negli ultimi anni assistiamo ad un sempre maggiore utilizzo della grafica 3D nell'ambito dei beni culturali, in interventi di conservazione e restauro di opere d'arte ed architettoniche.

Ciò è stato reso possibile grazie alle ricerche che numerosi centri, come il Visual Computing Group (VCG) del Cnr di Pisa, hanno effettuato sulle tecniche di 3D scanning. Tale tecnica consente di acquisire mediante scansione laser un modello tridimensionale estremamente accurato di un'opera d'arte, sul quale è possibile effettuare misurazioni e studiare interventi di restauro. Tali riproduzioni possono anche essere messe a disposizione dei visitatori di un museo per poter osservare più da vicino le opere d'arte che, per collocazione o per dimensione, altrimenti risulterebbero inaccessibili (pensiamo, ad esempio, al David di Michelangelo alto 5,17 metri).

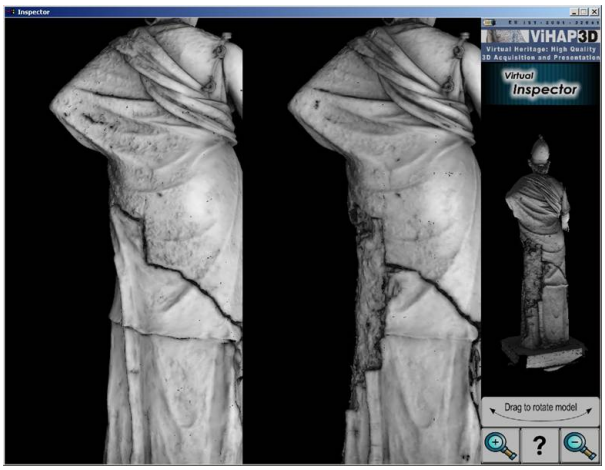


Figura 1.5: *Esempio di scena creata da Virtual Inspector: in questo caso si tratta della Minerva di Arezzo prima e dopo il restauro.*

A supporto di tale attività nel 2001 è stato implementato Virtual Inspector[9].

Tale software consente ad un utente medio, ad esempio il visitatore di un museo, di fruire di un'opera d'arte acquisita tramite la tecnica del 3D scanning, fornendo un approccio interattivo basato sulla tecnica del "point and click" e consentendo la manipolazione dell'oggetto "renderizzato" attraverso una trackball, che permette di ruotare il modello in uno spazio tridimensionale.

Nella realizzazione di Inspector si è tenuto conto di due ordini di problemi: gestire e visualizzare l'alta complessità dei dati 3D prodotti con la tecnologia 3D scanning su un normale computer e contemporaneamente fornire uno strumento di visualizzazione di semplice utilizzo.

Solitamente la scena creata da Inspector è suddivisa in due main frames (vedi figura 1.5): il frame di destra, che occupa la parte più piccola , costituisce l'area di selezione dove viene visualizzato il modello in scala dell'opera presentata insieme con qualche pulsante ed ha la funzione di non far perdere

l'orientamento all'utente; il frame di sinistra, dove è possibile interagire con la trackball, visualizza la porzione dell'oggetto selezionata nel frame di destra.

Per risolvere il problema della complessità dei modelli 3D, Virtual Inspector utilizza un modello LOD (Level of Details), che consiste nel visualizzare l'oggetto con differenti livelli di dettaglio a seconda della distanza corrente dall'osservatore.

Una delle caratteristiche più interessanti di Inspector, e che ha costituito spunto nell'implementazione di SL3D, è che la configurazione dell'intero sistema viene specificata attraverso un semplice file Xml, editabile con un comune editor di testo. Infine, nel 2004, in occasione del progetto "Arrigo VII" sono stati introdotti dei supporti per "Hot Spots", che consentono di associare dati multimediali (es. pagine HTML) a qualunque punto o regione di un modello 3D. Ciò permette di creare presentazioni interattive dove il modello 3D costituisce anche un link alle proprie informazioni.

1.5 OpenSG

OpenSG [1] è un sistema di real time rendering Open Source per la costruzione di scenegraph basato sulla libreria grafica OpenGL[4].

Quello che OpenSG fa è immagazzinare l'intera scena in un grafo al fine di ottimizzare i dati passati alla scheda video. Strutturando la scena con un grafo è possibile, ad esempio, tagliare le parti della scena non visibili, minimizzare i cambi di stato OpenGL, ordinare oggetti semitrasparenti in ordine di profondità, aumentando così la performance di rendering.

L'idea base dello Scenegraph è un grafo, cioè un insieme di nodi connessi tra loro, che rappresenta la scena che vogliamo visualizzare.

Nei nodi è contenuta la parte informativa sulla scena: immagazzinano informazioni sui materiali, trasformazioni, geometria ed altro ancora.

Il rendering della scena si ottiene visitando il grafo (solitamente con una politica "left first-depth first").

Lo Scenegraph è composto sempre da:

- una radice (dalla quale parte la visita);
- nodi intermedi: possono avere figli e di solito settano posizioni, applicano trasformazioni (rotazioni, scalature, traslazioni, ...) e definiscono relazioni tra nodi;
- le foglie, che per definizione non hanno figli e contengono geometria.

Ovviamente OpenSG non è l'unico sistema di costruzione di scenegraph: ne esistono altri come Performer, OpenInventor, Java3D, per nominarne solo alcuni. Ma rispetto agli altri ha il vantaggio di essere open source, di gestire strutture dati in multithread in modo molto semplice e di essere facilmente estendibile, integrabile in un qualunque sistema di windowing e portabile.

Iniziamo ad analizzare le caratteristiche principali di OpenSG.

1.5.1 Nodes e Cores

Il primo aspetto di OpenSG che analizzeremo riguarda quella che è la struttura di creazione dello scenegraph, che lo differenzia dagli altri sistemi suesposti.

Gli altri sistemi utilizzano il nodo inteso in maniera classica, OpenSG distingue tra:

- *node*: i *nodes* sono utilizzati solo per definire la gerarchia del grafo; contengono i loro eventuali figli ed un *core*;

- *core*: contiene la parte significativa dell'informazione (geometria, trasformazione, ...).

In OpenSG ogni *node* ha un solo "parent", ma può avere molti "children"; i *cores* invece possono essere condivisi da più *nodes* e non hanno figli.

Facciamo un esempio per chiarire meglio quanto detto. L'esempio classico è quello di uno scenegraph che costruisce un'automobile.

In uno scenegraph classico la radice del grafo è un nodo, che definisce la geometria della carrozzeria della macchina; dopodichè abbiamo quattro nodi, figli del nodo radice, che applicano una trasformazione e definiscono dove collocare le quattro ruote; infine c'è un ulteriore nodo, che definisce la geometria della ruota, figlio di ciascuno dei quattro precedenti e che ha perciò più padri.

Tutti gli scenegraph diversi da OpenSG sono perciò "multiparent" (vedi figura 1.6a).

OpenSg, essendo invece uno Scenegraph "singleparent", traduce quanto

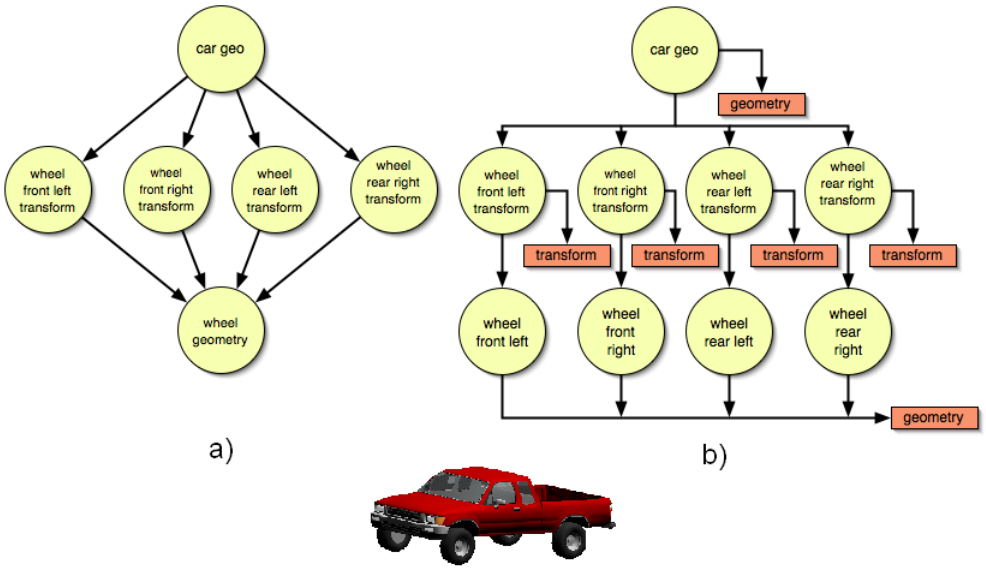


Figura 1.6: *Esempio di uno scenegraph di un'automobile con nodi multiparent (a) e singleparent in OpenSG (b).*

sopra in un grafo strutturato in nove *nodes* e quattro *cores*: i *nodes* sono identici tra loro e ciò che li differenzia è solo il *core* che è loro collegato.

Avremo quindi la radice iniziale (car geo) con il relativo *core* (geometry), che definisce la geometria della carrozzeria della macchina; i quattro *nodes* successivi con relativi *cores* trasformazione per il posizionamento delle ruote e ad ognuno di essi collegato un *node*; questi ultimi quattro *nodes*, condividono un *core* che definisce la geometria (vedi figura 1.6b).

Come vediamo dalla figura 1.6b i *cores* rispecchiano i nodi di uno scenegraph classico (vedi figura 1.6a) ed infatti ce ne sono tanti quanti nodi c'erano nella struttura precedente. Non ci sono *nodes* che hanno più padri, ma la caratteristica più importante è che i soli *nodes* definiscono non un grafo ma un albero, che per sua struttura risulta molto più facile da visualizzare

graficamente nell'editor (ad esempio mediante una TreeView).

1.5.2 Window, Viewport e Camera

OpenSG oltre a creare uno scenegraph mette a disposizione delle Windows e delle Viewports.

Una Window non è altro che una finestra nella quale è possibile renderizzare una scena; la Viewport, invece, delimita l'area di rendering all'interno della Window. Una Window può avere una o più Viewport.

Le Windows messe a disposizione da OpenSg sono di tipo diverso. Alcune (GLUTWindow, QTWindow, XWindow e Win32Window) servono per integrare OpenSG con i sistemi di Windowing:(ad esempio Win32Window alla piattaforma Microsoft) e creano un contesto OpenGL.

La PassiveWindow, al contrario delle altre, non crea il contesto OpenGL e può essere quindi facilmente integrata in ogni Window e sistema GUI che supporta OpenGL: in SL3D è stata appunto utilizzata una PassiveWindow per integrare OpenSG in una Window QT (GLWidget).

In ogni viewport e' possibile definire:

- una camera;
- un background;
- un puntatore a un nodo dal quale iniziare l'azione di rendering;
- uno o piu' foreground per l'aggiunta di loghi, immagini, ...

Una camera definisce cosa è visibile di una scena e cosa no, e quindi cosa deve essere renderizzato e cosa no. In OpenSg, diversamente da altri Scenegraph, oltre i parametri near e far, che rappresentano i piani che delimitano il volume di vista della camera, è necessario specificare il campo beacon, che

è un puntatore ad un nodo nello Scenegraph: la camera per quanto riguarda posizione ed orientamento segue le trasformazioni di tale nodo.
In OpenSg sono presenti diversi tipi di camera:

- PerspectiveCamera;
- MatrixCamera;
- OrthoCamera;

E' possibile definire il background di una Viewport attraverso colori, SolidBackground e GradientBackground, attraverso immagini, ImageBackground, oppure attraverso un PassiveBackground, che praticamente non renderizza nessun background e dà la possibilità di sovrapporre Viewports, generando un'immagine combinata.

Tra i vari Foregrounds, che consentono di inserire loghi ed immagini, sia 2D (ImageForeground) che 3D (PolygonForeground), troviamo anche i GrabForeground, che sono usati per catturare delle immagini della scena che viene renderizzata (consentendo ad esempio di creare dei filmati).

Per avere un'idea generale di come Windows, Viewports e camere si collegano allo Scenegraph e di come in OpenSG viene eseguito il rendering possiamo analizzare la figura 1.7.

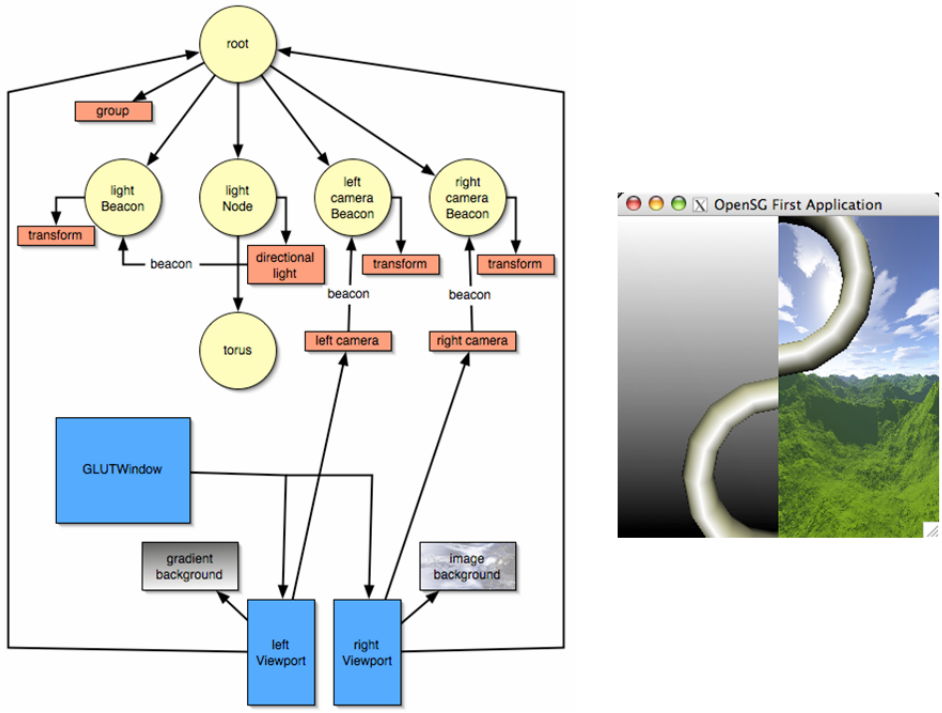


Figura 1.7: *Collegamento tra scenegraph e window, viewport e camera. Nella figura abbiamo una window con due viewports, la prima con un Gradient Background, la seconda con un ImageForeground. Entrambe le Viewports renderizzano l'intera scena che è composta da un toroide illuminato ma con camere differenti.*

Per il rendering della scena OpenSG partendo dalla Window scorre tutte le Viewports e per ognuna di esse:

1. disegna il background;
2. imposta tutti i parametri della camera;
3. esegue la visita del grafo che ha come radice il nodo beacon della Viewport;
4. renderizza eventuali foreground;

5. infine passa alla Viewport successiva.

1.5.3 I FieldContainers

I FieldContainers rappresentano gli elementi fondanti di OpenSG, dai quali scaturiscono tutte quelle caratteristiche che lo hanno fatto preferire ad altri Scenegraph nell'implementazione di SL3D. Quasi tutti gli oggetti OpenSg (node, core, ...) sono delle sottoclassi della classe FieldContainer.

I FieldContainer (d'ora in poi FC) sono, come dice il nome, dei contenitori di campi (Fields). I dati contenuti nei Field Container non vengono immagazzinati dentro semplici variabili ma dentro istanze derivate dalla classe Field.

Ogni FC ha una struttura descrittiva di sè stesso, dalla quale è possibile ricavare informazioni sia sul FC che su tutti i suoi Fields. Le informazioni sul FC riguardano la classe del FieldContainer, da quali classi è derivato, l' Id del FC (che è un identificatore assegnato al momento della creazione del FC). Ancora più interessanti sono le informazioni che è possibile ricavare sui Fields e che sono il numero dei Fields che il FC contiene, il loro nome (per stringa o per indice), il loro tipo ed il valore di ogni Fields (anche qui per stringa o per indice).

Tutto ciò consente un totale e generico accesso a qualunque tipo di FC conosciuto dal sistema solo conoscendone il nome o tramite un puntatore all'istanza e permette di scrivere programmi che trattano qualunque oggetto di OpenSG a livello di FieldContainer, astraendo dal tipo concreto dell'oggetto; può essere utilizzato per creare generiche operazioni di input/output e generiche interfacce utenti, come vedremo per SL3D.

Inoltre OpenSG mette a disposizione la possibilità di creare istanze di FieldContainer semplicemente attraverso il nome del FieldContainer.

Ad esempio per creare un *node* invece che farlo esplicitamente attraverso la funzione "create()" del *node*

```
Node::create();
```

è possibile farlo passando alla createContainer della classe FieldContainerFactory il nome della classe dell'oggetto che intendiamo creare, quindi "Node"

```
FieldContainerFactory::the().createContainer(‘‘Node’’);
```

I Field contenuti nel FC possono essere di tipo "single" (SField) o "multi'(MField):

- I SFields contengono un singolo valore;
- I MFields sono comparabili a vettori STL (array a dimensione dinamica).

E' possibile in OpenSG creare, grazie anche all'editor grafico fornito, nuovi tipi di FieldContainer con la possibilità quindi di estendere OpenSG.

Alla creazione di un nuovo FC vengono specificati quelli che sono i suoi Fields e nel far questo bisogna anche indicare per ognuno di essi se sono External o Internal.

I field External contengono le informazioni necessarie per ricreare l' FC. Quelli Internal vengono utilizzati solo internamente all' FC.

Come già detto, OpenSg è stato pensato per gestire dati in multithreaded; anche qui sono i FC a consentire tale operatività. Per far questo OpenSG mantiene una changeList (una lista dei cambiamenti) per ogni Thread nella

quale inserisce le modifiche apportate ai Field dell' FC e al momento opportuno esegue la sincronizzazione delle copie che ogni singolo Thread ha dei FC.

Conclusione

In questo capitolo abbiamo dedicato la prima parte all'analisi dei software disponibili, mettendone in risalto le caratteristiche ma anche l'inadeguatezza nel presentare al meglio lavori di grafica 3D in modo semplice. Manca, infatti, un software che integri uno strumento per presentazioni ad un viewer 3D, con un minimo di funzioni per l'editing di una scena 3D.

Successivamente, abbiamo analizzato Virtual Inspector e il suo campo applicativo, mettendo in risalto alcune sue caratteristiche che sono servite da spunto nella realizzazione di SL3D.

Infine abbiamo dedicato un'ampia sezione ad OpenSG che è l'elemento sul quale SL3D si fonda, assorbendone un po' tutte le caratteristiche come la portabilita e l'estendibilità.

A questo punto dovremmo avere ben chiaro in mente quella che è l'idea alla base di SL3D e la sua collocazione tra le varie tipologie di software.

Nel prossimo capitolo ne analizzeremo le funzionalità e le caratteristiche dal punto di vista utente.

Capitolo 2

Descrizione del sistema dal punto di vista utente

In questo capitolo analizzeremo come SL3D si presenta all'utente finale.

La presentazione può essere visualizzata in *Fullscreen Mode*, che rappresenta la modalità con la quale le slides verranno presentate al pubblico, oppure in *Edit Mode*, che è la modalità utilizzata in fase di editing della presentazione.

Ci occuperemo nella prima parte di descrivere l'interfaccia utente in *Edit Mode*; presenteremo poi i principali elementi (nodi) di cui si compone una slide, ed infine seguiremo passo per passo la costruzione di una slide.

2.1 L'interfaccia

L'idea iniziale di SL3D era quella di dare vita ad un motore di rendering di presentazioni tridimensionali da costruire editando un file xml, così come avviene in Inspector.

In corso d'opera abbiamo invece deciso di dotare SL3D di un editor grafico per

due motivi: il primo era che editare anche una semplice slide diventava complesso e generava un file xml piuttosto lungo tale da rendere il software poco usabile; l'altro era che OpenSG, come abbiamo visto nel precedente capitolo, basandosi sui FieldContainer, consente di costruire generiche interfacce utenti.

L'interfaccia grafica di SL3D è realizzata con le Qt 4.1.3 [2], che sono un toolkit multipiattaforma per lo sviluppo di applicazioni e costruzione di interfacce grafiche.
Sono state scelte perché, come OpenSG, sono open source e rendono l'applicazione portabile sui principali sistemi operativi (Microsoft Windows, Mac OS X, Linux).

L'interfaccia è composta da (vedi fig. 2.1):

- una MenùBar, immancabile in ogni interfaccia grafica, dalla quale è possibile accedere a tutte le funzionalità di SL3D;
- una ToolBar, che contiene un sottoinsieme di funzionalità di più frequente utilizzo, per un più rapido accesso alle stesse;
- una StatusBar, che si trova in basso e dà indicazioni sulle operazioni di volta in volta compiute;
- una finestra (Widget) centrale, composta da quattro viste, che visualizzano tutte le informazioni sulla presentazione.

Le quattro viste di cui si compone la Widget centrale sono:

- la Slide View;
- la Property View ed Editor View;
- la Render Area.

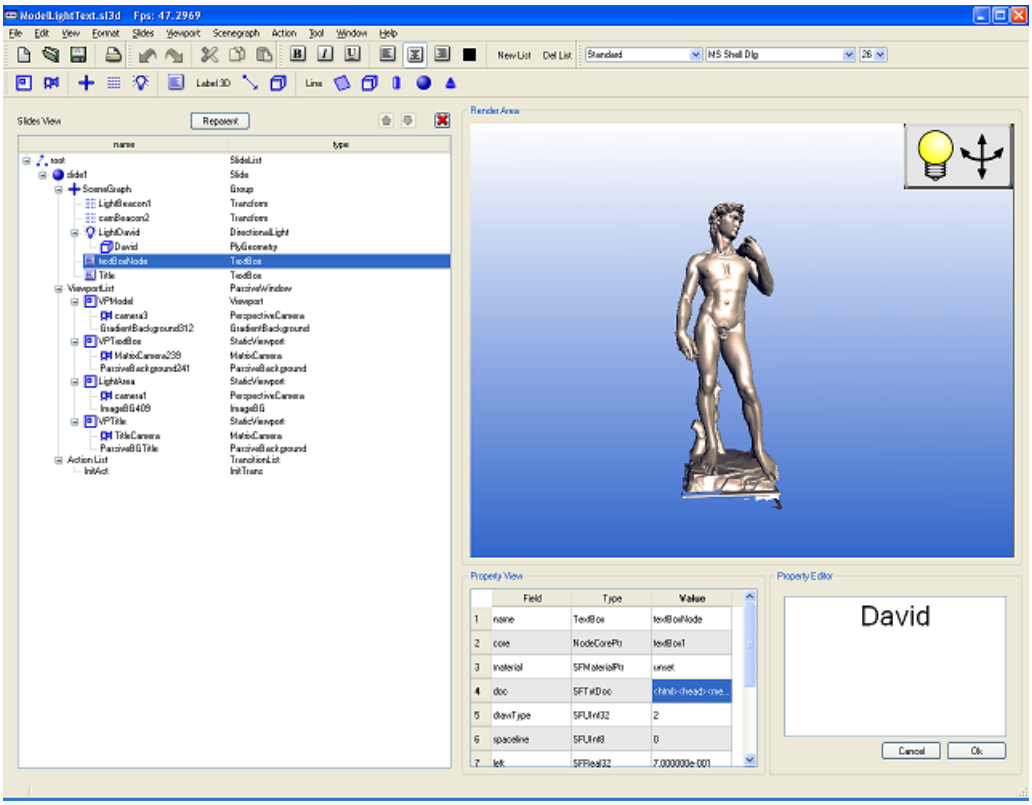


Figura 2.1: *Interfaccia utente di SL3D.*

2.1.1 La Slide View

Nella Slide View viene visualizzata la lista delle slides che compongono la presentazione e la struttura di ognuna di esse.

E' divisa in due colonne: in quella di sinistra (name) sono presenti gli oggetti (i FieldContainers) con i nomi da noi assegnati ed è visualizzata mediante una struttura ad albero, modello directory (Tree View); in quella di destra (type) sono indicati i tipi dei singoli oggetti (i tipi dei FieldContainer).

Ogni slide ha una sua struttura ben definita (vedi fig. 2.2) e composta da:

- uno Scenegraph che contiene la scena da renderizzare nella slide;

- una ViewportList che definisce il layout della presentazione tramite una lista di Viewport;
- una TransitionList che è l'elenco delle transizioni che verranno eseguite durante la visualizzazione della slide.

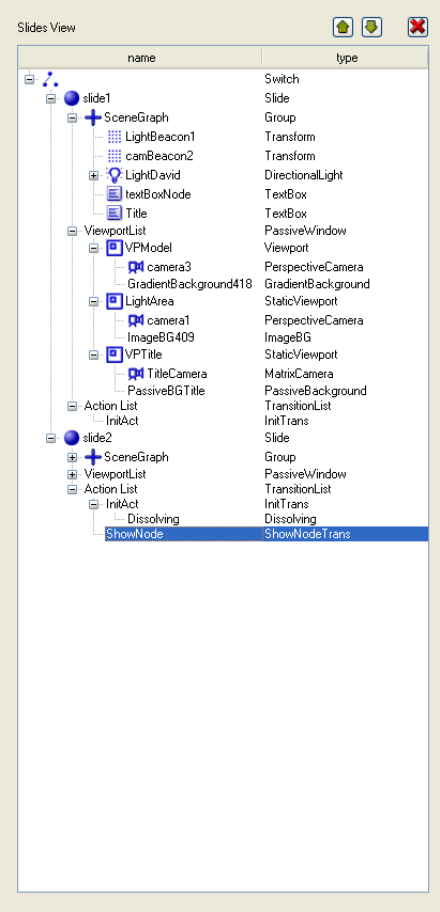


Figura 2.2: *Slide View di SL3D dove possiamo vedere una lista di slides e per ognuna di esse la struttura data dallo Scenegraph, VieportList e TransitionList.*

La Slide View consente di selezionare le slides da visualizzare e gli oggetti da editare; è anche possibile cancellare o cambiare l'ordine degli oggetti,

mediante i pulsanti posti nell'angolo in alto a destra.
L'editing delle proprietà di un oggetto avviene nella Property View ed Editor View.

2.1.2 Property View ed Editor View

La Property View visualizza i campi (Fields) dell'oggetto correntemente selezionato nella Slide View.
Si presenta come una tabella a tre colonne che visualizzano rispettivamente:

Field i nomi dei campi;

Type il tipo;

Value il valore.

La Property View permette l'editing del valore dei campi o direttamente nella cella della colonna "Value" o, se di contenuto più complesso, nella Property Editor (vedi fig. 2.3).

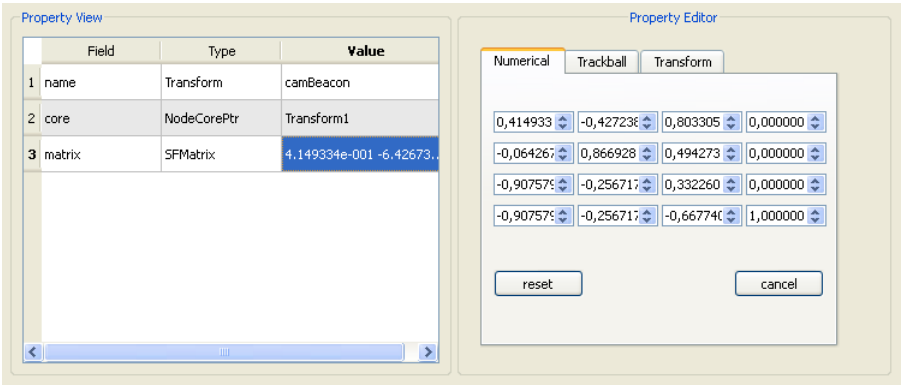


Figura 2.3: *Property View ed Editor View in fase di editing di una matrice.*

2.1.3 Render Area

La Render Area dà un feedback visivo nella fase di editing della Slide, rispettando le proporzioni della risoluzione finale in Fullscreen.
E' possibile interagire nella Render Area, modificando l'inquadratura della camera delle varie Viewport tramite una trackball; in futuro potrebbe essere possibile semplificare ancora di più l'editing della slide aumentando le possibilità di interazione nella Render Area (ad esempio definire il layout della slide con il mouse, selezionare i vari oggetti sui quali eseguire operazione di "copia ed incolla",. . .)

2.2 I Principali nodi che compongono una slide

In questo paragrafo elencheremo una serie di nodi che possono andare a comporre lo Scenegraph di una slide in SL3D.
Alcuni sono propri di OpenSG, altri sono stati creati appositamente per SL3D.
In questa sede ci limiteremo a descrivere a cosa servono e come vengono utilizzati, mentre nel successivo capitolo (cap. III) ne analizzeremo i più interessanti dal punto di vista implementativo. Prima di vedere nello specifico le funzionalità dei vari nodi è bene tenere presenti alcune regole generali sui nodi.

1. L'effetto di ogni nodo (trasformazione, illuminazione, animazione,. . .) ha influenza solamente sui figli, ma non sui nodi fratelli (vedi fig 2.4).

2. Per inserire lo stesso nodo due volte in una slide, non è necessario crearlo due volte, ma è sufficiente che i due *nodes* condividano lo stesso *core*

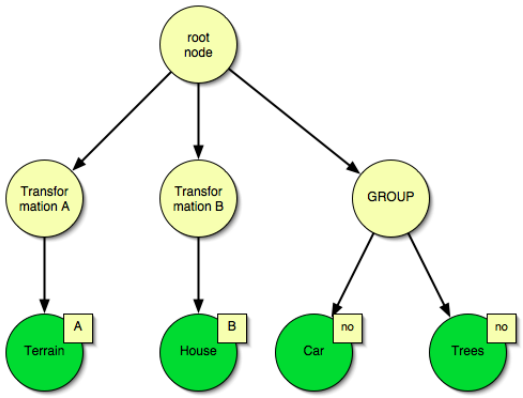


Figura 2.4: *Possiamo vedere in questa figura come il nodo Terrain subisce la trasformazione del nodo A, House la trasformazione B mentre Car e Trees non ne hanno nessuna.*

(basta modificare il campo *core* del *node* come in fig. 1.6 a pag. 12).

SL3D mantiene la divisione operata da OpenSG tra:

Nodi Group : sono i nodi interni al grafo e possono avere figli;

Nodi Drawable : sono i nodi "foglia" del grafo e definiscono oggetti geometrici.

2.2.1 I Nodi Group

I nodi Group si distinguono in:

Group : ha la sola funzione di raggruppare dei nodi. Renderizzare un nodo group equivale a renderizzare i suoi figli.

Switch : influenza la visita del grafo, limitandola a uno, tutti o a nessuno dei suoi figli, settando il campo "choice".

Transform : è uno dei più importanti e serve per spostare i vari oggetti nella scena (geometrie, camere, luci, ...). Ha un solo campo che è "matrix" dove conserva la matrice di trasformazione.

ComponentTransform : non è altro che un nodo "Transform", che però conserva le singole trasformazioni (rotazione, scalatura, ...) che vanno a comporre il valore del campo "matrix" del nodo Transform secondo questa formula:

```
M = Translation * Center * Rotation * ScaleOrientation *
   Scale * -ScaleOrientation * -Center;
```

AnimTransform : serve per creare delle scene animate ed applica delle rotazioni lungo gli assi x,y,z.

DistanceLOD : consente di visualizzare uno stesso modello con differenti livelli di dettaglio a seconda della distanza dell'oggetto rispetto all'osservatore (la camera) come in fig. 2.5.

Billboard : fa in modo che i figli dell'oggetto mostrino sempre la stessa faccia all'osservatore. Può essere ad esempio usato in modo che il testo sia sempre leggibile in una scena animata;

Light : ci sono tre differenti modelli di illuminazione:

DirectionalLight : è una luce direzionale, tipo la luce solare. E' la meno pesante da calcolare;

PointLight : è una luce puntiforme che emana luce in tutte le direzioni;

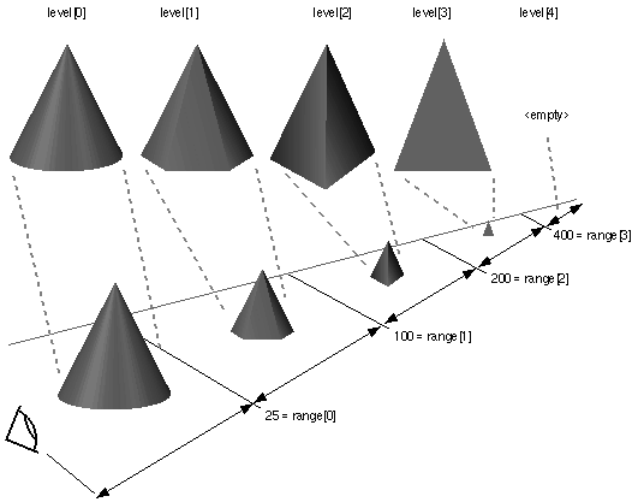


Figura 2.5: *Il cono nella figura viene renderizzato a seconda della distanza dall'osservatore, con diffente risoluzione fino a diventare un semplice triangolo al livello 3 e scomparire del tutto al livello 4.*

SpotLight : è una luce che ha un campo di illuminazione ristretto, come può essere ad esempio una torcia. E' la più dispendiosa da calcolare.

Per le luci vale quanto detto per le camere nel precedente capitolo: posizione ed orientamento della sorgente luminosa sono definiti dal *node* specificato nel campo "beacon" della luce. La figura 2.6 mostra un semplice grafo con una sorgente luminosa, con il campo "beacon" che punta ad un nodo trasformazione. In questo modo è possibile applicare le trasformazioni ad una luce modificando il nodo beacon.

2.2.2 I Nodi Drawable

Si distinguono in:

PlyGeometry : consente di caricare mesh 3D da files .ply, settando il

campo “FileName” e specificando la modalità di rendering nel campo “DMode” con un valore da 1 a 6;

ImportModel : consente di importare mesh 3D in diversi formati già supportati da OpenSG (.3ds, .wrl, .osg, .bin, . . .);

Axes3D : disegna gli assi del sistema di riferimento;

Connector : disegna un connettore in 3d dal punto “PntSource” del nodo “Source” al punto “PntDest” del nodo “Dest”. È anche possibile definire il colore del connettore.;

Label3D : permette l’inserimento di testo 3D anche su più linee, con differenti font, allineamenti, dimensioni e di controllare la larghezza del testo;

TextBox : questo nodo è senza dubbio il più utilizzato e consente di inserire testo formattato e liste, definendo margini (top, bottom, left e right) o in percentuale rispetto alla viewport nella quale viene visualizzato (valore compreso tra 0 e 1) o in pixel (valore maggiore di 1). La modi-

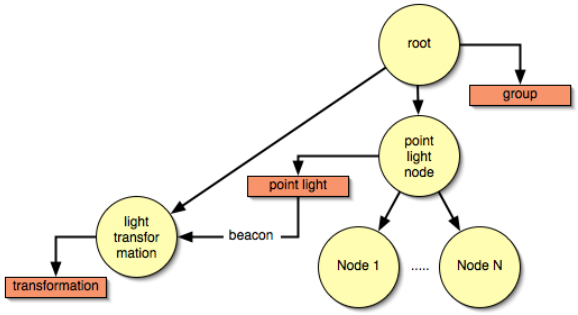


Figura 2.6: *Il nodo LightTrasformation definisce le trasformazioni della luce.*

fica del testo avviene editando il campo "TxtDoc" attraverso l'editor testuale che verra aperto e utilizzando la toolbar delle formattazioni;

Image3D : disegna un'immagine in 3D ;

Figure geometriche regolari quali linee, rettangoli, cubi, sfere, coni e cilindri.

L'editor di SL3D consente, attraverso la funzione "Add new node" del menù Scenegraph di inserire ogni altro nodo proprio di OpenSG specificando il nome del nodo.
In questo caso è però possibile che alcuni dei campi (Fields) del nuovo nodo non risultino editabili perché l'editor non è stato ancora sviluppato per riconoscerne i tipi. Questo argomento sarà oggetto di trattazione più ampia nel capitolo seguente, dove illustreremo come è stato implementato l'editing dei campi.

2.3 Come si costruisce una slide

Per facilitare la costruzione di una presentazione, SL3D all'apertura ci consente di scegliere un tipo/modello di slide già strutturato ed in linea con lo stile di presentazione che vogliamo proporre, tramite un dialog (vedi figura 2.7).

E' anche possibile costruire nuovi modelli e modificarli in corso d'opera.
In alternativa l'utente può costruire la slide in maniera autonoma ed in questo caso il sistema gli propone una "blank slide', composta da

- **Scenegraph** vuoto;
- **ViewportList** con una Viewport vuota;

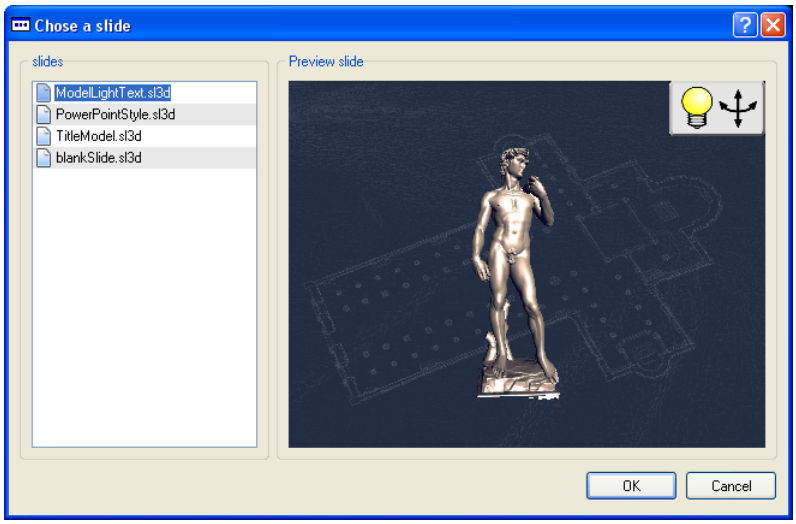


Figura 2.7: *Dialog per la scelta del modello della slide.*

- **TransitionList** con una "InitAction", entrambe vuote;

sulla quale lavorare.

In questo paragrafo descriveremo come è possibile costruire una slide partendo proprio da tale slide vuota.
Il primo passo per progettare una slide ben fatta e ridurre inutili e frustranti perdite di tempo è avere ben chiaro ciò che si intende realizzare.
Successivamente sarà necessario definire il layout della slide, dato dall'insieme delle Viewports che la costituiscono.

E' consigliabile inserire una Viewport che occupi tutta l'area di rendering, specificando quindi i campi "top", "bottom", "left" e "right" rispettivamente con i valori 1, 0, 0 ed 1, da utilizzare per lo sfondo, che potrà essere uno di quelli già elencati nella trattazione di OpenSG e che troveremo elencati nel menù "Viewport" sotto la voce "Background"; nelle Viewports successive, che serviranno per posizionare gli oggetti nella scena, utilizzeremo invece un PassiveBackground.

L'inserimento delle Viewport avviene selezionando la ViewportList della slide e scegliendo una delle Viewport nella voce "Viewport" del menù.

Oltre al background, per ogni Viewport dobbiamo specificare:

- uno o più foreground opzionali (per inserire loghi, immagini, ...);
- una camera: dovranno essere settati i campi "near", "far", "fov" ed "aspect" oltre al campo "beacon", come già specificato nel capitolo relativo ad OpenSG;
- un *node* radice.

Il *node* radice deve essere creato nello Scenegraph e rappresenta la radice del grafo della scena che stiamo editando e che visualizzeremo nella singola Viewport.
Generalmente lo Scenegraph conterrà tanti *nodes* radice quante sono le Viewports, anche se ciò non toglie che una Viewport possa renderizzare in tutto o in parte la stessa scena di un'altra (ad esempio con inquadrature differenti, come fa Inspector).
Il node radice appartiene alla categoria Group e deve essere assegnato al campo "root" della Viewport in questione, operando nella Property View.

Una volta collegato il node radice avremo nella Render Area un feedback visivo man mano che andremo ad editare la scena.

Ora andremo a creare lo Scenegraph della nostra scena con i nodi che abbiamo precedentemente elencato e secondo le regole per l'inserimento dei nodi già trattate.

Tutti i nodi, suddivisi nelle due categorie, li potremo trovare nella voce Scenegraph del Menù.

Gli ultimi due aspetti da definire sono le transizioni che avvengono al-

l'interno della slide e gli effetti di passaggio tra le slides stesse, da inserire entrambi nella TransitionList.

2.4 Le transizioni

Le transizioni all'interno della slide sono le operazioni eseguite durante la presentazione, in modalità *Fullscreen*, su input dell'utente.
A queste fa eccezione la InitAction che è sempre presente nella TransitionList e che contiene tutte le transizioni che inizializzano la slide all'apertura.

Le transizioni, un po' come avviene per i *nodes*, possono essere:

single che creano il singolo effetto;

group il cui effetto è dato dalla somma degli effetti delle transizioni che contiene.

Gli effetti "single"che sono stati implementati in SL3D sono:

Set : setta i campi di un qualunque FieldContainer. Ha quattro parametri: FC1, Field1, FC2, Val2. Ad esempio per settare il campo "far" della Camera1 con un valore di 100 possiamo assegnare ai campi questi valori:

```
FC1    = Camera1
field1 = far
FC2    = unset
val2   = 100
```

oppure per far sì che Camera1 e Camera2 abbiano lo stesso far:

```
FC1    = Camera1
field1 = far
FC2    = Camera2
val2   = far
```

Sleep : introduce un ritardo in millisecondi. Ha un solo campo, "sleep". Serve per introdurre azioni temporizzate;

ShowNode : mostra o nasconde il nodo specificato nel campo "node" a seconda che il valore del campo "show" sia TRUE o FALSE;

ShowText : consente di mostrare o nascondere linee di testo di una textbox specificata nel campo "node"; nel campo "cmd" inseriremo l'operazione da eseguire con la seguente sintassi :

- "show l1-l2": fa comparire dalla linea l1 alla linea l2;
- "hide l1-l2": nasconde dalla linea l1 alla linea l2
- "none": nasconde tutte le linee della TextBox;
- "all": mostra tutte le linee della TextBox.

Tra gli effetti group in SL3D ci sono:

group : esegue le azioni figlie contemporanemente e termina quando sono state eseguite tutte;

Sequence : esegue le azioni figlie in sequenza;

Sleep_seq : esegue le azioni figlie, inserendo un ritardo tra l'una e l'altra, specificato in millisecondi nel campo "sleep".

Gli effetti di passaggio tra slides in SL3D costituiscono anch'esse delle transizioni; devono essere infatti inserite nella InitAction come una normale operazione di inizializzazione di una slide.
In particolare, l'effetto di passaggio tra la slide1 e la slide2 va inserito in fondo alla InitAction della Slide2.
Gli effetti di passaggio finora implementati in SL3D sono:

ScreenDown : simula la "caduta" della schermata della slide1;

Dissolving : dissolve l'immagine della slide1 facendo comparire progressivamente quella della slide2;

Sliding : fa scorrere e scomparire la schermata della slide1 nelle quattro direzioni a seconda che il campo "side" sia 0, 1, 2 o 3.

Questi effetti si trovano nella voce "Slide Transition" del menù "Slide".

Capitolo 3

Architettura del sistema

3.1 Considerazioni generali sulle scelte implementative

Prima di entrare nel vivo della descrizione dell'architettura del sistema e di quelle che sono state le principali scelte implementative riepiloghiamo gli strumenti che abbiamo utilizzato, scelti tra le librerie Open Source disponibili, al fine di rendere l'applicazione portabile sui principali sistemi operativi:

- Qt 4.1.3 per la realizzazione dell'interfaccia grafica dell'editor di SL3D, ma anche per la realizzazione di alcuni nodi, come ad esempio la TextBox;
- OpenGL per il rendering grafico;
- Xml come formato per il salvataggio dei files, che li rende editabili anche con un normale editor di testo;
- FTGL[6] e FreeType2[5] per il rendering del testo formattato;
- OpenSG per lo Scenegraph ma anche come elemento fondante di tutta l'applicazione.

Nell'effettuare le scelte implementative che hanno caratterizzato la realizzazione di SL3D abbiamo dovuto tenere presenti alcuni aspetti.

Una slide solitamente non è popolata da un gran numero di oggetti, ma da pochi, tra i quali i principali sono modelli 3D (mesh), testo ed immagini.

I modelli 3D possono essere molto complessi e con tempi lunghi di caricamento, che impegnano pesantemente il sistema; allo stesso modo il testo in un contesto OpenGL richiede un grosso impegno, sia in termini di performance di rendering, che in termini di spazio (ad ogni uso di un font differente, come Arial o Courier, vengono create dal sistema le immagini dei 256 caratteri di cui sono composti e che verranno utilizzati per comporre il testo da renderizzare).

Per alleggerire il sistema si è dunque deciso di far condividere alle slides le meshes, i fonts e le textures di immagini, usate ad esempio come sfondo.

Un altro aspetto che abbiamo dovuto prendere in considerazione nella strutturazione della presentazione e delle relazioni tra slide è il fatto che le animazioni durante la presentazione e le interazioni che il sistema consente all'utente mediante la trackball comportano modifiche alla slide stessa non determinabili a priori.

Ciò ci ha indotto, innanzitutto, a rendere le slides indipendenti tra loro, onde evitare che modifiche effettuate in fase di presentazione su oggetti condivisi tra più slides si ripercuotessero su tutte; in secondo luogo ci ha imposto di mettere a punto un sistema di transizioni non reversibili, come vedremo nel prosieguo del capitolo.

L'indipendenza tra slides, oltre a risolvere il problema delle interazioni, ha anche reso l'implementazione del sistema più semplice, con un notevole risparmio di tempo.

3.2 La struttura di una presentazione

La presentazione è un nodo Switch, che indica quale slide deve essere correntemente disegnata e qual è la slide corrente che stiamo editando o guardando, a seconda che siamo nelle due modalità.

Il nodo switch ha tanti nodi figli quante sono le slides della presentazione. Questi nodi sono dei nodi di tipo Slide; la Slide è un *core* di tipo group, al quale sono stati aggiunti quattro campi (fields):

- "title" di tipo stringa, che contiene il titolo della slide e che verrà visualizzato nella SlideView;

- "scene", che è un puntatore ad un *node* che conterrà lo scenegraph della slide;

- "window", che è un puntatore ad una Window, che nel nostro caso è sempre una PassiveWindow per l'integrazione con Qt. dove inseriremo tutte le Viewports;

- "transList", che è un puntatore ad un Transition, più precisamente ad un TransitionList, che è un nuovo FC implementato in SL3D che contiene la lista delle transizioni tra le quali c'è sempre la InitAction.

Ogni Slide contiene anche l'Xml che descrive la propria struttura iniziale all'atto del caricamento attraverso il Loader. Questa informazione viene utilizzata durante la fase di presentazione per compiere passi indietro e annullare le modifiche che la slide subisce per effetto delle transizioni.

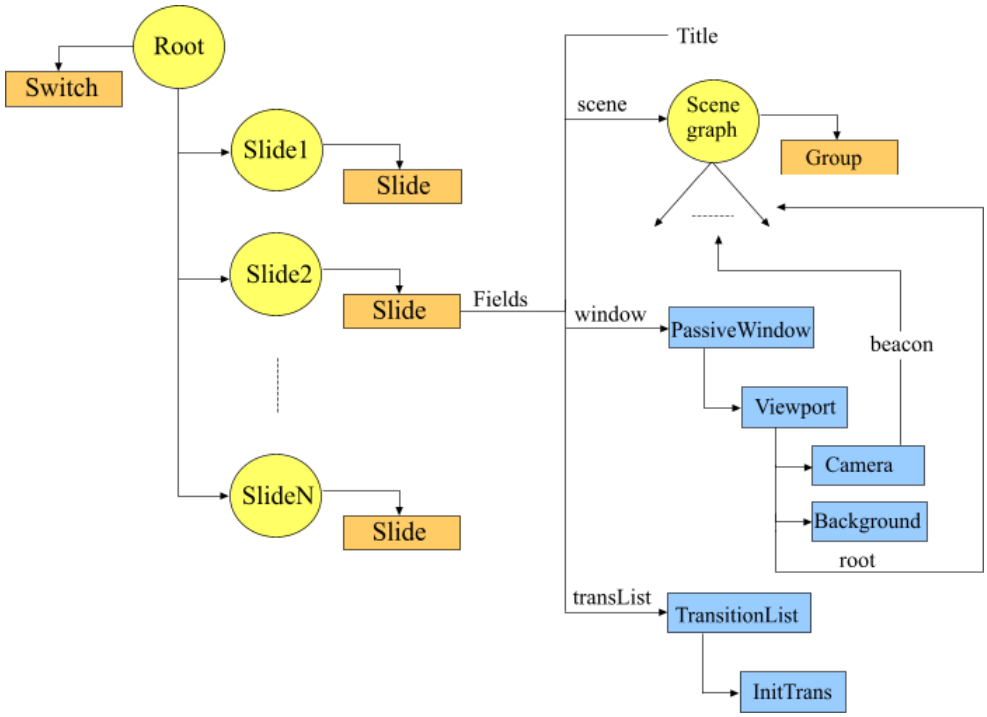


Figura 3.1: *Schema di una presentazione.*

3.3 L'applicazione

SL3D è un'unica applicazione che unisce in se l'Editor e il Viewer ed opera a seconda dei casi in modalità "EditMode" o "Fullscreen".

All'apertura si presenta nella modalità "EditMode", con l'interfaccia che abbiamo descritto nel capitolo II.

Al passaggio alla modalità "Fullscreen" viene creata una nuova finestra, speculare alla Render Area e che occupa tutto lo schermo; tramite il Writer viene creato l'Xml di tutta la presentazione, che viene passato al Loader che ne crea una copia.

La creazione di tale copia è resa necessaria dal fatto che nella modalità "Fullscreen" vengono abilitate le transizioni che possono apportare modifiche

alla presentazione durante la visualizzazione.

All'uscita dalla modalità "Fullscreen" la copia viene distrutta.

3.4 L'Editor.

Come abbiamo già anticipato nel primo capitolo, l'Editor, nelle sue due viste principali (SlideView e PropertyView), opera sulla base dei FieldContainers (FC).

Entrambe le View sono state costruite secondo l'architettura "Model-View and Delegate", introdotta nella release 4 delle Qt.
Questa architettura consente di gestire le relazioni tra dati (in questo caso FC) ed il modo in cui vengono presentati all'utente.
Consiste di tre tipi di oggetti:

- il *Model* comunica con una sorgente di dati fornendo un'interfaccia tra le altre componenti nell'architettura; la natura della comunicazione dipende dal tipo dei dati e dal modo in cui il Model è stato implementato;

- la *View* ottiene dal Model dei "model indexes" che sono riferimenti ai dati; (vedi guida Qt)

- *Delegate* si frappone tra la View ed il Model e specifica alla View come visualizzare i dati e come editarli.

Qt mette a disposizione dei model generici, adattabili alle più comuni viste, rappresentate dalle ListView per creare degli elenchi, dalle TableView per delle tabelle e dalle TreeView per creare delle strutture ad albero (ad esempio per visualizzare delle Directory) e sui quali basarsi per costruire il proprio specifico model.

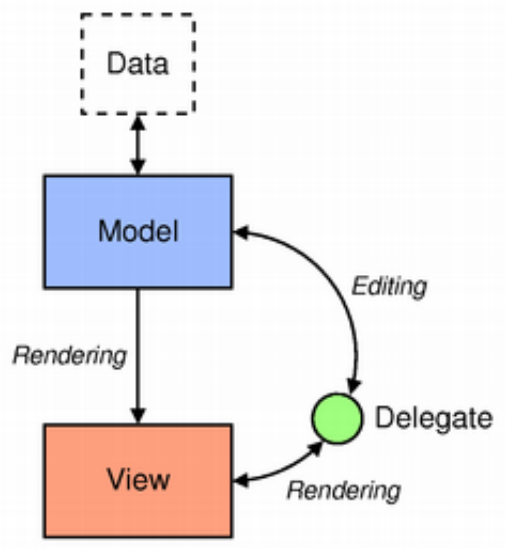


Figura 3.2: *Architettura Model-View and Delegate.*

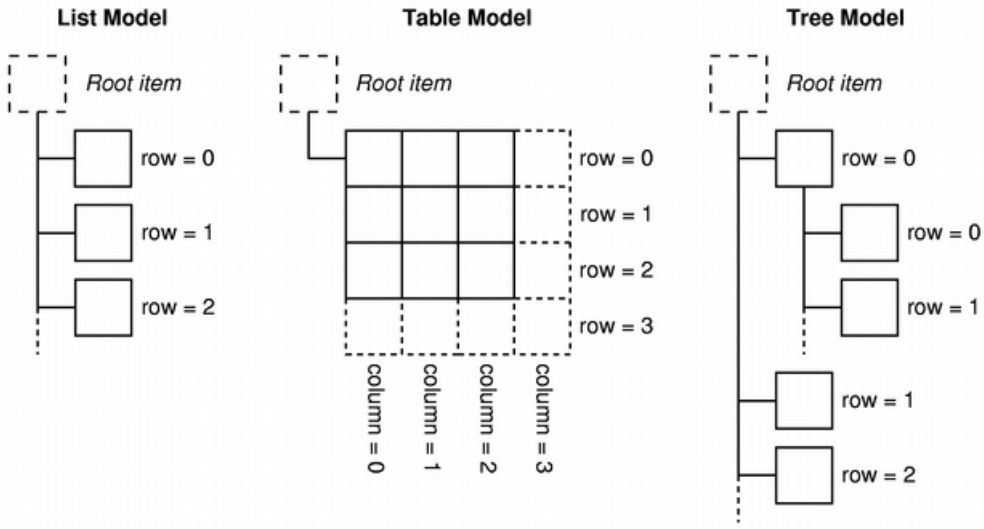


Figura 3.3: *Models per ListView, TableView e TreeView.*

3.4.1 La SlideView

Più in particolare, nella costruzione della SlideView è stato realizzato un TreeModel a partire da un QAbstractItemModel, nel quale ogni "mod-

el indexes" riferisce un *AttachmentContainer*, che è un particolare tipo di FieldContainer, così chiamato perché ad esso è possibile assegnare degli attachment, tra i quali il nome.

Rientrano negli AttachementContainers tutti i FC più "significativi", come i *nodes*, i *cores*, le camere, le windows, le viewports, ecc. (vedi fig. 3.4).

Per la View è stata utilizzata una TreeView, che era già fornita da Qt.

Non è stato necessario creare dei Delegate in quanto la SlideView non consente operazioni di editing sulle informazioni visualizzate, che avvengono attraverso la PropertyView.
Questo non vuol dire che non vengano utilizzati dei delegate, ma semplicemente che vengono utilizzati quelli di default di Qt.

3.4.2 La PropertyView

Nella costruzione della PropertyView, dove, come abbiamo già visto, vengono visualizzate le informazioni sui campi (i Fields) di un FC per poterle editare, abbiamo creato un model partendo da un TableModel, nel quale ogni "model indexes" riferisce un Field del FC.

A questo punto è necessario fare un'ulteriore precisazione sui Fields di un FC.
Alla creazione di un nuovo FC è possibile in OpenSG specificare per ogni Fields se trattasi di un Field external o internal.
I Fields dichiarati external vanno a formare i parametri che l'editor considera "editabili" e che vengono quindi visualizzati nella PropertyView, a condizione che sia stato sviluppato un Delegate per quello specifico tipo di Field.
L'insieme dei Fields external di un FC va a costituire le informazioni necessarie per poter ricreare il nodo, che devono quindi essere scritte nel file di

salvataggio della presentazione.

I Fields internal contengono le informazioni utili al nodo per lo svolgimento delle sue funzioni, utilizzati quindi dal programmatore all'atto della definizione del FC.

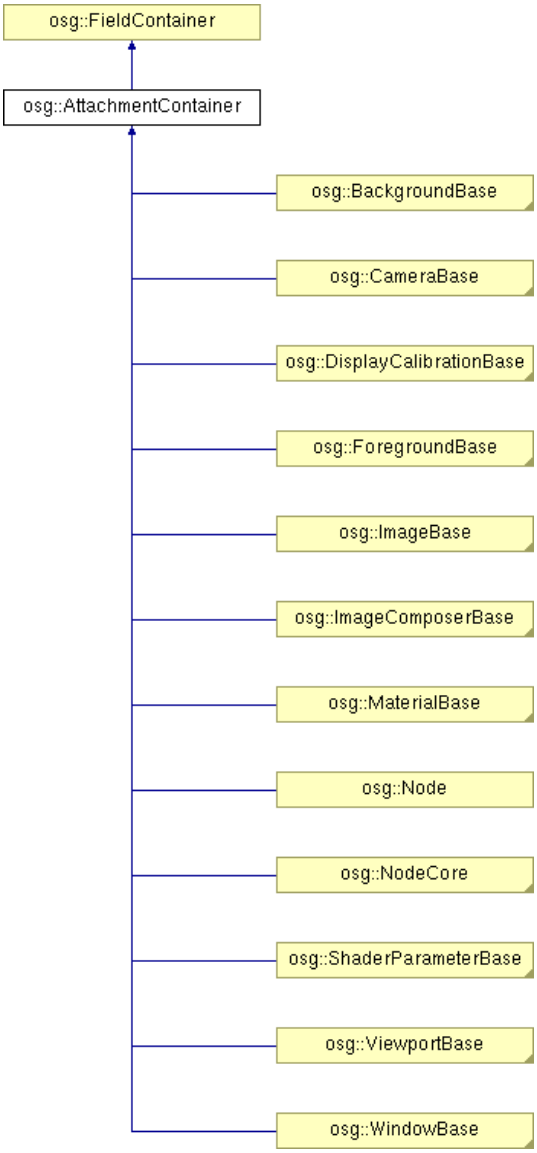


Figura 3.4: *Gli AttachmentContainers in OpenSG.*

Per la View è stata utilizzata una TableView.
I delegate utilizzati nella visualizzazione delle informazioni della PropertyView sono quelli di default delle Qt: infatti tutte le informazioni visualizzate nella tabella delle proprietà sono sotto forma di stringa (nome-tipo-valore del campo).
Nel momento in cui, invece, chiediamo alla PropertyView di editare il valore di un campo allora la PropertyView controlla che per quel tipo di campo sia stato sviluppato un delegate.

Attualmente sono stati sviluppati:

1. i delegate per tutti i tipi dei Fields base di OpenSg, sia single che multi, che sono:
   - Boolean
   - Integer
   - Floating points
   - Vector
   - Points
   - Colors
   - Matrix
   - Quaternion
   - String
2. i delegate per editare i campi che servono a creare le relazioni tra i vari oggetti, quali tutti i fields di tipo "puntatore a":
   - AttachmentContainer
   - Camera

- Viewport
- NodeCore
- Node
- Material

3. i delegate per tre dei nuovi tipi implementati in SL3D:

   - TxtDoc: non è altro che un QTextDocument di Qt, che serve a contenere un testo formattato;
   - FileName: è una stringa che rappresenta il path di un file
   - Marker: sono le coordinate di un punto 3D (Pnt3f) usato nel *core* Connector3D nei campi PntSource e PntDest.

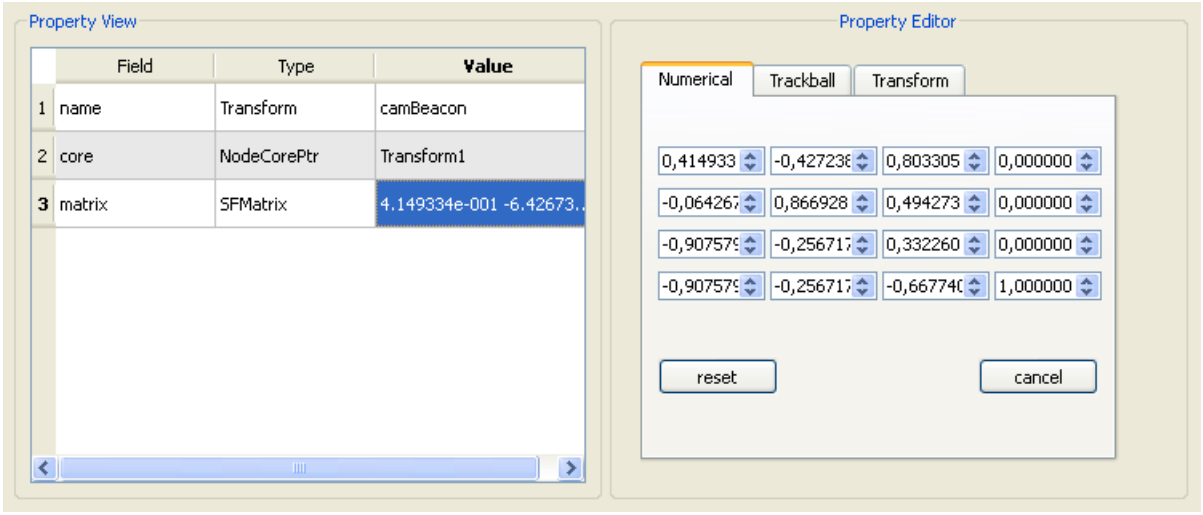


Figura 3.5: Editing di una matrice.

Nell'esempio di cui alla fig. 3.5 viene visualizzata la PropertyView che si ottiene selezionando un nodo Transform nella SlideView.

In generale in tutte le PropertyView abbiamo deciso di rendere il più possibile trasparente all'utente la divisione in *nodes* e *cores* di OpenSG.
Infatti, la prima riga della tabella invece di riportare il primo field del *core* visualizza il nome del *node*, il tipo del *core* e il suo valore; la seconda visualizza il campo *core* del *node* per poterlo editare e far sì che due *nodes* possano puntare eventualmente ad uno stesso *core*.
A prescindere da ciò le righe successive visualizzano sempre i fields del *core*.
In questo esempio il *core* Transform ha un solo field che è matrix di tipo SFMatrix. Per questo tipo di campo, come detto, è stato implementato un delegate, che è quello che vediamo di fianco nella PropertyEditor e che si attiva nel momento in cui riceve una richiesta di editing.
Nella PropertyView il delegate usato per la visualizzazione è quello di default ed infatti le informazioni sono in forma di stringa; nella PropertyEditor, invece, le stesse informazioni vengono presentati in tre modi differenti (Numerical-Trackball-Transform) e meglio editabili.

3.4.3 La costruzione di nuovi Fields

Il vantaggio di operare sui FieldContainers fa sì che l'Editor nelle due viste principali possa continuare a funzionare anche con FC che verranno implementati successivamente, senza necessità di dover intervenire sullo stesso.

Un intervento è necessario, invece, se vogliamo implementare non un nuovo FC, ma un nuovo tipo di Field.
In tal caso, dopo aver implementato il nuovo Field secondo lo stile di OpenSG, le operazioni da compiere sono:

- Creare l'interfaccia che permette di editare il nuovo tipo di Field;

- Creare il delegate;
- Registrare il nuovo tipo di Field creato in modo che l'Editor impari a riconoscerlo;
- Registrare il delegate associandolo al nuovo tipo.

Vediamo come nel concreto abbiamo operato quando abbiamo creato il nuovo Field SFFileName.

Il SFFileName non è altro che un field di tipo stringa (std::string), che contiene il path di un file.
E'stato creato per distinguere una stringa qualunque dalla stringa che rappresenta il nome di un file, che vogliamo editare attraverso un dialog, che ci permetta di cercare il file in modo visuale, come siamo abituati a fare solitamente.
Abbiamo definito il nuovo tipo nello stile OpenSG (vedi guida OpenSG) : si vedano a tal proposito i files filenametype.h e filenametype.cpp.
Successivamente abbiamo costruito la form da utilizzare per editare il Field, servendoci del designer di Qt. Questa consiste di una LineEdit, dove possiamo editare il path direttamente, affiancata da un pulsante che, in alternativa, ci permette di aprire il dialog per la ricerca dei files.

Figura 3.6: *Form per l'editing di un Field di tipo Filename.*

Dopodichè abbiamo costruito il delegate.

```
class FileName_Delegate:  public QItemEditorCreatorBase {
public:
```

```
  inline FileName_Delegate(const QByteArray &valuePropertyName)
    : propertyName(valuePropertyName), QItemEditorCreatorBase() {};

  inline QWidget *createWidget(QWidget *parent) const
  {
    LineEditFile *le = new LineEditFile(parent);
      return le;
  };
  inline QByteArray valuePropertyName() const {return propertyName;}

private:
  QByteArray propertyName;
};
```

Infine, abbiamo registrato il nuovo tipo ed il relativo nuovo delegate, inserendo le seguenti linee di codice nella InitFactory della classe DelegateFactory:

```
qRegisterMetaType<SFFilename>("SFFilename");
registerEditor((QVariant::Type)QMetaType::type("SFFilename"),new
FileName_Delegate("value"));
```

3.5 Il Parser

Il parser è l'insieme delle due operazioni di input ed output che ci permettono di passare dalla presentazione ad un file Xml e viceversa.
Il Parser è stato realizzato suddividendolo in un Writer ed un Loader.
Prima di analizzarli più nel dettaglio fissiamo qualche concetto generale.

Writer e Loader hanno in comune il fatto di operare su singole Slides, rispettivamente nel costruirle e nel salvarle, proprio perchè abbiamo originariamente deciso di rendere le Slides indipendenti tra loro.
Entrambi utilizzano Qt ed in particolare il modulo QtXml che permette di manipolare l'Xml organizzandolo in una struttura ad albero (un QDomDocument). (vedi QT)
Sia il Writer che il Loader operano, come ormai abbiamo capito sugli FC e sui Fields external: ciò fa sì che non abbiano bisogno di modifiche, né quando si costruiscono nuovi fields né quando si costruiscono nuovi FC.
Ogni FC ha una traduzione naturale in Xml, che è la seguente:

```
<TipoFC id = "nomeFC">
    ... xml dei Fields
</TipoFC>
```

I Fields si distinguono in Fields che sono riferimenti ad altri FC

```
<nameField >
    Xml del FC
</nameField >
```

e Fields che contengono un valore

```
<nameField value="valore_field"/>
```

Facciamo un esempio: una PerspectiveCamera, alla quale abbiamo dato il nome "camera4", che ha quattro campi a valori reali ed un campo "beacon" puntatore ad un *node*, ha una traduzione Xml di questo tipo:

```
<PerspectiveCamera id="camera4" >
  <beacon>
    <Node USE="camBeacon" />
```

```
    </beacon>
    <near value="1.000000e-002" />
    <far value="3.000000e+000" />
    <fov value="1.570796e+000" />
    <aspect value="1.000000e+000" />
  </PerspectiveCamera>
```

Notate che il nodo riferito dal campo "beacon" non è stato espletato per il fatto che si tratta di un riferimento ad un *node* nello Scenegraph, come vedremo meglio successivamente.

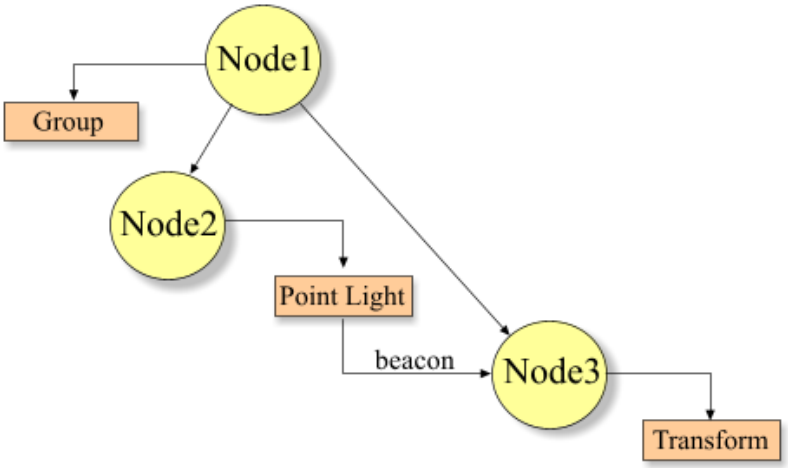
3.5.1 Il writer

Il Writer opera singolarmente su ogni Slide in due passate:

- nella prima viene creata una mappa di tutti i FC incontrati all'interno della Slide;

- nella seconda passata crea l'Xml del FC e di ogni Fields del FC, marcando come visitati i FC che incontra. Quando un campo è un riferimento ad un FC si controlla se questo è stato già visitato, ed in tal caso ci si comporta come abbiamo visto nell'esempio della camera a proposito del campo "beacon". Questo ci consente di trattare i FC condivisi, come ad esempio i *cores* tra più *nodes*.
  La seconda passata, a differenza della prima, opera distintamente e nell'ordine sullo Scenegraph, sulla ViewportList e sulla TransitionList. Nello Scenegraph quando trova un campo che riferisce un FC si chiede se tale FC non sia un *node* ed in caso positivo mette il riferimento "USE" anche se quel nodo non è stato ancora visitato: ciò consente di

preservare la struttura ad albero dei *nodes* che altrimenti non sarebbe rispettata nell'Xml (vedi fig. 3.7).




```
<Node id="Node1">
  <Node id="Node2">
    <core>
      <PointLight id="Point Light">
        <beacon>
          <Node id="Node3">
            ....
          </Node>
        </beacon>
      </PointLight>
    </core>
  </Node>
  <Node USE="Node3" />
</Node>
```

a)

```
<Node id="Node1">
  <Node id="Node2">
    <core>
      <PointLight id="Point Light">
        <beacon USE="Node3" />
      </PointLight>
    </core>
  </Node>
  <Node id="Node3">
    ....
  </Node>
</Node>
```

b)

Figura 3.7: *Esempio di uno scenegraph con un nodo luce, dove l'xml a) è ricavato trattando i nodi come FC, mentre l'xml b) dà l'idea di come si comporta SL3D.*

Il Writer oltre a scrivere il file Xml svolge la funzione di copiare tutti i files di Immagini e Modelli riferiti nella presentazione in una cartella appositamente creata chiamata "nomefile_sl3d".

Il Writer non è utilizzato solamente nelle operazioni di output, ma viene usato anche in fase di editing di alcuni campi nel reperire informazioni sugli oggetti della Slide.
L'esempio tipico è quello di reperire i nomi dei nodi di uno Scenegraph da inserire in una ComboBox.

3.5.2 Il loader

Il loader svolge la funzione di ricostruire una presentazione a partire dal file Xml.
Parsa il file Xml due volte:

- nella prima passata crea i FC corrispondenti all'Xml parsato e li inserisce in una tabella hash con chiave il nome del FC.
- nella seconda passata riempie i campi del FC creando i collegamenti tra i vari FC.

Il loader viene utilizzato anche quando la presentazione che abbiamo editato viene mandata in fullscreen, creando una copia delle Slides e assegnando ad esse la descrizione Xml.

3.6 Due cores: TextBox e PlyGeometry

OpenSG mette a disposizione un core Geometry generico nel quale è possibile inserire tutte le informazioni sulla topologia di un oggetto, specificando vertex , normal , color e index array, primitive type e coordinate di texture, lasciando poi ad OpenSg il compito di renderizzare il tutto.
Nei due nodi che analizzeremo di seguito, TextBox e PlyGeometry, il rendering non viene demandato ad OpenSG ma, come vedremo, viene affidato a

delle librerie esterne.
Ciò ha reso necessario l'implementazione di funzioni per il calcolo del volume occupato dall'oggetto (bounding box) e per il calcolo dell'intersezione raggio -triangolo per poter selezionare l'oggetto.

3.6.1 TextBox

Come già accennato permette il rendering di testo formattato, con la possibilità di utilizzare diversi fonts, dimensioni del carattere, elenchi puntati, grassetto, corsivo, ecc.
Per far questo nel *core* TextBox sono state implementate due funzioni principali:

- Layout()
- Draw()

La funzione Layout() permette di sezionare il testo in linee a seconda della dimensione della Viewport nella quale la TextBox viene renderizzata.

Tra i suoi campi troviamo :

- quattro campi, left, top, right e bottom, per i rispettivi margini , specificabili, come la Viewport, in percentuale, con un valore tra 0 e 1, o in pixel.
- la distanza tra le linee (linedistance) in pixel;
- il rientro dal margine sinistro degli elenchi puntati in pixel;
- drawtype, che assume un valore da 1 a 6, che indica una delle modalità di rendering messe a disposizione da FTGL:

  - due di tipo raster, che operano per pixel, e sono BitmapMode e PixmapMode;
  - quattro di tipo vettoriale, OutlineMode, PolygonMode, ExtrudeMode e TextureMode.

- TextDoc , che è un nuovo field implementato ed altro non è che un QTextDocument di Qt che mantiene il testo formattato.

Il QTextDocument è una struttura composta da una serie di blocchi (QTextBlock) che rappresentano i paragrafi (vedi fig.3.8); a sua volta un blocco è formato da frammenti (QTextFragment), che sono dei frammenti di testo che hanno un unico formato di carattere.
Il QTextDocument può contenere anche immagini, tabelle e frames, che non sono stati utilizzati in SL3D.
Un QTextDocument può essere scorso con un cursore (QTextCursor), che fornisce informazioni sul formato del carattere corrente e consente operazioni di avanzamento per parola, per frammento, per carattere, selezioni, inserzioni ed altro.

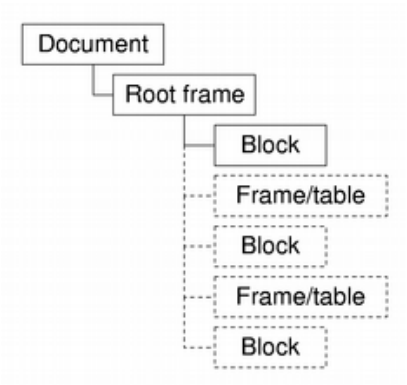


Figura 3.8: *Schema di un QTextDocument.*

La funzione di Layout si serve del cursore per scorrere il QTextDocument e utilizza la libreria FTGL per la conversione di una stringa di testo nel corrispondente testo OpenGL e per darci informazioni sul bounding box occupato dal testo.
Con queste informazioni riesce a sezionare il documento in linee di testo riempiendo una struttura dati che è stata appositamente creata per SL3D e contenuta nel campo "lines".
Tale campo è un Field internal, composto da un vettore di linee di testo (MFLineTxt), ognuna delle quali è un vettore di frammenti (FragTxt) , con informazioni per ogni frammento sul font, sulla dimensione, sulla posizione nella linea, sul tipo (bold, italic, ...), sull'allineamento (centrato, a destra, ...) e sugli indici del carattere iniziale e di quello finale all'interno del QTextDocument.

L'operazione di Layout viene fatta in base alle dimensioni della Viewport nella modalità fullscreen e per far ciò utilizza le informazioni sulla risoluzione per la quale la presentazione è preparata: nella modalità Edit i fonts vengono, quindi, opportunamente ridimensionati per far sì che il testo mantenga le stesse proporzioni che avrà in fullscreen.

La funzione di Draw scorre questa struttura dati e, grazie a FTGL, crea il testo OpenGL con i relativi allineamenti e formattazioni.

FTGL è una libreria che permette il rendering in un contesto OpenGL e per far ciò si appoggia a FreeType2, che è un font engine per la manipolazione di fonts di diversi formati, tra i quali TrueType e CFF font, e che restituisce informazioni sulla metrica di ogni singolo carattere del font (Glyph, vedi fig.3.9).

La funzione di drawing è composta da due cicli, quello esterno sulle linee, che si occupa degli allineamenti, e quello interno sui frammenti, attraver-

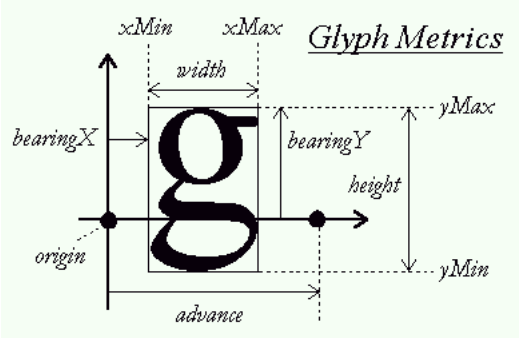


Figura 3.9: *Metrica di una glyph.*

so i quali scandisce la struttura creata nel field internal lines e costruisce la versione OpenGL del testo a seconda della modalità indicata nel field drawtype.

Ogni qual volta utilizziamo un font differente, anche solo nelle dimensioni, la TextBox è costretta a creare per quel font, attraverso FTGL, le Glyph di tutti i 256 caratteri dai quali i font sono solitamente formati, che vengono restituiti in una struttura dati (FTFont).
Per far questo la TextBox non crea direttamente un FTFont ma si appoggia ad un FontManager che conserva queste strutture dati per usi futuri, evitando di doverli ricreare ogni volta e consentendo di poterli condividere tra le varie TextBox.

La TextBox crea una DisplayList per ottimizzare il rendering del testo, che altrimenti farebbe degradare le prestazioni: ad ogni modifica del documento o al ridimensionamento della Viewport o dei margini della TextBox viene rieseguita l'operazione di Layout e viene ricreata la Display List.

L'ultimo aspetto da analizzare sul *core* TextBox è l'interazione con la ToolBar, che viene garantita dall'utilizzo del QTextDocument che emette tutti i segnali che vengono poi catturati dall'applicazione per aggiornare la

ToolBar delle formattazioni e viceversa.

Al Field TxtDoc è stato associato come Delegate un QTextEdit, che è l'editor per il testo fornito da Qt.

3.6.2 PlyGeometry

E' il *core* che ci permette di caricare dei file .ply, che è un formato non supportato da OpenSG.
Come detto il rendering del *core* non viene affidato ad OpenSG ma viene fatto appoggiandosi ad una libreria esterna (VCG Library), sviluppata dal Visual Computing Lab del CNR di Pisa.
La libreria è interamente scritta in C++, che la rende portabil, e consente la manipolazione e visualizzazione con OpenGL di meshes di triangoli e tetraedri; include un wrapper che permette di leggere files .ply e renderizzarli.

Nel core PlyGeometry, oltre al campo "filename", che contiene il path del file .ply, troviamo altri tre campi che sono poi i parametri di cui necessita la libreria per il rendering quali:

- DMode: per la modalità di rendering, che indica se visualizzare il solo bounding box della mesh, o la mesh per punti, smooth ecc;
- CMode: per la modalità di colorazione, che può essere, ad esempio, per vertice o per faccia;
- TMode: per la modalità di texturizzazione della mesh.

Nel nodo PlyGeometry sono poi state implementate altre due funzioni.
La prima , adjustVolume(), serve per il calcolo del bounding box dell'oggetto ed è utilizzata da OpenSG per mantenere aggiornate le informazioni contenute nei nodes sul bounding box del proprio sottografo; tali informazioni

vengono utilizzate per poter escludere dalla visita tutti quei nodi che ricadono fuori dal volume di rendering definito dalla camera istante per istante.
La seconda, intersectEnter(), consiste nella funzione di intersezione raggio-triangolo che serve per la selezione degli oggetti.

3.7 Le transizioni

Una transizione compie un'operazione che può essere immediata (come settare un campo con un valore) o perdurare nel tempo (come ad esempio la dissolvenza nel passaggio tra una slide ed un'altra).

Ogni transizione ha uno stato che può assumere tre valori:

- Executed
- NotExecuted
- InProgress

Inizialmente quando viene creata una slide tutte le azioni sono in stato di NotExecuted.
Quando la transizione deve essere eseguita su input dell'utente l'azione passa nello stato di InProgress: questo fa sì che ad ogni frame il sistema chiami la funzione

execTransition(const Time t)

che fa avanzare la transizione in funzione del parametro "t" che è il tempo.

Per implementare una nuova transizione occorre sempre specificare questi due metodi :

- execTransition()
- endTransition()

La execTransition() è la funzione che definisce l'azione svolta dalla transizione in funzione del tempo.

La endTransition() è la funzione che viene chiamata per far terminare l'azione e la porta nello stato di Executed.
Compie la stessa azione della execTransition() ma lo fa immediatamente e non in funzione del tempo, per portarla subito a termine.
Quando viene richiesta l'esecuzione della transizione n viene prima chiamata la endTransition() della transizione n-1, se ancora non è terminata, che la porta nello stato di Executed e dopodichè si attiva la transizione n.

Le transizioni vengono utilizzate per animare la slide e far avanzare la presentazione.
E' necessario in alcuni casi, quando ad esempio il pubblico lo richiede per riprendere dei concetti, poter compiere dei passi indietro.
In SL3D invece che scegliere di rendere le transizioni reversibili si è preferita una soluzione alternativa, che consiste nel reinizializzare la slide e rieseguire tutti i passi fino al precedente.
Più formalmente, per compiere un passo indietro dalla transizione n alla transizione n-1 viene ricreata la slide partendo dall'Xml che descrive lo stato iniziale della slide e vengono eseguite le n-1 transizioni.

Le operazioni che il sistema compie nelle transizioni tra slides, per passare dalla slide 1 alla slide 2, sono le seguenti:

- crea una Viewport che occupa tutta la finestra di rendering ed in essa inserisce un GrabForeground, che serve a catturare la schermata ;

- attacca la Viewport in fondo alla lista delle Viewports della Slide1 e ridisegna un altro frame in modo che il GrabForeground possa catturare l'immagine finale della slide1;

- infine, l'immagine del GrabForeground viene passata alla transizione che crea l'azione di passaggio all'interno della InitAction della Slide 2.

Capitolo 4

Esempio di presentazione

4.1 Introduzione

In questo capitolo descriveremo un'applicazione pratica di SL3D.

Abbiamo già detto come negli ultimi tempi la tecnologia 3D scanning si sia rivolta principalmente all'acquisizione di modelli digitali di opere d'arte, sia come strumento di supporto all'attività di restauro che come ausilio nella presentazione interattiva delle opere al pubblico.

Due dei progetti sui quali si è concentrata l'attenzione del VCG del Cnr di Pisa negli ultimi anni sono la costruzione di un modello digitale del David di Michelangelo[7] a sostegno del recente restauro e la ricostruzione digitale del complesso funerario di Arrigo VII[8].

Per quanto riguarda il David il modello è stato utilizzato come strumento su cui eseguire indagini specifiche.
Una simulazione sul modello ha permesso ad esempio di caratterizzare le diverse classi di esposizione della superficie della statua ad agenti esterni, quali pioggia o polveri; un'altra ha consentito di calcolare direttamente sul modello il baricentro, la linea di caduta verticale ed il volume dell'opera, tutte

informazioni indispensabili per valutare le condizioni statiche della stessa. Ma il modello è servito anche come mezzo di documentazione e presentazione dei dati relativi al restauro.

Diverse invece sono state le finalità che hanno ispirato il lavoro relativo al complesso funerario di Arrigo VII.
Il complesso fu scolpito nel 1315 da Tino di Camaino ed era destinato all'abside della cattedrale di Pisa.
Nel corso dei secoli il complesso è stato sottratto alla collocazione originaria e destinato a luoghi diversi di Piazza dei Miracoli, mentre le statue che lo componevano sono andate perse o sono state riutilizzate in altri contesti.
Sono state fatte molte ipotesi da parte degli storici dell'arte sulla disposizione originaria delle statue ma tali ipotesi risultano oggi poco convincenti dato che alcuni elementi che sicuramente facevano parte del complesso sono stati solo recentemente riconosciuti come tali e dato che la disposizione dell'abside nel XIV secolo si è scoperto recentemente essere diversa da quella che oggi conosciamo (erano ad esempio presenti due finestre oggi murate).
Il team di lavoro ha provveduto quindi alla acquisizione tramite la tecnica 3d scanning dei modelli e alla loro collocazione in un contesto 3D proponendo un'ipotesi di ricostruzione del complesso più adatta di quelle precedenti, che tiene conto dei nuovi elementi scoperti e che soprattutto si basa sulle accurate misurazioni che la tecnica 3D scanning consente.

I risultati di entrambi i progetti sono stati presentati al pubblico, rispettivamente nel 2002 e nel 2004, con l'ausilio di Virtual Inspector.

Nelle prossime pagine descriveremo una presentazione dei risultati dei due progetti effettuata con SL3D, diretta a porre in evidenza le potenzialità che la tecnica del 3D Scanning mette a disposizione; per ogni slide che compone la presentazione evidenzieremo le opportunità che lo strumento offre.

4.2 La presentazione

La presentazione si compone nove slides: un'intestazione, due slides dedicate al David e sei all'Arrigo VII.

Siamo partiti dal tipo di layout più semplice, costituito da quattro Viewports: una per lo sfondo, in questo caso un GradientBackground, che occupa tutto lo schermo; una per il titolo in 3D; una per il testo ed una per i modelli (vedi fig. 4.1).

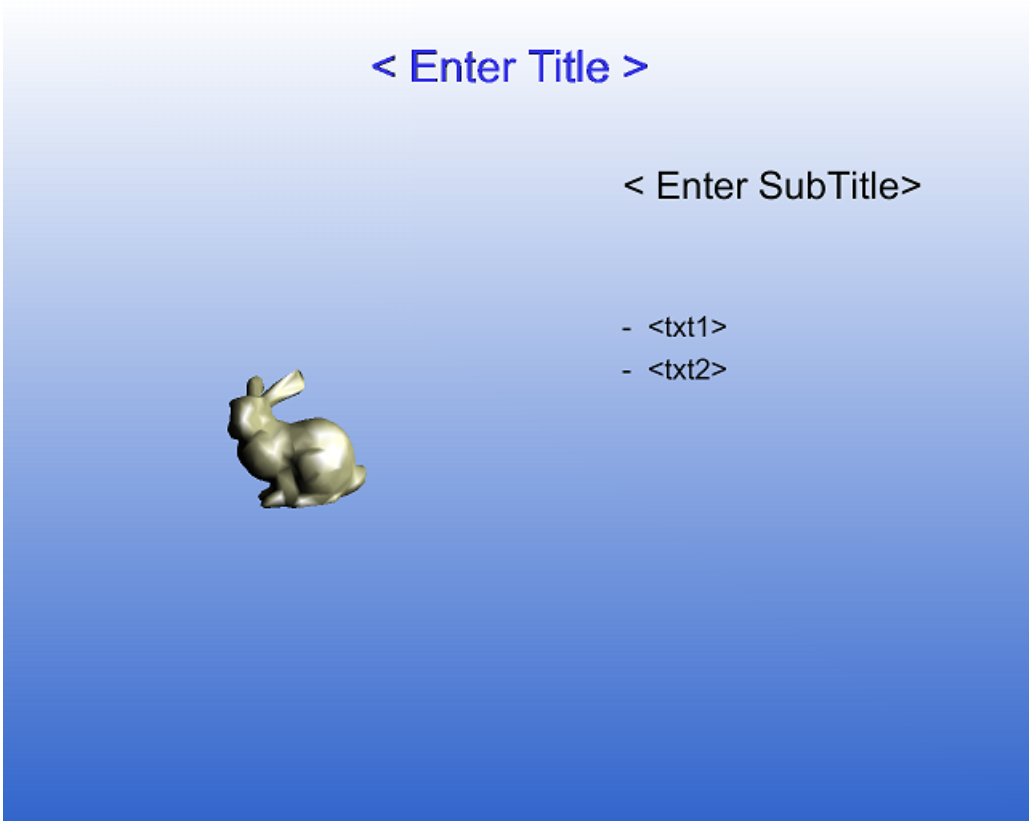


Figura 4.1: *Layout utilizzato per la presentazione.*

Lo Scenegraph è composto da tre nodi di tipo group che sono dati da due nodi-luce direzionali (DirectionalLight), che contengono il sottoscenegraph

del titolo e del modello, e da un nodo Group, che contiene il sottoscenegraph del testo, che rappresentano i nodi-radice delle rispettive Viewports.

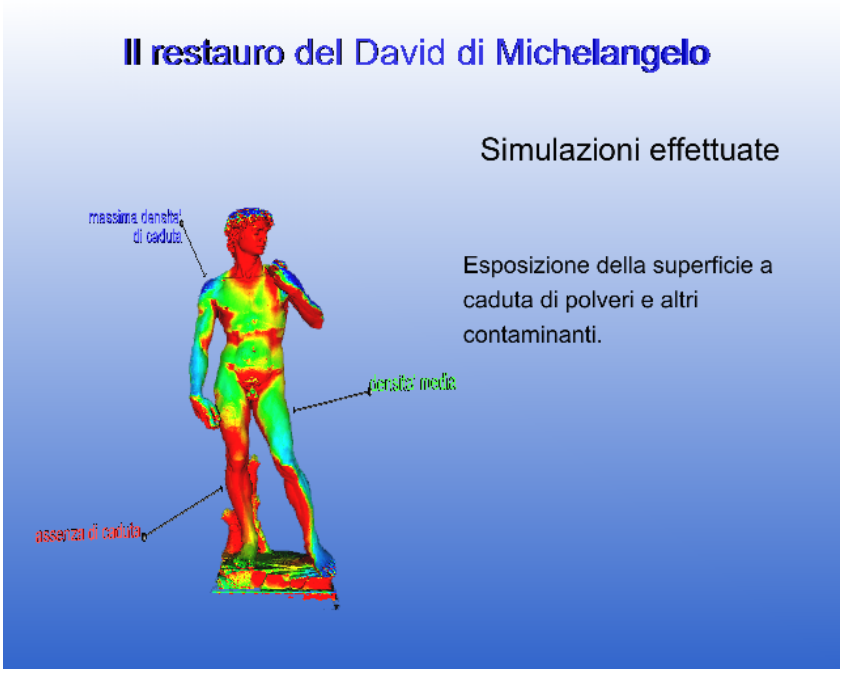


Figura 4.2: *Slide1 - David polveri.*

Rispetto alla slide-base, nella slide1 , dopo aver definito titolo, sottotitolo e testo, ci siamo limitati ad operare sullo Scenegraph della Viewport del Modello, caricando una mesh del David con i diversi colori che simulano l'incidenza degli agenti esterni ed inserendo delle label 3D e dei connettori 3D per attribuire ad ogni colore un grado di incidenza diverso. Inoltre abbiamo definito delle semplici transizioni (ShowNode) per la comparsa in successione su input utente rispettivamente del modello, del testo ed infine delle label con i connettori, che erano stati nascosti nella InitAction della slide.

Nella slide2 pur mantenendo lo stesso layout abbiamo complicato gli scenegraph del modello e della TextBox in modo da visualizzare in succes-

Figura 4.3: *Slide2 - David baricentro.*

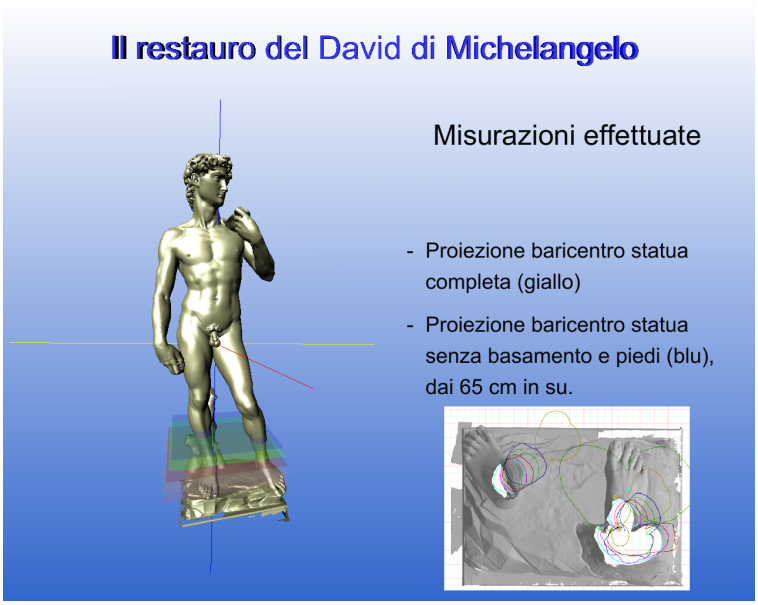


Figura 4.4: *Slide2 - David piani.*

sione prima il modello con il baricentro e la relativa Textbox e poi il modello con i piani e la sua TextBox, aggiungendo in quest'ultima un'immagine. Quindi oltre a caricare una mesh abbiamo inserito delle figure geometriche che il sistema mette a disposizione, costituite da delle linee colorate , per rappresentare il baricentro, e da dei piani per evidenziare delle sezioni a varie altezze operate sul modello, che ritroviamo poi nella immagine compresa nella TextBox.

Di seguito mostriamo lo Scenegraph visualizzato nella SlideView in fase di editing.

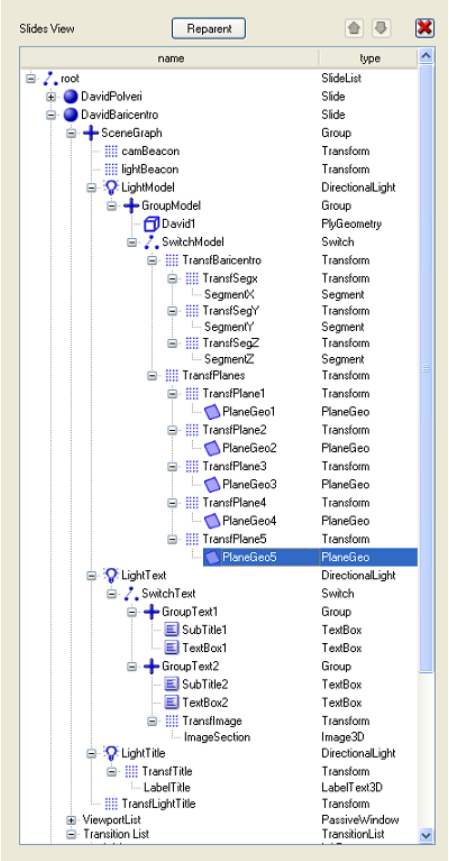


Figura 4.5: *Scenegraph della slide2.*

Nelle transizioni, rispetto alla slide1, abbiamo utilizzato la transizione "Set" che ci permette di modificare Fields, intervenendo sui campi choice dei nodi Switch, per visualizzare la slide2 nei due momenti, come sopra detto. Nella InitAction oltre alle operazioni di inizializzazione della slide, come nella slide1, troviamo la prima azione di passaggio tra slide1 e slide2 , in questo caso una Sliding.

Tutte le successive slides sono dedicate al monumento funerario di Arrigo VII: in particolare le prime quattro presentano i probabili elementi statuari che ne facevano parte e le ultime due sono dedicate alle ipotesi ricostruttive.

Figura 4.6: *Slide3 - Statue Arrigo.*

La prima slide, slide3, presenta il tipo di layout generale già visto, con la differenza che abbiamo deciso di suddividere l'area destinata al rendering dei modelli con altre Viewports, una per modello, in modo da avere la pos-

sibilità di operare con la trackball singolarmente su ogni elemento. Per lo sfondo abbiamo utilizzato un ImageBackground con l'immagine della pianta del Duomo di Pisa. Questa immagine viene dal sistema codificata con una texture e tale texture viene condivisa da tutte le altre slides successive. Nello scenegraph di questa slide abbiamo utilizzato un nodo AnimRotation per animare il modello dell'Arrigo in trono, facendolo ruotare su sé stesso; l'animazione viene attivata da una transizione.

A completamento per questa singola slide riportiamo di seguito la SlideView.

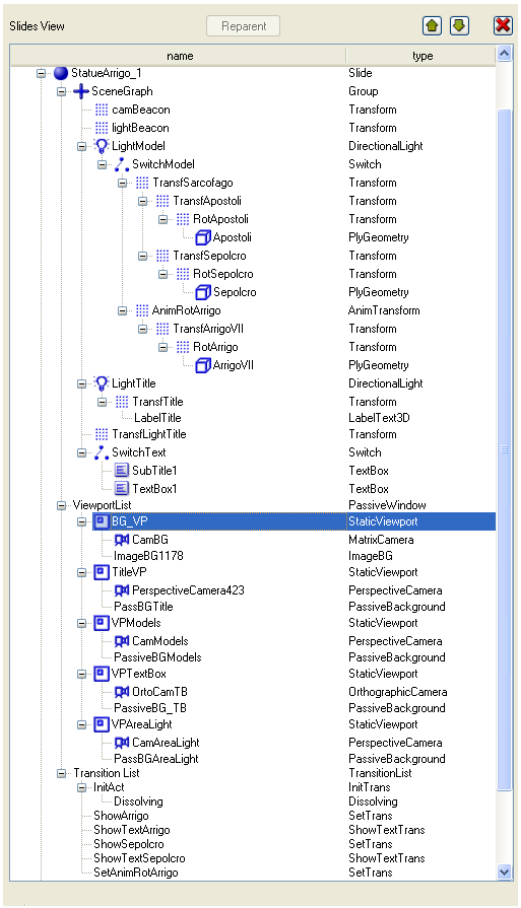


Figura 4.7: *SlideView della slide3.*

Figura 4.8: *Slide4 - Statue Arrigo.*

Le slides 4-5-6 sono simili, a parte le Viewports dei modelli che sono state dimensionate in maniera opportuna. Nella slide4 è stato inserito un connettore con una label, riferito ad un particolare elemento di una statua. Nella Slide5 uno dei modelli è renderizzato in Wire Mode anziché in Smooth. In tutte le slides di presentazione degli elementi statuari, così come nell'ultima, abbiamo inserito nell'angolo in alto a sinistra un'area trasparente all'utente che permette di manipolare l'orientamento dell'illuminazione (vedi figura). E' stata creata giocando sui meccanismi di OpenSG di interazione tra Trackball e Viewport . La Trackball messa a disposizione da OpenSG permette di modificare la matrice di trasformazione della camera (indicata nel beacon della camera) della Viewport sul quale andiamo ad operare con il mouse. Quello che tecnicamente abbiamo fatto è stato assegnare lo stesso nodo trasformazione sia al beacon della Camera della Viewport della LightArea che al beacon del nodo Light dei modelli.

Figura 4.9: *Slide5 - Statue Arrigo.*

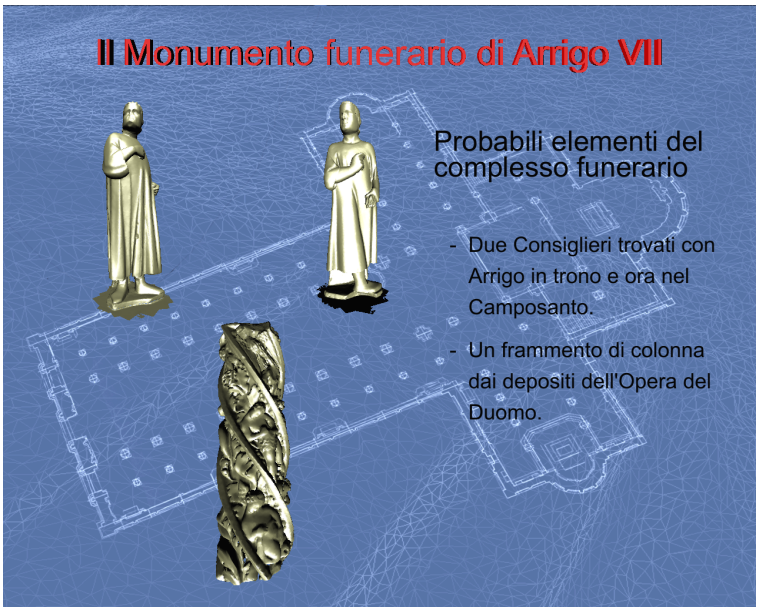


Figura 4.10: *Slide6 - Statue Arrigo.*

Figura 4.11: *Slide7 - Ipotesi ricostruttive.*

Figura 4.12: *Slide8 - La nostra ipotesi.*

Nell'ultima slide, slide8, abbiamo presentato la nostra ipotesi ricostruttiva del monumento. Si tratta di una scena composita che dimostra come il sistema permetta un editing sostanzialmente semplice: tutti i modelli, infatti, erano centrati nell'origine con un orientamento differente e sono stati agevolmente riposizionati l'uno rispetto all'altro intervenendo con delle trasformazioni (Transform).

In questa slide abbiamo importato tutti i modelli, che abbiamo precedentemente mostrato, con l'aggiunta di un'edicola, nella quale collocarli, che è stata realizzata a parte con un Editor 3D e salvata in formato .3ds, che è uno dei tanti formati supportati da SL3D.

E' composta da 19 modelli complessi con una media di tot. facce: nonostante ciò la slide viene caricata in tempi contenuti grazie al fatto che il sistema condivide le mesh che sono state caricate di volta in volta nelle precedenti slides.

In questa scena è stata sfruttata la caratteristica di SL3D di poter condividere i cores: in particolare nella realizzazione delle colonne tortili avevamo a disposizione solo un frammento delle stesse ed abbiamo perciò dovuto assemblare insieme più copie dello stesso.

Ciò è stato realizzato inserendo quattro nodi (due per la colonna di destra e due per quella di sinistra) che condividono un solo core PlyGeometry, che contiene la mesh del frammento.

Capitolo 5

Conclusioni

Nel corso dell'esposizione abbiamo cercato di illustrare le esigenze da cui è nato ed i possibili ambiti applicativi di SL3D, descrivendo il sistema sotto il profilo utente e sotto il profilo delle scelte implementative che ci hanno guidato.

Chiaramente questa tesi non poteva rappresentare l'occasione per sviluppare un prodotto completo e paragonabile ai software di uso commerciale. L'intento di fondo è stato quello di porre le basi per i futuri sviluppi.

In ragione di ciò abbiamo scelto di utilizzare librerie open source, come le Qt e soprattutto OpenSG, che ha consentito lo sviluppo di un sistema, oltre che portabile, estendibile, grazie alla caratteristica dei FieldContainers, sui quali si basano sia l'Editor che le funzioni di input/output.

Molti degli sforzi implementativi sono stati rivolti allo sviluppo di un Editor estendibile con pochi semplici passi.

Tuttavia accanto al vantaggio che l'utilizzo dei FieldContainers comporta, il fatto di essere basato così pesantemente su OpenSg fa sì che SL3D ne assorba anche alcuni lati negativi: ad esempio eventuali modifiche apportate ad OpenSg rendono inutilizzabile un file SL3D precedentemente salvato, nel

caso di eliminazione o ridenominazione anche di un solo campo di un nodo.

Inoltre, attualmente è necessaria una conoscenza generale di OpenSG e ciò rende il sistema non immediatamente approcciabile da qualsiasi utente.

Per quanto riguarda le funzionalità del sistema molto può essere ancora fatto, basti pensare che attualmente esso non sfrutta appieno neanche tutte le possibilità offerte da OpenSG.

In questo senso il settore maggiormente sviluppabile è senz'altro rappresentato da nuove animazioni e transizioni, che possano in fase di presentazione rendere più funzionale e accattivante la stessa.

Ringraziamenti

I miei ringraziamenti vanno in primo luogo ai professori Paolo Cignoni e Fabio Ganovelli dai quali è nata l'idea alla base del progetto e che hanno creduto in me, guidandomi nella giusta direzione.

Un grazie a tutti i componenti del Visual Computing Lab del CNR di Pisa che mi hanno fornito spunti ed assistito con i loro consigli nel risolvere i vari problemi incontrati nello sviluppo del progetto.

Una menzione speciale meritano gli utenti del forum di OpenSG, in particolare Dirk e Andreas, con i quali ho intrattenuto fitta e, per me, proficua corrispondenza.

Infine un grazie speciale ad Antonia per il supporto morale di tutti questi anni.

Bibliografia

[1] The OpenSG Starter guide 1.6.0 - *Generated by Doxygen 1.4.3, August 25, 2005.*

[2] Qt Reference Documentation 4.1.3 - *Trolltech, 2005.*

[3] R. Viscardi - Power Point no problem *McGraw Hill, 2003.*

[4] The OpenGL Reference Manual - *The Bluebook.*

[5] FreeType-2.1.10: Documentation.

[6] The FTGL Documentation 2.1.

[7] R. Scopigno et al. - *Il modello digitale 3D del David e il suo uso nel progetto di restauro*, Kermes - La rivista del restauro, Gennaio-Marzo 2003.

[8] C. Baracchini et al., - *Digital reconstruction of the Arrigo VII funerary complex*, The Eurographics Association 2004.

[9] Virtual Inspector v1.5 - User Manual - *Visual Computing Laboratory, 2003.*