TESI DI LAUREA

UNIVERSITA' DEGLI STUDI DI PERUGIA
Facoltà di Scienze Matematiche, Fisiche e Naturali

UN SOFTWARE PER L'ARREDAMENTO DI INTERNI

Candidato
Gulì Tullio

Relatore
Marcugini Stefano

CORSO DI LAUREA TRIENNALE IN INFORMATICA
A/A 2009/2010

INDICE

INTRODUZIONE 1

CAPITOLO 1 - LA GRAFICA 3D

1.1 Introduzione 3

1.2 La Computer Grafica 4

1.3 Modelli e Rendering 4

1.4 Le API Java 3D 5

CAPITOLO 2 - JAVA 3D: CLASSI ED OGGETTI

2.1 Creazione dell'universo virtuale: SceneGraph, SimpleUniverse, Branches 7

2.2 Oggetti e trasformazioni: BranchGroup, TransformGroup 9

2.3 Apparenza di un oggetto: Light, Material, Texture 11

2.4 Spostamento della scena: behavior 14

CAPITOLO 3 - BASE DI DATI

3.1 Java e JDBC 17

3.2 Perchè utilizzare un database? 18

3.3 Normalizzazione e dipendenze funzionali 20

3.4 Implementazione della base di dati 21

CAPITOLO 4 – REALIZZAZIONE DELL'APPLICAZIONE

4.1 Introduzione 26

4.2 Progettazione della planimetria 28

4.3 Calcolo dell'area di un poligono irregolare 30

4.4 Collisioni tra oggetti 31

4.5 Importazione modelli 3D 34

INDICE

4.6 Matrici di traslazione e rotazione 37
4.7 Importazione ed esportazione dei dati 38

CAPITOLO 5 - CONCLUSIONI E POSSIBILI SVILUPPI
5.1 Pannello amministrazione 41
5.2 Pannello utente 42
5.3 Planimetria avanzata 43
5.4 Gestione su vista 3D 44
5.5 Conclusioni 45

BIBLIOGRAFIA 47

INTRODUZIONE

Lo scopo di questo lavoro di tesi è la progettazione e la realizzazione di un'applicazione che permetta la creazione e l'arredamento di una stanza la quale potrà essere visualizzata in una rappresentazione tridimensionale.

A tale scopo è stata utilizzata tecnologia Java che consente la creazione di applicazioni indipendenti dalla piattaforma. Tale caratteristica, in particolar modo in questi ultimi tempi, grazie al crescente utilizzo del web, fa sì che l'interesse per questa tecnologia stia divenendo sempre maggiore. Tutti i suoi pregi, però, come la portabilità, la semplicità e la sicurezza, inevitabilmente introducono dei limiti in termini di performance compensati dalla presenza di un hardware sempre più potente.

Tutto ciò mi ha portato a realizzare un'applicazione JAVA con l'obiettivo principale di realizzare un software di possibile utilizzo aziendale i cui punti forza, oltre alle funzionalità, possano essere la semplicità d'uso e l'utilizzo di un interfaccia grafica moderna al passo con i tempi.
La scelta infine è caduta su Java 3D, un ricco set di API di medio-alto livello, che permette la creazione di applicazioni grafiche 3D. Grazie a computer dotati di grande potenza di calcolo e a componenti elettronici dedicati, come le moderne schede video, la grafica 3D sta prendendo piede in molti campi a partire da quello scientifico fino ad arrivare a quello medico e la sua presenza diventa sempre più richiesta.

La finestra di lavoro dell'applicazione è realizzata con interfaccia SWING composta da più schermate: una schermata per la creazione della stanza, una per il posizionamento e la scelta dell'arredamento da voler utilizzare, una relativa al resoconto spese ed infine una per la visualizzazione tridimensionale

dell'intera scena. Le informazioni relative agli oggetti (i mobili da selezionare per comporre l'arredamento) saranno memorizzate in una base di dati, mentre la loro posizione e le informazioni sulla struttura della stanza potranno essere salvate su un file di progettazione apposito. La scelta dell'utilizzo di una base di dati di oggetti rende l'applicazione flessibile e facilmente aggiornabile poiché, per poter lavorare con nuovi mobili, è sufficiente inserirli nel database.

Scendendo più nello specifico, nel primo capitolo prenderò in esame gli sviluppi della computer grafica 3D, descriverò i principali vantaggi e svantaggi nell'utilizzo delle API Java 3D e di come possa essere generata un'immagine a partire dalla sua descrizione matematica. Nel secondo capitolo andrò più in dettaglio descrivendo il funzionamento di Java 3D soprattutto per quanto riguarda la visualizzazione dell'universo 3D, la creazione degli oggetti e di tutte le loro caratteristiche, il posizionamento della vista e lo spostamento della scena. Nel capitolo successivo, invece, prenderò in esame la memorizzazione delle informazioni su una base di dati in particolar modo della struttura e delle dipendenze funzionali utilizzate.
Nel quarto capitolo parlerò delle principali problematiche riscontrate per quanto concerne la progettazione della stanza, il calcolo dell'area di un poligono irregolare, le collisioni fra gli oggetti, l'importazione dei modelli 3D, la traslazione e rotazione dei modelli ed infine l'importazione ed esportazione dei dati. Nell'ultimo capitolo vengono avanzate proposte per possibili sviluppi futuri e presentate le conclusioni sulla progettazione dell'applicazione.

CAPITOLO 1
LA GRAFICA 3D

1.1 Introduzione

Oggetti tridimensionali semplici possono essere rappresentati con equazioni definite su un sistema di riferimento cartesiano tridimensionale: per esempio, l'equazione $x^2+y^2+z^2=r^2$ definisce una sfera di raggio r. Poiché questo non basta per rappresentare tutto ciò che ci circonda, l'insieme degli oggetti realizzabili può essere ampliato con una tecnica chiamata geometria solida costruttiva la quale combina oggetti solidi come cubi, sfere, cilindri per formare oggetti più complessi attraverso le operazioni di unione, sottrazione o intersezione.

Queste equazioni non sono tuttavia sufficienti a descrivere con accuratezza le forme complesse che costituiscono la gran parte del mondo reale. L'impiego di pure equazioni matematiche richiede l'utilizzo di una gran quantità di potenza di calcolo e quindi non risulta pratico per le applicazioni in tempo reale come videogiochi e simulazioni.

Una tecnica più efficiente, tuttora la più diffusa e flessibile, è la modellazione poligonale. Questa permette un maggiore livello di dettaglio a spese però della maggiore quantità di informazioni necessarie a memorizzare l'oggetto risultante, chiamato modello poligonale. Un modello poligonale è “sfaccettato" come una scultura grezza, ma può essere comunque raffinato con algoritmi per rappresentare superfici curve. Il modello viene raffinato con un processo di interpolazione iterativa rendendolo sempre più denso di poligoni, che approssimeranno meglio curve ideali derivate matematicamente dai vari vertici del modello.

1.2 La Computer Grafica

La computer grafica si occupa della generazione e manipolazione di immagini per mezzo del computer. È anche quella disciplina che studia le tecniche e gli algoritmi per la visualizzazione di informazioni numeriche prodotte da un elaboratore. Gioca un ruolo sempre più importante in una moltitudine di ambiti professionali e di consumo come i videogiochi, il foto-ritocco, il montaggio di filmati, la tipografia, la progettazione grafica nelle industrie (CAD), l'elettronica, la visualizzazione di dati tecnico/scientifici (CAE) o i sistemi informativi territoriali (SIT o GIS). Al giorno d'oggi costituisce uno dei campi dell'informatica moderna più ricchi di applicazioni.

La maggior parte del software che oggi viene eseguito su personal computer utilizza interfacce grafiche basate su menu, icone, e oggetti sullo schermo. Esistono software che permettono di disegnare parti meccaniche, planimetrie di edifici, circuiti stampati, mappe per qualsiasi tipo di informazione geografica, schemi e diagrammi.

Ad ampliare tutto ciò c'è la computer grafica 3D che consente di descrivere ambienti ed oggetti dotati di tre dimensioni e di visualizzarli normalmente su superfici a 2 dimensioni come, ad esempio, lo schermo di un pc.

1.3 Modelli e Rendering

Il processo di produzione dell'immagine finale a partire da un modello matematico è detto rendering. Con il crescente perfezionamento della grafica computerizzata è diventato un oggetto di studio e ricerca sempre più importante.

Esistono molti algoritmi di rendering, ma tutti implicano la proiezione dei modelli 3D su una superficie 2D. Schematicamente, il metodo di produzione è

composto da due elementi: una descrizione di ciò che si intende visualizzare, composta da rappresentazioni matematiche di oggetti tridimensionali, detti "modelli", e un meccanismo di produzione di un'immagine 2D dalla scena, detto "motore di render" che si fa carico di tutti i calcoli necessari per la sua creazione, attraverso l'uso di algoritmi che simulano il comportamento della luce e le proprietà ottiche e fisiche degli oggetti e dei materiali.

Tutto ciò che ci circonda può essere analizzato in termini di una serie di fenomeni visibili. Le ricerche e i progressi nel campo del rendering sono state in gran parte motivate dal tentativo di simularli in modo accurato ed efficiente: le ombre, la dispersione della luce, la riflessione e la rifrazione, la trasparenza, la sfocatura degli oggetti in movimento sono tutti fenomeni che oggi possiamo riprodurre virtualmente grazie agli enormi progressi in questo campo.

1.4 Le API Java 3D

Le API Java 3D sono una libreria di classi e metodi specifiche per la realizzazione di applicazioni dotate di interfaccia grafica tridimensionale. Java 3D consente la produzione in tempo reale di grafica 3D tramite un insieme di API omogenee e multipiattaforma. Queste non si occupano del rendering a basso livello delle immagini poiché questa operazione è delegata dal motore di rendering di Java 3D alle librerie native della piattaforma sottostante, come OpenGL e Direct3D. Java3D non è ancora in grado di fornire completo supporto per alcune funzionalità, come la gestione delle ombre, delle texture animate, delle superfici trasparenti o degli specchi. Queste funzionalità sono infatti molto dispendiose in termini di risorse e si nota sensibilmente la lentezza della loro esecuzione.

Alla base di Java 3D c’è lo schene graph un albero a cui si appendono tutti gli oggetti della scena 3D. Questi oggetti saranno poi trasformati ed animati

utilizzando appositi costruttori e trasformatori. Il runtime engine si occupa di ottimizzazioni e operazioni aggiuntive che non sono previste nelle API native. Esso sceglie l'ordine di attraversamento dell'albero che non è limitato ad una direzione di tipo sinistra-destra o sopra-sotto e permette, quindi, di utilizzare strategie di ottimizzazione della visualizzazione.

La rappresentazione di un'immagine tridimensionale avviene seguendo il seguente schema:

- ✓ definizione della porzione di spazio visibile all'utente
- ✓ rendering: conversione dello spazio visibile in un'immagine 2D
- ✓ rasterization: l'immagine viene convertita in una griglia di pixel
- ✓ shading: viene impostato il colore dei pixel

Tutte le suddette operazioni sono eseguite direttamente, e in maniera trasparente al programmatore e all'utente, dalla Java Virtual Machine, che immediatamente dopo, mostra l'effettiva scena attiva, pronta all'eventuale interazione. In generale, Java3D, presenta una certa semplicità d'uso e nonostante i difetti rimane un potentissimo strumento a disposizione dei programmatori fornendo un'ulteriore funzionalità all'intera tecnologia Java.

CAPITOLO 2

JAVA 3D: CLASSI ED OGGETTI

2.1 Creazione dell'universo virtuale: SceneGraph, SimpleUniverse, Branches

Come già accennato Java 3D si appoggia sia a OpenGL sia a Direct3D. E' organizzato in due package distinti:

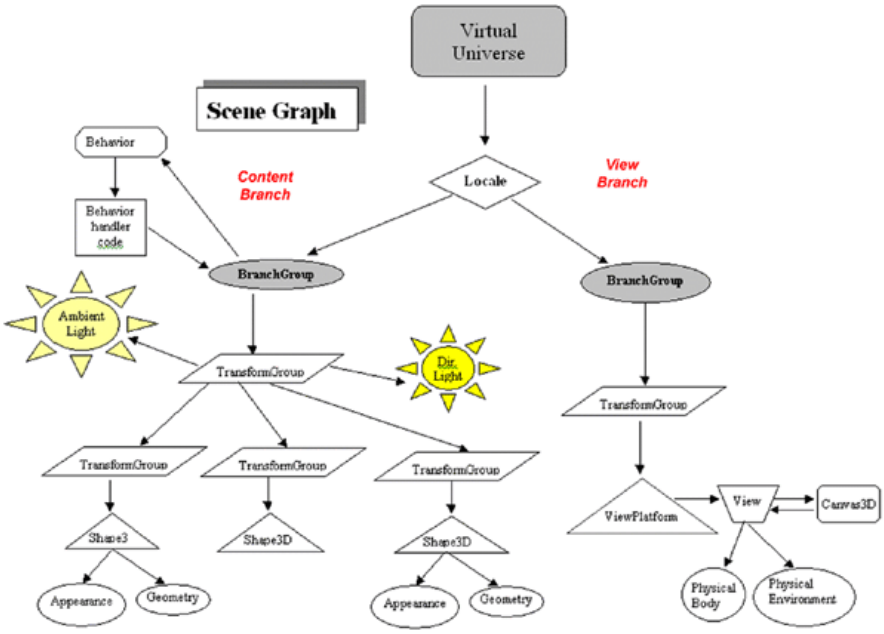
- javax.vecmath: è il package che contiene tutte le classi impiegate per effettuare operazioni e trasformazioni su vettori e matrici. Inoltre fornisce metodi per la rappresentazione di informazioni importanti come la posizione ed il colore.

- javax.media.j3d: si tratta di un package molto vasto che contiene svariate funzioni per la visualizzazione e la gestione della scena tramite scene graph.

Il concetto fondamentale è proprio quello dello scene graph: un albero a cui si appendono tutti gli oggetti della scena 3D intendendo non solo gli elementi da visualizzare ma anche le istruzioni per rappresentarli.

L'albero è radicato nell'Universo Virtuale (Virtual Universe) che, a sua volta, ha uno o più figli Locale, cioè dei luoghi all'interno dell'universo, che rappresentano le scene tridimensionali. Lo scene graph viene percorso dal motore di Java 3D per trovare i dati per creare l'immagine, perciò qualsiasi cosa si desideri abbia un'influenza sulla scena va inserita nello scene graph.
Da ogni nodo Locale discendono due rami principali: il content branch e il view branch (Figura 2.1): il primo contiene gli oggetti da visualizzare, il loro

colore, la loro posizione nello spazio e tutte le altre informazioni rilevanti; il secondo contiene tutto quello che riguarda “il vedere” la scena, ad esempio i dispositivi di input/output o il punto di vista.



(Figura 2.1)

Il disegno tridimensionale viene creato all'interno di un oggetto Canvas3D, che è un'estensione della classe Component, per cui può semplicemente essere aggiunto ad un frame, tuttavia, non è un container quindi non può contenere altri componenti. La Canvas3D viene creata a partire dal view branch e riempita, quindi, con gli elementi del content branch. Gli oggetti tridimensionali immessi nella scena, in effetti, non sono altro che dei rendering delle immagini descritte dai nodi, per cui non è possibile associare degli eventi da catturare e gestire, secondo la normale programmazione a eventi di Java, ma è possibile definire dei behavior, cioè delle reazioni degli oggetti tridimensionali all'interazione dell'utente con la scena.

I passi veri e propri per creare una scena sono molto semplici e li possiamo riassumere così:

- ✓ creare l'universo virtuale che contiene la nostra scena
- ✓ posizionare la vista dell'osservatore in modo che veda l'oggetto
- ✓ creare la struttura che contiene il gruppo di oggetti
- ✓ aggiungere un oggetto
- ✓ compilare la scena (questo passo è opzionale, ma è bene eseguirlo per rendere più efficiente, in termini di velocità di esecuzione, la gestione dell'universo)
- ✓ aggiungere il gruppo di oggetti all'universo

Il codice relativo a questi semplici passi può essere riassunto in:

```
SimpleUniverse universe = new SimpleUniverse(canvas3D);
universe.getViewingPlatform().setNominalViewingTransform();
BranchGroup group = new BranchGroup();
group.addChild(oggetto);
group.compile();
universe.addBranchGraph(group);
```

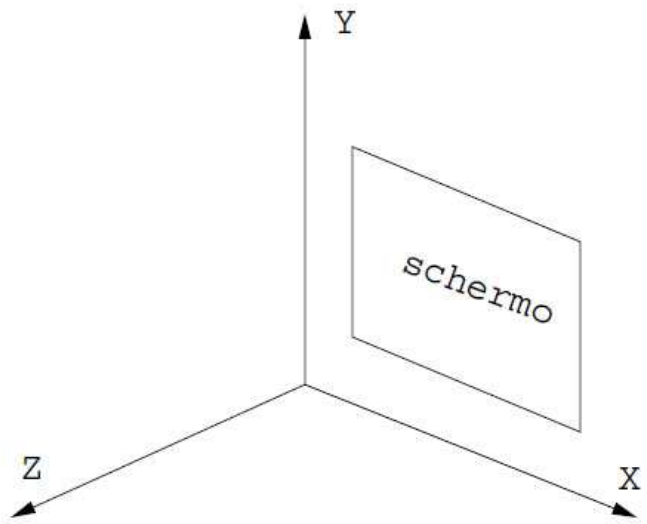
La telecamera è connessa ad una piattaforma di visualizzazione chiamata ViewingPlatform. Tra questa piattaforma e la vista c'è uno strato intermedio chiamo ViewPlatform o piattaforma visiva. SetNominalViewingTransform sposta leggermente indietro il sistema di riferimento permettendo la corretta visualizzazione.

2.2 Oggetti e trasformazioni: BranchGroup, TransformGroup

Come detto, ogni elemento della scena, sia esso un oggetto 3D, una luce o una qualsiasi entità che definisca le modalità di visualizzazione dell'ambiente, non è altro che un nodo del grafo della scena. Due nodi, tuttavia, hanno una

funzione particolare: i BranchGroup, radici di sottoalberi di oggetti con proprietà comuni, e i TransformGroup, che definiscono delle trasformazioni associate a tutti gli elementi del loro sottoalbero.

Vediamo adesso come spostare gli oggetti dell'universo con le operazioni di traslazione e rotazione. Ogni oggetto nell'universo ha tre coordinate x,y,z.
La posizione degli assi è quella classica e gli assi cartesiani X,Y,Z sono così posti:



(Figura 2.2)

A questo punto vediamo come è possibile modificare la posizioni di un oggetto, attraverso la traslazione e la rotazione. E' necessario usare le classi Transform3D e TransformGroup nel seguente modo:

```
Transform3D transf = new Transform3D();
TransformGroup tg = new TransformGroup();
Vector3f pos  = new Vector3f(0.0f,-1.0f,0.0f);
transf.setTranslation(pos);
```

```
tg.setTransform(transf);
tg.addChild(oggetto);
```

In questo modo abbiamo traslato l'oggetto oggetto lungo l'asse Y in basso rispetto allo schermo. Per la rotazione possiamo usare i metodi rotX(Math.PI+rotx), rotY(Math.PI+roty), rotZ Math.PI+rotz) che vengono richiamati su un oggetto di tipo Transform3D.

2.3 Apparenza di un oggetto: Light, Material, Texture

Si noti che senza luci tutto ciò che vedremmo sarebbe solo una schermata nera. Ma questo non è l'unico motivo per cui vengono utilizzate: le luci sono essenziali per aumentare il realismo della scena con ombre e riflessi. Bisogna però stare attenti a non esagerare, perché le luci sono computazionalmente costose e tendono a rallentare la scena.

I tipi di luce che si possono utilizzare sono quattro:

- ✓ Ambient: una luce diffusa che illumina uniformemente tutti gli oggetti, di solito se ne utilizza una sola
- ✓ Directional: raggi paralleli che puntano in una direzione, possono simulare per esempio il Sole
- ✓ Point: i raggi sono emessi da un punto e si irradiano in tutte le direzioni, ad esempio una lampadina
- ✓ Spot: i raggi sono emessi da un punto e si irradiano dentro un cono.

Tutte i tipi di luce hanno in comune metodi per attivarle o disattivarle e per cambiarne il colore. Inoltre un fattore estremamente importante da considerare è che ogni luce deve avere dei limiti ben fissati da un oggetto Bounds. I limiti servono a confinare le luci in una certa area, oltre che per motivi estetici anche per evitare troppi calcoli. Se non si specifica nulla una luce non ha alcun Bound e quindi non illumina nulla. Un'altra caratteristica delle luci è che

possono illuminare solo un certo gruppo di oggetti, anche se di standard illuminano tutto. I vari tipi di luce hanno metodi per decidere la direzione oppure, ad esempio, per l'attenuazione. Un esempio di codice:

```
Color3f colore = new Color3f(0.0f,0.0f,1.0f);
BoundingSphere raggio =
      new BoundingSphere(new Point3d(0.0,0.0,0.0),10.0);
Vector3f direzione = new Vector3f(1.0f,1.0f,1.0f);
DirectionalLight luce =
      new DirectionalLight(colore,direzione);
luce.setInfluencingBounds(raggio);
group.addChild(luce);
```

Di default la luce riflessa da un oggetto è quella che viene proiettata su di esso quando il materiale di cui è composto è bianco. Per cambiare il materiale all'oggetto è sufficiente definire una nuova apparenza creando un oggetto di tipo Appearance, applicare il materiale che si vuole creando un oggetto di tipo Material e settare la nuova apparenza così ottenuta all'oggetto.

Il costruttore di Material accetta 5 argomenti:

- ✓ Ambient color: è il colore riflesso dall'oggetto
- ✓ Emissive color: è il colore emesso dall'oggetto
- ✓ Diffuse color: è il colore dell'oggetto quando è illuminato
- ✓ Specular color: è il colore riflesso dall'oggetto in una particolare direzione, quando è colpito da un fascio di luce
- ✓ Shininess: è l'indice di brillantezza dell'oggetto

```
Color3f white = new Color3f(1.0f, 1.0f, 1.0f);
Color3f black = new Color3f(0.0f, 0.0f, 0.0f);
Material mat = new Material();
mat.setAmbientColor(white);
mat.setEmissiveColor(black);
mat.setDiffuseColor(white);
```

```
mat.setSpecularColor(white);
mat.setShininess(10.0f);
Appearance app = new Appearance();
app.setMaterial(mat);
Box oggetto = new Box(x, y, z, app);
```

Oltre all'aggiunta di un materiale ad un oggetto, è possibile aggiungere una texture cioè una immagine che riveste l'oggetto stesso. Per fare questo è necessario usare la classe TextureLoader ed impostare la risoluzione delle immagini. Si pensi ad esempio a un muro composto da tanti mattoni irregolari: il modo migliore per renderlo sarebbe avere un oggetto per ogni mattone e farlo incastrare, o per lo meno avere un unico oggetto più o meno a forma di parallelepipedo, con un lato irregolare a simulare le varie pietre.

Questi metodi presentano due svantaggi: sono lunghi da generare per il grafico e soprattutto sono complessi da gestire per la macchina. Allora di solito si lascia un parallelepipedo normalissimo (di 6 facce) e lo si ricopre di un disegno che può essere una fotografia oppure un'immagine ottenuta artificialmente che rappresenta il muro. Questa tecnica permette di utilizzare molte meno facce con risultati visivi paragonabili ai due precedenti, almeno fino a quando non ci si avvicina troppo al muro: essere troppo vicini infatti ha lo stesso effetto di fare una zoomata sull'immagine. Per ovviare a questo problema si utilizza una tecnica chiamata mip-mapping: in pratica, quando l'elaboratore deve "inventarsi" dei pixel perchè la texture è troppo vicina, non si limiterà a considerare solo il pixel più vicino al punto di osservazione, ma anche quelli limitrofi in modo da non avere bruschi cambiamenti di colore nella texture, anche se sembra esserci un po' di sfocatura.

La texture può essere ruotata, spostata e scalata, possono essere piazzate sull'oggetto una volta (clamping) oppure ripetersi per coprire l'intera superficie (wrapping).

```
Texture texture =
      new TextureLoader (url, this).getTexture();
Appearance app = new Appearance();
app.setTexture(texture);
TextureAttributes texAttr = new TextureAttributes();
texAttr.setTextureMode(TextureAttributes.MODULATE);
app.setTextureAttributes(texAttr);
Box oggetto =
      new Box(x, y, z, Box.GENERATE_TEXTURE_COORDS, app);
```

2.4 Spostamento della scena: behavior

Per rendere un mondo Java3D più interessante ed utile bisogna aggiungere interazioni ed animazioni. È possibile definire delle reazioni ai movimenti del mouse nella scena grazie alla classe MouseBehavior o alla pressione dei tasti di una tastiera grazie alla classe KeyNavigatorBehavior, o alle specifiche interazioni su un oggetto 3D grazie alla classe PickMouseBehavior, ma il meccanismo non è semplicissimo ed è sicuramente più macchinoso di quanto non accada con la normale gestione degli eventi.

L'interazione tramite mouse prevede diverse classi:

- ✓ MouseRotate per la rotazione
- ✓ MouseTraslate per la traslazione
- ✓ MouseZoom per avvicinarsi o allontanarsi da un oggetto

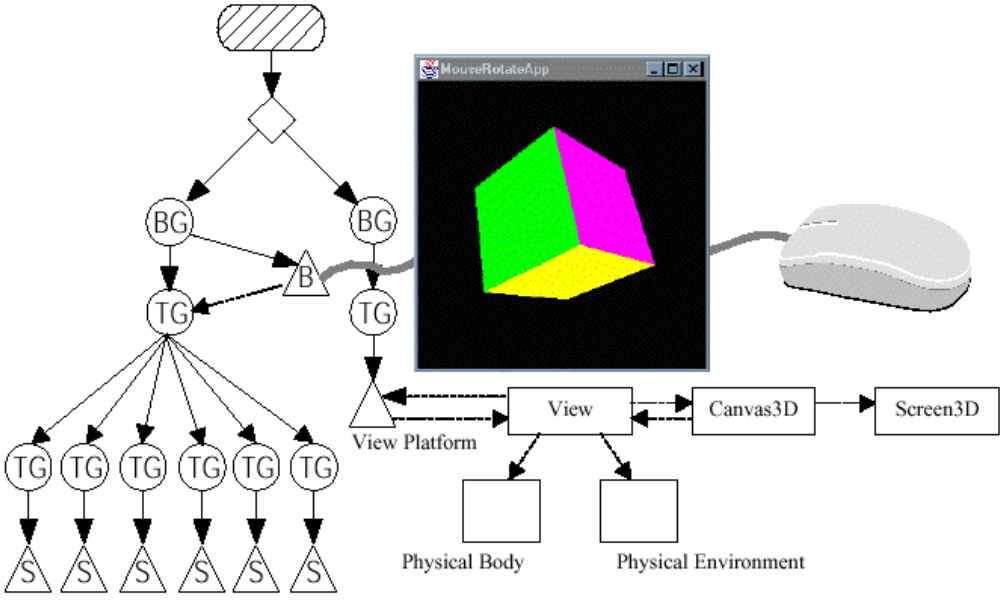
Le scelte relative ai componenti tridimensionali di una scena sono dovute anche al fatto che, chiaramente, i cambiamenti a runtime di un ambiente tridimensionali sono ben più “pesanti” di quelli all’interno di applicazioni 2D.

Limitare e standardizzare i movimenti e rendere più "statici" gli oggetti alleggerisce il lavoro in tempo di esecuzione del motore 3D.

Vediamo come viene realizzato lo spostamento della scena nella nostra applicazione:

```
// Associo al gruppo di trasformazione della scena la
// posizione del sistema di riferimento
TransformGroup all =
universe.getViewingPlatform().getViewPlatformTransform();

// Definisco la classe che si occuperà dello spostamento
// della scena alla pressione dei tasti della tastiera
// passando come parametro il sistema di riferimento
KeyNavigatorBehavior keyNav = new KeyNavigatorBehavior(all);
keyNav.setSchedulingBounds(
      new BoundingSphere(new Point3d(),1000.0));
group.addChild(keyNav);

// Definisco la classe che si occuperà della rotazione
// della scena in base allo spostamento del mouse
MouseRotate myMouseRotate =
      new MouseRotate(MouseBehavior.INVERT_INPUT);
myMouseRotate.setTransformGroup(all);
myMouseRotate.setSchedulingBounds(
      new BoundingSphere(new Point3d(),1000.0));
group.addChild(myMouseRotate);

// Definisco la classe che si occuperà dello spostamento
// della scena in base allo spostamento del mouse
MouseTranslate myMouseTranslate =
      new MouseTranslate(MouseBehavior.INVERT_INPUT);
myMouseTranslate.setTransformGroup(all);
myMouseTranslate.setSchedulingBounds(
      new BoundingSphere(new Point3d(),1000.0));
group.addChild(myMouseTranslate);
```

Il behavior viene attivato non appena si verificano gli eventi a cui deve rispondere. Acquisisce l'input e lo trasforma in "segnale" per il thread Java3D il quale non farà altro che modificare le matrici di rotazione e traslazione in accordo alla direzione che l'utente vuole impartire. Come possiamo vedere nello schema in figura 2.4, il mouse (o la tastiera) definiscono una serie di trasformazioni. Tali trasformazioni avranno ripercussione sulla View Platform che passiamo come parametro al trasformgroup.



(Figura 2.4)

CAPITOLO 3

LA BASE DI DATI

3.1 Java e JDBC

Uno degli strumenti offerti dalla tecnologia Java è l'interfaccia JDBC, un connettore per database che consente l'accesso alle basi di dati da qualsiasi programma scritto in linguaggio Java, indipendentemente dal tipo di DBMS utilizzato. È costituita da una API, situata nel package java.sql, che serve ai client per connettersi ad un database. Fornisce metodi per interrogare e modificare i dati. È orientata ai database relazionali ed è Object Oriented. Tutto ciò implica garantisce ad un'applicazione Java, di per sé già indipendente dalla piattaforma, anche l'indipendenza dal database engine.

Per poter accedere al nostro database l'applicazione deve:

- ✓ Caricare un driver
- ✓ Aprire una connessione con il database
- ✓ Creare un oggetto Statement per interrogare il database
- ✓ Interagire con il database
- ✓ Gestire i risultati ottenuti

Vediamo qui un esempio di esecuzione di una query:

```
Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection(url,user,pass);
Statement db = con.createStatement();
db.execute(query);
```

Il metodo execute esegue istruzioni di cancellazione, modifica ed inserimento di record. Per quando riguarda l'utilizzo dell'istruzione "select" è necessario utilizzare un'altra funzione che restituisce l'insieme degli elementi desiderati

in un struttura di tipo ResultSet che conterrà le righe e gli attributi della nostra ricerca.

```
ResultSet rs = db.executeQuery(query);
while (rs.next()) {
   int id = rs.getInt("id");
   String nome = rs.getString("nome");
}
```

3.2 Perchè utilizzare un database?

Una base di dati è una grande collezione di dati. Le ragioni per cui è meglio utilizzare un database per conservare i propri dati sono tante. I file di testo sono in un modo o in un altro visibili via browser, e per questo non sono sicuri, i database, invece, sono per lo più accessibili soltanto ai legittimi proprietari. Inoltre interrogare un file di testo può essere più difficile che interrogare una base di dati.
Memorizzare i dati in un DBMS offre anche altri vantaggi:

- Indipendenza dei dati ed efficienza negli accessi: i programmi sono indipendenti dai dettagli di rappresentazione e memorizzazione dei dati inoltre un DBMS ha un efficiente meccanismo di memorizzazione e reperimento dei dati.

- Riduzione del tempo di sviluppo delle applicazioni: il DBMS fornisce importati funzioni che sono richieste dalle applicazioni come il controllo della concorrenza e il ripristino. Inoltre l'implementazione del codice specifico è facilitato dagli strumenti sviluppati dai fornitori dei sistemi di amministrazione di molte basi di dati.

- Integrità e sicurezza dei dati: il DBMS può garantire il controllo degli accessi da parte di utenti non autorizzati e non permettere l'aggiornamento di dati che violano i vincoli imposti.

- Amministrazione dei dati: essendo la raccolta di dati ripartita tra più utenti, un amministratore centrale facilita l'attività di gestione e manipolazione dei dati. Un buon DBMS svolge attività periodiche di backup ed ottimizzazione.

- Concorrenza negli accessi e ripristino da crash: un DBMS organizza gli accessi concorrenti ai dati in modo che ogni utente possa pensare di essere l'unico ad utilizzarli e aggiorna i dati costantemente per ogni utente assicurando che le transazioni concorrenti non entrino in conflitto. Se il sistema va in crash, il DBMS ripristina solo le transazioni complete in modo tale che lo stato del database rimanga consistente.

- Indipendenza logica dei dati: gli utenti sono protetti dai cambiamenti nella struttura logica dei dati, cioè cambiamenti nella scelta delle relazioni da memorizzare. E' possibile aggiungere nuove relazioni, nuovi attributi e nuove associazioni senza cambiare il codice già esistente e senza che l'utente si accorga di nulla.

Per finire, le aziende hanno spesso la necessità di disporre di una grande quantità di dati relativi all'azienda stessa e all'ambiente in cui operano. L'utilizzo intelligente di questi dati può costituire un importante fattore di sviluppo.

3.3 Normalizzazione e dipendenze funzionali

La normalizzazione è un procedimento volto all'eliminazione della ridondanza e del rischio di incoerenza dal database. Esistono vari livelli di normalizzazione (forme normali) che certificano la qualità dello schema del database. Questo processo si fonda su un semplice criterio: se una relazione presenta più concetti tra loro indipendenti, la si decompone in relazioni più piccole, una per ogni concetto. Questo tipo di processo non è sempre applicabile in tutte le tabelle, dato che in alcuni casi potrebbe comportare una perdita d'informazioni. Quando una relazione non è normalizzata si presta a comportamenti poco desiderabili soprattutto durante l'esecuzione di aggiornamenti questo poiché, una o più informazioni, sono sicuramente duplicate.

Altra particolarità molto importante che descrive i legami tra gli attributi di una relazione sono le dipendenze funzionali. Quando un attributo A determina un altro attributo B, cioè quando il valore di un attributo è unico possiamo dire che esiste una dipendenza funzionale tra A e B. Non sempre è facile individuare le dipendenze funzionali perché esistono diversi modi di esprimere la stessa dipendenza, ed esistono espressioni simili tra loro che esprimono dipendenze diverse. Sono proprio le dipendenze funzionali a permetterci la decomposizione. Per decomporre, infatti, si crea una relazione per ogni gruppo di attributi coinvolti in una dipendenza funzionale e si verifica che ogni relazione contenga una chiave.

3.4 Implementazione della base di dati dell'applicazione

Prima di passare alla struttura della base di dati vera e propria è bene fermarsi a riflettere sulle funzioni che dovrà supportare e sulle problematiche correlate cercando di progettare la base di dati quanto meglio possibile. Un errore di progettazione può ripercuotersi sull'intero sviluppo del progetto dando non poche noie anche per possibili sviluppi futuri. Una base di dati ben progettata risulta invece più efficiente e semplifica lo sviluppo anche nell'evenienza di cambiamenti del codice.

Il primo passo nella progettazione di una base di dati viene effettuato studiando approfonditamente il problema, tramite interviste a chi ha commissionato l'applicazione e agli operatori chi ne faranno utilizzo. In questo caso possiamo solo tener conto delle funzionalità della nostra applicazione. Quando sono chiari i requisiti dell'applicazione e tutti i vincoli del problema verrà fuori una breve ma essenziale descrizione del mini-mondo da rappresentare.

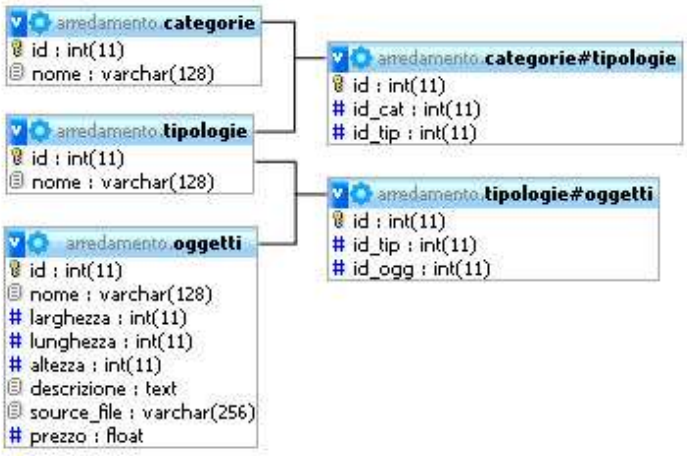
Una possibile descrizione del nostro mini-mondo potrebbe essere questa:
L'applicazione mostrerà agli utenti informazioni sui prodotti che saranno suddivisi in base alle loro caratteristiche e ai loro utilizzi. Sarà necessario che ogni prodotto sia caratterizzato da nome, dimensione, descrizione e prezzo per poter essere collocato correttamente e per dare all'utente un resoconto della spesa. Per semplificare la ricerca sarà necessario associare i prodotti a determinate categorie mentre per identificare prodotti dello stesso tipo sarà necessario raggrupparli in tipologie.

Cercando di capire quello di cui abbiamo bisogno vedremo subito che ogni oggetto dovrà possedere i seguenti attributi:

- ✓ un valore per identificarlo univocamente
- ✓ un nome per essere riconosciuto dall'utente
- ✓ una dimensione in termini di larghezza, lunghezza ed altezza
- ✓ una descrizione per capirne le caratteristiche essenziali
- ✓ un modello grafico per un'anteprima 2D
- ✓ un modello grafico per un'anteprima 3D
- ✓ un prezzo per il resoconto finale

Ogni oggetto dovrà essere raggruppato in determinate tipologie di oggetti identificate anch'esse univocamente e mostrate all'utente con un nome simbolico. A loro volta queste tipologie dovranno essere organizzate in categorie che ne mostrino l'uso quotidiano e ne semplifichino la ricerca.

La struttura della base di dati sarà la seguente:



(Figura 3.4.1)

Come possiamo notare le tabelle principali sono essenzialmente "categorie", "tipologie" ed "oggetti" che conterranno i dati veri e propri mentre le tabelle "categorie#tipologie" e "tipologie#oggetti" ci servono per le associazioni.

Possiamo vedere un esempio pratico di come lavorerà la nostra applicazione in risposta ai comandi dell'utente mostrando innanzitutto il contenuto delle tabelle principali:

CATEGORIE

id	nome
1	Cameda da letto
2	Cucina
3	Studio
4	Soggiorno

TIPOLOGIE

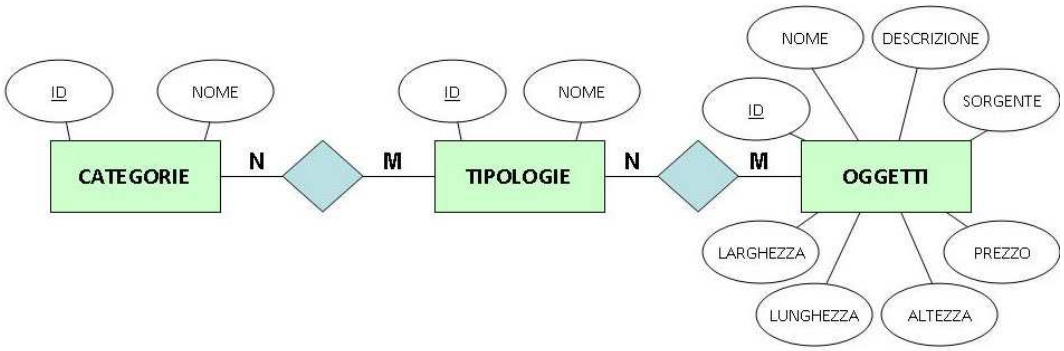
id	nome
1	Letti
3	Comodini
4	Tavoli
5	Elettrodomestici
6	Divani
7	Scrivanie

OGGETTI

id	nome	larghezza	lunghezza	altezza	descrizione	source_file	prezzo
1	Letto di prova	190	145	40	descrizione letto di prova	letto01	249.9
2	Comodino di prova	45	42	64	descrizione comodino di prova	comodino01	49.95
4	Tavolo di prova	100	100	70	descrizione tavolo di prova	tavolo01	109.6
5	Tavolo di prova 2	190	160	73	descrizione tavolo di prova 2	tavolo02	299
6	Frigorifero di prova	70	66	180	descrizione frigorifero di prova	frigorifero01	329.7
7	Scrivania di prova	110	45	100	descrizione scrivania di prova	scrivania01	119.9
8	Divano di prova	150	90	100	descrizione divano di prova	divano01	690
9	Cucina di prova	55	65	110	descrizione cucina di prova	cucina01	240.5
10	Letto di prova 2	190	85	40	descrizione letto di prova 2	letto02	119.6

(Figura 3.4.2)

In primo luogo un qualsiasi utente che si cimenterà nella progettazione della stanza avrà sicuramente in mente cosa dovrà contenere quello spazio. Se si tratterà di una camera da letto avrà bisogno di un letto, di un comodino o di un armadio, se si tratterà di un soggiorno avrà bisogno di un divano o di un tavolo, e via dicendo. La tabella "categorie#tipologie" non fa altro che realizzare queste semplici associazioni: a determinate categorie corrisponderanno determinate tipologie di oggetti. Da notare che un tavolo, ad esempio, può appartenere sia alla categoria "Cucina" sia alla categoria "Soggiorno". La nostra base di dati dovrà tener conto di ciò permettendo di realizzare associazioni N-M dove ad N categorie corrisponderanno M oggetti



(Figura 3.4.3)

Una volta che l'utente avrà identificato la categoria dovrà capire effettivamente di cosa ha bisogno. Se, ad esempio, ha bisogno di un letto questo potrà essere di vari formati (una piazza, una piazza e mezzo, due piazze), il materasso potrà essere di diversi materiali (in lattice, ad acqua) come anche la rete (tavole di legno, molla). La tabella "tipologie#oggetti" associa ad una tipologia un insieme di oggetti diversi, ma con caratteristiche comuni. In questo caso non sarà necessario utilizzare associazioni N-M, ma basterà un associazione di tipo 1-N dove ad una tipologia corrisponderanno N oggetti che apparterranno ad una ed una sola tipologia. In pratica, però, ho deciso di utilizzare un associazione N-M che da alla struttura una certa flessibilità. In questo modo un oggetto può essere associato a più tipologie, ad esempio, un "divano-letto" potrebbe esser associato sia alla tipologia "Letti" sia alla tipologia "Divani".

Tornando alla struttura della nostra base di dati (Figura 3.4.1) ogni attributo possiede un dominio, ovvero ad esso è possibile assegnare solo un ben preciso insieme di valori. Il dominio di un attributo corrisponde ai tipi di dati elementari:

- ✓ Stringhe di caratteri
- ✓ Valori numerici

- ✓ Valori temporali (date, ore, ...)
- ✓ Valori booleani (vero o falso)
- ✓ Enumerazioni (una lista di valori validi)

Una volta assegnato un certo dominio ad un attributo, esso potrà assumere solo i valori consentiti dal dominio ed eventualmente il valore speciale NULL. Il valore NULL sta a significare che per un particolare elemento dell'entità il valore dell'attributo in questione non è noto oppure non è applicabile.
Si può notare anche la presenza di un altro tipo di attributo che ha un significato speciale all'interno delle entità: l'attributo chiave primaria. Su ogni tabella è presente un attributo chiamato “id” che ci permette di identificare una tupla (una qualsiasi riga della tabella) in modo univoco. Ma perché utilizzare un ID invece che il nome stesso della categoria o della tipologia? E’ probabile che, in un prossimo futuro, alcuni nomi verranno modificati e se non ci fossero gli ID implicherebbe il dover cambiare nome su più tabelle, in questo modo il nome è identificato in una sola tabella e basterà modificare quella per avere una modifica globale inoltre c’è da considerare anche il fatto che un nome è composto da molti più caratteri rispetto ad un valore numerico e modificare questi valori è sicuramente più efficiente.

CAPITOLO 4

REALIZZAZIONE DELL'APPLICAZIONE

4.1 Introduzione

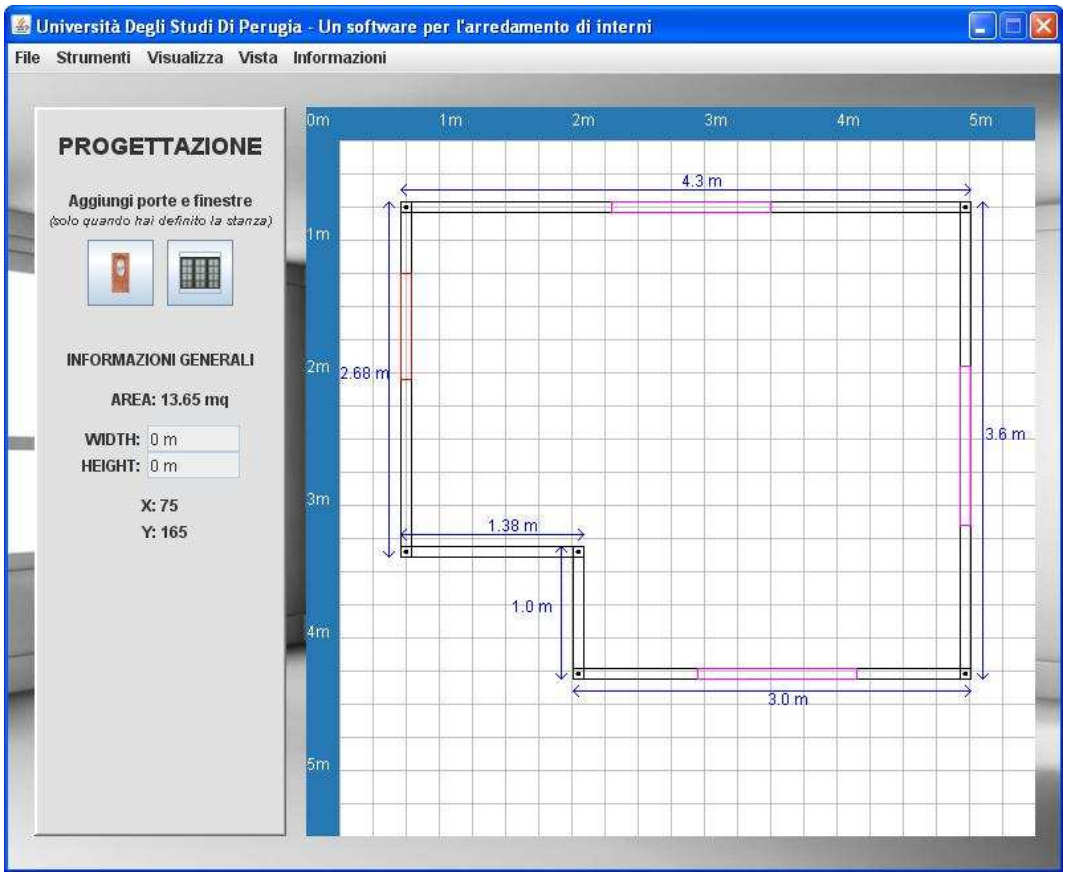
L'applicazione è suddivisa in vari moduli: la classe principale "Main", che estende la classe JFrame, si occupa della costruzione della finestra; la barra dei menu è costituita da 4 menu principali:

- File: ci permette di creare un nuovo progetto, di importarne uno esistente o di esportare il progetto corrente nel formato apposito.
- Strumenti: offre le funzionalità per la progettazione della stanza, una per l'inserimento dell'arredamento, una per il resoconto spese ed infine una per la vista tridimensionale.
- Visualizza: dà la possibilità all'utente di personalizzare la propria area di lavoro scegliendo se visualizzare i righelli, la griglia, le dimensioni delle pareti e infine la trasparenza per la vista 3D.
- Vista: permette di cambiare il punto di vista nella visualizzazione 3D.

Ad ogni sottomenu è associata una classe "esecutoreEventi" che identifica l'azione e ne esegue i relativi compiti. Ad esempio al click del sottomenu apri apparirà un openDialog che permetterà la scelta del file da importare; generalizzando la classe presenta questa struttura:

```
class esecutoreEventi implements ActionListener {
      public void actionPerformed (ActionEvent e) {
            if (e.getSource().equals(elemento)) {
                  esegui le seguenti azioni…
            }
      }
}
```

Ecco qui come si presenta l'interfaccia iniziale:



(Figura 4.1)

Ogni sottomenù di "Strumenti" visualizzerà un JPanel della dimensione dell'intera finestra. Ognuno di questi pannelli viene istanziato all'avvio dell'applicazione e durante l'esecuzione solo un pannello alla volta viene reso visibile. Questo va, sicuramente, a discapito dello spazio in memoria ma velocizza di molto l'esecuzione e migliora l'usabilità dell'applicazione poiché le finestre non devono essere continuamente distrutte e le informazioni non vanno mai perse.

Ciò è di particolare importanza soprattutto per quanto riguarda la vista 3D poiché, come già detto, risulta molto pesante. Questa, infatti, viene ridisegnata solo quando necessario: se ad esempio aggiungiamo un nuovo oggetto alla stanza la variabile globale refreshUniverse verrà impostata a true e quando

andremo nella schermata "Vista 3D" indicherà che la scena dovrà essere ridisegnata altrimenti verrà mantenuta la scena precedente.

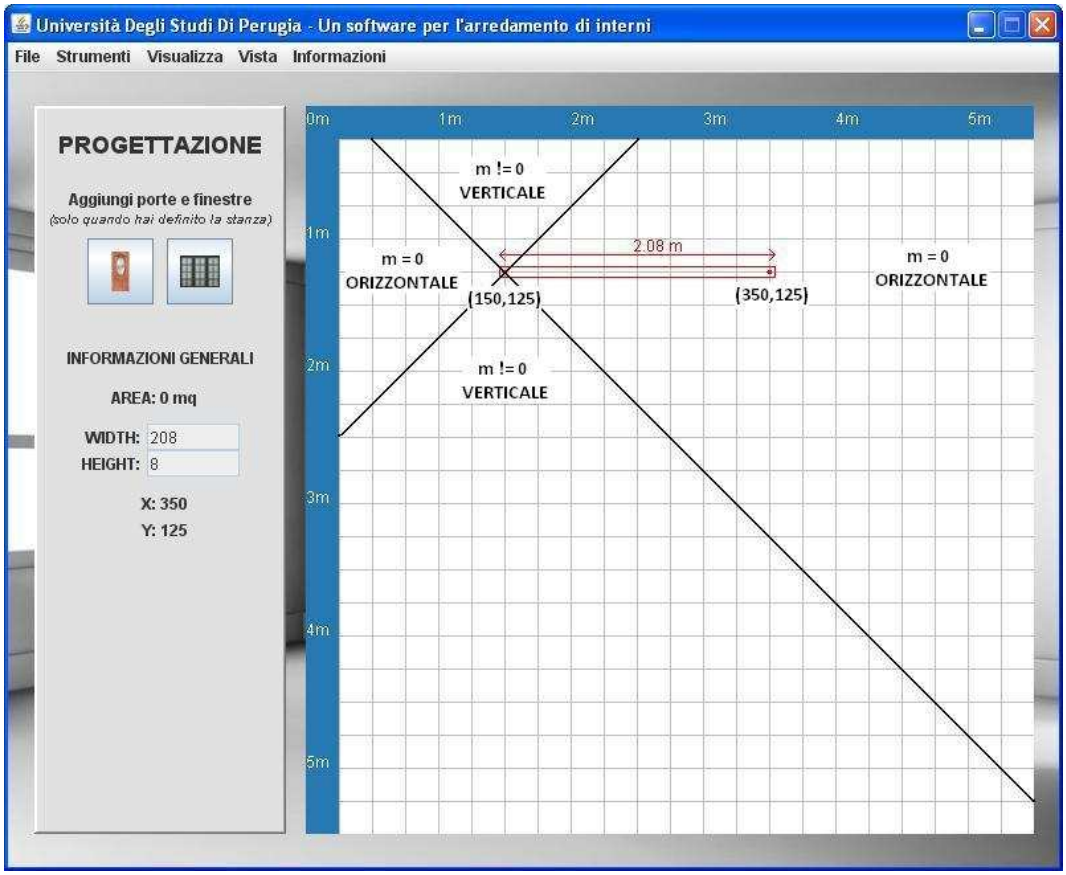
4.2 Progettazione della planimetria

Come possiamo vedere nella figura 4.1 abbiamo una colonna a sinistra che ci mostra alcune informazioni utili riguardanti la stanza e uno spazio di lavoro a destra dove disegnare le pareti ed indicare la posizione di porte e finestre. Ogni parete è un oggetto della classe "Parete" le cui variabili essenziali sono:

```
boolean finita;
boolean selezionata;
boolean orizzontale;

// identificativo, coefficiente angolare, dimensione,
altezza, spessore della parete
int id, m, dim, h, sp;
// punto iniziale e finale (anche delle linee informative e
della stringa che mostrano la dimensione)
int xi, xf, xline_i, xline_f, xstring;
int yi, yf, yline_i, yline_f, ystring;
// vettori delle coordinate dei vertici
int x[] = new int[4], y[] = new int[4];
```

Quando si clicca in un punto dello spazio di lavoro inizia la fase di creazione della parete richiamando il costruttore e settando come punto iniziale (xi,yi), la posizione corrente del mouse rispetto all'area di lavoro. Fatto ciò viene calcolato il coefficiente angolare rispetto alle posizioni successive del mouse per definire se si tratta di una parete orizzontale o verticale.
Come possiamo vedere nella figura 4.2 dopo aver definito il punto iniziale (xi,yi) è come se si formassero quattro quadranti virtuali dove:

```
if (xf - xi!=0) m = (yf - yi) / (xf - xi);
  else m = 1;
```



(Figura 4.2)

Al successivo click del mouse viene definito il punto finale (xf,yf) e generati i 4 vertici della parete negli array x[] e y[]. Fatto questo si continua con la costruzione della parete successiva che avrà come punto iniziale il punto finale della parete precedente ma con un ulteriore vincolo: se la parete precedente era orizzontale la nuova parete potrà essere solo verticale e viceversa.

Si continua con la stessa procedura fino a quando non saremo nei pressi della parete iniziale e sarà necessario chiudere la stanza facendo coincidere le coordinate finali dell'ultima parete con le coordinate iniziali della prima parete. Questo presenta non poche problematiche poiché dovranno essere eseguite una serie di trasformazioni e controlli anche su pareti non direttamente interessate. Utilizzando la procedura checkPOI() (POI: Point Of Interest) controllo se nei pressi del mio ultimo punto non sia presente la parete

iniziale se ciò si verifica dovrò fare un ulteriore controllo: se la parete finale e quella iniziale sono dello stesso tipo (due parete orizzontali o due pareti verticali) dovrò fonderle in modo da creare un'unica parete orizzontale o verticale, se la parete finale e quella iniziale non sono dello stesso tipo vengono ricalcolati i nuovi vertici per far combaciare perfettamente le due pareti. Questo purtroppo non basta poiché la parete precedente a quella finale dovrà essere ridimensionata per avere la stessa coordinata x o y di quella iniziale.

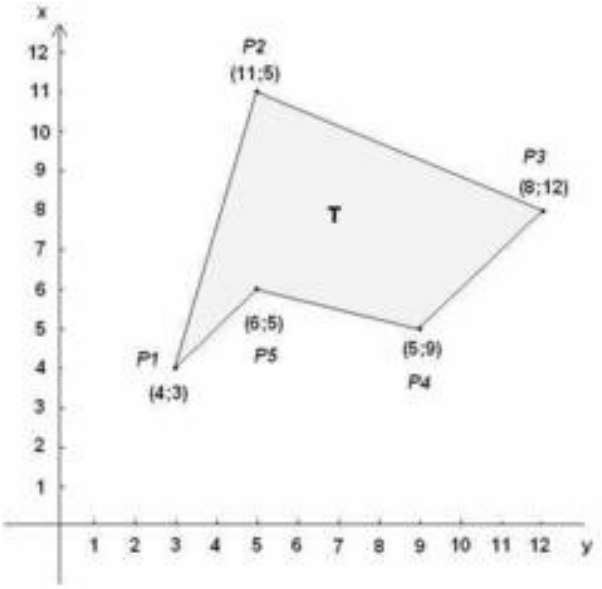
4.3 Calcolo dell'area di un poligono irregolare

Terminata la creazione della stanza mi è sembrato opportuno calcolare l'area della stanza per dare un'idea delle dimensioni. In un primo momento ho cercato di suddividere il problema cercando di calcolare le aree delimitate dalle proiezioni orizzontali o verticali dei lati. Questa strada, però, è risultata non applicabile poiché sarebbe stato necessario memorizzare molte più informazioni. Per risolvere il problema è stato necessario utilizzare la **formula di Shoelace**. Con la formula di Shoelace è possibile calcolare l'area di un poligono con n vertici ordinati aventi coordinate cartesiane $(x_i; y_i)_{i=1\ldots n}$ con la convenzione che (x[n+1];y[n+1]) = (x[1];y[1]) nel modo seguente:

$$A = \frac{1}{2}\left|\sum_{i=1}^{n} x_i y_{i+1} - \sum_{i=1}^{n} x_{i+1} y_i\right|$$

Ecco un esempio di calcolo dell'area con la formula di Shoelace. L'area del poligono in figura 4.3 di coordinate P1(4,3) P2(11,5) P3(8,12) P4(5,9) P5(6,5) sarà:

A = | 4 x 5 + 11 x 12 + 8 x 9 + 5 x 5 + 6 x 3 – 11 x 3 – 8 x 5 – 5 x 12 – 6 x 9 – 4 x 5 | / 2 = 60 / 2 = 30

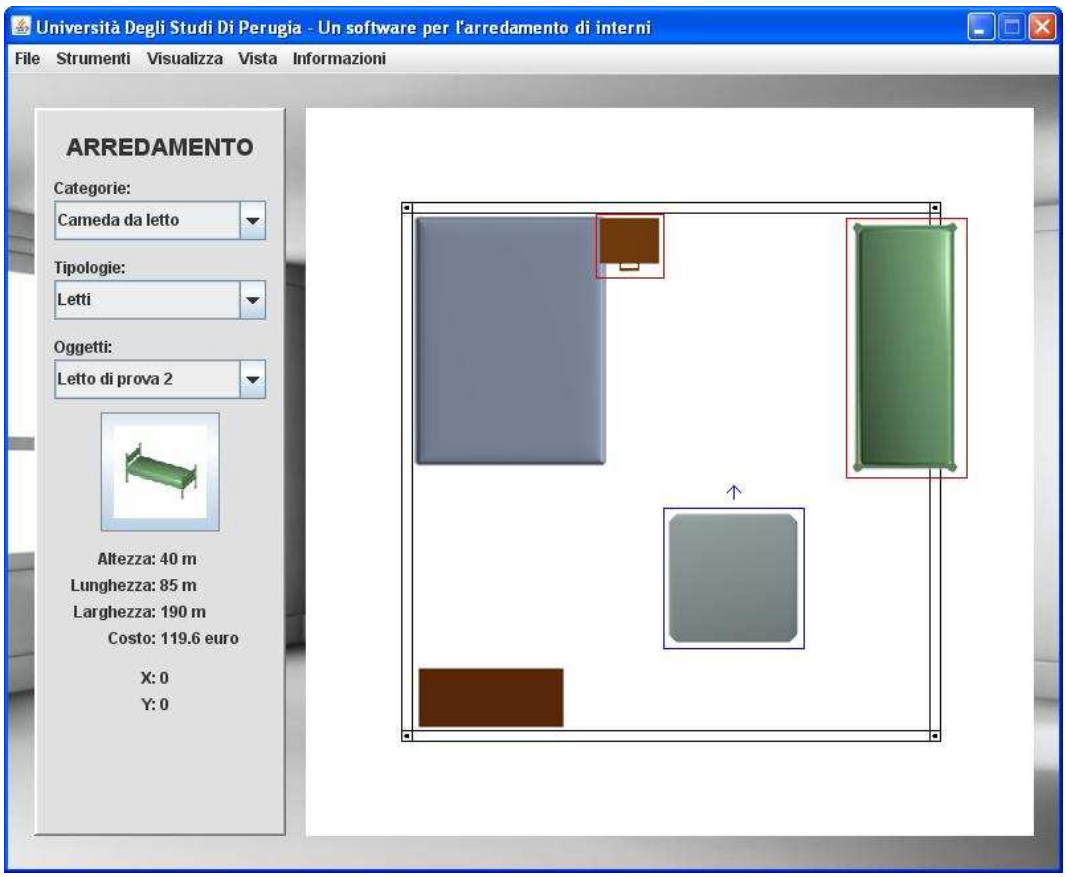


(Figura 4.3)

4.4 Collisioni tra oggetti

Il problema della rilevazione di collisioni è un problema molto comune nella realizzazione di software grafici e videogame. Ci sono molte tecniche per realizzare questo controllo, ma l'utilizzo di una tecnica specifica dipende dal tipo di applicazione che si sta realizzando. Fattori importanti per determinare tale scelta potrebbero essere quanto spesso avvengono le collisioni, quali sono le collisioni che hanno più importanza o con quanta precisione vogliamo rilevare la collisione.

Nel nostro caso esistono sostanzialmente due tipi di collisione: la collisione tra due oggetti della stanza e la collisione tra un oggetto e una parete. Ecco qua un esempio di come vengono visualizzati i due tipi di collisione:



(Figura 4.4)

Gli oggetti contornati di rosso sono gli oggetti in collisione. Questo è il funzionamento: quando si clicca sopra un oggetto questo viene selezionato (contorno blu) e può essere ruotato o spostato. Nel momento in cui svolgiamo una di queste azioni, dovrà essere effettuato un controllo: se si trova una collisione l'oggetto viene evidenziato anche nel caso in cui venga poi deselezionato per permettere all'utente di effettuare gli opportuni cambiamenti. Poiché i test per verificare una collisione possono essere molto costosi è necessario cercare dei compromessi. Un esempio è il raggruppamento di oggetti: un tavolo con delle sedie attorno potrebbe esser considerato un unico oggetto e racchiuso in un univo volume che lo contiene, certo in questo modo si perde molto in precisione ma si guadagna molto più in prestazioni. Un altro esempio potrebbe essere il controllo delle collisioni nel caso di figure irregolari, non è necessario che venga considerato ogni lato e spigolo della

figura ma si potrebbe racchiudere la figura in una figura più semplice in grado di contenerla come ad esempio un cerchio o un rettangolo, oppure nel caso di un sistema (x,y,z) una sfera o un parallelepipedo.

Adesso vediamo in dettaglio come è stata implementato questo controllo:
innanzitutto ad ogni spostamento o rotazione dell'oggetto viene richiamata la procedura updateCoord(int x, int y) che svolge una serie di funzioni: per prima cosa setCoord() imposta la nuova posizione, la variabile refreshUniverse viene impostata a true per identificare che l'universo ha subito modifiche e che quindi va ridisegnato e infine viene impostata la variabile collisione = checkColl() che restituirà true in caso di collisione o false altrimenti.
Queste sono le operazioni che svolte la funzione checkColl():

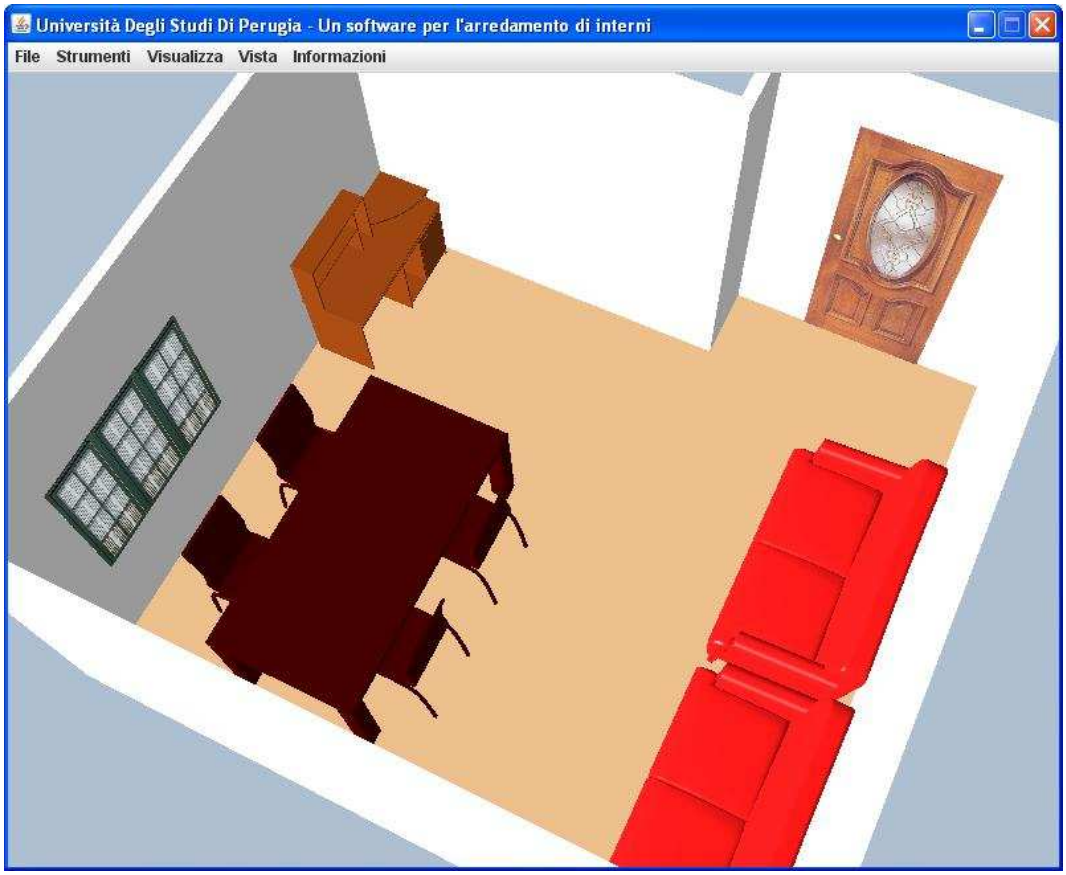
```
public boolean checkColl() {

  // Rilevazione della collisione con una parete
  for (Parete elem : Var.stanza.parete)
    for (int i = x[0]; i <= x[1]; i++)
      for (int j = y[0]; j <= y[2]; j++)
        if (i >= xm && i <= xM && j >= ym && j <= yM) return true;

    // Rilevazione della collisione con un altro oggetto
    for (Oggetto ogg : Var.stanza.oggetto)
      if (ogg!=this) {
        for (int i = x[0]; i <= x[1]; i++)
          for (int j = y[0]; j <= y[2]; j++)
            if (i >= ogg.x[0] && i <= ogg.x[1] &&
                j >= ogg.y[0] && j <= ogg.y[2]) return true;
      }

  return false;
}
```

Come possiamo vedere il costo per ciascun controllo è molto elevato poiché abbiamo almeno tre cicli nidificati. Il primo ciclo scorre la lista delle pareti, il secondo la coordinata x e il terzo la coordinata y, infine viene effettuato il seguente controllo: se il punto i è maggiore della xm (m sta per minimo ed è la coordinata x più piccola dell'oggetto) e minore della xM (M sta per massimo ed è la coordinata x più grande dell'oggetto), se il punto j è maggiore della ym (y più piccola) e j è minore della yM (y più grande) allora vuol dire che c'è almeno una delle due coordinate "dentro" il nostro oggetto è quindi c'è una collisione. Stessa cosa con tutti gli altri oggetti diversi dall'oggetto stesso.
Se non è mai stato restituito un true alla fine verrà restituito false.
In poche parole viene esaminata l'intera lunghezza e larghezza della parete e controllato se tra il minimo e il massimo dell'oggetto vi sia un riscontro.

4.5 Importazione modelli 3D

Esistono moltissimi formati 3D (obj, vrml, 3ds, max, ecc.) e possono sempre esserne inventati di nuovi (per esempio X3D) per questo motivo Java3D mette a disposizione un'interfaccia generica per la creazione e l'utilizzo di caricatori di file. Per caricare singoli oggetti o interi ambiente 3d prodotti con strumenti quali Anim8or, ArtOfIllusion, Maya, 3DStudio Max, Lightwave, e tanto altro ancora si possono usare i Loader. I loader sono degli utili accessori di Java3D che, come dice il nome, "caricano" i modelli. Java3D contiene un Loader predefinito che carica oggetti Wavefront da file in formato obj. Per utilizzarlo dobbiamo importare due package:

- import com.sun.j3d.loaders.*;
- import com.sun.j3d.loaders.objectfile.*;



(Figura 4.5)

Tutti i prodotto sopra menzionati sono in grado di salvare un modello nel formato obj ma ognuno di questi ha formati specifici che non permettono la modifica con altri prodotti. Per questo motivo, nel caso si voglia rendere disponibile il proprio modello ad altre persone, è preferibile l'utilizzo di questo formato. Tale formato però presenta delle limitazioni in quanto non supporta le animazioni, presenta un limite al numero di poligoni utilizzabili, è di difficile reperimento poiché chi crea gli oggetto con programmi professionali difficilmente ne vuole permettere il riutilizzo ed infine alcuni software non leggono correttamente informazioni su texture complesse, materiali e luci.

Per questi motivi ho preferito utilizzare un formato un po' più avanzato, il 3DS la quale mi ha permesso di trovare molti modelli disponibili da poter adattare alla mia applicazione.

Vediamo ora in generale cosa bisogna fare per poter caricare ed utilizzare correttamente i nostri modelli. Una classe Loader legge un modello da un file e ne crea la sua rappresentazione Java3D. Le operazioni da svolgere sono semplici:

- ✓ trovare un Loader compatibile con il formato del file
- ✓ caricare il file ed assegnare il risultato ad un oggetto Scene
- ✓ inserisci lo Scene nello scene graph

Nello specifico vediamo il package com.mnstarfire.loaders3d.Loader3DS utilizzato per i modelli 3DS e il codice relativo per la visualizzazione degli oggetti:

```
public void addOggetto(int x, int y, double rot, String source) {

  TransformGroup tg = new TransformGroup();
  Transform3D trans = new Transform3D();

  // Ruoto l'oggetto nel verso scelto al momento dell'inserimento
  // nella planimetria e lo posiziono nella posizione desiderata
  // infine aggiungo la trasformazione al TransformGroup.
  trans.rotY(-rot);
  trans.setTranslation(new Vector3f(x, 0, y));
  tg.setTransform(trans);

  try {
    // Carico il modello con il Loader3DS specificando il
    // percorso e lo memorizzo nella scena
    Loader3DS loader = new Loader3DS();
    Scene scene = loader.load(
      getClass().getResource("oggetti/"+source+".3ds"));

     // Aggiungi la scena al BranchGroup alla quale applicherò
     // la trasformazione ed infine lo aggiungo all'albero
      BranchGroup sceneGroup = scene.getSceneGroup();
```

```
        tg.addChild(sceneGroup);
        group.addChild(tg);
    } catch(Exception ex) {
      System.out.println("<<< EXCEPTION >>>
       Impossibile caricare l'oggetto 3D");
    }

}
```

4.6 Matrici di traslazione e rotazione

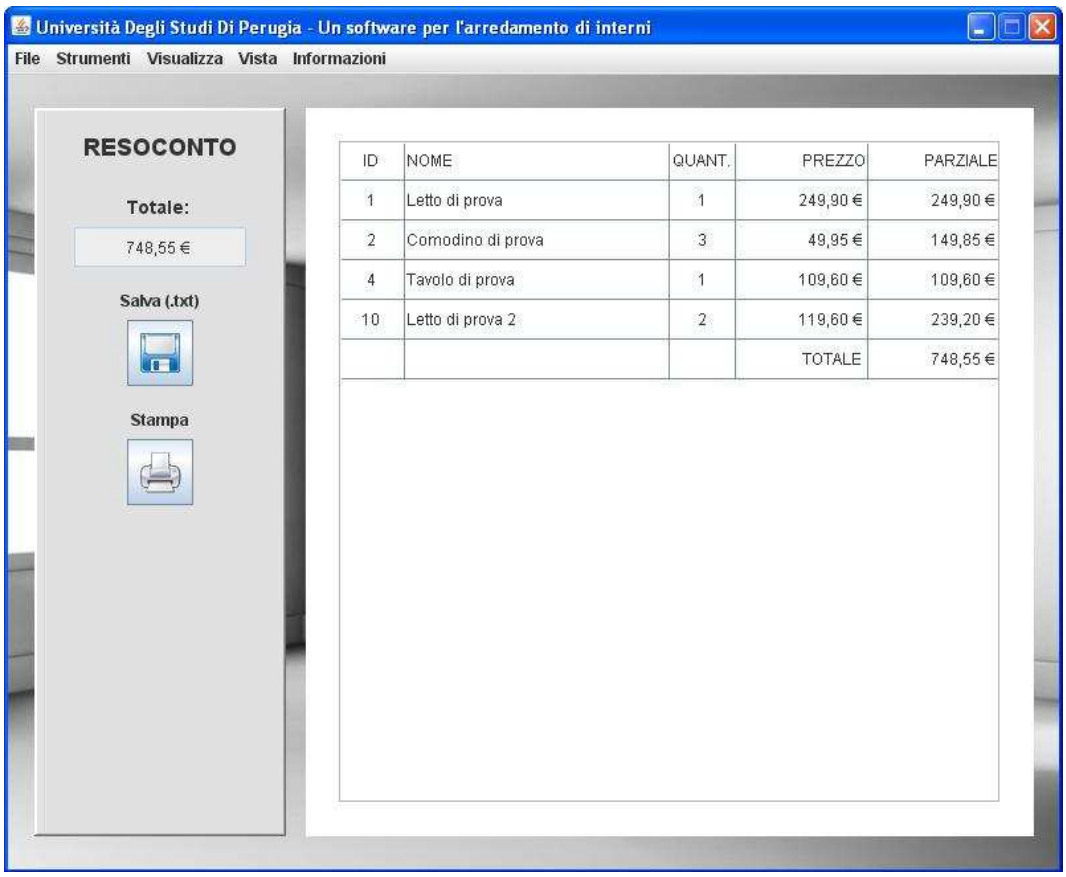
Vi sono essenzialmente tre tipi di trasformazione in grafica 3D e sono traslazione, rotazione e variazione di scala. Quest'ultima non è stata utilizzata poiché ogni modello importato presentava come caratteristica 100u = 1m che rappresenta la scala utilizzata nella visualizzazione tridimensionale.
La traslazione permette lo spostamento degli oggetti nello spazio mentre la rotazione permette di ruotarli in tutte le direzioni rispetto ad un punto detto pivot. In Java3D si realizza con semplici istruzione:

```
TransformGroup tg = new TransformGroup();
Transform3D trans = new Transform3D();
trans.setTranslation(new Vector3f(x, y, z));
trans.rotY(-rot);
tg.setTransform(trans);
```

Come abbiamo già visto, per ruotare, traslare o cambiare le dimensioni di un oggetto bisogna appoggiarsi alla classe TrasformGroup; questa permette di costruire un nuovo sistema di coordinate cartesiane rispetto ad un altro sistema. E' molto importante perchè le trasformazioni si accumulano a partire dalla radice fino alle foglie dello scene graph permettendo di trasformare con una semplice istruzione l'intera scena. Per specificare come trasformare un gruppo, si devono utilizzare delle matrici 4x4 che si possono definire via codice oppure utilizzando due oggetti Matrix4D o Transform3D.

4.7 Importazione ed esportazione dei dati

Una delle funzionalità più utili dell'applicazione dà la possibilità all'utente di salvare i propri progetti e di poterli utilizzare e modificare in qualsiasi momento.



Per aiutarci in questa operazione useremo una classe chiamata StringTokenizer contenuta nel package java.util. Questa classe permette di estrarre da una data stringa le sottostringhe che la compongono (token). I token sono separati da un carattere (o da una stringa) delimitatore. L'utilizzo di base è estremamente semplice, difatti occorre creare in prima istanza l'oggetto StringTokenizer. Il costruttore accetta due argomenti il primo è la stringa da suddividere, il secondo è il carattere delimitatore. Affinchè si scandisca l'intero testo utilizzeremo un ciclo while con condizione di continuazione

st.hasMoreTokens(), in pratica invochiamo il metodo hasMoreTokens() sull'oggetto StringTokenizer, questo metodo ritorna true se esistono altri token altrimenti ritorna false interrompendo così il nostro ciclo while. Per stampare il token appena recuperato invocheremo il metodo nextToken() sull'oggetto StringTokenizer:

```
StringTokenizer str = new StringTokenizer(data,"\r\n");
while (str.hasMoreTokens()) {
      Parete elem = new Parete();
      elem.importData(str.nextToken());
      parete.add(elem);
}
```

Detto questo passiamo al funzionamento vero e proprio: dopo aver progettato la stanza andremo sul sottomenù "Salva" del menu "File". Questo evento chiamerà la funzione exportFileData() che restituirà una stringa contenente tutti i dati necessari all'esportazione i quali saranno poi salvati su un file di estensione ".sai" (acronimo di "Software Arredamenti Interni").
Ogni classe utile alla creazione degli oggetti tra cui 'Parete', 'Oggetto', 'Porte', 'Finestre' possiede a sua volta exportData() ed importata(). La prima ha il compito di generare i token con opportuni delimitatori di riga e separatori di campo. La seconda utilizza la classe StringTokenizer per spacchettare queste informazioni e recuperare le variabili. Vediamone un esempio pratico:

```
public String exportData() {
    String del = "§";
    String sep = ";";
    String data = "";
    data += del;
    data += id+sep+finita+sep+selezionata+sep+orizzontale+sep;
    data += m+sep+dim+sep+h+sep+sp+sep;
    data += del;
    return data;
```

```
}

public void importData(String data) {
    StringTokenizer str = new StringTokenizer(data,";");
    id = Integer.parseInt(str.nextToken());
    finita = Boolean.getBoolean(str.nextToken());
    selezionata = Boolean.getBoolean(str.nextToken());
    orizzontale = Boolean.getBoolean(str.nextToken());
    m = Integer.parseInt(str.nextToken());
    dim = Integer.parseInt(str.nextToken());
    h = Integer.parseInt(str.nextToken());
    sp = Integer.parseInt(str.nextToken());
 }
```

Questo modello di esportazione/importazione dati è tipico dei file formato CSV (comma-separated values) dove ogni riga della tabella, rappresentata da una linea di testo, è divisa in campi separati da un apposito carattere separatore, ciascuno dei quali rappresenta un valore.

CAPITOLO 5

CONCLUSIONI E POSSIBILI SVILUPPI

5.1 Pannello amministrazione

La crescente domanda da parte degli utenti porterebbe ad un inevitabile sviluppo: un pannello nella quale un utente con speciali diritti, un amministratore, possa aggiungere nuovi prodotti e modificarne di vecchi. In questo momento questo è possibile soltanto accedendo alla base di dati ed eseguendo le opportune query. Questo tipo di operazioni non sono alla portata di tutti poiché richiedono la conoscenza del software e di determinate informazioni tecniche quali la conoscenza del Manager per la base di dati e del linguaggio di interrogazione, in questo caso SQL.

L'idea è quella di aggiungere un menu di gestione dove, per prima cosa, sarà necessario effettuare l'accesso con nome utente e password per verificare l'identità di colui che effettuerà le modifiche. Fatto questo sarà opportuno suddividere le operazioni in più sottosezioni, ad esempio inserimento, modifica e cancellazione. Sulla schermata di inserimento si potrà aggiungere una nuova categoria, una nuova tipologia o un nuovo oggetto ma, per fare questo, per ogni gruppo saranno necessari i campi corrispondenti alla base di dati, in più per gli oggetti dovrà esser definita la categoria e la tipologia. Altrettanto dovrà esser fatto per la modifica poiché si dovrà scegliere cosa modificare tenendo conto che esistono dei macrogruppi e dei sottogruppi, inoltre per gli oggetti la cosa è un po' più complicata poiché si dovrà inserire un'anteprima dell'oggetto e il modello 3D nel formato appropriato. Un'altra considerazione sta nel fatto che gli oggetti potrebbero non essere della scala appropriata e si dovrà inserire un ulteriore campo dove definire il fattore di scala. Infine, per quanto riguarda la cancellazione, dovrà esser fatta particolare attenzione poiché si dovranno adottare opportuni accorgimenti: l'eliminazione di una categoria o di una

tipologia non dovrà comportare l'eliminazione del loro contenuto ma solo delle loro associazioni.

Per finire, sarebbe opportuno suddividere gli amministratori a più livelli definendo, per ogni livello, determinati permessi. Ad esempio, l'amministratore con i permessi più alti potrebbe creare altri amministratori e definire a sua volta i permessi di ciascuno mentre all'amministratore con i permessi più bassi potrebbe esser negata la cancellazione.

5.2 Pannello utente

Mettiamo caso che un utente utilizzi molto questo software magari perché, come mestiere, si occupa ad esempio dell'arredamento di appartamenti o al quale piace rinnovare frequentemente l'arredamento della propria abitazione. Sarebbe alquanto utile aggiungere funzioni avanzate che possano garantirgli sicurezza e semplicità.

Per prima cosa, l'utente potrebbe effettuare una registrazione in modo da effettuare l'accesso via software. In questo modo tutte le sue informazioni sarebbero salvate nella base di dati compresi progetti, resoconti e carrelli virtuali. Ogni volta che effettuerà l'accesso non dovrà preoccuparsi di importare vecchi progetti, poiché saranno proposti immediatamente i progetti creati nelle sessioni precedenti ordinati per data o con particolari priorità. Si potrebbe fare in modo di effettuare il pagamento dei prodotti direttamente via software specificando se debba essere consegnato, ritirato o se è necessario il montaggio.

Infine, altra caratteristica aggiuntiva, potrebbe essere la scelta del materiale e del colore. In questo modo non si dovrebbero memorizzare tutti i possibili

prodotti ma per ogni utente, su ogni prodotto da aggiungere al carrello, verrebbero memorizzate le caratteristiche desiderate.

5.3 Planimetria avanzata

Per motivi di tempo, la progettazione della stanza presenta non poche lacune. Allo stato attuale non è possibile modificare o cancellare pareti esistenti. La gestione di queste due funzionalità prevede una serie di controlli e di trasformazioni. Poiché le pareti vengono realizzate una dopo l'altra fino alla conclusione della stanza, queste funzionalità potrebbero essere permesse solo in questa fase. Quando la stanza è completata si potrebbe selezionare una parete e indicare le nuove dimensioni direttamente nei textbox o in caso decidere di eliminarla. L'eliminazione, però, renderebbe la stanza nuovamente incompleta e, se si vuole continuare ad adottare il modello a costruzione continua, bisognerà permettere la cancellazione di una sola parete per volta che avvii nuovamente la funzione di costruzione. Bisognerà anche tener conto di finestre e porte che in caso di cancellazione dovranno essere eliminate dalle rispettive liste mentre in caso di modifica dovranno fungere da vincolo poiché una parete non dovrebbe essere più piccola della somma delle lunghezze di porte e finestre..

Altra limitazione consiste nel fatto che è possibile definire solo una stanza alla volta e la grandezza massima dell'area di lavoro ne limita la dimensione.
La soluzione consiste nell'utilizzare un area di lavoro dinamica dove sia possibile spostarsi o modificare le proporzioni. Nel nostro caso ad esempio l'area di lavoro è di circa 5 m. Per poter progettare stanze di grandezza maggiore si potrebbe adottare uno "zoom" che permetta di ingrandire o rimpicciolire l'area e di andare a lavorare su zone specifiche. Lo spostamento ci permetterebbe inoltre di ampliare la stanza senza dover necessariamente

ridurre le dimensioni. A questo punto la griglia potrebbe svolgere un ruolo essenziale permettendo di definire la precisione con cui lavoriamo.

Un'altra mancanza è la possibilità di creare pareti con angoli diversi da 90 gradi limitando un po' la tipologia delle stanze. Anche l'inserimento di questa funzionalità prevede una serie di considerazioni in quanto non esisterebbero più i vincoli parete-orizzontale e parete-verticale ma sarebbe possibile un qualsiasi tipo di parete. Inoltre ogni altro oggetto dovrà godere di questa proprietà poiché, ad esempio, porte e finestre sono associate alle pareti e un qualsiasi altro oggetto potrebbe essere accostato ad una parete.

5.4 Gestione su vista 3D

Per il momento la vista 3D ci permette soltanto la visualizzazione e lo spostamento nella scena. Questo perché per il momento non vengono gestiti una serie di oggetti in cui l'asse Z svolge un ruolo primario.
Ad esempio, aggiungendo un quadro alla stanza questo potrebbe esser piazzato in una determinata posizione della parete ma la sua altezza non potrebbe esser gestita. Una soluzione potrebbe essere quello di aggiungere un campo modificabile nella quale definire l'altezza da terra manualmente ma non sarebbe sicuramente professionale. Altra soluzione è permettere lo spostamento degli oggetti sulla vista 3D.

Per fare questo sono necessarie una serie di funzionalità correlate. In primo luogo si dovrebbe selezionare l'oggetto con lo stesso sistema della schermata di arredamento ma con il controllo sull'asse Z. Quindi, all'evento press, per ogni oggetto, controllerò se la posizione del mouse (x, y, z) gli appartiene. Se ciò si verifica imposterò la variabile booleana selezionato uguale a true e mostrerò la selezione nella vista 3D. Dopodichè, finché non avviene il rilascio,

gli spostamenti del mouse o la pressione dei tasti della tastiera non serviranno più per lo spostamento della scena ma dell'oggetto.

Questo però richiede funzioni di controllo delle collisioni più avanzate dove invece di far verificare una collisioni mostrando soltanto che sia avvenuta si dovrebbe cercare di accostare l'oggetto all'elemento. Un letto ad esempio dovrebbe essere accostato alla parete o una lampada da notte ad un comodino.

5.5 Conclusioni

Il software presenta tutte le caratteristiche prefissate. L'interfaccia è molto intuitiva e gradevole alla vista e sono da notare le innumerevoli funzionalità proposte che riprendono quelle di molti programmi professionali analoghi. Per quanto riguarda la vista 3D, permette libertà assoluta nello spostamento e la resa grafica è alquanto soddisfacente, in buona parte, grazie alla scelta appropriata di luci e materiali. Anche per quanto riguarda le prestazioni, si è cercato di ottimizzare il più possibile l'applicazione, in particolar modo nella sezione più critica: il rendering. Dopo un primo caricamento iniziale abbastanza pesante, dovuto alla generazione della scena, la risposta agli eventi è immediata.

Con questo lavoro è stato raggiunto sicuramente più di un obiettivo ma, nonostante questo, presenta delle mancanze dovute alle molteplici funzionalità che un software di questo tipo deve possedere. Bisogna tener conto delle persone a cui è rivolto poiché un software deve essere sviluppato essenzialmente attorno alle loro esigenze.

Ho cercato, in particolar modo, di mostrare le problematiche legate alla realizzazione di un'applicazione a partire dalla scelta del linguaggio utilizzato fino al sistema di memorizzazione delle informazioni. Il linguaggio, le librerie,

il sistema di memorizzazione, i modelli e le tecniche adottate, fanno parte dei dettagli realizzativi. La scelta di ognuno di essi ha portato ad un insieme di problematiche, molte delle quali, inizialmente, non erano state nemmeno prese in considerazione. Ognuna di queste problematiche è stata affrontata tenendo conto che non esiste la scelta o l'algoritmo in assoluto migliore ma quello più adatto alle esigenze e alla situazione.

BIBLIOGRAFIA

TESTI ED ARTICOLI CONSIGLIATI:

- **Karsten Samaschke**, *"Java Dai Fondamenti alla programmazione avanzata"*, Apogeo, 2005

- **Cay S.Horstmann**, **Gary Cornell**, *"Java 2 Tecniche Avanzate"*, McGraw-Hill, 2003

- **Herbert Schildt**, *"Fondamenti di Java 2"*, McGraw-Hill, 2004

- **Daniel Selman**, *"Java 3D Programming"*, Hanning, 2002

- **Aaron E. Walsh**, *"Java 3D API Jump-Start"*, Sun Microsystems, 2001

- **Ramez A. Elmasri**, **Shamkant B. Navate**, *"Sistemi di basi di dati Fondamenti"*, 5ª edizione, Pearson Education, 2007

- **Ramez A. Elmasri**, **Shamkant B. Navate**, *"Sistemi di basi di dati Complementi"*, 4ª edizione, Pearson Education, 2005

DOCUMENTAZIONE ONLINE:

- **Milanese Francesco**, *"Java 3D – Guida di base"*, http://www.redbaron85.com

- **Dario Guadagno**, *"Introduzione a Java 3D"*, http://www.programmazione.it/index.php?entity=eitem&idItem=37337

BIBLIOGRAFIA

- **Eugenio Polito**, *"Un breve tutorial su Java3D"*, http://www.redksr.com/~talos/JAVA/JAVA3D/Tutorial%20Java3D.pdf

- **Sun Microsystems**, *"Java 3D API Tutorial"*, http://java.sun.com/developer/onlineTraining/java3d/

- **Java.HTML.it**, *"JDBC"*, http://java.html.it/guide/lezione/799/jdbc-introduzione/

- **JDBC**, *"Java JDBC Tutorial"* , http://www.jdbc-tutorial.com/

- **Damiano Verda**, *"Normalizzazione di un database"*, http://www.mrwebmaster.it/sql/articoli/normalizzazione-database_1004.html

- **Francesco Carotenuto**, *"Progettare un database relazionale"*, http://www.programmazione.it/index.php?entity=eitem&idItem=44871

- **Charles L. Hamberg**, **Ronald Vavrinek**, *"Shoelace Algorithm"*, http://darcy.rsgc.on.ca/ACES/ICS3U/PDFs/Shoelace.pdf

- **Wikipedia**, *"Shoelace Formula"*, http://en.wikipedia.org/wiki/Shoelace_formula

- **Magillo P.**, *"Trasformazioni di coordinate"*, http://www.disi.unige.it/person/MagilloP/DISPENSE_IG/trasform.html

- **Cignoni**, *"Trasformazioni Geometriche"*, http://vcg.isti.cnr.it/~cignoni/FGT0607/FGT_03_Trasformazioni.pdf

BIBLIOGRAFIA

SOFTWARE UTILIZZATI:

- **Anim8or v0.95c**, “Free 3D Animation Software”, http://www.anim8or.com/

- **NetBeans IDE v6.8**, “Ambiente di sviluppo” http://netbeans.org/

- **Java JDK v1.6**, *“Java Development Kit”*, http://www.oracle.com/technetwork/java/javase/downloads/index.html

- **Java 3D v1.5.2**, *“Application Programming Interface”*, https://java3d.dev.java.net/binary-builds.html

- **phpMyAdmin v3.3.2**, *“Manager Database”*, http://www.phpmyadmin.net/home_page/index.php

- **MySQL v5.1.45**, *“Database Open Source”*, http://dev.mysql.com/downloads/mysql/