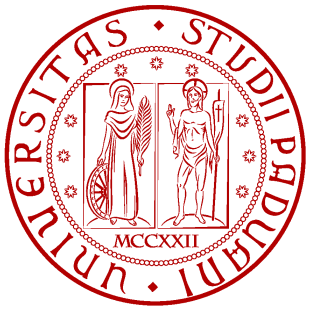

UNIVERSITA' DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria dell'Informazione

Corso di laurea in Ingegneria Biomedica

Lenti oftalmiche in Java3D

Laureando

Marco Matta

Relatore

Franco Bombi

Tutore Aziendale

Alessandro Cogo

Anno Accademico 2011/12

Elenco delle figure

Indice

Introduzione

Questa relazione descrive il lavoro svolto durante il tirocinio presso l'azienda PEGASO srl di Mestre (Venezia), una Software House rivolta al settore dell'Ottica Oftalmica. Nei due mesi di attività è stato possibile avvicinarsi ad una realtà lavorativa specifica ed utilizzare conoscenze e competenze sviluppate nel corso di laurea in Ingegneria Biomedica.

L'obiettivo del progetto era quello di proporre uno strumento di visualizzazione 3D che, partendo dal calcolo degli spessori delle lenti da produrre, rappresenti l'occhiale prima della sua effettiva fabbricazione.

Nella relazione viene inizialmente descritta l'attività dell'azienda, e il programma utilizzato dalla stessa per la progettazione e la costruzione delle lenti. Dopo una breve illustrazione dei vantaggi attesi dall'utilizzo di un software di simulazione delle lenti oftalmiche a partire dai dati di progettazione si propone l'analisi del concetto di lente e la sua implementazione in Java. Infine una breve guida allo sviluppatore per l'installazione del software.

Capitolo 1

Il software aziendale

L'azienda presso cui è stato svolto il tirocinio è una Software House che sviluppa applicazioni software nel settore dell'Ottica e che realizza vari progetti di informatizzazione attraverso sistemi IBM AS/400 (iSeries) per personal computer in ambiente Windows, il software prodotto è rivolto a fornitori ottici e a negozi di occhiali.

Il prodotto principale della PEGASO è un software sviluppato interamente in azienda che, dati i parametri di costruzione di una lente fornisce una serie di punti che la descrivono; questi punti una volta passati come input ai macchinari di produzione permettono di costruire la lente.

Il software è denominato pacchetto Argo, scritto in Visual Basic, il quale offre servizi di collegamento e gestione dei generatori di lenti e garantisce la possibilità di utilizzo contemporaneo di più macchinari dallo stesso terminale. Utilizzando il server AS/400 vengono eseguiti i calcoli degli spessori delle lenti da produrre, i quali vengono inviati ai generatori per la produzione.

1.1 I vantaggi di un Software di simulazione di una lente

La mole di informazioni sulla forma (curve e spessori) e peso (materiale) delle lenti è gestibile con tabelle o calcoli pratici solo accettando margini di incertezza. Invece, i software permettono, a partire da un notevole archivio di informazioni su come le singole lenti sono costruite, di definire attendibilmente caratteristiche di una lente prima e dopo la lavorazione.

La simulazione di una lente, dato un certo materiale ed altri parametri costruttivi, permette di mettere in evidenza gli spessori al centro e al bordo a varie distanze, il bordo e il peso della lente molata. Questi dati sono utili per verificare la possibiltà di realizzazione di una certa lente seguendo criteri di dimensioni, peso ed estetica dettati dal mercato.

In questo contesto si inserisce il progetto realizzato che, grazie all'accuratezza dei dati forniti, simula in modo preciso la forma geometrica dell'ipotetico occhiale finale.

Capitolo 2

Il progetto

Lo scopo del progetto è la rappresentazione in tre dimensioni delle lenti realizzando un applicativo in Java per integrare il pacchetto Argo che prevede la rappresentazione grafica solamente nel piano. L'obiettivo è ottenere una resa grafica degli occhiali finali il più realistica possibile e con buone prestazioni.

2.1 Il software

Il pacchetto Argo accede ai punti utilizzati dai generatori di lenti, i quali vengono opportunamente campionati per ottenere prestazioni hardware migliori; grazie a questi dati e agli strumenti messi a disposizione da Java 3D, risulta possibile un buon rendering del prodotto che metta in risalto gli spessori.

Questo software, opportunamente integrato, potrebbe essere molto utile sia all'ottico in fase di produzione delle lenti, sia ad un rivenditore di occhiali. Nel primo caso l'applicativo permetterebbe di vedere la lente ancora prima di produrla e di verificarne la possibilità di realizzazione fisica o la necessità di rivalutarne la progettazione.

Nel secondo caso il progetto potrebbe essere la base di un applicativo che permetta a un negozio di pubblicizzare i propri occhiali attraverso presentazioni 3D senza dover necessariamente esporli fisicamente in vetrina.

L'argomento è di particolare interesse al tirocinante in quanto permette di approfondire la conoscenze del linguaggio Java e mettere in pratica metodologie dell'ingegneria.

2.2 I risultati ottenuti

Al termine dell'esperienza di tirocinio si è giunti alla produzione di un software in Java denominato Visualizza3D il quale, sfruttando le potenzialità offerte dalla libreria open source JAVA 3D, permette di vedere da qualsiasi angolazione e a qualsiasi livello di ingrandimento il modello in tre dimensioni di un occhiale da vista.

I tre vantaggi del software prodotto si possono apprezzare nella precisione della rappresentazione, nel rendering e nella prestazione. La precisione è dovuta all'utilizzo dei dati reali utilizzati dai generatori, il che permette di stimare in modo attendibile il risultato finale. Il rendering del materiale è stato implementato utilizzando concetti di propagazione delle onde, Java 3D mette a disposizione delle classi che permettono di ottenere buoni risulta senza perderne in prestazioni hardware.

La prima versione di Visualizza3D si presenta in una forma molto semplice ma funzionale, in quanto permette di ruotare l'occhiale con il mouse.

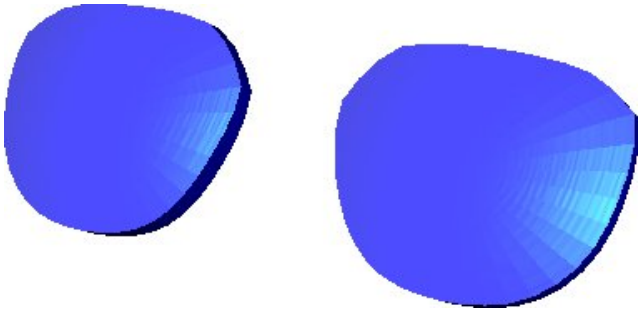

Figura 2.1: Rendering di un occhiale da vista.

Inizialmente l'idea della rappresentazione in 3D della lente si limitava ad essere un modesto abbellimento di un programma di visualizzazione spessori nel piano, ma i risultati ottenuti hanno superato le aspettative e attualmente il software è installato presso alcuni clienti; inoltre questo si potrebbe evolvere permettendo l'osservazione dettagliata di porzioni di lente in 3D affiancate dai parametri costruttivi.

2.3 Visualizzazione 3D: un caso pratico

Un fornitore di occhiali può servirsi del software di visualizzazione per decidere quale tipo di lente utilizzare per correggere i difetti visivi di un cliente. Di seguito i parametri costruttivi da fornire alla schermata iniziale del programma:

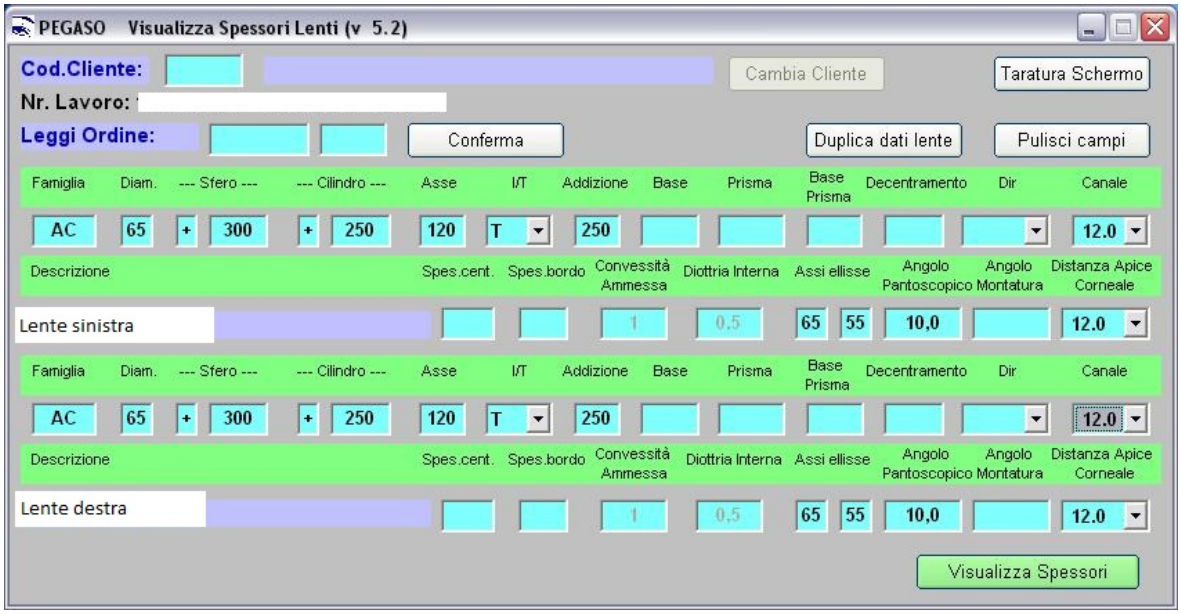

Figura 2.2: Inserimento dei parametri costruttivi delle due lenti.

- **Famiglia** scelta del semilavorato da utilizzare e il tipo di lente:
  - monofocale
  - bifocale
  - progressiva
  - progressiva interna freeform
  - progressiva esterna lavorazione interna
  - progressiva esterna lavorazione esterna

  è possibile scegliere anche l'indice di rifrazione;
- **Diametro** del semilavorato di partenza, varia tra i 60/80 mm;
- **Sfero** è un parametro legato alla presbiopia (per valori negativi) e alla miopia (per valori positivi), varia tra -24/+24 diottrie;
- **Cilindro** per correggere l'astigmatismo, varia tra 0/9 diottrie;
- **Asse del cilindro** angolazione dell'asse del cilindro che dipende dalla direzione del difetto visivo, varia da 0/180 gradi;
- **Addizione** per lenti multifocali e progressive, varia tra 0,50/4 diottrie;

- **Base** è la diottria esterna del semilavorato, scelta determinata dal tipo di occhiali da produrre, ad esempio con lenti molto curve la base dovrà essere grande, varia tra 0,25/12 diottrie;
- **Prisma** per la correzione dello strabismo varia tra 0/10 dottrie prismatiche;
- **Decentramento** del semilavorato per rendere possibila la produzione di una certa forma;
- **Canale** di progressione delle lenti progressive, varia tra 10/20 mm;

In base alla prescrizione del medico, inizialmente è necessario scegliere il semilavorato, il tipo di lente e il materiale. Poi in base ai gusti del cliente ai limiti costruttivi si sceglierà la forma della lente. Nelle decisioni è importante tenere presente:

- le **preferenze** del cliente;
- i **limiti fisici** di costruzione;
- i **costi** di produzione;
- il materiale del **magazzino** del fornitore, perciò la valutazione della scelta ottimale deve dipendere anche da questo parametro (in questa relazione non si entra in merito al problema).

Ad esempio, i parametri inseriti in fig. 2.2 potrebbero essere validi per la correzione di difetti visivi legati alla presbiopia e l' astigmatismo, magari di una persona con età media di 50 anni.

Eseguendo il programma si apre una schermata con i dati: intorno alle lenti in fig. 2.3 vengono mostrati i valori degli spessori al bordo; l'ideale sarebbe ottenere numeri esigui in modo da lavorare con lenti sottili quindi più leggere ed esteticamente più apprezzate.

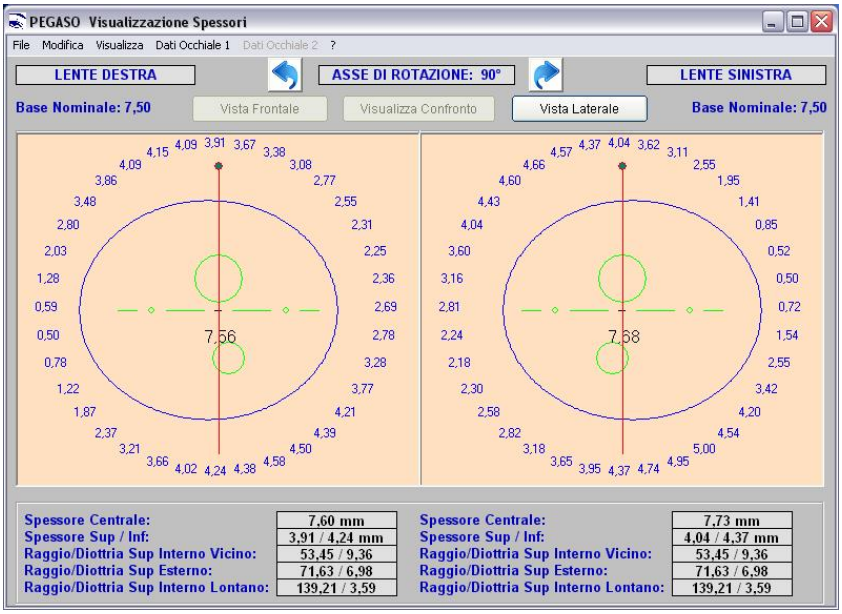

Figura 2.3: Spessori al bordo della lente finita prima della molatura.

Il paziente ha la possibilià di scegliere la forma delle lenti in base ai propri gusti personali (fig.2.4) e quindi la lente verrà tagliata (fig. 2.5) secondo una determinata geometria. Dai dati si osserva che gli spessori ai bordi sono un pò troppo grandi, il 3D dell'occhiale mette in risalto questo dettaglio (fig. 2.6).

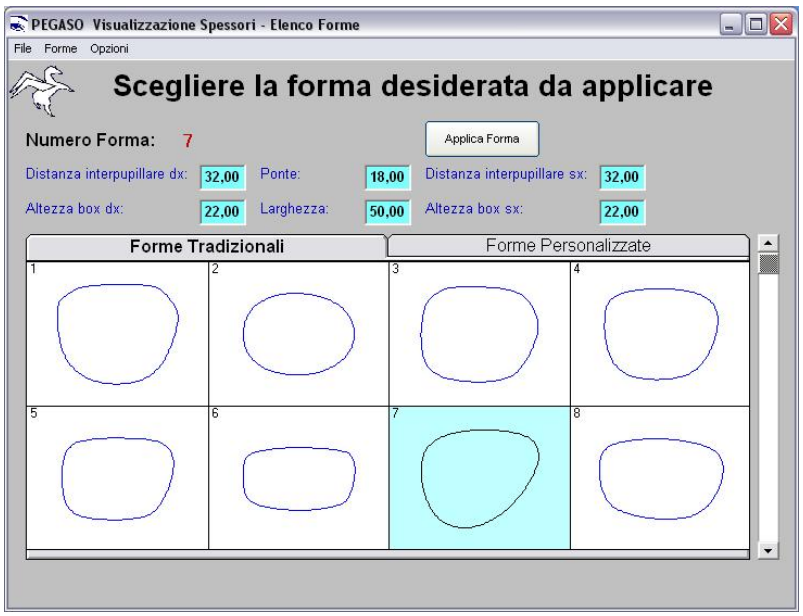

Figura 2.4: Scelta della forma dell'occhiale da parte del cliente.

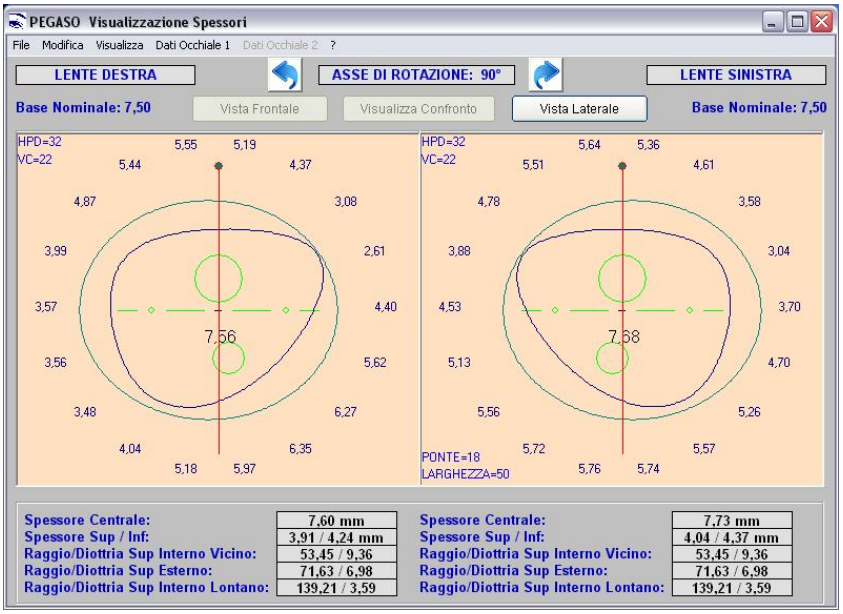

Figura 2.5: Spessori ai bordi della lente con la forma desiderata.

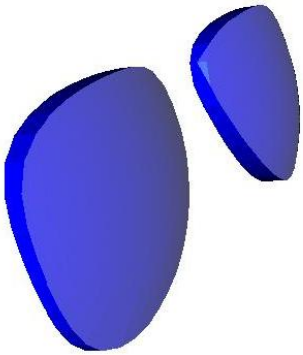

Figura 2.6: Visualizzazione 3D, risalto spessori ai bordi.

Riscontrato il difetto estetico, ma anche di funzionalità, dato il il peso eccessivo, l'ottico può consigliare un occhiale migliore al cliente, ad esempio provando a cambiare il diametro del semilavorato e valutare una forma che ottimizzi gli spessori al bordo.

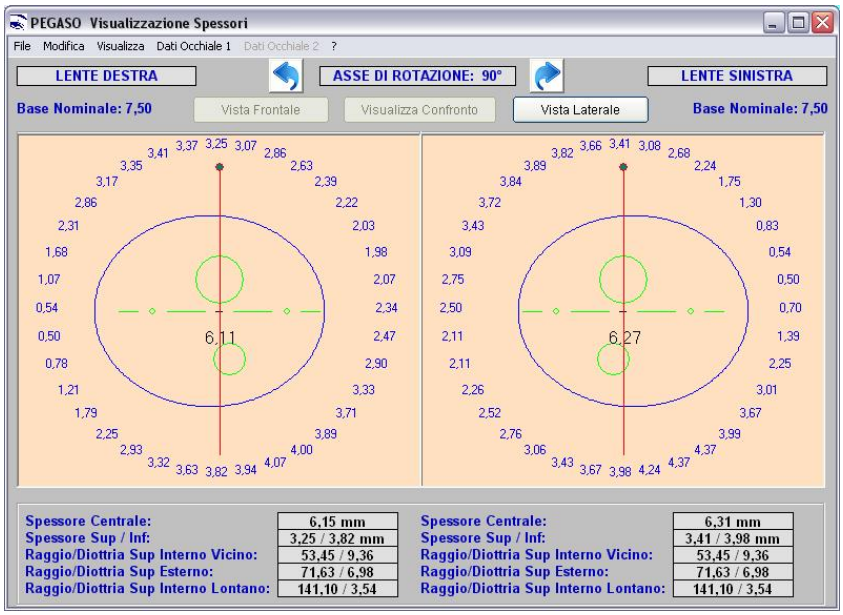

Figura 2.7: Spessori al bordo della lente finita prima della molatura.

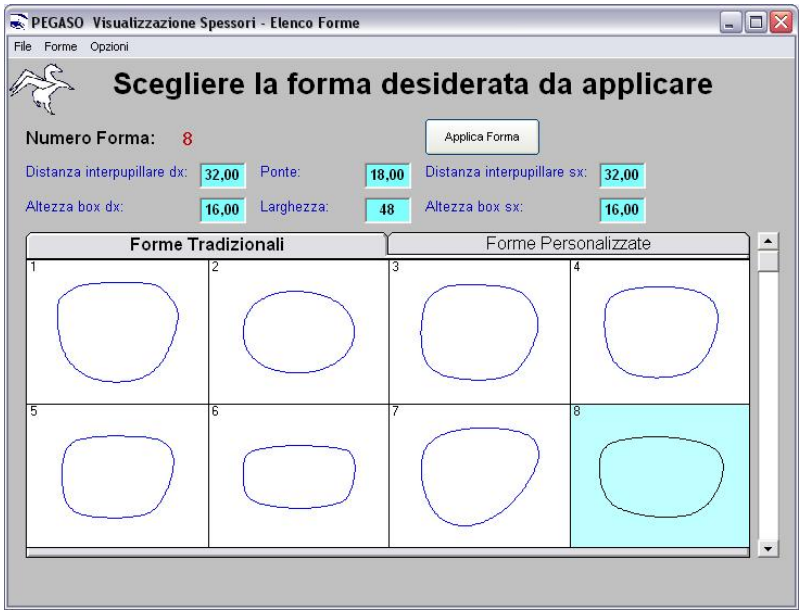

Figura 2.8: Forma dell'occhiale consigliata al cliente.

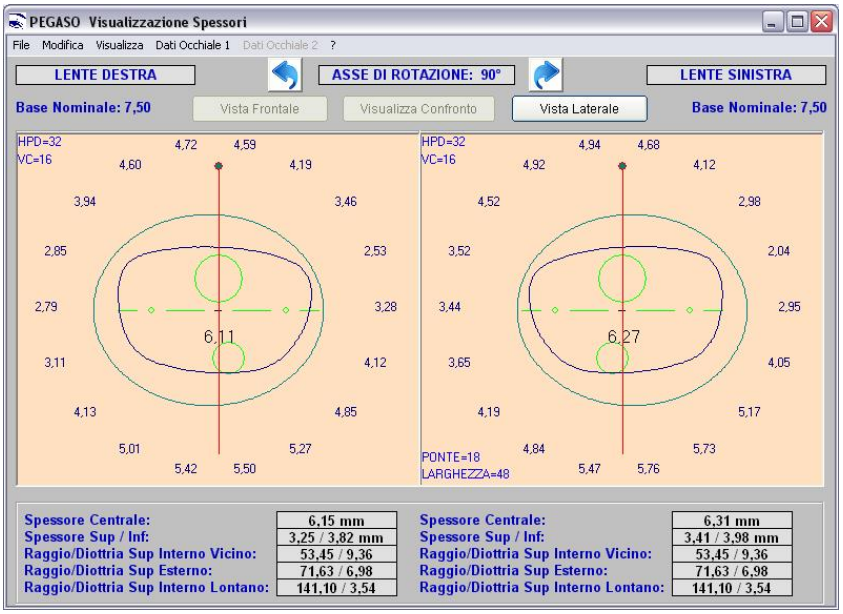

Figura 2.9: Spessori ai bordi della lente con la forma desiderata, esteticamente migliore.

In questo modo il cliente potrà valutare se il tipo di occhiale proposto potrebbe piacergli o se provare con un altro modello, magari con un altra forma; di seguito il modello 3D fig. 2.10 e fig. 2.11.

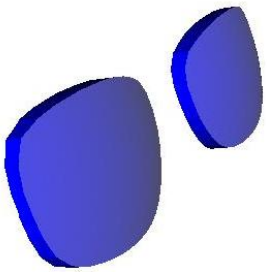

Figura 2.10: L'occhiale in 3D : complessivamente migliore rispetto alla prima scelta.

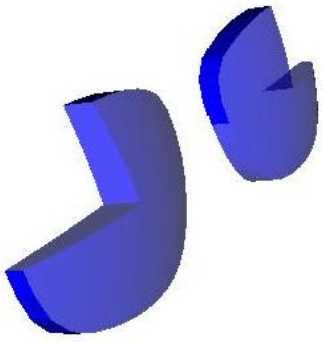

Figura 2.11: Sezione occhiale: osservazione migliore degli spessori.

Capitolo 3

Implementazione del software

Il progetto è stato sviluppato utilizzando il linguaggio Java così da poter sfruttare le potenzialità della programmazione orientata agli oggetti in vista di eventuali estensioni future. Le caratteristiche della Java Virtual Machine sono tali da poter offrire la possibilità d'utilizzo del software su tutte le piattaforme disponibili sul mercato. Nel particolare è stata usata la libreria Java 3D, l'Application Programming Interface (API), impiegata per realizzare applicazioni grafiche 3D, che si basa sul concetto di scene graph e che appartiene alla grande famiglia di API Java Media. Se comparata alle altre librerie più tradizionali, Java 3D non è solo una libreria di interfaccia ma permette anche l'implementazione della programmazione object-oriented. Il software è stato sviluppato con l'Integrated Development Environment (IDE) di programmazione ECLIPSE.

3.1 L'astrazione del concetto di lente

Si è deciso di suddividere la lente in 3 superfici: inferiore, superiore e laterale. La lente poi è stata suddivisa in modo astratto in sezioni definite da 18 assi equidistanti e la sua superficie viene definita mediante le coordinate spaziali di questi assi.

I dati forniti da AS/400 sono suddivisi in coordinate relative agli assi di simmetria, specificandoli sia per la parte superiore sia per la parte inferiore. Ogni asse dista dal precedente esattamente di 10 gradi.

I punti sono abbastanza vicini da consentire una buona descrizione della lente e sono 378 relativi a un certo asse; collegando tali coordinate con una linea continua si ottiene una porzione di lente (fig 3.2). Le 3 superfici, dunque, saranno formate dalla somma di tante superfici infinitesime: a questo scopo è stata utile la numerazione delle coordinate e degli assi di simmetria.

Il file contenente l'input (FileDati3D.seq) contiene un insieme di 18 record strutturati nel seguente modo:

- riga 1 xxx numero asse (xxx 000-017);
- riga 2 coordinate x 378 ;
- riga 3 coordinate y 378;
- riga 4 coordinate z superficie inferiore 178;
- riga 5 coordinate z superficie superiore 178;

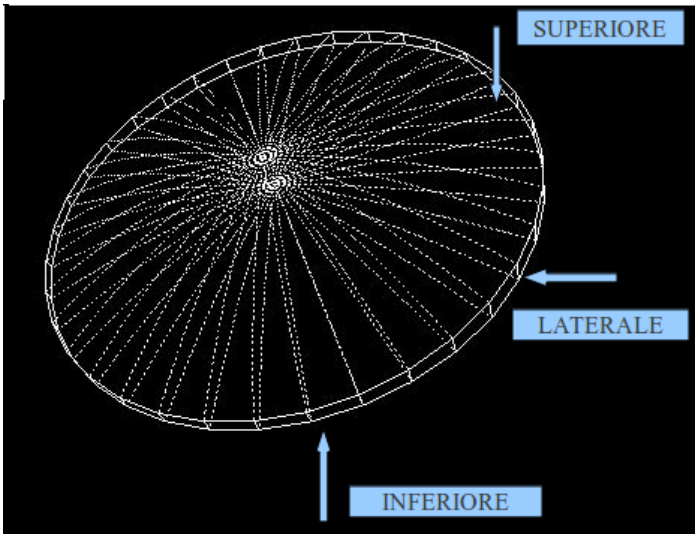

Figura 3.1: Struttura geometrica della lente.

le coordinate sono separate da uno spazio.

Ciò che ha riscontrato maggior difficoltà è stata l'identificazione dell'ubicazione esatta di ogni punto, in modo da poter progettare algoritmi che disegnino la lente. Il file può contenere da una a due lenti (destra e sinistra).

Sia R la superficie che rappresenta l'intera lente, questa viene partizionata in tante superfici infinitesime s tali che: $R = \bigcup_1^n s_i$, sia s_i è una regione connessa e $s_i \bigcap s_r = \emptyset$ per tutti gli i e r con $i \neq r$. Quindi, defininendo con k l'asse di simmetrica k_{esimo} e j l'asse j_{esimo} con $j = i + 1$, è possibile costruire i quadrilateri s_i con 2 punti di k e due di j come vertici.

Le superfici superiore ed inferiore sono composte da 36 spicchi ciascuna, ognuna contenente 94 quadrilateri; la superficie laterale contiene 36 quadrilateri che costituiscono il bordo della lente.

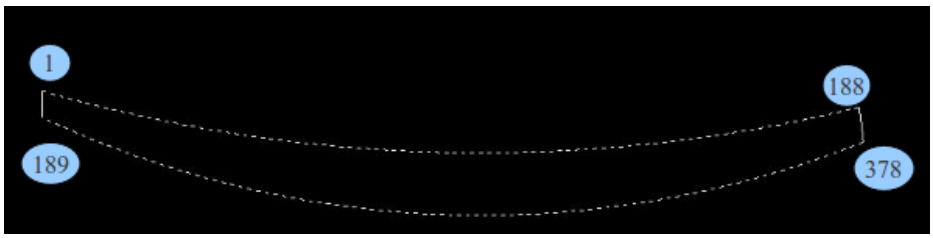

Figura 3.2: Asse di rotazione della lente.

3.2 Il package Visualizza3D

La parte più importante del progetto consiste nel package VISUALIZZA3D che ha il compito fondamentale di leggere in ingresso il FileDati3D.seq e di generare conseguentemente la struttura geometrica della lente.

Il risultato finale è un oggetto di tipo TransformGroup che contiene le informazioni relative alle superfici, ai vettori normali a queste, e alle proprietà di apparenza (come il tipo di materiale, di colore e di trasparenza).

Il pacchetto è composto dalle seguenti classi:

- Asse.java;
- GeneratorePunti.java ;
- GeneratoreTransformGroupLenti.java;

La classe **Asse.java** implementa il concetto di asse di rotazione di una lente; in questo progetto un asse di rotazione è definito da 378 punti di cui metà appartengono alla parte inferiore e metà alla parte superiore della lente.

L'idea è quella di predisporre un semplice array di Poin3D di 379 elementi ed occupare i posti da 1 a 378 con i punti; la classe prevede metodi di accesso e inserimento delle coordinate x, y e z.

```
1 /**
2  * Classe che rappresenta un asse di simmetria
3  * della lente.
4  *
5  * @author Marco Matta
6  *
7  */
8 public class Asse {
9
10    /**
11     * Variabile di classe,  un array di Point3D
12     * che contiene i punti di un certo asse di
13     * simmetria
14     */
15    public Point3d[] punti;
16
17    /**
18     * Costruttore che crea un array di Point3D
19     * [0-378] e inizializza tutti i punti con
20     * (0,0,0). Vengono utilizzati i posti da 1 a
21     * 318.
22     */
23    public Asse() {
24       punti = new Point3d[379];
25       for (int i = 0; i <= 378; i++) {
26          punti[i] = new Point3d();
27       }
28    }
29
30    /**
31     * Metodo di accesso al k-esimo punto
32     *
33     * @param k il k-esimo punto
34     * @return il punto k-esimo nello spazio 3D
35     */
36    public Point3d getPunto(int k) {
37       return punti[k];
38    }
```

```
39
40    /**
41     * Metodo di accesso alla coordinata x del
42     * k-esimo punto
43     *
44     * @param k il k-esimo punto
45     * @return la coordinata x del k-esimo punto
46     */
47    public double getX(int k) {
48       return punti[k].x;
49    }
50
51    /**
52     * Metodo di accesso alla coordinata y del
53     * k-esimo punto
54     *
55     * @param k il k-esimo punto
56     * @return la coordinata y del k-esimo punto
57     */
58    public double getY(int k) {
59       return punti[k].y;
60    }
61
62    /**
63     * Metodo di accesso alla coordinata z del
64     * k-esimo punto
65     *
66     * @param k il k-esimo punto
67     * @return la coordinata z del k-esimo punto
68     */
69    public double getZ(int k) {
70       return punti[k].z;
71    }
72
73    /**
74     * Metodo di modifica del k-esimo punto
75     *
76     * @param k il k-esimo punto
77     * @param x coordinata x
78     * @param y coordinata y
79     */
80    public void addPunto(int k, double x, double y) {
81       punti[k] = new Point3d(x, y, 0);
82    }
83
84    /**
85     * Metodo di modifica della coordinata x
86     * k-esimo punto
87     *
88     * @param k il k-esimo punto
89     * @param x coordinata x
90     */
91    public void addX(int k, double x) {
92       punti[k].x = x;
93    }
94
95    /**
96     * Metodo di modifica della coordinata y
97     * k-esimo punto
98     *
99     * @param k il k-esimo punto
```

```
100      * @param y coordinata y
101      */
102     public void addY(int k, double y) {
103        punti[k].y = y;
104     }
105
106     /**
107      * Metodo di modifica della coordinata z
108      * k-esimo punto
109      *
110      * @param k il k-esimo punto
111      * @param z coordinata z
112      */
113     public void addZ(int k, double z) {
114        punti[k].z = z;
115     }
116 }
```

La classe **GeneratorePunti.java** ha lo scopo di predisporre una struttura dati che contenga tutti gli assi di rotazione di una o due lenti, la lente viene quindi rappresenta da un array di 18 Assi.java.

L'input può essere un Filedati3D.seq contenente solo la lente destra o sinistra o entrambe, tutti questi casi sono gestiti in modo da non interrompere l'esecuzione del programma.

```
1 /**
2  * Classe che legge tutti i punti di tutti gli
3  * Assi dal file delle lenti e li salva in un
4  * TableLookUp.
5  *
6  * @author Matta Marco
7  */
8 public class GeneratorePunti {
9
10    private Asse[] asseSx;
11    private Asse[] asseDx;
12    private Asse[] asse;
13    private double ytras = 0.25;
14
15    /**
16     * Costruttore che crea un array di Asse [0-17]
17     * e inizializza tutti gli Assecon punti
18     * (0,0,0). Vengono utilizzati i posti da 0 a
19     * 17
20     */
21    public GeneratorePunti() {
22       asseSx = new Asse[18];
23       for (int i = 0; i <= 17; i++) {
24          asseSx[i] = new Asse();
25       }
26       asseDx = new Asse[18];
27       for (int i = 0; i <= 17; i++) {
28          asseDx[i] = new Asse();
29       }
30    }
31
32    /**
33     * @param k il k-esimo Asse
34     * @return il k-esimo Asse di simmetria con i
35     * relativi punti
36     */
37    public Asse getAsseSx(int k) {
38       return asseSx[k];
39    }
40
41    public Asse getAsseDx(int k) {
42       return asseDx[k];
43    }
44
45    public Asse getAsse(int k) {
46       return asse[k];
47    }
48
49    /**
50     * Genera i punti della lente destra.
51     *
52     * @param file
```

```
53     * @throws FileNotFoundException
54     */
55    public void generaAsseDx(String file)
56            throws FileNotFoundException {
57       asse = new Asse[18];
58       for (int i = 0; i <= 17; i++) {
59          asse[i] = new Asse();
60       }
61       FileReader reader = new FileReader(file);
62       Scanner in = new Scanner(reader);
63       in.nextLine();
64       in.nextLine();
65       int i, j;
66       double x = 0, y = 0, z = 0;
67       // LENTE
68       for (i = 0; i <= 17; i++) { // X
69          for (j = 1; j <= 189; j++) {
70             x = in.nextDouble() / 100000;
71             asse[i].getPunto(j).x = x;
72             asse[i].getPunto(j + 189).x = x;
73          }
74          // Y
75          for (j = 1; j <= 189; j++) {
76             y = in.nextDouble() / 100000;
77             asse[i].getPunto(j).y = y;
78             asse[i].getPunto(j + 189).y = y;
79          }
80          // Z
81          for (j = 190; j <= 378; j++) {
82             z = in.nextDouble() / 100000;
83             asse[i].getPunto(j).z = z;
84          }
85          for (j = 1; j <= 189; j++) {
86             z = in.nextDouble() / 100000;
87             asse[i].getPunto(j).z = z;
88          }
89          if (i == 17) {
90          } else {
91             in.nextDouble();
92          }
93       }
94       in.close();
95    }
96
97    /**
98     * Genera i punti della lente sinistra.
99     *
100    * @param file
101    * @throws FileNotFoundException
102    */
103   public void generaAsseSx(String file)
104           throws FileNotFoundException {
105      asse = new Asse[18];
106      for (int i = 0; i <= 17; i++) {
107         asse[i] = new Asse();
108      }
109      FileReader reader = new FileReader(file);
110      Scanner in = new Scanner(reader);
111      in.nextLine();
112      in.nextLine();
113      int i, j;
```

```
114       double x = 0, y = 0, z = 0;
115       // LENTE SX
116       for (i = 0; i <= 17; i++) { // X
117          for (j = 1; j <= 189; j++) {
118             x = in.nextDouble() / 100000;
119             asse[i].getPunto(j).x = x;
120             asse[i].getPunto(j + 189).x = x;
121          }
122          // Y
123          for (j = 1; j <= 189; j++) {
124             y = in.nextDouble() / 100000;
125             asse[i].getPunto(j).y = y;
126             asse[i].getPunto(j + 189).y = y;
127          }
128          // Z
129          for (j = 190; j <= 378; j++) {
130             z = in.nextDouble() / 100000;
131             asse[i].getPunto(j).z = z;
132          }
133          for (j = 1; j <= 189; j++) {
134             z = in.nextDouble() / 100000;
135             asse[i].getPunto(j).z = z;
136          }
137          if (i == 17) {
138          } else {
139             in.nextDouble();
140          }
141       }
142       in.close();
143       double xT = 0, yT = 0, zT = 0;
144       double xtras = asse[0].getPunto(378).x;
145       for (i = 0; i <= 17; i++) {
146          for (j = 1; j <= 378; j++) {
147             xT = asse[i].getPunto(j).x;
148             yT = asse[i].getPunto(j).y;
149             asse[i].getPunto(j).x = (xT)
150                      * Math.cos((-Math.PI)) - (yT)
151                      * Math.sin((-Math.PI)) + xtras;
152             asse[i].getPunto(j).y = (xT)
153                      * Math.sin((-Math.PI)) + (yT)
154                      * Math.cos((-Math.PI));
155          }
156       }
157       for (i = 0; i <= 17; i++) {
158          for (j = 1; j <= 378; j++) {
159             yT = asse[i].getPunto(j).y;
160             zT = asse[i].getPunto(j).z;
161             asse[i].getPunto(j).z = -((zT)
162                      * Math.cos((-Math.PI)) + (yT)
163                      * Math.sin((-Math.PI)));
164             asse[i].getPunto(j).y = -(zT)
165                      * Math.sin((-Math.PI)) + (yT)
166                      * Math.cos((-Math.PI));
167          }
168       }
169    }
170
171    /**
172     * Genera i punti della lente destra o
173     * sinistra.
174     *
```

```
175     * @param file F
176     * @throws FileNotFoundException
177     */
178    public void generaAsse(String file) throws FileNotFoundException {
179       FileReader reader = new FileReader(file);
180       Scanner in = new Scanner(reader);
181       in.nextLine();
182       in.nextLine();
183       in.nextLine();
184       int i, j;
185       // LENTE DX
186       double x = 0, y = 0, z = 0;
187       double xT = 0, yT = 0, zT = 0;
188       for (i = 0; i <= 17; i++) { // X
189          for (j = 1; j <= 189; j++) {
190             x = in.nextDouble() / 100000;
191             asseSx[i].getPunto(j).x = x;
192             asseSx[i].getPunto(j + 189).x = x;
193          }
194          // Y
195          for (j = 1; j <= 189; j++) {
196             y = in.nextDouble() / 100000;
197             asseSx[i].getPunto(j).y = y;
198             asseSx[i].getPunto(j + 189).y = y;
199          }
200          // Z
201          for (j = 190; j <= 378; j++) {
202             z = in.nextDouble() / 100000;
203             asseSx[i].getPunto(j).z = z;
204          }
205          for (j = 1; j <= 189; j++) {
206             z = in.nextDouble() / 100000;
207             asseSx[i].getPunto(j).z = z;
208          }
209          in.nextDouble();
210          in.nextLine();
211       }
212       // LENTE SX
213       for (i = 0; i <= 17; i++) { // X
214          for (j = 1; j <= 189; j++) {
215             x = in.nextDouble() / 100000;
216             asseDx[i].getPunto(j).x = x;
217             asseDx[i].getPunto(j + 189).x = x;
218          }
219          // Y
220          for (j = 1; j <= 189; j++) {
221             y = in.nextDouble() / 100000;
222             asseDx[i].getPunto(j).y = y;
223             asseDx[i].getPunto(j + 189).y = y;
224          }
225          // Z
226          for (j = 190; j <= 378; j++) {
227             z = in.nextDouble() / 100000;
228             asseDx[i].getPunto(j).z = z;
229          }
230          for (j = 1; j <= 189; j++) {
231             z = in.nextDouble() / 100000;
232             asseDx[i].getPunto(j).z = z;
233          }
234          if (i == 17) {
235          } else {
```

```
236             in.nextDouble();
237          }
238       }
239       in.close();
240       double xtras = Math.sqrt((asseDx[0].getPunto(1).x
241               * asseDx[0].getPunto(1).x)
242               + (asseSx[0].getPunto(1).x
243               * asseSx[0].getPunto(1).x));
244       for (i = 0; i <= 17; i++) {
245          for (j = 1; j <= 378; j++) {
246             xT = asseDx[i].getPunto(j).x;
247             yT = asseDx[i].getPunto(j).y;
248             asseDx[i].getPunto(j).x = (xT)
249                     * Math.cos((-Math.PI)) - (yT)
250                     * Math.sin((-Math.PI)) + xtras;
251             asseDx[i].getPunto(j).y = (xT)
252                     * Math.sin((-Math.PI)) + (yT)
253                     * Math.cos((-Math.PI));
254          }
255       }
256       for (i = 0; i <= 17; i++) {
257          for (j = 1; j <= 378; j++) {
258             yT = asseDx[i].getPunto(j).y;
259             zT = asseDx[i].getPunto(j).z;
260             asseDx[i].getPunto(j).z = -((zT)
261                     * Math.cos((-Math.PI)) + (yT)
262                     * Math.sin((-Math.PI)));
263             asseDx[i].getPunto(j).y = -(zT)
264                     * Math.sin((-Math.PI)) + (yT)
265                     * Math.cos((-Math.PI));
266          }
267       }
268    }
269
270    public void traslaSotto() {
271       double yT = 0;
272       for (int i = 0; i <= 17; i++) {
273          for (int j = 1; j <= 378; j++) {
274             yT = asse[i].getPunto(j).y;
275             asse[i].getPunto(j).y = yT - ytras;
276
277          }
278       }
279    }
280
281    public void traslaSottoDoppio() {
282       double yT = 0;
283       for (int i = 0; i <= 17; i++) {
284          for (int j = 1; j <= 378; j++) {
285             yT = asseDx[i].getPunto(j).y;
286             asseDx[i].getPunto(j).y = yT - ytras;
287             yT = asseSx[i].getPunto(j).y;
288             asseSx[i].getPunto(j).y = yT - ytras;
289
290          }
291       }
292    }
293
294    public void traslaSopra() {
295       double yT = 0, su = 0.25;
296       for (int i = 0; i <= 17; i++) {
```

```
297          for (int j = 1; j <= 378; j++) {
298             yT = asse[i].getPunto(j).y;
299             asse[i].getPunto(j).y = yT + ytras + su;
300
301          }
302       }
303    }
304
305    public void traslaSopraDoppio() {
306       double yT = 0, su = 0.25;
307       for (int i = 0; i <= 17; i++) {
308          for (int j = 1; j <= 378; j++) {
309             yT = asseDx[i].getPunto(j).y;
310             asseDx[i].getPunto(j).y = yT + ytras + su;
311             yT = asseSx[i].getPunto(j).y;
312             asseSx[i].getPunto(j).y = yT + ytras + su;
313          }
314       }
315    }
316 }
```

La classe **GeneratoreTransformGroupLenti.java** contiene il metodo getTranform-Group() che restituisce l'output finale: un insieme di shape3D che formano il Transform-Group.

Questi oggetti contengono le informazioni sulla geometria e sui vettori normali delle superfici e vengono creati nel metodo shape3DConVettoriNormali(Pointe3D[ ] poligono) che prende in ingresso un array contenente i punti che descrivono un poligono, nel caso particolare sono quadrilateri. Viene utilizzata la classe QuadArray di Java3D per la definizione di queste aree infinitesime.

```
1 /**
2  * Crea la TransformGrouo che descrive la lente.
3  *
4  * @author Marco Matta
5  */
6 public final class GeneratoreTransformGroupLenti {
7
8   private final TransformGroup myGroup;
9   private final GeneratorePunti gen;
10  private final Material material;
11  private final TransparencyAttributes trasparenza;
12
13  /**
14   * Legge il file dei punti e crea la geometria.
15   *
16   * @param fileDati3D
17   * @throws FileNotFoundException
18   */
19  public GeneratoreTransformGroupLenti(String fileDati3D)
20          throws FileNotFoundException {
21    // GESTIONE MATERIALE
22    Color3f blu = new Color3f(.0f, .0f, 1.0f);
23    Color3f black = new Color3f(.0f, .0f, .0f);
24    Color3f white = new Color3f(1.0f, 1.0f, 1.0f);
25    material = new Material(blu, black, blu, white, 90.0f);
26    // GESTIONE TRASPARENZA
27    trasparenza = new TransparencyAttributes();
28    trasparenza.setTransparency(0.3f);// (0.3f)
29    trasparenza.setTransparencyMode(TransparencyAttributes.BLENDED);
30    Transform3D myTrans3D = new Transform3D();
31    myTrans3D.set(new Vector3d(0.0, 0.15, 0.0));
32    myTrans3D.setScale(0.5828); // 1280x720 16:09 0.5625   ok
33    myGroup = new TransformGroup(myTrans3D);
34    gen = new GeneratorePunti();
35
36    try {
37      try {
38        gen.generaAsse(fileDati3D);
39      } catch (FileNotFoundException err) {
40        System.err.println("File3D non trovato");
41      }
42
43      for (int i = 0; i <= 17; i++) {
44        if (i == 17) {
45          myGroup.addChild(disegnaLaGeometriaSx(i, 0));
46          myGroup.addChild(disegnaLaGeometriaDx(i, 0));
47          myGroup.addChild(disegnaBordoSx(i, 0));
48          myGroup.addChild(disegnaBordoDx(i, 0));
49        } else {
```

```
50            myGroup.addChild(disegnaLaGeometriaSx(i, i + 1));
51            myGroup.addChild(disegnaLaGeometriaDx(i, i + 1));
52            myGroup.addChild(disegnaBordoSx(i, i + 1));
53            myGroup.addChild(disegnaBordoDx(i, i + 1));
54          }
55        }
56      } catch (NoSuchElementException err) {
57        FileReader reader = new FileReader(fileDati3D);
58        Scanner in = new Scanner(reader);
59        in.nextLine();
60        in.nextLine();
61        in.nextDouble();
62        double k = in.nextDouble();
63        in.close();
64        if (k < 0) { // Lente dx da sola
65          try {
66            gen.generaAsseDx(fileDati3D);
67          } catch (FileNotFoundException err1) {
68            System.err.println("File3D non trovato");
69          }
70
71          for (int i = 0; i <= 17; i++) {
72            if (i == 17) {
73              myGroup.addChild(disegnaLaGeometriaDxSolo(i, 0));
74              myGroup.addChild(disegnaBordoDxSolo(i, 0));
75            } else {
76              myGroup.addChild(disegnaLaGeometriaDxSolo(i, i + 1));
77              myGroup.addChild(disegnaBordoDxSolo(i, i + 1));
78            }
79          }
80        } else if (k > 0) { // Lente sx da sola
81          try {
82            gen.generaAsseSx(fileDati3D);
83          } catch (FileNotFoundException err1) {
84            System.err.println("File3D non trovato");
85          }
86          for (int i = 0; i <= 17; i++) {
87            if (i == 17) {
88              myGroup.addChild(disegnaLaGeometriaSxSolo(i, 0));
89              myGroup.addChild(disegnaBordoSxSolo(i, 0));
90            } else {
91              myGroup.addChild(disegnaLaGeometriaSxSolo(i, i + 1));
92              myGroup.addChild(disegnaBordoSxSolo(i, i + 1));
93            }
94          }
95        }
96      }
97    }
98
99    /**
100    * Costruzione di un shape3D associando ad ogni
101    * poligono le normali alla superficie in modo
102    * da poter utilizzare le propieta di
103    * illuminazione.
104    *
105    * @param poligono
106    * @return shape3d che descrive un pezzo di
107    * lente
108    */
109   private Shape3D shape3dConVettoriNormali(Point3d[] poligono) {
110     GeometryInfo geometryInfo =
```

```
111             new GeometryInfo(GeometryInfo.QUAD_ARRAY);
112     geometryInfo.setCoordinates(poligono);
113     NormalGenerator normalGenerator = new NormalGenerator();
114     normalGenerator.generateNormals(geometryInfo);
115     geometryInfo.recomputeIndices();
116     Stripifier stripifier = new Stripifier();
117     stripifier.stripify(geometryInfo);
118     geometryInfo.recomputeIndices();
119     Shape3D shapeSpicchio = new Shape3D();
120     shapeSpicchio.setGeometry(geometryInfo.getGeometryArray());
121     Appearance appearanceSpicchio = new Appearance();
122     appearanceSpicchio.setMaterial(material);
123     appearanceSpicchio.setTransparencyAttributes(trasparenza);
124     shapeSpicchio.setAppearance(appearanceSpicchio);
125     return shapeSpicchio;
126   }
127
128   private Shape3D disegnaLaGeometriaDx(int i, int j) {
129     Point3d[] spicchio = spicchioLenteSx(i, j);
130     return shape3dConVettoriNormali(spicchio);
131   }
132
133   private Shape3D disegnaBordoDx(int i, int j) {
134     Point3d[] bordoSx = bordoLenteSx(i, j);
135     return shape3dConVettoriNormali(bordoSx);
136   }
137
138   //Disegna la lente sx
139   private Shape3D disegnaLaGeometriaSx(int i, int j) {
140     Point3d[] spicchio = spicchioLenteDx(i, j);
141     return shape3dConVettoriNormali(spicchio);
142   }
143
144   private Shape3D disegnaBordoSx(int i, int j) {
145     Point3d[] bordoDx = bordoLenteDx(i, j);
146     return shape3dConVettoriNormali(bordoDx);
147   }
148
149   //Disegna la lente dx da sola
150   private Shape3D disegnaLaGeometriaDxSolo(int i, int j) {
151     Point3d[] spicchio = spicchioLenteDxSolo(i, j);
152     return shape3dConVettoriNormali(spicchio);
153   }
154
155   private Shape3D disegnaBordoDxSolo(int i, int j) {
156     Point3d[] bordo = bordoLenteDxSolo(i, j);
157     return shape3dConVettoriNormali(bordo);
158   }
159
160   //Disegna la lente sx da sola
161   private Shape3D disegnaLaGeometriaSxSolo(int i, int j) {
162     Point3d[] spicchio = spicchioLenteSxSolo(i, j);
163     return shape3dConVettoriNormali(spicchio);
164   }
165
166
167   private Shape3D disegnaBordoSxSolo(int i, int j) {
168     Point3d[] bordo = bordoLenteSxSolo(i, j);
169     return shape3dConVettoriNormali(bordo);
170   }
171
```

```
172    /**
173     * Ritorna l'occhiale finito.
174     *
175     * @return
176     */
177    public TransformGroup getTransformGroup() {
178       return myGroup;
179    }
```

E' stato possibile gestire le proprietà della trasparenza della lente grazie alla definizione dei vettori normali alle superfici, infatti quando il TransformGroup verrà posto all'interno di un Canvas3D sarà necessario utilizzare delle sorgenti di luce per vedere l'oggetto.

In Java3D l'intensità dell'illuminazione dipende dall'angolo tra il vettore normale ad una superficie ed il vettore che rappresenta la luce: l'intensità è massima quando i due vettori giacciono lungo la stessa direzione con verso opposto. Perciò mentre un oggetto ruota l'angolo varia e anche il modo in cui le sue superfici infinitesime si mostrano nell'universo 3D migliorando così il rendering.

3.3 Il package Pegaso3D

Il package Pegaso3D contiene la classe main **Universo.java**, che estende JFrame. All'interno del Frame viene aggiunto il componente Canvas3D che rappresenta l'universo nel quale si possono inserire i TransformGroup.

Una volta creato e inserito l'occhiale vengono aggiunte due sorgenti luminose: le DirectionalLight.

I TransformGroup sono oggetti che posso essere ruotati e/o traslati nello spazio, e grazie ai Behavior le varie trasformazioni spaziali vengono controllate direttamente dal muose posizionandosi sull'occhiale: con la pressione del tasto dello scroll muovendo il mouse l'oggetto viene ridimensionato mentre tenendo premuto il tasto destro è possibile controllare la rotazione.

```
1 /**
2  * Frame che contiene l'Universo3D per la
3  * visualizzazione delle lenti oftalmiche.
4  *
5  * @author Marco
6  */
7 public class Universo3D extends JFrame {
8
9    private static final long serialVersionUID = 1L;
10   static String FILE_DATI_3D = "FileDati3D.seq";
11
12   public Universo3D(String file_par) throws FileNotFoundException {
13      Image icon = Toolkit.getDefaultToolkit().getImage("Eye.JPG");
14      this.setIconImage(icon);
15      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16      Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
17
18      Dimension frameSize = this.getSize();
19      this.setSize(750, 500);
20      this.setLocation((screenSize.width - frameSize.width) / 2 - 325,
21              (screenSize.height - frameSize.height) / 2 - 250);
22      this.setResizable(false);
23      this.setTitle("VisualizzaSpessori 3D - Pegaso s.r.l 2011");
24      setLayout(new BorderLayout());
25
26      // impostazioni universo
27      GraphicsConfiguration config =
28              SimpleUniverse.getPreferredConfiguration();
29      Canvas3D canvas = new Canvas3D(config);
30      JMenuBar barraMenu = new JMenuBar();
31      try {
32         FileReader reader = new FileReader(FILE_DATI_3D);
33         Scanner in = new Scanner(reader);
34         String lente = in.nextLine();
35         JMenu Menu1 = new JMenu("Dati occhiale");
36         JMenuItem Etichetta1 = new JMenuItem(lente);
37         Menu1.add(Etichetta1);
38         in.close();
39         barraMenu.add(Menu1);
40      } catch (FileNotFoundException err) {
41         System.err.println("File3D non trovato");
42      }
43      this.setJMenuBar(barraMenu);
44      add("Center", canvas);
45      Background sfondo = new Background(new Color3f(0.85f, 0.85f, 0.85f));
```

```
46      BoundingSphere bounds = new BoundingSphere(new Point3d(0.0, 0.0, 0.0)
            ,
47              Double.MAX_VALUE);
48      sfondo.setApplicationBounds(bounds);
49      BranchGroup group = new BranchGroup();
50      group.addChild(sfondo);
51      group.addChild(luce());
52      group.addChild(luce2());
53      GeneratoreTransformGroupLenti occhiale =
54              new GeneratoreTransformGroupLenti(FILE_DATI_3D);
55      TransformGroup myGroup = occhiale.getTransformGroup();
56      group.addChild(myGroup);
57
58      SimpleUniverse universo3D = new SimpleUniverse(canvas);
59      universo3D.getViewingPlatform().setNominalViewingTransform();
60
61      PickRotateBehavior pickRotateBehavior = new PickRotateBehavior(group,
            canvas,
62              new BoundingSphere());
63
64      PickZoomBehavior pickZoomBehavior = new PickZoomBehavior(group,
            canvas,
65              new BoundingSphere());
66      group.addChild(pickRotateBehavior);
67      group.addChild(pickZoomBehavior);
68      PickingCallbackClass mcpc = new PickingCallbackClass();
69      pickRotateBehavior.setupCallback(mcpc);
70      pickZoomBehavior.setupCallback(mcpc);
71      universo3D.getViewer().getView().setFrontClipDistance(0.1);
72      myGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
73      myGroup.setCapability(TransformGroup.ENABLE_PICK_REPORTING);
74      group.compile();
75      universo3D.addBranchGraph(group);
76  }
77
78  /**
79   * Creazione luce
80   *
81   * @return DirectionalLight
82   */
83  public DirectionalLight luce() {
84      Color3f colore = new Color3f(.1f, .6f, .6f);
85      BoundingSphere raggio = new BoundingSphere(new Point3d(0.0, .0, .0),
86              2000.0);
87      Vector3f direzione = new Vector3f(-0.4f, 0.0f, -2.8f);
88      DirectionalLight nuovaLuce = new DirectionalLight(colore, direzione);
89      nuovaLuce.setInfluencingBounds(raggio);
90      return nuovaLuce;
91  }
92
93  /**
94   * Creazione luce
95   *
96   * @return DirectionalLight
97   */
98  public DirectionalLight luce2() {
99      Color3f colore = new Color3f(.1f, .6f, .6f);
100     BoundingSphere raggio = new BoundingSphere(new Point3d(0.0, .0, .0),
101             2000.0);
102     Vector3f direzione = new Vector3f(0.4f, 0.0f, -2.8f);
103     DirectionalLight nuovaLuce = new DirectionalLight(colore, direzione);
```

```
104         nuovaLuce.setInfluencingBounds(raggio);
105         return nuovaLuce;
106      }
107
108      /**
109       * Metodo Main
110       *
111       * @throws FileNotFoundException
112       */
113      public static void main(String[] args) throws FileNotFoundException {
114         Universo3D base = new Universo3D(FILE_DATI_3D);
115         base.setVisible(true);
116      }
117   }
```

Capitolo 4

Requisiti e installazione

Java3D 1.5 viene rilasciato sotto licenza della Sun Microsystems, la libreria può essere utilizzata per scopi individuali, commerciali e di ricerca.

Per sistemi operativi Windows con OpenGL, per lo sviluppo del software, i requisiti minimi di installazione sono i seguent:

- Windows NT 4.0 (Service Pack 3), Windows 98, Windows ME, Windows 2000;
- Java2 SDK 1.3.1 o superiori (Sun Microsystems), questo include il Java Plug-In (JPI) per l'interazione con il browser;
- OpenGL 1.1 (Microsoft) o acceleratori grafici che supportano OpenGL 1.1.

Il software è stato testato in Windows Xp, Vista e Seven mentre in ambiente Linux in Ubuntu 10.04 LTS e 12.04 LTS.

Nel computer del cliente deveno essere installati: il JRE Java Runtime Environment (in questo modo il file jar viene riconosciuto come eseguibile), Java3D e OpenGL.

Nel caso del pacchetto Argo il file jar verrà richiamato da una Form di Visual Basic, in modo da far interagire la visualizzazione 3D con il resto delle funzionalità già sviluppate nel software aziendale.

Capitolo 5

Conclusioni

La scelta dell'utilizzo di Java3D per lo sviluppo di un software di visualizzazione di lenti oftalmiche ha portato alla realizzazione di un programma semplice, performante e con un buon rendering con tempi di sviluppo brevi.

I risultati ottenuti sono soddisfacenti e gli obiettivi posti all'inizio dell'esperienza sono stati conseguiti: il software ha ricevuto un buon apprezzamento dal mercato.

L'idea di poter visualizzare in 3D una lente potrebbe evolversi nella visualizzazione contemporanea di vari tipi di lenti per un confronto immediato su determinati parametri con lo scopo di fornire agli oftalmici uno strumento di lavoro semplice e innovativo.

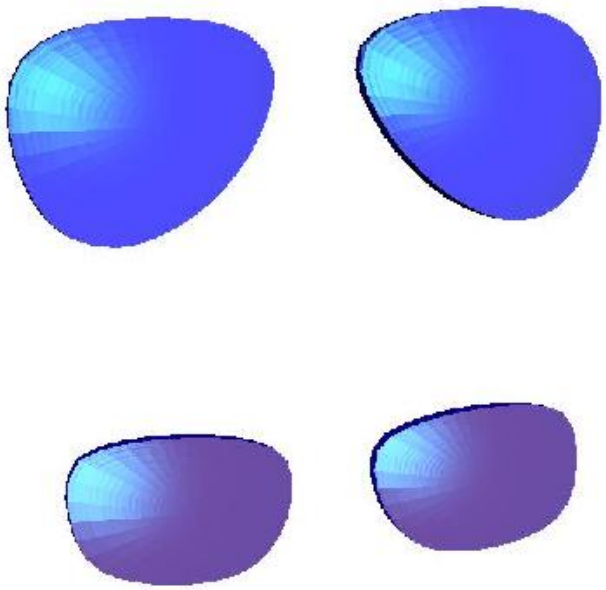

Figura 5.1: Esempio di applicazione contenente due occhiali diversi.

Bibliografia

[1] Rossetti A. (2003), *Lenti e occhiali. Un manuale di ottica oftalmica.*

[2] Milanese Francesco (2010), *Java 3D - Guida di base*, `www.redbaron85.com`.

[3] `http://it.wikipedia.org/wiki/Java_3D`.