

Java 3D

Guida di base

Francesco “RedBaron85” Milanese
www.redbaron85.com – 2010

Java 3D è un ricco set di API di programmazione di livello medio-alto che consente di creare facilmente applicazioni con grafica tridimensionale interattiva.

Questa guida parte dalle basi teoriche dello Scene Graph per poi trattare, mediante esempi pratici, la creazione, il posizionamento e l'animazione di oggetti nell'universo virtuale di Java 3D, oltre alla gestione della loro resa visiva, l'interazione mediante mouse e tastiera, l'animazione ed altro ancora.

Francesco Milanese ha tenuto un ciclo di seminari sull'argomento presso l'Università degli Studi di Catania durante l'A.A. 2008-09.

E' un Blender Foundation Certified Trainer ed ha realizzato e gestisce il sito www.redbaron85.com, dove pubblica periodicamente tutorial e guide riguardanti la Computer Grafica 3D.

SOMMARIO

Java 3D	12
LA COMPUTER GRAFICA 3D	12
LE API DI PROGRAMMAZIONE	13
JAVA 3D	14
DI COSA ABBIAMO BISOGNO ?	14
STRUTTURA DELLA GUIDA	15
UN SEMPLICE ESEMPIO 2D “FINTO 3D”: WITU	17
 Lo Scene Graph. Il Simple Universe	19
STRUTTURE ED ELEMENTI DELLO SCENE GRAPH	20
ELEMENTI DELLO SCENE GRAPH – IL VIRTUAL UNIVERSE	20
ELEMENTI DELLO SCENE GRAPH – IL LOCALE	21
ELEMENTI DELLO SCENE GRAPH – LA SUPERCLASSE NODE	21
ELEMENTI DELLO SCENE GRAPH – I GRUPPI	22
ELEMENTI DELLO SCENE GRAPH – LA SUPERCLASSE LEAF	22
ELEMENTI DELLO SCENE GRAPH – LA SUPERCLASSE NODECOMPONENT	22
RELAZIONI PADRE-FIGLIO E RELAZIONI REFERENCE	23
IL SIMPLE UNIVERSE	23
COMPILARE UN BRANCH GRAPH	25
LE CAPABILITY	25
OGGETTI “VIVI”	26
IL SISTEMA DI RIFERIMENTO DI JAVA 3D	26
UN PRIMO ESEMPIO: UN UNIVERSO VUOTO	27
SECONDO ESEMPIO: POPOLARE L’UNIVERSO	28
 Il Simple Universe in dettaglio	30
PERSONALIZZARE IL SIMPLE UNIVERSE	30
IL LOCALE	31
ELEMENTI DEL VIEW BRANCH GRAPH. OPERAZIONI.	32
LE NORMALI	34
IL CLIPPING	35
L’ESEMPIO “ESEMPIO SIMPLE UNIVERSE”	36
BOUND E SCOPE: INTRODUZIONE	36
BOUND E BOUNDING LEAF	37
SCOPE, CON UN PICCOLO INTRUSO	38
IL BACKGROUND	39
ANCORA SU BOUND E BACKGROUND	39
 Le classi “matematiche” in Java 3D	42
MATEMATICA 3D: CENNI	42
LE CLASSI MATEMATICHE IN JAVA E IN JAVA 3D	45

Gruppi. Oggetti visuali di base e da file	48
I GRUPPI: BRANCH E TRANSFORM	49
GLI OGGETTI VISUALI	50
GEOMETRIE DI BASE	51
GEOMETRIE DI BASE – COLORCUBE	51
GEOMETRIE DI BASE – BOX	52
GEOMETRIE DI BASE – CONE	52
GEOMETRIE DI BASE – CYLINDER	53
GEOMETRIE DI BASE – SPHERE	53
GEOMETRIE DI BASE – TEXT2D	54
GEOMETRIE DI BASE – TEXT 3D	55
GEOMETRIE DA FILE	57
Geometry e le sue sottoclassi	58
GEOMETRY E LE SUE SOTTOCLASSI: PANORAMICA	59
GEOMETRYARRAY E LE SUE SOTTOCLASSI	59
SOTTOCLASSI DI GEOMETRY ARRAY	60
SOTTOCLASSI DI GEOMETRYARRAY – POINTARRAY	61
SOTTOCLASSI DI GEOMETRYARRAY – LINEARRAY	62
SOTTOCLASSI DI GEOMETRYARRAY – TRIANGLEARRAY	63
SOTTOCLASSI DI GEOMETRYARRAY – QUADARRAY	64
SOTTOCLASSI DI GEOMETRYARRAY – GEOMETRYSTRIPARRAY	64
SOTTOCLASSI DI GEOMETRYSTRIPARRAY – LINESTRIPARRAY	65
SOTTOCLASSI DI GEOMETRYSTRIPARRAY – TRIANGLESTRIPARRAY	66
SOTTOCLASSI DI GEOMETRYSTRIPARRAY – TRIANGLEFANARRAY	67
INDEXEDGEOMETRYARRAY E LE SUE SOTTOCLASSI	68
GEOMETRYINFO	69
GEOMETRY INFO: UN ESEMPIO	71
RASTER	73
Le trasformazioni	74
TRASLAZIONI	75
ROTAZIONI	76
RIDIMENSIONAMENTI (SCALING)	77
IL PIVOTING	77
TRASLAZIONI E PIVOTING	78
ROTAZIONI E PIVOTING	79
SCALING IN CASCATA	81
UN ESEMPIO COMPLETO	82
L'interazione	83
I BEHAVIOR	84
CONDIZIONI DI WAKEUP	86
CREARE DEI SEMPLICI BEHAVIOR: ESEMPI	87
KEYNAVIGATORBEHAVIOR	88
BOUNDINGLEAF E BOUND DI DIMENSIONE INFINITA	90
BEHAVIOR PER IL MOUSE	92
MOUSE BEHAVIOR	93

IL PICKING	93
PICKMOUSEBEHAVIOR	94
MOUSEBEHAVIOR e PICKMOUSEBEHAVIOR	95
PICKINGCALLBACK	95
USERDATA. ESTENDERE LE CLASSI DI JAVA 3D	97
Le luci	99
I COLORI DEGLI OGGETTI IN JAVA 3D	100
LE LUCI NELLO SCENE GRAPH	100
LA CLASSE LIGHT	100
AMBIENT LIGHT	101
DIRECTIONAL LIGHT	102
POINT LIGHT	103
SPOT LIGHT	104
L'ATTENUAZIONE DELLE LUCI	106
BOUNDING LEAF	107
SCOPE	107
LE OMBRE	109
L'aspetto visivo degli oggetti	111
COLORI DELLE GEOMETRIE	112
I MODELLI DI OMBREGGIATURA	113
APPEARANCE E I SUOI ATTRIBUTI	113
ATTRIBUTI DI APPEARANCE – POINT ATTRIBUTES	115
ATTRIBUTI DI APPEARANCE – LINE ATTRIBUTES	115
ATTRIBUTI DI APPEARANCE – POLYGON ATTRIBUTES	116
ATTRIBUTI DI APPEARANCE – COLORING ATTRIBUTES	116
ATTRIBUTI DI APPEARANCE – TRANSPARENCY ATTRIBUTES	117
UTILIZZARE PIU' TECNICHE: UN ESEMPIO	118
ALTERNATE APPEARANCE	119
MATERIAL	121
LUCI, ASPETTO E NORMALI DELLE GEOMETRIE	122
LE TEXTURES E IL TEXTURE MAPPING	123
CARICARE UNA TEXTURE	124
IMPOSTARE UNA TEXTURE IN UN APPEARANCE	124
IMPOSTARE LE COORDINATE DI UNA TEXTURE	125
TEXCOORDGENERATION	125
IMPOSTARE LE COORDINATE DELLE TEXTURES MANUALMENTE	126
ESEMPI CON LE TEXTURES	128
BOUNDARY MODE	128
PIXEL E TEXEL	129
Animazioni	130
ANIMAZIONI DIPENDENTI DAL PUNTO DI VISTA DELL'OSSERVATORE	131
BILLBOARD	132
ORIENTEDSHAPE3D	133
ANIMAZIONI LOD	135
ANIMAZIONI TIME-BASED	137

ALPHA	139
INTERPOLATOR E LE SUE SOTTOCLASSI	143
SOTTOCLASSI DI INTERPOLATOR – COLOR INTERPOLATOR	144
SOTTOCLASSI DI INTERPOLATOR – TRANSPARENCY INTERPOLATOR	145
SOTTOCLASSI DI INTERPOLATOR – SWITCH VALUE INTERPOLATOR	146
SOTTOCLASSI DI INTERPOLATOR – TRANSFORM INTERPOLATOR	147
SISTEMI DI RIFERIMENTO: LOCALE E GLOBALE	148
SOTTOCLASSI DI TRANSFORM INTERPOLATOR – POSITION INTERPOLATOR	149
SOTTOCLASSI DI TRANSFORM INTERPOLATOR – ROTATION INTERPOLATOR .	150
SOTTOCLASSI DI TRANSFORM INTERPOLATOR – SCALE INTERPOLATOR	151
SOTTOCLASSI DI TRANSFORM INTERPOLATOR – PATH INTERPOLATOR	152
AXISANGLE4D E QUATERNIONI	153
SOTTOCLASSI DI PATH INTERPOLATOR – POSITION PATH INTERPOLATOR	155
SOTTOCLASSI DI PATH INTERPOLATOR – ROTATION PATH INTERPOLATOR	156
SOTTOCLASSI DI PATH INTERPOLATOR – ROTPOSPATH INTERPOLATOR	156
SOTTOCLASSI DI PATH INTERPOLATOR – ROTPOSSCALEPATH INTERPOLATOR	
.....	157
ANIMAZIONI “A COMANDO”	158
ANIMAZIONI MORPH	160
Extra	163
LE COLLISIONI IN JAVA 3D	164
IL METODO GET TRIGGERING PATH E L’OGGETTO SCENE GRAPH PATH	165
SCRIVERE SULLA CANVAS 3D	167
I THREAD IN JAVA	168
ANIMAZIONI MEDIANTE THREAD: UN SEMPLICE ESEMPIO	175
SET TEXTURE MODE DI TEXTURE ATTRIBUTES	178
Appendice	180
APPLICAZIONE JAVA 3D DI BASE	181
WITU.java	183
WITUapp.java	188
PRIMO ESEMPIO	189
SECONDO ESEMPIO	191
NOMINAL VIEWING PLATFORM	193
ESEMPIO SIMPLE UNIVERSE	195
ESEMPIO BACKGROUND	198
BACKGROUND CREATO CON SFERA ENORME	200
BACKGROUND CLASSICO	202
ESEMPIO COLOR CUBE	205
ESEMPIO BOX	207
ESEMPIO CONE	208
ESEMPIO CYLINDER	209
ESEMPIO SPHERE 1	210
ESEMPIO SPHERE 2	211
ESEMPIO TEXT 2D	212
ESEMPIO TEXT 3D	214
ESEMPIO POINT ARRAY	216

ESEMPIO LINE ARRAY QUATTRO VERTICI.....	218
ESEMPIO LINE ARRAY TRE VERTICI.....	220
ESEMPIO SISTEMA DI RIFERIMENTO.....	222
ESEMPIO TRIANGLE ARRAY	225
ESEMPIO QUAD ARRAY	227
ESEMPIO LINE STRIP ARRAY	230
ESEMPIO TRIANGLE STRIP ARRAY.....	233
ESEMPIO TRIANGLE FAN ARRAY	236
ESEMPIO INDEXED LINE ARRAY.....	239
ESEMPIO INDEXED QUAD ARRAY.....	241
ESEMPIO GEOMETRY INFO	243
ESEMPIO RASTER.....	246
DOWNTOWN	248
ROTAZIONE PIVOT 1	250
ROTAZIONE PIVOT 2	252
ESEMPIO SCALING IN CASCATA	255
ESEMPIO ROTO TRAS SCALE	257
ESEMPIO BEHAVIOR 1	259
MIO BEHAVIOR 1	261
ESEMPIO BEHAVIOR 2	264
MIO BEHAVIOR 2	266
ESEMPIO BEHAVIOR NAVIGAZIONE	269
ESEMPIO NAV BEHAVIOR 2.....	271
MIO BEHAVIOR TASTIERA	273
ESEMPIO MOUSE BEHAVIOR.....	276
ESEMPIO PICKING.....	278
ESEMPIO PICKING CALLBACK.....	280
MIA CLASSE DI PICKING CALLBACK	282
ESEMPIO PICKING CALLBACK 2.....	283
MIA CLASSE DI PICKING CALLBACK 2	285
MIO TRANSFORM GROUP	286
ESEMPIO LUCE AMBIENTALE.....	287
ESEMPIO LUCE DIREZIONALE.....	289
ESEMPIO POINT LIGHT	292
ESEMPIO SPOT LIGHT.....	295
ESEMPIO POINT LIGHT CON SCOPE	298
ESEMPIO COLORI DELLE GEOMETRIE 1	301
ESEMPIO COLORI DELLE GEOMETRIE 2	304
ESEMPIO POINT ATTRIBUTES.....	306
ESEMPIO LINE ATTRIBUTES.....	308
ESEMPIO POLYGON ATTRIBUTES	310
ESEMPIO COLORING ATTRIBUTES	312
ESEMPIO TRANSPARENCY ATTRIBUTES	314
ESEMPIO BORDI IN RILIEVO	316
ESEMPIO ALTERNATE APPEARANCE.....	319
ESEMPIO ALTERNATE APPEARANCE APP 2	322
ESEMPIO MATERIAL	326
ESEMPIO TEXTURE 1.....	329

ESEMPIO TEXTURE 2	331
ESEMPIO TEXTURE 3	333
ESEMPIO TEXTURE 4	336
ESEMPIO BOUNDARY MODE	339
ESEMPIO BILLBOARD	342
ESEMPIO BILLBOARD 2	345
ESEMPIO ORIENTED SHAPE 3D	348
ESEMPIO LOD	351
ESEMPIO COLOR INTERPOLATOR	353
ESEMPIO TRANSPARENCY INTERPOLATOR	356
ESEMPIO SWITCH VALUE INTERPOLATOR	359
ESEMPIO POSITION INTERPOLATOR 1	361
ESEMPIO POSITION INTERPOLATOR 2	363
ESEMPIO POSITION INTERPOLATOR 3	366
ESEMPIO ROTATION INTERPOLATOR	369
ESEMPIO SCALE INTERPOLATOR	371
ESEMPIO QUATERNIONI	373
ESEMPIO POSITION PATH INTERPOLATOR	376
ESEMPIO ROTPOSPATHINTERPOLATOR	379
ESEMPIO ROTPOSSCALEPATHINTERPOLATOR	382
ESEMPIO ICBM --- MAIN	385
ESEMPIO ICBM --- GUI	390
ESEMPIO MORPH BEHAVIOR	397
BEHAVIOR COLLISIONE	400
SIMULATORE --- MAIN	401
SIMULATORE --- THREAD CONTROLLO 1	402
SIMULATORE --- THREAD FRAME 1	403
SIMULATORE --- FRAME SIMULATORE 1	404
SIMULATORE --- MIO BEHAVIOR TASTIERA	406
SIMULATORE --- SCENA 3D	410
PIANETA TERRA	412

Capitolo 1

Java 3D

LA COMPUTER GRAFICA 3D

La **Computer Grafica** consiste nel **generare** e **manipolare immagini** mediante il computer.

Fa uso di un insieme di **tecniche** atte a **rappresentare visivamente**, su di un dispositivo, **dati numerici**.

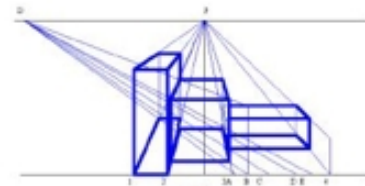
La **Computer Grafica 3D** permette di *descrivere* ambienti ed oggetti dotati di tre dimensioni e di *visualizzarli*, cosa che generalmente avviene su superfici a 2 dimensioni come, ad esempio, lo **schermo** di un pc.

Il tipo di **rappresentazione** da scegliere per i dati dipende, principalmente, dalla natura del destinatario.

Come **descrivere**, ad esempio, un cubo ?

Per il computer utilizzeremo **dati numerici** ((1), nella figura seguente), per gli esseri umani **un'immagine** ((2)), che poi è una **proiezione** di un oggetto 3D su di un piano bidimensionale, calcolata con precisi **strumenti matematici** ((3)).

```
# cube.obj
O
g cube
v 0.0 0.0 0.0
v 0.0 0.0 1.0
v 0.0 1.0 0.0
v 0.0 1.0 1.0
v 1.0 0.0 0.0
v 1.0 0.0 1.0
v 1.0 1.0 0.0
v 1.0 1.0 1.0
vn 0.0 0.0 1.0
vn 0.0 0.0 -1.0
vn 0.0 1.0 0.0
vn 0.0 -1.0 0.0
vn 1.0 0.0 0.0
vn -1.0 0.0 0.0
f 1//2 7//2 5//2
f 1//2 3//2 7//2
f 1//6 4//6 3//6
f 1//6 2//6 4//6
f 3//3 8//3 7//3
f 3//3 4//3 8//3
f 5//5 7//5 8//5
f 5//5 6//5 8//5
f 1//4 5//4 6//4
f 1//4 6//4 2//4
f 2//1 6//1 8//1
f 2//1 8//1 4//1
```



LE API DI PROGRAMMAZIONE

Le **API**, le **Interfacce di Programmazione di un'Applicazione**, costituiscono una serie di **istruzioni e procedure** fornite al **programmatore** al fine di realizzare un dato progetto.

Si tratta di **astrazioni**, fornite da un linguaggio che le implementa, in modo tale che il programmatore possa realizzare i propri programmi con tali strumenti.

E' così che le **API di Java** ci mettono a disposizione classi e metodi per realizzare applicativi provvisti o meno di interfaccia grafica, o un ambiente di programmazione per database ci consente di operare, con certe parole chiave e determinate istruzioni, sulle tabelle e sui campi.

Java3D mette a disposizione un proprio set di **API** per consentire agli sviluppatori di definire, personalizzare e visualizzare **ambienti virtuali 3D 'vivi'** (con i quali si può **interagire**, modificandone gli elementi in tempo reale) su una grande quantità di **dispositivi di output**.

La **documentazione per tali API** è disponibile gratuitamente per il **download** sul sito della **SUN Microsystems**, all'indirizzo web <http://www.sun.com>.

JAVA 3D

Java 3D è un'API di **alto livello** che permette di creare un *universo virtuale*, di riempirlo con **oggetti** (anche animati, utilizzando *trasformatori* ed altri elementi) e **visualizzarlo** su una **Canvas3D** (lett.: 'tela 3D'; si tratta di un oggetto di tipo **AWT**, *simile* ad un pannello, in quanto va inserito in un **Container**, ad esempio un **JFrame**, per visualizzare l'**universo 3D** a video).

Il principale vantaggio di **Java3D** consiste nel risolvere **problemi di basso livello**, lasciando il programmatore libero di personalizzare l'**universo virtuale**.

Java3D è un'API 'nata' lenta e pesante, ma fortunatamente la **potenza dell'hardware** attuale il più delle volte non fa avvertire questo problema.

Come suggerisce il nome, è scritta in **linguaggio Java**, per cui al **programmatore** è richiesta una discreta padronanza di tale **linguaggio**.

DI COSA ABBIAMO BISOGNO ?

Per realizzare i propri **progetti Java3D** è necessario disporre del **kit di sviluppo di Java (Java2 SDK)** e di **Java3D (Java3D SDK)**, entrambi disponibili gratuitamente per il **download** sul sito della **SUN Microsystem**, all'indirizzo web <http://www.sun.com>.

Le **versioni** dei pacchetti software utilizzati per testare tutti gli **esempi** presenti nella **guida** sono:

- **javac 1.6.0 02;**
- **Java3D 1.5.**

Java e **Java3D** sono disponibili per un gran numero di **piattaforme software**, come i sistemi della famiglia **MS Windows**, i sistemi **Linux** o **Apple**.

Dal punto di vista dell'**hardware**, **Java3D** non è particolarmente esigente, almeno per quanto riguarda la **configurazione minima**: è sufficiente un sistema provvisto di una CPU con velocità superiore a 1 Ghz e almeno 256 MB di memoria RAM; a sistemi migliori corrispondono, comunque, prestazioni migliori.

Per la scrittura dei **programmi** è sufficiente un semplice **editor di testo**, anche se si consiglia di far uso di un sistema di **sviluppo IDE**.

STRUTTURA DELLA GUIDA

I **capitoli della guida** accompagnano il lettore alla scoperta dell'**universo virtuale** definito da **Java3D**, partendo dai concetti teorici che ne definiscono **struttura ed elementi** per arrivare a trattare aspetti quali l'**interazione** e le varie forme di **animazione** messe a disposizione da tale linguaggio.

Il **Capitolo 2** introduce il lettore alla **logica dello Scene Graph** e agli elementi che compongono l'**universo virtuale di Java3D**.

Il **Capitolo 3** si occupa di alcuni dettagli dell'oggetto **Simple Universe**, dei concetti di **Normali alle superfici**, **clipping**, **bound**, **scope** e del **Background**.

Il **Capitolo 4** espone alcuni cenni di **matematica 3D** ed un **elenco di classi e metodi** matematici messi a disposizione sia da **Java** che da **Java3D**.

Il **Capitolo 5** descrive i **gruppi e gli oggetti visuali di base**, nativi di **Java3D**, e come fare per **caricare geometrie** definite esternamente e memorizzate su **file**.

Il **Capitolo 6** presenta l'**oggetto Geometry** e le sue sottoclassi, illustrando come poter generare **geometrie 'user defined'**, impostando le coordinate dei singoli vertici e le connessioni tra gli stessi.

Il **Capitolo 7** si occupa delle **trasformazioni** che riguardano gli oggetti **a livello di mesh**: traslazioni, rotazioni e ridimensionamenti.

Il **Capitolo 8** fornisce gli strumenti e le tecniche per permettere all'utente di **interagire** con l'**universo virtuale di Java3D**, modificandolo, mediante i **dispositivi di input** (mouse e tastiera) e non solo.

Il **Capitolo 9** si occupa delle **luci**: verranno presentate le **classi** che permettono di definire **fonti luminose nell'universo virtuale**, trattando anche gli argomenti legati all'**attenuazione**, alle **Bounding Leaves** e allo **Scope**.

Il **Capitolo 10** è dedicata all'**aspetto visivo degli oggetti**: alla definizione di **colori, trasparenze, materiali e textures** da applicare alle **mesh**.

Il **Capitolo 11** tratta un ampio spettro di **animazioni** realizzabili mediante **Java3D**: dalle animazioni dipendenti dal **punto di vista** dell'osservatore alle animazioni **Morph**, passando per quelle **time-based** e **Level of Detail (LoD)**.

Il **Capitolo 12** non è a tema, ma espone alcuni **argomenti 'extra'** utili per la creazione di applicativi **più elaborati**.

In **Appendice**, infine, si trova il **codice sorgente** (o porzioni di esso) per i vari **esempi** proposti.

Questa **guida non** sostituisce la **documentazione di Java3D**.

Delle varie **classi** trattate verranno mostrati i **costruttori ed i metodi più importanti**, ma per un elenco dettagliato si rimanda alla **documentazione ufficiale** del linguaggio, disponibile gratuitamente per il **download** sul sito della **SUN Microsystem**, all'indirizzo web <http://www.sun.com>.

In **Appendice**, il primo **esempio di codice** presente è quello dell'**applicazione Java 3D di base**, che definisce un **semplice universo virtuale** ove è possibile **navigare mediante i tasti freccia**.

Il lettore può utilizzare tale esempio come **'base di partenza'** già pronta per creare e testare velocemente i propri scenari.

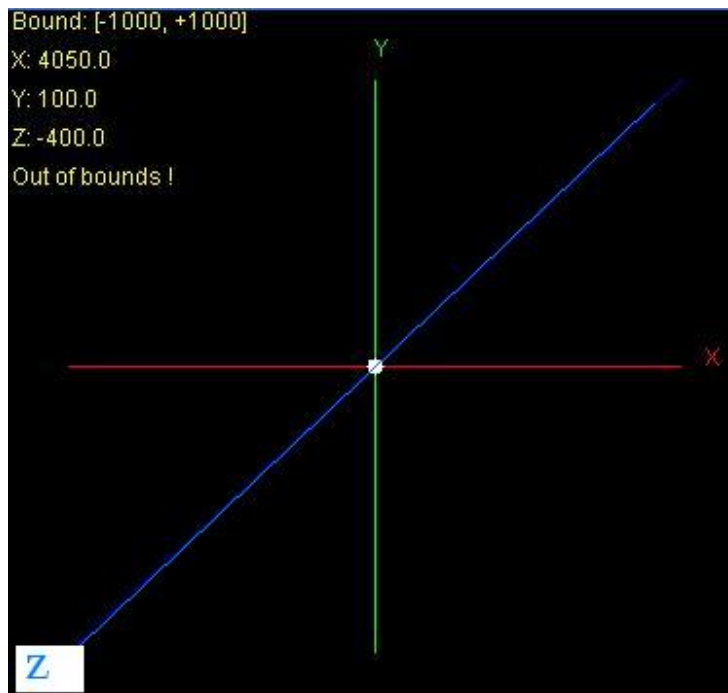
UN SEMPLICE ESEMPIO 2D “FINTO 3D”: WITU

Per familiarizzare con la **rappresentazione 2D** di un '**universo virtuale**' 3D viene qui presentata una piccola applicazione, una **minimappa** fornita di **tre assi** disposti in modo tale da rappresentare il **sistema di riferimento di Java3D**: *Y verso l'alto, X positivo a destra e Z positivo uscente dallo schermo*.

L'applicazione, chiamata '**WITU**' (**Where In The Universe**), è sostanzialmente un piccolo pannello **Java2D**, ridimensionabile e personalizzabile, che, presi in input i **valori delle coordinate X, Y e Z** di un oggetto (o dell'osservatore) in un **universo Java3D**, mostra una sferetta in una **minimappa**.

La sferetta rappresenta proprio l'oggetto o l'osservatore, per cui l'applicazione si comporta come un **pannello** del tipo '**voi siete qui**' (da qui, infatti, il nome **WITU**).

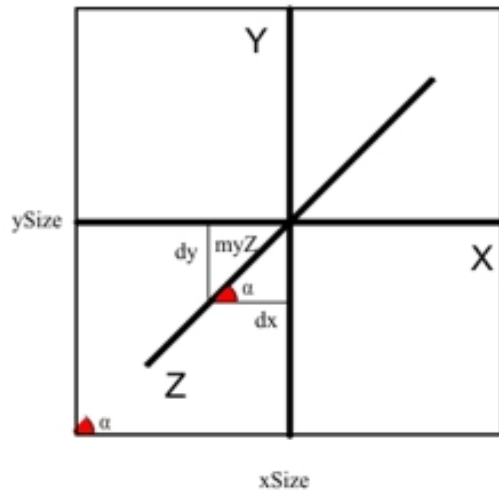
Nel pannello, un pallino bianco identificherà l'**origine** (coordinate (0.0, 0.0, 0.0)), mentre in alto a sinistra verranno mostrate le **coordinate 3D 'reali'** (cioè quelle in arrivo dalla Canvas3D) e un **avviso di 'out of bounds'**.



La **lunghezza, in pixel, degli assi X e Y** è pari rispettivamente a $(width/10)*8$ e $(height/10)*8$, la lunghezza dell'asse Z è calcolata mediante il **teorema di Pitagora**.

A causa delle dimensioni finite del Pannello, è possibile mostrare la posizione di un oggetto solo se questo si muove all'interno di uno **spazio dalle dimensioni finite**, chiamato '**bounding box**'; nel nostro caso, quindi, dobbiamo definire una '**bounding size**' (che poi equivarrà alla lunghezza dell'asse X, quindi la posizione 3D reale verrà tradotta, con una semplice proporzione e un paio di formule trigonometriche, in una **posizione 3D in scala**).

Serve un pò di **trigonometria** per **calcolare le 'finte' coordinate di Z** (dette, nel codice, *tmpZx* e *tmpZy*).



$$\alpha = \arctg\left(\frac{ySize}{xSize}\right)$$

$$dx = myZ * \cos(\alpha)$$

$$dy = myZ * \sin(\alpha)$$

In **Appendice** sono presenti i **codici sorgente** di due files: *WITU.java* e *WITUapp.java*.

Nel file sorgente '*WITU.java*' sono presenti, commentate, tutte le istruzioni che permettono di **simulare lo spazio 3D proiettandolo**, mediante semplici formule, su di un **pannello bidimensionale**.

La **classe WITU** contiene semplicemente il **pannello**, da associare ad un'applicazione che gli invii i dati delle **coordinate 3D**.

Il file '*WITUapp.java*' definisce appunto un piccolo **moke object** che crea un **pannello WITU** ed esegue alcune operazioni sullo stesso: sostanzialmente, **invia dei valori X, Y e Z** e li modifica nel tempo.

Il risultato di tali operazioni viene mostrato **in tempo reale dal pannellino WITU**.

Il lettore potrà, in seguito, **incorporare WITU** (che, tra l'altro, mette a disposizione metodi per la **personalizzazione dell'aspetto**, modificando ad esempio il colore o le etichette degli assi, modificando il colore della sfera rappresentante l'osservatore, disabilitando le scritte ecc...) nei propri progetti, senza penalizzarli in termini di **performances**, in quanto si tratta di una semplice **applicazione 2D**, non di una **Canvas3D**.

* * *
* * *

Capitolo 2

Lo Scene Graph. Il Simple Universe

Le **API di Java 3D** ci forniscono un **universo virtuale** ed un modo per **visualizzarlo** sulla cosiddetta **View Platform**.

Una implementazione di un **universo di Java 3D**, con tutto ciò che può contenere (oggetti, trasformazioni, luci, View Platform, ...), viene **rappresentata schematicamente** mediante un particolare **albero**, detto **Scene Graph**, comprendente alcuni **nodi** sempre presenti (ad esempio la radice) e **nodi 'gruppo' o 'oggetto'**, definiti dal programmatore, che costituiscono una **'scena'**.

Lo **Scene Graph** è dunque un albero, da realizzare **'con carta e penna'** prima di passare all'**implementazione** vera e propria mediante **codice**, in modo da avere sempre a portata di mano uno **schema del progetto**.

Il passaggio **dallo Scene Graph all'implementazione**, una volta note le **API** di programmazione di **Java 3D**, è molto semplice, paragonabile all'**implementare classi, campi e metodi** partendo da **diagrammi UML**.

STRUTTURE ED ELEMENTI DELLO SCENE GRAPH

Un **universo virtuale** creato mediante **Java 3D** ha degli **elementi ricorrenti**, relazionati tra di loro in modo ben definito e praticamente immutabile.

Tali elementi sono ben visibili ed individuabili negli **Scene Graph** di tutte le applicazioni che utilizzano **Java 3D**; è quindi possibile creare dei **gruppi 'logici'**, identificando delle **strutture ricorrenti**.

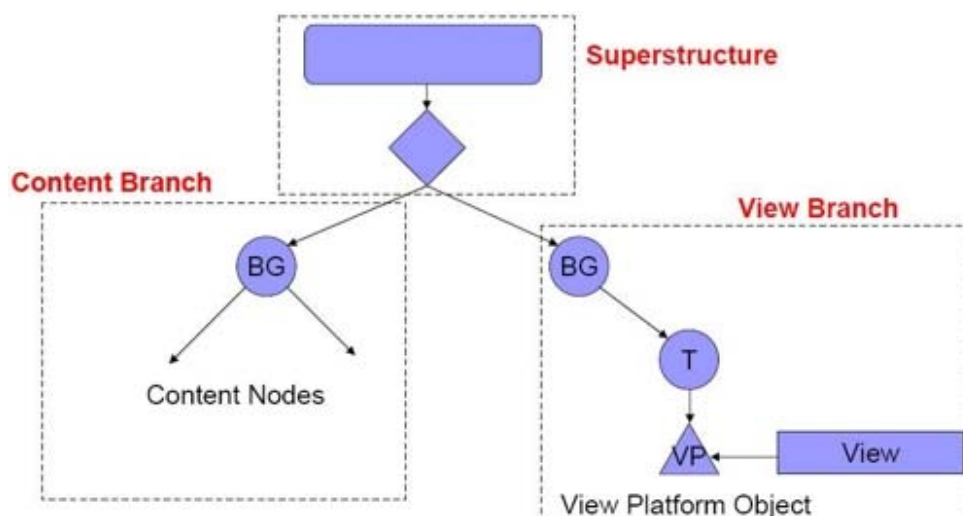
La **Superstruttura** di un **universo virtuale Java 3D** è costituita da un **Virtual Universe** e da uno o più **oggetti Locale**.

'Scendendo' nell'albero, dal **Locale** hanno origine due nodi particolari: i **nodi Branch Group**, che sono **radici di due sotto-alberi**, detti **Branch Graph**.

I due **Branch Graph** sono il **Content Branch** e il **View Branch**; come suggeriscono i nomi, il primo contiene le **definizioni di oggetti e trasformazioni** da inserire nell'**universo virtuale**, mentre il secondo si occupa della **visualizzazione** (non in termini di resa visiva, cioè colori e texture degli oggetti della scena 3D, ma di meccanismi per **mostrare**, a video o su qualunque altro **dispositivo**, l'universo virtuale) del **nostro universo 3D**.

ELEMENTI DELLO SCENE GRAPH – IL VIRTUAL UNIVERSE

La **radice dello Scene Graph** è sempre un oggetto **Virtual Universe**, rappresentato con un **rettangolo dai bordi arrotondati**, come visibile nella figura seguente.

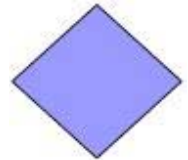


Gli **esempi** presentati in questa guida faranno uso di un particolare oggetto, il **Simple Universe**, che incorpora il **Virtual Universe** ed altri elementi di base dello **Scene Graph**, per cui certi **elementi** verranno trattati solo nella prossima sezione ma **non** verranno ripresi, in seguito, parlando degli **elementi** della **scena 3D** e mostrando i relativi **esempi**.

ELEMENTI DELLO SCENE GRAPH – IL LOCALE

Un **oggetto Locale** definisce una **posizione fissa** all'interno del **Virtual Universe**; in particolare, definisce una **origine** per quell'universo (proprio come (0.0, 0.0) identifica l'origine degli assi nel piano cartesiano).

Il simbolo da utilizzare per rappresentarlo nello **Scene Graph** è un **rombo**.



Un **universo** in genere ha un solo **Locale**, che ha **coordinate di default (0.0, 0.0, 0.0)**, ma per alcuni progetti potrebbe essere necessario creare due o **più oggetti Locale** (scelta sconsigliata).

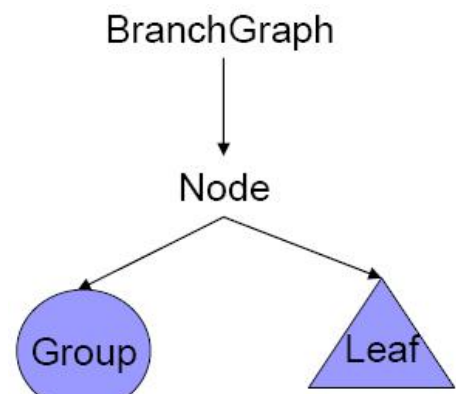
NOTA: quando un **oggetto** esegue **spostamenti molto 'grandi'**, può diventare necessario **sganciarlo dal Locale di partenza** ed imparentarlo ad un **nuovo Locale**.

Situazioni del genere si verificano comunque raramente.

ELEMENTI DELLO SCENE GRAPH – LA SUPERCLASSE NODE

All'interno dei **Branch Graph** si trovano degli **oggetti** che ereditano dalla **superclasse Node** e che possono essere **Gruppi (Group)** o **Foglie (Leaf)**.

Sono quindi le **sottoclassi di un Node** a costituire la gran parte dello **Scene Graph**.



ELEMENTI DELLO SCENE GRAPH – I GRUPPI

Group è la **superclasse** usata per *raggruppare logicamente* gli **oggetti** e per specificare le *trasformazioni operate* sugli stessi.

Viene indicata, nello **Scene Graph**, con un cerchio, dove all'interno vi è una **sigla** che identifica il **tipo di gruppo** che stiamo utilizzando; quelli di base, nativi di **Java 3D**, sono **Branch Group** e **Transform Group**.



ELEMENTI DELLO SCENE GRAPH – LA SUPERCLASSE LEAF

Leaf è la **superclasse** utilizzata per specificare le **forme (Shape3D)**, gli **oggetti**, i **suoni**, i **comportamenti (Behavior)**, i **bound (zone di influenza)** ed altri **elementi dello Scene Graph** che non hanno figli, ma possono agganciarsi mediante particolari **collegamenti**, detti **Reference**, ad **oggetti** di tipo **Node-Component**.



Vediamo quindi cosa sono gli **oggetti NodeComponent** e le **relazioni Reference**...

ELEMENTI DELLO SCENE GRAPH – LA SUPERCLASSE

NODECOMPONENT

NodeComponent è la **superclasse** utilizzata per specificare **geometrie**, **aspetto**, **texture** e **materiali**.

Viene rappresentata, nello **Scene Graph**, mediante un ovale.



Un **oggetto di tipo NodeComponent** può essere **linkato** da più di un **Node**, mediante **collegamenti Reference** (ad esempio, due **oggetti 'sedia'**, **distinti**, possono avere lo stesso aspetto, ossia una **Appearance** con **texture 'legno'**).

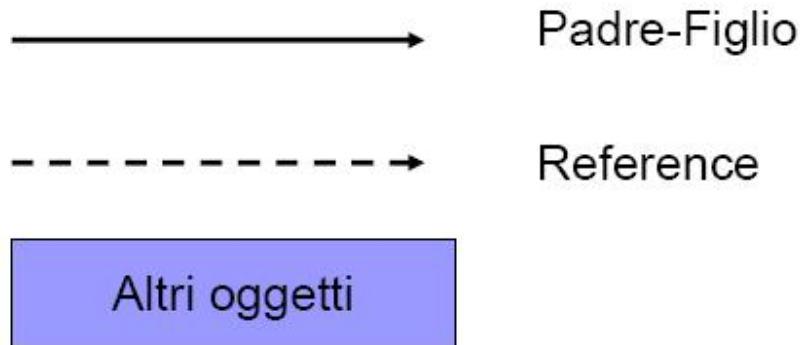
RELAZIONI PADRE-FIGLIO E RELAZIONI REFERENCE

Le **relazioni** di tipo **Reference**, che permettono ad un oggetto di essere utilizzato da più oggetti dello **Scene Graph** contemporaneamente, verranno indicate mediante **linee tratteggiate**.

Le **relazioni** di tipo **Padre-Figlio**, che **collegano** un gruppo o una foglia **ad UN SOLO genitore** (un gruppo), verranno indicate mediante **semplici linee**, senza tratteggio.

In entrambi i casi, si tratterà di **collegamenti 'orientati'**, per cui bisognerà mostrare una **freccia**.

Qualora vi fosse la necessità di inserire **altri oggetti** nello **Scene Graph**, li si potrà rappresentare mediante un **rettangolo**.



IL SIMPLE UNIVERSE

Ricapitolando: quando vogliamo creare un **universo 3D ex novo** dobbiamo **istanziare un Virtual Universe**, uno o più oggetti **Locale**, un **View Branch Graph** (con tutti gli oggetti, le proprietà ed i metodi collegati), un **Content Branch Graph** e tutti gli elementi che compongono la **scena 3D**...

...troppo complicato ??

In effetti sì, ed è per questo che **Java 3D** mette a disposizione una **utility**, una classe chiamata **SimpleUniverse**, che può fare la maggior parte del lavoro per noi, occupandosi di **Virtual Universe**, **Locale** e **View Branch Graph**, lasciandoci liberi di operare sul **Content Branch Graph**.

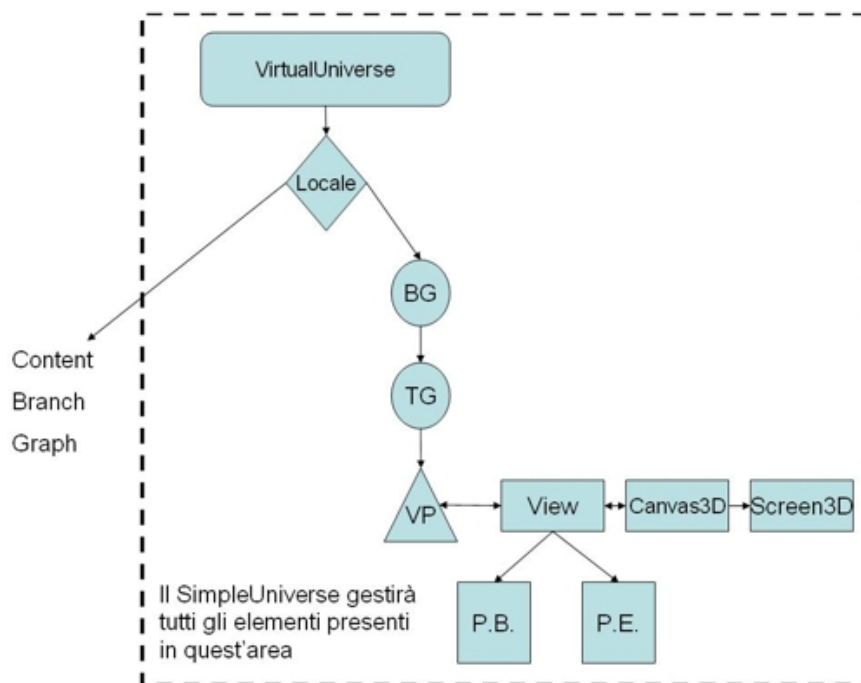
Il **risparmio**, in termini di tempo e di **codice**, è evidente se si confrontano i passi necessari per creare una **scena 3D** da zero e con l'ausilio del **SimpleUniverse**.

Da zero, infatti, dovremmo:

1. creare una **Canvas3D** ed un **Virtual Universe**, e **collegarli** tra di loro;
2. creare un **Locale** e **collegarlo** al **Virtual Universe**;
3. costruire un **View Branch Graph**, collegandogli gli **oggetti View**, **View-Platform**, **PhysicalBody**, **PhysicalEnvironment** ed impostandone i **parametri**;
4. costruire uno o più **Content Branch Graph**;
5. **compilare** tutti i **Branch Graph**;
6. **collegare** i **Content Branch Graph** al **Locale**;

mentre, con l'aiuto di **SimpleUniverse**, sarà sufficiente:

1. creare una **Canvas3D** ed un **SimpleUniverse**, e **collegarli** tra di loro;
2. (OPZIONALE) personalizzare il **SimpleUniverse**;
3. costruire uno o più **Content Branch Graph**;
4. **compilare** tutti i **Branch Graph**;
5. **collegare** i **Content Branch Graph** al **Locale** del **SimpleUniverse** (già presente, in quanto creato implicitamente dal **SimpleUniverse**).
- 6.



Tranne qualche eccezione, gli **esempi** che verranno presentati nel corso di questa guida faranno uso di **SimpleUniverse**.

COMPILARE UN BRANCH GRAPH

Parlando del **Simple Universe** e dei passi da seguire per creare una **scena in Java 3D** abbiamo usato l'espressione *'compilare i Branch Graph'*: cosa significa ?

Java 3D può utilizzare, internamente, una **rappresentazione più efficiente** della **scena** per migliorare le performances durante l'**esecuzione** dell'**applicazione**.

Per ottenere un tale miglioramento, è necessario *compilare un Branch Group* (o un intero **branch graph**), invocandone il metodo *compile*.

LE CAPABILITY

Quando si **compila** un **Branch Group** (o quando un oggetto è *live*, vivo, come vedremo a breve) si ha un effetto collaterale non sempre piacevole: i **valori delle trasformazioni** e di alcuni oggetti dello **Scene Graph** vengono **'bloccati'**.

Quando dobbiamo creare, per esempio, **un'animazione**, **modificando i valori di un TransformGroup**, dobbiamo **'sbloccare'** alcuni **campi** dello stesso, o meglio: dobbiamo *permettere delle operazioni su tali campi, impostare delle capability*.

Le varie **componenti** di un **universo virtuale di Java 3D** hanno le relative **capability** e, se **estendono** delle classi, ereditano le **capability** degli **'antenati'**.

Ad **esempio**, per permettere l'**impostazione dinamica di una trasformazione 3D** in un oggetto di tipo **TransformGroup**, scriveremo:

```
TransformGroup tg1 = new TransformGroup();  
tg1.setCapability(Transform.ALLOW_TRANSFORM_WRITE);
```

ALLOW_TRANSFORM_WRITE è, quindi, una **costante** (un intero, a livello implementativo) della classe **Transform**.

L'istruzione mostrata precedentemente permetterà l'**esecuzione dell'istruzione**:

```
tgl.setTransform3D(t3d1);  
ma non dell'istruzione  
tgl.getTransform3D(t3d1);
```

per la quale bisognerà **impostare una capability di READ**.

In generale, se vi è una **capability WRITE** ve ne sarà una **READ**, e viceversa.

OGGETTI “VIVI”

Un **oggetto** dello **Scene Graph** appartenente ad un **universo attivo** è detto *live* (**vivo**).

Effetto collaterale dell'essere un oggetto **vivo** è che le **capability** sono **bloccate**.

Per permettere determinate **operazioni a runtime** sarà quindi necessario **impostare** esplicitamente le **capability** delle operazioni stesse prima di chiamarle all'interno del programma.

IL SISTEMA DI RIFERIMENTO DI JAVA 3D

Il **sistema di riferimento** dell'**universo di Java 3D** è così 'orientato', quando viene visualizzato a schermo:

- **Asse X**: orizzontale, con valori positivi a destra;
- **Asse Y**: verticale, con valori positivi in alto;
- **Asse Z**: uscente dallo schermo, con valori positivi verso l'utente.

L'**unità di misura** è, di default, il **metro** (ma vedremo come passare ad un'altra unità di misura agendo sull'oggetto **Locale**, nella prossima sezione).

Le **coordinate** vengono espresse con **tre valori (x, y, z) FLOAT**.

UN PRIMO ESEMPIO: UN UNIVERSO VUOTO

A questo punto è giunto il momento di mostrare un pò di **codice**.

Il primo **esempio**, *'PrimoEsempio.java'*, il cui **codice sorgente** è reperibile in **Appendice**, mostrerà... l'**universo virtuale**, dunque non vedremo niente!

In effetti, lo scopo di **'PrimoEsempio'** è quello di introdurre le righe di **codice** di base e mostrare come **visualizzare la Canvas3D** in un **frame** di una classica **applicazione Java**.

L'**universo virtuale** così creato non conterrà nulla, per cui vedremo semplicemente lo **sfondo nero**.

Per prima cosa, ecco gli **import** di base:

```
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
```

Questi sono gli **import fondamentali**, necessari per creare un **universo Java 3D**, definire i **vettori** per le **trasformazioni** e le **geometrie** più semplici.

Qual è lo **Scene Graph** dell'**universo virtuale** che vogliamo definire ?

Esso è costituito semplicemente da un **SimpleUniverse**.

Ecco le righe di **codice** che, poste all'interno della **classe che eredita da JFrame**, creeranno un **universo virtuale** e ne permetteranno la **visualizzazione a video**:

```
// Recupera le configurazioni grafiche del sistema (computer)
GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();

// Crea la Canvas3D, passandole le configurazioni grafiche del sistema
mediante il costruttore.
Canvas3D c3D = new Canvas3D(config);

// Crea il Simple Universe, passandogli la Canvas3D appena creata
mediante il costruttore.
SimpleUniverse sU = new SimpleUniverse(c3D);

// La Canvas va quindi aggiunta al Frame dell'applicazione (la Canvas3D
eredita infatti da AWT Component, proprio come un JPanel); ovviamente,
```

```
this 'vale' solo se tale riga di codice è scritta all'interno di un  
JFrame !  
this.getContentPane().add(c3D, BorderLayout.CENTER);
```

Non è stato creato alcun **BranchGroup** da aggiungere al **SimpleUniverse** (come **Content Branch Graph**): il nostro **universo virtuale** è vuoto.

Il **codice sorgente completo** si trova, come indicato precedentemente, in **Appendice**.

SECONDO ESEMPIO: POPOLARE L'UNIVERSO

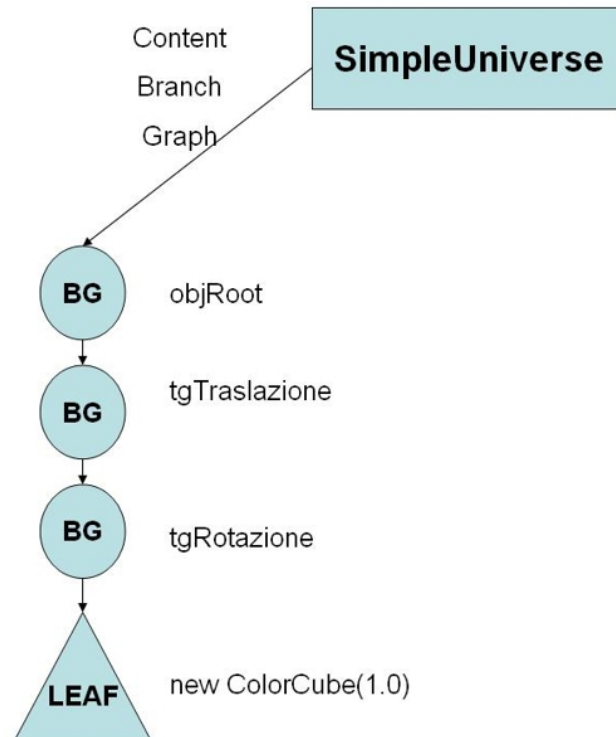
Il **secondo esempio**, definito nel file **'SecondoEsempio.java'**, ricopiato in **Appendice**, è molto simile a **'PrimoEsempio'**, ma inserisce un **Content Branch Graph**, definito tramite il **metodo 'creaLoSceneGraph'**, al **Simple Universe**.

Subito dopo aver creato il **SimpleUniverse** bisognerà, quindi, inserire le seguenti righe di **codice**:

```
// Creiamo una scena di base con un metodo da noi definito (vd. dopo) e la  
compiliamo  
BranchGroup scena = creaLoSceneGraph();  
scene.compile();  
  
// Aggiungiamo la scena al Simple Universe  
sU.addBranchGraph(scena);
```

La **scena** definita in **'creaLoSceneGraph'** è costituita da un oggetto nativo di **Java 3D**, il **ColorCube** (un cubo colorato, come suggerisce il nome), **ruotato** intorno a due assi e **traslato**, per renderlo visibile all'utente (all'avvio, l'utente si trova posizionato nell'**origine**, così come ogni **oggetto 3D** agganciato direttamente alla **radice del Content Branch Graph**, senza traslazioni; senza **traslazione**, l'utente si troverebbe **dentro il ColorCube** e non vedrebbe niente !).

Lo **Scene Graph** di questo esempio è mostrato nella **figura** seguente.



Le righe di **codice** presenti all'interno di '**creaLoSceneGraph**' al momento non vi diranno nulla, ma presto (sezioni 4 e 5) il loro significato vi sarà chiarissimo !

Prima di concludere questa prima sezione, soffermiamoci su una particolare riga di **codice** presente in '**SecondoEsempio.java**', ossia:

```
JPopupMenu.setDefaultLightWeightPopupEnabled(false); .
```

Cosa significa ? A che serve ?

Una **Canvas3D** è un oggetto AWT (**Heavyweight**), che potrebbe **nascondere oggetti Swing** anche quando questi dovrebbero essere visualizzati *sopra* la **Canvas3D**, come, ad esempio, un **Menù**.

L'**istruzione** precedentemente mostrata rappresenta un modo per risolvere questo problema; in alternativa, le **componenti Lightweight** e le **componenti Heavyweight** potrebbero essere inserite in **contenitori separati**.

```
*      *      *
*      *      *
```

Capitolo 3

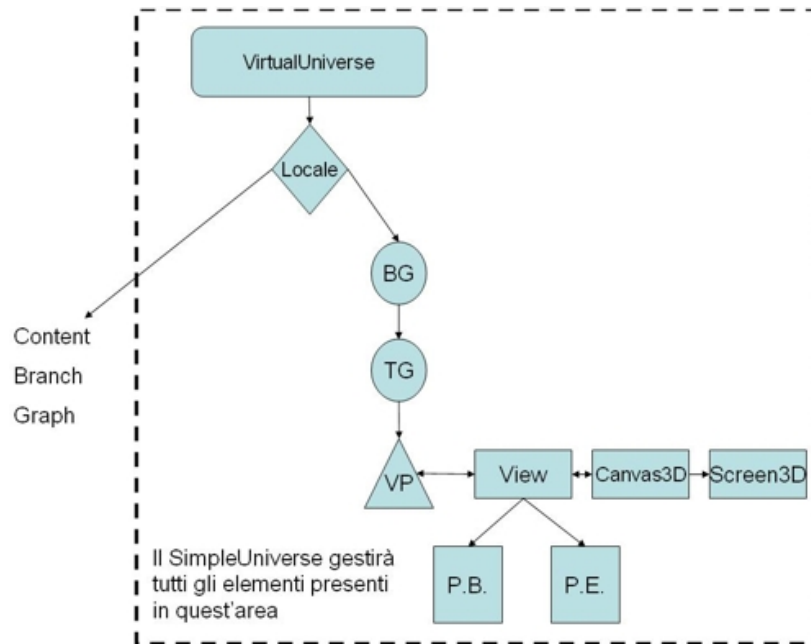
Il Simple Universe in dettaglio

In questa **sezione** tratteremo un certo numero di argomenti di carattere generale, necessari per conoscere meglio la **struttura dell'universo virtuale** creato da **Java 3D** prima di inserirvi **oggetti** e **trasformazioni**:

- **operazioni** sull'oggetto **Locale**;
- **operazioni** sul **View Branch Graph**, passando dal **Simple Universe** e parlando delle **Normali**;
- **clipping**;
- **bound** e **scope**;
- **background**.

PERSONALIZZARE IL SIMPLE UNIVERSE

L'aver creato un **SimpleUniverse** che fa per noi gran parte del lavoro non significa non poter **accedere** ai campi **Locale** e del **View Branch Graph** per modificarli, **personalizzandoli**...



Vedremo quindi come **recuperare** alcuni di questi elementi dal **SimpleUniverse** ed analizzeremo alcune delle possibili **operazioni** che li riguardano.

IL LOCALE

L'oggetto **Locale** specifica una **posizione** nell'**universo 3D** mediante **coordinate 'high resolution'**.

Le **coordinate del Locale**, che di default sono **(0.0, 0.0, 0.0)**, costituiranno l'**origine del sistema di riferimento** per tutti i **sottografi** che discendono da quel **Locale**; le coordinate degli oggetti presenti nell'**universo virtuale** vanno quindi poste **in relazione alle coordinate del Locale**.

Le **High Res Coords**, le coordinate high resolution, permettono di definire con estrema precisione un punto nell'**universo virtuale**.

Consistono di **tre numeri a 256 bit**: uno per ogni **coordinata x, y e z**; il valore 1.0 equivale ad un **metro**. A livello implementativo, ciascuna coordinata consiste di **8 valori**.

E' possibile specificare distanze, posizioni e spostamenti facendo uso di tutti i valori di una **coordinata HiResCoord**, passando quindi dalla **precisione subatomica** a 'passi' di diversi **anni luce**.

E' possibile impostare o recuperare le **coordinate high resolution** del **Locale** tramite i seguenti **metodi**:

- `getHiRes(HiResCoord hiRes) : void`
- `setHiRes(HiResCoord hiRes) : void`
- `setHiRes(int[] x, int[] y, int[] z) : void`

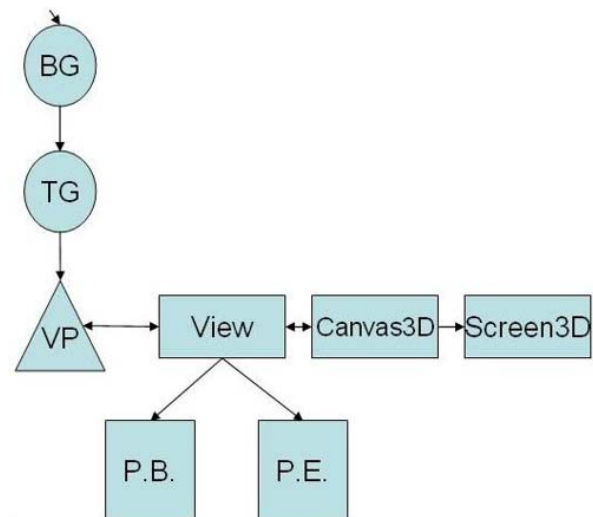
Infine, **Locale** fornisce dei **metodi** per aggiungere, rimuovere o enumerare i suoi **branch graph** 'figli':

- `finalize() : protected void`
- `addBranchGraph(BranchGroup) : void`
- `getAllBranchGraphs() : java.util.Enumeration`
- `getVirtualUniverse() : VirtualUniverse`
- `numBranchGraphs() : int`

ELEMENTI DEL VIEW BRANCH GRAPH. OPERAZIONI.

Vedremo ora come accedere agli **elementi del View Branch Graph**, implementati automaticamente dal **Simple Universe**.

A parte i **metodi** necessari per recuperare la **Canvas3D** (o le **Canvas3D**) ed il **Locale**, **SimpleUniverse** mette infatti a disposizione **tre metodi** che hanno a che fare con il **View Branch Graph**:



- `getPreferredConfiguration() : static java.awt.GraphicsConfiguration`
- `getViewer() : Viewer`
- `getViewingPlatform() : ViewingPlatform`

Il metodo *getPreferredConfiguration()* è stato analizzato nella sezione precedente: restituisce le impostazioni grafiche del sistema in uso.

Conoscere le impostazioni grafiche del sistema serve a **Java 3D** al momento di visualizzare la **Canvas3D** sul dispositivo.

Il metodo *getViewer()* restituisce l'oggetto **Viewer** (è uno solo) associato allo **Scene Graph**.

Tale oggetto contiene le informazioni circa gli aspetti 'fisici' (la visualizzazione della **Canvas3D**, l'hardware utilizzato) e 'virtuali' (ad esempio, tutte le informazioni e le policy necessarie per effettuare il rendering della scena 3D a partire da un punto di vista situato all'interno della stessa) dell'osservatore nell'universo 3D.

Il metodo *getViewingPlatform()* di **SimpleUniverse** restituisce l'oggetto **ViewingPlatform**, che si occupa della visualizzazione della scena virtuale sulla **Canvas3D** e tramite il quale è possibile accedere al **Transform Group** del **View Branch Graph**.

Tra i metodi più interessanti messi a disposizione da questo oggetto abbiamo:

- *getViewPlatformBehavior()* : **ViewPlatformBehavior**
- *getViewPlatformTransform()* : **TransformGroup** (*il Transform Group 'più vicino' al nodo ViewPlatform*)
- *setViewPlatformBehavior(ViewPlatformBehavior behavior)* : **void**
- *setNominalViewingTransform()* : **void**

L'ultimo metodo, *setNominalViewingPlatform()*, sposta leggermente indietro il sistema di riferimento.

Può essere d'aiuto nella visualizzazione di (piccoli) oggetti, se non si vuole applicare un 'pesante' **Transform Group** di traslazione agli stessi per allontanarli dall'osservatore.

Nel file d'esempio '*NominalViewingPlatform.java*', riportato in **Appendice**, si ha il cubo colorato visto in '**SecondoEsempio**', nel capitolo precedente, con la differenza che, in questo caso, non c'è un **Transform Group** di traslazione applicato al cubo, mentre è presente il comando:

```
sU.getViewingPlatform().setNominalViewingPlatform();
```

che sposta leggermente indietro l'osservatore; l'oggetto è piccolo e l'entità dello spostamento è sufficiente a permetterci di guardarlo da fuori.

LE NORMALI

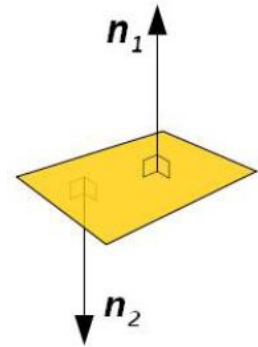
Quando ci si trova dentro un **oggetto**, anche illuminandolo con una fonte di luce posta all'interno dello stesso, di default non si vede niente.

Per estensione, a volte può capitare di non vedere una **geometria** (anche un semplice triangolo), se non da un solo lato: girandogli intorno, essa potrebbe risultare **invisibile...** perchè ?

Per rispondere a queste domande, bisogna introdurre i concetti di **NORMALE** e di **CULLING**.

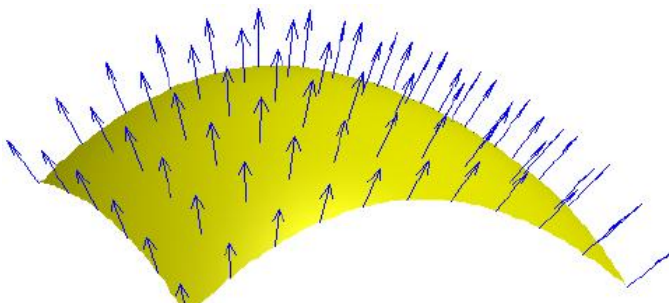
In matematica, una **NORMALE** a una **superficie** (piana) è un **vettore** uscente perpendicolarmente da tale **superficie**.

Con qualche accorgimento, tale definizione può essere estesa anche a **superfici** non piane, ed è possibile definire anche le **normali** ad un **vertice**.



Una **Normale** è quindi un **VETTORE**, dotato di un verso, e la visualizzazione di una **geometria** dipende proprio dalle **Normali**: se stiamo osservando il lato con la **Normale uscente**, tale **superficie** verrà renderizzata; tuttavia, in qualunque superficie è possibile individuare due **normali uscenti**, una per ciascun lato... come mai allora una faccia viene **visualizzata** e l'altra no ?

La risposta è la seguente: in computer grafica si fa uso di **superfici orientate**, cioè **superfici** dove, con una **convenzione** (definita da precise regole matematiche), è possibile chiamare '**positive**' alcune configurazioni di vettori e '**negative**' altre.



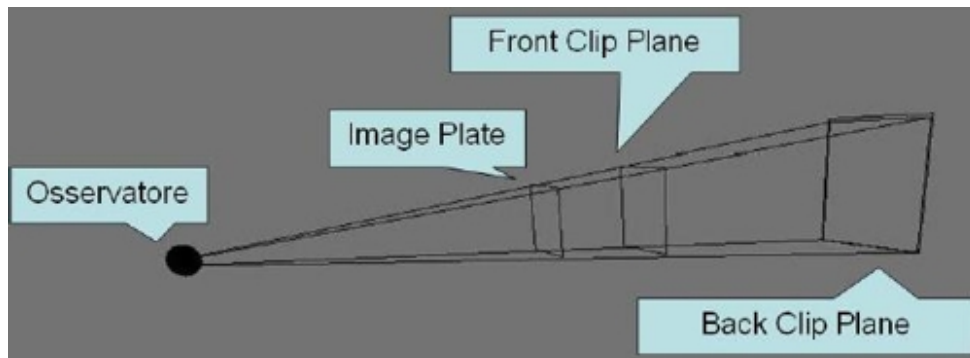
In particolare, una **Normale** sarà **positiva**, mentre l'altra sarà **negativa**.

Di default, **Java 3D** non renderizza le facce con la **Normale negativa**; tuttavia, è possibile impostare la resa di **entrambe le facce** o anche

delle sole facce con **Normali negative**, escludendo le facce con **Normali positive**: queste impostazioni sono dette **impostazioni di culling** e verranno discusse nella sezione dedicata all'**aspetto visivo** degli oggetti.

IL CLIPPING

Java 3D non renderizza, a video, tutti gli oggetti che si trovano tra l'*image plate* (il 'piano' del rendering) e l'infinito, ma solo gli oggetti che si trovano all'interno di un determinato **volume**, detto *Viewing Frustum*, definito dalla distanza tra due piani, il *front clip plane* e il *back clip plane*, e l'ampiezza dell'**angolo di visuale**.

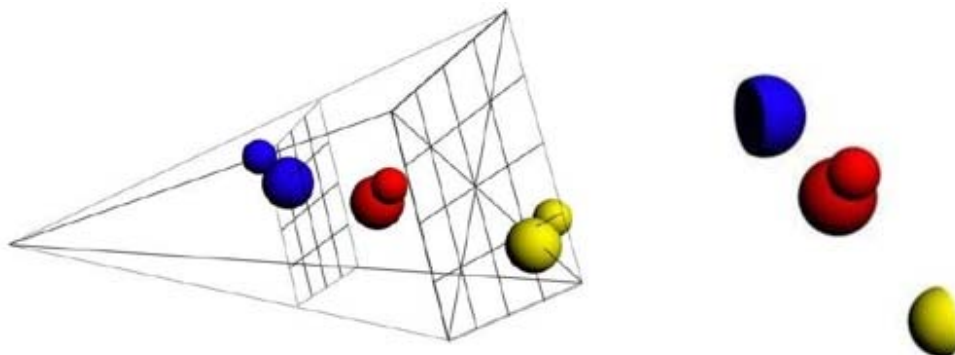


Gli oggetti posizionati fuori dal **Frustum** vengono quindi '**tagliati**' (**clipped**) o non vengono **renderizzati** affatto.

Ciò ovviamente può portare ad **effetti indesiderati**, come visibile in figura...



Quello che succede è mostrato schematicamente nella figura seguente, che illustra appunto il '**funzionamento**' del clipping.



E' l'**oggetto View** che si occupa di gestire i **valori di clipping**.

Per **modificare** tali **valori** è necessario scrivere le seguenti righe di codice (sia '**simpleU**' il nostro **oggetto SimpleUniverse**):

- `simpleU.getViewer().getView().setBackClipDistance([VALORE]);`
- `simpleU.getViewer().getView().setFrontClipDistance([VALORE]);`

dove '[VALORE]' è un **double**.

Un rapporto tra **back clip distance** e **front clip distance** considerato 'ragionevole' è, ad esempio, un valore compreso tra 100 e 1000, o comunque inferiore a 3000, ma a seconda delle scene da visualizzare potrebbe essere possibile utilizzare **rapporti** più grandi anche di qualche ordine di grandezza senza riscontrare particolari rallentamenti nell'esecuzione del programma.

L'ESEMPIO “ESEMPIO SIMPLEUNIVERSE”

Un **esempio** che illustra quanto detto fino a questo punto è '*EsempioSimpleUniverse*', ricopiato in **Appendice**.

Si tratta di una **versione semplificata** di '**NominalViewingPlatform.java**', dove sono state inserite delle operazioni di **modifica** e di **stampa a video** dei valori di alcuni campi di **SimpleUniverse**, **Locale** e **View**.

BOUND E SCOPE: INTRODUZIONE

Un **bound** definisce uno **spazio** all'interno dell'**universo virtuale**, detto '**bounding region**'.

E' **associato** tipicamente ad un **nodo** (luci, behavior, suoni, ...) per indicare in quale **regione di spazio** è possibile risentire degli **effetti del nodo**.

Esempi intuitivi sono la forza di gravità ed il cono di luce di un faretto 'spot'.

Lo **Scope** è un **insieme di oggetti** di tipo **Group** sui quali un **nodo** fa sentire la sua **influenza**.

Il **bound** è, quindi, una **zona di influenza 'spaziale'**, definita da coordinate, mentre lo **Scope** è un **raggruppamento 'logico'** di oggetti che risentono degli **effetti di un nodo**.

Al di là dei capitoli dedicati a questi due elementi in questa sezione, avremo modo di esaminarli ed utilizzarli praticamente in ogni capitolo della guida.

BOUND E BOUNDINGLEAF

Tre sono le **classi** che **ereditano** direttamente da **Bounds** e che permettono di **definire**, per l'appunto, delle **bounding regions**:

- BoundingSphere
- BoundingBox
- BoundingPolytope

E' preferibile, per motivi legati alle performance, scegliere **bound di piccole dimensioni**.

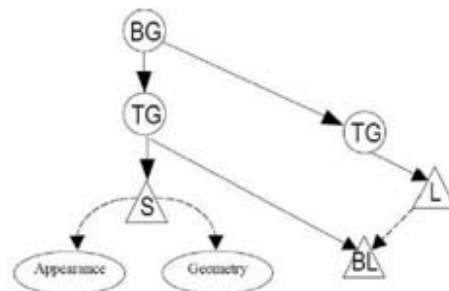
Nei **BoundingBox**, i lati sono paralleli agli assi del sistema di riferimento di **Java3D**, cosa che rende il calcolo delle **intersezioni tra bound** più semplice.

Oltre a **Bounds**, è possibile definire una **regione di influenza** tramite un particolare oggetto: la **BoundingLeaf**.

A differenza di un semplice **Bound**, una **BoundingLeaf** è una vera e propria **foglia dello Scene Graph** (dunque può essere aggiunta ad un **TransformGroup** e spostata nell'universo virtuale di **Java 3D**) che ha, associata, una **regione di influenza**.

La **bounding region** di una **BoundingLeaf** può muoversi indipendentemente dagli **oggetti che la utilizzano** (mediante **collegamenti Reference**) oppure insieme ad uno, e uno solo, di essi (con un **collegamento di tipo padre-figlio**).

Lo scene graph a lato descrive una situazione nella quale una BoundingLeaf è referenziata da un oggetto (Leaf), ma si muove insieme ad un altro gruppo di oggetti (quelli che, come S, sono figli del transform group TG).



Sia ad **esempio** 'light1' una **sorgente luminosa**; potremo quindi scrivere, a seconda dei casi:

- light1.setInfluencingBounds(Bounds b);
- light1.setInfluencingBoundingLeaf(BoundingLeaf bl); .

SCOPE, CON UN PICCOLO INTRUSO

Come detto precedentemente, lo **Scope** indica l'influenza di un **nodo** sugli altri ma, a differenza dei **bound**, tale influenza è determinata da un **raggruppamento logico** di elementi **Group**, non da una **'posizione fisica'**, presa in considerazione dai **bounds**.

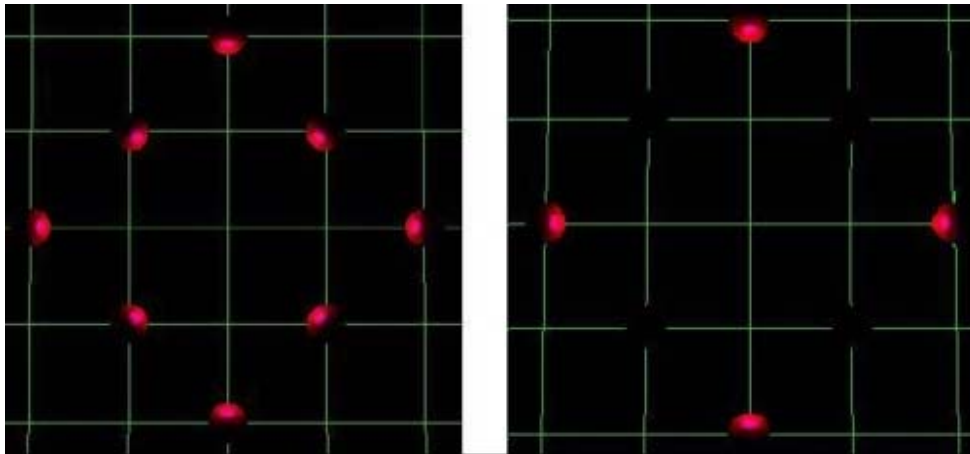
Scope e **Bound** non sono, comunque, mutuamente esclusivi.

Sia ad **esempio** 'light1' una **fonte luminosa**; potremo scrivere allora:

- `light1.addScope(Group g);`

per impostare, come **gruppo di elementi** che verranno illuminati da 'light1' (purchè si trovino, comunque, nella sua **bounding region di influenza**), 'g'.

L'immagine seguente, tratta dal capitolo dedicato alle luci, serve a chiarire con un **esempio visivo** come certi oggetti, pur trovandosi nella **bounding region** di influenza di una **fonte luminosa**, possano non subirne gli **effetti**, a differenza di altri.



L'**esclusione** di alcune sfere avviene **'logicamente'**, a livello di **Scene Graph**.

Codice sorgente e **Scene Graph** di questo piccolo intruso verranno mostrati nel capitolo dedicato alle **luci**.

IL BACKGROUND

L'universo di **Java 3D** è, di default, **nero**.

E' possibile comunque specificare l'**aspetto** da dare allo **sfondo (background)**, utilizzando a scelta un **colore, un'immagine o una geometria**.

Bisogna porre attenzione però a come disporre tali elementi, in quanto la '**pila di visibilità**' adottata da **Java 3D** è ben precisa e vede, sul fondo, il colore, quindi l'immagine e, all'apice, la geometria.

La **geometria** deve essere specificata come un insieme di punti (**PointArray**, **TriangleArray**, ed altri **oggetti** che verranno trattati nella sezione dedicata alla **classe Geometry** e alle sue figlie).

Un **Background** ha, associato, un **bound**: possono esserci quindi **più Background per zone diverse** del mondo virtuale, ma verrà visualizzato solo quello in cui si trova l'**osservatore**.

I passi necessari per creare un **Background** sono:

1. creare un **Branch Group** contenente una **sfera di raggio unitario**, dotata di un suo **Appearance** e con le **Normali** rivolte verso l'interno;
2. creare un **oggetto Background**, impostandone un **bound d'azione** e, come '**geometria**' (con il metodo **setGeometry([arg])**), il **Branch Group** creato al punto precedente;
3. creare un **Branch Group** contenente il **Background** così definito ed agganciarlo alla **scena**.

In Appendice è presente il codice sorgente dell'esempio '**EsempioBackground.java**' che mostra, appunto, come implementare un semplice **background** in un programma **Java 3D** e come tale **sfondo** verrà **visualizzato** durante l'esecuzione (NOTA: è necessario posizionare, nella stessa cartella del file .class dell'esempio, un file immagine di nome "immagineBackground.jpg").

ANCORA SU BOUND E BACKGROUND

Dovrebbe essere chiaro, a questo punto, che un **bound** non crea un **oggetto visuale** di una data forma (sfera, box, polytope), ma identifica semplicemente **una regione, un volume di spazio 3D**.

Il **bound**, associato ad un altro oggetto, indica la **zona di influenza**, il campo di validità e di azione di quell'oggetto.

Abbiamo visto inoltre l'**oggetto Background**, un oggetto che, per definire uno **sfondo**, fa uso di una **sfera di raggio unitario**, sulle cui pareti interne (impostando, tra l'altro, le **Normali** rivolte verso l'interno) proietta una **texture**, e di un **bound d'azione**.

Bisogna però porre l'attenzione su di un concetto: l'utilizzare un **bound** con una data dimensione non si traduce in un 'ingrandimento' della **sfera di raggio unitario**.

La **sfera di raggio unitario** è un **modello** utilizzato per definire il **Background**, che verrà poi proiettato come **sfondo** da **Java 3D**.

Lo **sfondo**, in **Java 3D**, si trova sempre a distanza infinita, ed anche avvicinandoci ('camminando' nell'universo virtuale) ad esso, la sua immagine non muterà.

Uscendo dal **bound d'influenza**, esso sparirà immediatamente.

Creando invece uno **sfondo** a partire da una **sfera di notevoli dimensioni** avremo un effetto radicalmente diverso: 'camminando' verso le pareti interne della **sfera**, la texture ad essa applicata diventerà sempre più grande, dunque l'immagine visibile cambierà; non solo: una volta usciti dalla **sfera**, girandoci per osservarla da fuori, vedremo le **pareti interne della sfera, texturizzate !**

Un **Background** vero e proprio è, invece, **invisibile dall'esterno**.

Gli **esempi 'Background Creato Con Sfera Enorme' e 'Background Classico'**, ricopiati in **Appendice**, permettono di valutare praticamente quanto appena descritto (NOTA: è necessario posizionare, nella stessa cartella dei file .class degli esempi, un file immagine di nome "immagineBackground.jpg").

Entrambi prevedono la **navigazione nell'universo 3D** mediante **tastiera**, con questi **comandi**:

ROTAZIONI

FrecciaDestra: ruota a destra

FrecciaSinistra: ruota a sinistra

PagSu: ruota verso il basso

PagGi`u: ruota verso l'alto

TRASLAZIONI

ALT e FrecciaSu: trasla in avanti
ALT e FrecciaGi`u: trasla indietro
ALT e FrecciaDestra: trasla a destra
ALT e FrecciaSinistra: trasla a sinistra
ALT e PagSu: trasla verso l'alto
ALT e PagGi`u: trasla verso il basso

Provare, ad esempio, ad **andare verso lo sfondo** ('camminando in avanti') fino ad uscire, a seconda dei casi, dal raggio d'azione del **bound** o dalla sfera di raggio 200: è possibile vedere, già a questo punto, che gli **effetti del movimento** sono radicalmente diversi.

Una volta usciti nello spazio esterno, girarsi (rotazione di 180 gradi, a destra o a sinistra) e vedere cosa c'è **dietro**: il **bound** e il **Background** sono **invisibili**, non ci sono oggetti di forma sferica o altro, mentre, nell'esempio con la **sfera**, possiamo vedere la **texture** applicata sulle **facce interne dell'oggetto** !

Molto probabilmente, poi, noterete una considerevole **differenza nei tempi di esecuzione dei due programmi**: una sfera texturizzata di raggio considerevole 'pesa' molto, in termini di performances.

In definitiva:

- il **bound** non identifica un oggetto, ma **un volume nello spazio 3D**. Applicato ad un altro oggetto, identifica la **regione di spazio 3D** ove è possibile risentire degli **effetti** prodotti da tale oggetto;
- un **Background** non posiziona uno **sfondo** ad una distanza definita dal raggio del bound ad esso associato, ma **visualizza (proietta) uno sfondo**, posto ad una **distanza fissa**, se ci troviamo **all'interno del suo bound d'azione**. Se usciamo dal **bound d'azione**, lo **sfondo** sparisce immediatamente.

* * *
* * *

Capitolo 4

Le classi “matematiche” in Java 3D

Per poter effettuare **trasformazioni** (**traslazioni**, **rotazioni**, **ridimensionamenti**) sugli oggetti, ma anche per poter **definire vertici** ed **applicare una texture**, è necessario avere ben chiari in mente alcuni **strumenti matematici** e come tali **strumenti** possano essere utilizzati in **Java**.

In questo breve capitolo vedremo quindi, anche se molto superficialmente, alcuni cenni di '**matematica 3D**' ed esamineremo le **classi matematiche** fornite da **Java** e da **Java 3D**.

MATEMATICA 3D: CENNI

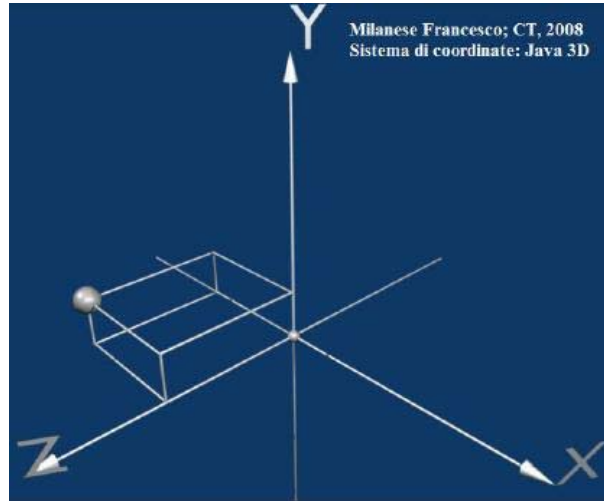
PUNTI, VERTICI

Partiamo dall'elemento fondamentale: il **punto**.

In un **universo virtuale 3D**, un **punto** viene indicato con **tre valori**, uno per ciascun asse:

$$P = (X_P, Y_P, Z_P).$$

Ricordiamoci che, in **Java 3D**, il **sistema di riferimento** vede l'asse **Y** come 'verticale' e l'asse **Z** positivo quando uscente dallo schermo.

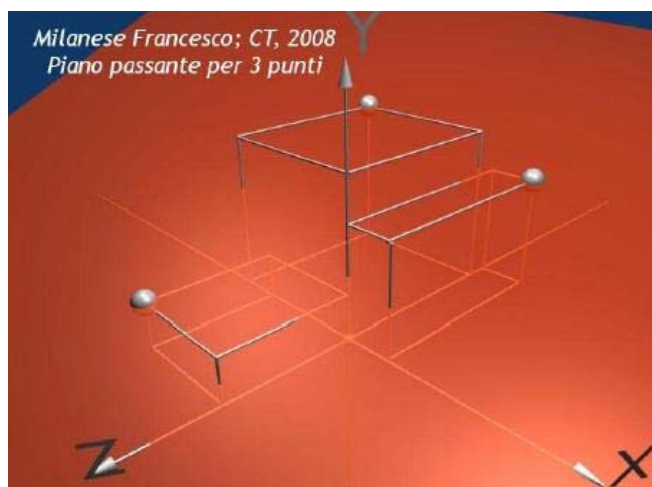


PIANI

Dal **punto** si passa al **piano**: per un punto nello **spazio 3D** passano infiniti piani, mentre per tre punti **non allineati** passa un solo piano.

Per trovare la **formula cartesiana di tale piano**, risolvere (annullarne il determinante) la seguente **matrice**:

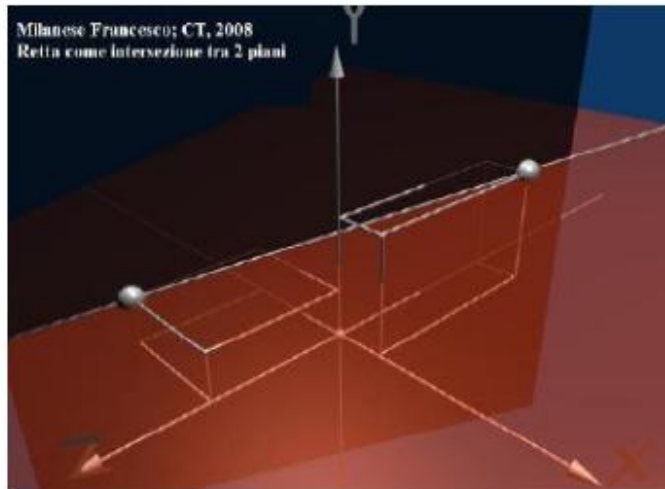
$$\begin{vmatrix} x - x_1 & y - y_1 & z - z_1 \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \end{vmatrix} = 0$$



RETTE

Nello **spazio tridimensionale**, una **retta** è identificata dall'**intersezione di due piani**, dunque da un sistema di **due equazioni**:

$$\begin{cases} ax + by + cz + d = 0; \\ a' + b'y + c'z + d' = 0; \end{cases}$$



VETTORI

In **computer grafica 3D**, la stessa **trippla (x, y, z)** che può identificare un **punto** può identificare un **vettore**: un **vettore tridimensionale**.

Identifichiamo un **vettore** con una **lettera maiuscola**:

$$V = (V_x, V_y, V_z).$$

Un **vettore** è caratterizzato da **direzione e modulo**.

Utilizzeremo i **vettori** per scopi diversi, come effettuare **traslazioni** o definire la **direzione di una sorgente** luminosa o sonora.

Di fondamentale importanza è la **regola della somma vettoriale**:

$$U + V = (U_x + V_x, U_y + V_y, U_z + V_z).$$

Osservare come la **somma di due vettori** possa dare origine ad un terzo vettore diverso sia in direzione che in modulo ai due vettori originali.

Il **modulo di un vettore** è così definito:

$$|V| = \sqrt{V_X^2 + V_Y^2 + V_Z^2}.$$

Moltiplicando un vettore per uno scalare, modifichiamo il **modulo** del vettore senza modificarne la **direzione**:

$$s \cdot V = (s \cdot V_X, s \cdot V_Y, s \cdot V_Z).$$

TRIGONOMETRIA

Infine, un piccolo richiamo delle **formule trigonometriche classiche**, definite nel **2D** ma che, 'composte', possono essere utilizzate per fare calcoli nel **3D**.

- Un cateto è uguale al prodotto dell'ipotenusa per il seno dell'angolo opposto.
- Un cateto è uguale al prodotto dell'ipotenusa per il coseno dell'angolo adiacente.
- Un cateto è uguale al prodotto dell'altro cateto per la tangente dell'angolo opposto.
- Un cateto è uguale al prodotto dell'altro cateto per la cotangente dell'angolo adiacente.

LE CLASSI MATEMATICHE IN JAVA E IN JAVA 3D

MATH

La **classe Math** di **Java** è definita in '**java.lang.Math**'.

Si tratta di uno dei **package** di base di **Java**, che dovrebbe essere noto alla maggior parte dei lettori; ecco comunque, di seguito, una lista di alcuni dei **metodi** più interessanti messi a disposizione da **Math**:

- **abs**, che prende in input un valore numerico (long, int, float o double) e ne restituisce il **valore assoluto** (a seconda dei casi, sotto forma di long, int, float o double);
- **sin**, che prende in input un double e restituisce, sotto forma di double, il **seno** del valore passato in input;

- **cos**, che prende in input un double e restituisce, sotto forma di double, il **coseno** del valore passato in input;
- **tan**, che prende in input un double e restituisce, sotto forma di double, la **tangente** del valore passato in input;
- **asin**, che prende in input un double e restituisce, sotto forma di double, l'**arcoseno** del valore passato in input;
- **acos**, che prende in input un double e restituisce, sotto forma di double, l'**arcocoseno** del valore passato in input;
- **atan**, che prende in input un double e restituisce, sotto forma di double, l'**arcotangente** del valore passato in input;
- **ceil**, che implementa la **funzione 'ceiling'**, prendendo in input e restituendo in output un double;
- **floor**, che implementa la **funzione 'floor'**, prendendo in input e restituendo in output un double;
- **pow**, che prende in input due double e che restituisce, in output, un double, rappresentante il valore del **primo argomento elevato al valore definito dal secondo argomento**;
- **round**, che prende in input un double o un float e restituisce, sotto forma di int o di long, il valore che **approssima** meglio l'input;
- **sqrt**, che restituisce, sotto forma di valore double, la **radice quadrata** di un valore double passato come parametro in input;
- **toDegrees**, che prende in input un double rappresentante un angolo espresso in radianti e che restituisce, in output, un double rappresentante il **valore dell'angolo espresso in gradi**;
- **toRadians**, che prende in input un double rappresentante un angolo espresso in gradi e che restituisce, in output, un double rappresentante il **valore dell'angolo espresso in radianti**.

I **campi** sono invece **E**, che identifica (con un double) la **base dei logaritmi naturali**, e **PI**, il valore (espresso anch'esso con un double) che meglio approssima π .

VECMATH

Il package **VECMATH** di **Java 3D** mette a disposizione diverse **classi** per gestire **punti, colori, vettori, matrici e coordinate delle texture**.

I nomi delle **classi** sono espressi, in generale, nella forma:

[oggetto] [quantità di elementi] [formato dati];

ad esempio, abbiamo:

- **Point3d, Point3f**, che servono a definire punti (una posizione nello spazio 3D) con 3 double (d) o float (f);
- **Color3b, Color3f, Color4b, Color4f**, che servono a definire i colori, con 3 o 4 valori (il quarto per il canale alpha) di tipo byte (b) o float (f);
- **Vector3d, Vector3f**, per i vettori;
- **TexCoord3f**, per le texture, come vedremo in seguito.

POINT*

Gli oggetti creati con le classi della famiglia **Point*** servono ad identificare un **punto** nella **scena 3D**, dunque possono servire per individuare, ad esempio, le coordinate di un vertice, di una sorgente luminosa, di una sorgente sonora, ecc...

COLOR*

Gli oggetti creati con le classi della famiglia **Color*** rappresentano, invece, **un colore**, ad esempio per colorare un vertice o la nebbia.

E' possibile creare colori con **valori alpha (trasparenza)**, utilizzando le **classi Color4***.

VECTOR*

Le classi **Vector*** servono ad identificare **vettori**, utili per descrivere, ad esempio, **le normali, una trasformazione di traslazione, la direzione** di un fascio luminoso, la direzione di un'onda sonora, ...

TEXCOORD*

Le **classi TexCoord*** gestiscono la **mappatura di una texture su un insieme di vertici**.

Verranno prese in esame nella sezione dedicata alla resa visiva degli oggetti, per il momento basti sapere che tali **coordinate** possono essere espresse con 2 o 3 valori float, definendo le classi **TexCoord2f** e **TexCoord3f**.

* * *
* * *

Capitolo 5

Gruppi. Oggetti visuali di base e da file

Finalmente, è giunto il momento di iniziare a **popolare il nostro universo virtuale**, aggiungendo **oggetti al Content Branch Graph** !

Java 3D mette a disposizione tre modi per **aggiungere oggetti ad un Branch Group**:

- attraverso le **geometrie di base**;
- caricando un oggetto **da un file esterno**, mediante un **Loader**;
- definendo vertici e facce, **manualmente**.

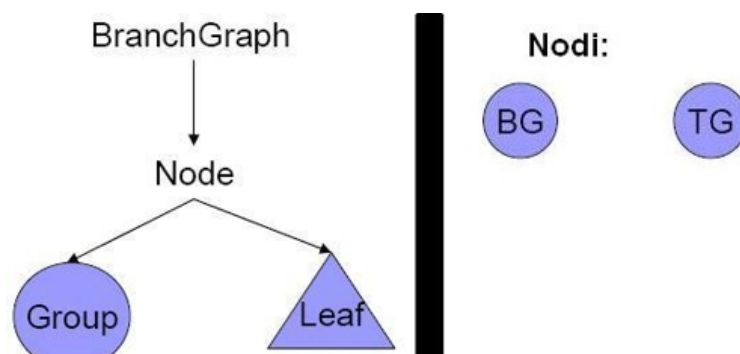
In questo capitolo, dopo aver chiarito alcuni punti sui **gruppi** e sugli **oggetti visuali**, prenderemo in esame le prime due tecniche, mentre dedicheremo il prossimo capitolo alla creazione di **geometrie 'vertice a vertice'**.

I GRUPPI: BRANCH E TRANSFORM

Prima di mostrare come creare degli **oggetti visuali**, soffermiamoci ancora su come **inserirli nello Scene Graph**.

Ormai dovrebbe essere chiaro che gli **oggetti visuali** sono **foglie** da agganciare, mediante il **metodo *addChild***, a **nodi Group**.

I **gruppi** sono di due tipi: **Branch** e **Transform**.



I **nodi Branch** servono solo a collezionare **Leaf** e altri **gruppi** (sia **Branch** che **Transform**), senza operare trasformazioni sui discendenti.

Aggiungendo più **oggetti visuali** direttamente ad un **BranchGroup**, senza **nodi di trasformazione** intermedi, tali oggetti verrebbero posizionati nello stesso posto **nell'universo 3D**.

I **gruppi TransformGroup** definiscono delle **trasformazioni** (rotazioni, traslazioni, scaling), i cui **valori** possono essere modificati a **runtime**, generando **animazioni** (spostamenti, scaling, ...).

Il punto fondamentale è che **le trasformazioni definite in un Transform Group verranno applicate a tutti i discendenti dello stesso**, consentendo quindi di posizionare ed orientare gli **oggetti**.

I **Transform Group** e le tecniche atte a definire trasformazioni statiche verranno discussi nel settimo capitolo; successivamente, in più capitoli, verrà mostrato come **modificare i valori delle trasformazioni a runtime**, in maniera **interattiva** o **'automatica'**.

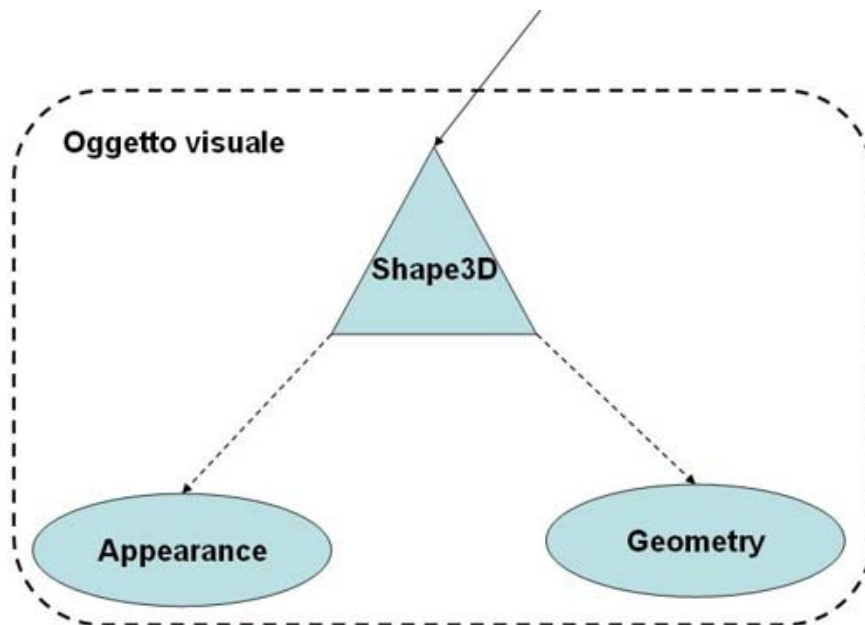
GLI OGGETTI VISUALI

Gli **oggetti** da visualizzare a schermo vengono chiamati ***oggetti visuali***.

Nello **Scene Graph**, un **oggetto visuale** è definito da un oggetto **Shape3D** (sottoclasse di **Leaf**, quindi per forza **foglia** nello **Scene Graph**), che può **referenziare** un **NodeComponent Geometry** e un **NodeComponent Appearance**.

Intuitivamente, la sola **Shape3D** non visualizzerà nulla: la **forma** dell'oggetto andrà definita in **Geometry**, il suo **aspetto visivo** (colori, textures, materiali) in **Appearance**.

Da notare che **Geometry** ed **Appearance** sono collegate ad una **Shape3D** tramite *reference*, **non** collegamenti padre/figlio: una **Geometry** o una **Appearance** possono essere **condivise** da più **oggetti visuali** distinti presenti nello stesso **Branch Group** (ad esempio, è possibile inserire 50 sfere identiche, in posizioni differenti, utilizzando un'unica **Geometry** ed un'unica **Appearance**, con un risparmio notevole in termini di utilizzo delle risorse e velocità di esecuzione dell'applicazione).



GEOMETRIE DI BASE

Per creare **geometrie di base**, native di **Java 3D**, faremo uso del **package** **'com.sun.j3d.utils.geometry'**, che mette a disposizione le seguenti **classi**:

- **ColorCube.**
- **Box.**
- **Cone.**
- **Cylinder.**
- **Sphere.**
- **Text2D** (un rettangolo texture-mapped).
- **Text3D** (simile a Text2D ma con estrusione dei caratteri del testo).

Dal momento che non stiamo definendo le **Appearance**, daremo ai nostri oggetti (tranne **ColorCube**) un **aspetto di base**, per cui verranno renderizzati bianchi, ma risolveremo presto questo problema...

GEOMETRIE DI BASE – COLORCUBE

Abbiamo già visto **ColorCube** al lavoro in qualche **esempio**, nei capitoli precedenti... ora, però, è giunto il momento di presentarlo formalmente.

Il **costruttore** di default, ***ColorCube()***, crea un **cubo colorato** (ogni faccia ha un colore diverso) di lato unitario pronto per il rendering a video, mentre con ***ColorCube(double scale)*** possiamo impostare manualmente **la misura del lato**.

Non ci sono altri **costruttori**, nè c'è bisogno di creare una **Appearance** per questo oggetto, che possiede un suo **aspetto predefinito**.



L'oggetto è ormai ben noto; in **Appendice**, comunque, è possibile reperire il **codice sorgente** dell'esempio **'EsempioColorCube.java'**.

GEOMETRIE DI BASE – BOX

Box definisce un **parallelepipedo rettangolo** centrato nell'**origine**.

Il **costruttore** di base è **Box()**, che crea un cubo di lato 2.

Con il **costruttore**:

```
Box(float xdim, float ydim, float zdim, Appearance ap);
```

è possibile **specificare**, già all'atto della creazione, **le dimensioni delle componenti x, y e z**.

Il quarto **parametro** assegna un'**Appearance** all'oggetto.

Esistono altri due **costruttori** che permettono di specificare **parametri** per applicare le **textures**, argomento che affronteremo tra qualche capitolo, per cui non li prenderemo in esame.

Un **esempio** di utilizzo del **Box** lo si ha in '**EsempioBox.java**', il cui **codice sorgente** è reperibile in **Appendice**.



Nel creare un **Box**, bisogna specificare che si tratta di un oggetto del **package** '**com.sun.j3d.utils.geometry**', poiché vi è un **Box** 'omonimo' in '**javax.swing**'; scriveremo, dunque:

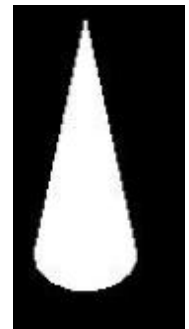
```
addChild(new com.sun.j3d.utils.geometry.box([xdim], [ydim], [zdim],  
new Apperance())); .
```

GEOMETRIE DI BASE – CONE

Cone permette di creare velocemente un **cono**.

I suoi due principali **costruttori**, **Cone()** (crea un cono di raggio 1 e altezza 2) e **Cone (float raggio, float altezza)**, non necessitano di informazioni sull'**Appearance**, cosa che porterà, però, alla creazione di un cono nero, praticamente invisibile se inserito in un **universo virtuale** vuoto.

Esistono altri tre **costruttori di Cone** che permettono di gestirne l'**aspetto** in fase di creazione; ad ogni modo, è sufficiente utilizzare il **metodo**:



```
setAppearance (Appearance app) ;
```

per associare un'**Appearance** all'oggetto.

Il **cono** visibile in figura è stato creato facendo uso del **costruttore** *Cone(float raggio, float altezza, Appearance app)*, passando come **Appearance** *new Appearance()*, che rende il **cono** bianco.

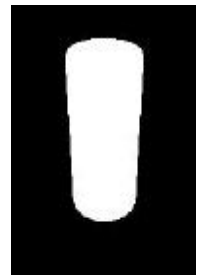
L'esempio '*EsempioCone*', in **Appendice**, mostra un **cono** di raggio 0,4 e altezza 1.

GEOMETRIE DI BASE – CYLINDER

Cylinder crea un **cilindro** la cui geometria è definita dai **parametri** (float) di **altezza e raggio**, proprio come per **Cone** (infatti, i **costruttori di Cylinder** sono identici a quelli di **Cone**, sia come numero che come lista di parametri !).

Con il **costruttore** *Cylinder()* creiamo il **cilindro di default**, di raggio 1 e altezza 2.

Con il **costruttore** *Cylinder(float raggio, float altezza, Appearance app)*, specifichiamo **raggio, altezza e Appearance** da associare all'oggetto.



L'esempio '*EsempioCylinder.java*', in **Appendice**, mostra un **cilindro** di raggio 0,4, altezza 1 e **Appearance** di base, creata passando *new Appearance()* al **costruttore**, che rende il **cilindro** bianco.

GEOMETRIE DI BASE – SPHERE

Sphere crea una sfera con un determinato raggio ed un certo numero di facce:

- di default, cioè utilizzando come **costruttore** *Sphere()*, il raggio vale 1;
- con il costruttore *Sphere(float r)*, il valore del raggio può essere impostato dall'utente in fase di creazione dell'oggetto.

Sphere mette a disposizione **altri 4 costruttori** che permettono di gestirne l'**Appearance**.

L'esempio '*EsempioSphere1.java*' fa uso del **costruttore** *Sphere(float r, Appearance app)*, passando come **app** *new Appearance()*, al fine di mostrare una sfera bianca (le trasformazioni di

rotazione su un oggetto sferico con tale **Appearance** sono del tutto inutili ai fini della visualizzazione...).



Una **sfera** del genere ha comunque una risoluzione bassa, per motivi legati alle performances; per **migliorare la risoluzione**, utilizzare il **costruttore** *Sphere(float radius, int primFlags, int divisions, Appearance app)*, impostando un valore maggiore di 50 per il **parametro divisions** (vedere, a tal proposito, l'esempio '*EsempioSphere2.java*').

GEOMETRIE DI BASE – TEXT2D

Text2D è un oggetto rettangolare con un testo applicato come texture.

Le parti del rettangolo che non fanno parte del testo sono trasparenti; in pratica, il testo è applicato come **mappa dell'opacità** del rettangolo.



Vi è un solo **costruttore**:

```
Text2D(String text, Color3f color, String fontName, int fontSize,  
        int fontStyle);
```

I **campi** che definiscono il **font** non sono 'propri' di **Java 3D**, ma vengono mutuati da **Java 2D**.

Il **testo** ed altri **parametri** possono essere modificati anche in un secondo momento (ad esempio, con *setString(String stringa)*).

In **Appendice** è possibile visualizzare il **codice sorgente** del file di esempio '*EsempioText2D*', che mostra come creare ed inserire un **testo 2D** nella **scena**.

Per gestire **Color3f** si è reso necessario **importare Vecmath**.

Di default, un **testo 2D** è renderizzato solo se visto frontalmente, per via delle **Normali** e delle impostazioni di default del **Culling** in **Java 3D**; ad ogni modo, anche se questi argomenti verranno discussi nella sezione dedicata all'**aspetto** degli **oggetti visuali**, nel **codice sorgente** dell'esempio sono presenti, commentate, le **istruzioni** necessarie per risolvere questo problema: per vederne gli effetti, è sufficiente **rimuovere i caratteri di commento e ricompilare il codice**.

GEOMETRIE DI BASE – TEXT 3D

Un **oggetto Text3D** è simile ad un **oggetto Text2D**, ma ha in più una terza dimensione, data dall'**estrusione** del testo.

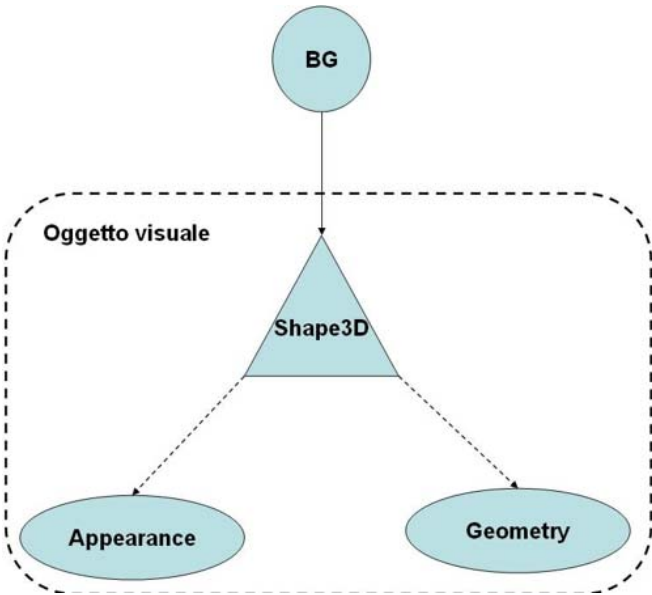
I **costruttori** sono 5 e sono tutti radicalmente diversi dall'unico costruttore di **Text2D**:

- `Text3D()` ;
- `Text3D(Font3D font3D)` ;
- `Text3D(Font3D font3D, String stringa)` ;
- `Text3D(Font3D font3D, String stringa, Point3f position)` ;
- `Text3D(Font3D font3D, String stringa, Point3f position, int alignment, int path)` ; .



Le differenze non si limitano però a quest'aspetto: **Text3D** è un **oggetto visuale** vero e proprio e, per definirlo completamente, è necessario seguire un certo numero di passi durante la sua creazione:

1. creare l'**aspetto (Appearance)** del nostro **testo 3D**;
2. creare la **geometria (Geometry)** del **testo 3D**;
3. creare una **Shape3D**;
4. impostare come **Appearance** della **Shape3D** l'aspetto creato al punto 1;
5. impostare come **Geometry** della **Shape3D** la geometria creata al punto 2;
6. collegare la **Shape3D** ad un **BranchGroup**, da collegare a sua volta alla **scena 3D**.



Da notare che è necessaria anche almeno una **fonte di luce** per visualizzare un **Text3D**.

Text3D fa uso di **Font3D** per gestire il font e di vari parametri (**Point3f**, ma anche flag rappresentati mediante interi) per gestire l'**allineamento**, la scrittura e l'aspetto dell'oggetto.

L'immagine seguente mostra come varia la **rappresentazione del testo** in base ai **flag** di **PATH** e di **ALIGNMENT** utilizzati.

	ALIGN_FIRST (default)	ALIGN_CENTER	ALIGN_LAST
PATH_RIGHT (default)			
PATH_LEFT			
PATH_DOWN			
PATH_UP			

In Appendice è possibile visualizzare il **codice sorgente** del file d'esempio *'EsempioText3D'*, che mostra come implementare le varie fasi della creazione di un **Text3D** e di una **fonte luminosa** per renderlo visibile.

GEOMETRIE DA FILE

Java 3D consente di **importare geometrie** create con vari **programmi di grafica 3D** nell'**universo virtuale** facendo uso di un **Loader**.

Un **Loader** è una **classe** che si occupa di **caricare un oggetto da file** ed inserirlo nello **Scene Graph**, sotto forma di **oggetto 'Scene'**.

L'**oggetto visuale** da inserire nello **Scene Graph** verrà quindi **recuperato dall'oggetto Scene** mediante il **metodo 'getSceneGroup()'**.

Il **Loader** è associato al **formato del file** che si vuole caricare.

Su Internet sono disponibili, a titolo d'esempio, **Loader** per i **formati 3DS, COB, DEM, DXF, LWS, OBJ**.

Java 3D mette comunque a disposizione dei **Loader 'nativi'**, utilizzabili **importando i package**:

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

Il **blocco di codice** tipicamente utilizzato per **caricare un oggetto da file** ed inserirlo nella **scena 3D** è il seguente:

```
BranchGroup gruppo = new BranchGroup();
ObjectFile f = new ObjectFile(ObjectFile.RESIZE);
try {
    Scene s = f.load('[nomefile.formato]');
    gruppo.addChild(s.getSceneGroup());
}
catch(Exception e) { System.out.println(e); }
```

Per poter provare queste righe di codice (partendo da un altro esempio pratico di visualizzazione delle geometrie e modificandolo) è necessario disporre di almeno **un file in un'estensione valida** per i **Loader di Java3D**; sul **sito web** dell'autore, www.redbaron85.com, sono disponibili, ad esempio, diversi **Modelli 3D** in formato **OBJ**, scaricabili liberamente e perfetti per questo scopo.

I file sono stati creati con **Blender**, per cui si consiglia di **rimuovere** (è sufficiente un semplice editor di testo) le prime 3-4 righe, che contengono **istruzioni “non standard”**, proprie di **Blender 3D**.

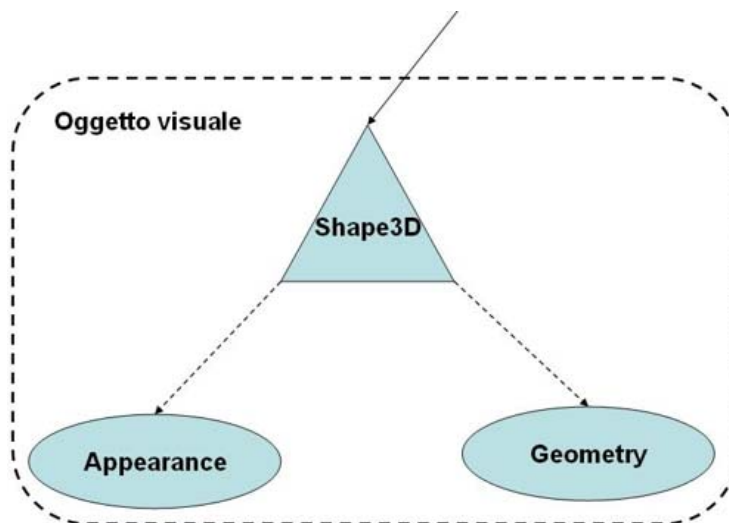
```
*      *      *
*      *      *
```


Capitolo 6

Geometry e le sue sottoclassi

Tutte le **geometrie** degli **oggetti visuali** sono definite da un **insieme di vertici collegati**.

Java 3D mette a disposizione **classi e metodi** per definire 'vertice per vertice' una **Geometry**, da associare poi a uno o più **oggetti Shape3D** (i collegamenti tra **Shape3D** e **Geometry** avvengono con **relazioni Reference**, mediante il metodo `setGeometry(Geometry)`).



GEOMETRY E LE SUE SOTTOCLASSI: PANORAMICA

La classe **Geometry** è la **superclasse** utilizzata per creare **geometrie** definendo i **singoli vertici** delle stesse.

Sono **figlie 'dirette'** di **Geometry** le classi **GeometryArray**, **CompressedGeometry**, **Raster** e **Text3D**; in questa sezione, prenderemo in esame **GeometryArray**, le sue (numerose) **figlie** e **Raster**.

GEOMETRYARRAY E LE SUE SOTTOCLASSI

In **GeometryArray** ad ogni vertice (che può essere anche uno solo, definendo quindi un punto) è associata una **locazione in un array**.

Nell'**array** verranno quindi definite le **coordinate** dei vari **vertici** componenti la **geometria**.

La **geometria** così definita verrà poi associata, mediante **Reference**, ad uno o più **Shape3D** dello **Scene Graph**.

Il **costruttore** principale di **GeometryArray** è **GeometryArray(int vertexCount, int vertexFormat)**, dove:

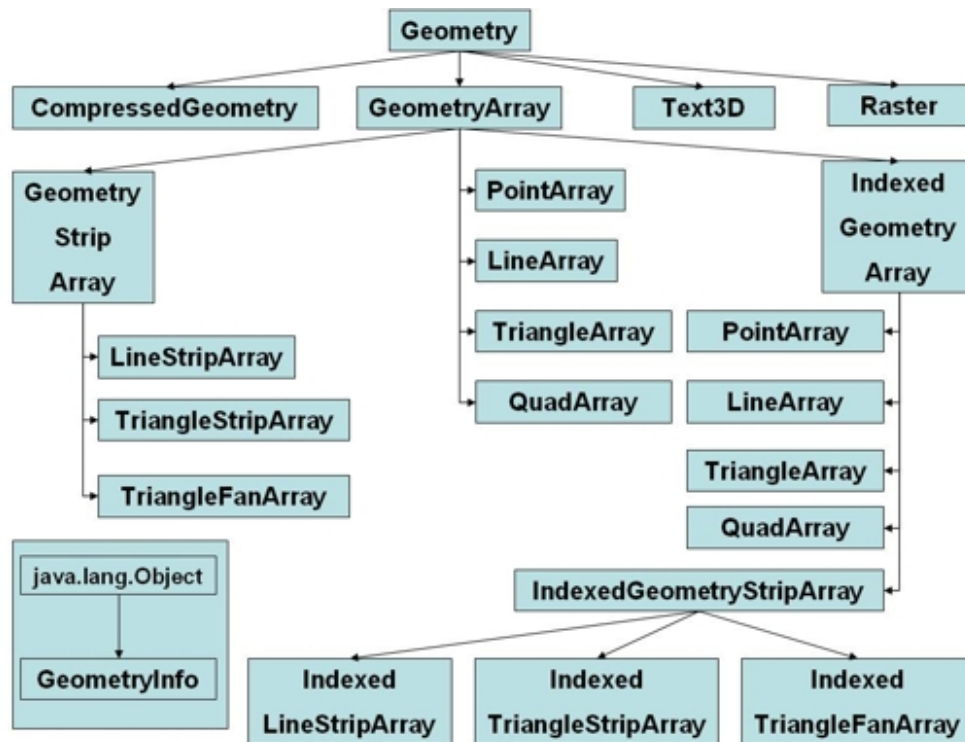
- **vertexCount** indica il numero di vertici presenti nell'array;
- **vertexFormat** è una maschera, cioè un campo dove ogni bit è un flag, dunque è possibile mettere anche più argomenti separati da un '|' (**or bit a bit**); i **flag** indicano quali informazioni sono presenti per ciascun vertice; campi possibili sono, ad esempio, **COORDINATES**, **NORMALS**, **COLOR 3** e **COLOR 4**.

E' qui che molte nozioni di **geometria** possono tornare utili...

Per definire le **coordinate dei vertici** di alcuni **poligoni**, infatti, è possibile fare affidamento, per esempio, su uno o più **cicli for** per generare, in base ad una **formula** precedentemente definita, i **parametri**: è così possibile creare **solidi regolari** con **poche righe di codice** anziché andando a definire le **coordinate vertice per vertice** (soluzione buona, tra l'altro, solo per un determinato problema, non scalabile).

I **vertici**, con le rispettive **coordinate**, vengono **aggiunti ad un GeometryArray** con:

- `setCoordinate(int index, double[] coordinate);`
- `setCoordinate(int index, float[] coordinate);`
- `SetCoordinate(int index, Point3d coordinate);`
- `setCoordinate(int index, Point3f coordinate);`



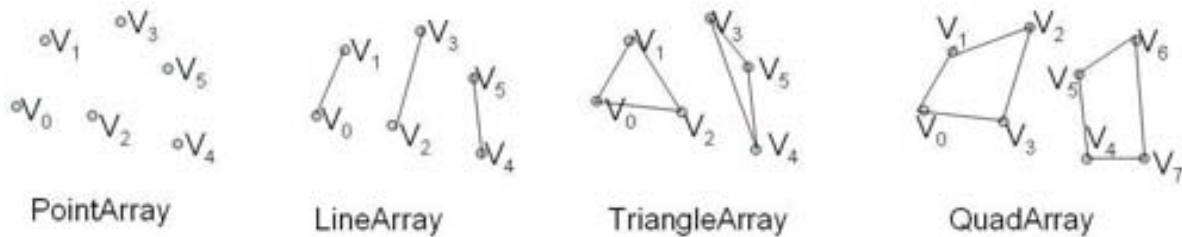
Le **firme** sono molto simili: **double** e **float** si differenziano per la precisione raggiunta, mentre la scelta tra valori numerici ed oggetti **Point** va fatta considerando che tipo di operazioni è necessario svolgere partendo dai **valori delle coordinate** (una semplice memorizzazione può essere effettuata usando solo **float** e **double**, mentre **Point** mette a disposizione **metodi** per calcolare facilmente, ad esempio, la distanza tra due punti, ecc..).

Vi sono poi **metodi** per gestire i **colori dei vertici** (**byte[]**, **color3***, **color4***, ...), le **normali** (con vettori o **float[]**) e le **coordinate delle textures**: verranno presi in esame prossimamente.

SOTTOCLASSI DI GEOMETRY ARRAY

GeometryArray mette a disposizione delle **sottoclassi** utili per gestire **geometrie particolari** (singoli punti, linee, poligoni), come:

- **PointArray;**
- **LineArray;**
- **TriangleArray;**
- **QuadArray.**



Vi è poi ***IndexedGeometryArray***, che permette di specificare come debbano essere **collegati i vertici indicizzati**, permettendone il **riuso**, per formare le **geometrie** desiderate.

Un'altra **figlia** interessante di **GeometryArray** è **GeometryStripArray**, che consente di **riutilizzare i vertici** secondo schemi ben definiti (una **geometria** creata con una **sottoclasse** diretta di **GeometryArray** non consente, infatti, il **riuso dei vertici**).

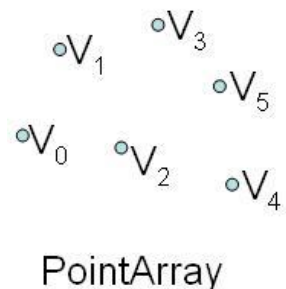
Le **figlie di GeometryStripArray** sono tre e verranno prese tutte in esame nei prossimi capitoli.

SOTTOCLASSI DI GEOMETRYARRAY – POINTARRAY

Un **PointArray** contiene un **insieme di coordinate** che verranno poi rappresentate visivamente come **punti non collegati**.

Costruttori:

- `PointArray(int vertexCount, int vertexFormat);`
- `PointArray(int vertexCount, int vertexFormat, int texCoordSetCount, int[] texCoordSetMap).`



Sebbene i punti di un **PointArray** siano visivamente scorrelati, essi fanno parte di **un unico oggetto**: spostando, ruotando e ridimensionando un oggetto con **geometria PointArray**, i punti verranno traslati e ruotati e le loro distanze relative ridimensionate.

I **metodi** per impostare le coordinate dei vari vertici sono quelli **ereditati da GeometryArray**.

In **Appendice** vi è il **codice sorgente** dell'esempio **'EsempioPointArray.java'**, che mostra come creare e visualizzare una **geometria PointArray**.

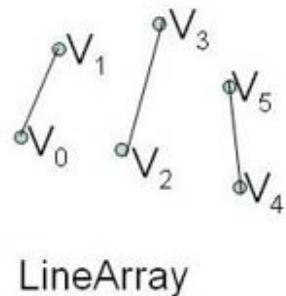
SOTTOCLASSI DI GEOMETRYARRAY – LINEARRAY

In un **LineArray** ogni **coppia di vertici** identifica un **segmento**.

Costruttori:

- `LineArray(int vertexCount, int vertexFormat);`
- `LineArray(int vertexCount, int vertexFormat, int texCoordSetCount, int[] texCoordSetMap).`

Il **numero dei vertici** deve essere necessariamente **pari**.

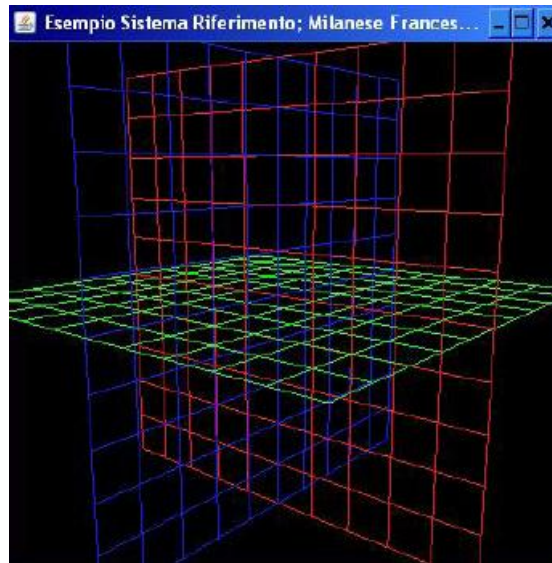


LineArray non consente il **riuso** dei vertici e non permette di definire **facce**: anche disponendo 4 segmenti a formare un quadrilatero, non avremo una figura piana ma solo 4 **segmenti**.

Per definire **facce (figure)** bisogna far uso di **TriangleArray**, **QuadArray** e simili.

L'esempio '*Esempio LineArray Quattro Vertici.java*' mostra come creare e visualizzare correttamente **geometrie di tipo LineArray**; l'esempio '*Esempio LineArray Tre Vertici.java*' mostra, invece, cosa succede creando un LineArray con un **numero dispari di vertici**: nessun problema in fase di compilazione, ma a **runtime** viene lanciata l'**eccezione** '**IllegalArgumentException: LineArray: illegalVertexCount**'.

Una **geometria** che spesso può tornare utile e che può essere definita già con un 'semplice' **LineArray** è una **griglia di riferimento** per la navigazione nello **spazio 3D**; l'immagine seguente fa riferimento alla **griglia** creata e visualizzata mediante il file d'esempio '*Esempio Sistema Riferimento.java*'.

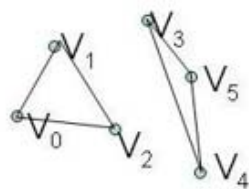


SOTTOCLASSI DI GEOMETRYARRAY – TRIANGLEARRAY

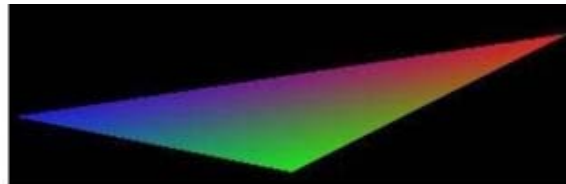
Come **LineArray** definisce segmenti, **TriangleArray** definisce triangoli, ma questa volta passiamo da semplici connessioni tra vertici a *facce*, la cui **superficie** può essere colorata o texturizzata.

TriangleArray crea triangoli prendendo **gruppi di tre vertici per volta** dall'**array delle coordinate** e collegandoli tra loro.

Il **numero di coordinate** deve essere, ovviamente, un **multiplo di tre**, altrimenti si avrà un **errore a runtime** (proprio come nel caso di un numero dispari di coordinate in un **LineArray**).



TriangleArray



Costruttori:

- `TriangleArray(int vertexCount, int vertexFormat);`
- `TriangleArray(int vertexCount, int vertexFormat, int texCoordSetCount, int[] texCoordSetMap).`

L'esempio '*EsempioTriangleArray.java*' mostra come creare **geometrie TriangleArray**.

Notare che, definendo il **colore per i vertici**, esso verrà 'esteso' a porzioni della **geometria**.

Java 3D gestirà automaticamente le sfumature, cosa che d'altronde viene fatta anche con i **LineArray**, in quel caso lungo il **segmento**.

SOTTOCLASSI DI GEOMETRYARRAY – QUADARRAY

QuadArray permette di generare **quadrilateri**.

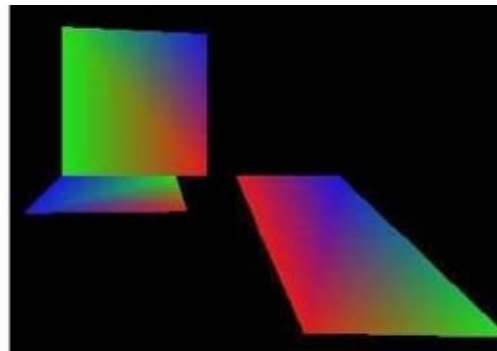
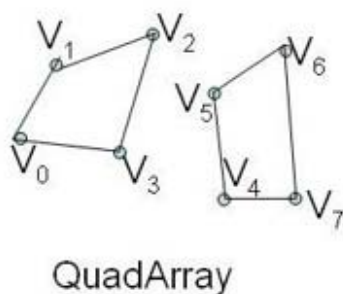
Come nel caso di **TriangleArray**, le figure così create saranno **poligoni chiusi**, con facce dalle **superfici colorabili o texturizzabili**.

Costruttori:

- `QuadArray(int vertexCount, int vertexFormat);`
- `QuadArray(int vertexCount, int vertexFormat, int texCoordSetCount, int[] texCoordSetMap).`

I **quadrilateri** vengono definiti prendendo le **coordinate dei vertici a gruppi di quattro** dall'**array** delle stesse.

Il **numero di coordinate** impostate nell'**array** deve essere un **multiplo di 4**, altrimenti si avrà una **eccezione a runtime**.



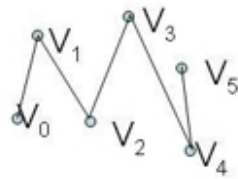
Esempio '*EsempioQuadArray.java*'.

SOTTOCLASSI DI GEOMETRYARRAY – GEOMETRYSTRIPARRAY

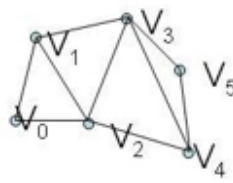
La classe **GeometryStripArray** permette di creare **geometrie** che implementano il **riuso** di uno o più **vertici** secondo 'schemi' predefiniti.

Ciò porta ad un **risparmio** in termini di linee di codice e di tempo di rendering (si fa meno lavoro, in entrambi i casi).

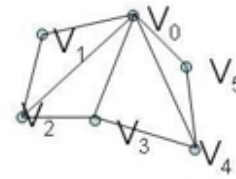
GeometryStripArray mette a disposizione le classi **LineStripArray**, **TriangleStripArray** e **TriangleFanArray**.



LineStripArray



TriangleStripArray



TriangleFanArray

Con un singolo oggetto **GeometryStripArray** è possibile definire più **gruppi, visivamente isolati ma facenti parte della stessa entità 'logica'** (lo stesso oggetto informatico).

Subentra quindi il problema di dover definire **a quale 'gruppo'** di un oggetto **Strip** appartiene un vertice: come dire a **Java 3D** quanti **strip separati** creare per la stessa istanza di un certo **GeometryStripArray** ?

Il **costruttore di GeometryStripArray** prende quindi in input un **terzo parametro**, oltre a **int vertexCount** e a **int vertexFormat: int[] stripVertexCounts**).

Questo **terzo parametro** è un **array di interi** che serve a specificare **quanti vertici** appartengono a ciascun **gruppo**.

Ad **esempio**, con 3 gruppi fatti di 4, 7 e 2 vertici (13 vertici in totale) avremo:

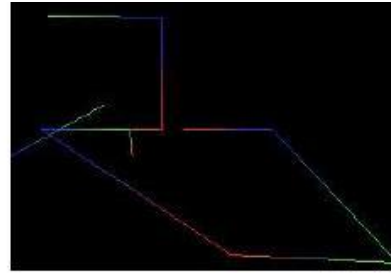
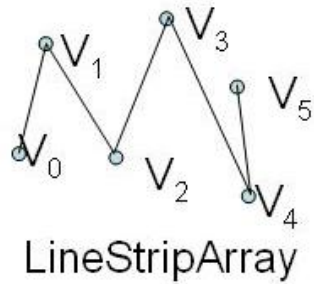
- `int[] verticiPerGruppo = {4, 7, 2};`
- `LineStripArray lsa = new LineStripArray(13, GeometryArray.COORDINATES | GeometryArray.COLOR 3, verticiPerGruppo);` .

SOTTOCLASSI DI GEOMETRYSTRIPARRAY – LINESTRIPARRAY

Un **LineStripArray** crea uno o più **gruppi di segmenti**.

I **vertici** sono collegati l'uno con l'altro seguendo il **valore del loro indice**, progressivamente.

E' possibile creare **più gruppi di segmenti (sconnessi)** in uno stesso **LineStripArray** mediante il **parametro stripVertexCounts**, come spiegato nell'introduzione alle classi della famiglia **GeometryStripArray**.



Costruttori principali:

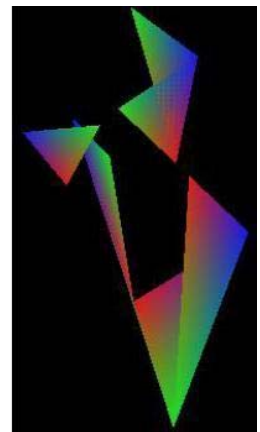
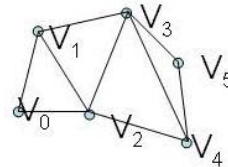
- `LineStripArray(int vertexCount, int vertexFormat, int[] stripVertexCounts);`
- `LineStripArray(int vertexCount, int vertexFormat, int texCoordSetCount, int[] texCoordSetMap, int[] stripVertexCounts);`

Esempio: 'Esempio LineStripArray.java'.

SOTTOCLASSI DI GEOMETRYSTRIPARRAY – TRIANGLESTRIPARRAY

Un **TriangleStripArray** genera uno o più **poligoni** composti da **facce triangolari**.

Il **numero di vertici per gruppo** non deve essere più un multiplo di 3, ma **almeno 3** (il triangolo di base).



Costruttori principali:

- `TriangleStripArray(int vertexCount, int vertexFormat, int[] stripVertexCounts);`
- `TriangleStripArray(int vertexCount, int vertexFormat, int texCoordSetCount, int[] texCoordSetMap, int[] stripVertexCounts);` .

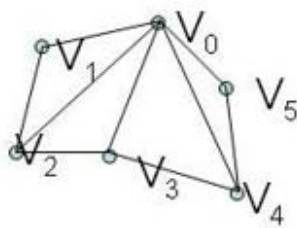
Esempio: 'Esempio TriangleStripArray.java'.

SOTTOCLASSI DI GEOMETRYSTRIPARRAY – TRIANGLEFANARRAY

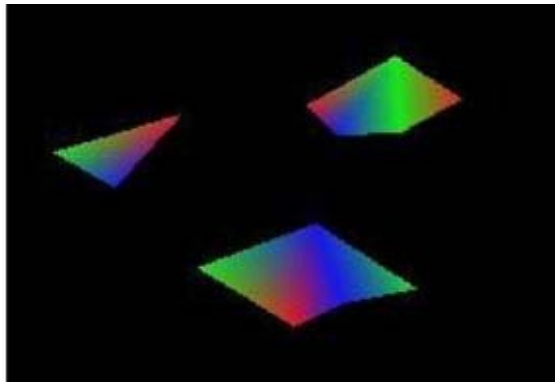
TriangleFanArray permette di creare **poligoni** collegando **'a ventaglio' (fan)** gruppi di **tre vertici**, formando quindi delle **facce triangolari**.

'A ventaglio' significa che, a partire **dal terzo vertice in poi**, i vertici successivi (dello stesso insieme, come definito dal **parametro stripVertexCounts**) verranno collegati con:

- il primo vertice in assoluto (V0, un 'pivot' per il gruppo);
- il vertice precedente (ad es., il quarto vertice, V3, verrà collegato a V2; V4 a V3 e così via).



TriangleFanArray



I **costruttori** sono:

- `TriangleFanArray(int vertexCount, int vertexFormat, int[] stripVertexCounts);`
- `TriangleFanArray(int vertexCount, int vertexFormat, int texCoordSetCount, int[] texCoordSetMap, int[] stripVertexCounts);` .

Esempio: 'Esempio TriangleFanArray.java'.

INDEXEDGEOMETRYARRAY E LE SUE SOTTOCLASSI

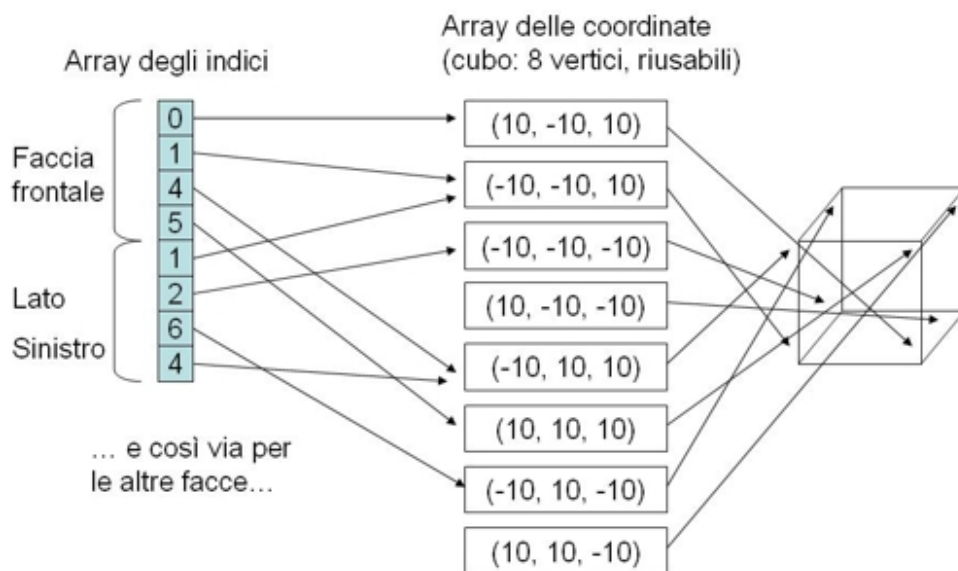
IndexedGeometryArray eredita direttamente da **GeometryArray**.

Tale **classe** fa uso di **due array**: in un array vengono inserite le **coordinate di un insieme di vertici**, nel secondo viene specificato **in che ordine collegare** gli stessi.

Ciò consente un notevole risparmio in termini di codice (**riuso di più vertici**), al prezzo di un **maggior tempo di rendering** e di una **maggior quantità di memoria** (viene introdotto un secondo array).

La figura seguente mostra come definire ad esempio un **cubo**, facendo uso di un **array di coordinate di otto elementi (vertici)** e dell'**array di indici** per definire le **facce**.

Notare come certi **vertici** vengano **riutilizzati** più volte, da varie locazioni dell'**array di indici**.



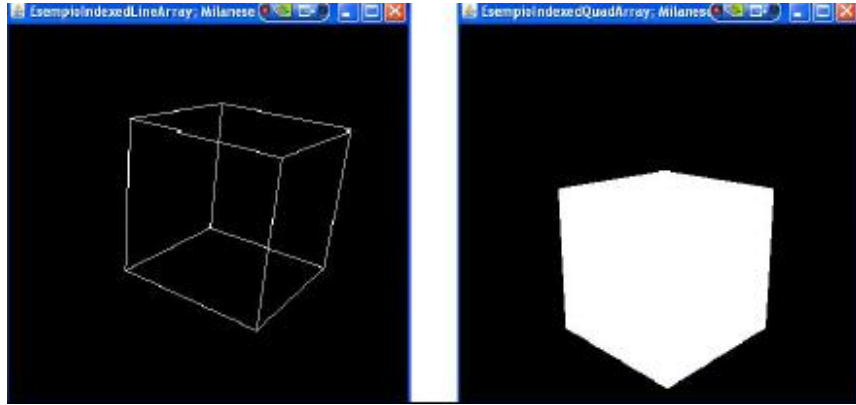
Esempi: *'Esempio IndexedLineArray.java'* e *'Esempio IndexedQuadArray'*.

I due **esempi** evidenziano che:

- **IndexedLineArray** crea sempre dei **segmenti**, **non facce**; anche disponendo 4 segmenti a formare, *visivamente*, un quadrato (servono 4 punti e 8 indici), avremo comunque 4 segmenti, non un poligono, quindi non potremo colorare o texturizzare una faccia (*non c'è una faccia !*);

- **IndexedQuadArray** (ma un discorso analogo può essere fatto per **IndexedTriangleArray**) crea invece **veri e propri poligoni chiusi**, prendendo 4 indici alla volta e chiudendoli, formando una

faccia che può essere colorata o texturizzata; inoltre, per formare un quadrato, stavolta 'reale', servono solo 4 punti e 4 indici.



GEOMETRYINFO

GeometryInfo è una **classe** che eredita direttamente da **java.lang.Object**.

Permette di definire **poligoni arbitrari** specificando solo i **vertici**, senza dover prevedere e definire i triangoli o i quadrilateri necessari per **collegare internamente i vertici**: tale compito verrà infatti svolto da **GeometryInfo**, tramite una delle tre **classi d'appoggio** che verranno prese in esame a breve.

Dobbiamo quindi **definire i vertici della nostra geometria** (anche molto complessa), delegando a **GeometryInfo** il compito di creare le **connessioni interne**, creando tante **facce triangolari**.

Dal punto di vista implementativo, questi sono i **passi necessari** per creare una **geometria** facendo uso di **GeometryInfo**:

1. creare un oggetto **GeometryInfo**, specificando come vogliamo che **GeometryInfo** raggruppi i vertici (triangoli, quadrilateri o poligoni);
2. impostare i **vertici della geometria**;
3. creare un oggetto **NormalGenerator** e passargli, come parametro, l'oggetto **GeometryInfo**;
4. creare un oggetto **Stripifier**, che raggruppa i triangoli adiacenti in '**triangle strips**', strisce di triangoli, e passargli come parametro l'oggetto **GeometryInfo**;
5. recuperare la **geometria del GeometryInfo** così definito.

I costruttori di **GeometryInfo** sono:

- `GeometryInfo(GeometryArray ga);`
- `GeometryInfo(int primitive);` .

Il parametro '**int primitive**', nel secondo **costruttore**, può assumere uno dei seguenti valori:

- **TRIANGLE ARRAY;**
- **QUAD ARRAY;**
- **TRIANGLE FAN ARRAY;**
- **TRIANGLE STRIP ARRAY;**
- **POLYGON ARRAY.**

Per impostare l'**array delle coordinate** (un array di float, double, Point3d o Point3f) e l'**array degli StripCounts** (un array di interi), utilizzare:

- `mioGI.setCoordinates(arrayCoordinate);`
- `mioGI.setStripCounts(stripCount);`

con '**mioGI**' oggetto di tipo **GeometryInfo**.

Per le fasi relative alla **creazione delle normali e allo Stripifier**, scrivere:

- `NormalGenerator mioNG = new NormalGenerator();`
- `mioNG.generateNormals(mioGI);`
- `mioGI.recomputeIndices();`
- `Stripifier mioSF = new Stripifier();`
- `mioSF.stripify(mioGI);`
- `mioGI.recomputeIndices();`

Si consiglia di utilizzare **prima il NormalGenerator, poi lo Stripifier**.

Per **recuperare la geometria vera e propria**, per farla **referenziare** da uno o più oggetti visuali **Shape3D** (con il metodo **setGeometry** di **Shape3D**) utilizzare:

`oggettoShape3D.setGeometry(mioGI.getGeometryArray());`

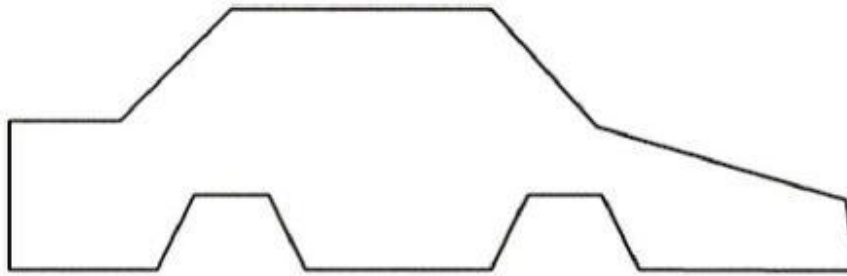
con '**oggettoShape3D**' un oggetto di tipo **Shape3D** e '**mioGI**' un oggetto di tipo **GeometryInfo**.

GEOMETRY INFO: UN ESEMPIO

File sorgente dell'esempio: 'Esempio GeometryInfo.java'.

Qui di seguito viene descritto *cosa* mostra l'esempio e *come* realizza il risultato.

L'idea è quella di creare una **geometria** un pò complessa, che richiami quella di un'automobile (ma ad un livello di dettaglio basso !). Il **profilo dell'auto** possiamo definirlo come nella figura seguente.



Impostiamo quindi le **coordinate** per un lato dell'auto, mantenendo il valore di **Z fisso**.

Scegliamo, come **coordinate dei vertici** della figura che descrive il profilo dell'auto, i seguenti **valori**:

(-10f, 0f, 0f); (-5f, 0f, 0f); (-3f, 2f, 0f); (0f, 2f, 0f); (2f, 0f, 0f); (7f, 0f, 0f); (9f, 2f, 0f); (12f, 2f, 0f); (14f, 0f, 0f); (19f, 0f, 0f); (19f, 2f, 0f); (12f, 5f, 0f); (7f, 10f, 0f); (0f, 10f, 0f); (-5f, 5f, 0f); (-10f, 5f, 0f); (-10f, 0f, 0f).

Gli stessi valori possono essere utilizzati per definire le **coordinate dei vertici per l'altro lato dell'auto**, cambiando la **coordinata Z** in, ad esempio, -10 (dunque l'auto sarà 'larga 10').

Nel file d'esempio vi sono, inoltre, quattro valori in più, utilizzati per definire i vertici che costituiscono il paraurti anteriore.

Potete provare a definire i vertici per ricoprire tutto il modello, per fare pratica.

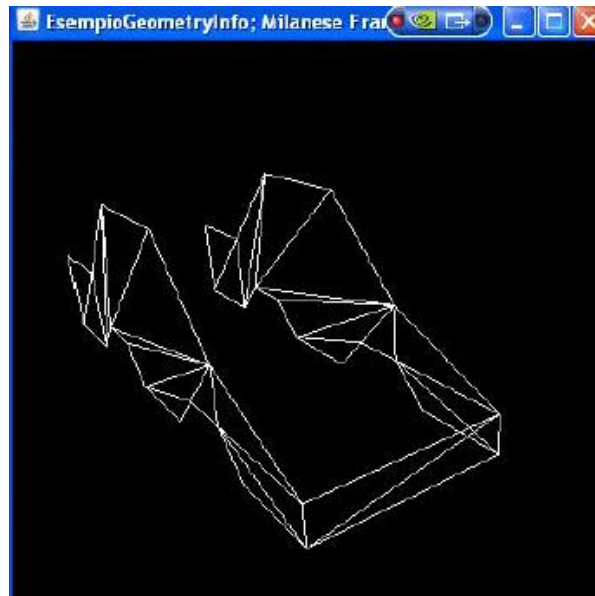
Impostiamo quindi il **numero di vertici per strip** (che sono tre: i due lati e il paraurti anteriore):

```
int[] stripCount = {17, 17, 4}; .
```

A questo punto seguono i passi necessari per creare un **GeometryInfo**, passandogli le **coordinate dei vertici** e lo **stripCount**; bisognerà, poi, richiamare il **NormalGenerator** e lo **Stripifier**.

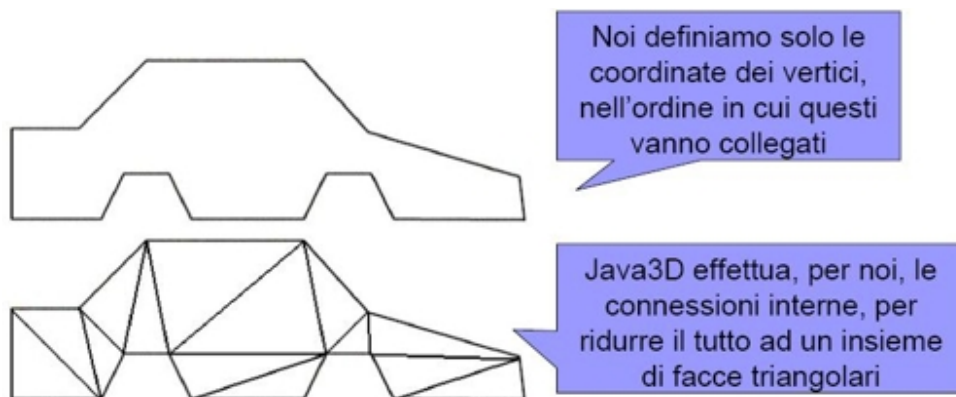
E' da osservare come non stiamo definendo esplicitamente le **connessioni tra i vertici**, nè stiamo dando indicazioni di sorta a **GeometryInfo** su come crearle.

L'**output a video** dell'applicazione in esecuzione ci mostra, comunque, come tali **connessioni** siano state create (figura seguente).



NOTA: vediamo la visualizzazione **struttura (wireframe)** per via di come è stata impostata la modalità di resa dei poligoni, '**LINE**', cioè appunto i **segmenti**, per mettere in evidenza il lavoro fatto da **GeometryInfo**; per visualizzare le facce piene, sostituire '**LINE**' con '**FILL**'.

GeometryInfo ha svolto per noi un bel pò di lavoro, creando 'al meglio' (**Java 3D** non ci dice comunque *come*, con che algoritmo) le **connessioni tra i vertici**.



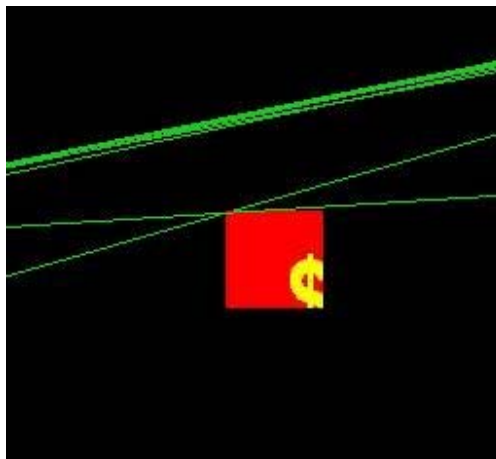
RASTER

Raster eredita direttamente da **Geometry**.

Un **oggetto Raster** permette di disegnare un'immagine in un punto dell'universo virtuale 3D.

GEOMETRICAMENTE, un **raster** 'risiede' nel punto ove è posizionato: **Java 3D** considererà, come estensione del **raster**, quella del punto, per cui tutte le **collisioni** o i **picking** basati su **geometrie** verranno presi in considerazione solo quando riguarderanno il punto che definisce il **raster** (l'immagine 2D mostrata a video non è '**pickable**' e non genera **collisioni**).

Il **costruttore** di base è **Raster()**, che crea un oggetto con **valori di default** per vari parametri, definibili in maniera esplicita con altri tre **costruttori**.



Tra i **metodi** più interessanti, abbiamo (significati ovvi):

- `setSize(int width, int height) : void;`
- `setPosition(Point3f pos) void;`
- `setImage(ImageComponent2D image); .`

Esempio: '*EsempioRaster.java*' (nota: è necessario creare un file **immagine** chiamato **Prova.JPG** – o cambiare il path del file immagine nel codice sorgente dell'esempio – e posizionarlo nella stessa cartella del file **.class** generato compilando *EsempioRaster.java*).

```
*      *      *
*      *      *
```

Capitolo 7

Le trasformazioni

Le **trasformazioni sugli oggetti** che studieremo in questa sezione riguarderanno gli oggetti come **mesh**, ossia applicando le trasformazioni **a tutti i vertici** degli stessi **in maniera omogenea** (tranne in un particolare caso, lo **scaling vettoriale**).

Tali **trasformazioni** consistono in **traslazioni, rotazioni e ridimensionamenti**.

Per il momento queste **trasformazioni** saranno **statiche**, vedremo come **animarle** nei paragrafi del capitolo dedicato, appunto, **all'animazione**; sempre nella sezione riguardante **l'animazione** vedremo, inoltre, come **animare indipendentemente gruppi di vertici di una mesh**, senza modificare l'oggetto a livello globale.

TRASLAZIONI

Quando un **oggetto** viene creato ed inserito come **figlio diretto** della **radice del Content Branch Graph** la sua posizione iniziale, all'interno dell'**universo virtuale**, è, di default, l'**origine (0, 0, 0)**.

Per spostarlo bisogna ricorrere ad un **vettore**: la **posizione finale** dell'oggetto corrisponderà alla **somma delle coordinate di partenza e delle coordinate espresse mediante il vettore**, secondo la regola della **somma vettoriale**.

Esempio di vettore utile a definire l'entità di una **traslazione**:

```
Vector3f vettoreTraslazione = new Vector3f(0.0f, 0.0f, -5.0f); .
```

Il **vettore della traslazione** è solo il primo passo: esso **rappresenta l'entità della trasformazione**.

Per definire una **trasformazione** vera e propria bisogna creare un '**oggetto trasformazione**':

```
Transform3D t3d = new Transform3D(); .
```

Un **oggetto Transform3D** è un oggetto **trasformazione** 'generico': viene utilizzato per creare **traslazioni, rotazioni, ridimensionamenti** (anche **combinazioni** di tali **trasformazioni**).

Per specificare la '**componente di traslazione**' del **Transform3D**, utilizziamo il **metodo**:

```
setTranslation(Vector3*);
```

di **Transform3D**.

A questo punto, bisogna inserire la **trasformazione nello Scene Graph** (**Transform3D** è un **oggetto** 'informatico', non un **nodo** o una **foglia** dello **Scene Graph**).

Per farlo, utilizzeremo l'**oggetto** (questa volta sia informatico che dello **Scene Graph**) **TransformGroup**:

```
TransformGroup tgTraslazione = new TransformGroup(t3d);  
objRoot.addChild(tgTraslazione); .
```

I **TransformGroup**, spesso abbreviati in **TG**, sono **Gruppi**: aggiungendo dei figli, questi saranno soggetti alla **trasformazione** definita dal padre:

```
tgTraslazione.addChild(oggetto1); tgTraslazione.addChild(oggetto5);  
tgTraslazione.addChild(oggettoN); .
```

Per aggiungere più **oggetti**, ciascuno con la sua **trasformazione**, si consiglia di creare un **array di posizioni e un array di oggetti** (quelli da posizionare, appunto), e di eseguire un ciclo che crei i vari vettori, le trasformazioni e i TG, aggiungendo di volta in volta i TG così definiti al gruppo (**BranchGroup**, o **TransformGroup** per creare trasformazioni in cascata), collegato allo **Scene Graph**.

Esempio: 'Downtown.java'.

E' possibile applicare una **traslazione** anche al TG che controlla la **ViewingPlatform**, in modo da **spostare il punto di vista** iniziale dell'osservatore all'interno della **scena 3D** di una quantità a piacere.

ROTAZIONI

La procedura necessaria per **ruotare un oggetto** è simile a quella di traslazione, nel senso che anche qui bisogna **creare un oggetto Transform3D** da associare ad un **TransformGroup** cui collegare uno o più **oggetti**; la differenza sta nel fatto che, questa volta, per **ruotare intorno agli assi X, Y e Z**, utilizzeremo i **metodi**:

- `rotX(double a);`
- `rotY(double a);`
- `rotZ(double a);`

con **a** = **angolo di rotazione**, espresso in *radianti*.

E' possibile **combinare più trasformazioni differenti**, creando ad esempio due trasformazioni di rotazione e **'moltiplicandole'**.

Per svolgere tale operazione, **Transform3D** mette a disposizione **due metodi**:

- `Transform3D nuovaRotazione = Transform3D.mul(t3d1, t3d2);`
- `t3d1.mul(t3d2);`

nel secondo caso, il risultato **sovrascriverà t3d1**.

Per un **esempio** di utilizzo 'semplice' delle trasformazioni di rotazione si rimanda a **'SecondoEsempio.java'**, visto nell'ultimo paragrafo del secondo capitolo, che mostra come effettuare una **rotazione intorno al centro dell'oggetto** (default).

Vedremo presto come effettuare **rotazioni più complesse**, combinate e intorno a punti da noi definiti.

RIDIMENSIONAMENTI (SCALING)

Per effettuare dei **ridimensionamenti (scaling)** degli oggetti visuali si parte sempre da **Transform Group** e da **Transform3D**, come detto per **Traslazione** e **Rotazione**.

Transform3D mette a disposizione due **metodi** per lo **scaling**: uno per lo **scaling uniforme (o 'scalare')**, l'altro per lo **scaling non uniforme (o 'vettoriale')**:

- `setScale(double scale) : void`
- `setScale(Vector3d scale) : void`

Nel paragrafo '**Scaling in cascata**' verrà presentato un file **d'esempio** che mostra come implementare lo **scaling uniforme**, mentre nel capitolo '**Un esempio completo**' verrà mostrato come implementare anche lo **scaling vettoriale**.

IL PIVOTING

Finora abbiamo creato dei **TransformGroup** semplici, per ruotare o traslare oggetti intorno all'**origine** del sistema.

La cosa si complica per creare **trasformazioni più complesse**, ad esempio per **combinare** rotazioni e traslazioni.

Questo concetto è di notevole importanza non solo per le **trasformazioni statiche** ma anche (e, forse, soprattutto) per creare le **animazioni**.

Fondamentale è il concetto di **pivot (lett.: 'perno')** delle trasformazioni.

Il **pivot** identifica un punto *a partire dal quale (traslazione) o intorno al quale effettuare una trasformazione*.

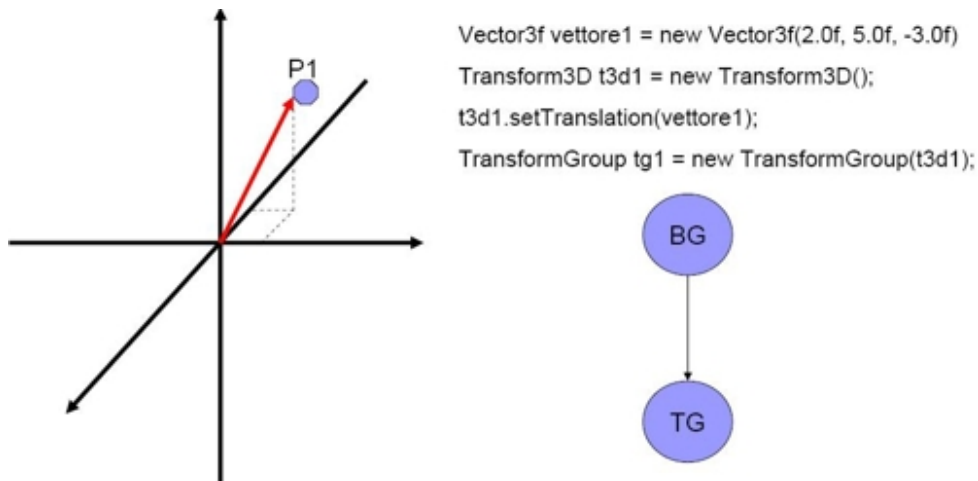
Esamineremo quindi il concetto di **pivoting** in relazione a **traslazione** e **rotazione** e vedremo come applicare questi concetti allo **Scene Graph** per implementare le **trasformazioni** volute.

TRASLAZIONI E PIVOTING

La **traslazione** è il caso più semplice: se aggiungiamo un **TransformGroup** (che chiameremo ad esempio tg1), con una sua **trasformazione di traslazione** (che chiameremo ad esempio t3d1), in cima al **Content Branch Graph**, la **traslazione** verrà effettuata *relativamente all'origine del sistema* (il **pivot**, in questo caso, è l'origine).

Il **vettore di traslazione** di t3d1 parte quindi dall'origine e **identifica un nuovo punto di arrivo nello spazio**. Chiamiamo tale punto P1.

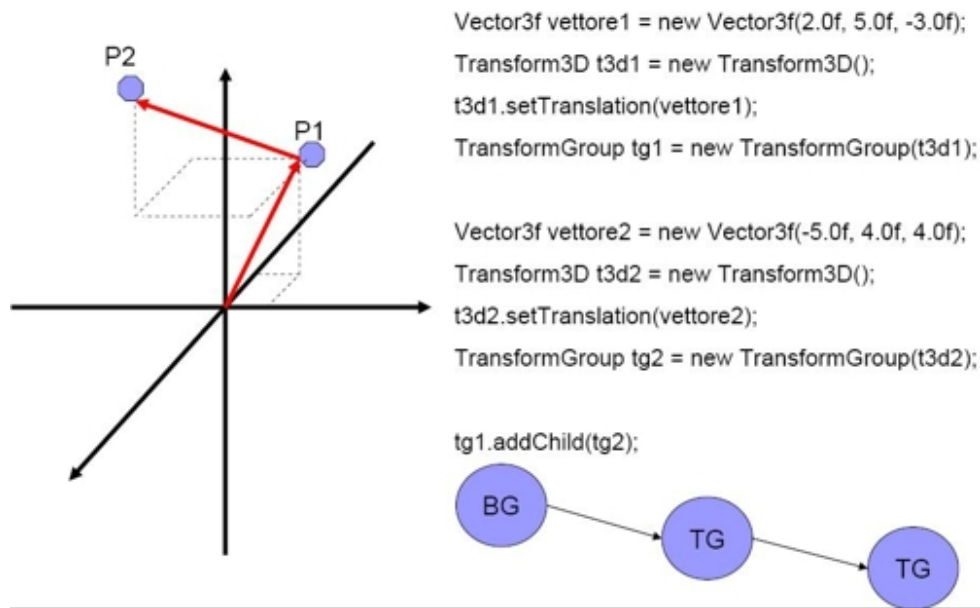
Un nuovo **oggetto visuale** collegato, come **figlio**, a tg1, verrà quindi **centrato** in P1.



Se a tg1 colleghiamo ora, come **figlio**, un nuovo **TransformGroup** (tg2), con una sua **trasformazione di traslazione** (t3d2), il **vettore** di t3d2 partirà a P1.

Le **coordinate di arrivo** di t3d2, *in riferimento all'origine dell'universo virtuale di Java3D*, vengono calcolate con la **somma vettoriale dei vettori** di t3d1 e t3d2 ed identificano un **secondo punto P2** (coordinate *globali* di P2: t3d1 + t3d2; *coordinate locali*, cioè *in riferimento al nodo padre* di P2: t3d2).

Un **oggetto visuale** collegato, come **figlio**, a tg2, verrà **centrato** in P2.



ROTAZIONI E PIVOTING

Finora abbiamo visto la **rotazione** di oggetti che si trovavano nell'origine; in quel caso, il **pivot della trasformazione** coincideva con il **centro dell'oggetto**.

Quando però un oggetto viene **traslato**, bisogna fare attenzione a dove **posizionare la trasformazione di rotazione**, al fine di evitare effetti indesiderati.

Sostanzialmente, quando bisogna **ruotare** un oggetto bisogna porsi la seguente domanda: **'rispetto a cosa lo stiamo ruotando'** ?

Ebbene, se dobbiamo **ruotare** un oggetto **intorno al suo centro**, possiamo applicare la **trasformazione** di rotazione **direttamente** al TG che controlla la posizione dell'oggetto, cioè suo **padre**.

Nell'esempio **'RotazionePivot1'** è possibile vedere un cubo spostato in avanti rispetto al punto di vista dell'osservatore di 5 unità e ruotato, intorno all'asse Y , di $\pi/4$ radianti; per far questo, si è reso necessario creare due TG: il primo per traslare il cubo, il secondo per **ruotarlo intorno al suo centro**.

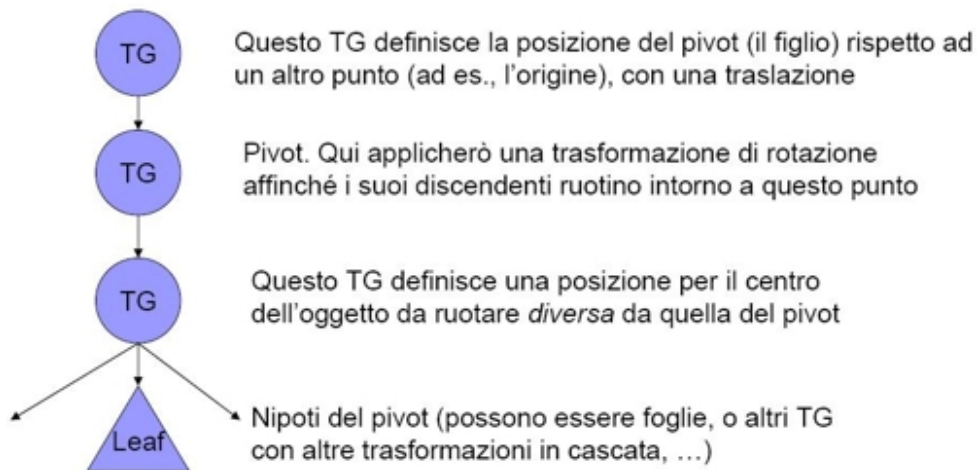
Per ruotare l'oggetto intorno ad un altro punto, **un punto esterno**, cosa bisogna fare ?

Bisogna creare una **catena di (almeno) due TG** ed applicare la trasformazione di rotazione al nonno dell'oggetto, o ancora più in alto, a seconda delle esigenze.

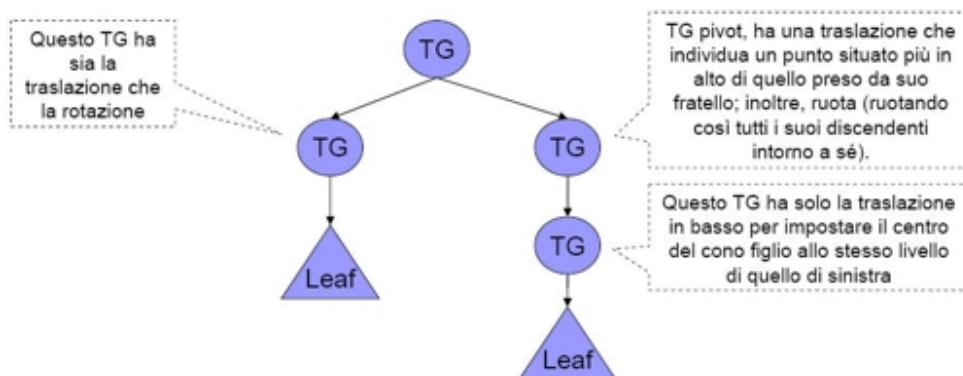
E' possibile creare e collegare, quindi, **rotazioni in cascata**.

Il **nonno** dell'oggetto può quindi definire la **posizione del pivot nello spazio 3D**, mentre suo figlio (il padre dell'oggetto) definisce la **posizione dell'oggetto**.

Applicando la **rotazione sul nodo TG padre**, l'oggetto ruoterà su se stesso; applicando la **rotazione sul nodo TG nonno**, l'oggetto ruoterà intorno al **pivot**.



Un **esempio** pratico è **'RotazionePivot2'**.



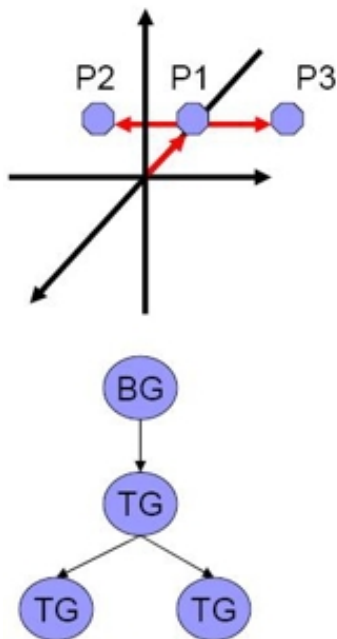
Qui abbiamo due coni, posti di fronte all'osservatore, uno a sinistra e l'altro a destra, solo che quello di sinistra è ruotato su se stesso, mentre quello di destra ruota intorno ad un **pivot esterno**, posizionato nel **vertice** (la 'punta' del cono).

SCALING IN CASCATA

Che succede se definiamo un **Transform Group tg1** con uno **scaling** e ad esso agganciamo due TG figli, tg2 e tg3 (tra loro fratelli), con tg3 provvisto di un'**ulteriore trasformazione di scaling** ?

Quello che si verifica è che tg1 crea uno **scaling** che verrà **applicato a tutti i suoi eredi**, quindi i figli di tg2 e di tg3 verranno scalati in prima istanza del valore impostato per tg1; inoltre, tg3 (per il quale avevamo previsto un'**ulteriore trasformazione di scaling**) verrà scalato una seconda volta, questa volta del fattore impostato in tg3.

In pratica, stiamo componendo una scena come quella rappresentata dalla figura seguente.



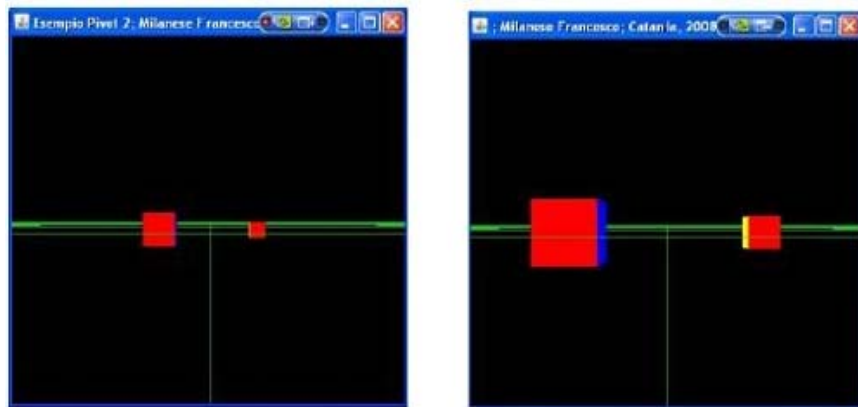
```
Vector3f vettore1 = new Vector3f(0.0f, 0.0f, -15.0f);
Transform3D t3d1 = new Transform3D();
t3d1.setTranslation(vettore1);
t3d1.setScale(0.5);
TransformGroup tg1 = new TransformGroup(t3d1);
```

```
Vector3f vettore2 = new Vector3f(-3.0f, 0.0f, 0.0f);
Transform3D t3d2 = new Transform3D();
t3d2.setTranslation(vettore2);
TransformGroup tg2 = new TransformGroup(t3d2);
```

```
Vector3f vettore3 = new Vector3f(3.0f, 0.0f, 0.0f);
Transform3D t3d3 = new Transform3D();
t3d3.setTranslation(vettore3);
t3d3.setScale(0.5);
TransformGroup tg3 = new TransformGroup(t3d3);
```

```
tg2.addChild(new ColorCube());
tg3.addChild(new ColorCube());
tg1.addChild(tg2);
tg1.addChild(tg3);
```

L'**output** a schermo di questo **esempio** (definito in '*EsempioScalingInCascata*') è visibile, invece, nella figura successiva, sulla sinistra.

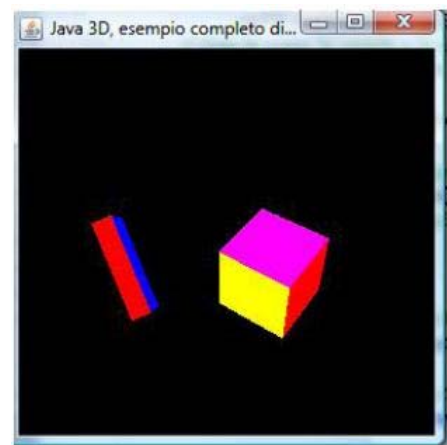


Nella stessa figura, ma sulla destra, è visibile l'**output ottenuto non scalando** t3d1 nello stesso file d'esempio (il punto di osservazione e le posizioni degli oggetti nell'universo 3D sono gli stessi per entrambi gli screenshots, sono le dimensioni degli oggetti che sono cambiate).

In pratica, nell'immagine di sinistra, il cubo di destra è ad un quarto della dimensione originale, in quanto è stato dimezzato prima da t3d1 e poi da t3d3.

UN ESEMPIO COMPLETO

Il file d'esempio '*EsempioRotoTrasScale.java*' mostra due oggetti ColorCube, creati inizialmente con dimensioni e posizioni di default, poi **traslati**, **ruotati** (il primo intorno ad un solo asse, il secondo intorno a tutti e tre gli assi) e **scalati** (il primo in maniera **non uniforme**, o '**vettoriale**', il secondo in maniera **uniforme**, '**scalare**').



Il lettore può, per esercizio, provare a disegnare lo **Scene Graph** di questo esempio.

* * *
* * *

Capitolo 8

L'interazione

Attraverso **mouse e tastiera** è possibile '**muoversi**' nell'**universo virtuale di Java3D** o **interagire** con gli oggetti.

L'interazione consiste quindi nel **modificare l'universo virtuale** (punto di osservazione incluso) in base ad **un'azione** compiuta dall'utente.

Per implementare queste operazioni è necessario introdurre il concetto di **Behavior**, oggetto che esamineremo a breve.

In questo capitolo partiremo dalle **interazioni di base** per arrivare ai **Behavior per il mouse** e al **Picking Callback**, mostrando anche come fare per **navigare nell'universo virtuale** mediante **tastiera**.

I BEHAVIOR

Behavior è una **classe astratta** che eredita da **LEAF**, dunque le sue implementazioni verranno rappresentata mediante un triangolo nello **Scene Graph** e non potranno avere figli (sempre in termini di oggetti dello **Scene Graph**).

Il compito di un **Behavior** è quello di modificare lo **Scene Graph** in base a stimoli prodotti dall'utente (limitatamente alle possibilità fornite dalle capability dei vari oggetti, che devono essere correttamente impostate).

Figli di **Behavior** sono, ad esempio:

- **Billboard;**
- **Interpolator;**
- **KeyNavigatorBehavior;**
- **LoD;**
- **Mouse6DPointerBehavior;**
- **MouseBehavior;**
- **PickMouseBehavior;**
- **ViewPlatformBehavior.**

In questa sezione prenderemo in esame i **Behavior di mouse e tastiera**.

Concetti chiave, di base di un **Behavior**, e che si tradurranno in metodi da implementare, sono *initialization* e *processStimulus*.

initialization viene invocato solo una volta: quando il **Behavior** prende vita, insieme al resto del mondo virtuale; si tratta, proprio come dice il nome, dell'inizializzazione di questo componente.

processStimulus viene invocato quando si verifica un **evento**; il metodo si occupa anche di decodificare i messaggi in input per capire *quale* evento si è verificato ed agire di conseguenza.

Un altro elemento di **Behavior** è la *scheduling region*: si tratta di una **regione dello spazio 3D** che definisce il **'raggio d'azione'** del **Behavior**.

Un **Behavior** è attivo (ossia: può ricevere stimoli, effettua azioni) se la **regione di attivazione della View Platform** interseca quella del **Behavior**.

Ad ogni modo, per poter portare ad una conseguenza, un'azione deve agire su una **wakeUpCondition** del **Behavior**.

Se più **Behavior** possono essere risvegliati dalla stessa condizione, è possibile specificare l'ordine di esecuzione delle risposte mediante lo **schedulingInterval**.

Infine, il **trigger** : si tratta di una o più condizioni **WakeUpCondition**.

Il **trigger** di un **Behavior** viene inizializzato per la prima volta da **initialization**, ma deve essere resettato esplicitamente alla fine del metodo **processStimulus**.

Behavior è una classe astratta.

All'atto di scrivere un proprio **Behavior**, bisogna definire **almeno un costruttore** e riscrivere **initialization** e **processStimulus**.

Ricapitolando, ecco come funziona la **gestione degli eventi mediante Behavior**:

- all'atto della **creazione**, viene richiamato **initialization**;
- **'quando succede qualcosa' nel bound d'azione del Behavior**, viene richiamato **processStimulus**, e...
- ...si fa un controllo: l'**evento** che si è verificato è tra quelli gestiti dal **Behavior** ? Si scorre quindi l'**elenco dei trigger (le condizioni di wakeup del Behavior)** e, se sì, viene eseguita l'azione definita nel blocco di codice relativo presente in **processStimulus**.

A questo punto abbiamo visto cos'è un **Behavior** e come è composto, ma non come agisce.

Ad un **Behavior** è associato un **Transform Group 'target'** (che poi nello **Scene Graph** sarà collegato al **Behavior**), un bersaglio le cui **proprietà possono essere modificate dal Behavior**.

Ad un **Behavior** è possibile associare più TG, quindi la **relazione tra i due non è di tipo padre-figlio**; **non** è neanche una vera e propria **Reference**, in quanto non vi è un metodo **setBehavior** o qualcosa di simile: semplicemente, **si passa il riferimento ad un TG mediante il costruttore o altri metodi del Behavior**.

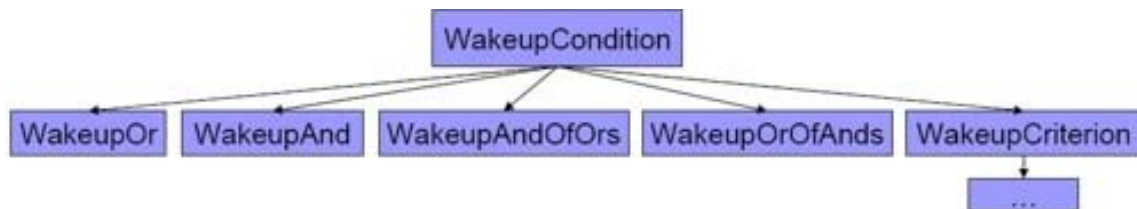
All'interno di **processStimulus** si renderà poi necessario **distinguere gli eventi che attivano il Behavior** e scrivere il codice che effettua le relative **azioni**.

CONDIZIONI DI WAKEUP

Le **condizioni di Wakeup** derivano dalla classe **WakeupCondition** e da **WakeupCriterion** e permettono di definire molte **condizioni** in base alle quali un **Behavior** può attivarsi, come ad esempio la **pressione di un tasto della tastiera o del mouse**.

Altre **quattro classi**, figlie dirette di **WakeupCondition**, permettono l'**OR**, l'**AND**, l'**OR di AND** e l'**AND di OR di insiemi di condizioni**, come suggeriscono i loro nomi:

- **WakeupOr**,
- **WakeupAnd**,
- **WakeupAndOfOrs**,
- **WakeupOrOfAnds**.



Tra le **classi** che derivano da **WakeupCriterion** abbiamo:

- **WakeupOnActivation**, **WakeupOnDeactivation** : quando il volume della **ViewPlatform** interseca il bound del **Behavior** o, al contrario, non lo interseca più;
- **WakeupOnAWTEvent** : quando si verifica un evento AWT (es.: tastiera);
- **WakeupOnBehaviorPost** : riguarda una 'interazione' tra più **Behavior**, messi in comunicazione tra loro;
- **WakeupOnCollisionEntry**, **WakeupOnCollisionExit** : se due oggetti collidono o la collisione è appena terminata;
- **WakeupOnCollisionMovement** : attivato durante la collisione tra due oggetti;
- **WakeupOnElapsedTime**, **WakeupOnElapsedFrames** : quando è trascorso un certo numero di millisecondi o di frames;
- **WakeupOnSensorEntry**, **WakeupOnSensorExit** : quando un dispositivo di input di qualunque tipo interseca o cessa di intersecare un bound (specificato);
- **WakeupOnTransformChange** : quando cambia la trasformazione affine di un determinato TG;
- **WakeupOnViewPlatformEntry**, **WakeupOnViewPlatformExit** : quando avviene l'intersezione o l'uscita della **ViewPlatform** con un **bound** (specificato).

CREARE DEI SEMPLICI BEHAVIOR: ESEMPI

ESEMPIO 1

L'esempio '*Eempio Behavior1*' mostra la creazione e l'utilizzo di un semplice **Behavior** che permette di **spostare un cubo nello spazio 3D** mediante i tasti freccia, pageup, pagedown e 'r' (per 'reset').

Il **Behavior**, definito nel file '*mioBehavior*', è figlio di un **TransformGroup** (il '**target**' del **Behavior**) che è a sua volta figlio del **Branch Group** agganciato al **Simple Universe**.

Il **Behavior**, provvisto del suo *bound d'azione*, viene quindi aggiunto allo **Scene Graph** attraverso il suo **TG target**.

Al **Behavior** che ci permette di muovere l'oggetto è associata una **BoundingSphere**, di centro l'origine e raggio 100.0, che definisce il luogo dello spazio ove il **Behavior** è valido.

NOTA: se il cubo e il **Behavior** si trovano a distanza 100 dall'origine, il **Behavior** sarà ancora valido, dunque sarà possibile spostare ancora il cubo (nel nostro caso, a 102), ma a questo punto, fuori dal **bound**, il **Behavior** non verrà più preso in considerazione da **Java3D** e non riusciremo più a muovere il cubo.

Vedremo, successivamente in questo stesso capitolo, due modi per risolvere questo problema.

ESEMPIO 2

In alcuni videogiochi, specialmente fps, dopo aver aperto, ad esempio, una porta, trascorso un pò di tempo questa spesso si richiude da sola.

L'esempio '*Eempio Behavior2*' mostra come implementare tale funzionalità: premendo il tasto 'e' si attiva l'apertura della porta (se questa non è già aperta); dopo pochi secondi, la porta si chiude da sola.

Il **Behavior** per questo esempio è definito nel file '*mioBehavior2*'.

Tale esempio fa quindi uso di due **condizioni di wakeup: WakeupOnAWTEvent (per la tastiera) e WakeupOnElapsedTime.**

In **initialization** e alla fine di **processStimulus**, con il reset, viene specificato che gli eventi da tastiera devono essere accettati solo se la 'porta' è 'chiusa', ovvero se l'angolo di rotazione del cubo vale 0.

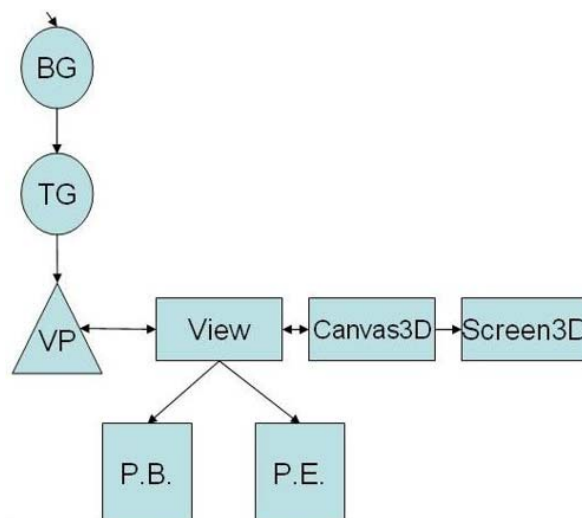
Se la porta è chiusa e viene premuto il tasto 'e', si modifica il **TransformGroup** che controlla il cubo e si lancia un evento **WakeupOnElapsedTime**, che verrà gestito dallo stesso **Behavior**, con apposite porzioni di codice.

L'esempio in esecuzione non mostra, in realtà, l'animazione della rotazione dell'oggetto, e questo proprio perché bisognerebbe gestire, appunto, una **animazione**, operazione complessa, non ancora presa in esame, il cui codice potrebbe confondere il lettore nell'analisi del sorgente del file d'esempio; è possibile vedere però l'effetto del cambio dei valori di rotazione del **TransformGroup** osservando **il colore delle faccia del ColorCube** posta di fronte all'osservatore.

KEYNAVIGATORBEHAVIOR

Abbiamo visto che è possibile **interagire con lo Scene Graph**, modificando le proprietà dei **Transform Group** e, quindi, degli **oggetti figli dei TG**.

Come anticipato nella sezione dedicata al **Simple Universe**, vi è un **TG che controlla la View Platoform**, e quindi il **punto di vista dell'osservatore** (e quello che verrà visualizzato sullo schermo, in un certo senso).



E' logico e corretto supporre quindi di poter **modificare le proprietà di questo TG** tramite un **Behavior**, al fine di **'navigare' nello spazio 3D**.

Java3D mette a disposizione un **Behavior** già pronto per l'uso, è sufficiente creare un **Simple Universe** e scrivere:

```
SimpleUniverse sU = new SimpleUniverse(canvas3D);  
TransformGroup vpTG =  
    sU.getViewingPlatform().getViewPlatformTransform();  
KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);  
keyNavBeh.setSchedulingBounds( new BoundingSphere( new Point3d(),  
    1000.0));  
scene.addChild(keyNavBeh); .
```

Stiamo modificando, quindi, la **posizione e l'orientamento della View Platform**.

Il **Behavior** 'precotto' messo a disposizione da **Java3D** permette di **navigare nello spazio virtuale** con questi **controlli**:

ROTAZIONI

FrecciaDestra: ruota a destra

FrecciaSinistra: ruota a sinistra

PagSu: ruota verso il basso

PagGiù: ruota verso l'alto

TRASLAZIONI

ALT e FrecciaSu: trasla in avanti

ALT e FrecciaGiù: trasla indietro

ALT e FrecciaDestra: trasla a destra

ALT e FrecciaSinistra: trasla a sinistra

ALT e PagSu: trasla verso l'alto

ALT e PagGiù: trasla verso il basso

La **scena** definita nel file di **esempio 'Esempio BehaviorNavigazione.java'** permette, appunto, di **navigare nello spazio 3D** utilizzando tale **Behavior**.

E' possibile creare, comunque, **un proprio Behavior di navigazione**.

E' quello che succede in '*Esempio NavBehavior2.java*'; in questo caso, i **comandi** sono:

W: spostamento in avanti di una unità;

A: spostamento a sinistra di una unità;

S: spostamento indietro di una unità;

D: spostamento a destra di una unità;

FracciaSu: rotazione dell'osservatore, verso 'l'alto';

FrecciaGiù: rotazione dell'osservatore, verso 'il basso';

FrecciaSinistra: rotazione dell'osservatore, verso 'sinistra';

FrecciaDestra: rotazione dell'osservatore, verso 'destra'.

In questo caso si fa uso di due **classi**: **EsempioNavBehavior2**, che definisce la scena e contiene il main dell'applicazione, e '**MioBehaviorTastiera**', che definisce i **controlli** (i tasti) da accettare e come **modificare il Transform Group target**.

Il **Behavior** di controllo definito in **MioBehaviorTastiera** non è diverso da quelli visti in **mioBehavior** e **mioBehavior2**: quello che cambia è che, in questo caso, il **Behavior** viene creato nel **costruttore**, prende come **parametro il View Platform TG** e viene agganciato direttamente alla **radice del Conten Branch Graph**.

BOUNDINGLEAF E BOUND DI DIMENSIONE INFINITA

I **Behavior** e molti altri oggetti di **Java3D** hanno bisogno, come abbiamo visto, di una '**bounding zone**' che determini dove tali oggetti agiscono e se i loro effetti debbano essere visualizzati o meno (a seconda dell'intersezione del **bound** con la **View Platform**).

Con alcuni oggetti può sorgere un problema di **dimensioni della bounding region**; ad esempio, se con la **View Platform** ci si sposta al di fuori della **zona d'influenza del Behavior** di navigazione (ossia: al di fuori della sua **bounding region**), potrebbe risultare impossibile muoversi !

Ci sono due metodi per risolvere questo genere di problemi:

- l'uso di **POSITIVE INFINITY** di Float;
- la creazione di una **Bounding Leaf**.

La prima soluzione, molto semplice da implementare (una sola riga di codice !), consiste nel mettere, come valore della **dimensione della bounding region**, '**Float.POSITIVE INFINITY**'.

La seconda soluzione, un pò più articolata, prevede l'utilizzo di una **Bounding Leaf**.

Una **Bounding Leaf** è un oggetto dello **Scene Graph** a tutti gli effetti, che definisce, internamente, una **bounding region**.

Il punto fondamentale è che, essendo la **Bounding Leaf un oggetto dello Scene Graph (una Leaf)**, può essere agganciata ad un **Transform Group**, dunque la **zona d'influenza** può **muoversi** nell'**universo 3D**.

La **Bounding Leaf** può essere condivisa da più oggetti dello **Scene Graph**, se questi mettono a disposizione un **metodo** del tipo '**setInfluencingBoundingLeaf**'; non si tratta, quindi, di **relazioni di tipo Padre-Figlio** tra elementi dello **Scene Graph**, ma di **collegamenti di tipo Reference**.

Per utilizzare una **Bounding Leaf** da 'muovere' insieme alla **View Platform**, associando la **Bounding Leaf al Behavior** di navigazione nello **spazio 3D** da tastiera, è necessario inserire la **Bounding Leaf** come **figlia del Transform Group che controlla la View Platform**.

Nel **Simple Universe**, il **View Platform Transform Group** è un TG particolare, che accetta, come figli creati esternamente, solo dei **Branch Group**.

Bisogna quindi creare un **Branch Group** temporaneo da inserire come **figlio del VPTG** nello **Scene Graph**, ed agganciare a tale **Branch Group** la **Bounding Leaf**, così:

```
TransformGroup vpTG =  
    mioSimpleUniverse.getViewingPlatform().getViewPlatformTransform(  
    );  
BoundingLeaf bl = new BoundingLeaf();  
bl.setRegion(new BoundingSphere(new Point3d(), 10.0));  
BranchGroup bgTemp = new BranchGroup();  
bgTemp.addChild(bl);  
((TransformGroup) (mioSimpleUniverse.getViewingPlatform().  
    getViewPlatform().getParent())).addChild(bgTemp);
```

Quando creeremo il **Behavior** per la navigazione da tastiera, imposteremo come **Bounding Leaf** dello stesso la **bounding leaf** creata precedentemente; in questo modo, la sua **regione di influenza** si muoverà con l'osservatore, rendendo il **Behavior** 'sempre attivo':

```
MioBehaviorNavigazione behaviorNavigazione = new
    MioBehaviorNavigazione(vpTG) ;
behaviorNavigazione.setSchedulingBoundingLeaf(bl) ;
miaScena.addChild(bbehaviorNavigazione) ; .
```

Quando utilizzare la soluzione di **PositiveInfinity** e quando utilizzare una **BoudingLeaf** ?

Dipende dal progetto: le **zone di influenza** possono essere piccole ma rendere i **Behavior** 'sempre attivi' ? Devono cioè muoversi ? Quante devono essere ?

E' possibile creare e **riusare una singola bounding region** sempre valida ?

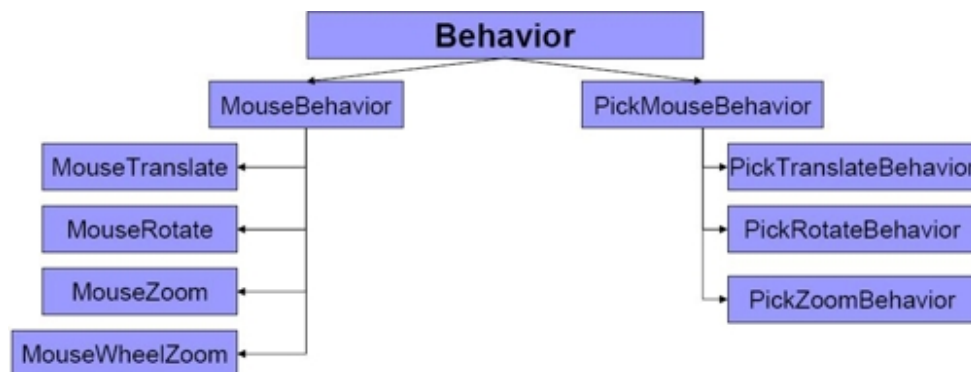
Abbiamo bisogno di una **bouding region** enorme o possiamo limitarci ad alcune **bounding box** disposte in una certa maniera (ad esempio, per definire i punti ove è possibile muoversi in uno scenario che rappresenta un piano di un edificio) ?

Il problema va valutato caso per caso.

BEHAVIOR PER IL MOUSE

E' possibile interagire con l'universo virtuale di **Java 3D** tramite il **mouse**, con diverse operazioni, come **selezioni** (identificando l'oggetto sul quale avviene il click), **traslazioni** o **rotazioni**.

Discendono direttamente da **Behavior**, ad esempio, le **classi** mostrate nella figura seguente.



MOUSE BEHAVIOR

Con i **MouseBehavior** è possibile, **muovendo il mouse**:

- **traslare** un oggetto *sul piano XY globale* (tenendo premuto il tasto destro del mouse);
- **traslare** un oggetto *lungo l'asse Z globale* (tenendo premuto il tasto centrale del mouse);
- **ruotare** un oggetto (tenendo premuto il tasto sinistro del mouse).

Il **Transform Group** da modificare deve diventare il **target del Mouse Behavior** (anche di più **Mouse Behavior** contemporaneamente).

Bisogna, inoltre, definire i **bound d'azione** per questi **Behavior**.

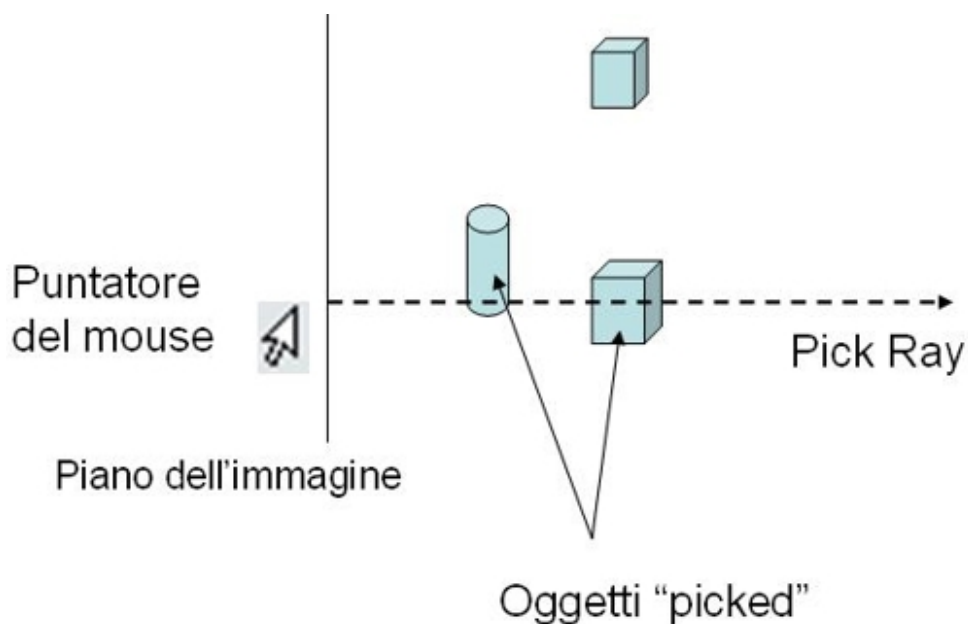
Esempio: 'Esempio MouseBehavior'.

IL PICKING

Il **Picking** consiste nell'identificare e selezionare un oggetto visibile sulla **Canvas3D** mediante un **click del mouse**.

Ad esempio, potrebbe essere utile per scegliere **quale oggetto ruotare con un MouseBehavior**, oppure per **recuperare** il nome o le **proprietà dell'oggetto** selezionato.

Java3D fa, per noi, il lavoro di **proiettare un ipotetico raggio attraverso l'universo virtuale e recuperare dette informazioni**.



E' possibile recuperare l'**oggetto più vicino** (il primo ad essere 'colpito' dal raggio) o la **lista di tutti gli oggetti intercettati**.

Ad ogni modo, non basta identificare l'oggetto selezionato: per certe operazioni potrebbe essere necessario, ad esempio, risalire al **Transform Group** che lo controlla.

Non tutti i **nodi** sono selezionabili con il **Pick Ray**: l'oggetto '**Node**' mette a disposizione proprio un **metodo** *setPickabel(boolean)* che, se invocato con **argomento false**, rende l'oggetto non selezionabile.

Escludere la **possibilità di selezionare** un oggetto con un **Pick Ray** ha molti scopi, non ultimo quello di abbassare la complessità dell'operazione, che richiede una quantità di risorse notevoli (l'operazione svolta dal **picking** non è per niente 'banale').

PICKMOUSEBEHAVIOR

La classe '**PickMouseBehavior**', appartenente al **package** *com.sun.j3d.utils.picking.behaviors* (esiste anche un altro package, deprecato), è la **superclasse per il Picking**.

Il **costruttore** è il seguente:

```
PickMouseBehavior(Canvas3D canvas, BranchGroup root, Bounds  
bounds) ; .
```

'**BranchGroup root**' serve ad identificare la **radice dell'albero / porzione di Scene Graph** da considerare quando si fa il test del **Picking**.

Selezionando un oggetto posto fuori dall'albero radicato in root, il **picking** non restituirà nulla.

Per **rendere 'pickable'** tutta la scena, impostare come root la **radice del Content Branch Graph**.

Sono **figlie di PickMouseBehavior**:

- `PickTranslateBehavior;`
- `PickRotateBehavior;`
- `PickZoomBehavior.`

E possibile usarle **tutte e tre contemporaneamente**, in quanto riguardano pulsanti del mouse diversi, proprio come avveniva con **MouseBehavior**.

E altresì possibile creare **classi di picking personalizzate**.

Un **esempio** di utilizzo di **PickMouseBehavior** è quello mostrato nel file *'EsempioPicking.java'*.

Leggendo il **codice** è possibile notare che qui il **Behavior** viene creato in fase di definizione della **scena**, non dentro il **metodo** che crea il **Content Branch Graph**.

Questo avviene principalmente perchè, per definire questo particolare **Behavior**, servono la **Canvas3D** e la **radice del Branch Graph** (è possibile passare questi parametri a *'createSceneGraph'* e creare il **Behavior** dentro quel **metodo**, ma questa rappresenta una soluzione meno 'pulita').

Si noti, infine, la nuova **capability** introdotta per i **Transform Group: ENABLE PICK REPORTING**, che include il TG nel **path del picking**.

MOUSEBEHAVIOR e PICKMOUSEBEHAVIOR.

Dagli **esempi** visti per **MouseBehavior** e per **PickMouseBehavior** le differenze tra queste due entità possono sembrare non rilevanti.

In realtà, i **MouseBehavior** hanno come target un **TransformGroup**.

PickMouseBehavior prende come parametri la **Canvas3D** e un **BranchGroup** che fa da 'tetto' allo stack di elementi che è possibile selezionare con **PickMouseBehavior**; inoltre, è anche possibile disattivare il **Picking** per alcuni **Transform Group figli del Branch Group 'tetto'**, semplicemente non impostando le **Capability** del **Picking**.

PICKINGCALLBACK

PickingCallback è un'interfaccia, utile per notificare il **picking** di un **oggetto** ad una **classe**.

Una classe interessata a sapere quando un oggetto viene selezionato col **picking** può infatti implementare tale **interfaccia**, sovrascrivendo il metodo *transformChanged(int type, TransformGroup tg)*, nel quale viene specificata l'azione da eseguire.

Il **Pick Behavior** da 'tracciare' deve invece impostare la **classe** di **callback** mediante il **metodo** *setupCallback(PickingCallback pc)*.

Le informazioni sul **picking** verranno quindi inviate alla **classe** che implementa **PickingCallback**.
In tale **classe** sarà possibile definire le azioni da intraprendere.

Un semplice **esempio** è quello definito nel file *'EsempioPickingCallback'*.

In questo caso abbiamo tre oggetti che effettuano il **Picking** con il **mouse** e un oggetto di tipo *miaClasseDiPickingCallback*.

Tale **classe** è definita nel file omonimo, che accompagna il file *'EsempioPickingCallback'*.

Con il metodo *setupCallback* impostiamo l'oggetto creato con *miaClasseDiPickingCallback* come **callback** dei tre oggetti di **picking**; dunque, quando effettueremo un **picking**, tale evento verrà notificato all'oggetto di tipo *miaClasseDiPickingCallback*.

In questo **esempio**, la classe che fa uso dell'**interfaccia** di **PickingCallback** è particolarmente povera: il **costruttore** è vuoto, mentre il metodo *transformChanged* stampa a video le informazioni sulle **trasformazioni** effettuate (il *toString* del **TransformGroup** selezionato e il tipo di **trasformazione** effettuata).

Il **tipo** di **trasformazione** effettuata viene indicato con **costanti** intere:

- **ROTATE (0);**
- **TRANSLATE (1);**
- **ZOOM (2);**
- **NO_PICK(3); .**

NO_PICK indica che è stata effettuata una **selezione**, ma non è stata intrapresa alcuna **azione**.

L'esempio *'Esempio PickingCallback2'*, che fa uso di *'miaClasse Di PickingCallback2'* e di *'mioTransformGroup'*, mostra un utilizzo più interessante dello strumento di **PickingCallback**.

In questo esempio, infatti, i **Transform Group** padri dei cubi visibili nella scena non sono 'semplici' TG ma oggetti *'mioTransformGroup'*, cioè TG provvisti di una **stringa** (il loro 'nome').

Quando verranno selezionati con il **picking**, la **classe** di **PickingCallback** associata ai **Behavior** del **mouse** leggerà i loro nomi e li stamperà a video.

All'estensione (informatica) delle **classi** di **Java3D** e su possibili utilizzi del **PickingCallback** è dedicato il prossimo paragrafo.

USERDATA. ESTENDERE LE CLASSI DI JAVA 3D

USERDATA

Spesso è desiderabile poter associare a degli oggetti dello **Scene Graph** delle 'informazioni extra', come un nome (una stringa, da visualizzare o da trasmettere a qualche altro oggetto), valori o anche qualcosa di più elaborato.

Se questa 'estensione' non ha a che fare con l'apporto di **campi** o **metodi supplementari**, e non c'è quindi bisogno di creare una nuova **classe** figlia dell'oggetto, è possibile far uso dei **metodi** *setUserData* e *getUserData* forniti dalla classe **SceneGraphObject** e disponibili per tutte le **classi** che la **estendono**.

Le **firme** complete dei due **metodi** sono:

- `getUserData() : java.lang.Object;`
- `setUserData(java.lang.Object) : void; .`

La loro **utilità** è chiara: permettono di associare un oggetto (nel senso informatico del termine) 'extra' ad un oggetto dello **Scene Graph**.

L'esempio più banale di utilizzo riguarda l'uso di una stringa contenente un'**informazione** (ad esempio, un nome) da mostrare a video quando l'oggetto (visuale, della **scena 3D**) che la contiene viene selezionato (**picking behavior**), 'urtato' (**collisioni**), ecc.

ESTENDERE LE CLASSI DI JAVA 3D

A volte si ha la tendenza a considerare le **classi di Java3D** come scatole nere, date così come sono dall'alto, senza possibilità di modifica: non è così, si tratta comunque di **classi** 'informatiche', **estendibili** come tutte le altre.

Ad esempio, possiamo creare una **classe** che **estenda** da **Transform Group** aggiungendo **campi** e **metodi**.

La porzione di **codice** riportata qui di seguito permette di creare, appunto, un **oggetto** che **estende** da **Transform Group**, ma che in più ha un campo testuale, il 'nome' dell'oggetto:

```
import javax.media.j3d.*;
public class mioTransformGroup extends TransformGroup
{
    private String nome;
    public mioTransformGroup(String s) { nome = s; }
    public void setName(String s) { nome = s; }
    public String getName() { return nome; }
}
```

Utilizzando questo oggetto insieme ad un **PickingCallback** sarà quindi possibile risalire al 'nome' dato all'oggetto utilizzato.

NOTA: nel **metodo transformChanged** della classe di **PickingCallback** dovremo effettuare un controllo; il **metodo getName()** sarà infatti presente solo nei 'nostri' oggetti che ereditano da **Transform Group**, per cui basterebbe **clickare** su un qualunque altro oggetto (anche su ciò che apparentemente sembrerebbe il nulla, cioè sullo sfondo) per generare un **errore a runtime**.

Ecco come controllare la 'natura' dell'oggetto selezionato:

```
public void transformChanged(int type, TransformGroup tg)
{
    if(tg instanceof mioTransformGroup)
        System.out.println(((mioTransformGroup)tg).getName());
}
```

Attenzione: l'esempio appena riportato è molto semplice, anche TROPPO: per fornire *poche* informazioni extra ad un oggetto e nessun **metodo** aggiuntivo, si può far uso di **USERDATA**, come mostrato precedentemente.

L'estensione (informatica) di classi dello **Scene Graph** è un'operazione 'pesante', alla quale bisognerebbe far ricorso solo in caso di reale **necessità**.

```
*      *      *
*      *      *
```

Capitolo 9

Le luci

Nel mondo reale, l'**illuminazione** di un oggetto dipende da fattori di diversa natura:

- le **proprietà fisiche** dell'oggetto;
- le direzioni dei **vettori delle luci**, dell'**osservatore** e delle **normali** alle superfici.

In **Java 3D**, le direzioni dei **vettori** contribuiscono a definire la resa visiva dell'oggetto proprio come nel mondo reale, mentre le proprietà **fisiche** degli oggetti vengono simulate agendo sulle impostazioni di **rendering**, definendo il **modello del colore** e il **modello di ombreggiatura** degli stessi.

Java 3D gestisce tre tipi di **riflessioni**:

- **ambientale**: dipende da una luce soffusa, sempre presente;
- **diffusa**: l'illuminazione 'normale', che mostra il colore dell'oggetto;
- **speculare**: il riflesso della luce sull'oggetto, generalmente di colore bianco.

Non vengono gestiti (non in maniera 'nativa', almeno) **ombre** e **riflessi tra più oggetti**.

I COLORI DEGLI OGGETTI IN JAVA 3D

Prima di iniziare, è bene sapere che **Java 3D** gestisce i colori basandosi sulle **componenti RGB**, non su un modello di comportamento **'fisico'**.

Le **componenti RGB** di una luce e quelle del colore di un oggetto vengono elaborate separando i **tre canali** e considerando, quindi, le **componenti RGB** della luce e del colore dell'oggetto come le componenti di due **vettori**; dunque, ad esempio, con una luce rossa (che ha solo componente R) e un oggetto verde (che ha solo una componente V) non vedremo nulla, o comunque vedremo una 'macchia nera' al posto del nostro oggetto: l'oggetto verde avrà, infatti, componente V nulla, dunque una luce R pura non lo **illuminerà-colorerà**.

LE LUCI NELLO SCENE GRAPH

Come (quasi) ogni altra cosa in **Java 3D**, le **luci** devono essere inserite nello **Scene Graph**; non solo: bisogna anche impostare le proprietà di **Material** degli oggetti da illuminare (operazione che verrà illustrata nel prossimo capitolo, dedicato all'aspetto visivo degli oggetti) e i **bound** associati alle **sorgenti luminose**.

A parte la **posizione** della **sorgente luminosa** all'interno della **scena 3D**, anche i **bound d'azione** dovranno essere impostati correttamente affinché tutto vada come previsto !

LA CLASSE LIGHT

La **classe Light** è una **classe astratta**, utilizzata per creare, appunto, le **luci**.

Costruttori:

- `Light()` ;
- `Light(boolean lightOn, Color3f color)` ;
- `Light(Color3f color)` .

La **classe** mette quindi a disposizione vari **metodi** per gestire lo **stato** ('**acceso/spento**'), il **colore**, la **regione di bound**, lo **scope** e altri **parametri**.

Java 3D mette a disposizione quattro **tipi** di luci e, quindi, di **classi**; di queste, le seguenti ereditano direttamente da **Light**:

- Luce ambientale → **AmbientLight**
- Luce direzionale → **DirectionalLight**
- Luce puntuale (omnidirezionale) → **PointLight**

mentre la quarta eredita da **PointLight**:

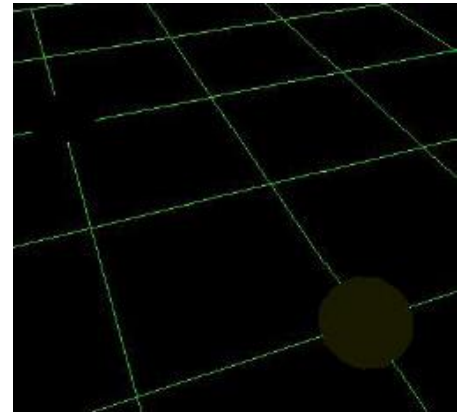
- Luce spot (faretto) → **SpotLight**

AMBIENT LIGHT

La **luce ambientale**, creata mediante **AmbientLight**, è un espediente utilizzato per simulare la luce che, nelle scene reali, viene **riflessa** dagli altri oggetti presenti, appunto, nell'**ambiente**.

Tale luce permette di vedere '**facce**' di oggetti che non sono illuminati in maniera diretta da una **fonte luminosa**. Si tratta di una luce **molto tenue**, presente in **eguale intensità in ogni punto della scena 3D** appartenente alla **bounding region** di influenza dell'oggetto **AmbientLight**.

Le **componenti RGB** della **luce ambientale** influiscono sul **colore ambientale dei materiali degli oggetti** presenti nel **bound d'influenza** della luce e che non siano esclusi dall'effetto della stessa mediante la definizione degli **Scope**.



Costruttori:

- `AmbientLight()` ;
- `AmbientLight(Color3f color)` ;
- `AmbientLight(boolean lightOn, Color3f color)` .

L'esempio '*AmbientLight.java*' mostra l'utilizzo di una **luce ambientale** con un **bound di influenza** molto piccolo: una sfera non è illuminata perché fuori dal **bound** (è possibile individuarla dal 'buco nero' che lascia nel **sistema di riferimento**; una seconda sfera è, invece, correttamente colorata del colore della **luce ambientale**).

DIRECTIONAL LIGHT

La **luce direzionale**, implementata mediante la **classe DirectionalLight**, fornisce un tipo di **illuminazione direzionale**, la cui **intensità** non si attenua.

I '**fasci**' luminosi sono **paralleli**.

DirectionalLight è indicata per simulare, ad esempio, la **luce solare**.

Ha effetti sui **colori diffuso e speculare** degli oggetti presenti nel suo **bound d'azione** e non esclusi mediante **Scope**.

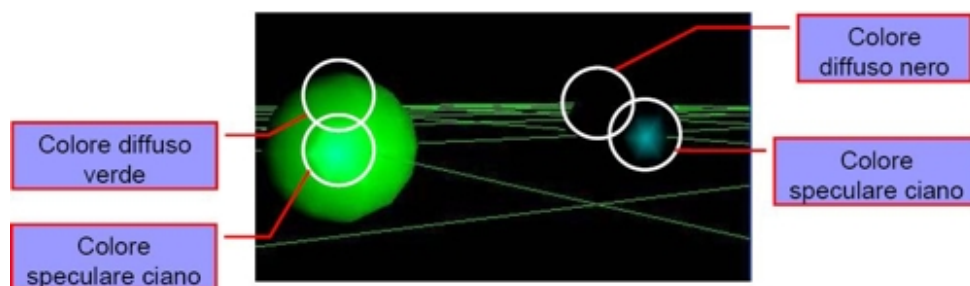
Costruttori:

- `DirectionalLight()` ;
- `DirectionalLight(Color3f color, Vector3f direction)` ;
- `DirectionalLight(boolean lightOn, Color3f color, Vector3f direction)` .

Il **vettore** passato come **argomento** (può essere comunque impostato / recuperato con appositi metodi, elencati nella documentazione di **Java 3D**) specifica la **direzione dei fasci luminosi**, considerando come punto di partenza l'**origine dell'universo virtuale** e come punto d'arrivo quello identificato dal **Vector3f**; ad ogni modo, l'origine dei fasci luminosi è '**at infinity**', cioè ad **infinito**, non nell'**origine**.

Il **vettore**, quindi, specifica solo la **direzione dei raggi luminosi**.

Questo concetto diverrà chiaro osservando l'immagine seguente, che fa riferimento **all'esempio LuceDirezionale**.



Nell'esempio, vi è una fonte di luce, di colore CIANO, che 'punta' a $(-5, 0, 0)$, che illumina solo parzialmente due sfere.

Soffermiamoci su come vengono colorate le due sfere.

La sfera nell'origine ha colore **diffuso** VERDE (0, 1, 0), la luce ha colore CIANO (0, 1, 1), dunque il colore *visibile* della sfera sarà VERDE, eccezion fatta per la regione del **riflesso speculare**: il colore speculare, che di default è bianco, verrà reso come CIANO, cioè il colore della fonte luminosa, che ha canale R nullo.

La seconda sfera ha colore **diffuso** ROSSO (1, 0, 0), dunque non subirà l'influenza della luce sul colore diffuso e verrà resa nera, eccezion fatta per la regione del riflesso speculare, che verrà resa color CIANO, per i motivi esposti precedentemente.

	Sfera 1	Sfera 2		Sfera 1	Sfera 2
Colore diffuso proprio	0.0--1.0--0.0	1.0--0.0--0.0	Colore speculare proprio	1.0--1.0--1.0	1.0--1.0--1.0
Colore luce	0.0--1.0--1.0	0.0--1.0--1.0	Colore luce	0.0--1.0--1.0	0.0--1.0--1.0
Colore diffuso finale	0.0--1.0--0.0	0.0--0.0--0.0	Colore speculare finale	0.0--1.0--1.0	0.0--1.0--1.0

POINT LIGHT

Il funzionamento di una **PointLight** è in tutto e per tutto simile a quello di una **lampadina a bulbo**: una sorgente di luce **omnidirezionale**, la cui **intensità luminosa** decresce (se lo si desidera, in **Java3D**) all'aumentare della **distanza**.

Costruttori:

- `PointLight()`;
- `PointLight(Color3f color, Point3f position, Point3f attenuation)`;
- `PointLight(boolean lightOn, Color3f color, Point3f position, Point3f attenuation)`.

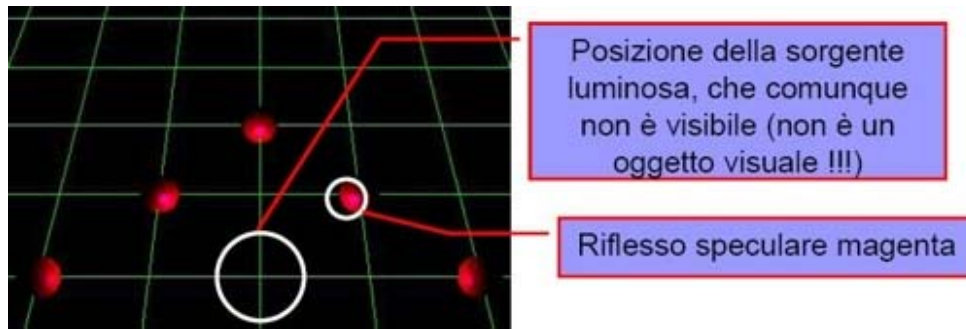
L'**attenuazione** è calcolata con un'equazione i cui **parametri** possono essere modificati; a tal proposito, **PointLight** mette a disposizione due **metodi**:

- `setAttenuation(float constant, float linear, float quadratic)`;
- `setAttenuation(Point3f attenuation)`.

Una **PointLight** influisce sui **colori diffuso e speculare degli oggetti** che si trovano nel suo **bound d'azione**.

Nel file di esempio *PointLight* abbiamo una **sorgente luminosa PointLight** di colore MAGENTA posta nell'**origine**.

Le 5 **sfere** presenti hanno colore diffuso ROSSO e colore speculare BIANCO (default), quindi con la luce MAGENTA il colore diffuso finale sarà il ROSSO e quello speculare il MAGENTA.



SPOT LIGHT

La classe **SpotLight** è figlia di **PointLight**: in effetti, è una **fonte di luce** che agisce in una determinata **direzione** e all'interno di un **cono d'azione**.

Costruttori:

- `SpotLight()` ;
- `SpotLight(Color3f color, Point3f position, Point3f attenuation, Vector3f direction, float spreadAngle, float concentration)` ;
- `SpotLight(boolean lightOn, Color3f color, Point3f position, Point3f attenuation, Vector3f direction, float spreadAngle, float concentration)` .

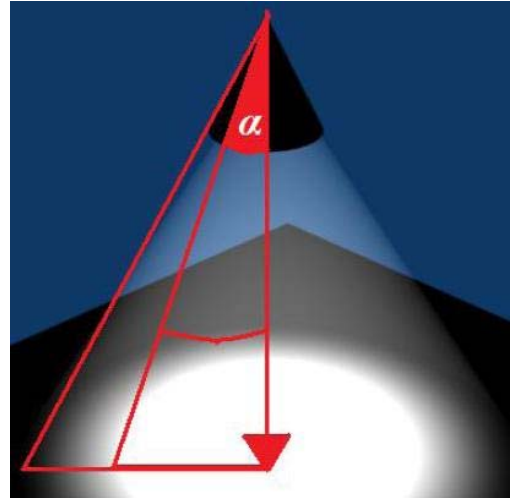
La **direzione** è espressa mediante un **vettore**.

Nel determinare l'**intensità luminosa** in un dato punto del **cono d'azione** non entra in gioco solo la **distanza** dalla **sorgente**, ma anche l'**angolo** formato dal **vettore direzione** e dal **vettore** passante per la **sorgente** e il punto interno al cono preso in esame.

Il valore di **concentration** varia da 0.0 (distribuzione uniforme) a 128.0 (luce concentrata).

Tra i **metodi** di **SpotLight** abbiamo ad esempio:

- `getConcentration() : float`
- `setConcentration(float concentration) : void`
- `getDirection(Vector3f direction) : void`
- `setDirection(Vector3f direction) : void`
- `setDirection(float x, float y, float z) : void`
- `getSpreadAngle() : float`
- `setSpreadAngle(float spreadAngle) : void`

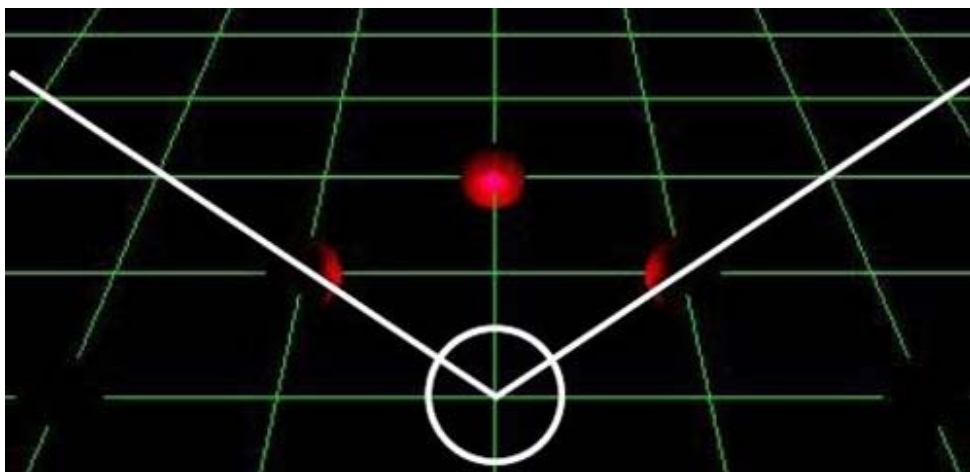


Tale tipo di **luce** ha effetti sui **colori diffuso e speculare** degli oggetti presenti nel suo **bound d'azione**.

E' in grado di illuminare anche solo **parti di oggetti** (le parti all'interno del **cono** di luce).

Nel file d'esempio *SpotLight.java* abbiamo una fonte di luce **SpotLight** posta nell'**origine** e diretta verso (0, 0, -10), con uno **SpreadAngle** di 45 gradi: solo la sfera centrale mostrerà il colore **diffuso** e il **riflesso speculare**, mentre due sfere risulteranno illuminate solo parzialmente, mostrando solo il colore **diffuso**, e altre due sfere non verranno illuminate affatto.

Il valore di **concentration** è quello di default: **0 (illuminazione uniforme)**.



L'ATTENUAZIONE DELLE LUCI

Può sembrare strano, ma è così: la **classe Light** e le sue **sottoclassi**, in **Java3D**, non mettono a disposizione un metodo per regolare la '**potenza luminosa** delle... luci (per non parlare delle '**luci negative**', presenti in alcuni ambienti di modellazione e resa **3D**, che non esistono affatto in **J3D**, almeno in maniera nativa).

E' possibile, ovviamente, '**sporcare**' una luce agendo sulle **componenti RGB**, 'scalando' il valore delle stesse (ad esempio, il bianco diventerà grigio, e così via), ma questa non è una operazione sulla **luminosità** in senso stretto !

Due **classi, figlie di Light**, mettono a disposizione un metodo che può essere utilizzato per agire sulla **potenza luminosa della sorgente: la classe PointLight e la classe SpotLight** (che eredita il **metodo** da **PointLight**); tale **metodo** si trova qui, e non in **Light**, perchè le altre due figlie di **Light**, **DirectionalLight** e **AmbientLight**, per definizione non sono soggette ad **attenuazione**.

I **metodi** che ci riguardano sono quindi:

- `getAttenuation(Point3f attenuation);`
- `setAttenuation(Point3f attenuation);`
- `setAttenuation(float constant, float linear, float quadratic);` .

Questi **metodi** permettono di **recuperare (get) o impostare (set)** tre valori, che **Java** utilizzerà per definire un **fattore di attenuazione**.

Tali valori sono:

- attenuazione costante;
- attenuazione lineare;
- attenuazione quadratica.

La **potenza luminosa** verrà quindi regolata in base alla **distanza dalla sorgente** e al **valore dell'attenuazione**.

Di default, il valore di **attenuazione** costante è 1, mentre gli altri due valori sono 0.

Quello dell'**attenuazione** è quindi un buon metodo per agire, almeno un pò, sulle **luci**, per **regolare** la loro **potenza**.

Va detto infine che, per agire in **tempo reale** (a runtime, durante l'esecuzione del programma) sulla componente **d'attenuazione**, è necessario impostare, per le **luci** da modificare, le seguenti **capability**:

- ALLOW ATTENUATION WRITE ;
- ALLOW ATTENUATION READ .

BOUNDING LEAF

Come è noto, un **oggetto BoundingLeaf** specifica una zona *fisica* (nella scena 3D virtuale) d'influenza.

Rappresenta un'alternativa al **bound** nella definizione di '**zone di influenza**' ed è indipendente dalla **luce** cui è associato, essendo un oggetto dello **Scene Graph** a parte.

Ciò consente anche **effetti particolari**: **luce** (con relativa **attenuazione**, dipendente dalla distanza) e **zona di influenza** possono muoversi **separatamente** !

Le **classi** che permettono di implementare le **luci** mettono a disposizione il **metodo** *setInfluencingBoundingLeaf* per l'occasione.

Gli **oggetti BoundingLeaf** vanno aggiunti allo **SceneGraph**, possono essere condivisi da più **luci** (il metodo che permette di impostare le **BoundingLeaf** crea **collegamenti reference**) e, in generale, da più **elementi dello Scene Graph** (**Behavior**, ad esempio) contemporaneamente.

SCOPE

Per **limitare** l'influenza di una fonte di **luce** ad **oggetti raggruppati 'logicamente'** nello **Scene Graph** - e non in base alla loro **posizione fisica**, come nel caso dei **bound d'influenza** - è possibile utilizzare lo **Scope**.

Con lo **Scope** è possibile realizzare una **illuminazione 'selettiva'** degli oggetti presenti nella scena. A livello implementativo, uno **Scope** è un oggetto **Group**: gli **oggetti** da... influenzare, vengono aggiunti come **figli di un gruppo**, col **metodo addChild**.

Le **luci** mettono a disposizione **metodi** quali:

- `insertScope(Group scope, int index);`
- `setScope(Group scope, int index).`

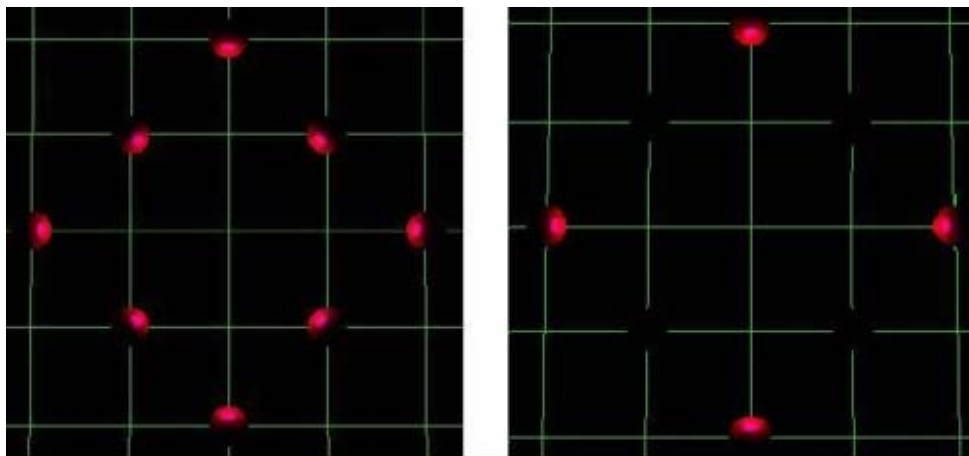
Il primo **metodo** permette di inserire più **gruppi Scope** (uno alla volta, comunque) nella **lista degli Scope di una sorgente luminosa**, associando a ciascuno di essi un indice intero; il secondo **metodo** permette, invece, di rimpiazzare uno **Scope** precedentemente associato alla **luce**, con un dato indice all'interno della lista di **Scope**, con uno nuovo.

Occorre molta cura nel progettare lo **Scene Graph** !

Ad esempio, bisogna evitare di creare più **BranchGroup** o **TransformGroup** solo perché è necessario inserire più **luci** con **scope** differenti: si tratta, infatti, di una scelta che porta a configurazioni non efficienti (e se, ad esempio, dovessimo ruotare tutti gli oggetti ?).

Nel file di esempio *PointLightConScope.java*, al **BranchGroup** principale vengono aggiunti due oggetti **Group** (gli **Scope**), contenenti delle sfere. E' presente una **fonte luminosa PointLight con BoundingSphere** ampia *ma* con un unico **Scope** associato.

L'immagine seguente mostra la **scena** con la **luce** attiva per entrambi i **gruppi Scope** e, nel secondo caso, solo per uno dei due. Le 'istruzioni' per provare le diverse **modalità** si trovano nel **file sorgente dell'esempio**.



LE OMBRE

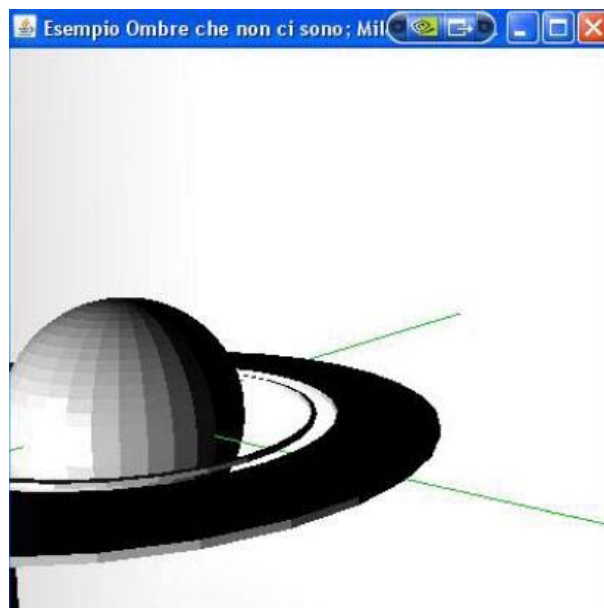
Gestire le **ombre** in **Java 3D** è abbastanza complicato e i risultati spesso non sono soddisfacenti.

Il tutto nasce dal fatto che il **motore 'nativo' di Java 3D** non mette a disposizione una funzionalità di **rendering delle ombre**.

Sta al programmatore trovare una **soluzione** per **calcolare** la posizione delle ombre e **rappresentarle**, cosa che è, in effetti, un'operazione particolarmente complessa da mettere in pratica.

L'immagine seguente è lo screenshot dell'esecuzione di un programma dove sono presenti, nell'universo virtuale di Java 3D, due mesh: il pianeta con l'anello e, dietro, un Box.

La mesh del pianeta le è sulla **traiettoria** del **fascio luminoso** di una fonte di **luce spot light**; si trova, inoltre, tra la **sorgente luminosa** e il box: dovrebbe quindi **proiettare un'ombra** sul box . . .



. . . cosa che, invece, **non** avviene !

Java 3D effettua quindi l'ombreggiatura all'interno di una mesh (il pianeta con l'anello), ma non tra più oggetti (il Box sullo sfondo).

Accenniamo qui a due **tecniche** di base per implementare manualmente le ombre: l'utilizzo di **poligoni-ombra** e l'utilizzo di **texture**.

I **poligoni-ombra** sono poligoni che devono essere '**disegnati**' sulle parti 'in ombra' degli **oggetti**.

Il **calcolo** sia delle coordinate che del colore da dare a tali poligoni (bisogna tenere conto del colore dell'oggetto da ricoprire e della luce) non è di facile attuazione ed è lasciato del tutto al **programmatore**; inoltre, il risultato spesso non è soddisfacente, anche per motivi legati a problemi di **prospettiva** e di **approssimazione** (quanti vertici utilizzare per 'disegnare' l'ombra di una sfera ?).

L'utilizzo delle **textures** per simulare le **ombre** può essere d'aiuto nel tentare di raggiungere un certo livello di **realismo**, visto che le ombre reali non sono mai 'nette'.

Le **texture** possono essere quindi usate sugli **oggetti** e sui **poligoni**, ma resta il problema del **calcolo** delle **proiezioni**.

Una **soluzione 'professionale'** per questo tipo di problema andrebbe quindi studiata appositamente, valutando caso per caso.

* * *
* * *

Capitolo 10

L'aspetto visivo degli oggetti

In questo capitolo vedremo come modificare la **resa** degli **oggetti visuali dello Scene Graph**.

Prenderemo in esame i **colori delle geometrie**, l'oggetto **Appearance** con i suoi **attributi**, l'oggetto **Material** e, infine, le **Textures**.

COLORI DELLE GEOMETRIE

Anche se “involontariamente”, abbiamo già modificato l'**aspetto visivo** degli oggetti trattando le **geometrie** delle **mesh**, colorandole, senza però far riferimento agli attributi di **Appearance**, l'oggetto che dovrebbe occuparsi proprio di questi argomenti.

Ad esempio, prendiamo in esame il **codice** utilizzato per creare la **griglia di riferimento** in alcuni degli **esempi** presentati nei capitoli precedenti:

```
VertexArray landGeom = new VertexArray(44, GeometryArray.COORDINATES |  
    GeometryArray.COLOR 3);  
Color3f c = new Color3f( 0.1f, 0.8f, 0.1f);  
for (int i=0; i<44; i++) landGeom.setColor(i,c); .
```

VertexArray discende da **Geometry** e, ancora più in alto, da **NodeComponent**. Non ha a che fare, quindi, con gli **Appearance** o i **Material**.

Le **Geometry** consentono dunque di specificare i **colori per i vertici**. Questo porta a vantaggi in termini di **performance** (non è necessario creare un **oggetto Appearance** per definire dei 'semplici' **colori**), ma anche di lettura del **codice**, con un numero minore di righe di codice da scrivere.

Le **Geometry** con **vertici** soggetti a **setColor** verranno **colorate** in questo modo:

- nel caso di **PointArray**, i **punti** saranno del **colore** impostato;
- nel caso di un **VertexArray**, le **linee** avranno un colore definito dai **colori** dei due **vertici** agli estremi del **segmento**;
- nel caso di un **TriangleArray** o di un **QuadArray**, che definiscono **poligoni**, la superficie della **geometria** sarà colorata **interpolando i colori** dei 3 o 4 **vertici** che la compongono.

E' importante sapere che *il colore della Geometry sovrascrive il colore dell'Appearance*.

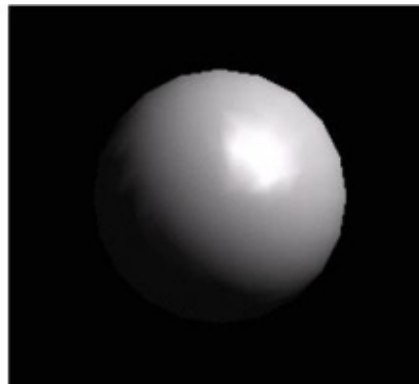
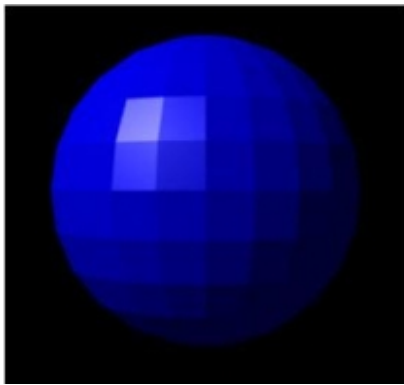
Abbiamo due file **d'esempio** riguardanti i **colori delle Geometry**: *'Esempio Colori Delle Geometrie 1'* ed *'Esempio Colori Delle Geometrie 2'*.

I MODELLI DI OMBREGGIATURA

In un oggetto visuale, l'**ombreggiatura** di un **vertice** è il risultato dell'ombreggiatura fornita da tutte le sorgenti di **luce** che colpiscono quel **vertice**.

A partire dai valori di un **vertice** vengono quindi 'ombreggiate' le **facce** dell'oggetto, a seconda di un **criterio** che è, appunto, il **modello di ombreggiatura**.

Avremo ad esempio **Gouraud**, che fornisce ottimi risultati visivi ma costosi in termini di calcolo, mentre il modello **FLAT** è più veloce ma le facce appaiono, appunto, 'piatte': il valore di ombreggiatura dell'intera faccia, in questo caso, è uguale a quello di un **vertice** o alla **media** dei valori di ombreggiatura dei **vertici** che compongono la **faccia** (nell'immagine seguente: a sinistra un esempio con **FLAT**, a destra con **GOURAUD**).



APPEARANCE E I SUOI ATTRIBUTI

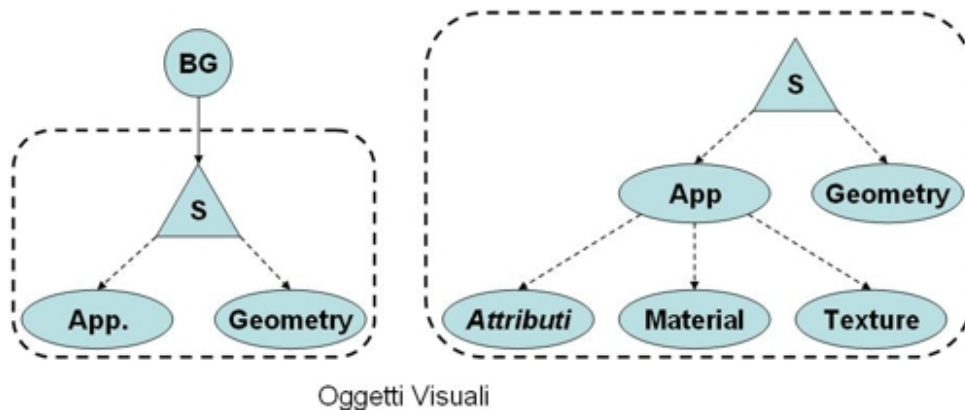
Un oggetto **Appearance** gestisce alcuni aspetti della **resa visiva** di uno **Shape3D** (da non confondere con le impostazioni di **visualizzazione della scena**, trattate dagli elementi del **View Branch Graph**).

Ad esempio, un oggetto **Appearance** si occupa di:

- **attributi di rendering** di punti, linee, colori;
- **materiali** (come si comporta un oggetto in presenza di luci);
- **texture** e attributi delle texture.

Un **Appearance** non è figlio di uno **Shape3D**, ma è legato ad uno o più di essi mediante collegamenti **Reference** (in particolare, è **Appearance** ad essere **referenziato**, in quanto **NodeComponent**).

Inoltre, per definire l'uso di **Appearance** più elaborate (con modello di illuminazione, ombreggiatura o con textures), bisogna utilizzare altri **oggetti** (rispettivamente, **Material** e **Texture**), che verranno trattati in un secondo momento.



Il **costruttore** di **Appearance** é uno solo:

```
Appearance() ; .
```

Esso crea un **oggetto Appearance** con valori di **default** per i vari campi interni, che sono:

- PointAttributes;
- LineAttributes;
- PolygonAttributes;
- ColoringAttributes;
- TransparencyAttributes.

Ad eccezione di **RenderingAttributes**, che gestisce alcuni aspetti del **rendering** delle primitive e che, di fatto, non viene modificato quasi mai, li prenderemo tutti in esame.

ATTRIBUTI DI APPEARANCE – POINT ATTRIBUTES

Il metodo *setPointAttributes(PointAttributes)* di **Appearance** ci permette di gestire il rendering di un **punto**, ad esempio un **vertice**.

I costruttori di **PointAttributes** sono:

- `PointAttributes();`
- `PointAttributes( float pointSize, boolean pointAntialiasing);` .

PointAttributes() imposta **pointSize** a 1 e **pointAntialiasing** a false.

E' possibile impostare **pointSize** e **pointAntialiasing** (che permette di visualizzare correttamente, e non come quadrati o rombi, punti di dimensioni maggiori di 1) anche in seguito, con **metodi** dedicati:

- `setPointSize(float);`
- `setPointAntialiasingEnable(boolean);` .

Esempio: '*EsempioPointAttributes*'.

ATTRIBUTI DI APPEARANCE – LINE ATTRIBUTES

Esistono tre parametri per modificare la visualizzazione di una **linea**: il **pattern** (continua, tratteggiata, ...), lo **spessore** e l'**Antialiasing** ('anti scalettatura').

I costruttori di tale oggetto sono:

- `LineAttributes();`
- `LineAttributes( float width, int pattern, boolean pointAA);` .

I valori di default impostati dal costruttore *LineAttributes()* sono: 1, PATTERN_SOLID, false.

Il campo **pattern** è un intero, con valori possibili:

PATTERN SOLID;
PATTERN DASH;
PATTERN DOT;
PATTERN DASH DOT;
PATTERN USER DEFINED.

Esempio: '*EsempioLineAttributes*'.

ATTRIBUTI DI APPEARANCE – POLYGON ATTRIBUTES

PolygonAttributes gestisce il **rendering** dei **poligoni**.

Il **costruttore** principale è:

```
PolygonAttributes();
```

Sono poi presenti numerosi **metodi** per cambiare alcune impostazioni di **rendering**; tra questi, *setPolygonMode* e *setCullFace*.

Di default, un poligono è renderizzato **pieno** (*POLYGON FILL*), ma con *setPolygonMode(int)* (o da 3 dei 4 costruttori) è possibile passare a **wireframe** (*POLYGON LINE*) o solo **vertici** (*POLYGON POINT*).

Il metodo *setCullFace(int)* imposta il criterio di **culling** (quali facce dei poligoni non renderizzare), in base a *CULL NONE*, *CULL BACK* (che poi è il valore di default) e *CULL FRONT*.

Esempio: *'Esempio PolygonAttributes'*.

ATTRIBUTI DI APPEARANCE – COLORING ATTRIBUTES

ColoringAttributes gestisce una colorazione 'di base' per le varie **primitive geometriche**; 'di base' in quanto viene ignorato se sono state attivate delle **luci**, per cui le interazioni con le **fonti luminose** non vengono prese in considerazione.

Costruttori:

- `ColoringAttributes;`
- `ColoringAttributes( Color3f color, int shadeModel);`
- `ColoringAttributes( float red, float green, float blue, int shadeModel);`

I **metodi** *setColor([argomenti])* permettono di impostare un **colore** per la **primitiva**, mediante argomenti di diversa natura.

Di default, gli **oggetti** avranno **colore bianco** (1, 1, 1).

setShadeModel(int) imposta il **modello di ombreggiatura** (interpolazione tra i colori) per la primitiva.

La scelta è tra (significati ovvi):

- FASTEST;
- NICEST;
- SHADE FLAT;
- SHADE GOURAUD (default).

Esempio: *'Esempio ColoringAttributes'*.

ATTRIBUTI DI APPEARANCE – TRANSPARENCY ATTRIBUTES

TransparencyAttributes gestisce la **trasparenza** degli oggetti.

Il valore di **trasparenza** viene espresso con un **valore float** compreso tra **0.0** (oggetto **opaco**) e **1.0** (oggetto completamente **trasparente**).

Costruttori principali:

- `TransparencyAttributes();`
- `TransparencyAttributes(int tMode, float tVal);` .

Il **metodo** *setTransparencyMode(int)* permette di definire come rendere la **trasparenza**; la scelta è tra:

- FASTEST;
- NICEST;
- SCREEN DOOR;
- BLENDED;
- NONE.

Esempio: *'Esempio TransparencyAttributes'*.

UTILIZZARE PIU' TECNICHE: UN ESEMPIO

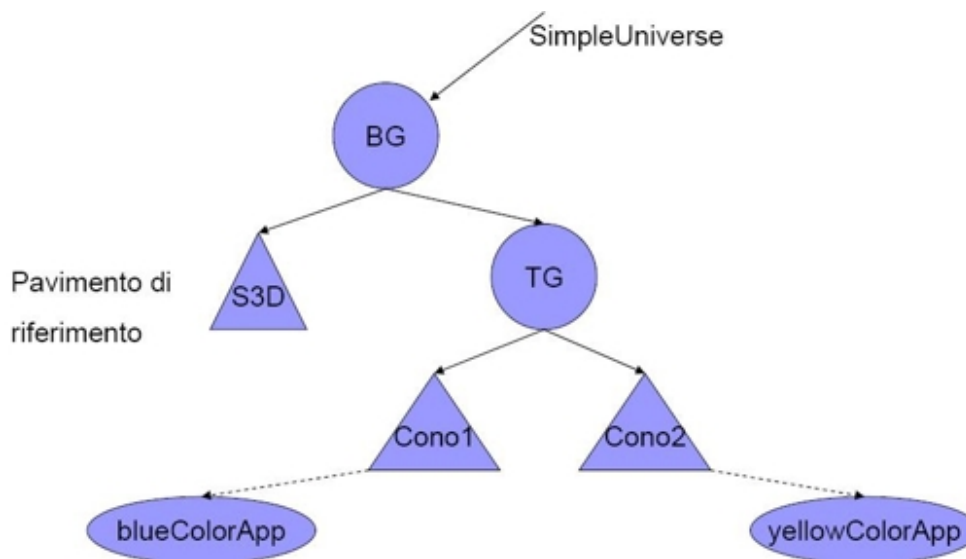
Gli attributi di **Appearance** sono abbastanza **selettivi** riguardo il tipo di **geometria** da considerare al momento del **rendering**; ad esempio, **PointAppearance** ha effetti sul rendering dei **PointArray**, ma nessun effetto sui 'puntini' mostrati a video nell'esempio **LineAppearance**, dove le **linee** venivano renderizzate come tratteggiate.

Come fare allora per **renderizzare** un oggetto con **più attributi di rendering** (ad esempio: un cono con un colore per le facce e i segmenti che lo compongono, la struttura, 'in evidenza', con un colore diverso) ?

Se non si può far ricorso a **textures** già pronte per scopi simili, per dare **due aspetti** (contemporaneamente) ad un oggetto bisogna **'duplicare'** una **Shape3D** (che possono condividere la stessa **Geometry**, anzi dovrebbero) ed assegnare una **Appearance** alla prima e l'altra **Appearance** alla seconda.

Le due **Leaf** andranno aggiunte allo stesso **Transform Group padre**, in modo tale da rendere consistenti le **trasformazioni** (applicando una rotazione ruoteranno entrambe, lasciando l'aspetto consistente; idem per le **traslazioni** e i **ridimensionamenti**).

Un **esempio** di quanto detto lo si può vedere in **'BordiInRilievo'**, che implementa la configurazione mostrata nella figura seguente.



ALTERNATE APPEARANCE

L'oggetto **AlternateAppearance** permette di definire, come suggerisce il nome, più **Appearance** 'alternativi' (o l'uno o l'altro, non insieme) per una **mesh**.

Dal punto di vista dello **Scene Graph**, un **AlternateAppearance** non è un **NodeComponent** (come **Appearance**), ma una vera e propria **Leaf**.

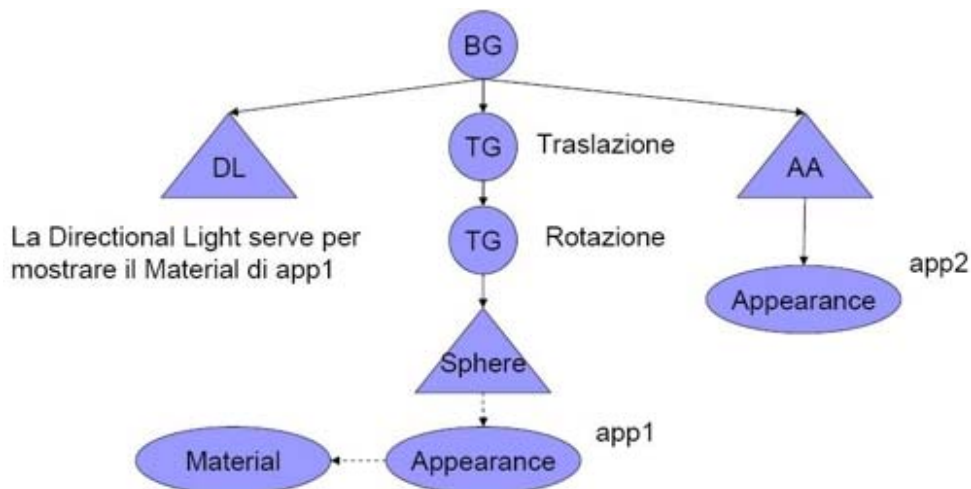
Costruttori:

- `AlternateAppearance();`
- `AlternateAppearance(Appearance a);` .

E' necessario impostare, poi, la **BoundingRegion** o la **BoundingLeaf** d'influenza.

Il funzionamento di **AA** è il seguente: ogni oggetto ha un suo **Appearance**, ma se collegato ad un **AA** e posizionato dentro il **bound d'azione** dell'**AA**, il suo **Appearance** 'nativo' verrà **sovrascritto** da quello dell'**AA**.

Esempio: '*Esempio AlternateAppearance*'; il **Content Branch Graph** di tale esempio è descritto nella figura seguente.



Oltre al **bound**, è possibile impostare lo **scope di validità di un AA**, il che significa che anche se un oggetto entra nel **bound d'azione** dell'**AA**, se non è discendente di un **Gruppo** inserito nella **lista dei gruppi di scope di AA**, non subirà l'effetto di tale oggetto.

Di default, la **lista di scope di AA** è vuota, ed in tal caso lo **scope di AA** è tutto lo **Scene Graph**, per cui tutti gli oggetti della scena presenti nel **bound d'azione** dell'oggetto ne subiranno l'effetto.

I **metodi** che permettono di impostare, rimuovere o recuperare un **gruppo nella/dalla lista degli scope di AA** sono diversi; tra questi, abbiamo ad esempio (significati ovvi):

- `addScope(Group g);`
- `numScopes();`
- `removeAllScopes();`
- `getAllScopes();`, che restituisce un oggetto di tipo **Enumeration**.

Un **esempio** di 'selezione logica' dell'area di influenza di un **AA** mediante **bound e scope** lo si ha nel file *'Esempio AlternateAppearanceApp2'*, simile ad *'Esempio AlternateAppearance'*, con la differenza che in questo caso sono presenti due sfere in movimento, che condividono **la stessa Appearance** ma appartenenti a **TG** diversi; l'**AA** ha, nella **lista dei gruppi di scope**, il **TG** antenato di una sola delle sfere, l'altra è **esclusa** (anche se passa nella **BoundingSphere** d'azione dell'**AA**, non ne subirà gli effetti).

Il lettore provi a **commentare l'istruzione** `aa.addScope(sfera1)`, ricompilare ed eseguire il file d'esempio per impostare, come **scope dell'AA**, tutto lo **Scene Graph**.

NOTA: se le **regioni di influenza** di più **AA** si **intersecano** (overlapping), la scelta di quale **AA** utilizzare è fatta da **Java 3D**, con un criterio definito dall'implementazione in uso.

In generale, comunque, verrà scelto l'**AA** più **'vicino'** all'oggetto.

MATERIAL

A differenza di **ColoringAttributes**, utilizzato solo in assenza di fonti luminose, **Material** determina il **colore di un oggetto** quando tale oggetto è **illuminato**.

Il **colore** (eventualmente) impostato con *setColor* di **Geometry** sovrascriverà, comunque, il colore impostato con **Material**.

I **costruttori** di **Material** sono:

- `Material()` ;
- `Material (Color3f ambientColor, Color3f emissiveColor, Color3f diffuseColor, Color3f specularColor, float shininess);` .

Ecco descritti i singoli campi:

- **Ambient color** : il colore della luce ambientale (se presente) riflessa sull'oggetto, espresso mediante tre valori float nell'intervallo [0, 1]. Il valore di default è (0.2, 0.2, 0.2).
- **Diffuse color** : colore RGB proprio del materiale illuminato da luce bianca, di default (1, 1, 1).
- **Specular color** : colore speculare (specular highlights) del material, di default (1, 1, 1).
- **Emissive color** : *se* il materiale emette luce, tale luce è di questo colore (il valore di default è 0, 0, 0).
- **Shininess** : la lucentezza del materiale, che determina la grandezza della regione di Specular Color, definita da un valore che va da 1.0 (nulla) a 128.0 (massima). Il valore di default è 64.

Esempio: *'EsempioMaterial'*.

LUCI, ASPETTO E NORMALI DELLE GEOMETRIE

Spesso, quando le **geometrie** vengono definite **vertice a vertice**, possono verificarsi problemi di visualizzazione: i **colori**, e in generale l'aspetto dell'oggetto, sembrano non comportarsi come dovrebbero.

Nella maggior parte dei casi ciò è dovuto alle **Normali dei vertici della geometria**, dunque potrebbe divenire necessario **impostarle manualmente**.

GeometryArray mette a disposizione, e rende disponibili alle sue sottoclassi, vari **metodi** per lavorare con le **Normali**; tra questi, abbiamo *setNormals(int index, Vector3f[] normals)*.

Tale **metodo** recupera un **array di vettori**, ciascuno dei quali è una **Normale** (definita da 3 valori float), e associa ciascun **elemento** di questo **vettore** ad un **vertice** della geometria che sta invocando il **metodo**.

Il parametro *index* indica a partire da quale **indice** (ossia: da quale **vertice**) *della geometria* effettuare la copia.

Ad **esempio**, impostando come indice 10 e passando un array di vettori di 30 elementi, le 30 normali verranno associate ai vertici di indice compreso tra 10 e 40 (il 10 incluso, il 40 no).

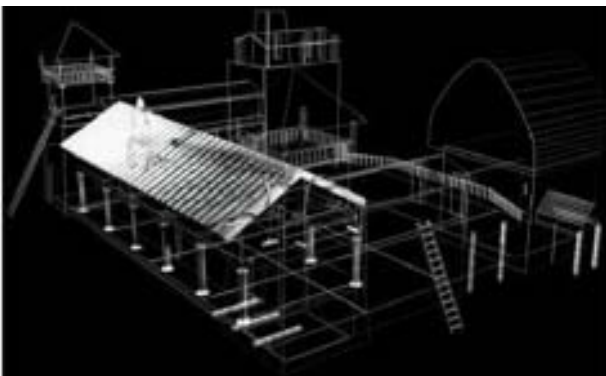
Si noti che in questo modo è possibile **definire esplicitamente** quale sarà il **verso** della '**normale positiva**' e, dal momento che ne stiamo impostando anche la **direzione** (ogni **normale** è un **Vector3f**), ciò avrà ripercussioni anche sulla **resa visiva degli oggetti**, in presenza di **fonti luminose** nella scena e di un **Material** 'sottostante' impostato per l'oggetto da visualizzare.

LE TEXTURES E IL TEXTURE MAPPING

Le **texture**, in **Computer Grafica**, sono **immagini** applicate ('mappate') su **geometrie**.

Esse permettono di aggiungere **realismo** ad una scena evitando di dover utilizzare un gran numero di **primitive geometriche**, altrimenti necessarie per ottenere lo stesso grado di **dettaglio**.

L'applicare le **texture** ad un **oggetto** è detto texturing o *texture mapping*.



Il texture mapping segue pochi semplici passi, ai quali corrispondono poche classi e righe di codice:

1. **caricare una texture** (un'immagine, in formato leggibile per Java 3D e di dimensioni 'corrette', come vedremo a breve);
2. impostare la **texture nell'Appearance**;
3. **posizionare correttamente la texture**, specificandone le coordinate.

I primi due punti sono molto semplici da implementare, mentre per il terzo è richiesto un pò più di tempo, in base all'oggetto da **mappare**.

CARICARE UNA TEXTURE

Una volta scelta l'**immagine** da utilizzare, si procede al **caricarla** nel programma.

Dal punto di vista strettamente informatico, l'importante è che tali immagini siano:

- in un **formato** leggibile da **TextureLoader**, come **jpg** e **gif**;
- di 'dimensioni corrette', ossia con le **misure di larghezza ed altezza potenze di 2**.

Il **package** da importare, necessario per l'utilizzo del **TextureLoader**, è

com.sun.j3d.utils.image.TextureLoader.

Gli oggetti (informatici) necessari sono quindi un **TextureLoader**, una **ImageComponent2D** e un oggetto **Texture2D**, da utilizzare come mostrato di seguito:

```
TextureLoader TL = new TextureLoader('[path]', this);  
ImageComponent2D immagine = TL.getImage();  
Texture2D texture = new Texture2D (Texture (int mipMapMode, int  
    format, int width, int height));  
texture.setImage(0, immagine); .
```

Bisogna porre particolare attenzione al **costruttore di Texture2D**; in particolare, evitare di utilizzare il **costruttore senza parametri** perchè il più delle volte genererà un errore a runtime.

Parametri 'tipici' del costruttore a quattro parametri sono: *Texture.BASE LEVEL*, *Texture.RGBA*, *immagine.getWidth()*, *immagine.getHeight()*.

IMPOSTARE UNA TEXTURE IN UN APPEARANCE

L'oggetto **Appearance** mette a disposizione il **metodo *setTexture(Texture2D)*** per impostare, appunto, una **Texture** in un **Appearance**.

Dopo aver creato un **Appearance** (chiamato, ad esempio, 'appl') e una **Texture2D** (chiamata, ad esempio, 'tex1'), sarà quindi sufficiente scrivere:

```
appl.setTexture(tex1);
```

per agganciare il nostro oggetto **Texture2D (NodeComponent)** allo **Scene Graph**, facendolo **referenziare** da uno o più **NodeComponent Appearance**.

IMPOSTARE LE COORDINATE DI UNA TEXTURE

Esistono due modi per specificare le **coordinate della texture**:

- lasciare che sia **Java3D** ad occuparsene, con **TexCoordGeneration**;
- impostare le **coordinate** delle texture **manualmente**, via codice.

Il primo **metodo** è molto semplice ma lascia meno 'libertà' rispetto al secondo che, di contro, è un pò più complesso e, spesso, si rivela molto noioso.

Li prenderemo entrambi in esame.

TEXCOORDGENERATION

TexCoordGeneration ha i seguenti costruttori:

- `TexCoordGeneration();`
- `TexCoordGeneration(int genMode, int format);`
- `TexCoordGeneration(int genMode, int format, Vector4f planeS);`
- `TexCoordGeneration(int genMode, int format, Vector4f planeS, Vector4f planeT);`
- `TexCoordGeneration(int genMode, int format, Vector4f planeS, Vector4f planeT, Vector4f planeR);`
- `TexCoordGeneration(int genMode, int format, Vector4f planeS, Vector4f planeT, Vector4f planeR, Vector4f planeQ);` .

Il **parametro 'mode'** è un campo **intero**, che può assumere i seguenti valori:

- `OBJECT LINEAR;`
- `EYE LINEAR;`
- `SPHERE MAP;`
- `NORMAL MAP`
- `REFLECTION MAP.`

Il **parametro 'format'** è un **intero** che può assumere i seguenti valori:

- `TEXTURE COORDINATE 2` (il più utilizzato);
- `TEXTURE COORDINATE 3;`
- `TEXTURE COORDINATE 4.`

Gli altri **parametri** si riferiscono ai **coefficienti** da utilizzare per definire **quattro piani** che **Java 3D** utilizzerà per **calcolare le coordinate** se il 'mode' è **OBJECT LINEAR** o **EYE LINEAR**.

Generalmente, vengono mantenuti i **valori di default** definiti da **Java3D**.

Il più delle volte, viene utilizzato il **primo costruttore** (d'altro canto, si fa ricorso a **TexCoordGeneration** proprio per evitare di dover fare del 'lavoro' !), che imposta i seguenti valori per i vari campi precedentemente esposti:

mode : OBJECT LINEAR;

format : TEXTURE COORDINATE 2;

S : (1, 0, 0, 0);

T : (0, 1, 0, 0);

R : (0, 0, 0, 0);

Q : (0, 0, 0, 0).

Una volta creato il nostro **TexCoordGeneration**, bisogna associare tale **oggetto** ad un **Appearance**; per tale scopo, **Appearance** mette a disposizione il **metodo**:

```
setTexCoordGeneration(TexCoordGeneration tcg); .
```

IMPOSTARE LE COORDINATE DELLE TEXTURES MANUALMENTE

Le texture possono essere 'imprese' sulle **geometrie** andando ad impostare **manualmente** i singoli **vertici**.

Creiamo ad esempio una **geometria QuadArray** impostando vari **vertici** (punti).

```
QuadArray miaGeometria = new QuadArray(4, QuadArray.COORDINATES  
| QuadArray.TEXTURE_COORDINATE_2);  
Point3f punto = new Point3f();  
punto.set(-1, 1, 0);  
miaGeometria.setCoordinate(0, punto);1  
punto.set(-1, -1, 0);  
miaGeometria.setCoordinate(1, punto);  
... ..
```

¹ Chiamiamo questo valore “vertice destinazione”.

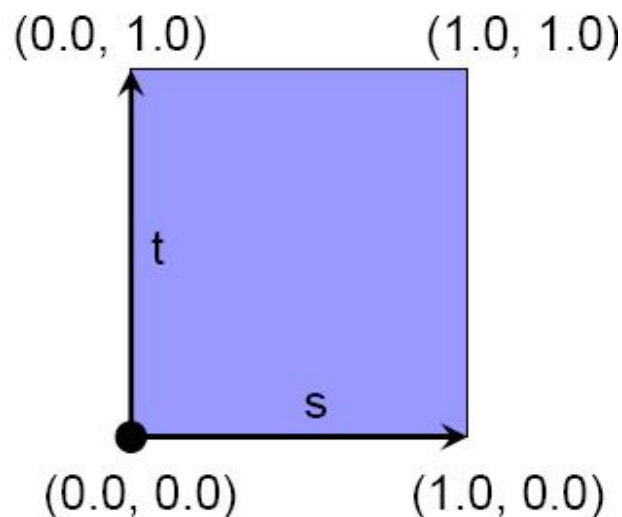
A questo punto si crea un oggetto **TexCoord2f** e vi si associano le **coordinate** *s* e *t* (rispettivamente: orizzontale e verticale) di una **texture**, un pò come si è fatto con la **geometria**, al passo precedente, associando ad ogni **Point3f** (in realtà un unico oggetto, riusato) le **coordinate**, come mostra la figura seguente.

```
□ TexCoord2f coordinata = new TexCoord2f();  
□ coordinata.set(0.0f,1.0f); // questa è una  
  coordinata della texture !!!  
□ miaGeometria.setTextureCoordinate(0, 0, coordinata);
```

Indice della texture per questa geometria (Java3D permette di definire più texture per una Geometria; in questo caso, tale valore sarà 0 per tutti i 4 punti).

Indice del "vertice destinazione"

NOTA IMPORTANTE: le **coordinate** *s* e *t* di una texture in **TexCoord2f** hanno valori float che variano **nell'intervallo** **[0.0, 1.0]**, indipendentemente dalla larghezza e dall'altezza, in pixel, dell'immagine usata come **texture**.



ESEMPI CON LE TEXTURES

Esempi riguardanti quanto visto finora sulla **textures** in **Java3D**:

- *EempioTexture1* mostra come caricare una Texture ed applicarla ad una Appearance. Non c'è gestione delle coordinate della Texture.
- *EempioTexture2* mostra come applicare una Texture ad un oggetto affidandosi ad un TexCoordGeneration per impostare le coordinate della Texture.
- *EempioTexture3* mostra vari esempi, su varie forme geometriche e diverse modalità di applicazione, di utilizzo del TexCoordGeneration.
- *EempioTexture4* mostra come mappare una Texture usando TexCoord2f.

NOTA – Per poter utilizzare correttamente questi esempi, è necessario creare delle immagini e posizionarle nella stessa cartella del file .class generato o modificare il path all'interno dell'esempio prima di compilarlo; in quest'ultimo caso, sarà possibile cambiare anche il nome del file immagine da utilizzare, che nella versione originale degli esempi è:

- “Texture.jpg” per EempioTexture1;
- “Texture.jpg” per EempioTexture2;
- “Terra.jpg” per EempioTexture3;
- “Sistina.jpg” per EempioTexture4.

BOUNDARY MODE

Una **Texture2D** mette a disposizione i **metodi** (ereditati da **Texture**) *setBoundaryModeT (int)* e *setBoundaryModeS (int)* , per impostare il **boundary mode** (comportamento ai **bordi**), rispettivamente per la **coordinata T** e per la **coordinata S** di una **texture**.

Il valore **intero** passato come **argomento** è una **costante**, scelta tra:

- CLAMP;
- WRAP;
- CLAMP TO EDGE;
- CLAMP TO BOUNDARY.

Ma cosa si intende per **'impostare il comportamento ai bordi'** ?

Capita, a volte, che una **texture** abbia dimensioni inferiori alla **superficie da rivestire**; in questo caso, è possibile scegliere tra due **alternative**:

- CLAMPING: la texture viene estesa fino ad occupare la superficie della geometria.
- WRAPPING: la texture viene 'ripetuta', come quando si effettua un tiling.

Le **combinazioni** di **clamping** e **wrapping** per le **coordinate t** ed **s** sono mostrate 'all'opera' nel file d'esempio **'Esempio BoundaryMode'** (per utilizzarlo correttamente, inserire un file immagine con nome “Wood.jpg” nella stessa cartella del file .class, oppure modificare il path nel codice e compilare).

PIXEL E TEXEL

Il **pixel** è, come è noto, l'elemento puntiforme che compone l'immagine sullo schermo, dunque che compone la **proiezione dell'universo 3D** sulla **Canvas**.

Il **texel** è, invece, l'unità fondamentale di una **texture** (proprio come il **pixel** lo è del dispositivo di visualizzazione).

Ovviamente, quando si deve rappresentare una **texture** su schermo, non tutti i **texel** possono essere convertiti in un eguale numero di **pixel** sul display.

Può capitare infatti che la **texture** possa essere rimpicciolita, ed in tal caso un **pixel** dovrà coprire più **texel**, o al contrario possa essere così grande da rendere necessario l'assegnare a più **pixel** un elemento **texel**.

I **metodi** per gestire queste situazioni sono essenzialmente due: **point sampling** (selezione di un pixel per tutti i texel, o di un texel per tanti pixel) o **l'interpolazione** (operazione matematica sui gruppi, di pixel o di texel a seconda dei casi).

La scelta non è delegata al programmatore: è **Java 3D** che **sceglie**, in base a **performance** e **qualità** (intuitivamente, l'interpolazione produce risultati più interessanti, ma richiede maggiore potenza di calcolo).

* * *
* * *

Capitolo 11

Animazioni

A parte le **animazioni 'implicite'** generate da **Java3D** quando ci si sposta **nell'universo virtuale** o, sempre mediante **l'interazione**, viene applicata una **trasformazione** ad un **oggetto**, sono possibili altri due tipi di **animazione**, che coinvolgono gli **oggetti dell'universo 3D** in base a:

- **posizione del punto di osservazione**: se l'utente si sposta, l'oggetto si sposta o cambia la sua modalità di rappresentazione a video.
- **tempo**, ossia ad ogni istante di tempo è associata una configurazione (**animazioni time-based**).

Un oggetto che **si orienta verso di noi** quando ci spostiamo appartiene al primo gruppo, un oggetto **animato 'di per sè'** (un faro, un orologio) al secondo.

In questo capitolo verranno presi in considerazione entrambi i casi; in particolare, per le **animazioni dipendenti dal punto di vista dell'osservatore** parleremo di:

- **Billboard**;
- **OrientedShape3D**;

- **LoD;**

mentre, per quanto riguarda le animazioni **Time-Based**, verranno mostrati gli **Interpolatori** messi a disposizione da **Java 3D**.

Vedremo poi come **generare delle animazioni 'a comando'**, calcolando cioè le traiettorie ed agganciando l'animazione allo **Scene Graph** durante l'esecuzione, e le animazioni **Morph**, che riguardano i **vertici** di una **mesh**, non necessariamente tutto l'oggetto.

ANIMAZIONI DIPENDENTI DAL PUNTO DI VISTA DELL'OSSERVATORE

Le animazioni prodotte in maniera automatica da **Java3D** quando muta la **posizione dell'osservatore** possono essere prodotte mediante **Billboard**, **OrientedShape3D** o **LoD** (Level of Detail).

Billboard permette di **orientare un oggetto** verso l'osservatore, in modo da renderlo sempre visibile. Questo si rivela molto utile con, ad esempio, le scritte degli oggetti **Text**, che così risultano **sempre leggibili**.

OrientedShape fa, sostanzialmente, quello che fa **Billboard**, ma in più gestisce le **viste multiple**.

Questi due oggetti vengono implementati in maniera diversa anche a livello di **costruttori** e di **Scene Graph**, come verrà mostrato a breve.

Le animazioni **LoD** (Level of Detail, livello di dettaglio) dipendono invece dalla **distanza** che intercorre **tra l'osservatore e l'oggetto**.

LoD permette di modificare le **impostazioni di resa degli oggetti**, cambiando la loro **risoluzione**.

Questo strumento, se usato bene, non dà luogo a particolari problemi in termini di resa visiva e permette un miglioramento in termini di performances.

BILLBOARD

Una **Billboard** modifica l'orientamento di un oggetto in maniera tale che l'asse **z** **positivo** dello stesso **'punti'** sempre verso l'osservatore, ossia 'fuori dallo schermo'.

I **costruttori** principali sono:

- `Billboard()` ;
- `Billboard(TransformGroup tg)` ; .

La **Billboard** fa quindi uso di un **Transform Group**, che *non* deve essere quello utilizzato per specificare la **posizione**, nello **spazio 3D**, dell'oggetto da orientare.

E' possibile vedere un esempio di creazione ed utilizzo di **Billboard** con *'Esempio Billboard'*.

Di seguito viene descritto come implementare un'animazione **Billboard**.

Per inserire una **Billboard**, bisogna creare un **TransformGroup** e un **bound**; il **TransformGroup** deve avere le **capability** di scrittura a *true* e *non* deve essere usato per altri scopi (es.: traslazione), ma **solo per la Billboard**.

```
TransformGroup tg2 = new TransformGroup();  
tg2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);  
tg2.addChild([inserire qui un oggetto visuale]);  
  
Billboard miaBillboard = new Billboard(tg2);  
BoundingSphere bounds = new BoundingSphere();  
miaBillboard.setSchedulingBounds(bounds);
```

Se si desidera applicare **ulteriori trasformazioni** all'oggetto legato alla **Billboard**, creare un nuovo **Transform Group 'a monte'**, impostare la trasformazione voluta ed associargli, **come figlio**, il **Transform Group** che gestisce la **Billboard**.

```
BranchGroup objRootInterno = new BranchGroup();  
Transform3D t3d = new Transform3D();  
t3d.setTranslation(new Vector3f(0.0f, 0.0f, -5.0f));  
TransformGroup tg1 = new TransformGroup(t3d);
```

```
// Definire qui la Billboard con il suo TG, detto "tg2"  
tg1.addChild(tg2);
```

Per **agganciare la Billboard e l'oggetto** ad essa legato allo **Scene Graph**, ricordarsi di aggiungere sia la **Billboard** che il **Transform Group** radice del sottoalbero così generato (padre, nonno o altro antenato dell'oggetto **Billboard**, a seconda dei casi).

```
objRoot.addChild(tg1);  
objRoot.addChild(miaBillboard);
```

E' disponibile anche un secondo file d'esempio riguardante gli oggetti **Billboard**, **'EsempioBillboard2'**, simile al primo ma che ha, come **pivot** per l'orientamento di un **testo 2D**, un **punto**.

Tale configurazione è stata implementata aggiungendo la seguente riga di codice:

```
miaBillboard.setAlignmentMode (Billboard.ROTATE ABOUT POINT);
```

La scelta di **quale tipo di pivot** utilizzare dipende dal tipo di progetto da realizzare, ma in generale la **rotazione intorno ad un punto** porta a risultati visivi migliori.

ORIENTEDSHAPE3D

Un oggetto **OrientedShape3D** non è una **trasformazione** ma un vero e proprio **oggetto visuale**, che **eredita da Shape3D**.

Nel definire un **OrientedShape3D** bisogna quindi creare a parte una **geometria**, da collegare (via **Reference**, con il metodo **addGeometry(Geometry)**) all'oggetto **OrientedShape3D**.

E' possibile aggiungere **più geometrie**, purchè siano tutte appartenenti alla stessa **'classe di equivalenza'** (punti, linee, poligoni).

L'**OrientedShape3D** può essere quindi collegato, mediante una **relazione Padre-Figlio**, ad un **Transform Group** o ad un **Branch Group**.

La figura seguente riporta la porzione di codice necessaria per **aggiungere un testo orientato e traslato mediante `OrientedShape3D`**.

```
private BranchGroup insiemeOrientedShape3D()
{
    BranchGroup objRootInterno = new BranchGroup();
    Transform3D t3d = new Transform3D();
    t3d.setTranslation(new Vector3f(0.0f, 0.0f, -5.0f));
    TransformGroup tgl = new TransformGroup(t3d);
    Text2D testo2D = new Text2D("Ciao !", new Color3f(1.0f,
        1.0f, 0.0f), "sansserif", 100, Font.PLAIN);

    OrientedShape3D orientedShape3D = new
        OrientedShape3D();

    orientedShape3D.addGeometry(testo2D.getGeometry());
    orientedShape3D.setAppearance(testo2D.getAppearance());

    tgl.addChild(orientedShape3D);
    objRootInterno.addChild(tgl);

    return objRootInterno;
}
```

Tale codice fa parte del file d'esempio *'Esempio `OrientedShape3D`'*.

Anche per **`OrientedShape3D`**, così come si è visto in **`Billboard`**, è possibile impostare la modalità di **allineamento**: nessuna, intorno ad un asse o intorno ad un punto; in quest'ultimo caso, bisogna utilizzare la seguente istruzione:

```
setAlignmentMode (OrientedShape3D.ROTATE ABOUT POINT); .
```

A tal proposito, valgono le considerazioni fatte per la **rotazione intorno ad un punto** in **`Billboard`**.

ANIMAZIONI LOD

'LoD' sta per 'Level of Detail', o 'Livello di dettaglio'.

Come suggerisce il nome, un'animazione **LoD** tiene conto della **distanza** tra l'**osservatore** e l'**oggetto** da renderizzare per evitare 'sprechi' in termini di costo computazionale.

Si tratta di un'animazione **dipendente dal punto di vista dell'osservatore**, come quelle realizzate mediante **Billboard** o **OrientedShape3D**.

Quando, ad esempio, intravediamo una nave all'orizzonte, non riusciamo a coglierne i dettagli più minuti, ma se la nave si trova a 1 metro da noi riusciamo a vedere anche un bullone; se abbiamo una **mesh** '**complessa**' ma molto **lontana** dall'osservatore, perchè far calcolare al motore di **rendering** di **Java3D** ogni minima linea o punto del modello ?

Ecco quindi che entra in scena il '**Level of Detail**': ad ogni **mesh** possiamo associare **n modelli**, dal più dettagliato al più semplice, e associare a ciascun modello una **soglia**: dalla **distanza** x alla distanza $x + d$ utilizzeremo un modello, ma dalla distanza $x+d$ alla distanza $x+2d$ utilizzeremo la **versione meno complessa**, e così via.

La classe che ci consente di creare oggetti in grado di impostare le 'soglie' delle distanze per scegliere la **mesh** da renderizzare si chiama **DistanceLoD** ed è un classe che eredita da **Behavior**.

I **costruttori** sono:

- `DistanceLoD()` ;
- `DistanceLoD(float[] distances)` ;
- `DistanceLoD(float[] distances, Point3f position)` ; .

Il primo costruttore crea un **LoD** con le impostazioni di default: un solo valore di **soglia**, impostato a 0.0, con l'oggetto **LoD** posizionato **nell'origine**; si tratta, quindi, di un **oggetto DistanceLoD** praticamente **inutile**, buono solo per definire un **LoD** e modificarne i **parametri** in seguito.

Il secondo costruttore ci permette di specificare un **array di float**: ogni float è una **soglia di distanza** dell'oggetto **DistanceLoD** dall'osservatore.

Il terzo costruttore ci permette infine di specificare **tutti i parametri**: l'array di float delle distanze / soglia e la **posizione**, specificata con un **Point3f**, dell'oggetto **DistanceLoD**.

E' possibile notare come le varie **geometrie** da associare alle **soglie / distanze** non siano specificate in alcun modo, qui... come fare ?

E' necessario ricorrere ad un **oggetto particolare** di **Java3D**, da creare con la **classe Switch**.

Un oggetto **Switch** è un **Group Node**: può contenere uno o più **figli**, come una **lista** (difatti i **metodi**, che verranno elencai a breve, somigliano molto a quelli di una generica **LinkedList**).

Ecco i **metodi di Switch** per l'inserimento, la ricerca e la rimozione di nodi (nota: **Switch**, in realtà, **eredita** questi **metodi** dalla classe **Group**):

- `public void addChild(Node child);`
- `public java.util.Enumeration getAllChildren();`
- `public Node getChild(int index);`
- `public int indexOfChild(Node child);`
- `public void insertChild(Node child, int index);`
- `public int numChildren();`
- `public void removeAllChildren();`
- `public void removeChild(int index);`
- `public void removeChild(Node child);`
- `public void setChild(Node child, int index);` .

Il loro **significato / funzionamento** è intuitivo.

Impostando le relative **capabilities**, è possibile operare su questa '**lista**' a **runtime**.

Ai valori di **soglia / distanza** specificati nell'oggetto **DistanceLoD** verranno quindi associati degli **oggetti** presenti nello **Switch** target del **DistanceLoD**.

Riassumendo, ecco i **passi** da seguire:

1. creare un nodo **Switch** e popolarlo;
2. creare un oggetto **DistanceLoD**, specificando una lista di distanze / soglia;
3. impostare l'oggetto **Switch** come **target** del **DistanceLoD**, tramite il metodo ***addSwitch(Switch s)*** di **DistanceLoD**;
4. aggiungere il tutto (**DistanceLoD** e **Switch**) allo **Scene Graph**.

Bisogna ricordarsi, infine, di impostare le **capabilities** per lo **Switch** (lo **Switch** deve poter selezionare a **runtime** un oggetto dalla sua lista) e di creare uno **SchedulingBounds** per il **DistanceLoD**.

LoD è uno strumento potente, ma con una limitazione: **non supporta le viste multiple** (proprio come **Billboard**, con la differenza che per **Billboard** c'era un'alternativa, rappresentata da **OrientedShape3D**... qui no).

Esempio pratico: *'EsempioLoD'*.

ANIMAZIONI TIME-BASED

Per poter creare animazioni **time-based** in **Java3D** è necessario conoscere alcuni oggetti fondamentali:

TARGET

Con questo termine identificheremo tutti gli **oggetti** (principalmente **TG**, ma non solo) da sottoporre ad **animazione**.

Le **animazioni** riguarderanno la **posizione** (traslazione), **l'orientamento** (rotazione), il **ridimensionamento** (scaling), la **trasparenza** ed il **colore**.

ALPHA

Una specie di **orologio interno** dell'universo virtuale di **Java3D**.

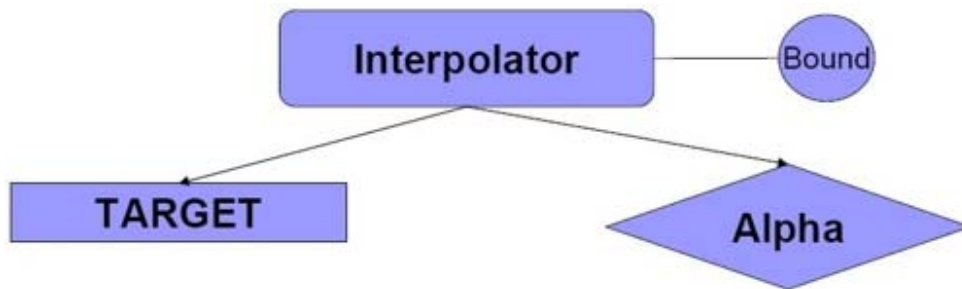
Verrà preso presto in esame.

INTERPOLATOR

Un **interpolatore** collega l'istante di **tempo** segnato da **Alpha** ad una **trasformazione** da applicare ad un **target**.

Deve avere un **bound** e va collegato allo **Scene Graph**.

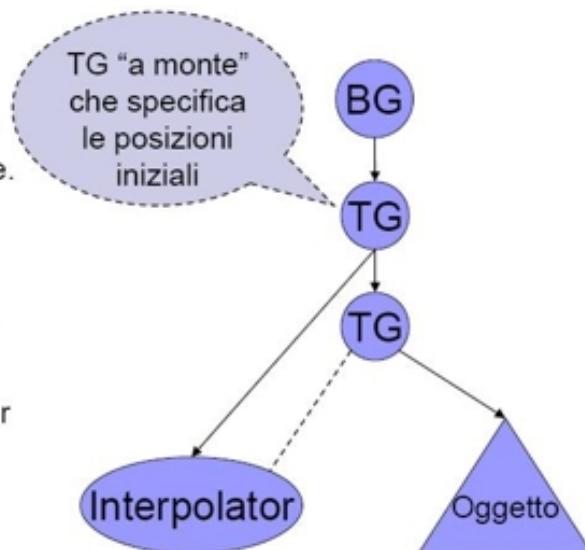
L'immagine seguente raffigura **come collegare gli elementi** appena presentati, con l'aggiunta del bound che definisce la zona d'influenza dell'Interpolator; attenzione: **non si tratta di un grafo** da inserire nello **Scene Graph**, ma di un semplice **diagramma delle classi** in gioco.



L'immagine seguente, invece, rappresenta **una possibile configurazione** da inserire nello **Scene Graph**, con specificati i passi da seguire per realizzarla.

Passi da seguire:

1. Creare il TARGET, figlio di un TG con le capability impostate.
2. Creare un Alpha, impostandone i parametri.
3. Creare un Interpolator, fornendogli Alpha e TARGET ed impostandone il BOUND.
4. Aggiungere TG ed Interpolator allo Scene Graph



Ci concentreremo quindi sull'oggetto **Alpha** per poi passare agli **Interpolator** messi a disposizione da **Java3D**.

ALPHA

Un oggetto **Alpha** fornisce un **valore tra 0 e 1** (estremi inclusi) basandosi sul **tempo**: è una funzione $f(t) = [0, 1]$.

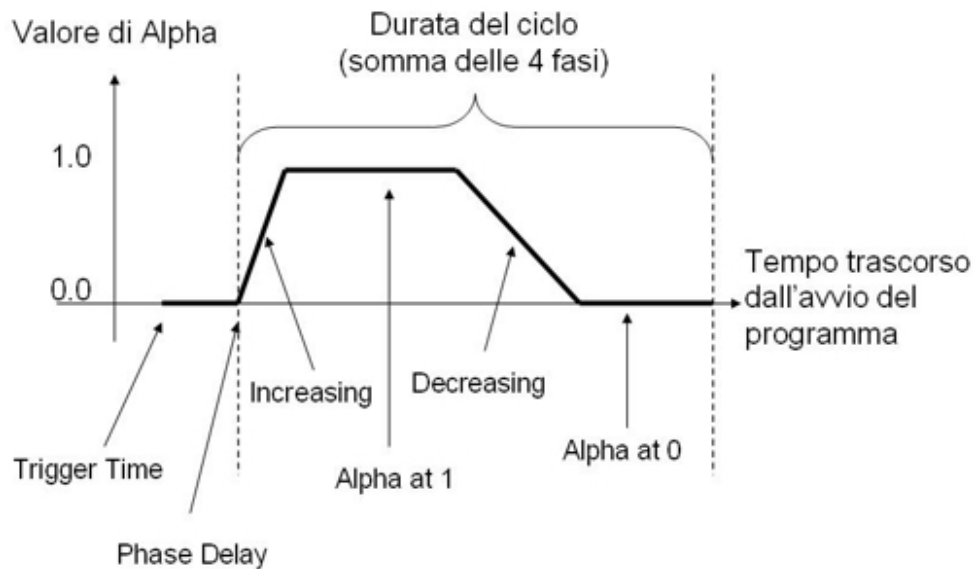
E' possibile inserire **più oggetti Alpha** in un **universo virtuale** per associarli a vari **interpolatori**.

Gli **Alpha**, se non specificato diversamente (vedremo in seguito come fare), partiranno tutti allo stesso istante.

Alpha ha una gran quantità di **parametri** che servono a definire come deve variare il valore restituito in **funzione del tempo**: durata di un ciclo, forma d'onda, ...

La **forma d'onda di un Alpha** è il grafico della funzione $f(t) = [0, 1]$.

Un esempio di **forma d'onda**, con indicate le zone che la compongono, è visibile nell'immagine seguente.



I **cicli** possono essere eseguiti una o più volte, specificando il **numero di esecuzioni** con il parametro **loopCount**, un intero (generalmente) positivo; utilizzando invece il valore **-1**, l'**Alpha** ciclerà **all'infinito**.

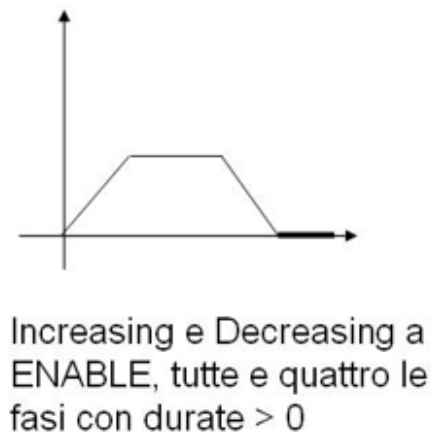
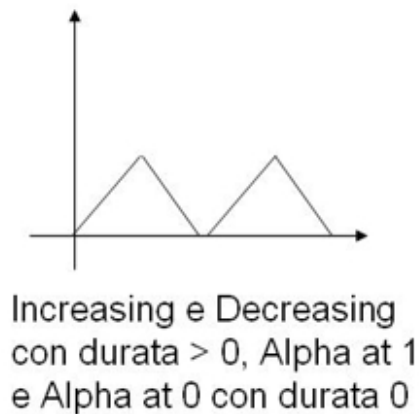
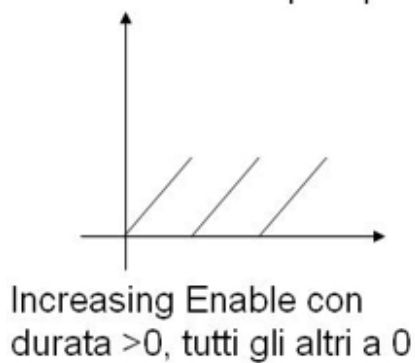
Il **phaseDelay**, un valore che **ritarda l'esecuzione del ciclo**, verrà applicato se presente **solo alla prima iterazione** del ciclo.

Come è possibile notare dalla figura, le **fasi del fronte d'onda** sono 4:

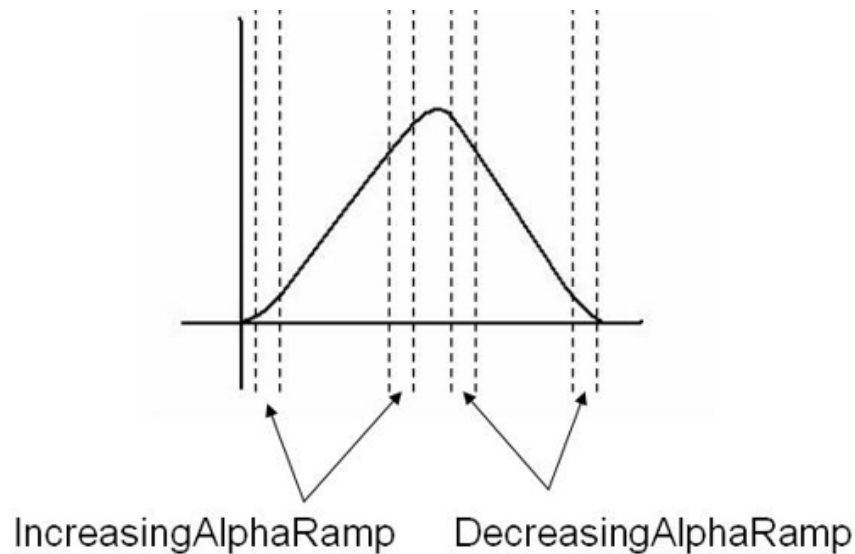
1. incremento;
2. valore di Alpha a 1;
3. decremento;
4. valore di Alpha a 0.

E' possibile comunque **scegliere** quali **fasi** della forma d'onda **implementare**, con **INCREASING_ENABLE**, che abilita la prima fase, **DECREASING_ENABLE**, che abilita la terza fase, e una **combinazione** delle due.

Esempi di possibili forme d'onda



Le **forme d'onda** presentate nelle immagini precedenti hanno **andamenti lineari**, tuttavia è possibile '**smussare**' tali forme al fine di ottenere **cambiamenti più gradual**i, necessari, ad esempio, per definire il moto di un pendolo, impostando opportunamente i valori dei **parametri Ramp**, corrispondenti a determinate **fasi della funzione d'onda**, come mostrato in figura.



Come detto precedentemente, tutti gli **Alpha** avranno lo stesso **START TIME** e tutti gli **Interpolator** saranno **sincronizzati** sullo **start time di sistema**; tuttavia, è possibile definire **ritardi** nell'avvio sfruttando i **parametri**:

- **TriggerTime**: il tempo che intercorre tra lo **0 del sistema** (all'avvio del programma) e l'inizio del primo ciclo di **Alpha**;
- **PhaseDelay**: ulteriore ritardo posto tra il **TriggerTime** e l'inizio vero e proprio del primo ciclo.

La **durata** dell'intero ciclo è data dalla somma dei **tempi** dedicati a ciascuna delle **4 fasi** del ciclo (incremento, 1, decremento, 0).

E' giunto il momento di presentare i costruttori e i metodi di Alpha.

Costruttori:

- Alpha(); valori di default: increasing enable, loop count a -1, increasingAlphaDuration: 1000, tutti gli altri valori a 0.
- Alpha(int loopCount, long increasingAlphaDuration).
- Alpha(int loopCount, long triggerTime, long phaseDelayDuration, long increasingAlphaDuration, long increasingAlphaRampDuration, long alphaAtOneDuration).
- Alpha(int loopCount, int mode, long triggerTime, long phaseDelayDuration, long increasingAlphaDuration, long increasingAlphaRampDuration, long alphaAtOneDuration, long decreasingAlphaDuration, long decreasingAlphaRampDuration, long alphaAtZeroDuration).

Metodi utili messi a disposizione da **Alpha** (il loro significato è ovvio):

- `setLoopCount(int loopCount) : void;`
- `setMode(int mode) : void;`
- `value() : float;`
- `value(long atTime) : float;`
- `setStartTime(long startTime) : void;` se non specificato, di default è quello di sistema;
- `setTriggerTime(long triggerTime) : void;`
- `setPhaseDelayDuration(long phaseDelayDuration) : void;`
- `setIncreasingAlphaDuration(long increasingAlphaDuration) : void;`
- `setAlphaAtOneDuration(long alphaAtOneDuration) : void;`
- `setDecreasingAlphaDuration(long decreasingAlphaDuration) : void;`
- `setAlphaAtZeroDuration(long alphaAtZeroDuration) : void;`
- `setIncreasingAlphaRampDuration(long increasingAlphaRampDuration) : void;`
- `setDecreasingAlphaRampDuration(long decreasingAlphaRampDuration) : void;`
- `pause() : void;`
- `pause(long time) : void;`
- `resume() : void;`
- `resume(long time) : void;`
- `isPaused() : boolean; .`

INTERPOLATOR E LE SUE SOTTOCLASSI

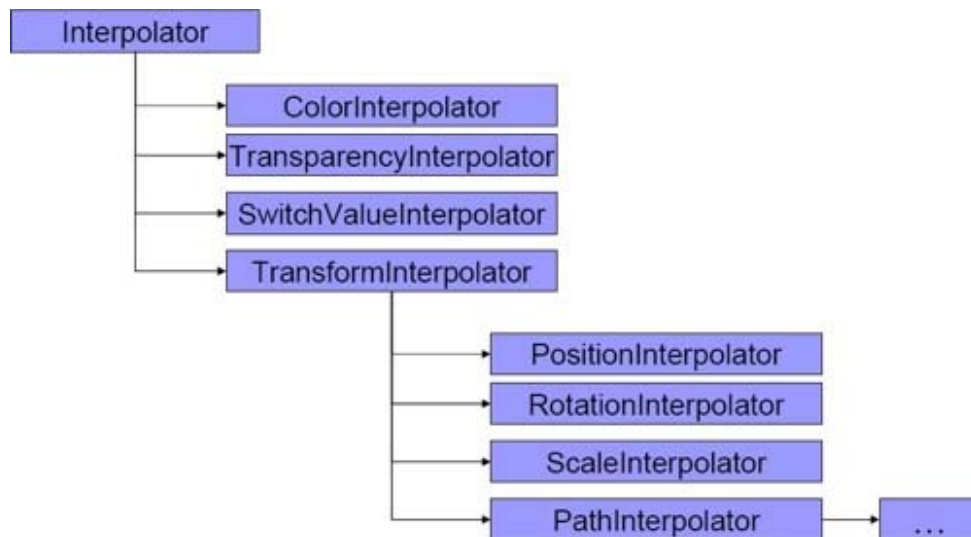
Un **Interpolator** è un oggetto che **eredita da Behavior**.

Esso **recupera** il valore dell'**Alpha** associato e **modifica**, in base a tale valore, una **proprietà** del **Target**.

Esistono vari **Interpolator**, che agiscono su **proprietà** diverse del **Target**: colore, trasparenza, posizione, ...

In tutti questi casi, bisogna ricordarsi di **impostare le relative capability** di trasformazione.

La **gerarchia** di classi che discendono da **Interpolator** è visibile nella figura seguente.



I vari **Interpolator** si differenziano tra loro sia per il **tipo** di azione svolta che per gli aspetti implementativi.

Verranno presi tutti in esame nel corso del capitolo.

SOTTOCLASSI DI INTERPOLATOR – COLOR INTERPOLATOR

Il **ColorInterpolator** modifica *un colore* di un **Material** effettuando una **interpolazione lineare** tra **due colori** tramite il valore di **Alpha**.

*Quale colore di **Material** viene modificato ?*

Questo viene specificato dallo stesso **Material**, tramite il metodo **setColorTarget(int)**, dove il **parametro** intero può essere un **valore** a scelta tra:

- AMBIENT;
- EMISSIVE;
- DIFFUSE;
- SPECULAR;
- AMBIENT AND DIFFUSE.

I **costruttori** di **ColorInterpolator** sono i seguenti:

- ColorInterpolator(Alpha alpha, Material target);
- ColorInterpolator(Alpha alpha, Material target, Color3f startColor, Color3f endColor); .

Il primo **costruttore** pone come **startColor** il **nero** e come **endColor** il **bianco**.

Metodi utili messi a disposizione da tale **classe** sono:

- setTarget(Material target) : void;
- setStartColor(Color3f color) : void;
- setEndColor(Color3f color) : void; .

Esempio '*Esempio ColorInterpolator*'.

SOTTOCLASSI DI INTERPOLATOR – TRANSPARENCY

INTERPOLATOR

TransparencyInterpolator si comporta come **ColorInterpolator**, con la differenza che qui non si va da un colore all'altro ma da un *valore di trasparenza* di partenza ad un finale, espressi mediante valori **float**.

I **costruttori** sono:

- `TransparencyInterpolator(Alpha alpha, TransparencyAttributes target);`
- `TransparencyInterpolator(Alpha alpha, TransparencyAttributes target, float minimumTransparency, float maximumTransparency);` .

Il primo **costruttore** varia il valore di **trasparenza da 0 a 1**.

Anche qui è necessario specificare un **TARGET**, che in questo caso è un **TransparencyAttributes**:

```
setTarget(TransparencyAttributes target) : void; .
```

Altri due **metodi** utili sono:

- `setMinimumTransparency(float transparency) : void;`
- `setMaximumTransparency(float transparency) : void; .`

NOTA: è interessante notare che un **Material** e un **TransparencyAttributes** possono essere **condivisi** da più oggetti, dunque un **ColorInterpolator** e un **TransparencyInterpolator**, che apparentemente accettano solo un **target**, possono gestire, rispettivamente, il **colore** e la **trasparenza** di **più oggetti contemporaneamente**.

Esempio: *'Esempio TransparencyInterpolator'*.

SOTTOCLASSI DI INTERPOLATOR – SWITCH VALUE INTERPOLATOR

SwitchValueInterpolator è un **interpolatore** che, come suggerisce il nome, fa uso di un **nodo Switch** (presentato nel paragrafo dedicato alle **animazioni LoD**) e dei **figli** ad esso collegati.

In particolare, si tratta di un **Interpolator** che **modifica il figlio** selezionato dallo **Switch interpolando** tra un paio di valori (iniziale e finale), che rappresentano **l'indice di partenza e l'indice di arrivo** all'interno della **lista di figli di Switch**.

L'interpolazione avviene tra i figli compresi tra questi due indici in base al **valore** recuperato dall'oggetto **Alpha** associato all'**Interpolator**.

Tutto ciò può apparire alquanto nebuloso in questa descrizione; **l'esempio 'Esempio SwitchValueInterpolator'** mostra in pratica cosa fa questo particolare **Interpolator**.

I **costruttori** di **SwitchValueInterpolator** sono:

- SwitchValueInterpolator(Alpha alpha, Switch target);
- SwitchValueInterpolator(Alpha alpha, Switch target, int firstChildIndex, int lastChildIndex); .

Il primo **costruttore** varia l'indice del figlio del nodo **Switch** selezionato scorrendo tutta la **lista dei figli del nodo Switch**.

Metodi utili messi a disposizione da questa **classe**:

- getTarget() : Switch;
- setTarget(Switch target) : void; .

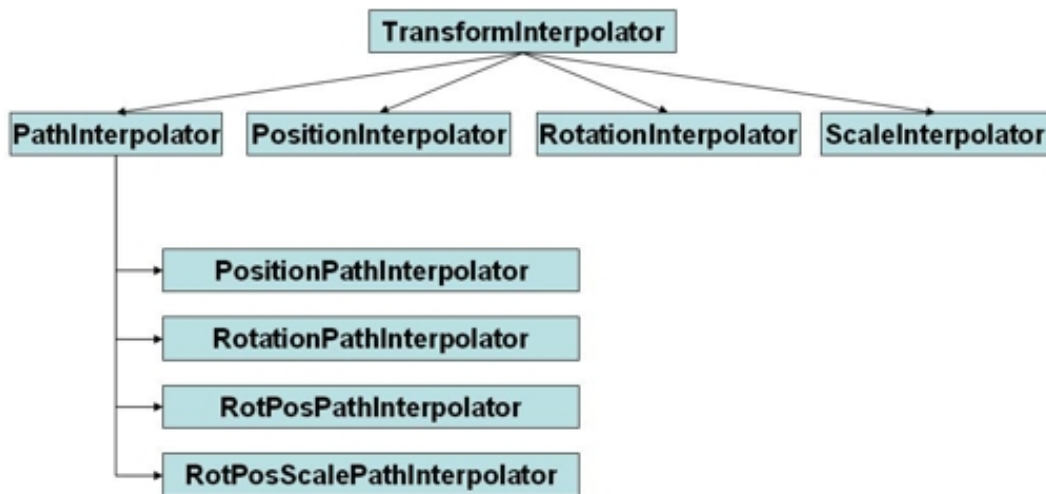
E' importante notare che questo **Interpolator** non modifica gli **oggetti** interpolando tra le loro forme, ma si limita a **scegliere**, in base al valore **Alpha**, quale renderizzare a video.

Parleremo di come **modificare le forme degli oggetti** mediante **interpolatori**, passando da una data **forma** ad un'altra, nel paragrafo dedicato alle animazioni **Morph**.

SOTTOCLASSI DI INTERPOLATOR – TRANSFORM INTERPOLATOR

TransformInterpolator è il capostipite di un insieme di **classi** che operano sui **TransformGroup**, consentendoci di **animare gli oggetti** spostandoli, ruotandoli e ridimensionandoli in base al valore restituito dall'oggetto **Alpha** associato e a due o più 'nodi', che definiscono i valori di **posizione**, **orientamento** e **scaling**.

Le **classi** che discendono da **TransformInterpolator** sono mostrate nella figura seguente.



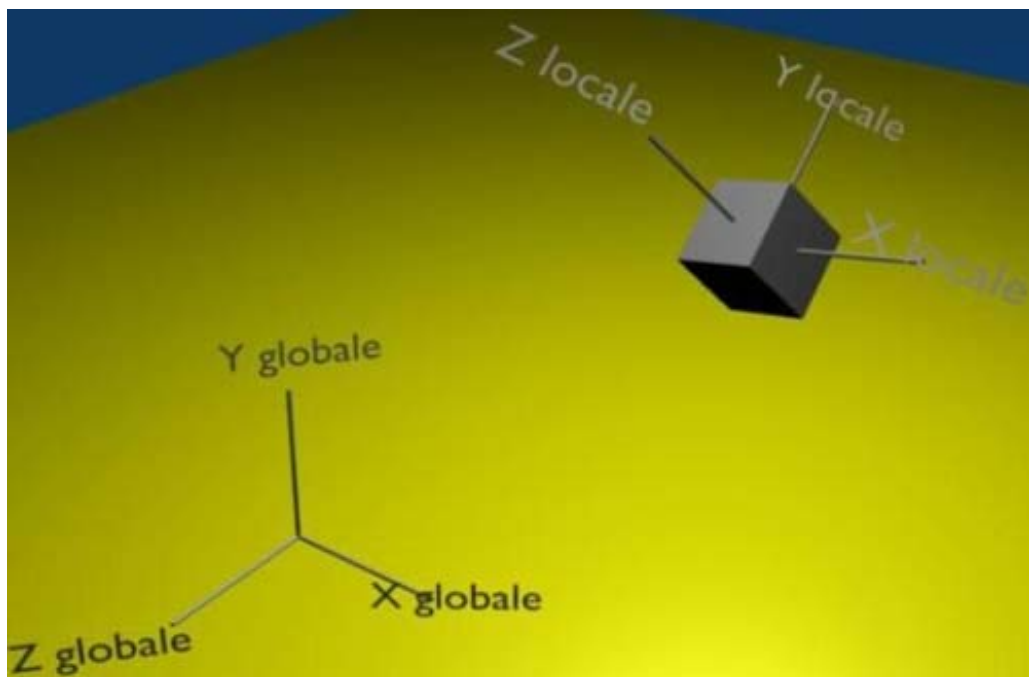
SISTEMI DI RIFERIMENTO: LOCALE E GLOBALE

Prima di proseguire con lo studio dei **TransformInterpolator** è bene chiarire il concetto di *sistema di riferimento locale*.

Quando un oggetto viene creato, di default assume il **sistema di riferimento dell'universo** virtuale di **Java3D**, detto sistema di riferimento *globale*.

Se, però, quest'oggetto viene **ruotato**, il suo **sistema di riferimento locale** non coinciderà più con quello *globale*.

Le **trasformazioni** degli **Interpolator** della famiglia **TransformInterpolator** si riferiscono al **sistema di riferimento locale**, ed è per questo motivo che diviene di fondamentale importanza riuscire a distinguere tra **locale** e **globale** ad esprimere le **trasformazioni** in base al sistema di riferimento **locale** degli oggetti.



SOTTOCLASSI DI TRANSFORM INTERPOLATOR – POSITION INTERPOLATOR

PositionInterpolator permette di variare la **posizione** di un oggetto (traslazione) interpolando, tramite **Alpha**, tra due valori che rappresentano le **posizioni iniziale e finale** (default: 0 e 1) individuate su un asse, che è **l'asse X del sistema di riferimento locale** dell'Interpolator.

Costruttori:

- `PositionInterpolator(Alpha alpha, TransformGroup target);`
- `PositionInterpolator(Alpha alpha, TransformGroup target, Transform3D axisTransform, float startPosition, float endPosition);` .

Metodi utili messi a disposizione da **PositionInterpolator** sono, ad esempio:

- `setStartPosition(float position) : void;`
- `setEndPosition(float position) : void;`
- `setTransformAxis(Transform3D axisOfTranslation) : void;` .

Il lettore ha a disposizione **tre esempi** riguardanti **PositionInterpolator**:

- **Esempio PositionInterpolator1** mostra una semplice traslazione da 0.0 a 10.0 sull'asse X.
- **Esempio PositionInterpolator2** mostra, oltre alla traslazione vista nell'esempio precedente, una traslazione lungo l'asse Y, ottenuta ruotando di 90 gradi intorno a Z l'asse di trasformazione dell'interpolatore.
- **Esempio PositionInterpolator3** serve a chiarire il significato dell'espressione 'lungo l'asse x del sistema di riferimento locale': il cubo è stato ruotato, per cui ora l'asse X LOCALE 'punta' verso l'alto e, infatti, il cubo trasla lungo l'asse Z GLOBALE.

SOTTOCLASSI DI TRANSFORM INTERPOLATOR – ROTATION INTERPOLATOR

RotationInterpolator opera sui **TransformGroup** al fine di far **ruotare** gli oggetti intorno ad un **asse** (di default, quello **X**) del sistema di riferimento **locale** dell'Interpolator.

La **rotazione** avviene tra due **estremi**, individuati con **due angoli (iniziale e finale)**, interpolando il valore recuperato dall'oggetto **Alpha** associato all'Interpolator.

Costruttori:

- `RotationInterpolator(Alpha a, TransformGroup t);`
- `RotationInterpolator(Alpha a, TransformGroup t, Transform3D axisOfTransform, float minimumAngle, float maximumAngle);` .

I **valori di default** del primo costruttore per **asse**, **angolo minimo** e **angolo massimo** sono, rispettivamente: **X**, **0**, **2π** .

Metodi utili messi a disposizione da **RotationInterpolator** sono, ad esempio:

- `setTransformAxis(Transform3D asseDiRotazione) : void;`
- `setMinimumAngle(float angolo) : void;`
- `setMaximumAngle(float angolo) : void;` .

Esempio: *'Esempio RotationInterpolator'*.

SOTTOCLASSI DI TRANSFORM INTERPOLATOR – SCALE INTERPOLATOR

ScaleInterpolator ridimensiona, nel tempo, gli oggetti.

Costruttori:

- `ScaleInterpolator(Alpha a, TransformGroup t);`
- `ScaleInterpolator(Alpha a, TransformGroup t, Transform3D axisOfTransform, float minimumScale, float maximumScale);` .

L'intervallo di scaling di default è `[0.1, 1.0]`.

Metodi utili sono:

- `setTransformAxis(Transform3D axisOfScale) : void;`
- `setMinimumScale(float scale) : void;`
- `setMaximumScale(float scale) : void;` .

Esempio: ***'Esempio ScaleInterpolator'***.

SOTTOCLASSI DI TRANSFORM INTERPOLATOR – PATH INTERPOLATOR

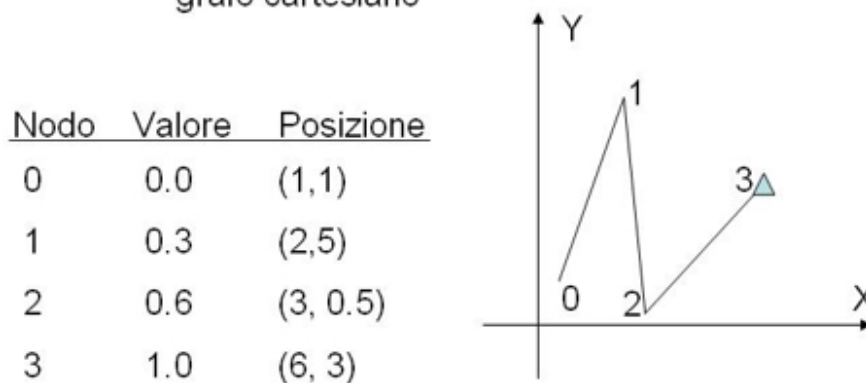
PathInterpolator è la superclasse per gli **Interpolator** che lavorano su un insieme di **nodi**.

Il valore dei **nodi** deve andare da **0.0f** a **1.0f**, in modo da metterli in **corrispondenza** con i **valori** assunti dall'oggetto **Alpha**.

A ciascun **nodo** sarà poi associato un **valore** che, a seconda dei casi (cioè della sottoclasse utilizzata), esprimerà una **posizione**, un **orientamento** o un fattore di **scaling**.

L'**animazione** derivante dall'interpolazione tra i vari valori verrà effettuata **automaticamente** da **Java3D**.

Esempio 2D: posizioni di un oggetto nel tempo in un grafo cartesiano



Il valore, tra 0 e 1, verrà poi associato al valore dell'Alpha ad un dato istante di tempo, per tutto il corso dell'animazione

Costruttori:

- PathInterpolator(Alpha a, TransformGroup t, float[] knots);
- PathInterpolator(Alpha a, TransformGroup t, Transform3D axisOfTransform, float[] knots);

Anche in questo caso, quindi, il **target** è un **TransformGroup**.

Le **sottoclassi** messe a disposizione sono:

- PositionPathInterpolator;
- RotationPathInterpolator;
- RotPosPathInterpolator;
- RotPosScalePathInterpolator.

Metodi utili messi a disposizione da **PathInterpolator** e dalle sue **sottoclassi**:

- `getArrayLengths()` : int;
- `getKnot(int index)` : float;
- `getKnots(float[] knots)` : void; i nodi verranno depositati nell'array di float passato come parametro;
- `setKnot(int index, float knot)` : void;
- `setKnots(float[] knots)` : void; .

AXISANGLE4D E QUATERNIONI

Alcune classi di **PathInterpolator** fanno uso di un particolare oggetto, *Quat4f*, per definire le **rotazioni**.

Prima di prendere in esame tali classi dovremo, quindi, soffermarci un pò sui **Quat4f**.

Un oggetto **Quat4f** serve a descrivere un *Quaternion* (quaternione) mediante **4 valori float**.

I **quaternioni** sono estensioni dei **numeri complessi** introdotti da **William Rowan Hamilton** nel 1843, sono formati da **quattro elementi**, utilizzati (tra le altre cose) in **Computer Grafica** per definire le **rotazioni nello spazio 3D**.

Dei quattro valori numerici che compongono i **quaternioni** solo uno rappresenta una dimensione reale, mentre gli altri tre si riferiscono a dimensioni immaginarie.

Il loro **prodotto non è commutativo** (ossia, $qa \cdot qb$ è diverso da $qb \cdot qa$), per cui, nel definire **rotazioni composte**, bisognerà prestare attenzione **all'ordine degli elementi**.

In questa sezione non li studieremo dal punto di vista matematico ma vedremo i concetti di base e qualche piccola scorciatoia al fine di comprendere **come utilizzarli** per le **animazioni** in **Java3D**.

Di seguito viene spiegato **come crearli** a partire da un altro oggetto: *AxisAngle4d*.

AXISANGLE4D

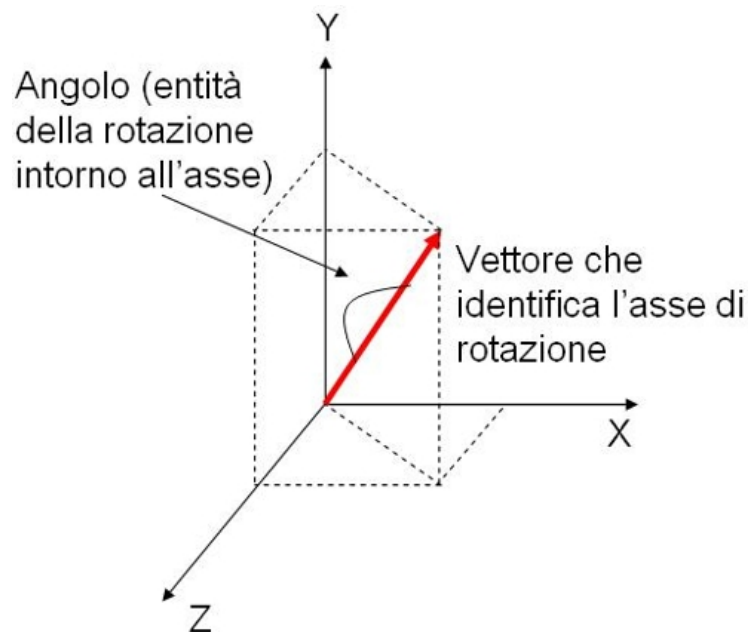
Un **AxisAngle4d** è un oggetto **AxisAngle**, utilizzato per descrivere le **rotazioni** nello **spazio 3D** con 4 valori double: 3 componenti di un **vettore** e un **angolo**.

In generale, con le tre componenti double identifichiamo un **vettore** che, da un punto di partenza, porta ad una destinazione, mediante i valori *x*, *y* e *z* trattati singolarmente (cioè secondo le regole della somma vettoriale).

Il **vettore** però, in questo caso, non verrà utilizzato per identificare un punto, ma un **asse di rotazione**.

L'asse di rotazione viene **calcolato sulla base delle coordinate locali dell'oggetto**.

Il **valore dell'angolo** rappresenta l'entità della **rotazione** intorno a tale **asse**.



Costruttori:

- AxisAngle4d;
- AxisAngle4d(Vector3d axis, double angle);
- AxisAngle4d(double[] a);
- AxisAngle4d(double x, double y, double z, double angle);
- AxisAngle4d(AxisAngle4d a1);
- AxisAngle4d(AxisAngle4d a1); .

Gli interpolatori accettano **Quaternioni**, dunque bisogna passare **da AxisAngle4d a Quat4f**, cosa che può essere fatta mediante le seguenti istruzioni:

```
Quat4f quats = new Quat4f();  
quats.set(new AxisAngle4d([x], [y], [z], [angolo])); .
```

Prima di vedere all'opera **Interpolatori** è **Quaternioni** può essere utile far **pratica** con oggetti statici, utilizzando a tal fine l'esempio *'EsempioQuaternioni'*.

Perchè, fino ad ora, i **Quaternioni** ed **AxisAngle4d** non sono stati trattati ?

La risposta è che **non ce n'era bisogno**: per le operazioni viste fino a questo punto, è sufficiente lavorare sui campi *rotX*, *rotY* e *rotZ*, molto più semplici da utilizzare.

NOTA: per creare **rotazioni complesse**, create cioè combinando **due o più rotazioni**, si consiglia di definire le due rotazioni in due **quaternioni** (passando, eventualmente, da **AxisAngle4d**), per creare in un secondo momento **la rotazione finale moltiplicando tra loro i quaternioni**, con il metodo *mul(quat2)* oppure con *mul(quat1, quat2)*, simili a quelli visti per i campi *rot*.

L'esempio *'EsempioQuaternioni'*, precedentemente menzionato, fa uso proprio di questa tecnica.

SOTTOCLASSI DI PATH INTERPOLATOR – POSITION PATH INTERPOLATOR

PositionPathInterpolator è un **PathInterpolator** per le **posizioni** (genera, quindi, traslazioni).

Costruttore:

- `PositionPathInterpolator(Alpha a, TransformGroup t, Transform3D axisOfTransform, float[] knots, Point3f[] positions); .`

Metodi utili messi a disposizione:

- `setPathArrays(float[] knots, Point3f[] positions) : void;`
- `setPosition(int index, Point3f position) : void; .`

Esempio: *'Esempio PositionPathInterpolator'*.

SOTTOCLASSI DI PATH INTERPOLATOR – ROTATION PATH INTERPOLATOR

RotationPathInterpolator è un '**PathInterpolator**' per le **rotazioni**.

Ad ogni **nodo** è associato un **valore angolare**, riferito **all'asse di rotazione X** del sistema **locale**; l'interpolazione genera l'**animazione** che raffigura il passaggio da un valore all'altro (una **rotazione**).

Costruttore:

- `RotationPathInterpolator(Alpha a, TransformGroup t, Transform3D axisOfTransform, float[] knots, Quat4f[] quats);`

Metodi utili:

- `setPathArrays(float[] knots, Quat4f[] quats) : void;`
- `setQuat(int index, Quat4f quat) : void; .`

SOTTOCLASSI DI PATH INTERPOLATOR – ROTPOSPATH INTERPOLATOR

RotPosPathInterpolator modifica sia la **posizione** che l'**orientamento**, facendo uso di un array di **Point3f** per le **posizioni** e di un array di **Quat4f** per le **rotazioni**.

Ciascun elemento di questi due **array** verrà associato ad un **nodo dell'array dei nodi**.

Costruttore:

- `RotPosPathInterpolator( Alpha a, TransformGroup t, Transform3D axisOfTransform, float[] knots, Quat4f[] quats, Point3f[] positions); .`

Metodi utili messi a disposizione:

- `setPathArrays(float[] knots, Quat4f[] quats, Point3f[] positions) : void;`
- `setPosition(int index, Point3f position) : void;`
- `setQuat(int index, Quat4f quat) : void;.`

Esempio: '*Esempio RotPosPathInterpolator*'.

SOTTOCLASSI DI PATH INTERPOLATOR – ROTPOSSCALEPATH INTERPOLATOR

RotPosPathInterpolator modifica sia la **posizione**, l'**orientamento** e le **dimensioni** di un oggetto, facendo uso di un array di **Point3f** per le **posizioni**, di un array di **Quat4f** per le **rotazioni** e di un array di **float** per definire i fattori di **scaling**.

Ciascun elemento di questi tre array verrà associato ad un **nodo** dell'array dei **nodi**.

Costruttore:

- `RotPosScalePathInterpolator( Alpha a, TransformGroup t, Transform3D axisOfTransform, float[] knots, Quat4f[] quats, Point3f[] positions, float[] scales);` .

Metodi utili messi a disposizione:

- `setPathArrays(float[] knots, Quat4f[] quats, Point3f[] positions, float[] scales) : void;`
- `setPosition(int index, Point3f position) : void;`
- `setQuat(int index, Quat4f quat) : void;`
- `setScale(int index, float scale) : void;` .

Esempio: *'Esempio RotPosScalePathInterpolator'*.

ANIMAZIONI “A COMANDO”

Finora abbiamo visto **animazioni** presenti nella scena **dall'inizio**, cioè inserite in maniera attiva nell'ambiente virtuale al momento dell'avvio del programma.

A volte è necessario, però, **inserire delle animazioni 'al volo'**, cioè a runtime, magari partendo da **dati o parametri inseriti dall'utente**.

Vedremo quindi come **inserire un'animazione** a comando nella **scena**.

A tal proposito, il lettore ha a disposizione l'esempio **'Esempio ICBM'**, una base di partenza per un **Artillery Game**, il cui funzionamento viene spiegato qui di seguito.

Si vuole simulare il lancio di un **missile** (da qui il nome **ICBM**, appunto) da una plancia verso una destinazione, con le **coordinate di destinazione** impostate dall'utente mediante una semplice GUI.

All'avvio, lo **scenario Java3D** deve essere visibile da subito, insieme ad una **GUI Java2D** con tre campi per impostare le **'coordinate di arrivo'** di un **missile** (non ancora visibile). Le **coordinate di partenza** saranno invece fisse: (0, 0, 0). L'utente quindi dovrà impostare le **coordinate di arrivo** nella **GUI** e cliccare sul pulsante **'Fuoco'**; a quel punto, verrà visualizzata a video **l'animazione**.

***NOTA:** è demandata al lettore la creazione o il reperimento di due modelli 3D in formato OBJ, chiamati “BaseLancio.obj” e “Polaris.obj” (ma ovviamente possono essere rinominati, cambiando il path nel codice sorgente), rappresentanti rispettivamente una piccola base di lancio per il missile e il missile stesso; sul sito web dell'autore, all'indirizzo www.redbaron85.com, sono disponibili diversi modelli 3D utilizzabili per questo scopo, scaricabili gratuitamente nella sezione Modelli 3D del sito.*

Il punto fondamentale è, quindi, **aggiungere un blocco (Interpolator, Oggetto, Alpha) 'al volo' nello Scene Graph** ed avviare l'animazione.

I **passi** da seguire, che riguardano in questo caso solo **Java3D** e non la **GUI**, sono:

- creare la radice del Content Branch Graph, rendendola globale e attivando la capability ALLOW CHILDREN EXTEND;
- creare ed agganciare al Content Branch Graph gli elementi 'fissi' della scena, quelli cioè che devono essere visualizzati all'avvio dell'applicazione, statici ed animati.

Bisogna quindi definire un **metodo** che, una volta **invocato** (ad esempio, alla pressione del tasto **'Fuoco'** sulla **GUI**):

- calcoli 'al volo' le posizioni del missile, tenendo conto delle coordinate di destinazione (nel nostro caso, a dire il vero molto banale, la posizione di partenza è fissa e coincide con l'origine);
- carichi la mesh del missile, agganciandola ad un TG;
- crei un Alpha e un RotPosPathInterpolator con i dati calcolati;
- agganci l'Interpolatore ed il TG alla radice del Content Branch Graph.

Tutte queste operazioni sono semplici da implementare, ma anche eseguendo tutti i passi potremmo avere una **spiacevole sorpresa**: una volta creato (prima ancora di creare l'**Interpolatore** ed agganciare il tutto alla **scena**), l'**Alpha** terrebbe conto **dell'orologio di sistema**, così da visualizzare il **missile** già **in volo** o addirittura sul bersaglio.

Per ovviare a questo **problema**, utilizzare la seguente **istruzione** (da inserire in un **blocco try-catch**) per '**sincronizzare**' l'esecuzione dell'animazione alla partenza dell'Alpha:

```
Thread.sleep( (int) ([durataAnimazione]  
( (System.currentTimeMillis() - tempoDiAvvioDelSistema)
```

ANIMAZIONI MORPH

Morph è un **interpolatore** (ma solo nel senso che 'passa da un nodo all'altro', non nel senso di oggetto **Interpolator Java3D**) di **Geometry Array**.

La classe **Morph**, comunque, non estende nè **Interpolator** nè **Behavior**, ma semplicemente **Node**. Consente di **modificare le geometrie delle mesh**, operando su sottoclassi di **GeometryArray**.

I **costruttori** sono:

- `Morph(GeometryArray[] geometryArrays);`
- `Morph(GeometryArray[] geometryArrays, Appearance appearance) ;` .

Un **Morph** è quindi una collezione di **geometrie** e di '**pesi**', che vengono definiti con il **metodo** *`setWeights(double[] weights)`* da applicare alle geometrie.

La **somma** di tutti i pesi presenti nel **Morph** deve valere 1.

A differenza di **DistanceLoD**, che estende da **Behavior** e si attiva automaticamente, quindi, è necessario **associare un Behavior al Node Morph**, altrimenti avremo solo una collezione di coppie geometria-peso, ma nessuna animazione.

Capabilities interessanti di Morph sono:

- `ALLOW COLLISION BOUNDS READ;`
- `ALLOW COLLISION BOUNDS WRITE;`
- `ALLOW GEOMETRY ARRAY READ;`
- `ALLOW GEOMETRY ARRAY WRITE;`
- `ALLOW WEIGHTS READ;`
- `ALLOW WEIGHTS WRITE;`
- `ALLOW APPEARANCE OVERRIDE READ;`
- `ALLOW APPEARANCE OVERRIDE WRITE;`
- `ALLOW APPEARANCE READ;`
- `ALLOW APPEARANCE WRITE.`

Di seguito viene quindi mostrato **come creare un Morph ed inserirlo in una scena**.

Per prima cosa bisogna **creare un oggetto Morph**, ad esempio con:

```
Morph mioMorph=new Morph(arrayGeometrie);  
mioMorph.setCapability(Morph.ALLOW WEIGHTS WRITE);
```

dove **'arrayGeometrie'** è, appunto, un array di oggetti di tipo **Geometry**, secondo le specifiche descritte all'inizio di questa sezione.

Morph è un **Node**, quindi va aggiunto a un **Branch Group** o ad un **Transform Group** per poterlo inserire nella scena.

A questo punto, per **riprodurre l'animazione**, bisogna creare, con una apposita **classe** la cui definizione spetta al programmatore (che può quindi personalizzarla in base alle proprie esigenze), un **oggetto erede di Behavior** per 'controllare' l'oggetto **Morph** e permetterne l'animazione; ossia, in maniera più corretta, il passaggio da una **Geometry** all'altra, tra quelle contenute nel **Morph**, costruito proprio con un **array di Geometry**.

Si tratta di una **classe** che, *concettualmente*, è a metà tra un **Behavior** e un **Interpolator**, in quanto **si attiva con degli eventi** (come i **Behavior**), fa uso (come i **Behavior**) di un **target** (il nostro **Morph** 'mioMorph') e (come gli **Interpolator**) di un oggetto **Alpha**, creato proprio per descrivere l'animazione; a livello *implementativo*, comunque, eredita solo da **Leaf**.

La costruzione di un simile **Behavior** avverrà, quindi, passandogli *almeno* l'**Alpha** e il **nodo / target Morph**; nulla vieta comunque di passare altri argomenti, in base alle proprie esigenze.

Ovviamente, dal momento che si tratta di un **Behavior**, bisognerà impostarne gli **SchedulingBounds** e si dovrà aggiungere tale oggetto, insieme al nodo **Morph**, alla **scena**.

A modificare la **geometria** di un oggetto, passando da una **geometria** all'altra in maniera 'dolce', è dunque un'azione combinata di **Morph** e del **Behavior** personalizzato.

Esempio: **'EsempioMorph'**.

Combinando gli effetti di **Morph** e le trasformazioni spaziali sulle **Leaf**, è possibile quindi generare **animazioni davvero complesse**; basti pensare, ad esempio, ad un giocatore di calcio: l'animazione degli **arti** può essere fatta con **Morph**, che agisce sulla **forma della geometria**, mentre lo **spostamento** del corpo avviene operando una trasformazione sul **TransformGroup** che controlla l'oggetto visuale, partendo, in entrambi i casi (ma con **Behavior** differenti), dai **tasti premuti dall'utente**.

* * *
* * *

Capitolo 12

Extra

In questo capitolo verranno presentati alcuni argomenti che permettono di implementare **funzionalità extra** nei **programmi** basati su **Java3D**.

Verrà accennato infatti a come **Java3D** rileva le **collisioni** e ad un **oggetto** ed un **metodo** ad esse correlati.

In seguito, verrà mostrato come '**scrivere sulla Canvas3D**', attraverso il **metodo** *postRender*.

Verrà quindi presentata una breve panoramica sui **thread** in **Java** al fine di mostrare, successivamente, come sfruttare i **Thread** per realizzare **applicazioni Java 3D** un pò più **complesse**, che prevedano, ad esempio, un **aggiornamento della scena automatico**, basato su **eventi** in arrivo dalla rete, ecc...

Infine, verrà introdotto il **metodo** *setTextureMode* di **TextureAttributes** e verranno discussi i suoi possibili **parametri** con i relativi **effetti** prodotti.

LE COLLISIONI IN JAVA 3D

Il sistema delle **collisioni** in **Java3D** si basa sui **Behavior** e sui **bound**.

La **collisione** viene rilevata da un **Behavior** associato alla **scena** che prende in input, al **costruttore**, almeno un **parametro** (l'oggetto da '**tenere d'occhio**', o un **bound** che lo rappresenta).

Il **Behavior** si attiverà, quindi, in base ad uno dei **criteri di wakeup** legati alle **collisioni**, che sono:

- `WakeupOnCollisionEntry;`
- `WakeupOnCollisionMovement;`
- `WakeupOnCollisionExit; .`

Un semplice **esempio** di **rilevamento delle collisioni** mediante un **Behavior** creato ad hoc è visibile in '**EsempioCollisioni**'.

In questo caso, tenendo premuto il tasto destro del mouse sul cubo in alto e spostandolo (**picking**) fino a farlo **collidere** col cubo posizionato nell'origine, verranno mostrate a video delle **notifiche** della **collisione** tra i due oggetti.

Ci si potrebbe aspettare di vedere gli oggetti '**fermarsi**', nei loro movimenti, quando entrano in **collisione** con altri oggetti: ciò non avverrà.

Il sistema delle **collisioni** di **Java3D**, infatti, non si occupa di **gestire** le collisioni, ma solo di **rilevarle**.

Sta al **programmatore** gestire tali **eventi**, definendo quale **azione** intraprendere.

Le **soluzioni**, a questo punto, possono essere **molteplici**; ad esempio, si può far uso di **bound** che '**anticipino**' il movimento che si vuol far fare agli **oggetti** (per verificare la presenza di **collisioni** ed evitare, eventualmente, di **muovere** l'oggetto), oppure si può decidere di **discretizzare lo spostamento** e di eseguire dei '**micro spostamenti**' dell'**oggetto** fintantochè lo stesso non si trova a **collidere** con un altro **oggetto**, ecc....

Fortunatamente, la rete **Internet** abbonda di possibili **soluzioni** e, comunque, nulla vieta agli **sviluppatori** di cercarne di proprie.

IL METODO GET TRIGGERING PATH E L'OGGETTO SCENE GRAPH PATH

getTriggeringPath è un **metodo** fornito dalle **classi** di **wakeup criterion** che permettono la gestione delle **collisioni**: **WakeupOnCollisionEntry**, **WakeupOnCollisionExit** e **WakeupOnCollisionMovement**.

Tale **metodo** recupera il **'percorso'** (nello **Scene Graph**) dell'oggetto con il quale il nostro **target** è, a seconda dei casi (ossia, della classe di **wakeup** cui si riferisce) entrato in **collisione**, **'è in collisione'** o **'non è più'** in collisione.

La firma del **metodo** è:

```
getTriggeringPath() : SceneGraphPath; .
```

Come è possibile notare dalla firma del **metodo**, **getTriggeringPath** restituisce un **oggetto** di tipo **SceneGraphPath**: cos'è ?

Si tratta di un **oggetto** costituito da un **Locale**, un nodo **'terminale'** (nel nostro caso, si tratta del **nodo** con il quale il nostro **target collide**) e un **array** interno di **nodi**: il **percorso (path, appunto)** dal **nodo terminale al Locale**.

NOTA: il **nodo** di indice 0, nell'**array** interno di **nodi** del **percorso**, è quello più vicino al **Locale**, mentre il **nodo** di **indice (length – 1)** è il padre del **nodo terminale**.

E' possibile creare uno **SceneGraphPath** esplicitamente, conoscendo il **Locale** e il **nodo terminale**, utilizzando i **costruttori** di **SceneGraphPath**, che sono:

- `SceneGraphPath()` ;
- `SceneGraphPath(Locale root, Node object)` ;
- `SceneGraphPath(Locale root, Node[] nodes, Node object)` ; .

Il funzionamento di ciascun **costruttore** è facilmente intuibile.

Metodi interessanti sono:

- `nodeCount()` : `int`, che restituisce il **numero di nodi** presenti nel **Path** ;

- `getLocale()` : `Locale`, che restituisce il **Locale** che sta in cima al **Path** ;
- `getNode(int index)` : `Node`, che restituisce il **Nodo** che si trova all'**indice** specificato ;
- `getObject()` : `Node`, che restituisce il **Nodo** **'terminale'** del **Path** .

Per gli ultimi tre **metodi** appena elencati esistono i corrispettivi **metodi set**, che ovviamente **prendono un parametro (Locale o Node, a seconda dei casi) e restituiscono void**.

SCRIVERE SULLA CANVAS 3D

A volte può rendersi necessario il dover **stampare delle stringhe o delle forme grafiche sulla Canvas3D**; si pensi, ad esempio, a **‘comunicazioni’** del sistema all'utente o a **maschere da applicare al frame** (ad esempio in un **videogioco**, per mostrare la ‘barra dell'energia vitale’ o i ‘colpi rimasti’, ecc).

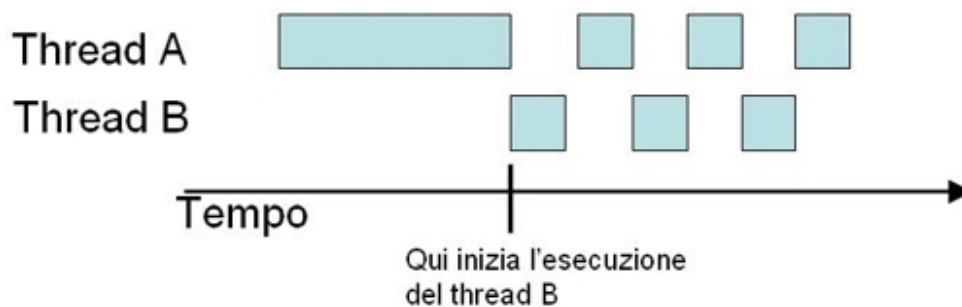
La **soluzione** non è particolarmente complessa: tutto sta nel **sostituire** la **Canvas3D** di default con un **nuovo oggetto personalizzato, che eredita da Canvas3D** e che sovrascrive il **metodo *postRender() : void***, come ad esempio:

```
public class MiaCanvas3D extends Canvas3D
{
    public MiaCanvas3D(GraphicsConfiguration conf)
    {
        super(conf);
    }
    public void postRender()
    {
        // metodo utilizzato per scrivere nella Canvas3D
        J3DGraphics2D draw;
        draw = this.getGraphics2D();
        draw.setColor(Color.red);
        draw.setFont(new Font(sanserif, Font.ITALIC, 20));
        draw.drawString(Premi Q per Uscire, 10, 70); // il testo
che viene scritto nella Canvas3D
        draw.flush(true);
    }
}
```

I THREAD IN JAVA

La **programmazione multithread** permette di svolgere più operazioni ‘contemporaneamente’.

Le virgolette sono d'obbligo in quanto **solo su sistemi multiprocessore**, con sistemi operativi adatti ed **in assenza di conflitti R—W** (lettura o scrittura) sui dati, due o più cose possono avvenire CONTEMPORANEAMENTE.



In pratica, su un **singolo processore** vengono eseguiti più **compiti** ('task': da qui il termine 'multitasking', ossia il poter lavorare con più cose e su oggetti diversi contemporaneamente), a ciascuno dei quali è dedicato un **intervallo di tempo** ('slice', lett.: fetta) per volta.

CONSIGLI D'UTILIZZO

Quando utilizzare i **thread** ?

Ad esempio, quando stiamo copiando dei file (nel frattempo potremmo utilizzare un browser web, ecc) o, nel caso dei videogiochi, per gestire la GUI del gioco e controllare gli eventi che arrivano dalla rete (gioco multiplayer).

Come per ogni cosa, anche **un uso eccessivo dei thread** può portare a **conseguenze spiacevoli**.

A parte lo spreco di **risorse** e la riduzione della **velocità** di esecuzione, potrebbe essere necessario dover **svolgere un'operazione** o un blocco di codice **COMPLETAMENTE**, cioè **senza interruzioni** (con salti ad altri **thread**): questo avviene, ad esempio, quando si lavora con **variabili condivise** (es.: il giocatore X si sposta dalla casella 1 alla casella 2, mentre il giocatore Y spara un colpo verso la casella 1: l'aver colpito o meno X dipenderà quindi dallo **scheduling dei thread** di gestione del movimento e di verifica delle collisioni; può anche verificarsi, ad esempio, il caso in cui il **thread** che aggiorna l'aspetto grafico venga interrotto quando l'aggiornamento è a metà dello schermo per passare al **thread** che calcola le nuove posizioni dei giocatori: avremo quindi giocatori che nella parte superiore dello

schermo avranno una parte del corpo in una posizione e, nella seconda metà dello schermo, la parte restante del corpo in un'altra posizione !!!).

Per risolvere alcuni problemi legati allo **SCHEDULING** e all'**ESECUZIONE ATOMICA** (atomica nel senso di 'indivisibile', **non frammentabile**) dei **thread** viene introdotto il concetto di **SINCRONIZZAZIONE**.

LA SINCRONIZZAZIONE

La **sincronizzazione** permette ad un metodo di **bloccare (lock)** e **sbloccare (unlock)** l'accesso ad una **variabile**.

Va quindi utilizzata quando due o più thread devono accedere allo stesso **campo** o **oggetto**.

Il **lock** viene rilasciato quando **l'esecuzione del metodo** o comunque della porzione di **codice** da tenere d'occhio è **terminata**, sia che questo avvenga normalmente che con una **eccezione**.

L'utilizzo della **sincronizzazione** introduce **ritardi** e la possibilità che si verifichino **deadlock**, per cui bisognerebbe evitarlo quando non strettamente necessario (ad es.: non c'è bisogno di **sincronizzare** metodi che usano solo **variabili locali**, o blocchi di codice che non vengono utilizzati da più **thread**; è inoltre possibile sincronizzare solo una parte, quella 'pericolosa', di un **metodo**, e non necessariamente TUTTO il **metodo**).

DEADLOCK

Un **deadlock** (o 'stallo') è una situazione che si verifica quando **due o più thread** restano l'uno **in attesa** dell'altro senza possibilità di **sbloccare** tale situazione (a meno di una terminazione esplicita dell'esecuzione, o di un'altra soluzione 'deus ex machina').

Ad esempio, una situazione che produce un **deadlock** è la seguente: supponiamo che ci siano due **thread** che necessitano di due variabili, **x** e **y**, per eseguire le loro operazioni; allora, se si verifica che:

1. il thread A fa un lock sulla variabile **x**;
2. il thread B fa un lock sulla variabile **y**;

3. il thread A tenta di acquisire la variabile y, ma non ci riesce perchè è già acquisita da B e, nel mentre...

4. ... il thread B tenta di acquisire la variabile x, ma non ci riesce perchè è già acquisita da A, che è fermo in attesa
si ha un **deadlock** !

(**NOTA:** questo esempio è un pò troppo banale, anche perchè esistono **primitive di lock** che permettono di effettuare il **lock su più variabili contemporaneamente** al fine di evitare problemi del genere... comunque, è sufficiente ad illustrare come possano verificarsi problemi di questo tipo).

C'è un rischio di **deadlock**, quindi, quando **più thread tentano di acquisire dei lock senza un ordine preciso**. Per evitare i **deadlock** è necessario **progettare** l'esecuzione tenendo conto di quali thread acquisiscono **lock**, su quali variabili e in quale **ordine**.

Cercare di considerare sempre ogni eventualità.

IMPLENTARE I THREAD IN JAVA

E' possibile implementare i **thread in Java** principalmente in due modi:

- estendendo la **CLASSE** 'Thread';
- implementando l'**INTERFACCIA** 'Runnable'.

In effetti, esiste un **terzo metodo**, che consiste nel **creare classi al volo** all'interno di altre classi per eseguire dei **thread ad hoc**; una soluzione del genere rende il codice difficile da leggere e da capire in fase di ricerca degli errori, per qui è da evitare.

ESTENDERE LA CLASSE THREAD

Per **estendere la classe Thread**, scrivere semplicemente '**extends Thread**' durante la dichiarazione della classe e **sovrascrivere il metodo run() : void**.

Le **azioni** da effettuare durante l'esecuzione del **thread** andranno messe quindi nel corpo del **metodo run**.

Il **thread** viene **attivato** invocandone il **metodo start()**.

Esempio: scrivo una classe `MioThread`...

```
public class MioThread extends Thread
{
    public MioThread()
    {
        // Costruttore
    }
    public void run()
    {
        System.out.println(Thread in esecuzione);
    }
}
```

e la creo ed avvio così:

```
MioThread mioThread1 = new MioThread(); mioThread1.start();
```

A questo punto ci sono, in esecuzione, due **thread** per lo stesso programma: il **thread** che gestisce il **main** del programma (che ha creato ed avviato **mioThread1**, e continua la sua esecuzione) e **mioThread1**, che fa quello che gli diciamo di fare.

IMPLEMENTARE L'INTERFACCIA RUNNABLE

Se non vogliamo creare una **classe ad hoc** per il **thread**, ma vogliamo che un'altra classe (che magari eredita già da un'altra) si comporti come una classe **Thread**, possiamo **implementare l'interfaccia runnable**.

Anche in questo caso **sovrascriveremo il metodo run()**, mentre nel **costruttore** della classe creeremo un oggetto **Thread** passandogli, come parametro, l'oggetto che lo sta creando; ad esempio:

```
public class MiaClasse2 extends MiaClasse1 implements Runnable
{
    public MiaClasse2
    {
        // Costruttore
        Thread mioThread1 = new Thread(this);
        mioThread1.start();
    }
    public void run()
    {
        System.out.println(Thread in esecuzione);
    }
}
```

LA SINCRONIZZAZIONE DEI THREAD IN JAVA

Del perché dover **sincronizzare** abbiamo parlato precedentemente, nella parte ‘teorica’... passiamo all’implementazione in **Java**.

In **Java** si fa uso della **parola chiave *synchronized***; ad esempio, per rendere **synchronized** un **metodo**:

```
public synchronized int getValoreX()
{
    return x;
}
```

o:

```
public synchronized void setValoreX(int in)
{
    x = in;
}
```

E’ possibile **sincronizzare l’oggetto** sul quale si lavora, al costo di qualche istruzione in più nel bytecode.

Il lato buono è che, con questo **metodo**, è possibile **sincronizzare solo una parte** di un **metodo** e non necessariamente tutto il **metodo**; ad esempio potremmo scrivere:

```
public void setValoreX(int in)
{
    System.out.println("Sto per entrare nel blocco sync");
    synchronized(this)
    {
        System.out.println("Dentro il blocco sync");
        x = in;
    }
    System.out.println("Sono uscito dal blocco sync, ma non ancora dal metodo setValoreX");
}
```

E’ possibile effettuare **lock** su **qualsiasi oggetto**, anche sugli **array**, ma **non** sui tipi **primitivi**.

METODI DI SINCRONIZZAZIONE: WAIT, NOTIFY e NOTIFYALL

wait() blocca l'esecuzione del **thread** invocante fino a che un altro **thread** invoca una **notify()** sull'oggetto.

Si fa sempre **dopo aver testato una condizione** (ed in un ciclo, per essere sicuri che al risveglio la condizione sia verificata).

Esempio:

```
while (! condition) // se non pu procedere
{
    this.wait (); // aspetta una notifica
}
```

Il **thread** invocante viene **bloccato**, il **lock** sull'oggetto è **rilasciato** automaticamente, mentre i **lock** su altri **oggetti** sono **mantenuti** (bisogna quindi fare attenzione a possibili **deadlock**).

La variante **wait(long timeout)** blocca il **thread** per al massimo **timeout** *millisecondi*.

notify() risveglia un solo **thread** tra quelli che aspettano sull'oggetto in questione.

Se più **thread** sono in attesa, la scelta di quale **thread** svegliare viene fatta dalla **JVM**.

Una volta **risvegliato**, il **thread** compete con ogni altro (non in **wait**) che vuole accedere ad una **risorsa protetta**.

Non è un buon metodo di risveglio perchè il **thread** risvegliato, scelto dalla **JVM**, potrebbe non essere in grado di proseguire.

notifyAll() risveglia **tutti i thread** che aspettano sull'oggetto in questione.

Da preferire rispetto a **notify**, quando possibile.

ALTRE OPERAZIONI INTERESSANTI POSSIBILI CON I THREAD IN JAVA --- PAUSA

Possiamo mettere in **pausa** un **Thread** per un pò di tempo con la seguente riga di codice:

```
Thread.sleep([tempo in millisecondi]); .
```

JOIN

Se vogliamo che un **thread** **attenda** che un altro **thread** termini la sua esecuzione per avviarsi, utilizziamo la seguente istruzione:

```
myThread.join();
```

INTERRUPT

Il metodo **interrupt()** permette di **interrompere l'esecuzione** del **thread** sul quale si invoca, solo quando lo stato dell'oggetto lo consente, cioè **quando non è in esecuzione**, ma in attesa di un evento (in modo da mantenere lo **stato consistente**).

ANIMAZIONI MEDIANTE THREAD: UN SEMPLICE ESEMPIO

L'esempio '*Simulatore1*' mostra come utilizzare i **thread** di **Java** per creare facilmente una semplice applicazione in grado di **recuperare gli eventi da tastiera** ed aggiornare di conseguenza sia il **SimpleUniverse** che la **GUI**, generando, tra l'altro, **animazioni** che non fanno uso degli **interpolatori**.

Una volta avviata l'applicazione viene mostrato un frame con una **Canvas3D** ed un **universo virtuale** popolato da pochi elementi.

L'idea è quella di simulare la **navigazione nell'universo 3D** mediante una navicella dotata di un **'motore'** di potenza regolabile che spinge in avanti l'osservatore.

E' presente un **behavior personalizzato** per il controllo degli **eventi** generati da **tastiera**.

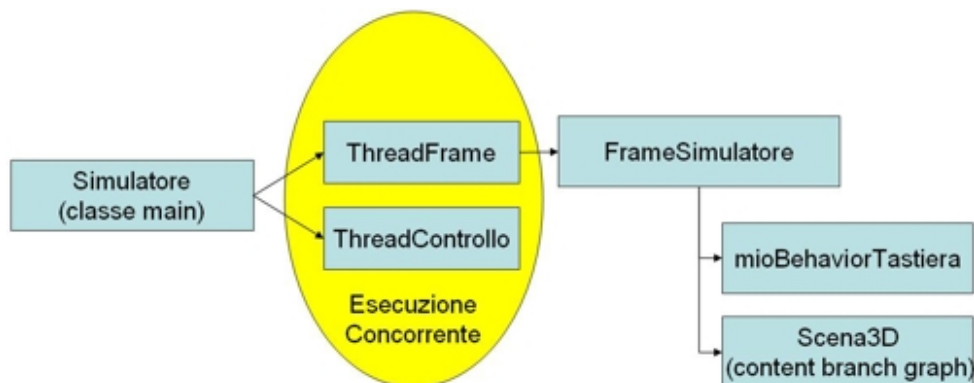
Alla pressione del tasto +, si ha un **aumento della potenza del 'motore'** e l'utente (ossia, il punto di vista dell'osservatore) viene **spostato in avanti** di una data quantità.

La **velocità di spostamento** dipende dal valore di **'potenza reattore'**, una quantità che può assumere valori tra 0 e 110 (iniziale = 0), a 'passi' di 10 (dunque i tasti + e incrementano e decrementano il valore di 'potenza reattore' di 10).

Attenzione: la **velocità di spostamento** non è uguale a questo **vettore** ma il **VETTORE DELLO SPOSTAMENTO** viene **calcolato** a partire da tale valore... ciò lascia spazio a **calcoli più elaborati**, che tengano conto di **altri fattori** (ad esempio, la **gravità**).

Dovendo gestire una sola variabile di **'stato del gioco'** non ho ritenuto necessario dover creare un oggetto **'statodelgioco'**, necessario invece per tenere conto di **più fattori** in maniera efficiente.

Un semplice **schema** degli **elementi** presenti nell'applicazione è mostrato nell'immagine seguente.



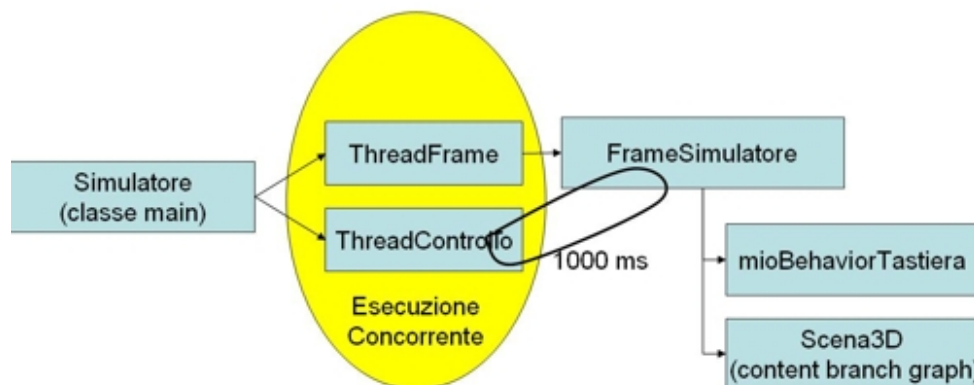
E' possibile notare come **all'avvio** del programma vengano creati ed avviati **due thread**: il primo, **'ThreadFrame'**, si occupa di gestire la **GUI** (rappresentata da **'FrameSimulatore'**, **'mioBehaviorTastiera'** e **'Scena3D'** (che crea il **ContentBranchGraph**), ossia aspetto grafico ed eventi generati da tastiera), mentre **'ThreadControllo'** non fa altro che verificare il valore di **'potenza reattore'** e muovere, di conseguenza, la posizione dell'osservatore.

'ThreadControllo' effettua il suo ciclo **ogni secondo** (ma è possibile abbassare il tempo che intercorre tra un ciclo e l'altro modificando il parametro di **Thread.sleep(tempo in millisecondi)**).

'ThreadFrame' avvia il Frame e, di conseguenza, la classe **mioBehaviorTastiera** e **Scena3D**.

'Scena3D' crea un universo virtuale provvisto di **griglia di riferimento** per identificare il piano XZ e di due coni (posizionati a (0.0f, 0.0f, -5.0f) e (0.0f, 0.0f, -15.0f)).

```
public void run()
{
    while(true)
    {
        try
        {
            Thread.sleep(1000);
            fs1.avanza(fs1.getPotenzaReattore());
        }
    }
}
```



La classe **'mioBehaviorTastiera'** fornisce un **behavior** personalizzato che reagisce alla pressione di alcuni tasti.

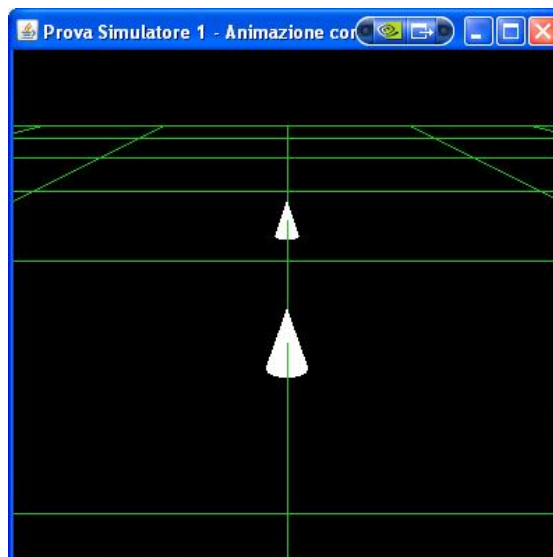
I **comandi** per l'interazione da **tastiera** sono mostrati nella figura seguente.

Tasto	Effetto
W	Avanza di 10 unità
A	Trasla a sinistra di 10 unità
S	Trasla indietro di 10 unità
D	Trasla a destra di 10 unità
Freccia ←	Ruota verso sinistra di 10 gradi
Freccia →	Ruota verso destra di 10 gradi
Freccia ↑	Ruota verso l'alto di 10 gradi
Freccia ↓	Ruota verso il basso di 10 gradi
+	Aumenta la potenza del reattore di 10 unità
-	Diminuisce la potenza del reattore di 10 unità

E' quindi possibile **modificare la posizione e l'orientamento** dell'utente/osservatore mediante i tasti **freccia** e **WASD**.

A questo movimento, che dipende dall'utente, va sommata la **traslazione in avanti** effettuata autonomamente dal programma se il valore di '**potenza reattore**' è maggiore di 0 (valore che può essere incrementato o decrementato, in una scala che va da 0 a 110, mediante i tasti + e -; il valore **massimo** è stato impostato a 110 per motivi 'storici', in quanto questa era la velocità raggiungibile con i **postbruciatori** accesi nella maggior parte dei **simulatori di volo**).

Provate, ad esempio, a **ruotare l'osservatore** verso il basso e a destra e a premere il tasto +.



Il flusso di esecuzione del programma è un po' particolare ma serve ad illustrare come **combinare Behavior** per l'**interazione** da tastiera (con animazioni proprie) e **thread concorrenti** (con un **thread** che può modificare alcune **proprietà dell'universo virtuale**).

SET TEXTURE MODE DI TEXTURE ATTRIBUTES

Tra gli **attributi** dell'**aspetto visivo** degli oggetti vi è *TextureAttributes*, che si occupa di definire in dettaglio alcuni **parametri** del processo di **texture mapping**.

Tali **parametri** sono:

- **Texture Mode** , che esamineremo successivamente;
- **Combine Mode** , che si occupa di definire l'operazione 'combine' quando il parametro precedente, *TextureMode*, è impostato, appunto, a 'COMBINE';
- **CombineColorSource, CombineColorFunction e CombineScaleFactor**, che si occupano di parametri supplementari legati sempre all'operazione 'combine';
- **Transform** , che specifica la trasformazione utilizzata per trasformare (traslare, ruotare, ridimensionare) le coordinate delle texture prima di applicare la texture ad un oggetto;
- **Blend Color** , che specifica un colore uniforme;
- **Perspective Correction** , che permette di scegliere il modello di correzione prospettica utilizzato;
- **Texture Color Table** , che definisce una look up table da impostare prima di applicare il texture mode.

In questa sezione prenderemo in esame solo il primo parametro, **TextureMode**.

TextureMode si occupa di definire **come verranno combinati i colori dell'oggetto e quelli della texture** da applicare all'oggetto.

Le **opzioni** possibili sono:

- **MODULATE** : combina il colore dell'oggetto con quello della texture;
- **DECAL** : la texture viene applicata sul colore dell'oggetto 'come un'etichetta adesiva' (decal); interessante per via della gestione implicita delle trasparenze eventualmente presenti nella texture;
- **BLEND** : combina il colore 'blend color' della texture con quello dell'oggetto;
- **REPLACE** : il colore dell'oggetto viene sostituito da quello della texture;

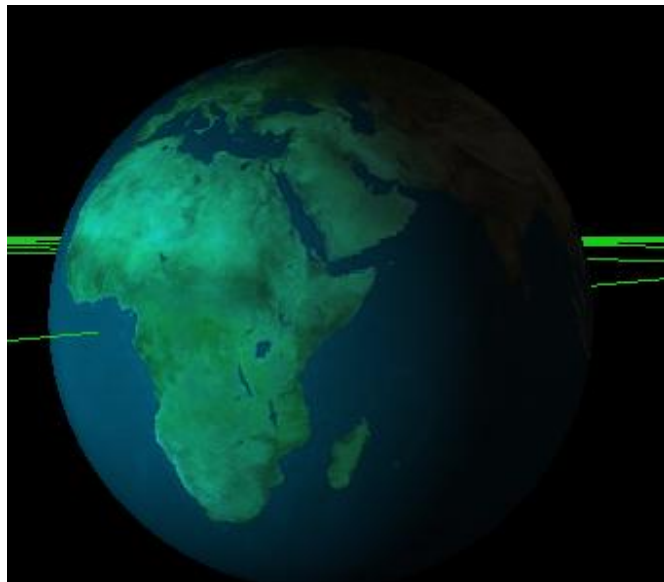
- **COMBINE** : elabora il colore dell'oggetto con il colore della texture o con il colore definito dal texture blend color, a seconda di quale metodo viene impostato nel parametro CombineMode.

Il più delle volte, la scelta ricade su **MODULATE**, specialmente per far sì che, in presenza di fonti luminose, le **texture** vengano ombreggiate.

E' questo l'effetto visibile **nell'esempio 'PianetaTerra'**, dove una sfera, texturizzata con un'immagine raffigurante la superficie terrestre (il lettore può reperire facilmente su Internet un'immagine da utilizzare per questo esempio, rinominandola in "Terra.jpg"), ruota intorno ad un punto dello **spazio virtuale** ove è posizionata una **fonte di luce PointLight**.

La sfera è dotata di un **Appearance** con **Material** (per la gestione delle **luci**) e **Texture** referenziante un oggetto di tipo *TextureAttributes*, con *TextureMode* impostato a **MODULATE**.

Per rendere leggermente visibile anche la parte non illuminata, impostare un colore (anche lieve: $(0.1f, 0.1f, 0.1f)$) per l'attributo **EmissiveColor** del **Material** associato alla sfera.



* * *
* * *

Appendice

In questa sezione sono riportati i **codici sorgente** degli **esempi** presentati nei vari capitoli.

I **commenti** alle porzioni di codice **comuni** a tutti gli esempi (recupero informazioni sistema, eventuale costruzione della griglia di riferimento, ...) spariscono dopo la loro prima apparizione, in modo da rendere **più leggibile** il codice.

Come anticipato nei vari capitoli, la definizione di certi **file-risorse** da utilizzare per alcuni esempi (**immagini** o file **OBJ**) è demandata al lettore, che comunque può **generarli facilmente** o reperirne alcuni gratuitamente dagli **archivi *Modelli 3D* e *Textures*** del **sito web dell'autore**, all'indirizzo www.redbaron85.com.

APPLICAZIONE JAVA 3D DI BASE

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

ROTAZIONI

FrecciaDestra : ruota a destra

FrecciaSinistra : ruota a sinistra

PagSu : ruota verso il basso

PagGiù : ruota verso l'alto

TRASLAZIONI

ALT + FrecciaSu : trasla in avanti

ALT + FrecciaGiù : trasla indietro

ALT + FrecciaDestra : trasla a destra

ALT + FrecciaSinistra : trasla a sinistra

ALT + PagSu : trasla verso il basso

ALT + PagGiù : trasla verso l'alto

*/

// Import di base

import java.awt.*;

import javax.swing.*;

import javax.vecmath.*;

import javax.media.j3d.*;

import com.sun.j3d.utils.universe.*;

import com.sun.j3d.utils.geometry.*;

import com.sun.j3d.utils.behaviors.keyboard.*;

public class ApplicazioneJava3Dbase extends JFrame

{

 public static void main(String[] args)

 {

 ApplicazioneJava3Dbase base = new ApplicazioneJava3Dbase();

 base.setVisible(true);

 }

 public ApplicazioneJava3Dbase()

 {

 // Impostazioni "classiche" di un JFrame...

 this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

 this.setLocation(100, 100);

 this.setSize(400, 400);

 this.setTitle("Applicazione Java3D di base; Milaneze Francesco; www.redbaron85.com");

 // Recupero le impostazioni grafiche del sistema

 GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();

 // Creo un oggetto Canvas3D, passandogli al costruttore le impostazioni grafiche del sistema

 Canvas3D canvas3D = new Canvas3D(config);

 // Aggiungo la Canvas3D al JFrame

 this.getContentPane().add(canvas3D, BorderLayout.CENTER);

 // Creo il Content Branch Graph

 BranchGroup scene = creaLoSceneGraph();

 // Creo un SimpleUniverse, passandogli come parametro la Canvas3D creata precedentemente

 SimpleUniverse simpleU = new SimpleUniverse(canvas3D);

```
// Imposto la BackClipDistance
simpleU.getViewer().getView().setBackClipDistance(10000);
// Imposto la FrontClipDistance
simpleU.getViewer().getView().setFrontClipDistance(0.1);
// Recupero il TransformGroup che controlla la ViewPlatformTransform,
// traslo il punto di osservazione iniziale leggermente indietro e
// associo, a tale TG, un KeyNavigatorBehavior, per "navigare" nello spazio virtuale con la tastiera
TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
Transform3D t3d = new Transform3D();
t3d.setTranslation(new Vector3f(0.0f,0.3f,0.0f));
vpTG.setTransform(t3d);
KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
scene.addChild(keyNavBeh);
// Compilo il ContentBranch Graph
scene.compile();
// Aggiungo al SimpleUniverse il ContentBranch Graph
simpleU.addBranchGraph(scene);
}

private BranchGroup creaLoSceneGraph()
{
    // Creo la radice del Content Branch Graph
    BranchGroup objRoot = new BranchGroup();
    // Aggiungo al Content Branch Graph un pavimento di riferimento
    objRoot.addChild(this.creaRiferimento());
    // Restituisco il tutto
    return objRoot;
}

private Shape3D creaRiferimento()
{
    // Creo una Geometry, di tipo LineArray
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    // Imposto un offset per le varie linee della griglia
    float l = -50.0f;
    // Creo le linee della griglia (in pratica, creo coppie di coordinate.
    //Ogni coppia di coordinate costituirà una linea della griglia).
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
        l += 10.0f; // Questo é l'offset vero e proprio: a ogni iterazione, il valore di l aumenta di 10 unità
    }
    // Creo un colore
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    // Coloro tutte le coordinate della griglia (ciò colorerà tutte le linee che collegano le coordinate)
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    // Restituisco un nuovo oggetto Shape3D, la cui geometria é il LineArray appena definito e colorato
    return new Shape3D(landGeom);
}
}
```

WITU.java

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
import java.awt.*;
import javax.swing.*;

public class WITU extends JPanel
{
    String LabelX, LabelY, LabelZ;
    double coordX, coordY, coordZ;
    int myX, myY, myZ;
    int sizeX;
    int sizeY;
    int xAxisSize;
    int yAxisSize;
    int zAxisSize;
    Color backgroundColor;
    Color originColor;
    Color xColor;
    Color zColor;
    Color yColor;
    Color targetColor;
    Color auxLabelsColor;
    int tmpX, tmpY, tmpZx, tmpZy;
    int boundSize;
    boolean auxLabelsBoolean;
    double axisAngle;

    public WITU() { this("", "", "", 0.0, 0.0, 0.0, 1000); }

    public WITU(String x, String y, String z) { this(x, y, z, 0.0, 0.0, 0.0, 1000); }

    public WITU(String x, String y, String z, double xc, double yc, double zc)
    {
        this(x, y, z, xc, yc, zc, 1000);
    }

    public WITU(String x, String y, String z, double xc, double yc, double zc, int aS)
    {
        boundSize = aS;
        LabelX = x;
        LabelY = y;
        LabelZ = z;
        sizeX = this.getWidth();
        sizeY = this.getHeight();
        xAxisSize = (int)(sizeX*8/10);
        yAxisSize = (int)(sizeY*8/10);
        zAxisSize = (int)(Math.sqrt((xAxisSize*xAxisSize) + (yAxisSize*yAxisSize)));
        coordX = xc;
        coordY = yc;
        coordZ = zc;
        myX = (int)((xAxisSize*coordX) / boundSize);
        myY = (int)((yAxisSize*coordY) / boundSize);
        myZ = (int)((zAxisSize*coordZ) / boundSize);
    }
}
```

```
this.setBackground(backgroundColor);
tmpX = 0;
tmpY = 0;
tmpZx = 0;
tmpZy = 0;
auxLabelsBoolean = true;
backgroundColor = new Color(0, 0, 0);
originColor = new Color(255, 255, 255);
xColor = new Color(255, 0, 0);
yColor = new Color(0, 255, 0);
zColor = new Color(0, 0, 255);
targetColor = new Color(125, 125, 125);
auxLabelsColor = new Color(255, 255, 0);
}
```

```
public void paintComponent(Graphics g)
{
```

```
    super.paintComponent(g);
    // RECUPERO DEI VALORI DI BASE
    sizeX = this.getWidth();
    sizeY = this.getHeight();
    xAxisSize = (int)(sizeX*8/10);
    yAxisSize = (int)(sizeY*8/10);
    zAxisSize = (int)(Math.sqrt((xAxisSize*xAxisSize) + (yAxisSize*yAxisSize)));
    myX = (int)((xAxisSize*coordX) / boundSize);
    myY = (int)((yAxisSize*coordY) / boundSize);
    myZ = (int)((zAxisSize*coordZ) / boundSize);
    // COLORE LO SFONDO
    this.setBackground(backgroundColor);
    // DISEGNO GLI ASSI (CON LE LORO LABEL)
    // Asse delle X (orizzontale)
    g.setColor(xColor);
    g.drawLine(sizeX/10, sizeY/2, sizeX-sizeX/10, sizeY/2);
    g.drawString(LabelX, sizeX-sizeX/15, sizeY/2);
    // Asse delle Y (verticale)
    g.setColor(yColor);
    g.drawLine(sizeX/2, sizeY/10, sizeX/2, sizeY-sizeY/10);
    g.drawString(LabelY, sizeX/2, sizeY/15);
    // Asse delle Z (obliquo)
    g.setColor(zColor);
    g.drawLine(sizeX/10, sizeY-sizeY/10, sizeX-sizeX/10, sizeY/10);
    g.drawString(LabelZ, sizeX/20, sizeY-sizeY/20);
    // DISEGNO L'ORIGINE
    g.setColor(originColor); // Disegno l'origine, al centro (bianca), dimensioni: 1/40 della dimensione del
```

pannello

```
    g.fillOval(sizeX/2-(sizeX/40)/2, sizeY/2-(sizeY/40)/2, sizeX/40, sizeY/40);
    // DISEGNO IL PUNTO CHE VAGA NELL'UNIVERSO 3D (solo se rientra nei bounds)
    if((Math.abs(coordX)<boundSize/2) && (Math.abs(coordY)<boundSize/2) &&
```

(Math.abs(coordZ)<boundSize/2))

```
    {
        g.setColor(targetColor);
        // Ricavo il tmpX
        tmpX = (sizeX/2) + myX;
        // Ricavo il tmpY
        tmpY = (sizeY/2) - myY;
        // Disegno i due vettori paralleli sul piano XY
```

```
g.drawLine(tmpX, sizeY/2, tmpX, tmpY);
g.drawLine(sizeX/2, tmpY, tmpX, tmpY);
// Ricavo il tmpZ, cio? la coordinata della proiezione del punto sull'asse Z... tmpZ ? fatto di due
coordinate: tmpZx e tmpZy...
axisAngle = Math.atan((double)(sizeY)/(double)(sizeX)); // In radianti
tmpZx = (int)(sizeX/2 - (myZ*Math.cos(axisAngle)));
tmpZy = (int)(sizeY/2 + (myZ*Math.sin(axisAngle)));
// Disegno i due vettori sul piano XZ
g.drawLine(tmpZx, tmpZy, tmpZx + myX, tmpZy);
g.drawLine(tmpX, sizeY/2, tmpZx + myX, tmpZy);
// Disegno i due vettori sul piano YZ
g.drawLine(tmpX, tmpY, tmpZx + myX, tmpY + (tmpZy - sizeY/2));
g.drawLine(tmpZx + myX, tmpZy, tmpZx + myX, tmpY + (tmpZy - sizeY/2));
// Disegno il target
g.fillOval(tmpZx + myX - sizeX/80, tmpY + (tmpZy - sizeY/2) - sizeY/80, sizeX/40, sizeY/40);
}
// DISEGNO LE SCRITTE AUSILIARIE
g.setColor(auxLabelsColor);
g.drawString("Bound: [-" + boundSize/2 + ", +" + boundSize/2 + "]", 10, 10);
g.drawString("X: " + coordX, 10, 30);
g.drawString("Y: " + coordY, 10, 50);
g.drawString("Z: " + coordZ, 10, 70);
if((Math.abs(coordX)>boundSize/2) || (Math.abs(coordY)>boundSize/2) ||
(Math.abs(coordZ)>boundSize/2)) g.drawString("Out of bounds !", 10, 90);
//else g.drawString("sizeX: " + sizeX + ", sizeY: " + sizeY + ", angolo: " + axisAngle, 10, sizeY-
(sizeY/40+10));
}
```

```
public String getLabelX() { return LabelX; }

public String getLabelY() { return LabelY; }

public String getLabelZ() { return LabelZ; }

public double getCoordX() { return coordX; }

public double getCoordY() { return coordY; }

public double getCoordZ() { return coordZ; }

public void setLabelX(String in)
{
    LabelX = in;
    repaint();
}

public void setLabelY(String in)
{
    LabelY = in;
    repaint();
}

public void setLabelZ(String in)
{
    LabelZ = in;
    repaint();
}
```

```
}

public void setCoordX(double in)
{
    coordX = in;
    repaint();
}

public void setCoordY(double in)
{
    coordY = in;
    repaint();
}

public void setCoordZ(double in)
{
    coordZ = in;
    repaint();
}

public int getBoundSize() { return boundSize; }

public void setBoundSize(int aS)
{
    boundSize = aS;
    repaint();
}

public boolean getAuxLabelsBoolean() { return auxLabelsBoolean; }

public void setAuxLabelsBoolean(boolean b) { auxLabelsBoolean = b; }

public void setOriginColor(int r, int g, int b)
{
    originColor = new Color(r, g, b);
    repaint();
}

public Color getOriginColor(){ return originColor; }

public void setXColor(int r, int g, int b)
{
    xColor = new Color(r, g, b);
    repaint();
}

public Color getXColor() { return xColor; }

public void setYColor(int r, int g, int b)
{
    yColor = new Color(r, g, b);
    repaint();
}

public Color getYColor() { return yColor; }

public void setZColor(int r, int g, int b)
```

```
{
    zColor = new Color(r, g, b);
    repaint();
}

public Color getZColor() { return xColor; }

public void setBackgroundColor(int r, int g, int b)
{
    backgroundColor = new Color(r, g, b);
    repaint();
}

public Color getBackgroundColor() { return backgroundColor; }

public void setAuxLabelsColor(int r, int g, int b)
{
    auxLabelsColor = new Color(r, g, b);
    repaint();
}

public Color getAuxLabelsColor() { return auxLabelsColor; }
}
```

WITUapp.java

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
import java.awt.*;
import javax.swing.*;

public class WITUapp extends JFrame
{
    public static WITU myWITU;

    public static void main(String[] args){
        WITUapp myApp = new WITUapp();
        myApp.setVisible(true);
        doTest();
    }

    public WITUapp(){
        // Inizializzazioni "classiche" di un frame...
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setTitle("WITU test application, Milanese Francesco, www.redbaron85.com");
        this.setSize(400, 400);
        this.setLocation(10, 10);
        // Aggiunta del pannello WITU
        myWITU = new WITU("X", "Y", "Z", -500.0, 100.0, -400.0, 2000);
        this.getContentPane().add(myWITU, BorderLayout.CENTER);
    }

    public static void doTest(){
        // Simulazioni
        try
        {
            for(int i=0; i<30; i++){
                Thread.sleep(100);
                myWITU.setCoordX(myWITU.getCoordX()+10);
                myWITU.repaint();
            }
            for(int i=0; i<50; i++){
                Thread.sleep(100);
                myWITU.setCoordY(myWITU.getCoordY()+10);
                myWITU.repaint();
            }
            for(int i=0; i<50; i++){
                Thread.sleep(100);
                myWITU.setCoordZ(myWITU.getCoordZ()+10);
                myWITU.repaint();
            }
            for(int i=0; i<50; i++){
                Thread.sleep(100);
                myWITU.setCoordY(myWITU.getCoordY()-10);
                myWITU.repaint();
            }
        }
        catch(Exception e){System.out.println(e);}
    }
}
```

PRIMO ESEMPIO

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra la Canvas3D con un Simple Universe.

*/

// Import necessari per gestire JFrame, JPanel, il menù, il SimpleUniverse

// con oggetti di base e vecmath per i vettori necessari per la definizione delle trasformazioni 3D.

import java.awt.*;

import javax.swing.*;

import javax.vecmath.*;

import javax.media.j3d.*;

import com.sun.j3d.utils.universe.*;

import com.sun.j3d.utils.geometry.*;

// La classe eredita da JFrame: è un JFrame che conterrà la Canvas3D e un menù.

public class PrimoEsempio extends JFrame

{

 // Il main non fa altro che istanziare un oggetto PrimoEsempio e mostrarlo a video.

 public static void main(String [] args)

 {

 PrimoEsempio pe = new PrimoEsempio();

 pe.setVisible(true);

 }

 // Ecco il costruttore di PrimoEsempio, che eredita da JFrame...

 public PrimoEsempio()

 {

 // ... per cui dobbiamo occuparci delle inizializzazioni classiche di un frame !

 this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

 this.setSize(300, 300);

 this.setLocation(100, 100);

 this.setTitle("Java 3D, primo esempio. Milanese Francesco, www.redbaron85.com");

 // Adesso creo il simple universe, la canvas3d e mostro il tutto inserendolo nel frame

 //Recupera le configurazioni grafiche del computer.

 GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();

 //Crea il Canvas3D, passando le configurazioni grafiche del computer al costruttore.

 Canvas3D c3D = new Canvas3D(config);

 //Crea il SimpleUniverse, passando la Canvas3D appena creata al costruttore.

 SimpleUniverse sU = new SimpleUniverse(c3D);

 // Creiamo una scena di base con un metodo nostro e la compiliamo.

 BranchGroup scena = creaLoSceneGraph();

 scena.compile();

 //Aggiunge la scena all'universo

 sU.addBranchGraph(scena);

 // La Canvas va quindi aggiunta al Frame dell'applicazione (eredita da AWT Component, come un

JPanel...);

 // ovviamente, this "vale" solo se tale riga è scritta all'interno di un JFrame...

 this.getContentPane().add(c3D, BorderLayout.CENTER);

 // Il SimpleUniverse e la Canvas3D sono stati creati; la Canvas3D è stata inoltre

 // aggiunta al centro del frame PrimoEsempio !

 }

 // Il metodo seguente crea una scena di base (un oggetto BranchGroup) --- in questo primo esempio, vuota (non vedremo nulla !)

```
public BranchGroup creaLoSceneGraph()
{
    // Creiamo la radice dello Scene Graph: è un oggetto BranchGroup
    BranchGroup objRoot = new BranchGroup();
    // ... e lo restituiamo subito: non lo stiamo popolando !
    return objRoot;
}
}
```

SECONDO ESEMPIO

```
// Milanesi Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*
Questo esempio mostra la Canvas3D con un Simple Universe e come inserire
correttamente elementi Swing, come un menù, in un Frame provvisto di Canvas3D
*/

// Import necessari per gestire JFrame, JPanel, il menù, il SimpleUniverse con oggetti di base
//e vecmath per i vettori necessari per la definizione delle trasformazioni 3D.
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;

// La classe eredita da JFrame: è un JFrame che conterrà la Canvas3D e un menù.
public class SecondoEsempio extends JFrame
{
    // Il main non fa altro che istanziare un oggetto PrimoEsempio e mostrarlo a video.
    public static void main(String [] args)
    {
        SecondoEsempio se = new SecondoEsempio();
        se.setVisible(true);
    }

    // Ecco il costruttore di PrimoEsempio, che eredita da JFrame...
    public SecondoEsempio()
    {
        // ... per cui dobbiamo occuparci delle inizializzazioni classiche di un frame !
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(300, 300);
        this.setLocation(100, 100);
        this.setTitle("Java 3D, secondo esempio. Milanesi Francesco, www.redbaron85.com");
        // Adesso creo il simple universe, la canvas3d e mostro il tutto inserendolo nel frame
        //Recupera le configurazioni grafiche del computer.
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        //Crea il Canvas3D, passando le configurazioni grafiche del computer al costruttore.
        Canvas3D c3D = new Canvas3D(config);
        //Crea il SimpleUniverse, passando la Canvas3D appena creata al costruttore.
        SimpleUniverse sU = new SimpleUniverse(c3D);
        // Creiamo una scena di base con un metodo nostro (vd. slide successiva) e la compiliamo.
        BranchGroup scena = creaLoSceneGraph();
        scena.compile();
        //Aggiunge la scena all'universo
        sU.addBranchGraph(scena);
        // La Canvas va quindi aggiunta al Frame dell'applicazione (eredita da AWT Component, come un
JPanel...);
        // ovviamente, this "vale" solo se tale riga è scritta all'interno di un JFrame...
        this.getContentPane().add(c3D, BorderLayout.CENTER);
        // Il SimpleUniverse e la Canvas3D sono stati creati;
        // la Canvas3D è stata inoltre aggiunta al centro del frame PrimoEsempio !
        // Aggiungiamo ora un semplice menù a PrimoEsempio...
        // ATTENZIONE ! Un Menu ha problemi di visibilità, in quanto la Canvas3D, che è una componente
AWT "heavyweight",
```

```
// copre le componenti Swing "lightweight", come le voci del menù !
// Bisogna quindi aggiungere la seguente riga di codice (commentarla, ricompilare
// ed eseguire il tutto per vedere cosa succede senza tale comando):
JPopupMenu.setDefaultLightWeightPopupEnabled(false);
// Ecco, ora possiamo definire ed aggiungere il nostro menù:
JMenuBar barraMenu = new JMenuBar();
JMenu Menu1 = new JMenu("Menu1");
JMenuItem Etichetta1 = new JMenuItem("Etichetta1");
Menu1.add(Etichetta1);
barraMenu.add(Menu1);
this.setJMenuBar(barraMenu);
}

// Il metodo seguente crea una scena di base (un oggetto BranchGroup), aggiungendogli come figlio
// un cubo colorato (oggetto Java3D di base) spostato e ruotato
public BranchGroup creaLoSceneGraph()
{
    // Creiamo la radice dello Scene Graph: è un oggetto BranchGroup
    BranchGroup objRoot = new BranchGroup();
    // Creiamo dei transform group, poi creiamo un cubo colorato, aggiungiamo il cubo ai transform group e
    // aggiungiamo tali transform group a objRoot, cioè la radice del BranchGroup
    // La "catena" è quindi: BranchGroup - TransformGroup traslazione - TransformGroup rotazione
    // (comprenderà due rotazioni) - Oggetto.
    // Adesso ci occupiamo della traslazione... senza traslazione, ci troveremmo "dentro" il cubo
    // (ogni oggetto creato ha come coordinata di default l'origine) e non riusciremmo a vedere niente !
    Vector3f vettoreTraslazione = new Vector3f(0.0f, 0.0f, -5.0f);
    Transform3D t3d = new Transform3D();
    t3d.setTranslation(vettoreTraslazione);
    // Ecco il nostro TG della rotazione:
    TransformGroup tgTraslazione = new TransformGroup(t3d);
    // Creo una prima trasformazione "rotazione"...
    Transform3D rotazione1 = new Transform3D();
    rotazione1.rotY(Math.PI/4.0);
    // Creo una seconda trasformazione "rotazione"...
    Transform3D rotazione2 = new Transform3D();
    rotazione2.rotX(Math.PI/5.0);
    // Compongo le due rotazioni mettendo il risultato in rotazione2.
    rotazione2.mul(rotazione1);
    // Ecco il nostro TG per le due rotazioni:
    TransformGroup tgRotazione = new TransformGroup(rotazione2);
    // Aggiungo al TG della traslazione un figlio "particolare": il TG delle rotazioni !
    tgTraslazione.addChild(tgRotazione);
    // Adesso aggiungo al TG delle rotazioni un oggetto ColorCube (oggetto "nativo" di Java3D) di lato 1.0.
    tgRotazione.addChild(new ColorCube(1.0));
    // Aggiungo il TG della traslazione, contenente il TG delle rotazioni l'oggetto ColorCube,
    // alla radice di questo branch graph, ossia a objRoot.
    // In generale, un qualunque oggetto o gruppo va aggiunto allo scene graph aggiungendolo ad un
    BranchGroup,
    // cosa che va fatta con il metodo addChild([figlio]) della radice del Branch Group
    objRoot.addChild(tgTraslazione);

    // Restituisco, infine, la radice del branch graph appena creato; restituendo la radice, restituisco in realtà
    // TUTTO il branch graph (trasformazioni incluse), che verrà quindi aggiunto alla scena corrente.
    return objRoot;
}
}
```

NOMINAL VIEWING PLATFORM

```
// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*
Questo esempio mostra come utilizzare setNominalViewingPlatform
per spostare leggermente indietro il punto di osservazione della ViewPlatform
*/

// Import necessari per gestire JFrame, JPanel, il menù, il SimpleUniverse con oggetti di base
// e vecmath per i vettori necessari per la definizione delle trasformazioni 3D.
import java.awt.*;
import javax.swing.*;
// import javax.vecmath.*; // Non serve qui
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;

// La classe eredita da JFrame: é un JFrame che conterrà la Canvas3D e un menù.
public class NominalViewingPlatform extends JFrame
{
    // Il main non fa altro che istanziare un oggetto NominalViewingPlatform e mostrarlo a video.
    public static void main(String [] args)
    {
        NominalViewingPlatform nwp = new NominalViewingPlatform();
        nwp.setVisible(true);
    }

    // Ecco il costruttore di NominalViewingPlatform, che eredita da JFrame...
    public NominalViewingPlatform()
    {
        // ... per cui dobbiamo occuparci delle inizializzazioni classiche di un frame !
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(300, 300);
        this.setLocation(100, 100);
        this.setTitle("Java 3D, esempio NominalViewingPlatform. Milaneze Francesco, www.redbaron85.co");
        // Adesso creo il simple universe, la canvas3d e mostro il tutto inserendolo nel frame
        //Recupera le configurazioni grafiche del computer.
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        //Crea il Canvas3D, passando le configurazioni grafiche del computer al costruttore.
        Canvas3D c3D = new Canvas3D(config);
        //Crea il SimpleUniverse, passando la Canvas3D appena creata al costruttore.
        SimpleUniverse sU = new SimpleUniverse(c3D);
        //Con "setNominalViewingPlatform()", spostiamo leggermente indietro
        // il sistema di riferimento nel nostro SimpleUniverse
        sU.getViewingPlatform().setNominalViewingTransform();
        // Creiamo una scena di base con un metodo nostro (vd. slide successiva) e la compiliamo.
        BranchGroup scena = creaLoSceneGraph();
        scena.compile();
        //Aggiunge la scena all'universo
        sU.addBranchGraph(scena);
        // La Canvas va quindi aggiunta al Frame dell'applicazione (eredita da AWT Component, come un
        JPanel...);

        // ovviamente, this "vale" solo se tale riga è scritta all'interno di un JFrame...
        this.getContentPane().add(c3D, BorderLayout.CENTER);
    }
}
```

```
// Il SimpleUniverse e la Canvas3D sono stati creati; la Canvas3D é stata inoltre aggiunta al centro del
frame PrimoEsempio !
// Aggiungiamo ora un semplice menù a PrimoEsempio...
// ATTENZIONE ! Un Menu ha problemi di visibilità, in quanto la Canvas3D, che é una componente
AWT "heavyweight",
// copre le componenti Swing "lightweight", come le voci del menù !
// Bisogna quindi aggiungere la seguente riga di codice (commentarla, ricompilare ed eseguire il tutto
// per vedere cosa succede senza tale comando):
JPopupMenu.setDefaultLightWeightPopupEnabled(false);
// Ecco, ora possiamo definire ed aggiungere il nostro menù:
JMenuBar barraMenu = new JMenuBar();
JMenu Menu1 = new JMenu("Menu1");
JMenuItem Etichetta1 = new JMenuItem("Etichetta1");
Menu1.add(Etichetta1);
barraMenu.add(Menu1);
this.setJMenuBar(barraMenu);
}

// Il metodo seguente crea una scena di base (un oggetto BranchGroup), aggiungendogli come figlio
//un cubo colorato (oggetto Java3D di base) ruotato, ma non spostato
public BranchGroup creaLoSceneGraph()
{
    // Creiamo la radice dello Scene Graph: è un oggetto BranchGroup
    BranchGroup objRoot = new BranchGroup();
    // Creiamo dei transform group, poi creiamo un cubo colorato, aggiungiamo il cubo al transform group
    // e aggiungiamo il transform group a objRoot, cioè la radice del BranchGroup
    // La "catena" é quindi: BranchGroup - TransformGroup rotazione (comprenderà due rotazioni) - Oggetto.
    // Creo una prima trasformazione "rotazione"...
    Transform3D rotazione1 = new Transform3D();
    rotazione1.rotY(Math.PI/4.0);
    // Creo una seconda trasformazione "rotazione"...
    Transform3D rotazione2 = new Transform3D();
    rotazione2.rotX(Math.PI/5.0);
    // Compongo le due rotazioni mettendo il risultato in rotazione2.
    rotazione2.mul(rotazione1);
    // Ecco il nostro TG per le due rotazioni:
    TransformGroup tgRotazione = new TransformGroup(rotazione2);
    // Adesso aggiungo al TG delle rotazioni un oggetto ColorCube (oggetto "nativo" di Java3D) di lato 0.4
    // (setNominalViewingPlatform ci sposta UN PO' indietro, ma non abbastanza da vedere "bene" un cubo di
lato 1 !)
    tgRotazione.addChild(new ColorCube(0.4));
    // Aggiungo il TG della rotazione, contenente l'oggetto ColorCube, alla radice di questo branch graph,
ossia a objRoot.
    // In generale, un qualunque oggetto o gruppo va aggiunto allo scene graph aggiungendolo ad un
BranchGroup,
    //cosa che va fatta con il metodo addChild([figlio]) della radice del Branch Group
    objRoot.addChild(tgRotazione);
    // Restituisco, infine, la radice del branch graph appena creato; restituendo la radice, restituisco in realtà
    //TUTTO il branch graph (trasformazioni incluse), che verrà quindi aggiunto alla scena corrente.
    return objRoot;
}
}
```

ESEMPIO SIMPLE UNIVERSE

```
// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*
Questo esempio mostra a video un Simple Universe e stampa sull'output
della "console" alcune informazioni sull'esecuzione
*/

// Import necessari per gestire JFrame, JPanel, il menù, il SimpleUniverse con oggetti di base
// e vecmath per i vettori necessari per la definizione delle trasformazioni 3D.
import java.awt.*;
import javax.swing.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import java.util.Enumeraion;

// La classe eredita da JFrame: é un JFrame che conterrà la Canvas3D e un menù.
public class EsempioSimpleUniverse extends JFrame
{
    // Il main non fa altro che istanziare un oggetto EsempioSimpleUniverse e mostrarlo a video.
    public static void main(String [] args)
    {
        EsempioSimpleUniverse esu = new EsempioSimpleUniverse();
        esu.setVisible(true);
    }

    // Ecco il costruttore di EsempioSimpleUniverse, che eredita da JFrame...
    public EsempioSimpleUniverse()
    {
        // ... per cui dobbiamo occuparci delle inizializzazioni classiche di un frame !
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(300, 300);
        this.setLocation(100, 100);
        this.setTitle("Java 3D, esempio SimpleUniverse. Milaneze Francesco, www.redbaron85.com");
        // Adesso creo il simple universe, la canvas3d e mostro il tutto inserendolo nel frame
        //Recupera le configurazioni grafiche del computer.
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        //Crea il Canvas3D, passando le configurazioni grafiche del computer al costruttore.
        Canvas3D c3D = new Canvas3D(config);
        //Crea il SimpleUniverse, passando la Canvas3D appena creata al costruttore.
        SimpleUniverse sU = new SimpleUniverse(c3D);
        //Con "setNominalViewingPlatform()", spostiamo leggermente indietro
        //il sistema di riferimento nel nostro SimpleUniverse
        sU.getViewingPlatform().setNominalViewingTransform();
        // Creiamo una scena di base con un metodo nostro (vd. slide successiva) e la compiliamo.
        BranchGroup scena = creaLoSceneGraph();
        scena.compile();
        //Aggiunge la scena all'universo
        sU.addBranchGraph(scena);
        // La Canvas va quindi aggiunta al Frame dell'applicazione (eredita da AWT Component, come un
        JPanel...);
        // ovviamente, this "vale" solo se tale riga è scritta all'interno di un JFrame...
        this.getContentPane().add(c3D, BorderLayout.CENTER);
        // Il SimpleUniverse e la Canvas3D sono stati creati; la Canvas3D é stata inoltre aggiunta al centro del
        frame PrimoEsempio !
    }
}
```

```
// Aggiungiamo ora un semplice menù a PrimoEsempio...
// ATTENZIONE ! Un Menu ha problemi di visibilità, in quanto la Canvas3D, che é una componente
AWT "heavyweight",
// copre le componenti Swing "lightweight", come le voci del menù !
// Bisogna quindi aggiungere la seguente riga di codice (commentarla, ricompilare ed eseguire il tutto
// per vedere cosa succede senza tale comando):
JPopupMenu.setDefaultLightWeightPopupEnabled(false);
// Ecco, ora possiamo definire ed aggiungere il nostro menù:
JMenuBar barraMenu = new JMenuBar();
JMenu Menu1 = new JMenu("Menu1");
JMenuItem Etichetta1 = new JMenuItem("Etichetta1");
Menu1.add(Etichetta1);
barraMenu.add(Menu1);
this.setJMenuBar(barraMenu);

//
=====
// Alcune operazioni che riguardano il SimpleUniverse, il Locale e il Clipping
// toString di PreferredConfiguration, Viewer e Viewing Platform di questo Simple Universe
System.out.println("toString della PreferredConfiguration di questo SimpleUniverse: " +
sU.getPreferredConfiguration().toString());
System.out.println("toString del Viewer di questo SimpleUniverse: " + sU.getViewer().toString());
System.out.println("toString del Viewing Platform di questo SimpleUniverse: " +
sU.getViewingPlatform().toString());
// Operazioni sul sistema delle HiRes Coords del Locale
HiResCoord hrc = new HiResCoord(); // Creo un oggetto HiResCoord, che conterrà le HiResCoord di
questo Locale
sU.getLocale().getHiRes(hrc); // Passo le HiResCoord del Locale all'oggetto hrc precedentemente creato
int [] X = new int[8];
int [] Y = new int[8];
int [] Z = new int[8];
hrc.getHiResCoordX(X);
hrc.getHiResCoordY(Y);
hrc.getHiResCoordZ(Z);
System.out.println("Stampiamo a video i valori HiResCoord del Locale corrente: ");
System.out.print("Gli 8 int dell'array di interi HiResCoordX sono: ");
for(int i=0; i<8; i++) System.out.print(" " + X[i] + " ");
System.out.println("");
System.out.print("Gli 8 int dell'array di interi HiResCoordY sono: ");
for(int i=0; i<8; i++) System.out.print(" " + Y[i] + " ");
System.out.println("");
System.out.print("Gli 8 int dell'array di interi HiResCoordZ sono: ");
for(int i=0; i<8; i++) System.out.print(" " + Z[i] + " ");
System.out.println("");
// Quanti Branch Graphs ci sono associati a questo Locale ?
System.out.println("Numero di Branch Graph associati a questo Locale: " +
sU.getLocale().numBranchGraphs());
// E quali sono questi Branch Graph ? "Enumeriamoli" !!!
System.out.println("Adesso \"enumererò\" i BranchGraph associati al Locale corrente: ");
for (Enumeration e = sU.getLocale().getAllBranchGraphs(); e.hasMoreElements() ;)
System.out.println(e.nextElement());
System.out.println("Fine dell'operazione di \"enumerazione\" dei Branch Graph !");
// Operazioni sul clipping
System.out.println("Valore di Back Clip Distance (default di Java 3D): " +
sU.getViewer().getView().getBackClipDistance());
```

```
        System.out.println("Valore di Front Clip Distance (default di Java 3D): " +
sU.getViewer().getView().getFrontClipDistance());
        System.out.println("Ora imposto a 2 il valore di Back Clip Distance e a 1 il valore di Front Clip
Distance...");
        sU.getViewer().getView().setBackClipDistance(2.0);
        sU.getViewer().getView().setFrontClipDistance(1.0); // Per tornare a vedere il cubo, commentare questa
linea
        System.out.println("Nuovo valore di Back Clip Distance (mio): " +
sU.getViewer().getView().getBackClipDistance());
        System.out.println("Nuovo valore di Front Clip Distance (mio): " +
sU.getViewer().getView().getFrontClipDistance());
        System.out.println("...logicamente, non vediamo il cubo, che ha lato 0.4, é al centro della scena e ha le
normali positive rivolte verso l'esterno !");
        //
=====
}

// Il metodo seguente crea una scena di base (un oggetto BranchGroup), aggiungendogli come figlio
// un cubo colorato (oggetto Java3D di base) ruotato, ma non spostato
public BranchGroup creaLoSceneGraph()
{
    // Creiamo la radice dello Scene Graph: è un oggetto BranchGroup
    BranchGroup objRoot = new BranchGroup();
    // Creiamo dei transform group, poi creiamo un cubo colorato, aggiungiamo il cubo al transform group
    // e aggiungiamo il transform group a objRoot, cioè la radice del BranchGroup
    // La "catena" é quindi: BranchGroup - TransformGroup rotazione (comprenderà due rotazioni) - Oggetto.
    // Creo una prima trasformazione "rotazione"...
    Transform3D rotazione1 = new Transform3D();
    rotazione1.rotY(Math.PI/4.0);
    // Creo una seconda trasformazione "rotazione"...
    Transform3D rotazione2 = new Transform3D();
    rotazione2.rotX(Math.PI/5.0);
    // Compongo le due rotazioni mettendo il risultato in rotazione2.
    rotazione2.mul(rotazione1);
    // Ecco il nostro TG per le due rotazioni:
    TransformGroup tgRotazione = new TransformGroup(rotazione2);
    // Adesso aggiungo al TG delle rotazioni un oggetto ColorCube (oggetto "nativo" di Java3D) di lato 0.4
    // (setNominalViewingPlatform ci sposta UN PO' indietro, ma non abbastanza da vedere "bene" un cubo
di lato 1 !)
    tgRotazione.addChild(new ColorCube(0.4));
    // Aggiungo il TG della rotazione, contenente l'oggetto ColorCube, alla radice di questo branch graph,
ossia a objRoot.
    // In generale, un qualunque oggetto o gruppo va aggiunto allo scene graph aggiungendolo ad un
BranchGroup,
    // cosa che va fatta con il metodo addChild([figlio]) della radice del Branch Group
    objRoot.addChild(tgRotazione);
    // Restituisco, infine, la radice del branch graph appena creato; restituendo la radice, restituisco in realtà
    // TUTTO il branch graph (trasformazioni incluse), che verrà quindi aggiunto alla scena corrente.
    return objRoot;
}
}
```

ESEMPIO BACKGROUND

```
// Milanesi Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*
Questo esempio mostra come inserire una immagine nel BackGround dell'Universo Virtuale
*/

// Import necessari per gestire JFrame, JPanel, il menù, il SimpleUniverse e j3d.geometry per gli oggetti di base.
import java.awt.*;
import javax.swing.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.image.*;

// Questo esempio non contiene niente, se non un background diverso dal solito sfondo nero...
public class EsempioBackground extends JFrame
{
    // Main dell'applicazione: crea un EsempioBackground e lo mostra a video
    public static void main(String[] args)
    {
        EsempioBackground eb = new EsempioBackground();
        eb.setVisible(true);
    }

    // Impostazioni di base di EsempioBackground, che eredita da JFrame...
    public EsempioBackground()
    {
        // ... per cui dobbiamo occuparci delle inizializzazioni classiche di un frame !
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(300, 300);
        setLocation(100, 100);
        setTitle("Java 3D, esempio background. Milanesi Francesco, www.redbaron85.com");
        // Adesso creo il simple universe, la canvas3d e mostro il tutto inserendolo nel frame
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D c3D = new Canvas3D(config);
        SimpleUniverse sU = new SimpleUniverse(c3D);
        sU.getViewingPlatform().setNominalViewingTransform();
        BranchGroup scena = creaLoSceneGraph();
        scena.compile();
        sU.addBranchGraph(scena);
        this.getContentPane().add(c3D, BorderLayout.CENTER);
        // Il SimpleUniverse e la Canvas3D sono stati creati; la Canvas3D é stata inoltre
        // aggiunta al centro del frame PrimoEsempio !
    }

    // Crea la scena, inserendo un nodo background, creato in un metodo a parte
    //(tecnica consigliata per creare scene con MOLTI oggetti... qui in teoria non ce ne sarebbe bisogno)
    private BranchGroup creaLoSceneGraph()
    {
        // Creo la radice del Branch Graph
        BranchGroup objRoot = new BranchGroup();
        // Aggiungiamo lo sfondo, creato in un metodo a parte
        objRoot.addChild(this.createBackground());
        // Restituisco il tutto
    }
}
```

```
        return objRoot;
    }

    // Creo uno sfondo, utilizzando un'immagine
    private BranchGroup createBackground()
    {
        // Creiamo un oggetto Background, associandogli un bound di "operatività", centrato nell'origine e di
raggio 200.
        Background back = new Background();
        back.setApplicationBounds(new BoundingSphere(new Point3d(0.0,0.0,0.0), 200.0));
        // Un Background ha inoltre bisogno di una geometria (una sfera), quindi ora creiamo
        //una sfera con texture, poi la associamo al Background...

        // A questo punto dobbiamo creare una sfera, di raggio unitario e con le normali delle facce
        // rivolte verso "dentro", che rappresenterà il nostro mondo.
        // La normale é il vettore perpendicolare ad una faccia. Le normali possono essere impostate
        // come uscenti (ad esempio, una cassa di legno) o verso l'interno (é il nostro caso:
        // sono le pareti INTERNE della sfera a dover essere dipinte !)
        // Prima di procedere alla creazione della sfera, creiamo il suo aspetto (Appearance),
        // caricando un'immagine di sfondo in un oggetto di tipo TEXTURE
        // Quindi:
        // TEXTURE
        Texture texture = new
TextureLoader(EsempioBackground.class.getResource("immagineBackground.jpg"), this).getTexture();
        // APPEARANCE, dalla Texture
        Appearance app = new Appearance();
        app.setTexture(texture);
        // SFERA CON APPEARANCE (e quindi Texture)
        Sphere sphere = new Sphere(1.0f,
Primitive.GENERATE_TEXTURE_COORDS|Primitive.GENERATE_NORMALS_INWARD, app);
        // BRANCH GROUP CHE CONTIENE LA SFERA
        BranchGroup bgGeometry = new BranchGroup();
        bgGeometry.addChild(sphere);
        // A questo punto aggiungo il branch group contenente la sfera texturizzata
        //all'oggetto Background precedentemente creato
        back.setGeometry(bgGeometry);
        // Creo quindi un nuovo Branch Group e gli assegno, come figlio, il Background così composto;
        //infine, restituisco tale Branch Group.
        BranchGroup backgroundGroup = new BranchGroup();
        backgroundGroup.addChild(back);
        return backgroundGroup;
    }
}
```

BACKGROUND CREATO CON SFERA ENORME

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come creare un "finto" background creando una sfera enorme, impostandole una texture e le normali verso l'interno.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale. I comandi per navigare nello Spazio Virtuale con tale Behavior sono:

ROTAZIONI

FrecciaDestra : ruota a destra

FrecciaSinistra : ruota a sinistra

PagSu : ruota verso il basso

PagGiù : ruota verso l'alto

TRASLAZIONI

ALT + FrecciaSu : trasla in avanti

ALT + FrecciaGiù : trasla indietro

ALT + FrecciaDestra : trasla a destra

ALT + FrecciaSinistra : trasla a sinistra

ALT + PagSu : trasla verso il basso

ALT + PagGiù : trasla verso l'alto

*/

// Import necessari per gestire JFrame, JPanel, il menù, il SimpleUniverse e j3d.geometry per gli oggetti di base.

import java.awt.*;

import javax.swing.*;

import javax.vecmath.*;

import javax.media.j3d.*;

import com.sun.j3d.utils.universe.*;

import com.sun.j3d.utils.geometry.*;

import com.sun.j3d.utils.behaviors.keyboard.*;

import com.sun.j3d.utils.image.*;

public class BackgroundCreatoConSferaEnorme extends JFrame

{

 public static void main(String[] args)

 {

 BackgroundCreatoConSferaEnorme bccse = new BackgroundCreatoConSferaEnorme();

 bccse.setVisible(true);

 }

 public BackgroundCreatoConSferaEnorme()

 {

 // Impostazioni "classiche" di un JFrame...

 this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

 this.setLocation(100, 100);

 this.setSize(400, 400);

 this.setTitle("Finto Background sfera enorme; Milaneze Francesco; www.redbaron85.com");

 // Creazione di Simple Universe, Canvas3D, Content Branch Graph (via chiamata a metodo esterno)

 //e behavior nativo per la navigazione coi tasti da tastiera

 GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();

 Canvas3D canvas3D = new Canvas3D(config);

 this.getContentPane().add(canvas3D, BorderLayout.CENTER);

 BranchGroup scene = createSceneGraph();

 SimpleUniverse simpleU = new SimpleUniverse(canvas3D);

```
simpleU.getViewer().getView().setBackClipDistance(10000);
simpleU.getViewer().getView().setFrontClipDistance(0.1);
TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
Transform3D t3d = new Transform3D();
t3d.setTranslation(new Vector3f(0.0f,0.3f,0.0f));
vpTG.setTransform(t3d);
KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
scene.addChild(keyNavBeh);
scene.compile();
simpleU.addBranchGraph(scene);
}

private BranchGroup createSceneGraph()
{
    // Creo la radice di questo BranchGraph
    BranchGroup objRoot = new BranchGroup();
    // Creo una sfera enorme con una texture (l'immagine di background) e le normali verso l'interno
    // Carico la TEXTURE
    Texture texture = new
TextureLoader(BackgroundClassico.class.getResource("immagineBackground.jpg"), this).getTexture();
    // APPEARANCE, dalla Texture
    Appearance app = new Appearance();
    app.setTexture(texture);
    // SFERA CON APPEARANCE (e quindi Texture) e alto numero di suddivisioni (per non darle un aspetto
troppo "sfaccettato")
    Sphere sphere = new Sphere(200.0f,
Primitive.GENERATE_TEXTURE_COORDS|Primitive.GENERATE_NORMALS_INWARD, 100, app);
    // Aggiungo la sfera appena creata alla radice del Content Branch Graph
    objRoot.addChild(sphere);
    // Assegno un pavimento di riferimento alla scena
    objRoot.addChild(creaRiferimento());
    // Restituisco il Branch Graph così composto
    return objRoot;
}

private Shape3D creaRiferimento()
{
    // Creo un pavimento di riferimento per gli spostamenti. E' un argomento che tratta le geometrie definite
vertice-a-vertice, ancora da esaminare, quindi non verrà commentato ora.
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(1,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(1,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(1.0f,1.0f,1.0f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

BACKGROUND CLASSICO

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come creare un background utilizzando l'oggetto background - definito a partire da una sfera di raggio unitario dotata di texture e normali verso l'interno - e un bound.

Attenzione ! Il BOUND non crea un oggetto visuale ! Definisce solo il volume di spazio OVE E' POSSIBILE VEDERE LO SFONDO.

Lo sfondo non é la proiezione della texture di fronte a noi ad una distanza individuata dal raggio del bound; lo sfondo é... lo sfondo: anche se camminiamo verso di esso, ci apparirà sempre alla stessa distanza, immutabile.

Se usciamo dal bound, non vedremo più uno sfondo, e non lo vedremo neanche voltandoci indietro (vedremo però altri oggetti, se ce ne erano, lasciati dietro di noi): dobbiamo rientrare NEL BOUND (VOLUME) D'AZIONE DELLO SFONDO per vederlo !

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

I comandi per navigare nello Spazio Virtuale con tale Behavior sono:

ROTAZIONI

FrecciaDestra : ruota a destra

FrecciaSinistra : ruota a sinistra

PagSu : ruota verso il basso

PagGiù : ruota verso l'alto

TRASLAZIONI

ALT + FrecciaSu : trasla in avanti

ALT + FrecciaGiù : trasla indietro

ALT + FrecciaDestra : trasla a destra

ALT + FrecciaSinistra : trasla a sinistra

ALT + PagSu : trasla verso il basso

ALT + PagGiù : trasla verso l'alto

*/

// Import necessari per gestire JFrame, JPanel, il menù, il SimpleUniverse e j3d.geometry per gli oggetti di base.

import java.awt.*;

import javax.swing.*;

import javax.vecmath.*;

import javax.media.j3d.*;

import com.sun.j3d.utils.universe.*;

import com.sun.j3d.utils.geometry.*;

import com.sun.j3d.utils.behaviors.keyboard.*;

import com.sun.j3d.utils.image.*;

```
public class BackgroundClassico extends JFrame
{
```

```
    public static void main(String[] args)
```

```
    {
```

```
        BackgroundClassico bc = new BackgroundClassico();
```

```
        bc.setVisible(true);
```

```
    }
```

```
    public BackgroundClassico()
```

```
    {
```

```
        // Impostazioni "classiche" di un JFrame...
```

```
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        this.setLocation(100, 100);
```

```
        this.setSize(400, 400);
```

```
        this.setTitle("Background Classico; Milanese Francesco; www.redbaron85.com");
```

```
GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
Canvas3D canvas3D = new Canvas3D(config);
this.getContentPane().add(canvas3D, BorderLayout.CENTER);
BranchGroup scene = createSceneGraph();
SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
simpleU.getViewer().getView().setBackClipDistance(10000);
simpleU.getViewer().getView().setFrontClipDistance(0.1);
TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
Transform3D t3d = new Transform3D();
t3d.setTranslation(new Vector3f(0.0f, 0.3f, 0.0f));
vpTG.setTransform(t3d);
KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
scene.addChild(keyNavBeh);
scene.compile();
simpleU.addBranchGraph(scene);
}
```

```
private BranchGroup createSceneGraph()
{
    // Creiamo un oggetto Background, associandogli un bound di "operatività", centrato nell'origine e di
raggio 200.
    Background back = new Background();
    back.setApplicationBounds(new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 200.0));
    // Un Background ha inoltre bisogno di una geometria (una sfera),
    // quindi ora creiamo una sfera con texture, poi la associamo al Background...
    // A questo punto dobbiamo creare una sfera, di raggio unitario e con le normali
    // delle facce rivolte verso "dentro", che rappresenterà il nostro mondo.
    // La normale è il vettore perpendicolare ad una faccia. Le normali possono essere impostate
    // come uscenti (ad esempio, una cassa di legno) o verso l'interno (è il nostro caso: sono le pareti
INTERNE della sfera a dover essere dipinte !)
    // Prima di procedere alla creazione della sfera, creiamo il suo aspetto (Appearance),
    // caricando un'immagine di sfondo in un oggetto di tipo TEXTURE
    // Quindi:
    // TEXTURE
    Texture texture = new
TextureLoader(BackgroundClassico.class.getResource("immagineBackground.jpg"), this).getTexture();
    // APPEARANCE, dalla Texture
    Appearance app = new Appearance();
    app.setTexture(texture);
    // SFERA CON APPEARANCE (e quindi Texture)
    Sphere sphere = new Sphere(1.0f,
Primitive.GENERATE_TEXTURE_COORDS|Primitive.GENERATE_NORMALS_INWARD, app);
    // BRANCH GROUP CHE CONTIENE LA SFERA
    BranchGroup bgGeometry = new BranchGroup();
    bgGeometry.addChild(sphere);
    // A questo punto aggiungo il branch group contenente la sfera texturizzata all'oggetto Background
precedentemente creato
    back.setGeometry(bgGeometry);
    // Creo quindi un nuovo Branch Group e gli assegno, come figlio, il Background così composto
    BranchGroup backgroundGroup = new BranchGroup();
    backgroundGroup.addChild(back);
    // Assegno un pavimento di riferimento alla scena
    backgroundGroup.addChild(creaRiferimento());
    // Restituisco il Branch Graph così composto
    return backgroundGroup;
}
```

```
}  
  
private Shape3D creaRiferimento()  
{  
    // Creo un nuovo oggetto LineArray, composto da 4 vertici (dunque verranno visualizzati due segmenti)  
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);  
    // Imposto le coordinate dei 4 vertici  
    float l = -50.0f;  
    for(int c=0; c<44; c+=4)  
    {  
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));  
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));  
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));  
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));  
        l += 10.0f;  
    }  
    Color3f c = new Color3f(1.0f,1.0f,1.0f);  
    for(int i=0; i<44; i++) landGeom.setColor(i,c);  
    return new Shape3D(landGeom);  
}  
}
```

ESEMPIO COLOR CUBE

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come inserire nel Content Branch Graph un oggetto "nativo" di Java3D, \$
un ColorCube, ruotato ma non spostato, fornito della sua Appearance originale, "propria" del ColorCube.

*/

import java.awt.*;

import javax.swing.*;

// import javax.vecmath.*; // Non serve qui

import javax.media.j3d.*;

import com.sun.j3d.utils.universe.*;

import com.sun.j3d.utils.geometry.*;

public class EsempioColorCube extends JFrame

{

 public static void main(String [] args)

 {

 EsempioColorCube ecc = new EsempioColorCube();

 ecc.setVisible(true);

 }

 public EsempioColorCube()

 {

 this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

 this.setSize(300, 300);

 this.setLocation(100, 100);

 this.setTitle("Java 3D, EsempioColorCube. Milaneze Francesco, www.redbaron85.com");

 GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();

 Canvas3D c3D = new Canvas3D(config);

 SimpleUniverse sU = new SimpleUniverse(c3D);

 sU.getViewingPlatform().setNominalViewingTransform();

 BranchGroup scena = creaLoSceneGraph();

 scena.compile();

 sU.addBranchGraph(scena);

 this.getContentPane().add(c3D, BorderLayout.CENTER);

 }

// Il metodo seguente crea una scena di base (un oggetto BranchGroup), aggiungendogli come

//figlio un cubo colorato (oggetto Java3D di base) ruotato, ma non spostato

public BranchGroup creaLoSceneGraph()

{

 // Creiamo la radice dello Scene Graph: è un oggetto BranchGroup

 BranchGroup objRoot = new BranchGroup();

 // Creo un TransformGroup per ruotare un pò la geometria (altrimenti vedremmo un quadrato colorato)

 // Creo una prima trasformazione "rotazione"...

 Transform3D rotazione1 = new Transform3D();

 rotazione1.rotY(Math.PI/4.0);

 // Creo una seconda trasformazione "rotazione"...

 Transform3D rotazione2 = new Transform3D();

 rotazione2.rotX(Math.PI/5.0);

 // Compongo le due rotazioni mettendo il risultato in rotazione2.

 rotazione2.mul(rotazione1);

 // Ecco il nostro TG per le due rotazioni:

```
TransformGroup tgRotazione = new TransformGroup(rotazione2);
// Creo un ColorCube di lato 0.2, centrato nell'origine, con la sua Appearance "propria" (in quanto Color
Cube..)
    tgRotazione.addChild(new ColorCube(0.2));
    // Aggiungo il TG a objRoot....
    objRoot.addChild(tgRotazione);
    // Restituisco, infine, la radice del branch graph con il TG e lo Shape3D
    return objRoot;
}
```

ESEMPIO BOX

// Milaneese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
/*
Questo esempio mostra come inserire nel Content Branch Graph un oggetto "nativo" di Java3D,
un Box, ruotato ma non spostato, fornito di una Appearance di default.
*/
import java.awt.*;
import javax.swing.*;
// import javax.vecmath.*; // Non serve qui
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;

public class EsempioBox extends JFrame
{
    public static void main(String [] args)
    {
        EsempioBox eb = new EsempioBox();
        eb.setVisible(true);
    }

    public EsempioBox()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(300, 300);
        this.setLocation(100, 100);
        this.setTitle("Java 3D, esempio Box. Milaneese Francesco, www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D c3D = new Canvas3D(config);
        SimpleUniverse sU = new SimpleUniverse(c3D);
        sU.getViewerPlatform().setNominalViewingTransform();
        BranchGroup scena = creaLoSceneGraph();
        scena.compile();
        sU.addBranchGraph(scena);
        this.getContentPane().add(c3D, BorderLayout.CENTER);
    }

    public BranchGroup creaLoSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        Transform3D rotazione1 = new Transform3D();
        rotazione1.rotY(Math.PI/4.0);
        Transform3D rotazione2 = new Transform3D();
        rotazione2.rotX(Math.PI/5.0);
        rotazione2.mul(rotazione1);
        TransformGroup tgRotazione = new TransformGroup(rotazione2);
        // Creo un Box con dimensioni x, y e z rispettivamente 0.4, 0.6, 0.2,
        // centrato nell'origine e con Appearance = null, e lo aggiungo a al TG.
        // NOTA: dobbiamo specificare che si tratta di un oggetto del package
        // "com.sun.j3d.utils.geometry" perché vi é un Box "omonimo" in javax.swing !!!
        tgRotazione.addChild(new com.sun.j3d.utils.geometry.Box(0.4f, 0.6f, 0.2f, new Appearance()));
        objRoot.addChild(tgRotazione);
        return objRoot;
    }
}
```

ESEMPIO CONE

// Milinese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
/*
Questo esempio mostra come inserire nel Content Branch Graph un oggetto "nativo" di Java3D,
un Cono, ruotato ma non spostato, fornito di una Appearance di default.
*/

import java.awt.*;
import javax.swing.*;
// import javax.vecmath.*; // Non serve qui
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;

public class EsempioCone extends JFrame
{
    public static void main(String [] args)
    {
        EsempioCone ec = new EsempioCone();
        ec.setVisible(true);
    }

    public EsempioCone()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(300, 300);
        this.setLocation(100, 100);
        this.setTitle("Java 3D, esempio Cone. Milinese Francesco, www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D c3D = new Canvas3D(config);
        SimpleUniverse sU = new SimpleUniverse(c3D);
        sU.getViewingPlatform().setNominalViewingTransform();
        BranchGroup scena = creaLoSceneGraph();
        scena.compile();
        sU.addBranchGraph(scena);
        this.getContentPane().add(c3D, BorderLayout.CENTER);
    }

    public BranchGroup creaLoSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        Transform3D rotazione1 = new Transform3D();
        rotazione1.rotY(Math.PI/4.0);
        Transform3D rotazione2 = new Transform3D();
        rotazione2.rotX(Math.PI/5.0);
        rotazione2.mul(rotazione1);
        TransformGroup tgRotazione = new TransformGroup(rotazione2);
        tgRotazione.addChild(new Cone(0.2f, 1.0f, new Appearance()));
        objRoot.addChild(tgRotazione);
        return objRoot;
    }
}
```

ESEMPIO CYLINDER

// Milinese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
/*
Questo esempio mostra come inserire nel Content Branch Graph un oggetto "nativo" di Java3D,
un Cilindro, ruotato ma non spostato, fornito di una Appearance di default.
*/

import java.awt.*;
import javax.swing.*;
// import javax.vecmath.*; // Non serve qui
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;

public class EsempioCylinder extends JFrame
{
    public static void main(String [] args)
    {
        EsempioCylinder ec = new EsempioCylinder();
        ec.setVisible(true);
    }

    public EsempioCylinder()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(300, 300);
        this.setLocation(100, 100);
        this.setTitle("Java 3D, esempio Cylinder. Milinese Francesco, www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D c3D = new Canvas3D(config);
        SimpleUniverse sU = new SimpleUniverse(c3D);
        sU.getViewingPlatform().setNominalViewingTransform();
        BranchGroup scena = creaLoSceneGraph();
        scena.compile();
        sU.addBranchGraph(scena);
        this.getContentPane().add(c3D, BorderLayout.CENTER);
    }

    public BranchGroup creaLoSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        Transform3D rotazione1 = new Transform3D();
        rotazione1.rotY(Math.PI/4.0);
        Transform3D rotazione2 = new Transform3D();
        rotazione2.rotX(Math.PI/5.0);
        rotazione2.mul(rotazione1);
        TransformGroup tgRotazione = new TransformGroup(rotazione2);
        tgRotazione.addChild(new Cylinder(0.2f, 1.0f, new Appearance()));
        objRoot.addChild(tgRotazione);
        return objRoot;
    }
}
```

ESEMPIO SPHERE 1

// Milinese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come inserire nel Content Branch Graph un oggetto "nativo" di Java3D,
una Sfera, a bassa risoluzione (poche facce), ruotata ma non spostata, fornita di una Appearance di default.

*/

import java.awt.*;

import javax.swing.*;

// import javax.vecmath.*; // Non serve qui

import javax.media.j3d.*;

import com.sun.j3d.utils.universe.*;

import com.sun.j3d.utils.geometry.*;

public class EsempioSphere1 extends JFrame

{

 public static void main(String [] args)

 {

 EsempioSphere1 es = new EsempioSphere1();

 es.setVisible(true);

 }

 public EsempioSphere1()

 {

 this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

 this.setSize(300, 300);

 this.setLocation(100, 100);

 this.setTitle("Java 3D, esempio Sphere1. Milinese Francesco, www.redbaron85.com");

 GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();

 Canvas3D c3D = new Canvas3D(config);

 SimpleUniverse sU = new SimpleUniverse(c3D);

 sU.getViewingPlatform().setNominalViewingTransform();

 BranchGroup scena = creaLoSceneGraph();

 scena.compile();

 sU.addBranchGraph(scena);

 this.getContentPane().add(c3D, BorderLayout.CENTER);

 }

 public BranchGroup creaLoSceneGraph()

 {

 BranchGroup objRoot = new BranchGroup();

 Transform3D rotazione1 = new Transform3D();

 rotazione1.rotY(Math.PI/4.0);

 Transform3D rotazione2 = new Transform3D();

 rotazione2.rotX(Math.PI/5.0);

 rotazione2.mul(rotazione1);

 TransformGroup tgRotazione = new TransformGroup(rotazione2);

 // Creo una Sphere a bassa risoluzione con raggio 0.5f, Appearance di default (colore bianco), centrato

nell'origine.

 tgRotazione.addChild(new Sphere(0.5f, new Appearance()));

 objRoot.addChild(tgRotazione);

 return objRoot;

 }

}

ESEMPIO SPHERE 2

// Milinese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come inserire nel Content Branch Graph un oggetto "nativo" di Java3D,
una Sfera, ad alta risoluzione (molte facce), ruotata ma non spostata, fornita di una Appearance di default.

*/

```
import java.awt.*;
import javax.swing.*;
// import javax.vecmath.*; // Non serve qui
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
```

```
public class EsempioSphere2 extends JFrame
{
    public static void main(String [] args)
    {
        EsempioSphere2 es = new EsempioSphere2();
        es.setVisible(true);
    }

    public EsempioSphere2()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(300, 300);
        this.setLocation(100, 100);
        this.setTitle("Java 3D, esempio Sphere2. Milinese Francesco, www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D c3D = new Canvas3D(config);
        SimpleUniverse sU = new SimpleUniverse(c3D);
        sU.getViewingPlatform().setNominalViewingTransform();
        BranchGroup scena = creaLoSceneGraph();
        scena.compile();
        sU.addBranchGraph(scena);
        this.getContentPane().add(c3D, BorderLayout.CENTER);
    }

    public BranchGroup creaLoSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        Transform3D rotazione1 = new Transform3D();
        rotazione1.rotY(Math.PI/4.0);
        Transform3D rotazione2 = new Transform3D();
        rotazione2.rotX(Math.PI/5.0);
        rotazione2.mul(rotazione1);
        TransformGroup tgRotazione = new TransformGroup(rotazione2);
        // Creo una Sphere ad alta risoluzione con raggio 0.5, 100 divisions e Appearance di default (colore
        bianco), centrata nell'origine.
        tgRotazione.addChild(new Sphere(0.5f, 0, 100, new Appearance()));
        objRoot.addChild(tgRotazione);
        return objRoot;
    }
}
```

ESEMPIO TEXT 2D

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come inserire nel Content Branch Graph un oggetto "nativo" di Java3D,
un Testo 2D, ruotato ma non spostato, fornito di una Appearance di default.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*; // Stavolta questo ci serve per Color3f, all'interno di Text2D
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
```

```
public class EsempioText2D extends JFrame
```

```
{
```

```
    public static void main(String [] args)
```

```
    {
```

```
        EsempioText2D et2d = new EsempioText2D();
        et2d.setVisible(true);
```

```
    }
```

```
    public EsempioText2D()
```

```
    {
```

```
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(300, 300);
        this.setLocation(100, 100);
        this.setTitle("Java 3D, esempio Text2D. Milaneze Francesco, www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D c3D = new Canvas3D(config);
        SimpleUniverse sU = new SimpleUniverse(c3D);
        sU.getViewerPlatform().setNominalViewingTransform();
        BranchGroup scena = creaLoSceneGraph();
        scena.compile();
        sU.addBranchGraph(scena);
        this.getContentPane().add(c3D, BorderLayout.CENTER);
```

```
    }
```

```
    public BranchGroup creaLoSceneGraph()
```

```
    {
```

```
        BranchGroup objRoot = new BranchGroup();
        Transform3D rotazione1 = new Transform3D();
        rotazione1.rotY(Math.PI/4.0);
        Transform3D rotazione2 = new Transform3D();
        rotazione2.rotX(Math.PI/5.0);
        rotazione2.mul(rotazione1);
        TransformGroup tgRotazione = new TransformGroup(rotazione2);
        // Creo un text2d e lo aggiungo al TG della rotazione
        Text2D testo2D = new Text2D("Ciao !", new Color3f(1.0f, 1.0f, 0.0f), "sansserif", 100, Font.PLAIN);
        // Di default, un Text2D é visibile solo frontalmente. Togliere i tag di commento
        //alle seguenti righe per renderizzare il testo anche posteriormente.
        /*
        Appearance textAppear = testo2D.getAppearance();
```

```
        PolygonAttributes polyAttrib = new PolygonAttributes();
        polyAttrib.setCullFace(PolygonAttributes.CULL_NONE);
        polyAttrib.setBackFaceNormalFlip(true);
        textAppear.setPolygonAttributes(polyAttrib);
        */
        tgRotazione.addChild(testo2D);
        objRoot.addChild(tgRotazione);
        return objRoot;
    }
}
```

ESEMPIO TEXT 3D

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come inserire nel Content Branch Graph un oggetto "nativo" di Java3D, un Testo 3D, ruotato ma non spostato, che é una Shape3D a tutti gli effetti dove bisogna specificare Geometry e Appearance (e l'esempio mostra come fare).

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*; // Stavolta questo ci serve per Point3f, all'interno di Text3D
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;

public class EsempioText3D extends JFrame
{
    public static void main(String [] args)
    {
        EsempioText3D et3d = new EsempioText3D();
        et3d.setVisible(true);
    }

    public EsempioText3D()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(300, 300);
        this.setLocation(100, 100);
        this.setTitle("Java 3D, esempio Text3D. Milanese Francesco, www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D c3D = new Canvas3D(config);
        SimpleUniverse sU = new SimpleUniverse(c3D);
        sU.getViewingPlatform().setNominalViewingTransform();
        BranchGroup scena = creaLoSceneGraph();
        scena.compile();
        sU.addBranchGraph(scena);
        this.getContentPane().add(c3D, BorderLayout.CENTER);
    }

    public BranchGroup creaLoSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        Transform3D t3D = new Transform3D();
        t3D.setTranslation(new Vector3f(0.0f, 0.0f, -3.0f));
        TransformGroup tgTraslazione = new TransformGroup(t3D);
        Transform3D rotazione1 = new Transform3D();
        rotazione1.rotY(Math.PI/4.0);
        Transform3D rotazione2 = new Transform3D();
        rotazione2.rotX(-Math.PI/11.0);
        rotazione2.mul(rotazione1);
        TransformGroup tgRotazione = new TransformGroup(rotazione2);

        // INIZIA LA FASE DI CREAZIONE DEL TESTO 3D: PRIMA SI CREA UN APPEARANCE,
        // POI LA GEOMETRIA, INFINE UN OGGETTO SHAPE3D CON SIFFATTE APPEARANCE E
        GEOMETRY:
```

```
// Creo l'Appearance
Appearance textAppear = new Appearance();
ColoringAttributes textColor = new ColoringAttributes();
textColor.setColor(1.0f, 1.0f, 0.0f);
textAppear.setColoringAttributes(textColor);
textAppear.setMaterial(new Material());

// Creo la Geometry
Font3D font3D = new Font3D(new Font("Serif", Font.PLAIN, 1), new FontExtrusion());
Text3D textGeom = new Text3D(font3D, new String("Testo 3D !"));
textGeom.setAlignment(Text3D.ALIGN_CENTER);

// Creo la Shape3D
Shape3D testo3D = new Shape3D();
testo3D.setGeometry(textGeom);
testo3D.setAppearance(textAppear);
// A QUESTO PUNTO, LA SHAPE 3D CHE IDENTIFICA IL NOSTRO TESTO 3D SI TROVA IN
testo3D...

// NOTA: c'è bisogno di una luce direzionale per mostrare Text3D, che è una Shape3D a tutti gli effetti...
BoundingSphere bounds = new BoundingSphere();
DirectionalLight lightD = new DirectionalLight();
lightD.setInfluencingBounds(bounds);
lightD.setDirection(new Vector3f(0.0f, 0.0f, -3.0f));
lightD.setColor(new Color3f(1.0f, 1.0f, 0.0f));
tgTraslazione.addChild(lightD);

// Mettiamo insieme tutta la scena...
// Aggiungo la SHAPE 3D del testo al TG di rotazione (NON posso aggiungere una Geometry ad un
TG !!!)
tgTraslazione.addChild(tgRotazione);
tgRotazione.addChild(testo3D);
objRoot.addChild(tgTraslazione);

// Restituisco, infine, la radice del branch graph con il TG e lo Shape3D
return objRoot;
}
```

ESEMPIO POINT ARRAY

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come inserire un Point Array (in pratica, un insieme di punti, non collegati visivamente ma appartenenti ad un'unica Geometry).

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

Ricordo che i comandi per navigare nello Spazio Virtuale con tale Behavior sono:

ROTAZIONI

FrecciaDestra : ruota a destra

FrecciaSinistra : ruota a sinistra

PagSu : ruota verso il basso

PagGiù : ruota verso l'alto

TRASLAZIONI

ALT + FrecciaSu : trasla in avanti

ALT + FrecciaGiù : trasla indietro

ALT + FrecciaDestra : trasla a destra

ALT + FrecciaSinistra : trasla a sinistra

ALT + PagSu : trasla verso il basso

ALT + PagGiù : trasla verso l'alto

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
```

```
public class EsempioPointArray extends JFrame
{
    public static void main(String[] args)
    {
        EsempioPointArray epa = new EsempioPointArray();
        epa.setVisible(true);
    }

    public EsempioPointArray()
    {
        this.setTitle("EsempioPointArray; Milaneze Francesco; www.redbaron85.com");
        this.setSize(400,400);
        this.setLocation(100,100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = creaLoSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f,0.3f,2.0f));
    }
}
```

```
vpTG.setTransform(t3d);
KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
scene.addChild(keyNavBeh);
scene.compile();
simpleU.addBranchGraph(scene);
}

private BranchGroup creaLoSceneGraph()
{
    BranchGroup objRoot = new BranchGroup();
    // Aggiungo alBranch Graph i punti generati con disegnaPunti()
    objRoot.addChild(this.disegnaPunti());
    return objRoot;
}

private Shape3D disegnaPunti()
{
    // Creo un nuovo oggetto PointArray, composto da tre elementi
    PointArray pa = new PointArray(3, GeometryArray.COORDINATES | GeometryArray.COLOR_3);
    // Imposto le coordinate dei tre punti
    pa.setCoordinate(0, new Point3f(0.0f, 0.0f, 0.0f));
    pa.setCoordinate(1, new Point3f(10.0f, 0.0f, 10.0f));
    pa.setCoordinate(2, new Point3f(10.0f, 0.0f, -10.0f));
    // Creo tre diversi colori e li utilizzo per colorare i tre punti, a ciascun punto un colore proprio
    Color3f xc = new Color3f(0.1f, 0.9f, 0.1f);
    Color3f yc = new Color3f(0.9f, 0.1f, 0.1f);
    Color3f zc = new Color3f(0.1f, 0.1f, 0.9f);
    pa.setColor(0,xc);
    pa.setColor(1,xc);
    pa.setColor(2,xc);
    // Restituisco una nuova Shape3D che avrà, come geometry, il PointArray appena definito
    return new Shape3D(pa);
}
}
```

ESEMPIO LINE ARRAY QUATTRO VERTICI

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come usare un LineArray: definendo un numero PARI di vertici (anche colorati), LineArray creerà dei segmenti (anche colorati) collegando tra loro coppie di vertici.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

ROTAZIONI

FrecciaDestra : ruota a destra

FrecciaSinistra : ruota a sinistra

PagSu : ruota verso il basso

PagGiù : ruota verso l'alto

TRASLAZIONI

ALT + FrecciaSu : trasla in avanti

ALT + FrecciaGiù : trasla indietro

ALT + FrecciaDestra : trasla a destra

ALT + FrecciaSinistra : trasla a sinistra

ALT + PagSu : trasla verso il basso

ALT + PagGiù : trasla verso l'alto

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
```

```
public class EsempioLineArrayQuattroVertici extends JFrame
{
    public static void main(String[] args)
    {
        EsempioLineArrayQuattroVertici ela4v = new EsempioLineArrayQuattroVertici();
        ela4v.setVisible(true);
    }

    public EsempioLineArrayQuattroVertici()
    {
        this.setTitle("EsempioLineArrayQuattroVertici; Milanese Francesco; www.redbaron85.com");
        this.setSize(400,400);
        this.setLocation(100,100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = creaLoSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f,0.3f,2.0f));
        vpTG.setTransform(t3d);
    }
}
```

```
KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
scene.addChild(keyNavBeh);
scene.compile();
simpleU.addBranchGraph(scene);
}

private BranchGroup creaLoSceneGraph()
{
    BranchGroup objRoot = new BranchGroup();
    // Aggiungo al Branch Graph una Shape3D
    Transform3D t3d = new Transform3D();
    t3d.setTranslation(new Vector3f(0.0f, 0.0f, -25.0f));
    TransformGroup tg = new TransformGroup(t3d);
    tg.addChild(this.disegnaLinee());
    objRoot.addChild(tg);
    return objRoot;
}

private Shape3D disegnaLinee()
{
    // Creo un nuovo oggetto LineArray, composto da 4 vertici (dunque verranno visualizzati due segmenti)
    LineArray rif = new LineArray(4, GeometryArray.COORDINATES | GeometryArray.COLOR_3);
    // Imposto le coordinate dei 4 vertici
    rif.setCoordinate(0, new Point3f(0.0f, 0.0f, 0.0f));
    rif.setCoordinate(1, new Point3f(10.0f, 0.0f, 10.0f));
    rif.setCoordinate(2, new Point3f(0.0f, 0.0f, -10.0f));
    rif.setCoordinate(3, new Point3f(-10.0f, 5.0f, -10.0f));
    // Creo tre colori e li utilizzo per colorare i quattro vertici (un colore verrà ripetuto).
    // I segmenti che collegheranno i vertici avranno due colori, a seconda del colore dei vertici che collegano.
    Color3f xc = new Color3f(0.1f, 0.9f, 0.1f);
    Color3f yc = new Color3f(0.9f, 0.1f, 0.1f);
    Color3f zc = new Color3f(0.1f, 0.1f, 0.9f);
    rif.setColor(0,xc);
    rif.setColor(1,yc);
    rif.setColor(2,zc);
    rif.setColor(3,xc);
    // Restituisco un nuovo oggetto Shape3D avente come geometry il LineArray appena definito.
    return new Shape3D(rif);
}
}
```

ESEMPIO LINE ARRAY TRE VERTICI

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come creare array di vertici composti da tre elementi
(visivamente dovremmo avere tre punti, con colori propri, e due segmenti colorati
in base ai colori dei punti che li delimitano).

ATTENZIONE ! QUESTO ESEMPIO NON FUNZIONA !

A runtime verrà generato un errore, difatti specificheremo solo tre punti mentre
in un LineArray il numero dei punti deve essere PARI (per ogni coppia di punti verrà creato un segmento).

*/

// Import necessari per gestire JFrame, JPanel, il menù, il SimpleUniverse e j3d.geometry per gli oggetti di base.

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
```

```
public class EsempioLineArrayTreVertici extends JFrame
```

```
{
    public static void main(String[] args)
    {
        EsempioLineArrayTreVertici ela3v = new EsempioLineArrayTreVertici();
        ela3v.setVisible(true);
    }

    public EsempioLineArrayTreVertici()
    {
        this.setTitle("EsempioLineArrayTreVertici; Milanese Francesco; www.redbaron85.com");
        this.setSize(400,400);
        this.setLocation(100,100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = creaLoSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f,0.3f,2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup creaLoSceneGraph()
}
```

```
{
    BranchGroup objRoot = new BranchGroup();
    // Aggiungo al Branch Graph una Shape3D
    objRoot.addChild(this.disegnaLinee());
    return objRoot;
}

private Shape3D disegnaLinee()
{
    // Creo un LineArray, ma "sbaglio" e gli associo un numero di punti DISPARI
    LineArray rif = new LineArray(3, GeometryArray.COORDINATES | GeometryArray.COLOR_3);
    // Imposto le coordinate dei punti
    rif.setCoordinate(0, new Point3f(0.0f, 0.0f, 0.0f));
    rif.setCoordinate(1, new Point3f(10.0f, 0.0f, 10.0f));
    rif.setCoordinate(2, new Point3f(0.0f, 0.0f, -10.0f));
    // Creo tre colori, che utilizzo per colorare i vertici
    Color3f xc = new Color3f(0.1f, 0.9f, 0.1f);
    Color3f yc = new Color3f(0.9f, 0.1f, 0.1f);
    Color3f zc = new Color3f(0.1f, 0.1f, 0.9f);
    rif.setColor(0,xc);
    rif.setColor(1,yc);
    rif.setColor(2,zc);
    // Restituisco un nuovo oggetto Shape3D avente, come geometry, il LineArray appena definito...
    //A tempo di compilazione non ci sono errori, a run-time sì...
    return new Shape3D(rif);
}
}
```

ESEMPIO SISTEMA DI RIFERIMENTO

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come creare ed inserire nel Content Branch Graph un "sistema di riferimento", composto da tre griglie colorate.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

ROTAZIONI

FrecciaDestra : ruota a destra

FrecciaSinistra : ruota a sinistra

PagSu : ruota verso il basso

PagGiù : ruota verso l'alto

TRASLAZIONI

ALT + FrecciaSu : trasla in avanti

ALT + FrecciaGiù : trasla indietro

ALT + FrecciaDestra : trasla a destra

ALT + FrecciaSinistra : trasla a sinistra

ALT + PagSu : trasla verso il basso

ALT + PagGiù : trasla verso l'alto

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
```

```
public class EsempioSistemaRiferimento extends JFrame
{
    public static void main(String[] args)
    {
        EsempioSistemaRiferimento esr = new EsempioSistemaRiferimento();
        esr.setVisible(true);
    }

    public EsempioSistemaRiferimento()
    {
        this.setTitle("Esempio Sistema Riferimento; Milanese Francesco; www.redbaron85.com");
        this.setSize(400,400);
        this.setLocation(100,100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = creaLoSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f,0.3f,2.0f));
    }
}
```

```
vpTG.setTransform(t3d);
KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
scene.addChild(keyNavBeh);
scene.compile();
simpleU.addBranchGraph(scene);
}

private BranchGroup creaLoSceneGraph()
{
    BranchGroup objRoot = new BranchGroup();
    // Aggiungo al Branch Graph una Shape3D
    objRoot.addChild(this.disegnaGriglia());
    return objRoot;
}

private Shape3D disegnaGriglia()
{
    // Creo una Geometry, di tipo LineArray
    LineArray rif = new LineArray(132, GeometryArray.COORDINATES | GeometryArray.COLOR_3);
    // Imposto un offset per le varie linee della griglia
    float l = -50.0f;
    // Creo le linee della prima griglia (in pratica, creo coppie di coordinate.
    // Ogni coppia di coordinate costituirà una linea della griglia).
    for(int x=0; x<44; x+=4)
    {
        rif.setCoordinate(x+0, new Point3f(-50.0f, 0.0f, l));
        rif.setCoordinate(x+1, new Point3f(50.0f, 0.0f, l));
        rif.setCoordinate(x+2, new Point3f(l, 0.0f, -50.0f));
        rif.setCoordinate(x+3, new Point3f(l, 0.0f, 50.0f));
        l+=10.0f; // Questo é l'offset vero e proprio: a ogni iterazione, il valore di l aumenta di 10 unità
    }
    // Resetto il valore dell'offset
    l = -50.0f;
    // Creo le linee della seconda griglia (in pratica, creo coppie di coordinate.
    // Ogni coppia di coordinate costituirà una linea della griglia).
    for(int y=44; y<88; y+=4)
    {
        rif.setCoordinate(y+0, new Point3f(-50.0f, l, 0.0f));
        rif.setCoordinate(y+1, new Point3f(50.0f, l, 0.0f));
        rif.setCoordinate(y+2, new Point3f(l, -50.0f, 0.0f));
        rif.setCoordinate(y+3, new Point3f(l, 50.0f, 0.0f));
        l+=10.0f; // Questo é l'offset vero e proprio: a ogni iterazione, il valore di l aumenta di 10 unità
    }
    // Resetto il valore dell'offset
    l = -50.0f;
    // Creo le linee della terza griglia (in pratica, creo coppie di coordinate.
    // Ogni coppia di coordinate costituirà una linea della griglia).
    for(int z=88; z<132; z+=4)
    {
        rif.setCoordinate(z+0, new Point3f(0.0f, -50.0f, l));
        rif.setCoordinate(z+1, new Point3f(0.0f, 50.0f, l));
        rif.setCoordinate(z+2, new Point3f(0.0f, l, -50.0f));
        rif.setCoordinate(z+3, new Point3f(0.0f, l, 50.0f));
        l+=10.0f; // Questo é l'offset vero e proprio: a ogni iterazione, il valore di l aumenta di 10 unità
    }
    // Creo tre colori, uno per ciascuno dei "piani" di riferimento
}
```

```
Color3f xc = new Color3f(0.1f, 0.9f, 0.1f);
Color3f yc = new Color3f(0.9f, 0.1f, 0.1f);
Color3f zc = new Color3f(0.1f, 0.1f, 0.9f);
// Imposto i colori ai punti (e quindi, automaticamente, alle linee che li collegano) dei tre piani di
riferimento
for(int x=0; x<44; x++) rif.setColor(x,xc);
for(int y=44; y<88; y++) rif.setColor(y,yc);
for(int z=88; z<132; z++) rif.setColor(z,zc);
// Restituisco un nuovo oggetto Shape3D, la cui geometria é il LineArray appena definito e colorato
return new Shape3D(rif);
    }
}
```

ESEMPIO TRIANGLE ARRAY

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come creare triangoli pieni (colorati) mediante un `TriangleArray`.

In questo esempio si fa uso del `KeyNavigatorBehavior` di Java3D per navigare nello Spazio Virtuale.

ROTAZIONI

FrecciaDestra : ruota a destra

FrecciaSinistra : ruota a sinistra

PagSu : ruota verso il basso

PagGiù : ruota verso l'alto

TRASLAZIONI

ALT + FrecciaSu : trasla in avanti

ALT + FrecciaGiù : trasla indietro

ALT + FrecciaDestra : trasla a destra

ALT + FrecciaSinistra : trasla a sinistra

ALT + PagSu : trasla verso il basso

ALT + PagGiù : trasla verso l'alto

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
```

```
public class EsempioTriangleArray extends JFrame
{
    public static void main(String[] args)
    {
        EsempioTriangleArray eta = new EsempioTriangleArray();
        eta.setVisible(true);
    }

    public EsempioTriangleArray()
    {
        this.setTitle("EsempioTriangleArray; Milanese Francesco; www.redbaron85.com");
        this.setSize(400,400);
        this.setLocation(100,100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = creaLoSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f,0.3f,2.0f));
        vpTG.setTransform(t3d);
    }
}
```

```
KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
scene.addChild(keyNavBeh);
scene.compile();
simpleU.addBranchGraph(scene);
}

private BranchGroup creaLoSceneGraph()
{
    BranchGroup objRoot = new BranchGroup();
    // Aggiungo al Branch Graph una Shape3D
    Transform3D t3d = new Transform3D();
    t3d.setTranslation(new Vector3f(0.0f, 0.0f, -5.0f));
    TransformGroup tg = new TransformGroup(t3d);
    tg.addChild(this.disegnaTriangoli());
    objRoot.addChild(tg);
    return objRoot;
}

private Shape3D disegnaTriangoli()
{
    // Creo un nuovo TriangleArray, specificando il numero dei vertici
    // (deve essere un multiplo di tre) e le modalità di creazione
    TriangleArray ta = new TriangleArray(3, GeometryArray.COORDINATES | GeometryArray.COLOR_3);
    // Imposto le coordinate dei vertici
    ta.setCoordinate(0, new Point3f(0.0f, 0.0f, 2.0f));
    ta.setCoordinate(1, new Point3f(10.0f, 5.0f, 2.0f));
    ta.setCoordinate(2, new Point3f(-10.0f, 2.0f, 2.0f));
    // Creo tre colori e, con essi, coloro i vertici che compongono il TriangleArray
    // (vedere come questo si riflette sulla colorazione dei segmenti)
    Color3f xc = new Color3f(0.1f, 0.9f, 0.1f);
    Color3f yc = new Color3f(0.9f, 0.1f, 0.1f);
    Color3f zc = new Color3f(0.1f, 0.1f, 0.9f);
    ta.setColor(0,xc);
    ta.setColor(1,yc);
    ta.setColor(2,zc);
    // Creo una nuova Shape3D, avente come geometry il TriangleArray appena definito, e la restituisco
    return new Shape3D(ta);
}
}
```

ESEMPIO QUAD ARRAY

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come creare ed inserire nel Content Branch Graph dei QuadArray, che qui sono veri e propri quadrilateri colorati.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

ROTAZIONI

FrecciaDestra : ruota a destra

FrecciaSinistra : ruota a sinistra

PagSu : ruota verso il basso

PagGiù : ruota verso l'alto

TRASLAZIONI

ALT + FrecciaSu : trasla in avanti

ALT + FrecciaGiù : trasla indietro

ALT + FrecciaDestra : trasla a destra

ALT + FrecciaSinistra : trasla a sinistra

ALT + PagSu : trasla verso il basso

ALT + PagGiù : trasla verso l'alto

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
```

```
public class EsempioQuadArray extends JFrame
{
    public static void main(String[] args)
    {
        EsempioQuadArray eqa = new EsempioQuadArray();
        eqa.setVisible(true);
    }

    public EsempioQuadArray()
    {
        this.setTitle("EsempioQuadArray; Milanese Francesco; www.redbaron85.com");
        this.setSize(400,400);
        this.setLocation(100,100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = creaLoSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f,0.3f,2.0f));
```

```
vpTG.setTransform(t3d);
KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
scene.addChild(keyNavBeh);
scene.compile();
simpleU.addBranchGraph(scene);
}

private BranchGroup creaLoSceneGraph()
{
    BranchGroup objRoot = new BranchGroup();
    // Aggiungo al Branch Graph una Shape3D
    objRoot.addChild(this.disegnaQuadrilateri());
    return objRoot;
}

private Shape3D disegnaQuadrilateri()
{
    // Creo un oggetto di tipo QuadArray, impostando il numero di vertici e la modalità di creazione
    QuadArray qa = new QuadArray(12, GeometryArray.COORDINATES | GeometryArray.COLOR_3);
    // Imposto le coordinate dei vertici...
    qa.setCoordinate(0, new Point3f(0.0f, 0.0f, 0.0f));
    qa.setCoordinate(1, new Point3f(10.0f, 0.0f, 0.0f));
    qa.setCoordinate(2, new Point3f(10.0f, 10.0f, 0.0f));
    qa.setCoordinate(3, new Point3f(0.0f, 10.0f, 0.0f));
    // ...altre coordinate dei vertici...
    qa.setCoordinate(4, new Point3f(12.0f, 0.0f, 0.0f));
    qa.setCoordinate(5, new Point3f(20.0f, 0.0f, 0.0f));
    qa.setCoordinate(6, new Point3f(27.0f, -10.0f, 10.0f));
    qa.setCoordinate(7, new Point3f(15.0f, -10.0f, 6.0f));
    // ...altre coordinate dei vertici...
    qa.setCoordinate(8, new Point3f(0.0f, 0.0f, 5.0f));
    qa.setCoordinate(9, new Point3f(7.0f, 0.0f, 5.0f));
    qa.setCoordinate(10, new Point3f(7.0f, -2.0f, 9.0f));
    qa.setCoordinate(11, new Point3f(-2.0f, -2.0f, 9.0f));
    // Creo tre colori
    Color3f xc = new Color3f(0.1f, 0.9f, 0.1f);
    Color3f yc = new Color3f(0.9f, 0.1f, 0.1f);
    Color3f zc = new Color3f(0.1f, 0.1f, 0.9f);
    // Coloro i vertici...
    qa.setColor(0,xc);
    qa.setColor(1,yc);
    qa.setColor(2,zc);
    qa.setColor(3,xc);
    // Coloro i vertici...
    qa.setColor(4,yc);
    qa.setColor(5,zc);
    qa.setColor(6,xc);
    qa.setColor(7,yc);
    // Coloro i vertici...
    qa.setColor(8,zc);
    qa.setColor(9,xc);
    qa.setColor(10,yc);
    qa.setColor(11,zc);

    // Di default c'è il culling sulle facce "posteriori", quindi non tutti i poligoni vengono visualizzati...
```

```
// Per risolvere questo problema, creiamo una Appearance con il culling disattivato e la associamo
// alla Shape3D che avrà come Geometry il QuadArray precedentemente definito
Shape3D miaShape = new Shape3D(qa);
Appearance miaAppearance = new Appearance();
miaAppearance.setPolygonAttributes(new PolygonAttributes(PolygonAttributes.POLYGON_FILL,
PolygonAttributes.CULL_NONE, 0.0f));
miaShape.setAppearance(miaAppearance);

// Restituisco la Shape3D così definita
return miaShape;
    }
}
```

ESEMPIO LINE STRIP ARRAY

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come creare delle linee, anche scollegate ma comunque appartenenti ad un'unica Geometry

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

ROTAZIONI

FrecciaDestra : ruota a destra

FrecciaSinistra : ruota a sinistra

PagSu : ruota verso il basso

PagGiù : ruota verso l'alto

TRASLAZIONI

ALT + FrecciaSu : trasla in avanti

ALT + FrecciaGiù : trasla indietro

ALT + FrecciaDestra : trasla a destra

ALT + FrecciaSinistra : trasla a sinistra

ALT + PagSu : trasla verso il basso

ALT + PagGiù : trasla verso l'alto

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
```

```
public class EsempioLineStripArray extends JFrame
{
    public static void main(String[] args)
    {
        EsempioLineStripArray elsa = new EsempioLineStripArray();
        elsa.setVisible(true);
    }

    public EsempioLineStripArray()
    {
        this.setTitle("EsempioLineStripArray; Milanese Francesco; www.redbaron85.com");
        this.setSize(400,400);
        this.setLocation(100,100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = creaLoSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f,0.3f,2.0f));
        vpTG.setTransform(t3d);
    }
}
```

```
KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
scene.addChild(keyNavBeh);
scene.compile();
simpleU.addBranchGraph(scene);
}

private BranchGroup creaLoSceneGraph()
{
    BranchGroup objRoot = new BranchGroup();
    // Aggiungo una Shape3D al Branch Graph
    Transform3D t3d = new Transform3D();
    t3d.setTranslation(new Vector3f(0.0f, 0.0f, -15.0f));
    TransformGroup tg = new TransformGroup(t3d);
    tg.addChild(this.disegnaLineStrip());
    objRoot.addChild(tg);
    return objRoot;
}

private Shape3D disegnaLineStrip()
{
    // Creo un array di interi. Ogni indice dell'array indica quanti vertici, presi consecutivamente
    // dalle coordinate del LineStripArray che verrà definito in seguito, dovranno costituire una spezzata
    int [] verticiPerGruppo = {4, 7, 2};
    // Creo un oggetto LineStripArray, impostando il numero di vertici, il mode e l'array
    // che indica quanti vertici dovrà avere ogni spezzata
    LineStripArray lsa = new LineStripArray(13, GeometryArray.COORDINATES |
GeometryArray.COLOR_3, verticiPerGruppo);
    // Imposto le coordinate dei vertici...
    lsa.setCoordinate(0, new Point3f(0.0f, 0.0f, 0.0f));
    lsa.setCoordinate(1, new Point3f(10.0f, 0.0f, 0.0f));
    lsa.setCoordinate(2, new Point3f(10.0f, 10.0f, 0.0f));
    lsa.setCoordinate(3, new Point3f(0.0f, 10.0f, 0.0f));
    // ...altre coordinate dei vertici...
    lsa.setCoordinate(4, new Point3f(12.0f, 0.0f, 0.0f));
    lsa.setCoordinate(5, new Point3f(20.0f, 0.0f, 0.0f));
    lsa.setCoordinate(6, new Point3f(27.0f, -10.0f, 10.0f));
    lsa.setCoordinate(7, new Point3f(15.0f, -10.0f, 6.0f));
    lsa.setCoordinate(8, new Point3f(0.0f, 0.0f, 5.0f));
    lsa.setCoordinate(9, new Point3f(7.0f, 0.0f, 5.0f));
    lsa.setCoordinate(10, new Point3f(7.0f, -2.0f, 9.0f));
    // ...altre coordinate dei vertici... stando a verticiPerGruppo, quest'ultima coppia identificherà
    // una spezzata "a sè" (ma sempre della stessa Geometry di tipo LineStripArray !!!)
    lsa.setCoordinate(11, new Point3f(-2.0f, -2.0f, 9.0f));
    lsa.setCoordinate(12, new Point3f(5.0f, 2.0f, 5.0f));
    // Creo tre colori
    Color3f xc = new Color3f(0.1f, 0.9f, 0.1f);
    Color3f yc = new Color3f(0.9f, 0.1f, 0.1f);
    Color3f zc = new Color3f(0.1f, 0.1f, 0.9f);
    // Coloro i vertici...
    lsa.setColor(0,xc);
    lsa.setColor(1,yc);
    lsa.setColor(2,zc);
    lsa.setColor(3,xc);
    // Coloro i vertici...
    lsa.setColor(4,yc);
```

```
        lsa.setColor(5,zc);
        lsa.setColor(6,xc);
        lsa.setColor(7,yc);
        lsa.setColor(8,zc);
        lsa.setColor(9,xc);
        lsa.setColor(10,yc);
        // Coloro i vertici...
        lsa.setColor(11,zc);
        lsa.setColor(12,xc);

        // Di default c'è il culling sulle facce "posteriori", quindi non tutti i poligoni vengono visualizzati...
        // Per risolvere questo problema, creiamo una Appearance con il culling disattivato e la associamo
        // alla Shape3D che avrà come Geometry il LineStripArray precedentemente definito
        Shape3D miaShape = new Shape3D(lsa);
        Appearance miaAppearance = new Appearance();
        miaAppearance.setPolygonAttributes(new PolygonAttributes(PolygonAttributes.POLYGON_FILL,
PolygonAttributes.CULL_NONE, 0.0f));
        miaShape.setAppearance(miaAppearance);

        // Restituisco la Shape3D così definita
        return miaShape;
    }
}
```

ESEMPIO TRIANGLE STRIP ARRAY

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come creare dei TriangleStripArray colorati.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

ROTAZIONI

FrecciaDestra : ruota a destra

FrecciaSinistra : ruota a sinistra

PagSu : ruota verso il basso

PagGiù : ruota verso l'alto

TRASLAZIONI

ALT + FrecciaSu : trasla in avanti

ALT + FrecciaGiù : trasla indietro

ALT + FrecciaDestra : trasla a destra

ALT + FrecciaSinistra : trasla a sinistra

ALT + PagSu : trasla verso il basso

ALT + PagGiù : trasla verso l'alto

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
```

```
public class EsempioTriangleStripArray extends JFrame
{
```

```
    public static void main(String[] args)
```

```
    {
```

```
        EsempioTriangleStripArray etsa = new EsempioTriangleStripArray();
        etsa.setVisible(true);
```

```
    }
```

```
    public EsempioTriangleStripArray()
```

```
    {
```

```
        this.setTitle("EsempioTriangleStripArray; Milanese Francesco; www.redbaron85.com");
        this.setSize(400,400);
        this.setLocation(100,100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = creaLoSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f,0.3f,2.0f));
        vpTG.setTransform(t3d);
```

```
KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
scene.addChild(keyNavBeh);
scene.compile();
simpleU.addBranchGraph(scene);
}

private BranchGroup creaLoSceneGraph()
{
    BranchGroup objRoot = new BranchGroup();
    // Aggiungo al Branch Graph una Shape3D
    objRoot.addChild(this.disegnaTriangleStrip());
    return objRoot;
}

private Shape3D disegnaTriangleStrip()
{
    // Creo un array di interi. Ogni indice dell'array indica quanti vertici, presi consecutivamente
    // dalle coordinate del TriangleStripArray che verrà definito in seguito, dovranno costituire una spezzata
    int [] verticiPerGruppo = {4, 6, 3};
    // Creo un nuovo TriangleStripArray, specificando il numero di vertici, la modalità di creazione
    // e l'array che contiene il numero di vertici per ogni gruppo
    TriangleStripArray tsa = new TriangleStripArray(13, GeometryArray.COORDINATES |
GeometryArray.COLOR_3, verticiPerGruppo);
    // Imposto le coordinate dei vertici...
    tsa.setCoordinate(0, new Point3f(0.0f, 0.0f, 0.0f));
    tsa.setCoordinate(1, new Point3f(10.0f, 0.0f, 0.0f));
    tsa.setCoordinate(2, new Point3f(10.0f, 10.0f, 0.0f));
    tsa.setCoordinate(3, new Point3f(0.0f, 10.0f, 0.0f));
    // ...altre coordinate dei vertici...
    tsa.setCoordinate(4, new Point3f(12.0f, 0.0f, 0.0f));
    tsa.setCoordinate(5, new Point3f(20.0f, 0.0f, 0.0f));
    tsa.setCoordinate(6, new Point3f(27.0f, -10.0f, 10.0f));
    tsa.setCoordinate(7, new Point3f(15.0f, -10.0f, 6.0f));
    tsa.setCoordinate(8, new Point3f(0.0f, 0.0f, 5.0f));
    tsa.setCoordinate(9, new Point3f(7.0f, 0.0f, 5.0f));
    // ...altre coordinate dei vertici...
    tsa.setCoordinate(10, new Point3f(7.0f, -2.0f, 9.0f));
    tsa.setCoordinate(11, new Point3f(-2.0f, -2.0f, 9.0f));
    tsa.setCoordinate(12, new Point3f(5.0f, 2.0f, 5.0f));
    // Creo tre colori
    Color3f xc = new Color3f(0.1f, 0.9f, 0.1f);
    Color3f yc = new Color3f(0.9f, 0.1f, 0.1f);
    Color3f zc = new Color3f(0.1f, 0.1f, 0.9f);
    // Coloro i vertici...
    tsa.setColor(0,xc);
    tsa.setColor(1,yc);
    tsa.setColor(2,zc);
    tsa.setColor(3,xc);
    // Coloro i vertici...
    tsa.setColor(4,yc);
    tsa.setColor(5,zc);
    tsa.setColor(6,xc);
    tsa.setColor(7,yc);
    tsa.setColor(8,zc);
    tsa.setColor(9,xc);
```

```
// Coloro i vertici...
tsa.setColor(10,yc);
tsa.setColor(11,zc);
tsa.setColor(12,xc);

// Di default c'è il culling sulle facce "posteriori", quindi non tutti i poligoni vengono visualizzati...
// Per risolvere questo problema, creiamo una Appearance con il culling disattivato e la associamo
// alla Shape3D che avrà come Geometry il TriangleStripArray precedentemente definito
Shape3D miaShape = new Shape3D(tsa);
Appearance miaAppearance = new Appearance();
miaAppearance.setPolygonAttributes(new PolygonAttributes(PolygonAttributes.POLYGON_FILL,
PolygonAttributes.CULL_NONE, 0.0f));
miaShape.setAppearance(miaAppearance);

// Restituisco la Shape3D così definita
return miaShape;
}
```

ESEMPIO TRIANGLE FAN ARRAY

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come creare dei TriangleFanArray colorati, geometrie che prevedono l'uso di un vertice "pivot" condiviso e di una o più facce triangolari.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

ROTAZIONI

FrecciaDestra : ruota a destra

FrecciaSinistra : ruota a sinistra

PagSu : ruota verso il basso

PagGiù : ruota verso l'alto

TRASLAZIONI

ALT + FrecciaSu : trasla in avanti

ALT + FrecciaGiù : trasla indietro

ALT + FrecciaDestra : trasla a destra

ALT + FrecciaSinistra : trasla a sinistra

ALT + PagSu : trasla verso il basso

ALT + PagGiù : trasla verso l'alto

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
```

```
public class EsempioTriangleFanArray extends JFrame
{
    public static void main(String[] args)
    {
        EsempioTriangleFanArray etfa = new EsempioTriangleFanArray();
        etfa.setVisible(true);
    }

    public EsempioTriangleFanArray()
    {
        this.setTitle("EsempioTriangleFanArray; Milanese Francesco; www.redbaron85.com");
        this.setSize(400,400);
        this.setLocation(100,100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = creaLoSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f,0.3f,2.0f));
```

```
vpTG.setTransform(t3d);
KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
scene.addChild(keyNavBeh);
scene.compile();
simpleU.addBranchGraph(scene);
}

private BranchGroup creaLoSceneGraph()
{
    BranchGroup objRoot = new BranchGroup();
    Transform3D t3d = new Transform3D();
    t3d.setTranslation(new Vector3f(0.0f, 0.0f, -45.0f));
    TransformGroup tg = new TransformGroup(t3d);
    tg.addChild(this.disegnaTriangleFan());
    objRoot.addChild(tg);
    return objRoot;
}

private Shape3D disegnaTriangleFan()
{
    // Creo un array di interi.
    // Ogni indice dell'array indica quanti vertici, presi consecutivamente dalle coordinate
    // del TriangleFanArray che verrà definito in seguito, dovranno costituire una spezzata
    int [] verticiPerGruppo = {5, 3, 5};
    // Creo un nuovo TriangleFanArray, specificando il numero di vertici,
    // la modalità di creazione e l'array che contiene il numero di vertici per ogni gruppo
    TriangleFanArray tfa = new TriangleFanArray(13, GeometryArray.COORDINATES |
GeometryArray.COLOR_3, verticiPerGruppo);
    // Imposto le coordinate dei vertici...
    tfa.setCoordinate(0, new Point3f(0.0f, 10.0f, 0.0f));
    tfa.setCoordinate(1, new Point3f(-5.0f, 7.0f, 2.0f));
    tfa.setCoordinate(2, new Point3f(-2.0f, 5.0f, 4.0f));
    tfa.setCoordinate(3, new Point3f(2.0f, 5.0f, 4.0f));
    tfa.setCoordinate(4, new Point3f(5.0f, 7.0f, 2.0f));
    // ...altre coordinate dei vertici...
    tfa.setCoordinate(5, new Point3f(-15.0f, 2.0f, 7.0f));
    tfa.setCoordinate(6, new Point3f(-17.0f, 4.0f, 10.0f));
    tfa.setCoordinate(7, new Point3f(-9.0f, 6.0f, 9.0f));
    // ...altre coordinate dei vertici...
    tfa.setCoordinate(8, new Point3f(20.0f, 10.0f, 0.0f));
    tfa.setCoordinate(9, new Point3f(15.0f, 7.0f, 7.0f));
    tfa.setCoordinate(10, new Point3f(18.0f, 5.0f, -7.0f));
    tfa.setCoordinate(11, new Point3f(22.0f, 5.0f, 7.0f));
    tfa.setCoordinate(12, new Point3f(25.0f, 7.0f, -7.0f));
    // Creo tre colori
    Color3f xc = new Color3f(0.1f, 0.9f, 0.1f);
    Color3f yc = new Color3f(0.9f, 0.1f, 0.1f);
    Color3f zc = new Color3f(0.1f, 0.1f, 0.9f);
    // Coloro i vertici...
    tfa.setColor(0,xc);
    tfa.setColor(1,yc);
    tfa.setColor(2,zc);
    tfa.setColor(3,xc);
    tfa.setColor(4,yc);
    // Coloro i vertici...
    tfa.setColor(5,zc);
```

```
tfa.setColor(6,xc);
tfa.setColor(7,yc);
// Coloro i vertici...
tfa.setColor(8,zc);
tfa.setColor(9,xc);
tfa.setColor(10,yc);
tfa.setColor(11,zc);
tfa.setColor(12,xc);

// Di default c'è il culling sulle facce "posteriori", quindi non tutti i poligoni vengono visualizzati...
// Per risolvere questo problema, creiamo una Appearance con il culling disattivato e la associamo
// alla Shape3D che avrà come Geometry il TriangleFanArray precedentemente definito
Shape3D miaShape = new Shape3D(tfa);
Appearance miaAppearance = new Appearance();
miaAppearance.setPolygonAttributes(new PolygonAttributes(PolygonAttributes.POLYGON_FILL,
PolygonAttributes.CULL_NONE, 0.0f));
miaShape.setAppearance(miaAppearance);

// Restituisco la Shape3D così definita
return miaShape;
}
```

ESEMPIO INDEXED LINE ARRAY

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come creare ed inserire nel Content Branch Graph un oggetto visuale la cui Geometry viene creata con un IndexedLineArray, che permette il riuso dei vertici da utilizzare.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

ROTAZIONI

FrecciaDestra : ruota a destra

FrecciaSinistra : ruota a sinistra

PagSu : ruota verso il basso

PagGiù : ruota verso l'alto

TRASLAZIONI

ALT + FrecciaSu : trasla in avanti

ALT + FrecciaGiù : trasla indietro

ALT + FrecciaDestra : trasla a destra

ALT + FrecciaSinistra : trasla a sinistra

ALT + PagSu : trasla verso il basso

ALT + PagGiù : trasla verso l'alto

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
```

```
public class EsempioIndexedLineArray extends JFrame
{
    public static void main(String[] args)
    {
        EsempioIndexedLineArray eila = new EsempioIndexedLineArray();
        eila.setVisible(true);
    }

    public EsempioIndexedLineArray()
    {
        this.setTitle("EsempioIndexedLineArray; Milanese Francesco; www.redbaron85.com");
        this.setSize(400,400);
        this.setLocation(100,100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = creaLoSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f,0.3f,2.0f));
```

```
vpTG.setTransform(t3d);
KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
scene.addChild(keyNavBeh);
scene.compile();
simpleU.addBranchGraph(scene);
}

private BranchGroup creaLoSceneGraph()
{
    BranchGroup objRoot = new BranchGroup();
    // Aggiungo al Branch Graph una Shape3D
    objRoot.addChild(this.disegnaLaGeometria());
    return objRoot;
}

private Shape3D disegnaLaGeometria()
{
    // Definisco le coordinate, inserendole in un array di Point3f
    Point3f[] arrayCoordinate =
    {
        new Point3f(-1.0f, 1.0f, -1.0f),
        new Point3f(-1.0f, -1.0f, -1.0f),
        new Point3f(1.0f, -1.0f, -1.0f),
        new Point3f(1.0f, 1.0f, -1.0f),
        new Point3f(-1.0f, 1.0f, 1.0f),
        new Point3f(-1.0f, -1.0f, 1.0f),
        new Point3f(1.0f, -1.0f, 1.0f),
        new Point3f(1.0f, 1.0f, 1.0f)
    };

    // Definisco gli indici, ossia la "sequenza" di vertici da collegare / disegnare
    // Notare che definisco linee, quindi i vertici vanno presi a coppie.
    // A runtime vedremo un cubo in visualizzazione "struttura", perché creando linee creo solo spigoli,
    // non facce quadrate (per questo, vedere EsempioIndexedQuadArray)
    int[] arrayIndici = {0, 3, 3, 2, 2, 1, 3, 7, 7, 6, 6, 2, 7, 4, 4, 5, 5, 6, 4, 0, 5, 1, 1, 0};

    // Ora creo l'IndexedLineArray vero e proprio: specifico quante sono le coordinate
    // e quanti i vertici da disegnare "veramente" (include i duplicati)
    IndexedLineArray mioIndexedLineArray = new IndexedLineArray(arrayCoordinate.length,
IndexedQuadArray.COORDINATES, arrayIndici.length);
    // le coordinate si trovano nell'array arrayCoordinate e si deve partire dall'indice 0 nel leggerle
    mioIndexedLineArray.setCoordinates(0, arrayCoordinate);
    // Specifico qual è l'array-sequenza di vertici
    mioIndexedLineArray.setCoordinateIndices(0, arrayIndici);

    // Restituisco la Shape3D così definita
    return new Shape3D(mioIndexedLineArray);
}
}
```

ESEMPIO INDEXED QUAD ARRAY

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come creare ed inserire nel Content Branch Graph un oggetto visuale la cui Geometry viene creata con un IndexedQuadArray.
IndexedQuadArray indicizza i vertici e permette il riuso delle coordinate di un vertice più volte, per definire - insieme ad altri vertici - facce quadrangolari.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

ROTAZIONI

FrecciaDestra : ruota a destra

FrecciaSinistra : ruota a sinistra

PagSu : ruota verso il basso

PagGiù : ruota verso l'alto

TRASLAZIONI

ALT + FrecciaSu : trasla in avanti

ALT + FrecciaGiù : trasla indietro

ALT + FrecciaDestra : trasla a destra

ALT + FrecciaSinistra : trasla a sinistra

ALT + PagSu : trasla verso il basso

ALT + PagGiù : trasla verso l'alto

*/

```
import java.awt.*;
```

```
import javax.swing.*;
```

```
import javax.vecmath.*;
```

```
import javax.media.j3d.*;
```

```
import com.sun.j3d.utils.universe.*;
```

```
import com.sun.j3d.utils.geometry.*;
```

```
import com.sun.j3d.utils.behaviors.keyboard.*;
```

```
public class EsempioIndexedQuadArray extends JFrame
```

```
{
```

```
    public static void main(String[] args) {
```

```
        EsempioIndexedQuadArray eiga = new EsempioIndexedQuadArray();
```

```
        eiga.setVisible(true);
```

```
    }
```

```
    public EsempioIndexedQuadArray() {
```

```
        this.setTitle("EsempioIndexedQuadArray; Milanese Francesco; www.redbaron85.com");
```

```
        this.setSize(400,400);
```

```
        this.setLocation(100,100);
```

```
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
```

```
        Canvas3D canvas3D = new Canvas3D(config);
```

```
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
```

```
        BranchGroup scene = creaLoSceneGraph();
```

```
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
```

```
        simpleU.getViewer().getView().setBackClipDistance(10000);
```

```
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
```

```
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
```

```
        Transform3D t3d = new Transform3D();
```

```
        t3d.setTranslation(new Vector3f(0.0f,0.3f,2.0f));
```

```
        vpTG.setTransform(t3d);
```

```
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
```

```
keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
scene.addChild(keyNavBeh);
scene.compile();
simpleU.addBranchGraph(scene);
}

private BranchGroup creaLoSceneGraph() {
    BranchGroup objRoot = new BranchGroup();
    // Aggiungo al Branch Graph una Shape3D
    objRoot.addChild(this.disegnaLaGeometria());
    return objRoot;
}

private Shape3D disegnaLaGeometria() {
    // Definisco le coordinate, inserendole in un array di Point3f
    Point3f[] arrayCoordinate =
    {
        new Point3f(-1.0f, 1.0f, -1.0f),
        new Point3f(-1.0f, -1.0f, -1.0f),
        new Point3f(1.0f, -1.0f, -1.0f),
        new Point3f(1.0f, 1.0f, -1.0f),
        new Point3f(-1.0f, 1.0f, 1.0f),
        new Point3f(-1.0f, -1.0f, 1.0f),
        new Point3f(1.0f, -1.0f, 1.0f),
        new Point3f(1.0f, 1.0f, 1.0f)
    };

    // Definisco gli indici, ossia la "sequenza" di vertici da collegare / disegnare
    // Dal momento che si tratta di un QUADarray, creeremo poligoni quadrangolari,
    // dunque gli indici vanno considerati "a 4 a 4" per definire una faccia, e non a 2 a 2,
    // per definire spigoli singoli, come in EsempioIndexedLineArray
    int[] arrayIndici = {0, 1, 2, 3, 4, 5, 6, 7, 0, 4, 7, 3, 1, 5, 6, 2, 0, 4, 5, 1, 3, 7, 6, 2};

    // Ora creo l'IndexedQuadArray vero e proprio: specifico quante sono le coordinate
    // e quanti i vertici da disegnare "veramente" (include i duplicati)
    IndexedQuadArray mioIndexedQuadArray = new IndexedQuadArray(arrayCoordinate.length,
IndexedQuadArray.COORDINATES, arrayIndici.length);
    // Le coordinate si trovano nell'array arrayCoordinate e si deve partire dall'indice 0 nel leggerle
    mioIndexedQuadArray.setCoordinates(0, arrayCoordinate);
    // Specifico qual é l'array-sequenza di vertici
    mioIndexedQuadArray.setCoordinateIndices(0, arrayIndici);

    // Creo la shape 3D a partire dalla geometria. L'appearance serve perché, creando facce quadrate (si tratta
di QuadArray),
    // alcune di esse potrebbero avere le facce non visibili; uso "cull_none" per renderizzarle SEMPRE, da
qualunque lato le si osservi
    Shape3D miaShape = new Shape3D(mioIndexedQuadArray);
    Appearance miaAppearance = new Appearance();
    miaAppearance.setPolygonAttributes(new PolygonAttributes(PolygonAttributes.POLYGON_FILL,
PolygonAttributes.CULL_NONE, 0.0f));
    miaShape.setAppearance(miaAppearance);

    // Restituisco la Shape3D così definita
    return miaShape;
}
}
```

ESEMPIO GEOMETRY INFO

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come creare ed inserire nel Content Branch Graph una forma geometrica personalizzata, specificando le coordinate dei vertici ed il numero di vertici per faccia ed affidando a GeometryInfo il compito di ricavarne i collegamenti.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

ROTAZIONI

FrecciaDestra : ruota a destra

FrecciaSinistra : ruota a sinistra

PagSu : ruota verso il basso

PagGiù : ruota verso l'alto

TRASLAZIONI

ALT + FrecciaSu : trasla in avanti

ALT + FrecciaGiù : trasla indietro

ALT + FrecciaDestra : trasla a destra

ALT + FrecciaSinistra : trasla a sinistra

ALT + PagSu : trasla verso il basso

ALT + PagGiù : trasla verso l'alto

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
```

```
public class EsempioGeometryInfo extends JFrame
{
    public static void main(String[] args)
    {
        EsempioGeometryInfo egi = new EsempioGeometryInfo();
        egi.setVisible(true);
    }

    public EsempioGeometryInfo()
    {
        this.setTitle("EsempioGeometryInfo; Milaneze Francesco; www.redbaron85.com");
        this.setSize(400,400);
        this.setLocation(100,100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = creaLoSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f,0.3f,2.0f));
```

```
vpTG.setTransform(t3d);
KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
scene.addChild(keyNavBeh);
scene.compile();
simpleU.addBranchGraph(scene);
}
```

```
private BranchGroup creaLoSceneGraph()
{
    BranchGroup objRoot = new BranchGroup();
    // Aggiungo al Branch Graph una Shape3D
    objRoot.addChild(this.disegnaLaGeometria());
    return objRoot;
}
```

```
private Shape3D disegnaLaGeometria()
{
    Point3f[] arrayCoordinate =
    {
        // Un lato
        new Point3f(-10f, 0f, 0f),
        new Point3f(-5f, 0f, 0f),
        new Point3f(-3f, 2f, 0f),
        new Point3f(0f, 2f, 0f),
        new Point3f(2f, 0f, 0f),
        new Point3f(7f, 0f, 0f),
        new Point3f(9f, 2f, 0f),
        new Point3f(12f, 2f, 0f),
        new Point3f(14f, 0f, 0f),
        new Point3f(19f, 0f, 0f),
        new Point3f(19f, 2f, 0f),
        new Point3f(12f, 5f, 0f),
        new Point3f(7f, 10f, 0f),
        new Point3f(0f, 10f, 0f),
        new Point3f(-5f, 5f, 0f),
        new Point3f(-10f, 5f, 0f),
        new Point3f(-10f, 0f, 0f),
        // L'altro lato
        new Point3f(-10f, 0f, -10f),
        new Point3f(-5f, 0f, -10f),
        new Point3f(-3f, 2f, -10f),
        new Point3f(0f, 2f, -10f),
        new Point3f(2f, 0f, -10f),
        new Point3f(7f, 0f, -10f),
        new Point3f(9f, 2f, -10f),
        new Point3f(12f, 2f, -10f),
        new Point3f(14f, 0f, -10f),
        new Point3f(19f, 0f, -10f),
        new Point3f(19f, 2f, -10f),
        new Point3f(12f, 5f, -10f),
        new Point3f(7f, 10f, -10f),
        new Point3f(0f, 10f, -10f),
        new Point3f(-5f, 5f, -10f),
        new Point3f(-10f, 5f, -10f),
    }
```

```
        new Point3f(-10f, 0f, -10f),
        // Paraurti anteriore
        new Point3f(19f, 0f, 0f),
        new Point3f(19f, 2f, 0f),
        new Point3f(19f, 2f, -10f),
        new Point3f(19f, 0f, -10f)
    };

    // Definisco il numero di vertici per strip (posso definire più strip,
    // per ciascuna di esse GeometryInfo richiamerà Triangulator, Stripifier e NormalGenerator
    // Il punto è che, anche se si tratta di più facce separate,
    // fanno comunque parte della stessa GEOMETRIA, costituiscono un unico oggetto.
    int[] stripCount = {17, 17, 4}; // Attenzione... queste strip definiscono solo i due lati e il paraurti anteriore...

    // Creo la GeometryInfo, impostandone coordinate e numero di vertici per strip
    GeometryInfo miaGI = new GeometryInfo(GeometryInfo.POLYGON_ARRAY);
    miaGI.setCoordinates(arrayCoordinate);
    miaGI.setStripCounts(stripCount);

    // Genero le normali con il Normal Generator: creo un NormalGenerator
    // e gli passo la GeometryInfo precedentemente definita
    NormalGenerator mioNG = new NormalGenerator();
    mioNG.generateNormals(miaGI);
    // Faccio un piccolo refresh degli indici della mia GeometryInfo, ora che ho generato le sue normali...
    miaGI.recomputeIndices();

    // Stripifier: raggruppo i triangoli adiacenti, semplificando la geometria
    Stripifier mioSF = new Stripifier();
    mioSF.stripify(miaGI);
    // Nuovo refresh della geometria....
    miaGI.recomputeIndices();

    // Creo una Shape e le assegno come geometria quella ricavata dalla GeometryInfo appena definita
    Shape3D shapeDaGeometryInfo = new Shape3D();
    shapeDaGeometryInfo.setGeometry(miaGI.getGeometryArray());
    // Creo anche una apperance per rendere le facce visibili da entrambi i lati
    Appearance miaAppearance = new Appearance();
    miaAppearance.setPolygonAttributes(new PolygonAttributes(PolygonAttributes.POLYGON_LINE,
    PolygonAttributes.CULL_NONE, 0.0f));
    shapeDaGeometryInfo.setAppearance(miaAppearance);
    // Restituisco il tutto
    return shapeDaGeometryInfo;
}
}
```

ESEMPIO RASTER

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import java.awt.Image.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import java.io.File;
import javax.swing.ImageIcon;

public class EsempioRaster extends JFrame
{
    public static void main(String[] args)
    {
        EsempioRaster er = new EsempioRaster();
        er.setVisible(true);
    }

    public EsempioRaster()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Raster; Milanese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 0.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(this.creaRiferimento());
    }
}
```

```
Raster mioRaster = new Raster();
mioRaster.setSize(50, 50);
mioRaster.setPosition(new Point3f(0f, 0f, 0f));
Image in = new ImageIcon("Prova.JPG").getImage();
BufferedImage miaBI = new BufferedImage(50, 50, BufferedImage.TYPE_INT_RGB);
Graphics2D g = miaBI.createGraphics();
g.drawImage(in, 0, 0, 100, 100, null);
g.dispose();
mioRaster.setImage(new ImageComponent2D(ImageComponent2D.FORMAT_RGB, miaBI));

objRoot.addChild(new Shape3D(mioRaster));

return objRoot;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

DOWNTOWN

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;

public class Downtown extends JFrame
{
    public static void main(String [] args)
    {
        Downtown dt = new Downtown();
        dt.setVisible(true);
    }

    public Downtown()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(300, 300);
        this.setLocation(100, 100);
        this.setTitle("Java 3D, esempio di traslazioni. Milanese Francesco, www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D c3D = new Canvas3D(config);
        SimpleUniverse sU = new SimpleUniverse(c3D);
        TransformGroup vpTG = sU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 5.0f, 30.0f));
        vpTG.setTransform(t3d);
        BranchGroup scena = creaLoSceneGraph();
        scena.compile();
        sU.addBranchGraph(scena);
        this.getContentPane().add(c3D, BorderLayout.CENTER);
    }

    // Il metodo seguente crea una scena di base (un oggetto BranchGroup),
    // aggiungendogli diversi oggetti Box bianchi posizionati in posti diversi
    public BranchGroup creaLoSceneGraph()
    {
        // Creiamo la radice dello Scene Graph: è un oggetto BranchGroup
        BranchGroup objRoot = new BranchGroup();
        // Creo un array di posizioni
        float[][] posizioni =
        {
            {13.0f, 0.0f, -10.0f},
            {4.0f, 0.0f, 0.0f},
            {-3.0f, 0.0f, 0.0f},
            {-6.0f, 0.0f, -4.0f},
            {0.0f, 0.0f, 0.0f},
            {0.0f, 0.0f, 0.0f},
            {8.0f, 0.0f, -4.0f},
            {9.0f, 0.0f, -0.0f},
            {-5.0f, 0.0f, 12.0f}
        }
    }
}
```

```
};
// Creo un array di box (o meglio, le loro dimensioni).
// Creo inoltre una Appearance di "appoggio", temporanea, la stessa per tutti i box (bianca)
float[][] dimensioni =
{
    {1.3f, 9.0f, -2.0f},
    {1.0f, 4.0f, -1.0f},
    {1.0f, 3.0f, -1.0f},
    {1.0f, 2.0f, -1.0f},
    {2.0f, 2.0f, -1.0f},
    {1.0f, 3.0f, -2.0f},
    {1.0f, 7.0f, -2.0f},
    {1.0f, 5.0f, -1.0f},
    {2.0f, 1.0f, -1.0f}
};
// Creo una Appearance di default
Appearance appearanceComune = new Appearance();
// Inserisco i box, correttamente posizionati, in objRoot
for (int i=0; i<posizioni.length; i++)
{
    Vector3f vettoreTraslazione = new Vector3f(posizioni[i]);
    Transform3D t3d = new Transform3D();
    t3d.setTranslation(vettoreTraslazione);
    TransformGroup tg = new TransformGroup(t3d);
    tg.addChild(new com.sun.j3d.utils.geometry.Box(dimensioni[i][0], dimensioni[i][1],
dimensioni[i][2], appearanceComune));
    objRoot.addChild(tg);
}
// Restituisco, infine, la radice del branch graph
return objRoot;
}
}
```

ROTAZIONE PIVOT 1

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class RotazionePivot1 extends JFrame
{
    public static void main(String[] args)
    {
        RotazionePivot1 rp1 = new RotazionePivot1();
        rp1.setVisible(true);
    }

    public RotazionePivot1()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Rotazione Pivot 1; Milaneze Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 0.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        // Creo la radice di questo sotto-albero e un pavimento di riferimento
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(this.creaRiferimento());

        // Creo un Transform Group con una trasformazione di traslazione
        // che sposta i figli di fronte all'osservatore (-5 sull'asse Z)
        Vector3f vettore1 = new Vector3f(0.0f, 0.0f, -5.0f);
        Transform3D t3d1 = new Transform3D();
        t3d1.setTranslation(vettore1);
    }
}
```

```
TransformGroup tg1 = new TransformGroup(t3d1);

// Creo un secondo Transform Group che non trasla i figli,
// ma li ruota "sul posto" intorno all'asse Y di PI/4 radianti (45 gradi)
Transform3D t3d2 = new Transform3D();
t3d2.rotY(Math.PI / 4);
TransformGroup tg2 = new TransformGroup(t3d2);

// Aggancio il secondo TG al primo: il primo definisce la posizione,
// il secondo la rotazione intorno a Y
tg1.addChild(tg2);
// Aggiungo un cubo colorato al secondo TG
tg2.addChild(new ColorCube(1.0f));

// Aggiungo il TransformGroup tg1 alla radice di questo sottoalbero
objRoot.addChild(tg1);

// Restituisco la radice di questo sottoalbero, restituendo, così, tutto il sottoalbero.
return objRoot;
}

// Creo un pavimento di riferimento per la scena.
private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(1,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(1,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ROTAZIONE PIVOT 2

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
/*
Questo esempio mostra la rotazione intorno a due pivot differenti: il cono a sx
ruota intorno al suo centro (cioé si ha la rotazione nel TG padre), quello di destra
intorno ad un punto esterno (cioé si ha la rotazione nel TG nonno),
in questo caso individuato nel suo vertice (la "punta").
*/

import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class RotazionePivot2 extends JFrame
{
    public static void main(String[] args)
    {
        RotazionePivot2 rp2 = new RotazionePivot2();
        rp2.setVisible(true);
    }

    public RotazionePivot2()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Rotazione intorno ad un pivot - 2; Milanese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(100);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 0.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        // Creo la radice di questo content branch graph
        BranchGroup objRoot = new BranchGroup();
    }
}
```

```
// Aggiungo un "pavimento" di riferimento alla scena
objRoot.addChild(this.creaRiferimento());

//
=====
3) // Vogliamo due coni, posti di fronte a noi (coordinata z = -15), uno a sinistra (x = -3) e l'altro a destra (x =
// I due coni sono ruotati di PI/2 radianti (cioé 90 gradi) intorno all'asse Z, solo che quello di sinistra
// avrà come pivot della rotazione il centro del cono stesso, quello di destra avrà come pivot il vertice (la
"punta")

// Definisco l'altezza del cono (tale grandezza ci tornerà utile quando dovremo definire la posizione del
pivot)
float altezzaCono = 6f;

// Creo un primo transform group per spostare indietro entrambi i coni, lungo l'asse Z (lo chiamerò,
appunto, tgZ):
Vector3f vettore3f = new Vector3f(0.0f, 0.0f, -15.0f);
Transform3D t3d = new Transform3D();
t3d.setTranslation(vettore3f);
TransformGroup tgZ = new TransformGroup(t3d);

// A partire da tgZ partiranno due diramazioni che porteranno a due transform group, al fine di traslare
// i coni a sinistra e a destra: chiamerò i due TG figli tgXsx e tgXdx
// tgXsx
vettore3f.set(-3.0f, 0.0f, 0.0f);
t3d.setTranslation(vettore3f);
TransformGroup tgXsx = new TransformGroup(t3d);
// tgXsx
vettore3f.set(3.0f, 0.0f, 0.0f);
t3d.setTranslation(vettore3f);
TransformGroup tgXdx = new TransformGroup(t3d);

// Non finisce qui... il
// I transform group precedentemente creati servivano a posizionare nello spazio gli oggetti, ora vogliamo
ruotarli
// Per l'oggetto di sinistra é sufficiente modificare rotZ del transform group che lo controlla, visto che ruota
su se stesso...
// tgXsx --- Recuperiamo quindi la trasformazione tgXsx e impostiamo una rotazione intorno a Z
Transform3D t3dTemp1 = new Transform3D();
tgXsx.getTransform(t3dTemp1);
Transform3D t3dTemp2 = new Transform3D();
t3dTemp2.rotZ(Math.PI/2);
t3dTemp1.mul(t3dTemp2);
tgXsx.setTransform(t3dTemp1);
// Aggiungiamo un cono a sinistra
tgXsx.addChild(new Cone(1f, altezzaCono, new Appearance()));
// Passiamo all'oggetto di destra: qui la cosa si complica un pò...
// Per ruotare il cono intorno ad un pivot che non é il centro dello stesso, dobbiamo definire questo "punto"
altrove
// In pratica, il centro della rotazione sarà tgXdx. Se però agganciamo il cono direttamente a tgXdx,
// il centro di tgXdx coinciderà col centro del cono, e non avremo ottenuto nulla...
// Ecco quindi il da farsi: traslare in alto tgXdx di un valore pari ad altezzaCono/2, aggiungergli
// un nuovo transform group (quello del cono) che trasla in basso il cono di un valore pari a altezzaCono/2,
ruotare tgXdx (il "pivot").
```

```
15.0) // Trasliamo quindi tgXdx in alto. tgXdx sarà il nostro pivot, e si troverà quindi a (3.0, altezzaCono/2, -
// Creiamo un nuovo transform group per il cono di destra, che sposti il cono in basso di altezzaCono/2
// Attenzione... fino a questo punto abbiamo impostato una traslazione e una rotazione intorno all'asse Z
centrata nel punto (3, 2, -15)...
// Il nostro cono deve però trovarsi centrato a distanza altezzaCono/2 dal pivot
Transform3D t3dCono2 = new Transform3D();
t3dCono2.setTranslation(new Vector3f(0.0f, -(altezzaCono/2), 0.0f));
TransformGroup tgCono2 = new TransformGroup(t3dCono2);
// Il cono va aggiunto a questo TransformGroup
tgCono2.addChild(new Cone(1f, altezzaCono, new Appearance()));
// E questo transform group va aggiunto, come figlio, a tgXdx
tgXdx.addChild(tgCono2);
// ... calma, perché non abbiamo ancora finito ! Adesso ruotiamo intorno al pivot, cioè intorno a tgXdx
tgXdx.getTransform(t3dTemp1);
t3dTemp2 = new Transform3D();
t3dTemp2.rotZ(Math.PI/2);
t3dTemp1.mul(t3dTemp2);
tgXdx.setTransform(t3dTemp1);

// Bene, aggancio i vari elementi tra loro e a objRoot
tgZ.addChild(tgXsx);
tgZ.addChild(tgXdx);
objRoot.addChild(tgZ);
//
=====

// Restituisco questo sotto-albero
return objRoot;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(1,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(1,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ESEMPIO SCALING IN CASCATA

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra due cubi colorati posti alla stessa distanza dall'osservatore.

I due cubi sono, a livello di scene graph, cugini.

Il TG antenato comune ha una trasformazione di scaling che si propaga ai suoi figli, dunque sono già scalati (dimezzati).

Il cubo di destra ha, inoltre, una seconda trasformazione di scaling (definita nel TG padre) che lo dimezza ancora una volta, dunque la lunghezza reale del suo lato é 1/4 di quella originale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioScalingInCascata extends JFrame
{
    public static void main(String[] args)
    {
        EsempioScalingInCascata esic2 = new EsempioScalingInCascata();
        esic2.setVisible(true);
    }

    public EsempioScalingInCascata()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Scaling in cascata; Milaneze Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 0.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        // Creo la radice di questo sotto-albero e un pavimento di riferimento
```

```
BranchGroup objRoot = new BranchGroup();
objRoot.addChild(this.creaRiferimento());

// Creo un Transform Group "nonno" che posiziona i suoi figli in avanti rispetto all'osservatore
// (a -15 dall'origine sull'asse Z) e, inoltre, li scala (dimezzandoli).
Vector3f vettore1 = new Vector3f(0.0f, 0.0f, -15.0f);
Transform3D t3d1 = new Transform3D();
t3d1.setTranslation(vettore1);
t3d1.setScale(0.5);
TransformGroup tg1 = new TransformGroup(t3d1);

// Creo il TG che posizionerà il cubo di sinistra. Questo transform group effettua
// solo una traslazione (ma eredita, e propaga ai figli, lo scaling definito nel padre).
Vector3f vettore2 = new Vector3f(-3.0f, 0.0f, 0.0f);
Transform3D t3d2 = new Transform3D();
t3d2.setTranslation(vettore2);
TransformGroup tg2 = new TransformGroup(t3d2);

// Creo il TG per il cubo di destra. Questo Transform Group eredita dal padre una trasformazione
// di scaling e la propaga ai figli; inoltre, applica ai figli una ulteriore trasformazione di scaling propria.
Vector3f vettore3 = new Vector3f(3.0f, 0.0f, 0.0f);
Transform3D t3d3 = new Transform3D();
t3d3.setTranslation(vettore3);
t3d3.setScale(0.5);
TransformGroup tg3 = new TransformGroup(t3d3);

// Collego i nodi: tg1 alla radice di questo sotto-albero e tg2 e tg3 a tg1 (tg2 e tg3 sono fratelli).
tg2.addChild(new ColorCube());
tg3.addChild(new ColorCube());
tg1.addChild(tg2);
tg1.addChild(tg3);
objRoot.addChild(tg1);

// Restituisco la radice di questo sottoalbero, restituendo, così, tutto il sottoalbero.
return objRoot;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(1,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(1,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ESEMPIO ROTO TRAS SCALE

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra due oggetti ColorCube ai quali vengono applicate trasformazioni di traslazione, rotazione e scaling.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
```

```
public class EsempioRotoTrasScale extends JFrame
{
```

```
    public static void main(String [] args)
    {
        EsempioRotoTrasScale erts = new EsempioRotoTrasScale();
        erts.setVisible(true);
    }
```

```
    public EsempioRotoTrasScale()
    {
```

```
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(300, 300);
        this.setLocation(100, 100);
        this.setTitle("Java 3D, esempio completo di rotazioni, traslazioni e ridimensionamenti. Milaneze  
Francesco, www.redbaron85.com");
```

```
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D c3D = new Canvas3D(config);
        SimpleUniverse sU = new SimpleUniverse(c3D);
        TransformGroup vpTG = sU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 5.0f, 15.0f));
        vpTG.setTransform(t3d);
        BranchGroup scena = creaLoSceneGraph();
        scena.compile();
        sU.addBranchGraph(scena);
        this.getContentPane().add(c3D, BorderLayout.CENTER);
    }
```

```
    public BranchGroup creaLoSceneGraph()
    {
```

```
        // Creiamo la radice dello Scene Graph: è un oggetto BranchGroup
        BranchGroup objRoot = new BranchGroup();
```

```
        // Creo due oggetti ColorCube con dimensioni e posizioni di default
        ColorCube cubo1 = new ColorCube();
        ColorCube cubo2 = new ColorCube();
```

```
        // Creo una Transform3D e un TransformGroup per cubo1: devo traslarlo,
        // ruotarlo intorno ad un asse e scalarlo in maniera non uniforme.
        Transform3D trasformazione1 = new Transform3D();
```

```
trasformazione1.rotZ(Math.PI/8.0);
trasformazione1.setTranslation(new Vector3f(-4.0f, 4.0f, -5.0f));
trasformazione1.setScale(new Vector3d(0.4, 2.0, 1.0));
TransformGroup tg1 = new TransformGroup(trasformazione1);
// Aggiungo cubo1 al primo TransformGroup e, quindi, aggiungo il tutto a objRoot
tg1.addChild(cubo1);
objRoot.addChild(tg1);

// Creo una Transform3D e un TransformGroup per cubo2: devo traslarlo,
// ruotarlo intorno a tutti e tre i suoi assi e scalarlo in maniera uniforme
Transform3D rotazione1 = new Transform3D();
rotazione1.rotX(Math.PI/3.0);
Transform3D rotazione2 = new Transform3D();
rotazione2.rotY(Math.PI/5.0);
rotazione2.mul(rotazione1);
Transform3D trasformazione2 = new Transform3D();
trasformazione2.rotZ(Math.PI/4.0);
trasformazione2.mul(rotazione2);
trasformazione2.setTranslation(new Vector3f(3.0f, 3.0f, -18.0f));
trasformazione2.setScale(2.5);
TransformGroup tg2 = new TransformGroup(trasformazione2);
// Aggiungo cubo2 al secondo TransformGroup e, quindi, aggiungo il tutto a objRoot
tg2.addChild(cubo2);
objRoot.addChild(tg2);

// Restituisco, infine, la radice del branch graph con il TG e lo Shape3D
return objRoot;
}
}
```

ESEMPIO BEHAVIOR 1

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
/*
Questo esempio mostra come interagire con la scena 3D: la pressione dei tasti freccia
e pagu/paggiù provoca la traslazione del ColorCube presente nell'Universo Virtuale
(il tasto "r" riporta l'oggetto nella posizione iniziale).
A questo file é infatti associato un Behavior creato ad hoc che reagisce alla pressione
dei suddetti tasti (tale Behavior é definito nel file mioBehavior).
*/

import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioBehavior1 extends JFrame
{
    public static void main(String[] args)
    {
        EsempioBehavior1 eb1 = new EsempioBehavior1();
        eb1.setVisible(true);
    }

    public EsempioBehavior1()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Primo Esempio Behavior; Milaneze Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 1.0f, 5.0f));
        vpTG.setTransform(t3d);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(this.creaGruppoBehavior());
        objRoot.addChild(this.creaRiferimento());
        return objRoot;
    }
}
```

```
private TransformGroup creaGruppoBehavior()
{
    // Creo un Transform Group, impostandone le capability
    TransformGroup tg1 = new TransformGroup();
    tg1.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    // Aggiungo un cubo colorato al TransformGroup
    tg1.addChild(new ColorCube(0.4f));
    // Creo un mio behavior in grado di rispondere agli eventi da tastiera, ne imposto il bound d'azione e lo
aggiungo a tg1 (dopo aver impostato, come target, proprio tg1)
    mioBehavior mb1 = new mioBehavior();
    mb1.setSchedulingBounds(new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 100.0));
    mb1.impostaTGtarget(tg1);
    tg1.addChild(mb1);
    // Restituisco il tutto
    return tg1;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

MIO BEHAVIOR 1

// Milaneese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
/*
Una classe "ad hoc" che estende Behavior e che trasla gli oggetti controllati
dal TransformGroup "target" in base ai tasti premuti dall'utente.
In particolare, con i tasti freccia, pagu e paggiù un oggetto viene traslato nell'Universo Virtuale
(in realtà, modifico i valori di trasformazione del TG che controlla l'oggetto),
mentre il tasto "r" viene riportato nella posizione di partenza.
*/

import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import java.awt.event.*; // Necessario per catturare gli eventi da tastiera
import java.util.*; // Necessario per Enumeration

// Creo un Behavior che, in base ai tasti premuti dall'utente, muove gli oggetti
// figli del TransformGroup che ha come target.
public class mioBehavior extends Behavior
{
    // Variabile interna che specifica qual é il Transform Group "target" di questo Behavior
    // (per richiamarlo quando sarà il momento di trasformarlo !)
    private TransformGroup TGtarget;

    // Le variabili che seguono si riferiscono a QUESTO specifico behavior, che ha
    // il compito di muovere il cubo di "EsempioBehavior1" in base ai tasti premuti dall'utente
    private Transform3D t3dTraslazione = new Transform3D();
    private Vector3f posizioneIniziale = new Vector3f(0.0f, 0.0f, 0.0f);

    // Costruttore, in questo caso vuoto...
    public mioBehavior()
    {
    }

    // Imposto il TG "target" (quello sul quale il Behavior farà valere le sue azioni)
    public void impostaTGtarget(TransformGroup tgIn)
    {
        this.TGtarget = tgIn;
    }

    // Specifico qual é il criterio di attivazione del Behavior;
    // in questo caso, si tratta degli eventi generati da tastiera (AWT EVENT)
    public void initialize()
    {
        this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
    }

    // Definisco l'azione da intraprendere e RESETTO IL TRIGGER DEL BEHAVIOR
```

```
// (operazione che va sempre fatta in coda al metodo !!!)
public void processStimulus(Enumeration criteria)
{
    // Azioni da intraprendere: se l'utente preme uno dei tasti freccia,
    // il cubo si sposta di 2 unità nella direzione indicata (per spostarsi in alto o in basso, pagu e paggiù)
    // Per tornare alla posizione di partenza (l'origine), premere R ("reset").

    // Recupero gli stimoli generati, inserendoli in un array
    // ATTENZIONE ! Gli stimoli possono essere di qualunque natura,
    // ma qui noi vogliamo considerare solo quelli di tipo AWTEvent !!!
    AWTEvent [] eventi = null;
    while(criteria.hasMoreElements())
    {
        Object temp = criteria.nextElement();
        if (temp instanceof WakeupOnAWTEvent) eventi =
            ((WakeupOnAWTEvent)temp).getAWTEvent();
    }

    // Creiamo alcune variabili d'appoggio (da riusare) e scorriamo quindi l'array
    // appena generato in cerca di eventi da tastiera, da trattare come vogliamo noi...
    int codiceEvento;
    KeyEvent eventoDaTastiera;
    Transform3D t3dAux = new Transform3D();
    Transform3D t3dAux2 = new Transform3D();
    Vector3f v3fAux = new Vector3f(0.0f, 0.0f, 0.0f);

    if (eventi != null) // Se l'array degli eventi contiene elementi...
    {
        for(int i=0; i < eventi.length; i++) // ...scorri l'array degli eventi...
        {
            if(eventi[i] instanceof KeyEvent) // ... e se l'elemento corrente é un evento generato da
            tastiera...
            {
                eventoDaTastiera = (KeyEvent)eventi[i];
                codiceEvento = eventoDaTastiera.getKeyCode();
                // I codici degli eventi da tastiera sono quelli visti a Interazione e Multimedia: i
                codici Java AWT...
                if(codiceEvento == KeyEvent.VK_LEFT)
                {
                    // Ottengo la trasformazione associata al TG target PRIMA della
                    modifica e la deposito in t3dAux
                    TGtarget.getTransform(t3dAux);
                    // Preparo una trasformazione di rotazione intorno all'asse voluto e
                    della quantità voluta
                    t3dAux2.setTranslation(new Vector3f(-2.0f, 0.0f, 0.0f));
                    // Moltiplico la trasformazione appena creata a quella del TG target
                    t3dAux.mul(t3dAux2);
                    // Imposto, come nuova trasformazione del TG target, la
                    trasformazione appena calcolata
                    TGtarget.setTransform(t3dAux);
                }
                else if(codiceEvento == KeyEvent.VK_RIGHT)
                {
                    TGtarget.getTransform(t3dAux);
                    t3dAux2.setTranslation(new Vector3f(2.0f, 0.0f, 0.0f));
                    t3dAux.mul(t3dAux2);
                    TGtarget.setTransform(t3dAux);
                }
            }
        }
    }
}
```

```
    }
    else if(codiceEvento == KeyEvent.VK_UP)
    {
        TGtarget.getTransform(t3dAux);
        t3dAux2.setTranslation(new Vector3f(0.0f, 0.0f, -2.0f));
        t3dAux.mul(t3dAux2);
        TGtarget.setTransform(t3dAux);
    }
    else if(codiceEvento == KeyEvent.VK_DOWN)
    {
        TGtarget.getTransform(t3dAux);
        t3dAux2.setTranslation(new Vector3f(0.0f, 0.0f, 2.0f));
        t3dAux.mul(t3dAux2);
        TGtarget.setTransform(t3dAux);
    }
    else if(codiceEvento == KeyEvent.VK_PAGE_UP)
    {
        TGtarget.getTransform(t3dAux);
        t3dAux2.setTranslation(new Vector3f(0.0f, 2.0f, 0.0f));
        t3dAux.mul(t3dAux2);
        TGtarget.setTransform(t3dAux);
    }
    else if(codiceEvento == KeyEvent.VK_PAGE_DOWN)
    {
        TGtarget.getTransform(t3dAux);
        t3dAux2.setTranslation(new Vector3f(0.0f, -2.0f, 0.0f));
        t3dAux.mul(t3dAux2);
        TGtarget.setTransform(t3dAux);
    }
    else if(codiceEvento == KeyEvent.VK_R)
    {
        TGtarget.getTransform(t3dAux);
        t3dTraslazione.setTranslation(new Vector3f(0.0f, 0.0f, 0.0f));
        t3dAux.mul(t3dTraslazione);
        t3dTraslazione.get(v3fAux);
        t3dAux.set(v3fAux);
        TGtarget.setTransform(t3dAux);
    }
    }
}

// RESETTO IL TRIGGER DEL BEHAVIOR
this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
}
```

ESEMPIO BEHAVIOR 2

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
/*
Questo esempio mostra come interagire con la scena 3D:
alla pressione del tasto "e" viene ruotato un ColorCube che, dopo quattro secondi, torna nella posizione originale.
A questo file é infatti associato un Behavior creato ad hoc che reagisce alla pressione del tasto "e"
e che dopo 4 secondi genera una nuova azione (tale Behavior é definito nel file mioBehavior2).
*/
```

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioBehavior2 extends JFrame
{
    public static void main(String[] args)
    {
        EsempioBehavior2 eb2 = new EsempioBehavior2();
        eb2.setVisible(true);
    }

    public EsempioBehavior2()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Behavior 2; Milanese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 1.0f, 5.0f));
        vpTG.setTransform(t3d);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(this.creaGruppoBehavior());
        objRoot.addChild(this.creaRiferimento());
        return objRoot;
    }

    private TransformGroup creaGruppoBehavior()
    {
```

```
{
    // Creo un Transform Group, impostandone le capability
    TransformGroup tg1 = new TransformGroup();
    tg1.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    // Aggiungo un cubo colorato al TransformGroup
    tg1.addChild(new ColorCube(0.4f));
    // Creo un mio behavior in grado di rispondere agli eventi da tastiera e all'elapsed time,
    // ne imposto il bound d'azione e lo aggiungo a tg1 (dopo aver impostato, come target, proprio tg1)
    mioBehavior2 mb1 = new mioBehavior2(tg1);
    mb1.setSchedulingBounds(new BoundingSphere());
    tg1.addChild(mb1);
    // Restituisco il tutto
    return tg1;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

MIO BEHAVIOR 2

// Milinese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
/*
Una classe "ad hoc" che estende Behavior e che fa due cose: se l'utente preme il tasto "e",
ruota un oggetto e - dopo 4 secondi - lancia automaticamente un nuovo evento che
riporta l'oggetto all'orientamento iniziale.
*/

import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import java.awt.event.*; // Necessario per catturare gli eventi da tastiera
import java.util.*; // Necessario per Enumeration

// Creo un Behavior che "apre una porta" quando l'utente preme "e" e, dopo pochi secondi, la richiude
public class mioBehavior2 extends Behavior
{
    // Variabile interna che specifica qual è il Transform Group "target" di questo Behavior
    // (per richiamarlo quando sarà il momento di trasformarlo !)
    private TransformGroup TGtarget;

    // La variabile che segue si riferisce a QUESTO specifico behavior, che ha il compito di ruotare
    // un oggetto in base a due stimoli diversi (il primo da tastiera, il secondo temporale)
    private double angolo;

    // Costruttore: stavolta imposto qui il target e altre variabili
    public mioBehavior2(TransformGroup in)
    {
        this.TGtarget = in;
        this.angolo = 0.0;
    }

    // Specifico qual è il criterio di attivazione del Behavior; in questo caso,
    // si tratta degli eventi generati da tastiera (AWT EVENT)
    public void initialize()
    {
        if(this.angolo==0) this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
        else this.wakeupOn(new WakeupOnElapsedTime(4000));
    }

    // Definisco l'azione da intraprendere e RESETTO IL TRIGGER DEL BEHAVIOR
    // (operazione che va sempre fatta in coda al metodo !!!)
    public void processStimulus(Enumeration criteria)
    {
        // Azioni da intraprendere: se l'utente preme il tasto "e", apri la porta (rotazione intorno all'asse Y).
        // Dopo pochi secondi, chiudere automaticamente la porta.

        // Recupero gli stimoli generati, inserendoli in un array
    }
}
```

```
// ATTENZIONE ! Gli stimoli possono essere di qualunque natura, ma qui noi
// vogliamo considerare solo quelli di tipo AWTEvent e "OnElapsedTime".
Object [] eventi = null;
Object temp = null;
// Successivamente avrò bisogno di un array di eventi, dunque creo un array ausiliario temporaneo
// di eventi WakeupOnElapsedTime, che conterrà un solo elemento
WakeupOnElapsedTime [] aux = new WakeupOnElapsedTime[1];

while(criteria.hasMoreElements())
{
    temp = criteria.nextElement();
    if (temp instanceof WakeupOnAWTEvent) eventi =
((WakeupOnAWTEvent)temp).getAWTEvent(); // Restituisce un array di eventi di tipo AWT
    else if(temp instanceof WakeupOnElapsedTime)
    {
        // eventi vuole un array di eventi, dunque creo un array ausiliario temporaneo
        // di eventi WakeupOnElapsedTime, che conterrà un solo elemento !
        aux[0] = (WakeupOnElapsedTime)temp;
        // Passo dunque l'evento all'array di eventi
        eventi = aux;
    }
}

// Creiamo alcune variabili d'appoggio (da riusare) e scorriamo quindi l'array appena generato
// in cerca di eventi da tastiera, da trattare come vogliamo noi...
int codiceEvento;
KeyEvent eventoDaTastiera;
Transform3D t3dAux = new Transform3D();

// Scorriamo l'array degli eventi...
if (eventi != null) // Se l'array degli eventi contiene elementi...
{
    for(int i=0; i < eventi.length; i++) // ...scorri l'array degli eventi...
    {
        if(eventi[i] instanceof KeyEvent) // ... e se l'elemento corrente é un evento generato da tastiera...
        {
            // Recupero i codici degli eventi da tastiera, per capire quale tasto é stato premuto
            // I codici degli eventi da tastiera sono quelli visti a Interazione e Multimedia: i codici
            eventoDaTastiera = (KeyEvent)eventi[i];
            codiceEvento = eventoDaTastiera.getKeyCode();
            if(codiceEvento == KeyEvent.VK_E)
            {
                if(angolo==0.0) // Applica la trasformazione solo se la porta é chiusa
                {
                    // Applico la trasformazione: cambio il valore di "angolo" in 90° e lo
                    this.angolo = Math.PI / 2;
                    t3dAux.rotY(angolo);
                    TGtarget.setTransform(t3dAux);
                    // Invoco un WakeupOnElapsedTime, che "scatterà" tra 4 secondi
                    this.wakeupOn(new WakeupOnElapsedTime(4000));
                }
            }
        }
        else if(eventi[i] instanceof WakeupOnElapsedTime)
        {

```

Java AWT...

passo al TG target

```
        // Applico la trasformazione: cambio il valore di "angolo" in 0° e lo passo al TG target
        this.angolo = 0.0;
        t3dAux.rotY(angolo);
        TGtarget.setTransform(t3dAux);
    }
}

// RESETTO IL TRIGGER DEL BEHAVIOR
// ATTENZIONE !!! Gli sto dicendo che, se l'angolo é a 0.0, deve accettare eventi da tastiera,
// altrimenti deve accettare quelli da ElapsedTime
if(this.angolo != 0.0) this.wakeupOn(new WakeupOnElapsedTime(4000));
else this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
}
}
```

ESEMPIO BEHAVIOR NAVIGAZIONE

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

In questo esempio viene mostrato l'utilizzo di KeyNavigatorBehavior,
Behavior di navigazione nell'Universo Virtuale mediante tastiera "nativo" di Java3D.

I comandi per navigare nello Spazio Virtuale con tale Behavior sono:

ROTAZIONI

FrecciaDestra : ruota a destra

FrecciaSinistra : ruota a sinistra

PagSu : ruota verso il basso

PagGiù : ruota verso l'alto

TRASLAZIONI

ALT + FrecciaSu : trasla in avanti

ALT + FrecciaGiù : trasla indietro

ALT + FrecciaDestra : trasla a destra

ALT + FrecciaSinistra : trasla a sinistra

ALT + PagSu : trasla verso il basso

ALT + PagGiù : trasla verso l'alto

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
```

```
public class EsempioBehaviorNavigazione extends JFrame
{
    public static void main(String[] args)
    {
        EsempioBehaviorNavigazione ebn = new EsempioBehaviorNavigazione();
        ebn.setVisible(true);
    }

    public EsempioBehaviorNavigazione()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Behavior Navigazione; Milanese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
    }
}
```

```
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(new ColorCube(1.0f));
        objRoot.addChild(this.creaRiferimento());
        return objRoot;
    }

    private Shape3D creaRiferimento()
    {
        LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
        float l = -50.0f;
        for(int c=0; c<44; c+=4)
        {
            landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
            landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
            landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
            landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
            l += 10.0f;
        }
        Color3f c = new Color3f(0.1f,0.8f,0.1f);
        for(int i=0; i<44; i++) landGeom.setColor(i,c);
        return new Shape3D(landGeom);
    }
}
```

ESEMPIO NAV BEHAVIOR 2

// Milinese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

In questo esempio, il Behavior per la navigazione mediante tastiera é un Behavior "personalizzato",
definito in un file esterno (MioBehaviorTastiera) ed agganciato al ViewPlatformTransform.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
```

```
public class EsempioNavBehavior2 extends JFrame
{
```

```
    public static void main(String[] args)
```

```
    {
```

```
        EsempioNavBehavior2 enb2 = new EsempioNavBehavior2();
        enb2.setVisible(true);
```

```
    }
```

```
    public EsempioNavBehavior2()
```

```
    {
```

```
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Behavior Navigazione Personalizzato; Milinese Francesco;
www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 1.0f, 5.0f));
        vpTG.setTransform(t3d);
        MioBehaviorTastiera behaviorTastiera = new MioBehaviorTastiera(vpTG);
        behaviorTastiera.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(behaviorTastiera);
        scene.compile();
        simpleU.addBranchGraph(scene);
```

```
    }
```

```
    private BranchGroup createSceneGraph()
```

```
    {
```

```
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(new ColorCube(1.0f));
        objRoot.addChild(this.creaRiferimento());
        return objRoot;
```

```
}  
  
private Shape3D creaRiferimento()  
{  
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);  
    float l = -50.0f;  
    for(int c=0; c<44; c+=4)  
    {  
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));  
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));  
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));  
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));  
        l += 10.0f;  
    }  
    Color3f c = new Color3f(0.1f,0.8f,0.1f);  
    for(int i=0; i<44; i++) landGeom.setColor(i,c);  
    return new Shape3D(landGeom);  
}  
}
```

MIO BEHAVIOR TASTIERA

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Una classe "ad hoc" che estende Behavior e che modifica la posizione e l'orientamento della ViewPlatformTransform (questo se passiamo come TransformGroup "target" il ViewPlatformTransformGroup, ovviamente...)

In particolare:

- tasto A: trasla a sinistra (spostamento laterale);
- tasto S: trasla indietro;
- tasto D: trasla a destra (spostamento laterale);
- tasto W: trasla in avanti;
- freccia avanti: rotazione verso l'alto;
- freccia in basso: rotazione verso il basso;
- freccia a sinistra: rotazione a sinistra;
- freccia a destra: rotazione a destra.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import java.util.*;
import java.awt.event.*;
```

```
public class MioBehaviorTastiera extends Behavior
```

```
{
```

```
    // Variabile interna che specifica qual é il Transform Group "target"
    //di questo Behavior (per richiamarlo quando sarà il momento di trasformarlo !)
    private TransformGroup TGtarget;
```

```
    public MioBehaviorTastiera(TransformGroup tgIn)
```

```
    {
```

```
        // Costruttore: prende in input il TG "target" e lo associa alla variabile interna
        TGtarget = tgIn;
```

```
    }
```

```
    // Metodo di inizializzazione: deve essere sovrascritto da tutte le classi che estendono Behavior
```

```
    public void initialize()
```

```
    {
```

```
        // Questo Behavior verrà attivato da eventi di tipo KeyEvent (da tastiera, alla pressione di un tasto)
        this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
```

```
    }
```

```
    // Metodo processStimulus: deve essere sovrascritto da tutte le classi che estendono Behavior.
```

```
    // Viene richiamato quando si verifica un'azione o "stimolo" all'interno del bound d'azione nel quale agisce il
```

```
Behavior
```

```
    public void processStimulus(Enumeration criteria)
```

```
    {
```

```
        // Creo un array di eventi AWTEvent: conterrà la "coda" di eventi da considerare
        AWTEvent [] eventi = null;
        Object temp = null;
```

```
// Scorro criteria per vedere se ci sono stimoli in ingresso
while(criteria.hasMoreElements())
{
    temp = criteria.nextElement();
    if (temp instanceof WakeupOnAWTEvent) eventi =
((WakeupOnAWTEvent)temp).getAWTEvent(); // Restituisce un array di eventi di tipo AWT
}

// Creiamo alcune variabili d'appoggio (da riusare) e scorriamo quindi l'array
// appena generato in cerca di eventi da tastiera, da trattare come vogliamo noi...
int codiceEvento;
KeyEvent eventoDaTastiera;
Transform3D t3dAux = new Transform3D();
Transform3D t3dAux2 = new Transform3D();
Transform3D t3dTraslazione = new Transform3D();
Vector3f v3fAux = new Vector3f(0.0f, 0.0f, 0.0f);

// Scorriamo l'array degli eventi...
if (eventi != null) // Se l'array degli eventi contiene elementi...
{
    for(int i=0; i < eventi.length; i++) // ...scorri l'array degli eventi...
    {
        if(eventi[i] instanceof KeyEvent) // ... e se l'elemento corrente é un evento generato da
tastiera...
        {
            // Recupero i codici degli eventi da tastiera, per capire quale tasto é stato
premutato
            // I codici degli eventi da tastiera sono quelli visti a Interazione e Multimedia: i
codici Java AWT...

            eventoDaTastiera = (KeyEvent)eventi[i];
            codiceEvento = eventoDaTastiera.getKeyCode();

            // Segue la gestione degli eventi: a seconda di quale tasto é stato premuto,
            // modifico il TG associato a questo Behavior
            if(codiceEvento == KeyEvent.VK_A) // Tasto A: spostamento laterale a sinistra
            {
                // Ottengo la trasformazione associata al TG target PRIMA della
modifica e la deposito in t3dAux

                TGtarget.getTransform(t3dAux);
                // Preparo una trasformazione di rotazione intorno all'asse voluto e
della quantità voluta

                t3dAux2.setTranslation(new Vector3f(-1.0f, 0.0f, 0.0f));
                // Moltiplico la trasformazione appena creata a quella del TG target
                t3dAux.mul(t3dAux2);
                // Imposto, come nuova trasformazione del TG target, la
trasformazione appena calcolata

                TGtarget.setTransform(t3dAux);
            }
            else if(codiceEvento == KeyEvent.VK_S) // Tasto S: indietro
            {
                TGtarget.getTransform(t3dAux);
                t3dAux2.setTranslation(new Vector3f(0.0f, 0.0f, 1.0f));
                t3dAux.mul(t3dAux2);
                TGtarget.setTransform(t3dAux);
            }
            else if(codiceEvento == KeyEvent.VK_W) // Tasto W: avanti
```

```

        {
            TGtarget.getTransform(t3dAux);
            t3dAux2.setTranslation(new Vector3f(0.0f, 0.0f, -1.0f));
            t3dAux.mul(t3dAux2);
            TGtarget.setTransform(t3dAux);
        }
        else if(codiceEvento == KeyEvent.VK_D) // Tasto D: spostamento laterale
verso destra
        {
            TGtarget.getTransform(t3dAux);
            t3dAux2.setTranslation(new Vector3f(1.0f, 0.0f, 0.0f));
            t3dAux.mul(t3dAux2);
            TGtarget.setTransform(t3dAux);
        }
        else if(codiceEvento == KeyEvent.VK_UP) // Freccia in alto: ROTAZIONE
verso l'alto (rotazione intorno all'asse X)
        {
            TGtarget.getTransform(t3dAux);
            t3dAux2.rotX(Math.PI/18);
            t3dAux.mul(t3dAux2);
            TGtarget.setTransform(t3dAux);
        }
        else if(codiceEvento == KeyEvent.VK_DOWN) // Freccia in alto:
ROTAZIONE verso il basso (rotazione intorno all'asse X)
        {
            TGtarget.getTransform(t3dAux);
            t3dAux2.rotX(-Math.PI/18);
            t3dAux.mul(t3dAux2);
            TGtarget.setTransform(t3dAux);
        }
        else if(codiceEvento == KeyEvent.VK_LEFT) // Freccia in alto: ROTAZIONE
verso sinistra (rotazione intorno all'asse Z)
        {
            TGtarget.getTransform(t3dAux);
            t3dAux2.rotY(Math.PI/18);
            t3dAux.mul(t3dAux2);
            TGtarget.setTransform(t3dAux);
        }
        else if(codiceEvento == KeyEvent.VK_RIGHT) // Freccia in alto:
ROTAZIONE verso destra (rotazione intorno all'asse Z)
        {
            TGtarget.getTransform(t3dAux);
            t3dAux2.rotY(-Math.PI/18);
            t3dAux.mul(t3dAux2);
            TGtarget.setTransform(t3dAux);
        }
    }
}
// RESETTO il trigger di questo behavior
this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
}
}

```

ESEMPIO MOUSE BEHAVIOR

// Milanesi Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come interagire con la scena 3D mediante i tasti del mouse:

- tenendo premuto il tasto sinistro, muovendo il mouse si provoca la rotazione dell'oggetto;
- tenendo premuto il tasto centrale, muovendo il mouse si provoca la traslazione dell'oggetto lungo l'asse Z;
- tenendo premuto il tasto destro, muovendo il mouse si provoca la rotazione dell'oggetto sul piano XY.

Qui si fa uso dei Behavior per il mouse "nativi" di Java3D: MouseTranslate, MouseRotate, MouseZoom.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import com.sun.j3d.utils.behaviors.mouse.*;

public class EsempioMouseBehavior extends JFrame
{
    public static void main(String[] args)
    {
        EsempioMouseBehavior emb = new EsempioMouseBehavior();
        emb.setVisible(true);
    }

    public EsempioMouseBehavior()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Mouse Behavior; Milanesi Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 2.0f, 4.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
```

```
{
    BranchGroup objRoot = new BranchGroup();
    // Creo un Transform Group e gli aggancio, come figlio, un cubo colorato
    TransformGroup tg = new TransformGroup();
    tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    tg.addChild(new ColorCube(1.0f));
    // Qui creo un MouseBehavior per la traslazione, uno per la rotazione e uno per lo "zoom",
    // e li associo al TG che controlla il cubo creato precedentemente
    MouseTranslate mioMT = new MouseTranslate(tg);
    mioMT.setSchedulingBounds(new BoundingSphere());
    MouseRotate mioMR = new MouseRotate(tg);
    mioMR.setSchedulingBounds(new BoundingSphere());
    MouseZoom mioMZ = new MouseZoom(tg);
    mioMZ.setSchedulingBounds(new BoundingSphere());
    // Aggiungo il TG col cubo alla scena
    objRoot.addChild(tg);
    // Devo aggiungere anche i Behavior alla scena !
    objRoot.addChild(mioMT);
    objRoot.addChild(mioMR);
    objRoot.addChild(mioMZ);
    // Aggiungo un riferimento alla scena, scollegato dal TG e dai Behavior
    objRoot.addChild(this.creaRiferimento());
    // Restituisco il tutto
    return objRoot;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ESEMPIO PICKING

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come utilizzare i tre Behavior di Picking col mouse "nativi" di Java3D per selezionare gli oggetti "pickable" (o meglio, i TransformGroup che li controllano) presenti nella scena e:

- ruotarli, tenendo premuto il tasto sinistro del mouse e muovendo il mouse;
- traslarli sull'asse Z, tenendo premuto il tasto centrale del mouse e muovendo il mouse;
- traslarli sul piano XY, tenendo premuto il tasto destro del mouse e muovendo il mouse.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import com.sun.j3d.utils.behaviors.mouse.*;
import com.sun.j3d.utils.picking.behaviors.*;

public class EsempioPicking extends JFrame
{
    public static void main(String[] args)
    {
        EsempioPicking ep = new EsempioPicking();
        ep.setVisible(true);
    }

    public EsempioPicking()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Picking Callback; Milaneze Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 2.0f, 4.0f));
        vpTG.setTransform(t3d);

        // ATTENZIONE !!! Sto creando QUI i Behavior per il picking !!!
        PickTranslateBehavior mioPTB = new PickTranslateBehavior(scene, canvas3D, new BoundingSphere());
        PickRotateBehavior mioPRB = new PickRotateBehavior(scene, canvas3D, new BoundingSphere());
        PickZoomBehavior mioPZB = new PickZoomBehavior(scene, canvas3D, new BoundingSphere());
        scene.addChild(mioPTB);
        scene.addChild(mioPRB);
    }
}
```

```
scene.addChild(mioPZB);  
// Adesso creo il "consueto" behavior per la tastiera  
KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);  
keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));  
scene.addChild(keyNavBeh);  
scene.compile();  
simpleU.addBranchGraph(scene);  
}
```

```
private BranchGroup createSceneGraph()  
{
```

```
    // Creo la radice dello Scene Graph  
    BranchGroup objRoot = new BranchGroup();  
    // Creo un Transform Group e gli aggancio, come figlio, un cubo colorato  
    Transform3D t3d1 = new Transform3D();  
    t3d1.setTranslation(new Vector3f(-5.0f, 0.0f, -5.0f));  
    TransformGroup tg1 = new TransformGroup(t3d1);  
    tg1.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);  
    tg1.setCapability(TransformGroup.ENABLE_PICK_REPORTING); // "Nuova" capability da impostare
```

per il TG

```
    tg1.addChild(new ColorCube(1.0f));  
    // Creo un secondo TransformGroup e gli aggancio, come figlio, un secondo cubo colorato  
    Transform3D t3d2 = new Transform3D();  
    t3d2.setTranslation(new Vector3f(5.0f, 0.0f, -5.0f));  
    TransformGroup tg2 = new TransformGroup(t3d2);  
    tg2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);  
    tg2.setCapability(TransformGroup.ENABLE_PICK_REPORTING); // "Nuova" capability da impostare
```

per il TG

```
    tg2.addChild(new ColorCube(1.0f));  
    // Assemblo la scena e restituisco il tutto  
    objRoot.addChild(tg1);  
    objRoot.addChild(tg2);  
    return objRoot;
```

```
}
```

```
}
```

ESEMPIO PICKING CALLBACK

// Milinese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come utilizzare i tre Behavior di Picking col mouse "nativi" di Java3D per selezionare gli oggetti "pickable" (o meglio, i TransformGroup che li controllano) presenti nella scena e:

- ruotarli, tenendo premuto il tasto sinistro del mouse e muovendo il mouse;
- traslarli sull'asse Z, tenendo premuto il tasto centrale del mouse e muovendo il mouse;
- traslarli sul piano XY, tenendo premuto il tasto destro del mouse e muovendo il mouse.

Inoltre, un oggetto che implementa l'interfaccia PickingCallback (definito nel file miaClasseDiPickingCallback) associato ai Behavior di Picking restituisce informazioni sulle azioni compiute dall'utente.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import com.sun.j3d.utils.behaviors.mouse.*;
import com.sun.j3d.utils.picking.behaviors.*;

public class EsempioPickingCallback extends JFrame
{
    public static void main(String[] args)
    {
        EsempioPickingCallback epc = new EsempioPickingCallback();
        epc.setVisible(true);
    }

    public EsempioPickingCallback()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio di utilizzo di una classe di Picking Callback; Milinese Francesco;
www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 2.0f, 4.0f));
        vpTG.setTransform(t3d);
    }
}
```

```
// ATTENZIONE !!! Sto creando QUI i Behavior per il picking !!!
PickTranslateBehavior mioPTB = new PickTranslateBehavior(scene, canvas3D, new BoundingSphere());
PickRotateBehavior mioPRB = new PickRotateBehavior(scene, canvas3D, new BoundingSphere());
PickZoomBehavior mioPZB = new PickZoomBehavior(scene, canvas3D, new BoundingSphere());
scene.addChild(mioPTB);
scene.addChild(mioPRB);
scene.addChild(mioPZB);

// A questo punto, imposto come classe di PickingCallback per i tre PickingBehavior una mia classe di
PickingCallback, definita altrove...
miaClasseDiPickingCallback mcpc = new miaClasseDiPickingCallback();
mioPTB.setupCallback(mcpc);
mioPRB.setupCallback(mcpc);
mioPZB.setupCallback(mcpc);

// Adesso creo il "consueto" behavior per la tastiera
KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
scene.addChild(keyNavBeh);
scene.compile();
simpleU.addBranchGraph(scene);
}

private BranchGroup createSceneGraph()
{
    BranchGroup objRoot = new BranchGroup();
    Transform3D t3d1 = new Transform3D();
    t3d1.setTranslation(new Vector3f(-5.0f, 0.0f, -5.0f));
    TransformGroup tg1 = new TransformGroup(t3d1);
    tg1.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    tg1.setCapability(TransformGroup.ENABLE_PICK_REPORTING); // "Nuova" capability da impostare
per il TG

    tg1.addChild(new ColorCube(1.0f));
    Transform3D t3d2 = new Transform3D();
    t3d2.setTranslation(new Vector3f(5.0f, 0.0f, -5.0f));
    TransformGroup tg2 = new TransformGroup(t3d2);
    tg2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    tg2.setCapability(TransformGroup.ENABLE_PICK_REPORTING); // "Nuova" capability da impostare
per il TG

    tg2.addChild(new ColorCube(1.0f));
    objRoot.addChild(tg1);
    objRoot.addChild(tg2);
    return objRoot;
}
}
```

MIA CLASSE DI PICKING CALLBACK

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
/*
Una classe "ad hoc" che implementa l'interfaccia PickingCallback.
*/

import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import com.sun.j3d.utils.behaviors.mouse.*;
import com.sun.j3d.utils.picking.behaviors.*;
import com.sun.j3d.utils.picking.*;

public class miaClasseDiPickingCallback implements PickingCallback
{
    public miaClasseDiPickingCallback()
    {
        // Costruttore personalizzabile
    }

    public void transformChanged(int type, TransformGroup tg)
    {
        // In questo caso, quando il TG associato all'oggetto creato con questa classe
        // viene modificato, stampo a video una descrizione della trasformazione avvenuta
        System.out.println("Modificato il tg: " + tg + ", con una trasformazione di tipo: " + type);
    }
}
```

ESEMPIO PICKING CALLBACK 2

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come utilizzare i tre Behavior di Picking col mouse "nativi" di Java3D per selezionare gli oggetti "pickable" (o meglio, i TransformGroup che li controllano) presenti nella scena e:

- ruotarli, tenendo premuto il tasto sinistro del mouse e muovendo il mouse;
- traslarli sull'asse Z, tenendo premuto il tasto centrale del mouse e muovendo il mouse;
- traslarli sul piano XY, tenendo premuto il tasto destro del mouse e muovendo il mouse.

Inoltre, un oggetto che implementa l'interfaccia PickingCallback (definito nel file miaClasseDiPickingCallback2) associato ai Behavior di Picking restituisce informazioni sulle azioni compiute dall'utente.

In questo esempio, inoltre, i TransformGroup padri dei cubi visibili nella scena non sono "semplici" TG ma oggetti "mioTransformGroup", cioè TG provvisti di una stringa (il loro "nome").
Quando verranno selezionati con il picking, la classe di pickingcallback associata ai behavior del mouse leggerà i loro nomi e li stamperà a video.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import com.sun.j3d.utils.behaviors.mouse.*;
import com.sun.j3d.utils.picking.behaviors.*;
```

```
public class EsempioPickingCallback2 extends JFrame
{
    public static void main(String[] args)
    {
        EsempioPickingCallback2 epc2 = new EsempioPickingCallback2();
        epc2.setVisible(true);
    }

    public EsempioPickingCallback2()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio di utilizzo di una classe di Picking Callback - 2; Milanese Francesco;
www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
```

```
t3d.setTranslation(new Vector3f(0.0f, 2.0f, 4.0f));
vpTG.setTransform(t3d);

// ATTENZIONE !!! Sto creando QUI i Behavior per il picking !!!
PickTranslateBehavior mioPTB = new PickTranslateBehavior(scene, canvas3D, new BoundingSphere());
PickRotateBehavior mioPRB = new PickRotateBehavior(scene, canvas3D, new BoundingSphere());
PickZoomBehavior mioPZB = new PickZoomBehavior(scene, canvas3D, new BoundingSphere());
scene.addChild(mioPTB);
scene.addChild(mioPRB);
scene.addChild(mioPZB);
// A questo punto, imposto come classe di PickingCallback per i tre PickingBehavior una mia classe di
PickingCallback, definita altrove...
miaClasseDiPickingCallback2 mcpc2 = new miaClasseDiPickingCallback2();
mioPTB.setupCallback(mcpc2);
mioPRB.setupCallback(mcpc2);
mioPZB.setupCallback(mcpc2);

KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
scene.addChild(keyNavBeh);
scene.compile();
simpleU.addBranchGraph(scene);
}

private BranchGroup createSceneGraph()
{
    // Creo la radice dello Scene Graph
    BranchGroup objRoot = new BranchGroup();

    // Creo un mioTransform Group, con un suo nome, e gli aggancio, come figlio, un cubo colorato
    Transform3D t3d1 = new Transform3D();
    t3d1.setTranslation(new Vector3f(-5.0f, 0.0f, -5.0f));
    mioTransformGroup mtg1 = new mioTransformGroup("TG del cubo di sinistra");
    mtg1.setTransform(t3d1);
    mtg1.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    mtg1.setCapability(TransformGroup.ENABLE_PICK_REPORTING); // "Nuova" capability da impostare
per il TG
    mtg1.addChild(new ColorCube(1.0f));

    // Creo un secondo mioTransformGroup, con un suo nome, e gli aggancio, come figlio, un secondo cubo
colorato
    Transform3D t3d2 = new Transform3D();
    t3d2.setTranslation(new Vector3f(5.0f, 0.0f, -5.0f));
    mioTransformGroup mtg2 = new mioTransformGroup("TG del cubo di destra");
    mtg2.setTransform(t3d2);
    mtg2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    mtg2.setCapability(TransformGroup.ENABLE_PICK_REPORTING); // "Nuova" capability da impostare
per il TG
    mtg2.addChild(new ColorCube(1.0f));

    // Assemblo la scena e restituisco il tutto
    objRoot.addChild(mtg1);
    objRoot.addChild(mtg2);
    return objRoot;
}
}
```

MIA CLASSE DI PICKING CALLBACK 2

// Milinese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
/*
Una classe "ad hoc" che implementa l'interfaccia PickingCallback.
*/

// Import di base (non tutti necessari, qui, a dire il vero...)
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import com.sun.j3d.utils.behaviors.mouse.*;
import com.sun.j3d.utils.picking.behaviors.*;
import com.sun.j3d.utils.picking.*;

public class miaClasseDiPickingCallback2 implements PickingCallback
{
    public miaClasseDiPickingCallback2()
    {
        // Il costruttore in questo caso é vuoto...
    }

    public void transformChanged(int type, TransformGroup tg)
    {
        // Quando viene modificato un TransformGroup, se questo é un TransformGroup di tipo
        "mioTransformGroup"
        // (e quindi ha una stringa "nome"), viene stampato a video il suo... "nome".
        if(tg instanceof mioTransformGroup) System.out.println(((mioTransformGroup)tg).getNome());
    }
}
```

MIO TRANSFORM GROUP

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questa classe estende TransformGroup, creando un TransformGroup che ha, in più, una Stringa, chiamata "nome".

*/

// Import di base

import javax.media.j3d.*;

```
public class mioTransformGroup extends TransformGroup
{
    private String nome;

    public mioTransformGroup(String s)
    {
        nome = s;
    }

    public void setNome(String s)
    {
        nome = s;
    }

    public String getNome()
    {
        return nome;
    }
}
```

ESEMPIO LUCE AMBIENTALE

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra l'utilizzo di una luce AMBIENTALE
(bassa intensità, non ha un'origine ma é presente ovunque in eguale intensità), gialla.
Solo una delle due sfere viene illuminata in quanto l'altra é fuori dal bound d'azione della luce.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioLuceAmbientale extends JFrame
{
    public static void main(String[] args)
    {
        EsempioLuceAmbientale ela = new EsempioLuceAmbientale();
        ela.setVisible(true);
    }

    public EsempioLuceAmbientale()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Luce Ambientale; Milanese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        // Creo una luce ambientale con un piccolo bound d'azione
```

```
objRoot.addChild(this.creaUnaLuceAmbientale());
// La prima sfera si trova entro il raggio d'azione della luce e verrà visualizzata
objRoot.addChild(this.creaUnaSfera(10.0f, 0.0f, -10.0f));
// La seconda sfera si trova al di fuori del raggio d'azione e verrà colorata di nero
// (é possibile individuarla per via del "buco" che crea nel sistema di riferimento)
objRoot.addChild(this.creaUnaSfera(0.0f, 0.0f, -30.0f));
// Creo un pavimento di riferimento e lo aggiungo alla scena
objRoot.addChild(this.creaRiferimento());
// Restituisco il tutto
return objRoot;
}

private TransformGroup creaUnaSfera(float x, float y, float z)
{
    // Creo un TransformGroup impostando una trasformazione di traslazione definita dai parametri x, y e z.
    Transform3D t3d1 = new Transform3D();
    t3d1.setTranslation(new Vector3f(x, y, z));
    TransformGroup tg1 = new TransformGroup(t3d1);
    // Aggiungo al TransformGroup un oggetto Sphere di raggio 2.0f.
    tg1.addChild(new Sphere(2.0f));
    // Restituisco il TransformGroup
    return tg1;
}

private AmbientLight creaUnaLuceAmbientale()
{
    // Creo una luce di tipo Ambient Light, rendendola attiva (primo parametro) e con colore bianco
    // (secondo parametro), fornendole un bound d'azione del raggio di 20 unità
    AmbientLight aL = new AmbientLight(true, new Color3f(1.0f, 1.0f, 0.0f));
    aL.setInfluencingBounds(new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 20.0));
    // Restituisco la luce così definita.
    return aL;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ESEMPIO LUCE DIREZIONALE

// Milinese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra l'utilizzo di una luce DIREZIONALE
(posizionata a "infinito" e orientata verso (-5.0f, 0.0f, 0.0f)), color CIANO.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
```

```
public class EsempioLuceDirezionale extends JFrame
{
```

```
    public static void main(String[] args)
```

```
    {
        EsempioLuceDirezionale eld = new EsempioLuceDirezionale();
        eld.setVisible(true);
    }
```

```
    public EsempioLuceDirezionale()
```

```
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Luce Direzionale; Milinese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }
```

```
    private BranchGroup createSceneGraph()
```

```
    {
        // Creo la radice del Branch Graph
        BranchGroup objRoot = new BranchGroup();
        // Creo una luce direzionale di colore CIANO
```

```
objRoot.addChild(this.creaUnaLuceDirezionale());
// La prima sfera si trova nell'origine, è colorata di verde
objRoot.addChild(this.creaUnaSfera(0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f));
// La seconda sfera si trova al di fuori del raggio d'azione della luce direzionale.
// E' colorata di rosso ma vedremo solo il colore che deriva dallo Specular Color di default.
objRoot.addChild(this.creaUnaSfera(0.0f, 0.0f, -10.0f, 1.0f, 0.0f, 0.0f));
// Creo un pavimento di riferimento e lo aggiungo alla scena
objRoot.addChild(this.creaRiferimento());
// Restituisco il tutto
return objRoot;
}

private TransformGroup creaUnaSfera(float x, float y, float z, float r, float g, float b)
{
    // Creo un TransformGroup con una trasformazione di traslazione per posizionare gli oggetti dove
    // specificato dai valori x, y e z.
    Transform3D t3d1 = new Transform3D();
    t3d1.setTranslation(new Vector3f(x, y, z));
    TransformGroup tg1 = new TransformGroup(t3d1);
    // Creo una sfera di raggio 2.0f
    Sphere sfera = new Sphere(2.0f);
    // Creo un'Appearance e un Material per l'Appearance.
    // Il Material avrà un colore diffuse color le cui componenti rgb vengono prese dai float r, g e b in ingresso.
    Appearance app = new Appearance();
    Material mat = new Material();
    mat.setDiffuseColor(new Color3f(r, g, b));
    app.setMaterial(mat);
    // Imposto l'Appearance così definita come appearance della sfera
    sfera.setAppearance(app);
    // Aggiungo la sfera al Transform Group
    tg1.addChild(sfera);
    // Restituisco il Transform Group
    return tg1;
}

private DirectionalLight creaUnaLuceDirezionale()
{
    // Creo una Directional Light, rendendola attiva (primo parametro),
    // impostandone colore (secondo parametro), direzione (terzo parametro) e bound d'influenza (metodo
    // esterno)
    DirectionalLight dL = new DirectionalLight(true, new Color3f(0.0f, 1.0f, 1.0f), new Vector3f(-5.0f, 0.0f,
    0.0f));
    dL.setInfluencingBounds(new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 200.0));
    // Restituisco la Directional Light così definita
    return dL;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
    }
}
```

```
        l += 10.0f;  
    }  
    Color3f c = new Color3f(0.1f,0.8f,0.1f);  
    for(int i=0; i<44; i++) landGeom.setColor(i,c);  
    return new Shape3D(landGeom);  
}  
}
```

ESEMPIO POINT LIGHT

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra l'utilizzo di una luce di tipo Point Light, posizionata nell'origine, color MAGENTA.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioLucePointLight extends JFrame
{
    public static void main(String[] args)
    {
        EsempioLucePointLight elpl = new EsempioLucePointLight();
        elpl.setVisible(true);
    }

    public EsempioLucePointLight()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Luce PointLight; Milaneze Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        // Creo la radice del Branch Graph
        BranchGroup objRoot = new BranchGroup();
        // Creo una sorgente di luce PointLight posizionata nell'origine,
        // color MAGENTA, funzione di attenuazione di default
    }
}
```

```
objRoot.addChild(this.creaUnaLucePointLight());
// Sfera 1, colore diffuso ROSSO, colore speculare BIANCO
objRoot.addChild(this.creaUnaSfera(-20.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f));
// Sfera 2, colore diffuso ROSSO, colore speculare BIANCO
objRoot.addChild(this.creaUnaSfera(-10.0f, 0.0f, -10.0f, 1.0f, 0.0f, 0.0f));
// Sfera 3, colore diffuso ROSSO, colore speculare BIANCO
objRoot.addChild(this.creaUnaSfera(0.0f, 0.0f, -20.0f, 1.0f, 0.0f, 0.0f));
// Sfera 4, colore diffuso ROSSO, colore speculare BIANCO
objRoot.addChild(this.creaUnaSfera(10.0f, 0.0f, -10.0f, 1.0f, 0.0f, 0.0f));
// Sfera 5, colore diffuso ROSSO, colore speculare BIANCO
objRoot.addChild(this.creaUnaSfera(20.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f));
// Creo un pavimento di riferimento e lo aggiungo alla scena
objRoot.addChild(this.creaRiferimento());
// Restituisco il tutto
return objRoot;
}

private TransformGroup creaUnaSfera(float x, float y, float z, float r, float g, float b)
{
    // Creo un TransformGroup con una trasformazione di traslazione
    // per posizionare gli oggetti dove specificato dai valori x, y e z.
    Transform3D t3d1 = new Transform3D();
    t3d1.setTranslation(new Vector3f(x, y, z));
    TransformGroup tg1 = new TransformGroup(t3d1);
    // Creo una sfera di raggio 2.0f
    Sphere sfera = new Sphere(2.0f);
    // Creo un'Appearance e un Material per l'Appearance.
    // Il Material avrà un colore diffuse color le cui componenti rgb vengono prese dai float r, g e b in ingresso.
    Appearance app = new Appearance();
    Material mat = new Material();
    mat.setDiffuseColor(new Color3f(r, g, b));
    app.setMaterial(mat);
    // Imposto l'Appearance così definita come appearance della sfera
    sfera.setAppearance(app);
    // Aggiungo la sfera al Transform Group
    tg1.addChild(sfera);
    // Restituisco il Transform Group
    return tg1;
}

private PointLight creaUnaLucePointLight()
{
    // Creo una luce Point Light, impostandone colore, posizione e bound d'influenza
    PointLight pL = new PointLight();
    pL.setColor(new Color3f(1.0f, 0.0f, 1.0f));
    pL.setPosition(new Point3f(0.0f, 0.0f, 0.0f));
    pL.setInfluencingBounds(new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 200.0));
    // Restituisco la luce così definita
    return pL;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {

```

```
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ESEMPIO SPOT LIGHT

// Milinese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra l'utilizzo di una luce di tipo spot light, posizionata nell'origine, orientata verso (0.0, 0.0, -10.0), color MAGENTA, con spread angle 45 gradi.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioSpotLight extends JFrame
{
    public static void main(String[] args)
    {
        EsempioSpotLight esl = new EsempioSpotLight();
        esl.setVisible(true);
    }

    public EsempioSpotLight()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Luce SpotLight; Milinese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        // Creo la radice del Branch Graph
        BranchGroup objRoot = new BranchGroup();
        // Creo una sorgente di luce SpotLight posizionata nell'origine,
```

```
// orientata verso (0.0, 0.0, -10.0), color MAGENTA, spread angle 45 gradi
objRoot.addChild(this.creaUnaLuceSpotLight());
// Sfera 1, colore diffuso ROSSO, colore speculare BIANCO
objRoot.addChild(this.creaUnaSfera(-20.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f));
// Sfera 2, colore diffuso ROSSO, colore speculare BIANCO
objRoot.addChild(this.creaUnaSfera(-10.0f, 0.0f, -10.0f, 1.0f, 0.0f, 0.0f));
// Sfera 3, colore diffuso ROSSO, colore speculare BIANCO
objRoot.addChild(this.creaUnaSfera(0.0f, 0.0f, -20.0f, 1.0f, 0.0f, 0.0f));
// Sfera 4, colore diffuso ROSSO, colore speculare BIANCO
objRoot.addChild(this.creaUnaSfera(10.0f, 0.0f, -10.0f, 1.0f, 0.0f, 0.0f));
// Sfera 5, colore diffuso ROSSO, colore speculare BIANCO
objRoot.addChild(this.creaUnaSfera(20.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f));
// Creo un pavimento di riferimento e lo aggiungo alla scena
objRoot.addChild(this.creaRiferimento());
// Restituisco il tutto
return objRoot;
}

private TransformGroup creaUnaSfera(float x, float y, float z, float r, float g, float b)
{
    // Creo un TransformGroup con una trasformazione di traslazione
    // per posizionare gli oggetti dove specificato dai valori x, y e z.
    Transform3D t3d1 = new Transform3D();
    t3d1.setTranslation(new Vector3f(x, y, z));
    TransformGroup tg1 = new TransformGroup(t3d1);
    // Creo una sfera di raggio 2.0f
    Sphere sfera = new Sphere(2.0f);
    // Creo un'Appearance e un Material per l'Appearance.
    // Il Material avrà un colore diffuse color le cui componenti rgb vengono prese dai float r, g e b in ingresso.
    Appearance app = new Appearance();
    Material mat = new Material();
    mat.setDiffuseColor(new Color3f(r, g, b));
    app.setMaterial(mat);
    // Imposto l'Appearance così definita come appearance della sfera
    sfera.setAppearance(app);
    // Aggiungo la sfera al Transform Group
    tg1.addChild(sfera);
    // Restituisco il Transform Group
    return tg1;
}

private SpotLight creaUnaLuceSpotLight()
{
    // Creo una sorgente di luce SpotLight posizionata nell'origine,
    // orientata verso (0.0, 0.0, -10.0), color MAGENTA, spread angle 45 gradi
    SpotLight sL = new SpotLight();
    sL.setColor(new Color3f(1.0f, 0.0f, 1.0f));
    sL.setPosition(new Point3f(0.0f, 0.0f, 0.0f));
    sL.setDirection(0.0f, 0.0f, -10.0f);
    sL.setSpreadAngle((float)(Math.PI/4));
    // Imposto il bound d'azione di questa fonte di luce: una sfera di raggio 200.0
    sL.setInfluencingBounds(new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 200.0));
    // Restituisco la fonte di luce così definita
    return sL;
}

private Shape3D creaRiferimento()
```

```
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
```

ESEMPIO POINT LIGHT CON SCOPE

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra l'utilizzo degli SCOPE per determinare quali oggetti debbano risentire o meno dell'illuminazione di una fonte di luce.

Lo SCOPE é un raggruppamento "logico", effettuato sullo Scene Graph, non "fisico" (basato sulle posizioni) come avviene con i bound (senza scope, le sfere presenti in questa scena risulterebbero tutte illuminate perché tutte poste entro il bound d'azione della luce !).

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioPointLightConScope extends JFrame
{
    public static void main(String[] args)
    {
        EsempioPointLightConScope elplcs = new EsempioPointLightConScope();
        elplcs.setVisible(true);
    }

    public EsempioPointLightConScope()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Luce PointLight con SCOPE; Milanese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
```

```
{
    // Creo la radice del Branch Graph
    BranchGroup objRoot = new BranchGroup();
    // Creo una sorgente di luce PointLight posizionata nell'origine,
    // color MAGENTA, funzione di attenuazione di default
    PointLight sorgenteDiLuce = this.creaUnaLucePointLight();
    objRoot.addChild(sorgenteDiLuce);

    // Creo un primo SCOPE, quello che conterrà gli oggetti da illuminare, e gli "aggancio" tali oggetti
    Group scopeIlluminazioneOn = new Group();
    sorgenteDiLuce.insertScope(scopeIlluminazioneOn, 0);
    // Sfera 1, colore diffuso ROSSO, colore speculare BIANCO
    scopeIlluminazioneOn.addChild(this.creaUnaSfera(-20.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f));
    // Sfera 3, colore diffuso ROSSO, colore speculare BIANCO
    scopeIlluminazioneOn.addChild(this.creaUnaSfera(0.0f, 0.0f, -20.0f, 1.0f, 0.0f, 0.0f));
    // Sfera 5, colore diffuso ROSSO, colore speculare BIANCO
    scopeIlluminazioneOn.addChild(this.creaUnaSfera(20.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f));
    // Sfera 7, colore diffuso ROSSO, colore speculare BIANCO
    scopeIlluminazioneOn.addChild(this.creaUnaSfera(0.0f, 0.0f, 20.0f, 1.0f, 0.0f, 0.0f));

    // Creo il secondo SCOPE con i suoi figli, quelli da non illuminare
    Group scopeIlluminazioneOff = new Group();
    // Sfera 2, colore diffuso ROSSO, colore speculare BIANCO
    scopeIlluminazioneOff.addChild(this.creaUnaSfera(-10.0f, 0.0f, -10.0f, 1.0f, 0.0f, 0.0f));
    // Sfera 4, colore diffuso ROSSO, colore speculare BIANCO
    scopeIlluminazioneOff.addChild(this.creaUnaSfera(10.0f, 0.0f, -10.0f, 1.0f, 0.0f, 0.0f));
    // Sfera 6, colore diffuso ROSSO, colore speculare BIANCO
    scopeIlluminazioneOff.addChild(this.creaUnaSfera(10.0f, 0.0f, 10.0f, 1.0f, 0.0f, 0.0f));
    // Sfera 8, colore diffuso ROSSO, colore speculare BIANCO
    scopeIlluminazioneOff.addChild(this.creaUnaSfera(-10.0f, 0.0f, 10.0f, 1.0f, 0.0f, 0.0f));

    // Aggancio gli scope a objRoot e restituisco il tutto
    objRoot.addChild(scopeIlluminazioneOn);
    objRoot.addChild(scopeIlluminazioneOff);

    // Creo un pavimento di riferimento e lo aggiungo alla scena
    objRoot.addChild(this.creaRiferimento());

    // Restituisco il tutto
    return objRoot;
}

private TransformGroup creaUnaSfera(float x, float y, float z, float r, float g, float b)
{
    // Creo un TransformGroup con una trasformazione di traslazione
    // per posizionare gli oggetti dove specificato dai valori x, y e z.
    Transform3D t3d1 = new Transform3D();
    t3d1.setTranslation(new Vector3f(x, y, z));
    TransformGroup tgl = new TransformGroup(t3d1);
    // Creo una sfera di raggio 2.0f
    Sphere sfera = new Sphere(2.0f);
    // Creo un'Appearance e un Material per l'Appearance. Il Material avrà un colore
    // diffuso color le cui componenti rgb vengono prese dai float r, g e b in ingresso.
    Appearance app = new Appearance();
    Material mat = new Material();
    mat.setDiffuseColor(new Color3f(r, g, b));
    app.setMaterial(mat);
}
```

```
// Imposto l'Appearance così definita come appearance della sfera
sfera.setAppearance(app);
// Aggiungo la sfera al Transform Group
tg1.addChild(sfera);
// Restituisco il Transform Group
return tg1;
}

private PointLight creaUnaLucePointLight()
{
    // Creo una luce di tipo PointLight, impostandone colore, posizione e bound d'influenza
    PointLight pL = new PointLight();
    pL.setColor(new Color3f(1.0f, 0.0f, 1.0f));
    pL.setPosition(new Point3f(0.0f, 0.0f, 0.0f));
    pL.setInfluencingBounds(new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 200.0));
    // Restituisco la Point Light così definita
    return pL;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(1,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(1,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ESEMPIO COLORI DELLE GEOMETRIE 1

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

In questo esempio viene creato un oggetto Shape3D, la cui geometria é un QuadArray con colori dei vertici e provvisto di un'appearance (con colore blu).

Notare che l'appearance blu non verrà visualizzata, in quanto i colori dei vertici "sovrascrivono" i colori dell'appearance.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioColoriDelleGeometrie1 extends JFrame
{
    public static void main(String[] args)
    {
        EsempioColoriDelleGeometrie1 ecdg1 = new EsempioColoriDelleGeometrie1();
        ecdg1.setVisible(true);
    }

    public EsempioColoriDelleGeometrie1()
    {
        this.setTitle("EsempioColoriDelleGeometrie1; Milaneze Francesco; www.redbaron85.com");
        this.setSize(400,400);
        this.setLocation(100,100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = creaLoSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f,0.3f,2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup creaLoSceneGraph()
    {

```

```
// Creo la radice del Branch Group
BranchGroup objRoot = new BranchGroup();
// Aggiungo al Branch Graph una Shape3D, provvista di una sua appearance,
// ma con i colori della geometria impostati esplicitamente
objRoot.addChild(this.disegnaQuadrilateri());
// Restituisco il tutto
return objRoot;
}

private Shape3D disegnaQuadrilateri()
{
    // Creo un oggetto di tipo QuadArray, impostando il numero di vertici e la modalità di creazione
    QuadArray qa = new QuadArray(12, GeometryArray.COORDINATES | GeometryArray.COLOR_3);
    // Imposto le coordinate dei vertici...
    qa.setCoordinate(0, new Point3f(0.0f, 0.0f, 0.0f));
    qa.setCoordinate(1, new Point3f(10.0f, 0.0f, 0.0f));
    qa.setCoordinate(2, new Point3f(10.0f, 10.0f, 0.0f));
    qa.setCoordinate(3, new Point3f(0.0f, 10.0f, 0.0f));
    // ...altre coordinate dei vertici...
    qa.setCoordinate(4, new Point3f(12.0f, 0.0f, 0.0f));
    qa.setCoordinate(5, new Point3f(20.0f, 0.0f, 0.0f));
    qa.setCoordinate(6, new Point3f(27.0f, -10.0f, 10.0f));
    qa.setCoordinate(7, new Point3f(15.0f, -10.0f, 6.0f));
    // ...altre coordinate dei vertici...
    qa.setCoordinate(8, new Point3f(0.0f, 0.0f, 5.0f));
    qa.setCoordinate(9, new Point3f(7.0f, 0.0f, 5.0f));
    qa.setCoordinate(10, new Point3f(7.0f, -2.0f, 9.0f));
    qa.setCoordinate(11, new Point3f(-2.0f, -2.0f, 9.0f));
    // Creo tre colori
    Color3f xc = new Color3f(0.1f, 0.9f, 0.1f);
    Color3f yc = new Color3f(0.9f, 0.1f, 0.1f);
    Color3f zc = new Color3f(0.1f, 0.1f, 0.9f);
    // Coloro i vertici...
    qa.setColor(0,xc);
    qa.setColor(1,yc);
    qa.setColor(2,zc);
    qa.setColor(3,xc);
    // Coloro i vertici...
    qa.setColor(4,yc);
    qa.setColor(5,zc);
    qa.setColor(6,xc);
    qa.setColor(7,yc);
    // Coloro i vertici...
    qa.setColor(8,zc);
    qa.setColor(9,xc);
    qa.setColor(10,yc);
    qa.setColor(11,zc);

    // Di default c'è il culling sulle facce "posteriori", quindi non tutti i poligoni vengono visualizzati...
    // Per risolvere questo problema, creiamo una Appearance con il culling disattivato e la associamo
    // alla Shape3D che avrà come Geometry il QuadArray precedentemente definito
    Shape3D miaShape = new Shape3D(qa);
    Appearance miaAppearance = new Appearance();
    miaAppearance.setPolygonAttributes(new PolygonAttributes(PolygonAttributes.POLYGON_FILL,
    PolygonAttributes.CULL_NONE, 0.0f));
}
```

```
blu...    // Imposto adesso un ColoringAttributes per l'appearance, in modo da colorare (teoricamente) l'oggetto di
          ColoringAttributes ca = new ColoringAttributes(0, 0, 1, ColoringAttributes.NICEST);
          miaAppearance.setColoringAttributes(ca);
          //... ad ogni modo, Java3D non prenderà in esame questo attributo perché i colori delle geometrie
          // "sovrascrivono" quelli dell'appearance

          miaShape.setAppearance(miaAppearance);

          // Restituisco la Shape3D così definita
          return miaShape;
        }
    }
```

ESEMPIO COLORI DELLE GEOMETRIE 2

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

In questo esempio viene creato un oggetto Shape3D, la cui geometria é un QuadArray senza colori dei vertici e provvisto di un'apparence (con colore blu).

In questo caso, a differenza che in EsempioColoriDelleGeometrie1, verrà visualizzato il colore dell'apparence, perché non vi sono colori dei vertici impostati a "sovrascriverlo".

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioColoriDelleGeometrie2 extends JFrame
{
    public static void main(String[] args)
    {
        EsempioColoriDelleGeometrie2 ecdg2 = new EsempioColoriDelleGeometrie2();
        ecdg2.setVisible(true);
    }

    public EsempioColoriDelleGeometrie2()
    {
        this.setTitle("EsempioColoriDelleGeometrie2; Milanese Francesco; www.redbaron85.com");
        this.setSize(400,400);
        this.setLocation(100,100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = creaLoSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f,0.3f,2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup creaLoSceneGraph()
    {

```

```
// Creo la radice del Branch Group
BranchGroup objRoot = new BranchGroup();
// Aggiungo al Branch Graph una Shape3D, provvista di una sua appearance,
// ma con i colori della geometria impostati esplicitamente
objRoot.addChild(this.disegnaQuadrilateri());
// Restituisco il tutto
return objRoot;
}

private Shape3D disegnaQuadrilateri()
{
    // Creo un oggetto di tipo QuadArray, impostando il numero di vertici
    // e la modalità di creazione: stavolta NON colorerò i vertici
    QuadArray qa = new QuadArray(12, GeometryArray.COORDINATES);
    // Imposto le coordinate dei vertici...
    qa.setCoordinate(0, new Point3f(0.0f, 0.0f, 0.0f));
    qa.setCoordinate(1, new Point3f(10.0f, 0.0f, 0.0f));
    qa.setCoordinate(2, new Point3f(10.0f, 10.0f, 0.0f));
    qa.setCoordinate(3, new Point3f(0.0f, 10.0f, 0.0f));
    // ...altre coordinate dei vertici...
    qa.setCoordinate(4, new Point3f(12.0f, 0.0f, 0.0f));
    qa.setCoordinate(5, new Point3f(20.0f, 0.0f, 0.0f));
    qa.setCoordinate(6, new Point3f(27.0f, -10.0f, 10.0f));
    qa.setCoordinate(7, new Point3f(15.0f, -10.0f, 6.0f));
    // ...altre coordinate dei vertici...
    qa.setCoordinate(8, new Point3f(0.0f, 0.0f, 5.0f));
    qa.setCoordinate(9, new Point3f(7.0f, 0.0f, 5.0f));
    qa.setCoordinate(10, new Point3f(7.0f, -2.0f, 9.0f));
    qa.setCoordinate(11, new Point3f(-2.0f, -2.0f, 9.0f));

    // Di default c'è il culling sulle facce "posteriori", quindi non tutti i poligoni vengono visualizzati...
    // Per risolvere questo problema, creiamo una Appearance con il culling disattivato e la associamo
    // alla Shape3D che avrà come Geometry il QuadArray precedentemente definito
    Shape3D miaShape = new Shape3D(qa);
    Appearance miaAppearance = new Appearance();
    miaAppearance.setPolygonAttributes(new PolygonAttributes(PolygonAttributes.POLYGON_FILL,
PolygonAttributes.CULL_NONE, 0.0f));

    // Imposto adesso un ColoringAttributes per l'appearance, in modo da colorare l'oggetto di blu...
    ColoringAttributes ca = new ColoringAttributes(0, 0, 1, ColoringAttributes.NICEST);
    miaAppearance.setColoringAttributes(ca);

    miaShape.setAppearance(miaAppearance);

    // Restituisco la Shape3D così definita
    return miaShape;
}
}
```

ESEMPIO POINT ATTRIBUTES

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come creare un'Apperance per dei punti impostandone gli attributi di PointAttributes per personalizzarne l'aspetto.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioPointAttributes extends JFrame
{
    public static void main(String[] args)
    {
        EsempioPointAttributes epa = new EsempioPointAttributes();
        epa.setVisible(true);
    }
    public EsempioPointAttributes()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Point Attributes; Milaneze Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        // Creo la radice del Branch Graph
        BranchGroup objRoot = new BranchGroup();
        // Creo una Shape3D, impostando come Geometry un insieme di punti (PointArray)
        // e come Appearance la nostra Appearance personalizzata
        // Creo questa Shape3D con un metodo ad hoc, quindi la aggiungo al BranchGroup
    }
}
```

```
objRoot.addChild(this.creaInsiemeDiPunti());
// Aggiungo un pavimento di riferimento alla scena
objRoot.addChild(this.creaRiferimento());
// Restituisco il tutto
return objRoot;
}

private Shape3D creaInsiemeDiPunti()
{
    // Creo una Shape3D, impostando come Geometry un insieme di punti (PointArray)
    // e come Appearance la nostra Appearance personalizzata
    Shape3D miaShape3D = new Shape3D();
    // Creo la Geometry: un array di punti
    PointArray arrayDiPunti = new PointArray(5, PointArray.COORDINATES);
    arrayDiPunti.setCoordinate(0, new Point3f(-2.0f, 1.0f, -2.0f));
    arrayDiPunti.setCoordinate(1, new Point3f(-1.0f, 2.0f, -2.0f));
    arrayDiPunti.setCoordinate(2, new Point3f(0.0f, 3.0f, -2.0f));
    arrayDiPunti.setCoordinate(3, new Point3f(1.0f, 2.0f, -2.0f));
    arrayDiPunti.setCoordinate(4, new Point3f(2.0f, 1.0f, -2.0f));
    // Imposto come geometry della Shape3D l'array di punti appena creato
    miaShape3D.setGeometry(arrayDiPunti);
    // Imposto come appearance della Shape3D una Appearance creata con creaAppearancePointAttributes()
    miaShape3D.setAppearance(creaAppearancePointAttributes());
    // Restituisco la Shape3D così definita
    return miaShape3D;
}

private Appearance creaAppearancePointAttributes()
{
    // Creo un nuovo oggetto Appearance
    Appearance app = new Appearance();
    // Creo un nuovo oggetto PointAttributes, specificando la dimensione dei punti e settando a true
    l'AntiAliasing PointAttributes pa = new PointAttributes(15.0f, true);
    // Imposto come PointAttributes dell'Appearance l'oggetto PointAttributes appena definito
    app.setPointAttributes(pa);
    // Restituisco l'Appearance
    return app;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ESEMPIO LINE ATTRIBUTES

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come impostare un'Appearance e gli attributi LineAttributes per personalizzare l'aspetto delle linee.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioLineAttributes extends JFrame
{
    public static void main(String[] args)
    {
        EsempioLineAttributes ela = new EsempioLineAttributes();
        ela.setVisible(true);
    }

    public EsempioLineAttributes()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Line Attributes; Milanese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        // Creo la radice del Branch Graph
        BranchGroup objRoot = new BranchGroup();
        // Aggiungo un pavimento di riferimento alla scena, provvisto di una semplice geometria
        // E dell'appearance personalizzato, con il LineAttributes impostato ad hoc
    }
}
```

```
objRoot.addChild(this.creaRiferimento());
// Restituisco il tutto
return objRoot;
}

private Appearance creaAppearanceLineAttributes()
{
    // Creo un nuovo oggetto Appearance
    Appearance app = new Appearance();
    // Creo un nuovo oggetto LineAttributes, specificando lo spessore e l'aspetto delle linee ed impostando a
true l'AntiAliasing
    LineAttributes la = new LineAttributes(3.0f, LineAttributes.PATTERN_DASH, true);
    // Imposto come LineAttributes dell'Appearance l'oggetto LineAttributes appena definito
    app.setLineAttributes(la);
    // Restituisco l'Appearance
    return app;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(1,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(1,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    Shape3D miaShape3D = new Shape3D();
    miaShape3D.setGeometry(landGeom);
    miaShape3D.setAppearance(creaAppearanceLineAttributes());
    return miaShape3D;
}
}
```

ESEMPIO POLYGON ATTRIBUTES

// Milanesi Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come impostare un'Appearance con attributi di PolygonAttributes personalizzati ad un oggetto visuale.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioPolygonAttributes extends JFrame
{
    public static void main(String[] args)
    {
        EsempioPolygonAttributes epa = new EsempioPolygonAttributes();
        epa.setVisible(true);
    }
    public EsempioPolygonAttributes()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Polygon Attributes; Milanesi Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        // Creo la radice del Branch Graph
        BranchGroup objRoot = new BranchGroup();
        // Creo un Box, renderizzandolo in modalità "wireframe"
        // (linee che collegano tutti i vertici consecutivi; in pratica, la struttura di un oggetto)
```

```
// Inoltre, imposto come metodo di culling CULL_NONE, per mostrare
// anche le connessioni tra i vertici delle facce nascoste
com.sun.j3d.utils.geometry.Box mioBox = new com.sun.j3d.utils.geometry.Box(0.4f, 0.6f, 0.2f,
creaAppearancePolygonAttributes());

// Non aggiungo il Box direttamente alla scena, ma tramite un TransformGroup
// che serve a ruotarlo un pò, per renderlo un pò più "interessante"
// Creo quindi un TransformGroup...
TransformGroup tg1 = new TransformGroup();
Transform3D t3d1 = new Transform3D();
t3d1.rotZ(Math.PI/4);
Transform3D t3d2 = new Transform3D();
t3d2.rotX(Math.PI/4);
t3d1.mul(t3d2);
tg1.setTransform(t3d1);

// Aggiungo il Box al Transform Group
tg1.addChild(mioBox);

// Aggiungo quindi tale Transform Group a objRoot
objRoot.addChild(tg1);

// Aggiungo un pavimento di riferimento alla scena
objRoot.addChild(this.creaRiferimento());

// Restituisco il tutto
return objRoot;
}
private Appearance creaAppearancePolygonAttributes()
{
    // Creo un nuovo oggetto Appearance
    Appearance app = new Appearance();
    // Creo un nuovo oggetto PolygonAttributes, specificando il metodo di disegno, il culling e l'offset
    PolygonAttributes pa = new PolygonAttributes(PolygonAttributes.POLYGON_LINE,
PolygonAttributes.CULL_NONE, 0.0f);
    // Imposto come PolygonAttributes dell'Appearance l'oggetto PolygonAttributes appena definito
    app.setPolygonAttributes(pa);
    // Restituisco l'Appearance
    return app;
}
private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ESEMPIO COLORING ATTRIBUTES

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come creare un'Apparence per un oggetto visuale impostandone il colore mediante gli attributi di ColoringAttributes.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
```

```
public class EsempioColoringAttributes extends JFrame
{
```

```
    public static void main(String[] args)
```

```
    {
```

```
        EsempioColoringAttributes eca = new EsempioColoringAttributes();
        eca.setVisible(true);
```

```
    }
```

```
    public EsempioColoringAttributes()
```

```
    {
```

```
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Coloring Attributes; Milaneze Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
```

```
    }
```

```
    private BranchGroup createSceneGraph()
```

```
    {
```

```
        // Creo la radice del Branch Graph
        BranchGroup objRoot = new BranchGroup();
        // Aggiungo un Box alla scena. Il colore di un oggetto con Appearance
```

```
// di default é bianco, ma questa volta coloreremo il nostro oggetto di giallo.
objRoot.addChild(new com.sun.j3d.utils.geometry.Box(0.4f, 0.6f, 0.2f,
creaAppearanceColoringAttributes()));
// Aggiungo un pavimento di riferimento alla scena
objRoot.addChild(this.creaRiferimento());
// Restituisco il tutto
return objRoot;
}

private Appearance creaAppearanceColoringAttributes()
{
    // Creo un nuovo oggetto Appearance
    Appearance app = new Appearance();
    // Creo un nuovo oggetto ColoringAttributes, specificando i valori dei canali-colore rgb e lo shading
    ColoringAttributes ca = new ColoringAttributes(1.0f, 1.0f, 0.0f, ColoringAttributes.NICEST);
    // Imposto come ColoringAttributes dell'Appearance l'oggetto ColoringAttributes appena definito
    app.setColoringAttributes(ca);
    // Restituisco l'Appearance
    return app;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ESEMPIO TRANSPARENCY ATTRIBUTES

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come rendere trasparente un oggetto agendo sugli attributi TransparencyAttributes dell'Appearance applicata all'oggetto.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioTransparencyAttributes extends JFrame
{
    public static void main(String[] args)
    {
        EsempioTransparencyAttributes eta = new EsempioTransparencyAttributes();
        eta.setVisible(true);
    }

    public EsempioTransparencyAttributes()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Transparency Attributes; Milanese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        // Creo la radice del Branch Graph
        BranchGroup objRoot = new BranchGroup();
        // Aggiungo un Box alla scena, dal colore originale bianco
    }
}
```

```
// ma con trasparenza al 0.5 (su una scala da 0 a 1)
objRoot.addChild(new com.sun.j3d.utils.geometry.Box(0.4f, 0.6f, 0.2f,
creaAppearanceTransparencyAttributes()));
// Aggiungo un pavimento di riferimento alla scena
objRoot.addChild(this.creaRiferimento());
// Restituisco il tutto
return objRoot;
}

private Appearance creaAppearanceTransparencyAttributes()
{
    // Creo un nuovo oggetto Appearance
    Appearance app = new Appearance();
    // Creo un nuovo oggetto TransparencyAttributes, specificando il metodo di rendering della trasparenza e
    // l'ammontare della stessa
    TransparencyAttributes ta = new TransparencyAttributes(TransparencyAttributes.NICEST, 0.5f);
    // Imposto come TransparencyAttributes dell'Appearance l'oggetto TransparencyAttributes appena definito
    app.setTransparencyAttributes(ta);
    // Restituisco l'Appearance
    return app;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ESEMPIO BORDI IN RILIEVO

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class BordiInRilievo extends JFrame
{
    public static void main(String[] args)
    {
        BordiInRilievo bir = new BordiInRilievo();
        bir.setVisible(true);
    }

    public BordiInRilievo()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio bordi in rilievo; Milanese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 0.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(this.creaRiferimento());
        objRoot.addChild(creaConi(2.0f, 5.0f));
        return objRoot;
    }

    // Metodo per la creazione di un TransformGroup e dei coni suoi figli
    private TransformGroup creaConi(float radius, float height)
    {
        // Creazione di un TransformGroup con trasformazione di traslazione
    }
}
```

```
// (per permetterci di visualizzare subito i coni)
Vector3f vettore = new Vector3f(0.0f, 0.0f, -10.0f);
Transform3D t3d = new Transform3D();
t3d.setTranslation(vettore);
TransformGroup tg = new TransformGroup(t3d);

// Creo, con un metodo esterno, un'appearance per la visualizzazione delle linee
// (modalità di resa dei poligoni POLYGON.LINE) con spessore 2 e colore blu
Appearance blueColorAppearance = getBlueColorAppearance();

// Creo, con un metodo esterno, un'appearance per la visualizzazione delle facce
// con colore rosso e valore di opacità 0.5
Appearance redColorAppearance = getredColorAppearance();

// Creo due coni con raggio e altezza dati ed impostando per il primo un'appearance
// rossa semitrasparente (ottenuta con getredColorAppearance),
// per il secondo l'appearance "fil di ferro" blu ottenuta con getBlueColorAppearance()
Cone cono = new Cone(radius, height);
cono.setAppearance(redColorAppearance);
Cone secondoCono = new Cone(radius, height);
secondoCono.setAppearance(blueColorAppearance);

// Aggiungo i due coni al TransformGroup e restituisco il Transform Group
tg.addChild(cono);
tg.addChild(secondoCono);
return tg;
}

// Creo un'appearance "fil di ferro" blu
private Appearance getBlueColorAppearance()
{
    // Creo un oggetto Appearance
    Appearance blueColorAppearance=new Appearance();
    // Creo gli attributi di resa dei poligoni per l'appearance: imposto cull_none
    // e resa "fil di ferro" o "struttura" (POLYGON_LINE)
    PolygonAttributes polygonAttributes = new PolygonAttributes(PolygonAttributes.POLYGON_LINE,
PolygonAttributes.CULL_NONE, 0.0f);
    blueColorAppearance.setPolygonAttributes(polygonAttributes);
    // Creo gli attributi per colorare l'appearance, in questo caso imposto il colore blu
    ColoringAttributes blueColoring=new ColoringAttributes();
    blueColoring.setColor(0.0f,0.0f,1.0f);
    blueColorAppearance.setColoringAttributes(blueColoring);
    // Imposto la modalità di resa delle linee per questa Appearance, mettendo come spessore "2.0f"
    LineAttributes lineAttrib=new LineAttributes();
    lineAttrib.setLineWidth(2.0f);
    blueColorAppearance.setLineAttributes(lineAttrib);
    // Restituisco l'appearance così definita
    return blueColorAppearance;
}

// Creo un'appearance trasparente rossa
private Appearance getredColorAppearance()
{
    // Creo un oggetto Appearance
    Appearance redColorAppearance=new Appearance();
    // Creo l'attributo per gestire la trasparenza di questa appearance,
    // mettendo come valore di opacità 0.5
```

```
        TransparencyAttributes transparencyAttributes = new
TransparencyAttributes(TransparencyAttributes.NICEST, 0.5f);
        redColorAppearance.setTransparencyAttributes(transparencyAttributes);
        // Creo gli attributi per colorare l'appearance, in questo caso imposto il colore rosso
        ColoringAttributes redColoring=new ColoringAttributes();
        redColoring.setColor(1.0f,0.0f,0.0f);
        redColorAppearance.setColoringAttributes(redColoring);
        // Restituisco l'appearance così definita
        return redColorAppearance;
    }

    private Shape3D creaRiferimento()
    {
        LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
        float l = -50.0f;
        for(int c=0; c<44; c+=4)
        {
            landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
            landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
            landGeom.setCoordinate(c+2, new Point3f(1,0.0f,-50.0f));
            landGeom.setCoordinate(c+3, new Point3f(1,0.0f,50.0f));
            l += 10.0f;
        }
        Color3f c = new Color3f(0.1f,0.8f,0.1f);
        for(int i=0; i<44; i++) landGeom.setColor(i,c);
        return new Shape3D(landGeom);
    }
}
```

ESEMPIO ALTERNATE APPEARANCE

// Milinese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
```

//Questo programma dimostra come sia possibile fornire un aspetto "alternato",
// a seconda della posizione in cui si trova un nodo e delle regioni di influenza degli altri nodi
public class AlternateAppearanceApp extends JFrame

```
{
    public static void main(String args[])
    {
        AlternateAppearanceApp alternate = new AlternateAppearanceApp();
        alternate.setVisible(true);
    }

    public AlternateAppearanceApp()
    {
        this.setSize(400, 400);
        this.setLocation(100, 100);
        this.setTitle("AlternateAppearance");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        scene.compile();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewingPlatform().setNominalViewingTransform();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();

        // ===== Creo un oggetto BoundingSphere
        BoundingSphere bounds = new BoundingSphere();

        // ===== AGGIUNGO UNA LUCE DIREZIONALE,
        // LE ASSOCIO IL BOUND E LA AGGIUNGO ALLA SCENA
        DirectionalLight light = new DirectionalLight(new Color3f(0.7f, 0.7f, 0.7f), new Vector3f(-1.0f, -1.0f, -1.0f));
        light.setInfluencingBounds(bounds);
        objRoot.addChild(light);
        // ===== FINE DELLA SEZIONE DEDICATA ALLA LUCE

        // ===== CREO UNA SFERA DI RAGGIO 0.1,
        // NE IMPOSTO LE CAPABILITY E LE ASSEGNO UN ASPETTO
        Sphere sfera = new Sphere(0.1f, Sphere.GENERATE_NORMALS, 30);
```

```
//Imposta la capacità di sovrascrivere l'aspetto della sfera
sfera.getShape().setAppearanceOverrideEnable(true);
//Crea un aspetto per la sfera
Appearance app = new Appearance();
//Imposta un material per l'aspetto
app.setMaterial(new Material());
//Imposta l'aspetto della sfera
sfera.setAppearance(app);
// ===== CREO UNA SFERA DI RAGGIO 0.1

// ===== CREAZIONE DELL'ALTERNATE APPEARANCE
//Crea un appearance per l'oggetto AlternateAppearance
Appearance app2 = new Appearance();
//Imposta la visualizzazione a linee
PolygonAttributes pa = new PolygonAttributes();
pa.setPolygonMode(pa.POLYGON_LINE);
app2.setPolygonAttributes(pa);
//Imposta un colore diverso per la visualizzazione a linee
ColoringAttributes ca = new ColoringAttributes(0,1,1,ColoringAttributes.SHADE_GOURAUD);
app2.setColoringAttributes(ca);
//Creo l'oggetto AlternateAppearance vero e proprio (una leaf dello scene graph),
// passando al costruttore l'appearance definito precedentemente
AlternateAppearance alternateApp = new AlternateAppearance(app2);
//Imposta il bound dell'AlternateAppearance
alternateApp.setInfluencingBounds(new BoundingSphere(new Point3d(0,0,0),0.2));
//Aggiunge l'Alternate Appearance alla scena
objRoot.addChild(alternateApp);
// ===== FINE DELLA SEZIONE DEDICATA ALL'ALTERNATE APPEARANCE

// ===== LE OPERAZIONI CHE SEGUONO SERVONO A FAR RUOTARE
// LA SFERA SU SE STESSA. USEREMO IL PRIMO BOUND
//Crea un gruppo per le trasformazioni affini
TransformGroup objSpin = new TransformGroup();
//Imposta la capacità di scrivere la trasformazione
objSpin.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
//Crea un timer
Alpha rotationAlpha = new Alpha(-1,8000);
//Crea un interpolatore per le rotazioni collegato con il gruppo di trasformazione
RotationInterpolator rotator = new RotationInterpolator(rotationAlpha, objSpin);
//=====IMPORTANTE=====Imposta un raggio d'azione all'interpolatore
rotator.setSchedulingBounds(bounds);
//aggiunge l'interpolatore alla gruppo di trasformazione
objSpin.addChild(rotator);
// ===== FINE DELLA PARTE LEGATA ALLA ROTAZIONE

// ===== LE OPERAZIONI CHE SEGUONO SERVONO A TRASLARE
// LA SFERA AVANTI E INDIETRO. USEREMO IL PRIMO BOUND
// Creo un gruppo per le trasformazioni affini
TransformGroup objTrans = new TransformGroup();
//Imposta la capacità di scrivere la trasformazione
objTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
//Crea un timer
Alpha translationAlpha = new Alpha();
translationAlpha.setMode(Alpha.DECREASING_ENABLE+Alpha.INCREASING_ENABLE);
```

```
translationAlpha.setLoopCount(-1);
translationAlpha.setDecreasingAlphaDuration(5000);
translationAlpha.setIncreasingAlphaDuration(5000);
//Crea un interpolatore per le posizioni collegato con il gruppo di trasformazione
PositionInterpolator translator = new PositionInterpolator(translationAlpha,objTrans);
translator.setStartPosition(-1);
translator.setEndPosition(1);
//Imposta un raggio d'azione all'interpolatore
translator.setSchedulingBounds(bounds);
//aggiunge l'interpolatore alla gruppo di trasformazione
objTrans.addChild(translator);
// ===== FINE DELLA PARTE LEGATA ALLA TRASLAZIONE
```

```
// ===== ASSEMBLIAMO LA SCENA E RESTITUIAMO OBJROOT
objSpin.addChild(sfera);
objTrans.addChild(objSpin);
objRoot.addChild(objTrans);
return objRoot;
```

```
}
}
```

ESEMPIO ALTERNATE APPEARANCE APP 2

// Milanesi Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come utilizzare una leaf AlternateAppearance per sovrascrivere l'appearance degli oggetti quando questi entrano nel bound d'azione di un AlternateAppearance, SE il gruppo dal quale discendono è uno dei gruppi di scope dell'AlternateAppearance.

Se l'AA non ha gruppi di scope impostati, lo scope è tutto lo scene graph, dunque tutti gli oggetti (che si trovano nel bound d'azione) subiranno l'effetto dell'AA.

Per verificare quanto appena detto, commentare la riga di codice con l'istruzione "aa.addScope(sfera1);", ricompilare ed eseguire.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
```

```
public class AlternateAppearanceApp2 extends JFrame
{
    public static void main(String args[])
    {
        AlternateAppearanceApp2 alternate2 = new AlternateAppearanceApp2();
        alternate2.setVisible(true);
    }

    public AlternateAppearanceApp2()
    {
        this.setSize(400, 400);
        this.setLocation(100, 100);
        this.setTitle("AlternateAppearance2");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        scene.compile();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewingPlatform().setNominalViewingTransform();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(getDirectionalLight());

        // ===== Creo due TransformGroup, ciascuno dei quali avrà per figlio
        // un transformgroup di rotazione e una sfera, e li aggiungo alla scena (ma in posizioni diverse)
        TransformGroup sfera1 = sferaRotanteInMovimento(0.0f, 0.2f, 0.0f);
        TransformGroup sfera2 = sferaRotanteInMovimento(0.0f, 0.0f, 0.0f);
```

```
objRoot.addChild(sfera1);
objRoot.addChild(sfera2);

// ===== Creo un oggetto AlternateAppearance, aggiungo alla sua lista di Scope
// solo uno dei due TG creati al passo precedente e lo aggiungo alla scena
// In questo modo, lo scope dell'AA sarà limitato ai figli del tg "sfera1".
// Lasciando vuota la lista dei gruppi di scope di un AA, lo scope sarà tutto lo scene graph,
// dunque tutti gli oggetti (presenti nel bound d'azione) subiranno l'effetto di AA.
// Per verificare quanto appena detto, commentare la riga
// con l'istruzione "aa.addScope(sfera1);", ricompilare ed eseguire.
AlternateAppearance aa = aa();
aa.addScope(sfera1);
objRoot.addChild(aa);

// Restituisco il tutto
return objRoot;
}

public DirectionalLight getDirectionalLight()
{
    BoundingSphere bounds = new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 100.0);
    DirectionalLight light = new DirectionalLight(new Color3f(0.7f,0.7f,0.7f),new Vector3f(-1.0f,-1.0f,-1.0f));
    light.setInfluencingBounds(bounds);
    return light;
}

public TransformGroup sferaRotanteInMovimento(float x, float y, float z)
{
    // ===== Creo un oggetto BoundingSphere per definire il bound d'azione delle trasformazioni
    BoundingSphere bounds = new BoundingSphere();

    // Posiziono a xyz tutto il gruppo
    Vector3f vettore = new Vector3f(x, y, z);
    Transform3D t3d1 = new Transform3D();
    t3d1.setTranslation(vettore);
    TransformGroup tgDiPosizionamento = new TransformGroup(t3d1);

    // ===== LE OPERAZIONI CHE SEGUONO SERVONO A FAR RUOTARE LA SFERA SU SE
    // Creo un gruppo per le trasformazioni affini
    TransformGroup objSpin = new TransformGroup();
    // Imposto la capacità di scrivere la trasformazione
    objSpin.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    // Creo un timer
    Alpha rotationAlpha = new Alpha(-1,8000);
    // Creo un interpolatore per le rotazioni collegato con il gruppo di trasformazione
    RotationInterpolator rotator = new RotationInterpolator(rotationAlpha, objSpin);
    // =====IMPORTANTE===== Imposto un raggio d'azione all'interpolatore
    rotator.setSchedulingBounds(bounds);
    // Aggiungo l'interpolatore alla gruppo di trasformazione
    objSpin.addChild(rotator);
    // ===== FINE DELLA PARTE LEGATA ALLA ROTAZIONE

    // ===== LE OPERAZIONI CHE SEGUONO SERVONO A TRASLARE
    // LA SFERA AVANTI E INDIETRO. USEREMO IL PRIMO BOUND
    // Creo un gruppo per le trasformazioni affini
    TransformGroup objTrans = new TransformGroup();
```

STESSA

```
// Imposto la capacità di scrivere la trasformazione
objTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
// Creo un timer e ne imposto i parametri
Alpha translationAlpha = new Alpha();
translationAlpha.setMode(Alpha.DECREASING_ENABLE+Alpha.INCREASING_ENABLE);
translationAlpha.setLoopCount(-1);
translationAlpha.setDecreasingAlphaDuration(5000);
translationAlpha.setIncreasingAlphaDuration(5000);
// Creo un interpolatore per le posizioni collegato con il gruppo di trasformazione
PositionInterpolator translator = new PositionInterpolator(translationAlpha,objTrans);
translator.setStartPosition(-1);
translator.setEndPosition(1);
// Imposto un raggio d'azione all'interpolatore
translator.setSchedulingBounds(bounds);
// Aggiungo l'interpolatore alla gruppo di trasformazione
objTrans.addChild(translator);
// ===== FINE DELLA PARTE LEGATA ALLA TRASLAZIONE
```

ASSEGNO UN ASPETTO
// ===== CREO UNA SFERA DI RAGGIO 0.1, NE IMPOSTO LE CAPABILITY E LE

```
Sphere sfera = new Sphere(0.1f, Sphere.GENERATE_NORMALS, 30);
// Imposto la capacità di sovrascrivere l'aspetto della sfera
sfera.getShape().setAppearanceOverrideEnable(true);
// Imposto l'appearance per la sfera
sfera.setAppearance(appearanceDellaSfera());
// ===== CREO UNA SFERA DI RAGGIO 0.1
```

```
// Assembla la catena di tg, aggiungo come foglia la sfera appena definita e restituisco il tutto
tgDiPosizionamento.addChild(objTrans);
objTrans.addChild(objSpin);
objSpin.addChild(sfera);
return tgDiPosizionamento;
```

```
}
```

// Questo metodo serve a creare un appearance per le sfere
public Appearance appearanceDellaSfera()

```
{
    // Creo un aspetto per le sfere
    Appearance app1 = new Appearance();
    // Imposto un material per l'aspetto
    app1.setMaterial(new Material());
    // Restituisco l'appearance
    return app1;
}
```

// Creo un oggetto AlternateAppearance. E' una leaf dello scene graph.
public AlternateAppearance aa()

```
{
    // Creo un aspetto per l'oggetto AlternateAppearance: gli oggetti della scena
    // influenzati da questo AA avranno l'aspetto qui impostato
    Appearance app2 = new Appearance();
    // Imposto la visualizzazione a linee
    PolygonAttributes pa = new PolygonAttributes();
    pa.setPolygonMode(pa.POLYGON_LINE);
    app2.setPolygonAttributes(pa);
}
```

```
precedenti // Imposto un colore diverso per la visualizzazione a linee
ColoringAttributes ca = new ColoringAttributes(0,1,1,ColoringAttributes.SHADE_GOURAUD);
app2.setColoringAttributes(ca);
// Creo l'oggetto AlternateAppearance vero e proprio, passandogli al costruttore l'aspetto definito nei passi
AlternateAppearance alternateApp = new AlternateAppearance(app2);
// Imposto il bound d'azione dell'AlternateAppearance
alternateApp.setInfluencingBounds(new BoundingSphere(new Point3d(0,0,0), 0.4));
// Restituisco l'AlternateAppearance
return alternateApp;
}
```

ESEMPIO MATERIAL

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come applicare dei materiali
(creati con varie combinazioni di colore ambientale, colore diffuso, colore d'emissione,
colore speculare e lucentezza) a degli oggetti visuali.

Nella scena é presente, posizionata nell'origine e con colore bianco, una Point Light per illuminare gli oggetti.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioMaterial extends JFrame
{
    public static void main(String[] args)
    {
        EsempioMaterial em = new EsempioMaterial();
        em.setVisible(true);
    }

    public EsempioMaterial()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Material; Milanese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {

```

```
// Creo la radice dello Scene Graph
BranchGroup objRoot = new BranchGroup();

// Adesso creerò delle sfere con posizioni e materiali propri.
// Il metodo che permette di crearle ha questa firma:
// creaTGsfera(float coordX, float coordY, float coordZ, Color3f ambientColor,
// Color3f emissiveColor, Color3f diffuseColor, Color3f specularColor, float shininess);
TransformGroup tg1 = creaTGsfera(-2.0f, 0.0f, -10.0f, new Color3f(0.2f, 0.2f, 0.2f), new Color3f(0.0f,
0.0f, 0.0f), new Color3f(0.0f, 1.0f, 1.0f), new Color3f(0.0f, 0.0f, 0.0f), 0.2f);
TransformGroup tg2 = creaTGsfera(-1.0f, 1.0f, -10.0f, new Color3f(0.2f, 0.2f, 0.2f), new Color3f(1.0f,
0.0f, 0.0f), new Color3f(0.0f, 0.0f, 0.0f), new Color3f(0.0f, 0.0f, 0.0f), 0.0f);
TransformGroup tg3 = creaTGsfera(0.0f, 2.0f, -10.0f, new Color3f(0.2f, 0.2f, 0.2f), new Color3f(0.0f, 0.0f,
0.0f), new Color3f(0.4f, 0.4f, 0.4f), new Color3f(0.0f, 0.0f, 0.0f), 0.6f);
TransformGroup tg4 = creaTGsfera(1.0f, 1.0f, -10.0f, new Color3f(0.2f, 0.2f, 0.2f), new Color3f(0.0f, 0.0f,
0.0f), new Color3f(1.0f, 0.2f, 0.0f), new Color3f(0.0f, 0.0f, 0.0f), 0.2f);
TransformGroup tg5 = creaTGsfera(2.0f, 0.0f, -10.0f, new Color3f(0.2f, 0.2f, 0.2f), new Color3f(0.0f, 0.0f,
0.0f), new Color3f(1.0f, 0.2f, 0.0f), new Color3f(1.0f, 1.0f, 1.0f), 1.0f);

// Aggiungo i TransformGroup con le sfere al BranchGroup
objRoot.addChild(tg1);
objRoot.addChild(tg2);
objRoot.addChild(tg3);
objRoot.addChild(tg4);
objRoot.addChild(tg5);

// Devo aggiungere (almeno) una fonte di luce o non vedrò niente:
// Material imposta l'aspetto degli oggetti ILLUMINATI
// NOTARE COME LA SFERA CON COLORE EMISSIVO NON NULLO
// SIA VISIBILE ANCHE DA DIETRO (dove non é illuminata e le altre sono nere).
PointLight pL = new PointLight();
pL.setColor(new Color3f(1.0f, 1.0f, 1.0f));
pL.setPosition(new Point3f(0.0f, 1.0f, 2.0f));
pL.setInfluencingBounds(new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 200.0));
objRoot.addChild(pL);

// Aggiungo un pavimento di riferimento allo Scene Graph
objRoot.addChild(this.creaRiferimento());

// Restituisco lo Scene Graph così composto
return objRoot;
}

private TransformGroup creaTGsfera(float coordX, float coordY, float coordZ, Color3f ambientColor, Color3f
emissiveColor, Color3f diffuseColor, Color3f specularColor, float shininess)
{
    // Creo la radice di questo sotto-grafo: un TransformGroup
    TransformGroup tg = new TransformGroup();

    // Imposto la trasformazione di traslazione per il TransformGroup (posiziona la sfera)
    Transform3D t3d = new Transform3D();
    t3d.setTranslation(new Vector3f(coordX, coordY, coordZ));
    tg.setTransform(t3d);

    // Creo la sfera
    Sphere miaSfera = new Sphere(0.4f);

    // Creo l'Appearance per la sfera
```

```
shininess);

Appearance app = new Appearance();

// Creo il Material e lo imposto all'Appearance
Material mioMateriale = new Material(ambientColor, emissiveColor, diffuseColor, specularColor,

app.setMaterial(mioMateriale);

// Imposto l'Appearance della sfera
miaSfera.setAppearance(app);

// Aggiungo la sfera al TransformGroup
tg.addChild(miaSfera);

// Restituisco il TransformGroup
return tg;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ESEMPIO TEXTURE 1

// Milinese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come caricare un'immagine da file, crearne un Texture ed applicare tale texture ad un Appearance.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import com.sun.j3d.utils.image.TextureLoader;

public class EsempioTexture1 extends JFrame
{
    public static void main(String[] args)
    {
        EsempioTexture1 et1 = new EsempioTexture1();
        et1.setVisible(true);
    }

    public EsempioTexture1()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Texture 1; Milinese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
```

```
primo esempio    // Creo una Appearance con una Texture. Non c'è alcuna gestione delle coordinate della Texture, in questo
                  Appearance miaApp = creaUnaAppearanceConTexture();

                  // Creo un box impostandogli come Appearance "miaApp"
                  com.sun.j3d.utils.geometry.Box mioBox = new com.sun.j3d.utils.geometry.Box(0.4f, 0.6f, 0.2f, miaApp);

                  // Aggiungo il box allo Scene Graph
                  objRoot.addChild(mioBox);
                  // Creo un pavimento di riferimento e lo aggiungo allo Scene Graph
                  objRoot.addChild(this.creaRiferimento());
                  // Restituisco il tutto
                  return objRoot;
            }

            private Appearance creaUnaAppearanceConTexture()
            {
                // Creo un nuovo oggetto Appearance
                Appearance app = new Appearance();

                // Carico una texture
                TextureLoader TL = new TextureLoader("Texture.jpg", this);
                ImageComponent2D immagine = TL.getImage();
                Texture2D texture = new Texture2D(Texture.BASE_LEVEL, Texture.RGBA, immagine.getWidth(),
immagine.getHeight());
                texture.setImage(0, immagine);

                // Imposto la texture nell'Appearance
                app.setTexture(texture);

                // Restituisco l'oggetto Appearance
                return app;
            }

            private Shape3D creaRiferimento()
            {
                LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
                float l = -50.0f;
                for(int c=0; c<44; c+=4)
                {
                    landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
                    landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
                    landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
                    landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
                    l += 10.0f;
                }
                Color3f c = new Color3f(0.1f,0.8f,0.1f);
                for(int i=0; i<44; i++) landGeom.setColor(i,c);
                return new Shape3D(landGeom);
            }
        }
```

ESEMPIO TEXTURE 2

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come caricare un'immagine da file, crearne un Texture ed applicare tale texture ad un Appearance, lasciando a TexCoordGeneration il compito di mappare la texture sulla geometria (a modo suo..).

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import com.sun.j3d.utils.image.TextureLoader;
```

```
public class EsempioTexture2 extends JFrame
{
    public static void main(String[] args)
    {
        EsempioTexture2 et2 = new EsempioTexture2();
        et2.setVisible(true);
    }

    public EsempioTexture2()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Texture 2; Milanese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        // Creo la radice del Branch Group
        BranchGroup objRoot = new BranchGroup();
```

```
// Creo una Appearance con una Texture. L'impostazione delle coordinate della texture é fatta in maniera
"automatica", con TexCoordGenerator
Appearance miaApp = creaUnaAppearanceConTexture();

// Creo un box impostandogli come Appearance "miaApp"
com.sun.j3d.utils.geometry.Box mioBox = new com.sun.j3d.utils.geometry.Box(0.4f, 0.6f, 0.2f, miaApp);

// Aggiungo il box allo Scene Graph
objRoot.addChild(mioBox);

// Creo un pavimento di riferimento e lo aggiungo allo Scene Graph
objRoot.addChild(this.creaRiferimento());

// Restituisco il tutto
return objRoot;
}

private Appearance creaUnaAppearanceConTexture()
{
    // Creo un nuovo oggetto Appearance
    Appearance app = new Appearance();
    // Carico una texture
    TextureLoader TL = new TextureLoader("Texture.jpg", this);
    ImageComponent2D immagine = TL.getImage();
    Texture2D texture = new Texture2D(Texture.BASE_LEVEL, Texture.RGBA, immagine.getWidth(),
immagine.getHeight());
    texture.setImage(0, immagine);
    // Imposto la texture nell'Appearance
    app.setTexture(texture);
    // Creo un TexCoordGeneration, con le impostazioni di default
    TexCoordGeneration mioTCG = new TexCoordGeneration();
    // Imposto il TexCoordGeneration nell'Appearance
    app.setTexCoordGeneration(mioTCG);
    // Restituisco l'oggetto Appearance
    return app;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(1,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(1,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ESEMPIO TEXTURE 3

// Milinese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra alcune combinazioni possibili per applicare una Texture ad oggetti con geometrie differenti (cubi e sfere) con TexCoordGeneration.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import com.sun.j3d.utils.image.TextureLoader;

public class EsempioTexture3 extends JFrame
{
    public static void main(String[] args)
    {
        EsempioTexture3 et3 = new EsempioTexture3();
        et3.setVisible(true);
    }

    public EsempioTexture3()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Texture 3; Milinese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        // Creo la radice del Branch Group
        BranchGroup objRoot = new BranchGroup();
```

```
// Creo una diverse Appearance che si differenziano tra loro per come viene creato (ossia, con quali
parametri) il TexCoordGeneration che gestisce la mappatura della Texture
Appearance miaApp1 = creaUnaAppearanceConTexture(TexCoordGeneration.OBJECT_LINEAR,
TexCoordGeneration.TEXTURE_COORDINATE_2);
Appearance miaApp2 = creaUnaAppearanceConTexture(TexCoordGeneration.OBJECT_LINEAR,
TexCoordGeneration.TEXTURE_COORDINATE_2);
Appearance miaApp3 = creaUnaAppearanceConTexture(TexCoordGeneration.EYE_LINEAR,
TexCoordGeneration.TEXTURE_COORDINATE_2);
Appearance miaApp4 = creaUnaAppearanceConTexture(TexCoordGeneration.EYE_LINEAR,
TexCoordGeneration.TEXTURE_COORDINATE_2);
Appearance miaApp5 = creaUnaAppearanceConTexture(TexCoordGeneration.SPHERE_MAP,
TexCoordGeneration.TEXTURE_COORDINATE_2);
Appearance miaApp6 = creaUnaAppearanceConTexture(TexCoordGeneration.SPHERE_MAP,
TexCoordGeneration.TEXTURE_COORDINATE_2);

// Creo una serie di oggetti visuali, posizionandoli in varie parti della scena, applicando a ciascuno di essi
una Appearance scelta tra quelle create precedentemente
// Per ogni coppia di mode di TexCoordGeneration creo una Sfera e un Box.
TransformGroup tg1 = creaOggettiVisuali(-3.0f, 0.0f, -5.0f, new com.sun.j3d.utils.geometry.Box(1.0f,
1.0f, 1.0f, miaApp1));
TransformGroup tg2 = creaOggettiVisuali(-6.0f, 2.0f, -5.0f, new Sphere(1.5f, miaApp2));
TransformGroup tg3 = creaOggettiVisuali(-3.0f, 4.0f, -5.0f, new com.sun.j3d.utils.geometry.Box(1.0f,
1.0f, 1.0f, miaApp3));
TransformGroup tg4 = creaOggettiVisuali(3.0f, 4.0f, -5.0f, new Sphere(1.5f, miaApp4));
TransformGroup tg5 = creaOggettiVisuali(6.0f, 2.0f, -5.0f, new com.sun.j3d.utils.geometry.Box(1.0f, 1.0f,
1.0f, miaApp5));
TransformGroup tg6 = creaOggettiVisuali(3.0f, 0.0f, -5.0f, new Sphere(1.5f, miaApp6));

// Aggiungo gli oggetti visuali (o meglio, i TransformGroup che li contengono) allo scene graph
objRoot.addChild(tg1);
objRoot.addChild(tg2);
objRoot.addChild(tg3);
objRoot.addChild(tg4);
objRoot.addChild(tg5);
objRoot.addChild(tg6);

// Aggiungo un pavimento di riferimento allo Scene Graph
objRoot.addChild(this.creaRiferimento());

// Restituisco il tutto
return objRoot;
}

private Appearance creaUnaAppearanceConTexture(int mode, int format)
{
    // Creo un nuovo oggetto Appearance
    Appearance app = new Appearance();

    // Carico una texture
    TextureLoader TL = new TextureLoader("Terra.jpg", this);
    ImageComponent2D immagine = TL.getImage();
    Texture2D texture = new Texture2D(Texture.BASE_LEVEL, Texture.RGBA, immagine.getWidth(),
immagine.getHeight());
    texture.setImage(0, immagine);

    // Imposto la texture nell'Appearance
```

```
app.setTexture(texture);

// Creo un TexCoordGeneration, con le impostazioni specificate prima
TexCoordGeneration mioTCG = new TexCoordGeneration(mode, format);

// Imposto il TexCoordGeneration nell'Appearance
app.setTexCoordGeneration(mioTCG);

// Restituisco l'oggetto Appearance
return app;
}

private TransformGroup creaOggettiVisuali(float coordX, float coordY, float coordZ, Node oggetto)
{
    // Creo il TransformGroup che conterrà e posizionerà l'oggetto visuale
    TransformGroup tg = new TransformGroup();

    // Creo la trasformazione di traslazione per il TransformGroup appena creato
    Transform3D t3d = new Transform3D();
    t3d.setTranslation(new Vector3f(coordX, coordY, coordZ));
    tg.setTransform(t3d);

    // Aggiungo al TransformGroup l'oggetto visuale con la sua appearance
    tg.addChild(oggetto);

    // Restituisco il tutto
    return tg;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ESEMPIO TEXTURE 4

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come applicare una Texture ad una geometria mappando le coordinate dei vertici della texture ai vertici della geometria manualmente.

Ricordo che le coordinate dei vertici di una geometria, intese come s (orizzontale) e t (verticale) variano in entrambe le direzioni nell'intervallo [0.0f, 1.0f], indipendentemente dalle dimensioni (width ed height, in pixel) dell'immagine usata come texture !!!

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import com.sun.j3d.utils.image.TextureLoader;

public class EsempioTexture4 extends JFrame
{
    public static void main(String[] args)
    {
        EsempioTexture4 et4 = new EsempioTexture4();
        et4.setVisible(true);
    }

    public EsempioTexture4()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Texture 4; Milaneze Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
```

```
{
    // Creo la radice dello Scene Graph
    BranchGroup objRoot = new BranchGroup();

    // Creo un nuovo oggetto visuale
    Shape3D miaShape = miaShape3D();

    // Aggancio l'oggetto visuale alla radice dello Scene Graph
    objRoot.addChild(miaShape);

    // Aggiungo un pavimento di riferimento alla scena
    objRoot.addChild(this.creaRiferimento());

    // Restituisco il tutto
    return objRoot;
}

private Shape3D miaShape3D()
{
    // Creo un oggetto visuale...
    Shape3D miaShape3D = new Shape3D();

    // =====

    // PARTE 1: CREO LA GEOMETRIA
    // Si tratta di un QuadArray.
    QuadArray miaGeometria = new QuadArray(4, QuadArray.COORDINATES |
QuadArray.TEXTURE_COORDINATE_2);
    Point3f punto = new Point3f();
    punto.set(-5,10,20);
    miaGeometria.setCoordinate(0, punto);
    punto.set(-5, 10, -20);
    miaGeometria.setCoordinate(1, punto);
    punto.set(5, 10, -20);
    miaGeometria.setCoordinate(2, punto);
    punto.set(5, 10, 20);
    miaGeometria.setCoordinate(3, punto);
    // Imposto tale geometria come Geometry di miaShape3D
    miaShape3D.setGeometry(miaGeometria);

    // =====

    // PARTE 2: MAPPATURA DELLE COORDINATE DELLA TEXTURE SULLA GEOMETRY
    // Notate come ancora non é stata definita alcuna texture o appearance: la mappatura delle coordinate di
una texture sui vertici riguarda solo la Geometry !!!
    // Creo un oggetto TexCoord2f
    TexCoord2f coordinate = new TexCoord2f();
    // NOTA IMPORTANTE: le coordinate x e y di una texture variano nell'intervallo 0.0f-1.0f,
indipendentemente dalla larghezza e dall'altezza in pixel dell'immagine usata come texture.
    // Imposto il primo punto sulla texture e lo assegno al vertice della geometria di indice 0
    coordinate.set(0.0f, 0.0f);
    miaGeometria.setTextureCoordinate(0, 0, coordinate);
    // Imposto il secondo punto sulla texture e lo assegno al vertice della geometria di indice 1
    coordinate.set(0.0f, 1.0f);
    miaGeometria.setTextureCoordinate(0, 1, coordinate);
    // Imposto il terzo punto sulla texture e lo assegno al vertice della geometria di indice 2
    coordinate.set(1.0f, 1.0f);
```

```
miaGeometria.setTextureCoordinate(0, 2, coordinate);
// Imposto il quarto punto sulla texture e lo assegno al vertice della geometria di indice 3
coordinate.set(1.0f, 0.0f);
miaGeometria.setTextureCoordinate(0, 3, coordinate);

// =====

// PARTE 3: CREO L'APPEARANCE DI BASE, CARICANDO LA TEXTURE
// Creo un nuovo oggetto Appearance, con gli attributi PolygonAttributes impostati (altrimenti non vedrò
niente in ogni caso)
Appearance miaAppearance = new Appearance();
miaAppearance.setPolygonAttributes(new PolygonAttributes(PolygonAttributes.POLYGON_FILL,
PolygonAttributes.CULL_NONE, 0.0f));
// Carico una texture
TextureLoader TL = new TextureLoader("Sistina.jpg", this);
ImageComponent2D immagine = TL.getImage();
Texture2D texture = new Texture2D(Texture.BASE_LEVEL, Texture.RGBA, immagine.getWidth(),
immagine.getHeight());
texture.setImage(0, immagine);
// Imposto la texture nell'Appearance
miaAppearance.setTexture(texture);
// Imposto l'Appearance in miaShape3D
miaShape3D.setAppearance(miaAppearance);

// =====

// Restituisco l'oggetto visuale così composto
return miaShape3D;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ESEMPIO BOUNDARY MODE

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra i possibili BoundaryMode per determinare il comportamento di una texture quando questa viene applicata ad oggetti con dimensioni maggiori della propria.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import com.sun.j3d.utils.image.TextureLoader;

public class EsempioBoundaryMode extends JFrame
{
    public static void main(String[] args)
    {
        EsempioBoundaryMode ebm = new EsempioBoundaryMode();
        ebm.setVisible(true);
    }

    public EsempioBoundaryMode()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Boundary Mode; Milaneze Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        // Creo la radice dello Scene Graph
        BranchGroup objRoot = new BranchGroup();
```

```
// Creo una prima Shape3D, la cui Geometry é un QuadArray e la cui texture ha il BoundaryMode per S a
Clamping e per T a Clamping
objRoot.addChild(miaShape3D(-20.0f, Texture2D.CLAMP, Texture2D.CLAMP));
// Creo una prima Shape3D, la cui Geometry é un QuadArray e la cui texture ha il BoundaryMode per S a
Clamping e per T a Wrapping
objRoot.addChild(miaShape3D(-10.0f, Texture2D.CLAMP, Texture2D.WRAP));
// Creo una prima Shape3D, la cui Geometry é un QuadArray e la cui texture ha il BoundaryMode per S a
Wrapping e per T a Clamping
objRoot.addChild(miaShape3D(0.0f, Texture2D.WRAP, Texture2D.CLAMP));
// Creo una prima Shape3D, la cui Geometry é un QuadArray e la cui texture ha il BoundaryMode per S a
Wrapping e per T a Wrapping
objRoot.addChild(miaShape3D(10.0f, Texture2D.WRAP, Texture2D.WRAP));

// Aggiungo un pavimento di riferimento alla scena
objRoot.addChild(this.creaRiferimento());

// Restituisco il tutto
return objRoot;
}

private Shape3D miaShape3D(float xCoord, int boundaryModeS, int boundaryModeT)
{
    // Creo la nuova Shape3D
    Shape3D miaShape = new Shape3D();

    // Creo la Geometry e la imposto come Geometry di miaShape
    QuadArray miaGeometria = new QuadArray(4, QuadArray.COORDINATES |
QuadArray.TEXTURE_COORDINATE_2);
    Point3f punto = new Point3f();
    punto.set(xCoord, 0.0f, -10.0f);
    miaGeometria.setCoordinate(0, punto);
    punto.set(xCoord, 10.0f, -10.0f);
    miaGeometria.setCoordinate(1, punto);
    punto.set(xCoord+5.0f, 10.0f, -10.0f);
    miaGeometria.setCoordinate(2, punto);
    punto.set(xCoord+5.0f, 0.0f, -10.0f);
    miaGeometria.setCoordinate(3, punto);
    miaShape.setGeometry(miaGeometria);

    // Creo una Appearance e carico una Texture, impostandola come texture dell'Appearance.
    // Tra le altre cose, userò un TexCoordGeneration e imposterò i Boundary Mode
    Appearance miaAppearance = new Appearance();
    miaAppearance.setPolygonAttributes(new PolygonAttributes(PolygonAttributes.POLYGON_FILL,
PolygonAttributes.CULL_NONE, 0.0f));
    TextureLoader TL = new TextureLoader("Wood.jpg", this);
    ImageComponent2D immagine = TL.getImage();
    Texture2D texture = new Texture2D(Texture.BASE_LEVEL, Texture.RGBA, immagine.getWidth(),
immagine.getHeight());
    texture.setImage(0, immagine);
    miaAppearance.setTexture(texture);
    TexCoordGeneration mioTCG = new TexCoordGeneration();
    miaAppearance.setTexCoordGeneration(mioTCG);
    texture.setBoundaryModeS(boundaryModeS);
    texture.setBoundaryModeT(boundaryModeT);

    // Imposto la nuova Appearance come Appearance di miaShape
```

```
        miaShape.setAppearance(miaAppearance);

        // Restituisco la Shape3D così composta
        return miaShape;
    }

    private Shape3D creaRiferimento()
    {
        LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
        float l = -50.0f;
        for(int c=0; c<44; c+=4)
        {
            landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
            landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
            landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
            landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
            l += 10.0f;
        }
        Color3f c = new Color3f(0.1f,0.8f,0.1f);
        for(int i=0; i<44; i++) landGeom.setColor(i,c);
        return new Shape3D(landGeom);
    }
}
```

ESEMPIO BILLBOARD

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra un'animazione dipendente in maniera "indiretta" dalle azioni dell'utente: Billboard infatti modifica modifica l'orientamento di un oggetto in modo da fargli "seguire" sempre la posizione dell'osservatore.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioBillboard1 extends JFrame
{
    public static void main(String[] args)
    {
        EsempioBillboard1 eb1 = new EsempioBillboard1();
        eb1.setVisible(true);
    }

    public EsempioBillboard1()
    {
        this.setTitle("EsempioBillboard1; Milaneze Francesco; www.redbaron85.com");
        this.setSize(400,400);
        this.setLocation(100,100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = creaLoSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f,0.3f,2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup creaLoSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(this.insiemeBillboard());
        objRoot.addChild(this.disegnaGriglia());
    }
}
```

```
        return objRoot;
    }

    private BranchGroup insiemeBillboard()
    {
        // Creo un BranchGroup per questo sotto-grafo
        BranchGroup objRootInterno = new BranchGroup();

        // Creo un primo TransformGroup al quale applico una trasformazione di traslazione
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.0f, -5.0f));
        TransformGroup tg1 = new TransformGroup(t3d);

        // Creo un secondo TransformGroup, impostando la capability di TRANSFORM_WRITE, e gli aggiungo
        come figlio un Text2D
        TransformGroup tg2 = new TransformGroup();
        tg2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        tg2.addChild(new Text2D("Ciao !", new Color3f(1.0f, 1.0f, 0.0f), "sansserif", 100, Font.PLAIN));

        // Creo un oggetto Billboard, associandola al secondo TransformGroup (che controlla il testo 2D) ed
        impostandole un bound d'azione
        Billboard miaBillboard = new Billboard(tg2);
        BoundingSphere bounds = new BoundingSphere();
        miaBillboard.setSchedulingBounds(bounds);

        // Aggiungo al primo TransformGroup (che specifica la posizione dell'oggetto) il secondo
        TransformGroup
        tg1.addChild(tg2);
        // Aggiungo il primo Transform Group alla radice di questo sotto-grafo
        objRootInterno.addChild(tg1);
        // Aggiungo la Billboard alla radice di questo sotto-grafo
        objRootInterno.addChild(miaBillboard);

        // Restituisco la radice del sotto-grafo
        return objRootInterno;
    }

    private Shape3D disegnaGriglia()
    {
        LineArray rif = new LineArray(132, GeometryArray.COORDINATES | GeometryArray.COLOR_3);
        float l = -50.0f;
        for(int x=0; x<44; x+=4)
        {
            rif.setCoordinate(x+0, new Point3f(-50.0f, 0.0f, l));
            rif.setCoordinate(x+1, new Point3f(50.0f, 0.0f, l));
            rif.setCoordinate(x+2, new Point3f(l, 0.0f, -50.0f));
            rif.setCoordinate(x+3, new Point3f(l, 0.0f, 50.0f));
            l+=10.0f;
        }
        l = -50.0f;
        for(int y=44; y<88; y+=4)
        {
            rif.setCoordinate(y+0, new Point3f(-50.0f, l, 0.0f));
            rif.setCoordinate(y+1, new Point3f(50.0f, l, 0.0f));
            rif.setCoordinate(y+2, new Point3f(l, -50.0f, 0.0f));
            rif.setCoordinate(y+3, new Point3f(l, 50.0f, 0.0f));
            l+=10.0f;
        }
    }
}
```

```
    }  
    l = -50.0f;  
    for(int z=88; z<132; z+=4)  
    {  
        rif.setCoordinate(z+0, new Point3f(0.0f, -50.0f, l));  
        rif.setCoordinate(z+1, new Point3f(0.0f, 50.0f, l));  
        rif.setCoordinate(z+2, new Point3f(0.0f, l, -50.0f));  
        rif.setCoordinate(z+3, new Point3f(0.0f, l, 50.0f));  
        l+=10.0f;  
    }  
    Color3f xc = new Color3f(0.1f, 0.9f, 0.1f);  
    Color3f yc = new Color3f(0.9f, 0.1f, 0.1f);  
    Color3f zc = new Color3f(0.1f, 0.1f, 0.9f);  
    for(int x=0; x<44; x++) rif.setColor(x,xc);  
    for(int y=44; y<88; y++) rif.setColor(y,yc);  
    for(int z=88; z<132; z++) rif.setColor(z,zc);  
    return new Shape3D(rif);  
}  
}
```

ESEMPIO BILLBOARD 2

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra un'animazione dipendente in maniera "indiretta" dalle azioni dell'utente: Billboard infatti modifica l'orientamento di un oggetto in modo da fargli "seguire" sempre la posizione dell'osservatore. In questo caso, il pivot della rotazione della Billboard é un punto.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioBillboard2 extends JFrame
{
    public static void main(String[] args)
    {
        EsempioBillboard2 eb2 = new EsempioBillboard2();
        eb2.setVisible(true);
    }

    public EsempioBillboard2()
    {
        this.setTitle("EsempioBillboard2; Milaneze Francesco; www.redbaron85.com");
        this.setSize(400,400);
        this.setLocation(100,100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = creaLoSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f,0.3f,2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup creaLoSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(this.insiemeBillboard());
    }
}
```

```
objRoot.addChild(this.disegnaGriglia());
return objRoot;
}

private BranchGroup insiemeBillboard()
{
    // Creo un BranchGroup per questo sotto-grafo
    BranchGroup objRootInterno = new BranchGroup();

    // Creo un primo TransformGroup al quale applico una trasformazione di traslazione
    Transform3D t3d = new Transform3D();
    t3d.setTranslation(new Vector3f(0.0f, 0.0f, -5.0f));
    TransformGroup tg1 = new TransformGroup(t3d);

    // Creo un secondo TransformGroup, impostando la capability di TRANSFORM_WRITE, e gli aggiungo
    come figlio un Text2D
    TransformGroup tg2 = new TransformGroup();
    tg2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    tg2.addChild(new Text2D("Ciao !", new Color3f(1.0f, 1.0f, 0.0f), "sansserif", 100, Font.PLAIN));

    // Creo un oggetto Billboard, associandola al secondo TransformGroup (che controlla il testo 2D) ed
    impostandole un bound d'azione
    Billboard miaBillboard = new Billboard(tg2);
    BoundingSphere bounds = new BoundingSphere();
    miaBillboard.setSchedulingBounds(bounds);

    // Con questo comando la rotazione ha come pivot un punto
    miaBillboard.setAlignmentMode(Billboard.ROTATE_ABOUT_POINT);

    // Aggiungo al primo TransformGroup (che specifica la posizione dell'oggetto) il secondo
    TransformGroup
    tg1.addChild(tg2);
    // Aggiungo il primo Transform Group alla radice di questo sotto-grafo
    objRootInterno.addChild(tg1);
    // Aggiungo la Billboard alla radice di questo sotto-grafo
    objRootInterno.addChild(miaBillboard);

    // Restituisco la radice del sotto-grafo
    return objRootInterno;
}

private Shape3D disegnaGriglia()
{
    LineArray rif = new LineArray(132, GeometryArray.COORDINATES | GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int x=0; x<44; x+=4)
    {
        rif.setCoordinate(x+0, new Point3f(-50.0f, 0.0f, l));
        rif.setCoordinate(x+1, new Point3f(50.0f, 0.0f, l));
        rif.setCoordinate(x+2, new Point3f(l, 0.0f, -50.0f));
        rif.setCoordinate(x+3, new Point3f(l, 0.0f, 50.0f));
        l+=10.0f;
    }
    l = -50.0f;
    for(int y=44; y<88; y+=4)
    {
        rif.setCoordinate(y+0, new Point3f(-50.0f, l, 0.0f));
```

```
        rif.setCoordinate(y+1, new Point3f(50.0f, 1, 0.0f));
        rif.setCoordinate(y+2, new Point3f(1, -50.0f, 0.0f));
        rif.setCoordinate(y+3, new Point3f(1, 50.0f, 0.0f));
        l+=10.0f;
    }
    l = -50.0f;
    for(int z=88; z<132; z+=4)
    {
        rif.setCoordinate(z+0, new Point3f(0.0f, -50.0f, 1));
        rif.setCoordinate(z+1, new Point3f(0.0f, 50.0f, 1));
        rif.setCoordinate(z+2, new Point3f(0.0f, 1, -50.0f));
        rif.setCoordinate(z+3, new Point3f(0.0f, 1, 50.0f));
        l+=10.0f;
    }
    Color3f xc = new Color3f(0.1f, 0.9f, 0.1f);
    Color3f yc = new Color3f(0.9f, 0.1f, 0.1f);
    Color3f zc = new Color3f(0.1f, 0.1f, 0.9f);
    for(int x=0; x<44; x++) rif.setColor(x,xc);
    for(int y=44; y<88; y++) rif.setColor(y,yc);
    for(int z=88; z<132; z++) rif.setColor(z,zc);
    return new Shape3D(rif);
}
}
```

ESEMPIO ORIENTED SHAPE 3D

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra un'animazione dipendente in maniera "indiretta" dalle azioni dell'utente: OrientedShape3D infatti modifica l'orientamento di un oggetto in modo da fargli "seguire" sempre la posizione dell'osservatore.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioOrientedShape3D extends JFrame
{
    public static void main(String[] args)
    {
        EsempioOrientedShape3D eos3d = new EsempioOrientedShape3D();
        eos3d.setVisible(true);
    }

    public EsempioOrientedShape3D()
    {
        this.setTitle("EsempioOrientedShape3D; Milanese Francesco; www.redbaron85.com");
        this.setSize(400,400);
        this.setLocation(100,100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = creaLoSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f,0.3f,2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup creaLoSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(this.insiemeOrientedShape3D());
        objRoot.addChild(this.disegnaGriglia());
    }
}
```

```
        return objRoot;
    }

    private BranchGroup insiemeOrientedShape3D()
    {
        // Creo un BranchGroup per questo sotto-grafo
        BranchGroup objRootInterno = new BranchGroup();

        // Creo un primo TransformGroup al quale applico una trasformazione di traslazione
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.0f, -5.0f));
        TransformGroup tg1 = new TransformGroup(t3d);

        // Creo un testo2D
        Text2D testo2D = new Text2D("Ciao !", new Color3f(1.0f, 1.0f, 0.0f), "sansserif", 100, Font.PLAIN);

        // Creo un oggetto OrientedShape3D, impostando come pivot della rotazione un punto
        OrientedShape3D orientedShape3D = new OrientedShape3D();
        orientedShape3D.setAlignmentMode(OrientedShape3D.ROTATE_ABOUT_POINT);

        // L'orientedShape é un vero e proprio oggetto visuale: come geometry e appearance avrà la geometry e
        // l'appearance del Testo2D precedentemente creato
        orientedShape3D.addGeometry(testo2D.getGeometry());
        orientedShape3D.setAppearance(testo2D.getAppearance());

        // Aggiungo al primo Transform Group (che definisce la posizione) l'oggetto OrientedShape3D appena
        // definito
        tg1.addChild(orientedShape3D);

        // Aggiungo il Transform Group alla radice del sotto-grafo
        objRootInterno.addChild(tg1);

        // Restituisco la radice del sotto-grafo così definito
        return objRootInterno;
    }

    private Shape3D disegnaGriglia()
    {
        LineArray rif = new LineArray(132, GeometryArray.COORDINATES | GeometryArray.COLOR_3);
        float l = -50.0f;
        for(int x=0; x<44; x+=4)
        {
            rif.setCoordinate(x+0, new Point3f(-50.0f, 0.0f, l));
            rif.setCoordinate(x+1, new Point3f(50.0f, 0.0f, l));
            rif.setCoordinate(x+2, new Point3f(l, 0.0f, -50.0f));
            rif.setCoordinate(x+3, new Point3f(l, 0.0f, 50.0f));
            l+=10.0f;
        }
        l = -50.0f;
        for(int y=44; y<88; y+=4)
        {
            rif.setCoordinate(y+0, new Point3f(-50.0f, l, 0.0f));
            rif.setCoordinate(y+1, new Point3f(50.0f, l, 0.0f));
            rif.setCoordinate(y+2, new Point3f(l, -50.0f, 0.0f));
            rif.setCoordinate(y+3, new Point3f(l, 50.0f, 0.0f));
            l+=10.0f;
        }
    }
}
```

```
    }  
    l = -50.0f;  
    for(int z=88; z<132; z+=4)  
    {  
        rif.setCoordinate(z+0, new Point3f(0.0f, -50.0f, l));  
        rif.setCoordinate(z+1, new Point3f(0.0f, 50.0f, l));  
        rif.setCoordinate(z+2, new Point3f(0.0f, l, -50.0f));  
        rif.setCoordinate(z+3, new Point3f(0.0f, l, 50.0f));  
        l+=10.0f;  
    }  
    Color3f xc = new Color3f(0.1f, 0.9f, 0.1f);  
    Color3f yc = new Color3f(0.9f, 0.1f, 0.1f);  
    Color3f zc = new Color3f(0.1f, 0.1f, 0.9f);  
    for(int x=0; x<44; x++) rif.setColor(x,xc);  
    for(int y=44; y<88; y++) rif.setColor(y,yc);  
    for(int z=88; z<132; z++) rif.setColor(z,zc);  
    return new Shape3D(rif);  
}  
}
```

ESEMPIO LOD

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioLoD extends JFrame{
    public static void main(String[] args){
        EsempioLoD eLoD = new EsempioLoD();
        eLoD.setVisible(true);
    }

    public EsempioLoD(){
        this.setTitle("EsempioLoD; Milanese Francesco; www.redbaron85.com");
        this.setSize(400,400);
        this.setLocation(100,100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = creaLoSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup creaLoSceneGraph(){
        // Creo la radice dello SceneGraph
        BranchGroup objRoot = new BranchGroup();
        // Creo un oggetto Switch, impostandone le Capabilities necessarie e popolandolo con degli oggetti,
        // che rappresenteranno il nostro oggetto a "risoluzioni" via via più basse
        Switch oggettoSwitch = new Switch();
        oggettoSwitch.setCapability(Switch.ALLOW_SWITCH_WRITE);
        oggettoSwitch.addChild(new Sphere(1f, 0, 25));
        oggettoSwitch.addChild(new Sphere(1f, 0, 15));
        oggettoSwitch.addChild(new Sphere(1f, 0, 10));
        oggettoSwitch.addChild(new Sphere(1f, 0, 4));

        // Creo un array di distanze-soglia e, a partire da questo, un oggetto DistanceLoD.
        // Associa, quindi, lo Switch creato al passo precedente al DistanceLoD, e ne imposto la
        SchedulingRegion.
        float[] arrayDiDistanze = {5.0f, 10.0f, 20.0f};
```

```
DistanceLOD oggettoLoD = new DistanceLOD(arrayDiDistanze);
oggettoLoD.addSwitch(oggettoSwitch);
oggettoLoD.setSchedulingBounds(new BoundingSphere());
// Aggiungo l'oggetto DistanceLoD e l'oggetto Switch alla scena
objRoot.addChild(oggettoLoD);
objRoot.addChild(oggettoSwitch);
// Aggiungo alla scena una "griglia" di riferimento" ( un passo: 10 unità Java3D)
objRoot.addChild(this.disegnaGriglia());
// Restituisco lo Scene Graph così definito
return objRoot;
}

private Shape3D disegnaGriglia() {
    LineArray rif = new LineArray(132, GeometryArray.COORDINATES | GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int x=0; x<44; x+=4)
    {
        rif.setCoordinate(x+0, new Point3f(-50.0f, 0.0f, l));
        rif.setCoordinate(x+1, new Point3f(50.0f, 0.0f, l));
        rif.setCoordinate(x+2, new Point3f(l, 0.0f, -50.0f));
        rif.setCoordinate(x+3, new Point3f(l, 0.0f, 50.0f));
        l+=10.0f;
    }
    l = -50.0f;
    for(int y=44; y<88; y+=4)
    {
        rif.setCoordinate(y+0, new Point3f(-50.0f, l, 0.0f));
        rif.setCoordinate(y+1, new Point3f(50.0f, l, 0.0f));
        rif.setCoordinate(y+2, new Point3f(l, -50.0f, 0.0f));
        rif.setCoordinate(y+3, new Point3f(l, 50.0f, 0.0f));
        l+=10.0f;
    }
    l = -50.0f;
    for(int z=88; z<132; z+=4)
    {
        rif.setCoordinate(z+0, new Point3f(0.0f, -50.0f, l));
        rif.setCoordinate(z+1, new Point3f(0.0f, 50.0f, l));
        rif.setCoordinate(z+2, new Point3f(0.0f, l, -50.0f));
        rif.setCoordinate(z+3, new Point3f(0.0f, l, 50.0f));
        l+=10.0f;
    }
    Color3f xc = new Color3f(0.1f, 0.9f, 0.1f);
    Color3f yc = new Color3f(0.9f, 0.1f, 0.1f);
    Color3f zc = new Color3f(0.1f, 0.1f, 0.9f);
    for(int x=0; x<44; x++) rif.setColor(x,xc);
    for(int y=44; y<88; y++) rif.setColor(y,yc);
    for(int z=88; z<132; z++) rif.setColor(z,zc);
    return new Shape3D(rif);
}
}
```

ESEMPIO COLOR INTERPOLATOR

// Milinese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come generare animazioni time-based, indipendenti dalle azioni dell'utente.
In particolare, qui il colore del materiale di un oggetto verrà modificato mediante ColorInterpolator.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioColorInterpolator extends JFrame
{
    public static void main(String[] args)
    {
        EsempioColorInterpolator eci = new EsempioColorInterpolator();
        eci.setVisible(true);
    }

    public EsempioColorInterpolator()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("EsempioColorInterpolator; Milinese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(this.conoCambiaColore());
        objRoot.addChild(this.creaRiferimento());
    }
}
```

```
objRoot.addChild(this.creaUnaLucePointLight());
return objRoot;
}

private BranchGroup conoCambiaColore()
{
    // Creo la radice di questo gruppo
    BranchGroup objRootInterno = new BranchGroup();

    // Creo un Appearance e un Material
    Appearance app = new Appearance();
    Material mtl = new Material();
    // Devo specificare quale colore di questo materiale é il colore TARGET del Position Interpolator; devo,
    inoltre, impostare le Capability
    mtl.setDiffuseColor(new Color3f(1.0f, 1.0f, 1.0f));
    mtl.setColorTarget(Material.DIFFUSE);
    mtl.setCapability(Material.ALLOW_COMPONENT_WRITE);
    // Imposto il materiale così definito come materiale dell'Appearance
    app.setMaterial(mtl);

    // Creo l'Alpha: 1 secondo per ogni fase
    Alpha mioAlpha = new Alpha(-1, Alpha.INCREASING_ENABLE | Alpha.DECREASING_ENABLE, 0,
    0, 1000, 0, 1000, 1000, 0, 1000);

    // Creo il Target: un TransformGroup, con le capability impostate
    TransformGroup tg = new TransformGroup();
    tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

    // Creo l'Interpolator, ne imposto i dati e il bound d'azione
    ColorInterpolator ci = new ColorInterpolator(mioAlpha, mtl, new Color3f(1.0f, 1.0f, 1.0f), new
    Color3f(1.0f, 1.0f, 0.0f));
    ci.setSchedulingBounds(new BoundingSphere());

    // Creo il Cono, gli associo l'Appearance con il relativo Material, e lo aggancio al TransformGroup
    Cone mioCono = new Cone(2.0f, 5.0f);
    mioCono.setAppearance(app);
    tg.addChild(mioCono);

    // Assemblo la scena
    objRootInterno.addChild(tg);
    objRootInterno.addChild(ci);

    // Restituisco il gruppo
    return objRootInterno;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(1,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(1,0.0f,50.0f));
        l += 10.0f;
    }
}
```

```
    }  
    Color3f c = new Color3f(0.1f,0.8f,0.1f);  
    for(int i=0; i<44; i++) landGeom.setColor(i,c);  
    return new Shape3D(landGeom);  
}  
  
private PointLight creaUnaLucePointLight()  
{  
    PointLight pL = new PointLight();  
    pL.setColor(new Color3f(1.0f, 1.0f, 1.0f));  
    pL.setPosition(new Point3f(0.0f, 0.0f, 10.0f));  
    pL.setInfluencingBounds(new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 200.0));  
    return pL;  
}  
}
```

ESEMPIO TRANSPARENCY INTERPOLATOR

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come generare animazioni time-based, indipendenti dalle azioni dell'utente.
In particolare, qui il valore di Trasparenza di un oggetto (TransparencyAttributes di un Appearance)
verrà modificato con TransparencyInterpolator

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioTransparencyInterpolator extends JFrame
{
    public static void main(String[] args)
    {
        EsempioTransparencyInterpolator eti = new EsempioTransparencyInterpolator();
        eti.setVisible(true);
    }

    public EsempioTransparencyInterpolator()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("EsempioTransparencyInterpolator; Milaneze Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(this.conoFantasma());
    }
}
```

```
objRoot.addChild(this.creaRiferimento());
return objRoot;
}

private BranchGroup conoFantasma()
{
    // Creo la radice di questo gruppo
    BranchGroup objRootInterno = new BranchGroup();

    // Creo un Appearance e un TransparencyAttribute (con relative Capabilities impostate)
    Appearance app = new Appearance();
    TransparencyAttributes ta = new TransparencyAttributes(TransparencyAttributes.NICEST, 0.0f);
    ta.setCapability(TransparencyAttributes.ALLOW_VALUE_WRITE);
    app.setTransparencyAttributes(ta);

    // Creo l'Alpha: 1 secondo per ogni fase
    Alpha mioAlpha = new Alpha(-1, Alpha.INCREASING_ENABLE | Alpha.DECREASING_ENABLE, 0,
0, 1000, 0, 1000, 1000, 0, 1000);

    // Creo il Target: un TransformGroup, con le capability impostate
    TransformGroup tg = new TransformGroup();
    tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

    // Creo l'Interpolator, ne imposto i dati e il bound d'azione
    TransparencyInterpolator ti = new TransparencyInterpolator(mioAlpha, ta, 0.0f, 1.0f);
    ti.setSchedulingBounds(new BoundingSphere());

    // Creo il Cono, gli associo l'Appearance e lo aggancio al TransformGroup
    Cone mioCono = new Cone(2.0f, 5.0f);
    mioCono.setAppearance(app);
    tg.addChild(mioCono);

    // Assemblo la scena
    objRootInterno.addChild(tg);
    objRootInterno.addChild(ti);

    // Restituisco il gruppo
    return objRootInterno;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
```

```
private PointLight creaUnaLucePointLight()
{
    PointLight pL = new PointLight();
    pL.setColor(new Color3f(1.0f, 1.0f, 1.0f));
    pL.setPosition(new Point3f(0.0f, 0.0f, 10.0f));
    pL.setInfluencingBounds(new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 200.0));
    return pL;
}
```

ESEMPIO SWITCH VALUE INTERPOLATOR

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioSwitch extends JFrame
{
    public static void main(String[] args)
    {
        EsempioSwitch base = new EsempioSwitch();
        base.setVisible(true);
    }

    public EsempioSwitch()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Switch Value Interpolator; Milanese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 0.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    public BranchGroup createSceneGraph() {

        BranchGroup objRoot = new BranchGroup();
        BoundingSphere bounds = new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 200.0);

        Switch objSwitch = new Switch();
        objSwitch.setCapability(Switch.ALLOW_SWITCH_WRITE);

        objSwitch.addChild(new Cylinder(2f, 4f, new Appearance()));
        objSwitch.addChild(new ColorCube(2f));
        objSwitch.addChild(new Cone(2f, 4f, new Appearance()));
    }
}
```

```
Alpha alpha = new Alpha(-1, Alpha.INCREASING_ENABLE  
+ Alpha.DECREASING_ENABLE, 0, 0, 1000, 0, 1000, 1000, 0, 1000);
```

```
SwitchValueInterpolator swiInt = new SwitchValueInterpolator(alpha,  
objSwitch);  
swiInt.setSchedulingBounds(bounds);  
swiInt.setLastChildIndex(2);
```

```
Transform3D t3d = new Transform3D();  
t3d.setTranslation(new Vector3f(0f, 0f, -10f));  
TransformGroup tg = new TransformGroup(t3d);  
tg.addChild(objSwitch);  
tg.addChild(swiInt);
```

```
objRoot.addChild(tg);
```

```
return objRoot;
```

```
}
```

```
private Shape3D creaRiferimento()
```

```
{
```

```
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
```

```
    float l = -50.0f;
```

```
    for(int c=0; c<44; c+=4)
```

```
    {
```

```
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
```

```
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
```

```
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
```

```
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
```

```
        l += 10.0f;
```

```
    }
```

```
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
```

```
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
```

```
    return new Shape3D(landGeom);
```

```
}
```

```
}
```

ESEMPIO POSITION INTERPOLATOR 1

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come generare animazioni time-based, indipendenti dalle azioni dell'utente.
In particolare, qui un oggetto viene traslato lungo il suo asse X locale (quello di default del PositionInterpolator),
andando avanti e indietro nell'arco di 4 secondi.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioPositionInterpolator1 extends JFrame
{
    public static void main(String[] args)
    {
        EsempioPositionInterpolator1 epi1 = new EsempioPositionInterpolator1();
        epi1.setVisible(true);
    }

    public EsempioPositionInterpolator1()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("EsempioPositionInterpolator1; Milaneze Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(this.cuboInMovimento());
    }
}
```

```
objRoot.addChild(this.creaRiferimento());
return objRoot;
}

private BranchGroup cuboInMovimento()
{
    // Creo la radice di questo gruppo
    BranchGroup objRootInterno = new BranchGroup();

    // Creo l'Alpha: 1 secondo per ogni fase
    Alpha mioAlpha = new Alpha(-1, Alpha.INCREASING_ENABLE | Alpha.DECREASING_ENABLE, 0,
0, 1000, 0, 1000, 1000, 0, 1000);

    // Creo il Target: un TransformGroup, con le capability impostate
    TransformGroup tg = new TransformGroup();
    tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

    // Creo l'Interpolatore, ne imposto i dati e il bound d'azione
    PositionInterpolator pi = new PositionInterpolator(mioAlpha, tg);
    pi.setStartPosition(0.0f);
    pi.setEndPosition(10.0f);
    pi.setSchedulingBounds(new BoundingSphere());

    // Creo il ColorCube e lo aggancio al TransformGroup
    ColorCube mioCuboColorato = new ColorCube(1.0f);
    tg.addChild(mioCuboColorato);

    // Assemblo la scena
    objRootInterno.addChild(tg);
    objRootInterno.addChild(pi);

    // Restituisco il gruppo
    return objRootInterno;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ESEMPIO POSITION INTERPOLATOR 2

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come generare animazioni time-based, indipendenti dalle azioni dell'utente. In particolare, qui abbiamo la traslazione di DUE oggetti posti in posizioni iniziali diversi e con orientamenti iniziali differenti.

PositionInterpolator trasla infatti lungo l'asse X locale di un oggetto, mentre qui un cubo trasla lungo l'asse Y globale: tale cubo é stato ruotato inizialmente di 90° intorno all'asse Z (locale e globale all'inizio coincidono) per permettere tale trasformazione.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioPositionInterpolator2 extends JFrame
{
    public static void main(String[] args)
    {
        EsempioPositionInterpolator2 epi2 = new EsempioPositionInterpolator2();
        epi2.setVisible(true);
    }

    public EsempioPositionInterpolator2()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("EsempioPositionInterpolator2; Milanese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }
}
```

```
private BranchGroup createSceneGraph()
{
    BranchGroup objRoot = new BranchGroup();
    objRoot.addChild(this.cuboInMovimento1());
    objRoot.addChild(this.cuboInMovimento2());
    objRoot.addChild(this.creaRiferimento());
    return objRoot;
}

private BranchGroup cuboInMovimento1()
{
    // Creo la radice di questo gruppo
    BranchGroup objRootInterno = new BranchGroup();

    // Creo l'Alpha: 1 secondo per ogni fase
    Alpha mioAlpha = new Alpha(-1, Alpha.INCREASING_ENABLE | Alpha.DECREASING_ENABLE, 0,
0, 1000, 0, 1000, 1000, 0, 1000);

    // Creo il Target: un TransformGroup, con le capability impostate
    TransformGroup tg = new TransformGroup();
    tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

    // Creo l'Interpolatore, ne imposto i dati e il bound d'azione
    PositionInterpolator pi = new PositionInterpolator(mioAlpha, tg);
    pi.setStartPosition(0.0f);
    pi.setEndPosition(10.0f);
    pi.setSchedulingBounds(new BoundingSphere());

    // Creo il ColorCube e lo aggancio al TransformGroup
    ColorCube mioCuboColorato = new ColorCube(1.0f);
    tg.addChild(mioCuboColorato);

    // Assemblo la scena
    objRootInterno.addChild(tg);
    objRootInterno.addChild(pi);

    // Restituisco il gruppo
    return objRootInterno;
}

private BranchGroup cuboInMovimento2()
{
    // Creo la radice di questo gruppo
    BranchGroup objRootInterno = new BranchGroup();

    // Creo un TG che serve a traslare in alto il secondo cubo
    Transform3D t3d1 = new Transform3D();
    t3d1.setTranslation(new Vector3f(0.0f, 10.0f, 0.0f));
    TransformGroup tg1 = new TransformGroup(t3d1);

    // Creo l'Alpha: 1 secondo per ogni fase
    Alpha mioAlpha = new Alpha(-1, Alpha.INCREASING_ENABLE | Alpha.DECREASING_ENABLE, 0,
0, 1000, 0, 1000, 1000, 0, 1000);

    // Creo il Target: un TransformGroup, con le capability impostate
    TransformGroup tg2 = new TransformGroup();
```

```
tg2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

// Creo l'Interpolatore, ne imposto i dati, il bound d'azione e il nuovo ASSE DI TRASFORMAZIONE
Transform3D asseTrasformazione = new Transform3D();
asseTrasformazione.rotZ(Math.PI/2);
PositionInterpolator pi = new PositionInterpolator(mioAlpha, tg2, asseTrasformazione, 0.0f, 10.0f);
pi.setSchedulingBounds(new BoundingSphere());

// Creo il ColorCube e lo aggancio al TransformGroup
ColorCube mioCuboColorato = new ColorCube(1.0f);
tg2.addChild(mioCuboColorato);

// Assemblo la scena: in cima c'è la traslazione in alto, poi l'interpolatore e il tg di traslazione animata
tg1.addChild(tg2);
tg1.addChild(pi);
objRootInterno.addChild(tg1);

// Restituisco il gruppo
return objRootInterno;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ESEMPIO POSITION INTERPOLATOR 3

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come generare animazioni time-based, indipendenti dalle azioni dell'utente.

In particolare, qui un oggetto trasla lungo l'asse Y globale.

Il PositionInterpolator ha come asse di traslazione l'asse X locale di un oggetto, dunque per ottenere la traslazione lungo l'asse Y globale il cubo viene prima ruotato di 90° intorno all'asse Z (locale e globale all'inizio coincidono).

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioPositionInterpolator3 extends JFrame
{
    public static void main(String[] args)
    {
        EsempioPositionInterpolator3 epi3 = new EsempioPositionInterpolator3();
        epi3.setVisible(true);
    }

    public EsempioPositionInterpolator3()
    {
        // Impostazioni "classiche" di un JFrame...
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("EsempioPositionInterpolator3; Milanese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
```

```
{
    // Creo la radice del Content Branch Graph
    BranchGroup objRoot = new BranchGroup();

    // Aggiungo al Content Branch Graph un altro BranchGroup
    objRoot.addChild(this.cuboInMovimento());

    // Creo un pavimento di riferimento e lo aggiungo alla scena
    objRoot.addChild(this.creaRiferimento());

    // Restituisco il tutto
    return objRoot;
}

private BranchGroup cuboInMovimento()
{
    // Creo la radice di questo gruppo
    BranchGroup objRootInterno = new BranchGroup();

    // Ruoto leggermente il cubo intorno al suo asse Z: per farlo, creo un TransformGroup con una
trasformazione di Rotazione...
    Transform3D t3d1 = new Transform3D();
    t3d1.rotZ(Math.PI/2);
    TransformGroup tg1 = new TransformGroup(t3d1);

    // Creo l'Alpha: 1 secondo per ogni fase
    Alpha mioAlpha = new Alpha(-1, Alpha.INCREASING_ENABLE | Alpha.DECREASING_ENABLE, 0,
0, 1000, 0, 1000, 1000, 0, 1000);

    // Creo il Target: un TransformGroup, con le capability impostate
    TransformGroup tg2 = new TransformGroup();
    tg2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

    // Creo l'Interpolatore, ne imposto i dati e il bound d'azione
    PositionInterpolator pi = new PositionInterpolator(mioAlpha, tg2);
    pi.setStartPosition(0.0f);
    pi.setEndPosition(10.0f);
    pi.setSchedulingBounds(new BoundingSphere());

    // Creo il ColorCube e lo aggancio al TransformGroup
    ColorCube mioCuboColorato = new ColorCube(1.0f);
    tg2.addChild(mioCuboColorato);

    // Assemblo la scena
    tg1.addChild(tg2);
    tg1.addChild(pi);
    objRootInterno.addChild(tg1);

    // Restituisco il gruppo
    return objRootInterno;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
```

```
{
    landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
    landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
    landGeom.setCoordinate(c+2, new Point3f(1,0.0f,-50.0f));
    landGeom.setCoordinate(c+3, new Point3f(1,0.0f,50.0f));
    l += 10.0f;
}
Color3f c = new Color3f(0.1f,0.8f,0.1f);
for(int i=0; i<44; i++) landGeom.setColor(i,c);
return new Shape3D(landGeom);
}
```

ESEMPIO ROTATION INTERPOLATOR

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come generare animazioni time-based, indipendenti dalle azioni dell'utente. In particolare, qui un oggetto ruota intorno al suo asse Y locale mediante un RotationInterpolator, associato ad un Alpha con loop infinito, solo increasing enable e durata del ciclo 4 secondi.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
```

```
public class EsempioRotationInterpolator extends JFrame{
    public static void main(String[] args){
        EsempioRotationInterpolator eri = new EsempioRotationInterpolator();
        eri.setVisible(true);
    }

    public EsempioRotationInterpolator(){
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("EsempioRotationInterpolator; Milanese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph(){
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(this.rotazioneCubo());
        objRoot.addChild(this.creaRiferimento());
        return objRoot;
    }
}
```

```
private BranchGroup rotazioneCubo() {
    // Creo la radice di questo gruppo
    BranchGroup bgRotazione = new BranchGroup();

    // Sposto leggermente in alto il cubo
    Transform3D t3d1 = new Transform3D();
    t3d1.rotZ(Math.PI/2);
    t3d1.setTranslation(new Vector3f(0.0f, 1.0f, 0.0f));
    TransformGroup tg1 = new TransformGroup(t3d1);

    // Creo l'Alpha: la rotazione durerà in totale 4 secondi, solo increasing enable
    Alpha mioAlpha = new Alpha(-1, Alpha.INCREASING_ENABLE, 0, 0, 4000, 0, 0, 0, 0, 0);

    // Creo il Target: un TransformGroup, con le capability impostate
    TransformGroup tg2 = new TransformGroup();
    tg2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

    // Creo l'Interpolatore, ne imposto i dati e il bound d'azione. La rotazione avviene, di default, intorno
    all'asse X locale
    // Adesso creo una trasformazione per impostare, come asse di rotazione, l'asse Z globale
    Transform3D tRot = new Transform3D();
    tRot.rotZ(Math.PI/2);
    // Creo dunque l'interpolatore...
    RotationInterpolator ri = new RotationInterpolator(mioAlpha, tg2, tRot, 0.0f, (float)(Math.PI*2));
    ri.setSchedulingBounds(new BoundingSphere());

    // Creo il cubo e lo aggancio al TG dell'intrpolazione
    ColorCube cuboColorato = new ColorCube(1.0f);
    tg2.addChild(cuboColorato);

    // Assemblo la scena
    tg1.addChild(tg2);
    tg1.addChild(ri);
    bgRotazione.addChild(tg1);

    // Restituisco il gruppo
    return bgRotazione;
}

private Shape3D creaRiferimento() {
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(1,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(1,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ESEMPIO SCALE INTERPOLATOR

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come generare animazioni time-based, indipendenti dalle azioni dell'utente.
In particolare, qui un oggetto viene scalato con uno ScaleInterpolator associato ad un'alpha
che ha durata 4 secondi, solo increasing enable e loop infinito.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
```

```
public class EsempioScaleInterpolator extends JFrame
{
    public static void main(String[] args)
    {
        EsempioScaleInterpolator esi = new EsempioScaleInterpolator();
        esi.setVisible(true);
    }

    public EsempioScaleInterpolator()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("EsempioScaleInterpolator; Milaneze Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(this.scalingCubo());
    }
}
```

```
objRoot.addChild(this.creaRiferimento());
return objRoot;
}

private BranchGroup scalingCubo()
{
    // Creo la radice di questo gruppo
    BranchGroup bgScaling = new BranchGroup();

    // Sposto leggermente in alto il cubo
    Transform3D t3d1 = new Transform3D();
    t3d1.rotZ(Math.PI/2);
    t3d1.setTranslation(new Vector3f(0.0f, 1.0f, 0.0f));
    TransformGroup tg1 = new TransformGroup(t3d1);

    // Creo l'Alpha: lo scaling durerà in totale 4 secondi, solo increasing enable
    Alpha mioAlpha = new Alpha(-1, Alpha.INCREASING_ENABLE, 0, 0, 4000, 0, 0, 0, 0);

    // Creo il Target: un TransformGroup, con le capability impostate
    TransformGroup tg2 = new TransformGroup();
    tg2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

    // Creo l'Interpolatore, ne imposto i dati e il bound d'azione.
    ScaleInterpolator si = new ScaleInterpolator(mioAlpha, tg2);
    si.setSchedulingBounds(new BoundingSphere());

    // Creo il cubo e lo aggancio al TG dell'intrpolazione
    ColorCube cuboColorato = new ColorCube(1.0f);
    tg2.addChild(cuboColorato);

    // Assemblo la scena
    tg1.addChild(tg2);
    tg1.addChild(si);
    bgScaling.addChild(tg1);

    // Restituisco il gruppo
    return bgScaling;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(1,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(1,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ESEMPIO QUATERNIONI

// Milinese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come "configurare" ed utilizzare i quaternioni per specificare degli orientamenti nello spazio3D, facendo uso di oggetti visuali statici per mostrare gli effetti.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioQuaternioni extends JFrame
{
    public static void main(String[] args)
    {
        EsempioQuaternioni eq = new EsempioQuaternioni();
        eq.setVisible(true);
    }

    public EsempioQuaternioni()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Quaternioni; Milinese Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(this.creaRiferimento());
    }
}
```

```
altri)

// Creo un cubo, ruotandolo di 45 gradi intorno all'asse X (valore del vettore di rotazione: 1 per x, 0 per gli
objRoot.addChild(cuboRuotato(-10f, 0f, 0f, 1.0, 0.0, 0.0, Math.PI/4));

// Creo un cubo, ruotandolo di 45 gradi intorno all'asse Y
objRoot.addChild(cuboRuotato(-5f, 0f, 0f, 0.0, 1.0, 0.0, Math.PI/4));

// Al centro in alto metto un cubo colorato non ruotato, come riferimento
objRoot.addChild(cuboRuotato(0f, 5f, 0f, 0.0, 0.0, 0.0, 0.0));

// Creo un cubo, ruotandolo di 45 gradi intorno all'asse Z
objRoot.addChild(cuboRuotato(0f, 0f, 0f, 0.0, 0.0, 1.0, Math.PI/4));

// Creo un cubo, ruotandolo di 45 gradi intorno all'asse X e di 45 gradi intorno all'asse Y
objRoot.addChild(cuboRuotatoConMul(5f, 0f, 0f, 1.0, 1.0, 0.0, Math.PI/4));

// Creo un cubo, ruotandolo di 45 gradi intorno a X, 45 gradi intorno a Y e 45 gradi intorno a Z
objRoot.addChild(cuboRuotatoConMul(10f, 0f, 0f, 1.0, 1.0, 1.0, Math.PI/4));

// Restituisco il tutto
return objRoot;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(1,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(1,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}

private TransformGroup cuboRuotato(float posX, float posY, float posZ, double aax, double aay, double aaz,
double angolo)
{
    // Primo TransformGroup: serve a posizionare l'oggetto
    Transform3D t3d1 = new Transform3D();
    t3d1.setTranslation(new Vector3f(posX, posY, posZ));
    TransformGroup tg1 = new TransformGroup(t3d1);

    // Secondo TransformGroup, quello con la rotazione
    Quat4f quat4f = new Quat4f();
    quat4f.set(new AxisAngle4d(aax, aay, aaz, angolo));
    Transform3D t3d2 = new Transform3D();
    t3d2.setRotation(quat4f);
    TransformGroup tg2 = new TransformGroup(t3d2);

    // Aggiungo un cubo colorato al secondo TransformGroup
    tg2.addChild(new ColorCube(1.0f));
}
```

```
// Aggancio il secondo TransformGroup al primo
tg1.addChild(tg2);

// Restituisco il tutto
return tg1;
}

private TransformGroup cuboRuotatoConMul(float posX, float posY, float posZ, double aax, double aay, double
aaz, double angolo)
{
    // Primo TransformGroup: serve a posizionare l'oggetto
    Transform3D t3d1 = new Transform3D();
    t3d1.setTranslation(new Vector3f(posX, posY, posZ));
    TransformGroup tg1 = new TransformGroup(t3d1);

    // Secondo TransformGroup, quello con la rotazione: prima creo i quaternioni per le trasformazioni
    // lungo i vari assi, poi li combino tra di loro e infine assegno il quaternion finale al Transform3D
    Quat4f quat4f1 = new Quat4f();
    quat4f1.set(new AxisAngle4d(aax, 0, 0, angolo));
    Quat4f quat4f2 = new Quat4f();
    quat4f2.set(new AxisAngle4d(0, aay, 0, angolo));
    Quat4f quat4f3 = new Quat4f();
    quat4f3.set(new AxisAngle4d(0, 0, aaz, angolo));
    quat4f1.mul(quat4f2);
    quat4f1.mul(quat4f3);
    Transform3D t3d2 = new Transform3D();
    t3d2.setRotation(quat4f1);
    TransformGroup tg2 = new TransformGroup(t3d2);

    // Aggiungo un cubo colorato al secondo TransformGroup
    tg2.addChild(new ColorCube(1.0f));

    // Aggancio il secondo TransformGroup al primo
    tg1.addChild(tg2);

    // Restituisco il tutto
    return tg1;
}
}
```

ESEMPIO POSITION PATH INTERPOLATOR

// Milanesi Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come generare animazioni time-based, indipendenti dalle azioni dell'utente.
In particolare, qui un oggetto segue un "path" di posizioni ben definite nel corso di 10 secondi, controllato da un PositionPathInterpolator.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioPositionPathInterpolator extends JFrame
{
    public static void main(String[] args)
    {
        EsempioPositionPathInterpolator eppi = new EsempioPositionPathInterpolator();
        eppi.setVisible(true);
    }

    public EsempioPositionPathInterpolator()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("EsempioPositionPathInterpolator; Milanesi Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 0.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(this.cuboConPositionPath());
    }
}
```

```
objRoot.addChild(this.creaRiferimento());
return objRoot;
}

private BranchGroup cuboConPositionPath()
{
    // Creo la radice di questo gruppo
    BranchGroup bgPositionPathInterpolator = new BranchGroup();

    // Sposto leggermente in alto il cubo
    Transform3D t3d1 = new Transform3D();
    t3d1.rotZ(Math.PI/2);
    t3d1.setTranslation(new Vector3f(0.0f, 1.0f, 0.0f));
    TransformGroup tg1 = new TransformGroup(t3d1);

    // Creo l'Alpha: la trasformazione durerà in totale 10 secondi, solo increasing enable
    Alpha mioAlpha = new Alpha(-1, Alpha.INCREASING_ENABLE, 0, 0, 10000, 0, 0, 0, 0, 0);

    // Creo il Target: un TransformGroup, con le capability impostate
    TransformGroup tg2 = new TransformGroup();
    tg2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

    // Preparo un array di nodi per il RotationPathInterpolator
    float[] arrayNodi = {0.0f, 0.2f, 0.4f, 0.6f, 0.8f, 1.0f};

    // Preparo un array di posizioni
    Point3f[] arrayPosizioni = {
        new Point3f(0.0f, 0.0f, 0.0f),
        new Point3f(0.0f, 0.0f, -5.0f),
        new Point3f(5.0f, 5.0f, -10.0f),
        new Point3f(10.0f, 5.0f, -10.0f),
        new Point3f(5.0f, 0.0f, -5.0f),
        new Point3f(0.0f, 0.0f, 0.0f)
    };

    // Creo l'Interpolatore
    PositionPathInterpolator ppi = new PositionPathInterpolator(mioAlpha, tg2, new Transform3D(),
arrayNodi, arrayPosizioni);
    ppi.setSchedulingBounds(new BoundingSphere());

    // Creo il cubo e lo aggancio al TG dell'interpolazione
    ColorCube cuboColorato = new ColorCube(1.0f);
    tg2.addChild(cuboColorato);

    // Assemblo la scena
    tg1.addChild(tg2);
    tg1.addChild(ppi);
    bgPositionPathInterpolator.addChild(tg1);

    // Restituisco il gruppo
    return bgPositionPathInterpolator;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
```

```
for(int c=0; c<44; c+=4)
{
    landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,1));
    landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,1));
    landGeom.setCoordinate(c+2, new Point3f(1,0.0f,-50.0f));
    landGeom.setCoordinate(c+3, new Point3f(1,0.0f,50.0f));
    l += 10.0f;
}
Color3f c = new Color3f(0.1f,0.8f,0.1f);
for(int i=0; i<44; i++) landGeom.setColor(i,c);
return new Shape3D(landGeom);
}
```

ESEMPIO ROTPOSPATHINTERPOLATOR

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come generare animazioni time-based, indipendenti dalle azioni dell'utente.
In particolare, qui un oggetto segue un "path" di posizioni ben definite nel corso di 10 secondi, ruotando su se stesso, controllato da un RotPosPathInterpolator.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioRotPosPathInterpolator extends JFrame
{
    public static void main(String[] args)
    {
        EsempioRotPosPathInterpolator erppi = new EsempioRotPosPathInterpolator();
        erppi.setVisible(true);
    }

    public EsempioRotPosPathInterpolator()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("EsempioRotPosPathInterpolator; Milaneze Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(this.cuboConRotPosPath());
    }
}
```

```
objRoot.addChild(this.creaRiferimento());
return objRoot;
}

private BranchGroup cuboConRotPosPath()
{
    // Creo la radice di questo gruppo
    BranchGroup bgPositionPathInterpolator = new BranchGroup();

    // Sposto leggermente in alto il cubo
    Transform3D t3d1 = new Transform3D();
    t3d1.rotZ(Math.PI/2);
    t3d1.setTranslation(new Vector3f(0.0f, 1.0f, 0.0f));
    TransformGroup tg1 = new TransformGroup(t3d1);

    // Creo l'Alpha: la trasformazione durerà in totale 10 secondi, solo increasing enable
    Alpha mioAlpha = new Alpha(-1, Alpha.INCREASING_ENABLE, 0, 0, 10000, 0, 0, 0, 0, 0);

    // Creo il Target: un TransformGroup, con le capability impostate
    TransformGroup tg2 = new TransformGroup();
    tg2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

    // Preparo un array di nodi per il RotPosPathInterpolator
    float[] arrayNodi = {0.0f, 0.1f, 0.2f, 0.3f, 0.4f, 0.5f, 0.6f, 0.7f, 0.8f, 0.9f, 1.0f};

    // Preparo un array di posizioni
    Point3f[] arrayPosizioni = {
        new Point3f(0.0f, -10.0f, -10.0f),
        new Point3f(0.0f, -8.0f, -8.0f),
        new Point3f(0.0f, -6.0f, -6.0f),
        new Point3f(0.0f, -4.0f, -4.0f),
        new Point3f(0.0f, -2.0f, -2.0f),
        new Point3f(0.0f, 0.0f, 0.0f),
        new Point3f(0.0f, 2.0f, 2.0f),
        new Point3f(0.0f, 4.0f, 4.0f),
        new Point3f(0.0f, 6.0f, 6.0f),
        new Point3f(0.0f, 8.0f, 8.0f),
        new Point3f(0.0f, 10.0f, 10.0f)
    };

    // Preparo un array di quaternioni (orientamenti)
    Quat4f[] arrayQuaternioni = new Quat4f[11];
    arrayQuaternioni[0]=new Quat4f();
    arrayQuaternioni[0].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI/5));
    arrayQuaternioni[1]=new Quat4f();
    arrayQuaternioni[1].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI/5*2));
    arrayQuaternioni[2]=new Quat4f();
    arrayQuaternioni[2].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI/5*3));
    arrayQuaternioni[3]=new Quat4f();
    arrayQuaternioni[3].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI/5*4));
    arrayQuaternioni[4]=new Quat4f();
    arrayQuaternioni[4].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI));
    arrayQuaternioni[5]=new Quat4f();
    arrayQuaternioni[5].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI/5*6));
    arrayQuaternioni[6]=new Quat4f();
    arrayQuaternioni[6].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI/5*7));
    arrayQuaternioni[7]=new Quat4f();
```

```
arrayQuaternioni[7].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI/5*8));
arrayQuaternioni[8]=new Quat4f();
arrayQuaternioni[8].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI/5*9));
arrayQuaternioni[9]=new Quat4f();
arrayQuaternioni[9].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI*2));
arrayQuaternioni[10]=new Quat4f();
arrayQuaternioni[10].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI/5));

// Creo l'Interpolatore
RotPosPathInterpolator rppi = new RotPosPathInterpolator(mioAlpha, tg2, new Transform3D(),
arrayNodi, arrayQuaternioni, arrayPosizioni);
rppi.setSchedulingBounds(new BoundingSphere());

// Creo il cubo e lo aggancio al TG dell'interpolazione
ColorCube cuboColorato = new ColorCube(1.0f);
tg2.addChild(cuboColorato);

// Assemblo la scena
tg1.addChild(tg2);
tg1.addChild(rppi);
bgPositionPathInterpolator.addChild(tg1);

// Restituisco il gruppo
return bgPositionPathInterpolator;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(1,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(1,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ESEMPIO ROTPOSSCALEPATHINTERPOLATOR

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Questo esempio mostra come generare animazioni time-based, indipendenti dalle azioni dell'utente.
In particolare, qui un oggetto segue un "path" di posizioni ben definite nel corso di 10 secondi, ruotando su se stesso e ridimensionandosi, controllato da un RotPosScalePathInterpolator.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioRotPosScalePathInterpolator extends JFrame
{
    public static void main(String[] args)
    {
        EsempioRotPosScalePathInterpolator erpspi = new EsempioRotPosScalePathInterpolator();
        erpspi.setVisible(true);
    }

    public EsempioRotPosScalePathInterpolator()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("EsempioRotPosScalePathInterpolator; Milaneze Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(this.cuboConRotPosPath());
    }
}
```

```
objRoot.addChild(this.creaRiferimento());
return objRoot;
}

private BranchGroup cuboConRotPosPath()
{
    // Creo la radice di questo gruppo
    BranchGroup bgPositionPathInterpolator = new BranchGroup();

    // Sposto leggermente in alto il cubo
    Transform3D t3d1 = new Transform3D();
    t3d1.rotZ(Math.PI/2);
    t3d1.setTranslation(new Vector3f(0.0f, 1.0f, 0.0f));
    TransformGroup tg1 = new TransformGroup(t3d1);

    // Creo l'Alpha: la trasformazione durerà in totale 10 secondi, solo increasing enable
    Alpha mioAlpha = new Alpha(-1, Alpha.INCREASING_ENABLE, 0, 0, 10000, 0, 0, 0, 0, 0);

    // Creo il Target: un TransformGroup, con le capability impostate
    TransformGroup tg2 = new TransformGroup();
    tg2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

    // Preparo un array di nodi per il RotPosScalePathInterpolator
    float[] arrayNodi = {0.0f, 0.1f, 0.2f, 0.3f, 0.4f, 0.5f, 0.6f, 0.7f, 0.8f, 0.9f, 1.0f};

    // Preparo un array di posizioni
    Point3f[] arrayPosizioni = {
        new Point3f(0.0f, -10.0f, -10.0f),
        new Point3f(0.0f, -8.0f, -8.0f),
        new Point3f(0.0f, -6.0f, -6.0f),
        new Point3f(0.0f, -4.0f, -4.0f),
        new Point3f(0.0f, -2.0f, -2.0f),
        new Point3f(0.0f, 0.0f, 0.0f),
        new Point3f(0.0f, 2.0f, 2.0f),
        new Point3f(0.0f, 4.0f, 4.0f),
        new Point3f(0.0f, 6.0f, 6.0f),
        new Point3f(0.0f, 8.0f, 8.0f),
        new Point3f(0.0f, 10.0f, 10.0f)
    };

    // Preparo un array di quaternioni (orientamenti)
    Quat4f[] arrayQuaternioni = new Quat4f[11];
    arrayQuaternioni[0]=new Quat4f();
    arrayQuaternioni[0].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI/5));
    arrayQuaternioni[1]=new Quat4f();
    arrayQuaternioni[1].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI/5*2));
    arrayQuaternioni[2]=new Quat4f();
    arrayQuaternioni[2].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI/5*3));
    arrayQuaternioni[3]=new Quat4f();
    arrayQuaternioni[3].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI/5*4));
    arrayQuaternioni[4]=new Quat4f();
    arrayQuaternioni[4].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI));
    arrayQuaternioni[5]=new Quat4f();
    arrayQuaternioni[5].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI/5*6));
    arrayQuaternioni[6]=new Quat4f();
    arrayQuaternioni[6].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI/5*7));
    arrayQuaternioni[7]=new Quat4f();
```

```
arrayQuaternioni[7].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI/5*8));
arrayQuaternioni[8]=new Quat4f();
arrayQuaternioni[8].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI/5*9));
arrayQuaternioni[9]=new Quat4f();
arrayQuaternioni[9].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI*2));
arrayQuaternioni[10]=new Quat4f();
arrayQuaternioni[10].set(new AxisAngle4d(1.0, 0.0, 0.0, Math.PI/5));

// Preparo un array di dimensioni
float[] arrayDimensioni= {5f, 4f, 3f, 2f, 1f, 0f, 1f, 2f, 3f, 4f, 5f};

// Creo l'Interpolatore
RotPosScalePathInterpolator rppi = new RotPosScalePathInterpolator(mioAlpha, tg2, new Transform3D(),
arrayNodi, arrayQuaternioni, arrayPosizioni, arrayDimensioni);
rppi.setSchedulingBounds(new BoundingSphere());

// Creo il cubo e lo aggancio al TG dell'interpolazione
ColorCube cuboColorato = new ColorCube(1.0f);
tg2.addChild(cuboColorato);

// Assemblo la scena
tg1.addChild(tg2);
tg1.addChild(rppi);
bgPositionPathInterpolator.addChild(tg1);

// Restituisco il gruppo
return bgPositionPathInterpolator;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

ESEMPIO ICBM --- MAIN

// Milinese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

/*

Esempio che mostra come inserire "a comando" un'animazione basata su Alpha e RotPosPathInterpolator, recuperando i dati in arrivo da classi esterne (un pannello GUI) e creando un Interpolator e un BranchGroup ad hoc per generare l'animazione. Il BranchGroup andrà quindi aggiunto al Content Branch Graph.

Viene effettuata una chiamata di Thread.sleep particolare per far partire l'animazione quando Alpha sta iniziando un ciclo, in modo da non mostrare l'animazione a video quando Alpha é, ad esempio, a metà ciclo o al termine dello stesso.

Nell'esempio, il Branch Group NON viene sganciato dalla scena al termine dell'esecuzione, ma viene lasciato con un loop infinito per poter osservare la scena.

In questo esempio si fa uso del KeyNavigatorBehavior di Java3D per navigare nello Spazio Virtuale.

Ricordo che i comandi per navigare nello Spazio Virtuale con tale Behavior sono:

ROTAZIONI

FracciaDestra : ruota a destra
FrecciaSinistra : ruota a sinistra
PagSu : ruota verso il basso
PagGiù : ruota verso l'alto

TRASLAZIONI

ALT + FrecciaSu : trasla in avanti
ALT + FrecciaGiù : trasla indietro
ALT + FrecciaDestra : trasla a destra
ALT + FrecciaSinistra : trasla a sinistra
ALT + PagSu : trasla verso il basso
ALT + PagGiù : trasla verso l'alto

*/

// Import necessari per gestire JFrame, JPanel, il menù, il SimpleUniverse e j3d.geometry per gli oggetti di base.

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import java.util.*;
import java.awt.event.*;
import com.sun.j3d.loaders.*;
import com.sun.j3d.loaders.objectfile.*;
```

```
public class EsempioICBM extends JFrame
{
```

```
    private PannelloPlancia pp;                // Pannello per la plancia (GUI)
    private BranchGroup objRoot;                // Attenzione, dichiaro qui la radice del Content Branch Graph
    private double tempoDiAvvioDelSistema;      // Recupero il momento di avvio del gioco (tornerà utile in seguito)
```

```
    public static void main(String[] args)
    {
```

```
        EsempioICBM eICBM = new EsempioICBM();
        eICBM.setVisible(true);
```

```
}
```

```
public EsempioICBM()  
{
```

```
    // Impostazioni "classiche" di un JFrame...  
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    this.setLocation(100, 100);  
    this.setSize(400, 400);  
    this.setTitle("Esempio ICBM; Milanese Francesco; www.redbaron85.com");  
    // Creo un pannello per la plancia e lo aggancio al Frame  
    pp = new PannelloPlancia(this);  
    this.getContentPane().add(pp, BorderLayout.SOUTH);  
    // Recupero il momento di avvio del programma  
    tempoDiAvvioDelSistema = System.currentTimeMillis();  
    // Recupero le impostazioni grafiche del sistema  
    GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();  
    // Creo un oggetto Canvas3D, passandogli al costruttore le impostazioni grafiche del sistema  
    Canvas3D canvas3D = new Canvas3D(config);  
    // Aggiungo la Canvas3D al JFrame  
    this.getContentPane().add(canvas3D, BorderLayout.CENTER);  
    // Creo il Content Branch Graph  
    BranchGroup scene = createSceneGraph();  
    // Creo un SimpleUniverse, passandogli come parametro la Canvas3D creata precedentemente  
    SimpleUniverse simpleU = new SimpleUniverse(canvas3D);  
    // Imposto la BackClipDistance  
    simpleU.getViewer().getView().setBackClipDistance(10000);  
    // Imposto la FrontClipDistance  
    simpleU.getViewer().getView().setFrontClipDistance(0.1);  
    // Recupero il TransformGroup che controlla la ViewPlatformTransform, traslo il punto di osservazione  
    // iniziale leggermente indietro e associo, a tale TG, un KeyNavigatorBehavior, per "navigare" nello spazio virtuale con la  
    // tastiera  
    TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();  
    Transform3D t3d = new Transform3D();  
    t3d.setTranslation(new Vector3f(0.0f, 0.3f, 2.0f));  
    vpTG.setTransform(t3d);  
    KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);  
    keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));  
    scene.addChild(keyNavBeh);  
    // Compilo il ContentBranch Graph  
    scene.compile();  
    // Aggiungo al SimpleUniverse il ContentBranch Graph  
    simpleU.addBranchGraph(scene);  
}
```

```
private BranchGroup createSceneGraph()  
{
```

```
    // objRoot l'ho dichiarato fuori...  
    objRoot = new BranchGroup();  
    objRoot.setCapability(Group.ALLOW_CHILDREN_EXTEND);  
  
    // Carico la plancia  
    objRoot.addChild(this.caricaPlancia());  
  
    // Creo un piano di riferimento  
    objRoot.addChild(this.creaRiferimento());
```

```
// Aggiungo una luce
objRoot.addChild(this.creaPointLight());

return objRoot;
}

private Shape3D creaRiferimento()
{
    // Creo una nuova Shape3D.
    Shape3D miaShape3D = new Shape3D();
    // Creo una Geometry, di tipo QuadArray, impostando coordinate e colori dei vertici componenti. Imposto
    // questo QuadArray come geometria della Shape3D.
    QuadArray terra = new QuadArray(4, GeometryArray.COORDINATES |
    GeometryArray.COLOR_3);
    terra.setCoordinate(0, new Point3d(-50.0, 0.0, -50));
    terra.setCoordinate(1, new Point3d(50.0, 0.0, -50));
    terra.setCoordinate(2, new Point3d(50.0, 0.0, 50));
    terra.setCoordinate(3, new Point3d(-50.0, 0.0, 50));
    terra.setColor(0, new Color3f(1.0f, 1.0f, 0.0f));
    terra.setColor(1, new Color3f(1.0f, 1.0f, 0.0f));
    terra.setColor(2, new Color3f(1.0f, 1.0f, 0.0f));
    terra.setColor(3, new Color3f(1.0f, 1.0f, 0.0f));
    miaShape3D.setGeometry(terra);
    // Creo una Appearance per questa Shape3D, impostando la resa a facce piene e cull_none. Imposto questa
    // Appearance come Appearance della Shape3D.
    Appearance miaAppearance = new Appearance();
    miaAppearance.setPolygonAttributes(new
    PolygonAttributes(PolygonAttributes.POLYGON_FILL, PolygonAttributes.CULL_NONE, 0.0f));
    miaShape3D.setAppearance(miaAppearance);
    // Restituisco la Shape3D così definita.
    return miaShape3D;
}

private BranchGroup caricaPlancia()
{
    // Creo un BranchGroup, carico da file un oggetto e lo aggiungo al BranchGroup, che aggancerò al
    // Content Branch Graph.
    BranchGroup objRootLocale = new BranchGroup();

    ObjectFile f1 = new ObjectFile(ObjectFile.RESIZE);
    try
    {
        Scene s = f1.load("BaseLancio.obj");
        Node nodoBase = s.getSceneGroup();
        objRootLocale.addChild(nodoBase);
    }
    catch(Exception e)
    {
        System.out.println(e.toString());
    }
    return objRootLocale;
}

private BranchGroup caricaMissile()
```

```
{
    // Creo un BranchGroup, carico da file un oggetto e lo aggiungo al BranchGroup, che aggancerò al
Content Branch Graph.
    BranchGroup objRootLocale = new BranchGroup();

    ObjectFile f1 = new ObjectFile(ObjectFile.RESIZE);
    try
    {
        Scene s = f1.load("Polaris.obj");
        Node nodoMissile = s.getSceneGroup();
        objRootLocale.addChild(nodoMissile);
    }
    catch(Exception e)
    {
        System.out.println(e.toString());
    }
    return objRootLocale;
}

private PointLight creaPointLight()
{
    // Creo e definisco una luce di tipo PointLight
    PointLight pl = new PointLight();
    pl.setPosition(0.0f, 10.0f, 0.0f);
    pl.setColor(new Color3f(1.0f, 1.0f, 1.0f));
    pl.setInfluencingBounds(new BoundingSphere(new Point3d(0.0f, 4.0f, 0.0f), 200.0f));
    return pl;
}

public void generaSimulazione(Vector3f vettoreIn)
{
    BranchGroup bgTemp = new BranchGroup();

    // Creo il TransformGroup temporaneo. Posizione iniziale: l'origine
    Transform3D t3d1 = new Transform3D();
    t3d1.setTranslation(new Vector3f(0.0f, 0.0f, 0.0f));
    TransformGroup tgTemp = new TransformGroup(t3d1);
    tgTemp.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    tgTemp.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);

    // Carico il missile nel TransformGroup temporaneo
    tgTemp.addChild(this.caricaMissile());

    // Aggiungo il TG con il missile al BranchGroup temporaneo
    bgTemp.addChild(tgTemp);

    // Genero l'array delle posizioni
    Point3f[] arrayPosizioni = this.calcolaPosizioni(vettoreIn);

    // Creo l'interpolatore per questa animazione e lo aggiungo al BranchGroup
    RotPosPathInterpolator rpqi = this.generaInterpolatore(tgTemp, arrayPosizioni);
    bgTemp.addChild(rpqi);

    // Aggiungo il tutto alla scena, DOPO aver aspettato un pò per sincronizzare l'orologio...
    try
    {
        Thread.sleep((int)(5000-((System.currentTimeMillis()-tempoDiAvvioDelSistema)%5000)));
    }
}
```

```
}
catch(Exception e)
{
}
objRoot.addChild(bgTemp);
}

public Point3f[] calcolaPosizioni(Vector3f vettoreIn)
{
    // Genero 5 posizioni intermedie nello spazio, considerando l'origine come posizione di partenza (ma tale
    vincolo può essere rimosso facilmente...).
    // Anche il numero di passi intermedi può essere cambiato (portando, ad esempio, ad una animazione più
    "fine", aumentando il numero di passi).
    Point3f[] vettorePosizioni = new Point3f[5];
    vettorePosizioni[0] = new Point3f(0.0f, 0.0f, 0.0f);
    vettorePosizioni[1] = new Point3f(new Vector3f(vettoreIn.x/4, vettoreIn.y/4 + 5, vettoreIn.z/4));
    vettorePosizioni[2] = new Point3f(new Vector3f(vettoreIn.x/4*2, vettoreIn.y/4*2 + 5, vettoreIn.z/4*2));
    vettorePosizioni[3] = new Point3f(new Vector3f(vettoreIn.x/4*3, vettoreIn.y/4*3 + 5, vettoreIn.z/4*3));
    vettorePosizioni[4] = new Point3f(vettoreIn);
    return vettorePosizioni;
}

public RotPosPathInterpolator generaInterpolator(TransformGroup tgTemp, Point3f[] arrayPosizioni)
{
    // Questo metodo crea un RotPosPathInterpolator a partire dalle posizioni da individuare nello spazio e dal
    TG da modificare

    // Creo un array di 5 quaternioni (abbiamo definito 5 passi tra la partenza e la destinazione) e li definisco.
    Quat4f[] arrayQuaternioni =
    {
        new Quat4f(),
        new Quat4f(),
        new Quat4f(),
        new Quat4f(),
        new Quat4f()
    };
    arrayQuaternioni[0].set(new AxisAngle4f(0.0f, 0.0f, 1.0f, 0.0f));
    arrayQuaternioni[1].set(new AxisAngle4f(0.0f, 0.0f, 1.0f, -(float)(Math.PI/5*1)));
    arrayQuaternioni[2].set(new AxisAngle4f(0.0f, 0.0f, 1.0f, -(float)(Math.PI/5*2)));
    arrayQuaternioni[3].set(new AxisAngle4f(0.0f, 0.0f, 1.0f, -(float)(Math.PI/5*3)));
    arrayQuaternioni[4].set(new AxisAngle4f(0.0f, 0.0f, 1.0f, -(float)(Math.PI/5*4)));

    // Genero l'array dei nodi. In questo caso, abbiamo 5 nodi.
    float[] arrayNodi = {0.0f, 0.25f, 0.5f, 0.75f, 1.0f};

    // Creo un oggetto Alpha, impostando il loop infinito, increasing enable e 5 secondi di animazione totale
    Alpha mioAlpha = new Alpha(-1, Alpha.INCREASING_ENABLE, 0, 0, 2500, 2500, 0, 0, 0, 0);

    // A partire da tutti i dati raccolti fino a questo punto, creo il RotPosPathInterpolator e lo restituisco
    RotPosPathInterpolator rppi = new RotPosPathInterpolator(mioAlpha, tgTemp, new Transform3D(),
    arrayNodi, arrayQuaternioni, arrayPosizioni);
    rppi.setSchedulingBounds(new BoundingSphere(new Point3d(0.0f, 0.0f, 0.0f), 100.0));
    return rppi;
}
}
```

ESEMPIO ICBM --- GUI

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
/*
    Semplice Pannello-GUI per l'esempio "EsempioICBM".
    La GUI conterrà tre campi di testo e un bottone.
    Alla pressione del bottone, le coordinate inserite nei campi di testo - se valide - verranno inviate a "EsempioICBM"
    per generare la simulazione.
*/
```

```
// Import di base per gestire l'aspetto grafico e i Vector3f
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.vecmath.*;
```

```
public class PannelloPlancia extends JPanel implements ActionListener
{
```

```
    // Elementi della GUI, vettore da creare e oggetto EsempioICBM
    private JTextField coordX, coordY, coordZ;
    private JButton bottone;
    private EsempioICBM esempioICBM;
    private Vector3f vettoreDestinazione;
    private float X, Y, Z;
```

```
    // Costruttore, definisce i vari elementi
    public PannelloPlancia(EsempioICBM eICBM)
    {
        esempioICBM = eICBM;
        coordX = new JTextField("CoordinataX");
        coordY = new JTextField("CoordinataY");
        coordZ = new JTextField("CoordinataZ");
        bottone = new JButton("Go!");
        this.add(coordX);
        this.add(coordY);
        this.add(coordZ);
        this.add(bottone);
        bottone.addActionListener(this);
    }
```

```
    public void actionPerformed(ActionEvent e)
    {
        // La gestione dell'evento è delegata ad un altro metodo
        if(e.getActionCommand() == "Go!")    this.entraInModalitaSimulazione();
    }
```

```
    public void entraInModalitaSimulazione()
    {
        // Vengono recuperati i valori inseriti nei campi testo. Se tali valori rientrano in un range "valido",
        // viene creato - a partire dagli stessi - un Vector3f, il "vettore destinazione", da inviare a EsempioICBM
        // per dare il via alla simulazione.
        X = Float.parseFloat(coordX.getText());
```

```
Y = Float.parseFloat(coordY.getText());
Z = Float.parseFloat(coordZ.getText());
if((X<-50) || (X>50) || (Y<0) || (Z<-50) || (Z>50));
else
{
    vettoreDestinazione = new Vector3f(X, Y, Z);
    esempioICBM.generaSimulazione(vettoreDestinazione);
    bottone.setVisible(false);
}
}
```

ESEMPIO MORPH MAIN

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;

public class EsempioMorphMain extends JFrame
{
    public static void main(String[] args)
    {
        EsempioMorphMain emm = new EsempioMorphMain();
        emm.setVisible(true);
    }

    public EsempioMorphMain()
    {
        this.setTitle("Esempio Morph; Milaneze Francesco; www.redbaron85.com");
        this.setSize(400,400);
        this.setLocation(100,100);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D c3D = new Canvas3D(config);
        BranchGroup scena = creaLoSceneGraph();
        SimpleUniverse sU = new SimpleUniverse(c3D);
        sU.getViewer().getView().setBackClipDistance(10000);
        sU.getViewer().getView().setFrontClipDistance(0.1);
        sU.getViewingPlatform().setNominalViewingTransform();
        TransformGroup vpTG = sU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d=new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f,0.3f,20.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
        scena.addChild(keyNavBeh);
        scena.compile();
        sU.addBranchGraph(scena);
        this.getContentPane().add(c3D,BorderLayout.CENTER);
    }

    private BranchGroup creaLoSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(this.disegnaGriglia());

        // =====
```

[illegible]

```
        return qa;
    }

    private QuadArray creaGeometriaPiramidale()
    {
        float[] vertici =
        {
            // La punta
            0f, 10f, 0f,
            0f, 10f, 0f,
            0f, 10f, 0f,
            0f, 10f, 0f,
            // Parete inferiore
            -10f, -10f, 10f,
            10f, -10f, 10f,
            10f, -10f, -10f,
            -10f, -10f, -10f,
            // Parete Ovest
            0f, 10f, 0f,
            0f, 10f, 0f,
            -10f, -10f, 10f,
            -10f, -10f, -10f,
            // Parete Nord
            0f, 10f, 0f,
            0f, 10f, 0f,
            -10f, -10f, -10f,
            10f, -10f, -10f,
            // Parete Est
            0f, 10f, 0f,
            0f, 10f, 0f,
            10f, -10f, 10f,
            10f, -10f, -10f,
            // Parete Sud
            0f, 10f, 0f,
            0f, 10f, 0f,
            -10f, -10f, 10f,
            10f, -10f, 10f
        };
        QuadArray qa = new QuadArray(24, QuadArray.COORDINATES);
        qa.setCoordinates(0, vertici);
        return qa;
    }

    private QuadArray creaGeometriaCubica()
    {
        float[] vertici =
        {
            // Faccia superiore
            -10f, 10f, -10f,
            10f, 10f, -10f,
            10f, 10f, 10f,
            -10f, 10f, 10f,
            // Faccia inferiore
            -10f, -10f, 10f,
            10f, -10f, 10f,
```

```
        10f, -10f, -10f,
        -10f, -10f, -10f,
        // Faccia Ovest
        -10f, 10f, 10f,
        -10f, 10f, -10f,
        -10f, -10f, -10f,
        -10f, -10f, 10f,
        // Faccia Est
        10f, 10f, 10f,
        10f, 10f, -10f,
        10f, -10f, -10f,
        10f, -10f, 10f,
        // Faccia Nord
        -10f, 10f, -10f,
        10f, 10f, -10f,
        10f, -10f, -10f,
        -10f, -10f, -10f,
        // Faccia Sud
        -10f, 10f, 10f,
        10f, 10f, 10f,
        10f, -10f, 10f,
        -10f, -10f, 10f
    };
    QuadArray qa = new QuadArray(24, QuadArray.COORDINATES);
    qa.setCoordinates(0, vertici);
    return qa;
}

private Shape3D disegnaGriglia()
{
    LineArray rif = new LineArray(132, GeometryArray.COORDINATES | GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int x=0; x<44; x+=4)
    {
        rif.setCoordinate(x+0, new Point3f(-50.0f, 0.0f, l));
        rif.setCoordinate(x+1, new Point3f(50.0f, 0.0f, l));
        rif.setCoordinate(x+2, new Point3f(l, 0.0f, -50.0f));
        rif.setCoordinate(x+3, new Point3f(l, 0.0f, 50.0f));
        l+=10.0f;
    }
    l = -50.0f;
    for(int y=44; y<88; y+=4)
    {
        rif.setCoordinate(y+0, new Point3f(-50.0f, l, 0.0f));
        rif.setCoordinate(y+1, new Point3f(50.0f, l, 0.0f));
        rif.setCoordinate(y+2, new Point3f(l, -50.0f, 0.0f));
        rif.setCoordinate(y+3, new Point3f(l, 50.0f, 0.0f));
        l+=10.0f;
    }
    l = -50.0f;
    for(int z=88; z<132; z+=4)
    {
        rif.setCoordinate(z+0, new Point3f(0.0f, -50.0f, l));
        rif.setCoordinate(z+1, new Point3f(0.0f, 50.0f, l));
        rif.setCoordinate(z+2, new Point3f(0.0f, l, -50.0f));
        rif.setCoordinate(z+3, new Point3f(0.0f, l, 50.0f));
    }
}
```

```
        l+=10.0f;
    }
    Color3f xc = new Color3f(0.1f, 0.9f, 0.1f);
    Color3f yc = new Color3f(0.9f, 0.1f, 0.1f);
    Color3f zc = new Color3f(0.1f, 0.1f, 0.9f);
    for(int x=0; x<44; x++) rif.setColor(x,xc);
    for(int y=44; y<88; y++) rif.setColor(y,yc);
    for(int z=88; z<132; z++) rif.setColor(z,zc);
    return new Shape3D(rif);
}
}
```

ESEMPIO MORPH BEHAVIOR

// Milanese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
import java.util.*;
import javax.media.j3d.*;

public class EsempioMorphBehavior extends Behavior{
    private Alpha mioAlpha; // L'Alpha che determina "l'andamento" dell'animazione
    private Morph mioMorph; // Il Morph-target
    private double pesi[]; // L'array dei pesi. Ad ogni nodo (geometria) é associato un peso. La somma dei pesi deve
    fare 1.
    private WakeupOnElapsedFrames mioWakeup = new WakeupOnElapsedFrames(0); // WakeUp che attiva il
    Behavior: qui é un onElapsedFrames

    public EsempioMorphBehavior(Morph mIn, Alpha aIn){
        mioAlpha = aIn;
        mioMorph= mIn;
        pesi = mioMorph.getWeights();
    }

    public void initialize(){
        mioAlpha.setStartTime(System.currentTimeMillis());
        this.wakeupOn(mioWakeup);
    }

    public void processStimulus(Enumeration criteria){
        double l = pesi.length-1;

        //Azzera l'array dei pesi
        Arrays.fill(pesi, 0);

        //Ricava il valore di alfa
        double val = mioAlpha.value();

        for (int i = 0; i < pesi.length-1; i++)    {
            double start = i/l;
            double stop = (i+1)/l;

            if ((start <= val) && (val <= stop))
            {
                //Calcola il nuovo peso e lo imposta
                double nuovoPeso = (stop-val)/(stop-start);
                pesi[i] = nuovoPeso;
                pesi[i+1] = 1-nuovoPeso;

                //Imposta i nuovi pesi
                mioMorph.setWeights(pesi);

                wakeupOn(mioWakeup);
                return;
            }
        }
    }
}
```

ESEMPIO COLLISIONI

// Milinese Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import com.sun.j3d.utils.picking.behaviors.PickTranslateBehavior;

public class EsempioCollisioni extends JFrame
{
    public static void main(String[] args)
    {
        EsempioCollisioni ec = new EsempioCollisioni();
        ec.setVisible(true);
    }

    public EsempioCollisioni()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Collisioni; Milinese Francesco; Catania, 2009; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 0.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        // Aggiungiamo un Behavior per muovere il cubo
        PickTranslateBehavior pickTranslate = new PickTranslateBehavior(scene, canvas3D, new
BoundingSphere(new Point3d(0.0, 0.0, 0.0), 100.0));
        scene.addChild(pickTranslate);
        //
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(this.creaCuboFisso());
    }
}
```

```
ColorCube cuboDaMuovere = new ColorCube(1f);
Transform3D t3d = new Transform3D();
t3d.setTranslation(new Vector3f(0f, 5f, -10f));
TransformGroup tg = new TransformGroup(t3d);
tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
tg.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
tg.setCapability(TransformGroup.ENABLE_PICK_REPORTING);
tg.addChild(cuboDaMuovere);
objRoot.addChild(tg);

objRoot.addChild(this.creaRiferimento());

BehaviorCollisione mioBC = new BehaviorCollisione(cuboDaMuovere);
objRoot.addChild(mioBC);

return objRoot;
}

private TransformGroup creaCuboFisso()
{
    ColorCube cuboFisso = new ColorCube(1f);
    cuboFisso.setUserData("Il cubo fisso");
    Transform3D t3d = new Transform3D();
    t3d.setTranslation(new Vector3f(0f, 0f, -10f));
    TransformGroup tg = new TransformGroup(t3d);
    tg.addChild(cuboFisso);
    return tg;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(l,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(l,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

BEHAVIOR COLLISIONE

// Esempio Collisioni --- Behavior delle collisioni --- Milaneze Francesco, CT 2009 --- redbaron85@gmail.com --- www.redbaron85.com

```
import javax.media.j3d.Behavior;
import javax.media.j3d.Shape3D;
import javax.media.j3d.Bounds;
import javax.media.j3d.Node;
import javax.media.j3d.WakeupCondition.*;
import javax.media.j3d.WakeupCriterion;
import javax.media.j3d.WakeupOnCollisionEntry;
import java.util.*;
import javax.vecmath.Point3d;
import javax.media.j3d.BoundingSphere;

class BehaviorCollisione extends Behavior
{
    protected Shape3D collidingShape;

    public BehaviorCollisione(Shape3D theShape)
    {
        collidingShape = theShape;
        this.setSchedulingBounds(new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 100.0));
    }

    public void initialize()
    {
        wakeupOn(new WakeupOnCollisionEntry(collidingShape));
    }

    public void processStimulus(Enumeration criteria)
    {
        WakeupCriterion theCriterion = (WakeupCriterion) criteria.nextElement();
        if (theCriterion instanceof WakeupOnCollisionEntry)
        {
            Node theLeaf = ((WakeupOnCollisionEntry) theCriterion).getTriggeringPath().getObject();
            System.out.println("Collisione con " + theLeaf.getUserData());
        }
        wakeupOn(new WakeupOnCollisionEntry(collidingShape));
    }
}
```

SIMULATORE --- MAIN

```
// Francesco Milanese --- Java 3D - Guida di base --- www.redbaron85.com
```

```
/*  
    Progetto: "Simulatore1" (animazione con thread ed eventi da tastiera).  
    Questa classe crea due thread, che si occuperanno di gestire in maniera concorrente (non ci sono primitive di  
    sincronizzazione, qui) la GUI (compresi gli eventi da tastiera) e il "controllo" del "gioco".  
*/
```

```
public class Simulatore1  
{  
    public static void main(String[] args)  
    {  
        FrameSimulatore1 fs1 = new FrameSimulatore1();  
        ThreadFrame1 tf1 = new ThreadFrame1(fs1);  
  
        ThreadControllo1 tc1 = new ThreadControllo1(fs1);  
  
        tf1.run();  
        tc1.run();  
    }  
}
```

SIMULATORE --- THREAD CONTROLLO 1

// Francesco Milanese --- Java 3D - Guida di base --- www.redbaron85.com

```
/*
    Progetto: "Simulatore1" (animazione con thread ed eventi da tastiera).
    Questa classe eredita da Thread e si occupa di recuperare, ad ogni ciclo (l'intervallo tra un ciclo e l'altro dura 1000
    millisecondi ma tale valore può essere modificato),
    il valore della potenza del reattore e di inviare tale informazione al metodo "avanza" del frame FrameSimulatore.
*/
```

```
public class ThreadControllo1 extends Thread
{
    FrameSimulatore1 fs1;
    public ThreadControllo1(FrameSimulatore1 fsIn)
    {
        fs1 = fsIn;
    }

    public void run()
    {
        while(true)
        {
            try
            {
                Thread.sleep(1000);
                fs1.avanza(fs1.getPotenzaReattore());
            }
            catch(Exception e)
            {
                System.out.println(e.toString());
            }
        }
    }
}
```

SIMULATORE --- THREAD FRAME 1

// Francesco Milanese --- Java 3D - Guida di base --- www.redbaron85.com

```
/*  
    Progetto: "Simulatore1" (animazione con thread ed eventi da tastiera).  
    Questa classe eredita da Thread. Si occupa di creare un oggetto FrameSimulatore e di visualizzarlo all'avvio.  
*/
```

```
public class ThreadFrame1 extends Thread  
{  
    FrameSimulatore1 fs1;  
    public ThreadFrame1(FrameSimulatore1 fsIn)  
    {  
        fs1 = fsIn;  
    }  
  
    public void run()  
    {  
        fs1.setVisible(true);  
    }  
}
```

SIMULATORE --- FRAME SIMULATORE 1

// Francesco Milanese --- Java 3D - Guida di base --- www.redbaron85.com

/*

Progetto: "Simulatore1" (animazione con thread ed eventi da tastiera).

Questa classe genera un Frame con una Canvas3D e un SimpleUniverse.

Il SimpleUniverse viene popolato mediante un oggetto "Scena3D".

Al Transform Group della View Platform del SimpleUniverse é associato un Behavior per il controllo degli eventi da tastiera: un oggetto di tipo "mioBehaviorTastiera".

Vi é poi un valore, un intero, che definisce la potenza del reattore del "gioco". Vi sono metodi che permettono di incrementare o decrementare, a passi di 10, tale valore, in un range tra 0 e 110.

Il metodo "avanza" si occupa invece di recuperare il valore della potenza del reattore e calcolare, a partire da questo (ma volendo é possibile tenere conto di altri fattori...), l'entità dello spostamento (=il vettore di trasformazione).

*/

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
```

```
public class FrameSimulatore1 extends JFrame
{
```

```
    private mioBehaviorTastiera mbt;
    private int potenzaReattore;
    private TransformGroup vpTG;
    private Transform3D t3d;
```

```
    public FrameSimulatore1()
    {
```

```
        // Impostazioni "classiche" di un JFrame...
```

```
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        this.setLocation(100, 100);
```

```
        this.setSize(400, 400);
```

```
        this.setTitle("Prova Simulatore 1 - Animazione con Thread ed eventi tastiera; Milanese Francesco; www.redbaron85.com");
```

```
        // Recupero le impostazioni grafiche del sistema
```

```
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
```

```
        // Creo un oggetto Canvas3D, passandogli al costruttore le impostazioni grafiche del sistema
```

```
        Canvas3D canvas3D = new Canvas3D(config);
```

```
        // Aggiungo la Canvas3D al JFrame
```

```
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
```

```
        // Creo il Content Branch Graph: creo un oggetto Scena3D e mi faccio passare la scena di default
```

```
        Scena3D oggettoScena3D = new Scena3D();
```

```
        BranchGroup scena3D = oggettoScena3D.get3Dscene();
```

```
        // Creo un SimpleUniverse, passandogli come parametro la Canvas3D creata precedentemente
```

```
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
```

```
        // Imposto la BackClipDistance
```

```
        simpleU.getViewer().getView().setBackClipDistance(10000);
```

```
// Imposto la FrontClipDistance
    simpleU.getViewer().getView().setFrontClipDistance(0.1);
// Imposto la potenza iniziale del reattore
    potenzaReattore = 0;
// Recupero il TransformGroup che controlla la ViewPlatformTransform, traslo il punto di osservazione
iniziale leggermente indietro e associo, a tale TG, un KeyNavigatorBehavior, per "navigare" nello spazio virtuale con la
tastiera
    vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
    t3d = new Transform3D();
    t3d.setTranslation(new Vector3f(0.0f,0.3f,0.0f));
    vpTG.setTransform(t3d);
    mbt = new mioBehaviorTastiera(vpTG, this);
    mbt.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
    scena3D.addChild(mbt);
// Compilo il ContentBranch Graph
    scena3D.compile();
// Aggiungo al SimpleUniverse il ContentBranch Graph
    simpleU.addBranchGraph(scena3D);
}

public int getPotenzaReattore()
{
    return potenzaReattore;
}

public void setPotenzaReattore(int in)
{
    potenzaReattore = in;
}

public void incPotenzaReattore()
{
    if(potenzaReattore<110) potenzaReattore += 10;
}

public void decPotenzaReattore()
{
    if(potenzaReattore>0) potenzaReattore -= 10;
}

public void avanza(int velocita)
{
    // Trasformazione del TG che controlla la vpTG: quando questo metodo viene invocato dal Thread di
    Controllo, la posizione dell'osservatore viene modificata tenendo conto di alcuni fattori (qui: solo del motore)
    Transform3D t3dAux = new Transform3D();
    Transform3D t3dAux2 = new Transform3D();
    vpTG.getTransform(t3dAux);
    t3dAux2.setTranslation(new Vector3f(0.0f, 0.0f, (float)(-velocita/10)));
    t3dAux.mul(t3dAux2);
    vpTG.setTransform(t3dAux);
}
}
```

SIMULATORE --- MIO BEHAVIOR TASTIERA

// Francesco Milanese --- Java 3D - Guida di base --- www.redbaron85.com

```
/*
    Progetto: "Simulatore1" (animazione con thread ed eventi da tastiera).
    Questa classe cattura e gestisce gli eventi da tastiera.

    Tasto W:      trasla in avanti l'osservatore di 10 unità
    Tasto A: trasla a sinistra l'osservatore di 10 unità
    Tasto S: trasla indietro l'osservatore di 10 unità
    Tasto D: trasla a destra l'osservatore di 10 unità
    Tasti freccia: rotazione dell'osservatore, sul posto, di 10 gradi, nella direzione indicata dalla freccia (in alto, in
basso, a destra, a sinistra).
    Tasto +: richiama il metodo di FrameSimulatore1 che incrementa la potenza del reattore.
    Tasto -: richiama il metodo di FrameSimulatore1 che decrementa la potenza del reattore.
*/

import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import java.util.*;
import java.awt.event.*;

public class mioBehaviorTastiera extends Behavior
{
    // Variabile interna che specifica qual è il Transform Group "target" di questo Behavior (per richiamarlo quando
    // sarà il momento di trasformarlo !)
    private TransformGroup TGtarget;

    private FrameSimulatore1 fs1;

    public mioBehaviorTastiera(TransformGroup tgIn, FrameSimulatore1 fs1in)
    {
        // Costruttore: prende in input il TG "target" e lo associa alla variabile interna
        TGtarget = tgIn;

        fs1 = fs1in;
    }

    // Metodo di inizializzazione: deve essere sovrascritto da tutte le classi che estendono Behavior
    public void initialize()
    {
        // Questo Behavior verrà attivato da eventi di tipo KeyEvent (da tastiera, alla pressione di un tasto)
        this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
    }

    // Metodo processStimulus: deve essere sovrascritto da tutte le classi che estendono Behavior. Viene richiamato
    // quando si verifica un'azione o "stimolo" all'interno del bound d'azione nel quale agisce il Behavior
    public void processStimulus(Enumeration criteria)
    {

```

```
// Creo un array di eventi AWTEvent: conterrà la "coda" di eventi da considerare
AWTEvent [] eventi = null;
Object temp = null;

// Scorro criteria per vedere se ci sono stimoli in ingresso
while(criteria.hasMoreElements())
{
    temp = criteria.nextElement();
    if (temp instanceof WakeupOnAWTEvent)          eventi =
((WakeupOnAWTEvent)temp).getAWTEvent();    // Restituisce un array di eventi di tipo AWT
}

// Creiamo alcune variabili d'appoggio (da riusare) e scorriamo quindi l'array appena generato in cerca di
eventi da tastiera, da trattare come vogliamo noi...
int codiceEvento;
KeyEvent eventoDaTastiera;
Transform3D t3dAux = new Transform3D();
Transform3D t3dAux2 = new Transform3D();
Transform3D t3dTraslazione = new Transform3D();
Vector3f v3fAux = new Vector3f(0.0f, 0.0f, 0.0f);

// Scorriamo l'array degli eventi...
if (eventi != null) // Se l'array degli eventi contiene elementi...
{
    for(int i=0; i < eventi.length; i++) // ...scorri l'array degli eventi...
    {
        if(eventi[i] instanceof KeyEvent) // ... e se l'elemento corrente è un evento generato da
tastiera...
        {
            // Recupero i codici degli eventi da tastiera, per capire quale tasto è stato
premutato
            // I codici degli eventi da tastiera sono quelli visti a Interazione e Multimedia: i
codici Java AWT...
            eventoDaTastiera = (KeyEvent)eventi[i];
            codiceEvento = eventoDaTastiera.getKeyCode();

            // Segue la gestione degli eventi: a seconda di quale tasto è stato premuto,
modifico il TG associato a questo Behavior
            if(codiceEvento == KeyEvent.VK_A) // Tasto A:
spostamento laterale a sinistra
            {
                // Ottengo la trasformazione associata al TG target PRIMA della
modifica e la deposito in t3dAux
                TGtarget.getTransform(t3dAux);
                // Preparo una trasformazione di traslazione
                t3dAux2.setTranslation(new Vector3f(-1.0f, 0.0f, 0.0f));
                // Moltiplico la trasformazione appena creata a quella del TG target
                t3dAux.mul(t3dAux2);
                // Imposto, come nuova trasformazione del TG target, la
trasformazione appena calcolata
                TGtarget.setTransform(t3dAux);
            }
            else if(codiceEvento == KeyEvent.VK_S) // Tasto S: indietro
            {
                // Ottengo la trasformazione associata al TG target PRIMA della
modifica e la deposito in t3dAux
                TGtarget.getTransform(t3dAux);
```

```

        // Preparo una trasformazione di traslazione
        t3dAux2.setTranslation(new Vector3f(0.0f, 0.0f, 1.0f));
        // Moltiplico la trasformazione appena creata a quella del TG target
        t3dAux.mul(t3dAux2);
        // Imposto, come nuova trasformazione del TG target, la
trasformazione appena calcolata
        TGtarget.setTransform(t3dAux);
    }
    else if(codiceEvento == KeyEvent.VK_W)           // Tasto W: avanti
    {
        // Ottengo la trasformazione associata al TG target PRIMA della
modifica e la deposito in t3dAux
        TGtarget.getTransform(t3dAux);
        // Preparo una trasformazione di traslazione
        t3dAux2.setTranslation(new Vector3f(0.0f, 0.0f, -1.0f));
        // Moltiplico la trasformazione appena creata a quella del TG target
        t3dAux.mul(t3dAux2);
        // Imposto, come nuova trasformazione del TG target, la
trasformazione appena calcolata
        TGtarget.setTransform(t3dAux);
    }
    else if(codiceEvento == KeyEvent.VK_D)           // Tasto D: spostamento
laterale verso destra
    {
        // Ottengo la trasformazione associata al TG target PRIMA della
modifica e la deposito in t3dAux
        TGtarget.getTransform(t3dAux);
        // Preparo una trasformazione di traslazione
        t3dAux2.setTranslation(new Vector3f(1.0f, 0.0f, 0.0f));
        // Moltiplico la trasformazione appena creata a quella del TG target
        t3dAux.mul(t3dAux2);
        // Imposto, come nuova trasformazione del TG target, la
trasformazione appena calcolata
        TGtarget.setTransform(t3dAux);
    }
    else if(codiceEvento == KeyEvent.VK_UP)           // Freccia in alto:
ROTAZIONE verso l'alto (rotazione intorno all'asse X)
    {
        // Ottengo la trasformazione associata al TG target PRIMA della
modifica e la deposito in t3dAux
        TGtarget.getTransform(t3dAux);
        // Preparo una trasformazione di rotazione intorno all'asse voluto e
della quantità voluta
        t3dAux2.rotX(Math.PI/18);
        // Moltiplico la trasformazione appena creata a quella del TG target
        t3dAux.mul(t3dAux2);
        // Imposto, come nuova trasformazione del TG target, la
trasformazione appena calcolata
        TGtarget.setTransform(t3dAux);
    }
    else if(codiceEvento == KeyEvent.VK_DOWN)           // Freccia in alto:
ROTAZIONE verso il basso (rotazione intorno all'asse X)
    {
        // Ottengo la trasformazione associata al TG target PRIMA della
modifica e la deposito in t3dAux
        TGtarget.getTransform(t3dAux);
    }

```

```

// Preparo una trasformazione di rotazione intorno all'asse voluto e
della quantità voluta
t3dAux2.rotX(-Math.PI/18);
// Moltiplico la trasformazione appena creata a quella del TG target
t3dAux.mul(t3dAux2);
// Imposto, come nuova trasformazione del TG target, la
trasformazione appena calcolata
TGtarget.setTransform(t3dAux);
}
else if(codiceEvento == KeyEvent.VK_LEFT) // Freccia in alto:
ROTAZIONE verso sinistra (rotazione intorno all'asse Z)
{
// Ottengo la trasformazione associata al TG target PRIMA della
modifica e la deposito in t3dAux
TGtarget.getTransform(t3dAux);
// Preparo una trasformazione di rotazione intorno all'asse voluto e
della quantità voluta
t3dAux2.rotY(Math.PI/18);
// Moltiplico la trasformazione appena creata a quella del TG target
t3dAux.mul(t3dAux2);
// Imposto, come nuova trasformazione del TG target, la
trasformazione appena calcolata
TGtarget.setTransform(t3dAux);
}
else if(codiceEvento == KeyEvent.VK_RIGHT) // Freccia in alto:
ROTAZIONE verso destra (rotazione intorno all'asse Z)
{
// Ottengo la trasformazione associata al TG target PRIMA della
modifica e la deposito in t3dAux
TGtarget.getTransform(t3dAux);
// Preparo una trasformazione di rotazione intorno all'asse voluto e
della quantità voluta
t3dAux2.rotY(-Math.PI/18);
// Moltiplico la trasformazione appena creata a quella del TG target
t3dAux.mul(t3dAux2);
// Imposto, come nuova trasformazione del TG target, la
trasformazione appena calcolata
TGtarget.setTransform(t3dAux);
}
else if(codiceEvento == KeyEvent.VK_PLUS) // Tasto + del
tastierino numerico: aumento la potenza del reattore
{
fs1.incPotenzaReattore();
}
else if(codiceEvento == KeyEvent.VK_MINUS) // Tasto - del
tastierino numerico: diminuisco la potenza del reattore
{
fs1.decPotenzaReattore();
}
}
}
// RESETTO il trigger di questo behavior
this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
}
}

```

SIMULATORE --- SCENA 3D

// Francesco Milaneze --- Java 3D - Guida di base --- www.redbaron85.com

```
/*  
    Progetto: "Simulatore1" (animazione con thread ed eventi da tastiera).  
    Questa classe crea un BranchGroup al quale vengono associati, come figli, due coni e una griglia che serve da  
    riferimento (identifica il piano XZ) per l'osservatore.  
    Si tratta, a tutti gli effetti, di un Content Branch Graph elementare.  
*/
```

```
import java.awt.*;  
import javax.swing.*;  
import javax.vecmath.*;  
import javax.media.j3d.*;  
import com.sun.j3d.utils.universe.*;  
import com.sun.j3d.utils.geometry.*;  
import com.sun.j3d.utils.behaviors.keyboard.*;  
  
public class Scena3D  
{  
    BranchGroup objRoot;  
  
    public Scena3D()  
    {  
        objRoot = new BranchGroup();  
  
        // Aggiungo al Content Branch Graph un pavimento di riferimento  
        objRoot.addChild(this.creaRiferimento());  
  
        // Aggiungo un cono alla scena  
        Cone cono1 = new Cone(0.4f, 1.2f);  
        cono1.setAppearance(new Appearance());  
        Transform3D t3d = new Transform3D();  
        t3d.setTranslation(new Vector3f(0.0f, 0.0f, -5.0f));  
        TransformGroup tg1 = new TransformGroup(t3d);  
        tg1.addChild(cono1);  
        objRoot.addChild(tg1);  
  
        // Aggiungo un secondo cono alla scena  
        Cone cono2 = new Cone(0.4f, 1.2f);  
        cono2.setAppearance(new Appearance());  
        Transform3D t3d2 = new Transform3D();  
        t3d2.setTranslation(new Vector3f(0.0f, 0.0f, -15.0f));  
        TransformGroup tg2 = new TransformGroup(t3d2);  
        tg2.addChild(cono2);  
        objRoot.addChild(tg2);  
    }  
  
    public BranchGroup get3Dscene()  
    {
```

```
        return(this.objRoot);
    }

    private Shape3D creaRiferimento()
    {
        // Creo una Geometry, di tipo LineArray
        LineArray landGeom = new LineArray(44,
        GeometryArray.COORDINATES|GeometryArray.COLOR_3);
        // Imposto un offset per le varie linee della griglia
        float l = -50.0f;
        // Creo le linee della griglia (in pratica, creo coppie di coordinate. Ogni coppia di coordinate costituirà una
        linea della griglia).
        for(int c=0; c<44; c+=4)
        {
            landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
            landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
            landGeom.setCoordinate(c+2, new Point3f(1,0.0f,-50.0f));
            landGeom.setCoordinate(c+3, new Point3f(1,0.0f,50.0f));
            l += 10.0f; // Questo é l'offset vero e proprio: a ogni iterazione, il valore di l
            aumenta di 10 unità
        }
        // Creo un colore
        Color3f c = new Color3f(0.1f,0.8f,0.1f);
        // Coloro tutte le coordinate della griglia (ciò colorerà tutte le linee che collegano le coordinate)
        for(int i=0; i<44; i++) landGeom.setColor(i,c);
        // Restituisco un nuovo oggetto Shape3D, la cui geometria é il LineArray appena definito e colorato
        return new Shape3D(landGeom);
    }
}
```

PIANETA TERRA

// Milaneze Francesco --- Java 3D - Guida di base --- www.redbaron85.com

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import com.sun.j3d.utils.image.TextureLoader;

public class PianetaTerra extends JFrame
{
    public static void main(String[] args)
    {
        PianetaTerra base = new PianetaTerra();
        base.setVisible(true);
    }

    public PianetaTerra()
    {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocation(100, 100);
        this.setSize(400, 400);
        this.setTitle("Esempio Texture mode Modulate; Milaneze Francesco; www.redbaron85.com");
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        this.getContentPane().add(canvas3D, BorderLayout.CENTER);
        BranchGroup scene = createSceneGraph();
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
        simpleU.getViewer().getView().setBackClipDistance(10000);
        simpleU.getViewer().getView().setFrontClipDistance(0.1);
        TransformGroup vpTG = simpleU.getViewingPlatform().getViewPlatformTransform();
        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0.0f, 0.3f, 0.0f));
        vpTG.setTransform(t3d);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTG);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(), 1000.0));
        scene.addChild(keyNavBeh);
        scene.compile();
        simpleU.addBranchGraph(scene);
    }

    private BranchGroup createSceneGraph()
    {
        BranchGroup objRoot = new BranchGroup();
        objRoot.addChild(this.creaRiferimento());

        Transform3D t3d = new Transform3D();
        t3d.setTranslation(new Vector3f(0f, 0f, -3f));
        TransformGroup tg = new TransformGroup(t3d);
        Sphere sfera = new Sphere(1f, Sphere.GENERATE_TEXTURE_COORDS |
Sphere.GENERATE_NORMALS, 100);
```

```
Appearance app = new Appearance();
TextureLoader TL = new TextureLoader("Terra.jpg", null);
ImageComponent2D immagine = TL.getImage();
Texture2D miaTexture = new Texture2D(Texture.BASE_LEVEL, Texture.RGBA, immagine.getWidth(),
immagine.getHeight());
miaTexture.setImage(0, immagine);
TextureAttributes ta = new TextureAttributes();
ta.setTextureMode(TextureAttributes.MODULATE);
app.setMaterial(new Material(new Color3f(1.0f,1.0f,1.0f),new Color3f(0.1f,0.1f,0.1f),new
Color3f(0.0f,1.0f,1.0f),new Color3f(0.0f,0.0f,0.0f),128));
app.setTexture(miaTexture);
app.setTextureAttributes(ta);
sfera.setAppearance(app);

PointLight pl = new PointLight();
pl.setInfluencingBounds(new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 200.0));

objRoot.addChild(pl);
tg.addChild(sfera);
objRoot.addChild(tg);
return objRoot;
}

private Shape3D creaRiferimento()
{
    LineArray landGeom = new LineArray(44, GeometryArray.COORDINATES|GeometryArray.COLOR_3);
    float l = -50.0f;
    for(int c=0; c<44; c+=4)
    {
        landGeom.setCoordinate(c+0, new Point3f(-50.0f,0.0f,l));
        landGeom.setCoordinate(c+1, new Point3f(50.0f,0.0f,l));
        landGeom.setCoordinate(c+2, new Point3f(1,0.0f,-50.0f));
        landGeom.setCoordinate(c+3, new Point3f(1,0.0f,50.0f));
        l += 10.0f;
    }
    Color3f c = new Color3f(0.1f,0.8f,0.1f);
    for(int i=0; i<44; i++) landGeom.setColor(i,c);
    return new Shape3D(landGeom);
}
}
```

```
*      *      *

*      *      *

*      *      *
```