

Dev.

La rivista che ti insegna a programmare

DATABASE

Esempio di ETL: caricare una Dimensione

PHP

Thumbnails con php

JSP

creare tag personalizzati con l'interfaccia SimpleTag

SICUREZZA

Tecniche di Pharming

GLADE 3

la nuova release del celebre costruttore di GUI

Provato per voi

Compuware DevPartner
SecurityCheker 2.0 codice
ASP.NET in tutta sicurezza

**Programmo
Subito**

Qt Jambi

Programmare adeskleets con
Python e adeskleets

Grafica 3D su cellulari in Retained Mode con JSR 184

- Algomatica del doctor Brom
- Costruzione Interfacce:
le primitive grafiche
- Anjuta IDE

di Roberto Casarini
rcasarini@infomedias.it

Lettori ... speciali

Come molti di voi avranno intuito dalle pagine presenti in questi numeri di DEV, stiamo varando una nuova creatura: Java Journal, la prima rivista italiana completamente dedicata al Java. Il linguaggio di Sun Microsystems in questi anni si è fatto largo tra i più blasonati C e C++: dai numerosi libri pubblicati in italiano che trattano Java come linguaggio base, ai corsi universitari che già da diversi anni cercano di formare programmatori Java, viste anche le richieste di mercato.

Come ricorderete, nello scorso editoriale vi ho posto un quesito in merito alla presenza dei listati nella rivista. Abbiamo ricevuto moltissimi pareri favorevoli: a giudizio dei più, lo spazio dedicato ai listati è gradito, essenziale e va preservato (se non incrementato). Venendo all'ultima domanda, ossia se il livello di difficoltà raggiunto da DEV sia adeguato, abbiamo ricevuto pareri molto vari e discordanti.

Innanzitutto, ringrazio vivamente tutti coloro che ci hanno scritto; comprendendo la ritrosia a rispondere per iscritto via email a quesiti di questo genere; in futuro cercheremo di prediligere soluzioni più a portata di mano, e soprattutto anonime, come ad esempio i forum (stiamo diventando sempre più pigri...questa è la verità...). Nel frattempo, se qualcuno vuole ancora esprimere un parere con una "tradizionale" email è comunque il benvenuto. Dovendo modellare DEV a vostra "immagine e somiglianza" è importante scambiare opinioni sul livello di difficoltà e di interesse per ogni numero di DEV che progettiamo. DEV è sì la rivista che "insegna a programmare", ma col passar del tempo si sono aperti anche spazi di "livello superiore", graditi maggiormente dai professionisti del mondo IT che vogliono mantenersi aggiornati. Qualche lettore, in merito a una opinione sul livello di difficoltà di DEV, si è generosamente prodigato in una attenta analisi personale degli ultimi speciali pubblicati. A questo proposito mi incuriosisce sapere quale è stato per voi lo speciale più interessante, e anche qual è quello che avete meno gradito. In questo modo possiamo provare a stilare una classifica sul gradimento "speciale" per gli argomenti trattati ad oggi e tenerne conto nella progettazione futura. Per poter riuscire nell'impresa, al solito, abbiamo bisogno dei vostri pareri, dei vostri consigli e di essere pungolati anche delle vostre critiche. Dopotutto ... non c'è rosa senza spine!

Roberto Casarini



10



19



45



78

Sei agli inizi? Non riesci a risolvere i primi problemi? Niente paura! Iscriviti alla mailing-list Informedia!

programmato subito@informedia.it
e knuxit@informedia.it

I listati e i programmi di utilità che corredano gli articoli sono disponibili gratuitamente dal sito <http://www.informedia.it/DEV> oppure da <ftp://ftp.informedia.it/pub/DEV/Listati/>

Lo speciale di questo numero:

JSR184: Retained Mode

- 10 **JSR184 - In Retained Mode - Parte 1**
di Mauro Pesci
- 16 **JSR184 - In Retained Mode - Parte 2**
di Mauro Pesci
- 19 **Esportare un modello tridimensionale in M3G da 3D studio MAX**
di Maurizio Terzo

Programmazione

- 23 **Automazione - Puntatori in Step7**
di Fulvio Mazzeo
- 27 **Cellulari - Short Message Service - Sezioni DCS, OA e DA e SCTS**
di Vito Vessia
- 33 **Cellulari - .NET 3.0 uno sguardo al futuro**
di Gian Maria Ricci
- 38 **Database - Esempio di ETL: caricare una Dimensione**
di Angelo Bruno
- 45 **PHP - Thumbnails con php**
di Max Arcidiacono
- 49 **JSP - JSP: creare tag personalizzati con l'interfaccia SimpleTag - parte 2**
di Barry Bassi
- 57 **Sicurezza - Tecniche di Pharming**
di Stefano Corti

Rubriche

- 60 **Algoritmi del dottor Brom - Oltre i quadrati**
di Maurizio Codogno
- 67 **Costruzione Interfacce - Le primitive grafiche**
di Antonio Cisternino
- 96 **Vista Business - Vista Business**
di Paolo De Niccolis

Programmo Subito

Numero 97

- 71 **Livello 1** Qt Jambi
di Fabrizio Angius
- 73 **Livello 2** Programmare adeskleets con Python e adeskleets
di Antonio Di Salvo

Linux Subito

Numero 77

- 78 **Livello 2** Anjuta IDE
di Alessio Saltarin
- 86 **Livello 1** Glade 3 la nuova release del celebre costruttore
di GUI
di Matteo De Felice

In Prova

Compuware DevPartner SecurityChecker 2.0
codice ASP.NET in tutta sicurezza
di Paolo De Nicolis

88

Ai lettori

Planet Open Source a cura di Alessio Saltarin
#6 - Google Summer Of Code

44

Java ME World a cura di Stefano Sanna
Web 2.0

72

DEV in DirICTo a cura di Massimo Farina
La decompilazione del software: i diritti dell'utilizzatore

77

DEV Libri a cura di Roberto Casarini
INGEGNERIA DEL SOFTWARE: Creatività e metodo

77

www.infomedia.it - The Software Side Of Your Mind

Dev Developing Software Solutions Numero 145
Novembre 2006
Anno XIV

Direttore Responsabile
Mariateresa Mari (mmari@infomedia.it)

Coordinamento Editoriale
Roberto Casarini (rcasarini@infomedia.it)

Collaboratori

Fabrizio Angius, Max Arcidiacono
Barry Bassi, Angelo Bruno,
Antonio Cisternino, Maurizio Codogno,
Stefano Corti, Matteo De Felice,
Paolo De Nicolis, Massimo Farina,
Pulvio Matteo, Mauro Pesci,
Gian Maria Ricci, Alessio Saltarin,
Stefano Sanna, Maurizio Terto,
Vito Vesna

Grafica e Impaginazione
Valentina Mucci (grafica@gruppoinfomedia.it)

GRUPPO EDITORIALE
INFOMEDIA

Gruppo Editoriale Infomedia srl
Via Valdara P.116 56038 Pontassio (PI)
Tel. 0587 73 64 60 (r.a.)
Fax 0587 73 22 32
www.infomedia.it - red_dev@infomedia.it

Direzione

Natale Fico (nfico@infomedia.it)

Divisione Libri

Lisa Vanni (lvanni@infomedia.it)

Marketing & Advertising

Tel. 0587 73 64 60
marketing@infomedia.it

Supporto Sistemistico

webmaster@infomedia.it

Amministrazione

Sara Mattei (samattei@infomedia.it)

Segreteria

Enrica Nanni (enr@infomedia.it)

Stampa

Betagraf Spa - Fano di Argelato (BO)

Distribuzione

Parrini & C. (Roma)

Ufficio Abbonamenti

Tel. 0587 73 64 60
Fax 0587 73 22 32
abbonamenti@infomedia.it
http://online.infomedia.it

Si prega di inviare i comunicati stampa e gli inviti stampa per la redazione all'indirizzo: comunicatistampa@infomedia.it

La riproduzione con qualsiasi mezzo di illustrazioni e di articoli pubblicati sulla rivista, nonché la loro traduzione, è riservata e non può avvenire senza espressa autorizzazione dell'editore. I manoscritti e le illustrazioni inviati alla redazione non saranno restituiti anche se non pubblicati e l'editore non si assume alcuna responsabilità per il caso che si tratti di esemplari unici.

"DEV. Developing Software & Solutions"
è una rivista del Gruppo Editoriale Infomedia S.r.l.
Divisione e Amministrazione
Via Valdara P. 116 - Pontassio
Registrazione presso il Tribunale di Pisa
n. 21 del 25.11.1999
Consenso pubblicitario inferiore al 45%

JSR184 – In Retained Mode

Il Retained Mode di M3G permette di lavorare alle scene di un gioco tramite una rappresentazione gerarchica della struttura del mondo 3D e delle sue componenti

PARTE 1

di Mauro Pesci > mpesci@infomedia.it

Nello speciale di Settembre ho spiegato come è possibile creare scene 3D utilizzando l'*Immediate Mode*, il quale permette di disegnare direttamente a video gli oggetti. È lecito pensare a questo metodo di utilizzo delle M3G come ad un accesso a funzioni di basso livello.

È ovvio che tale approccio mal si adatta a scopi dove la complessità dei mondi da gestire rappresenta una componente fortemente determinante. Anziché effettuare il rendering di ogni singolo oggetto e cambiare le impostazioni della scena ad ogni fotogramma, può risultare molto utile disporre in memoria di tutti i dati strutturati in qualche modo.

Come vedremo, questa modalità è anche particolarmente utile quando si disegnano intere scene con programmi di modellazione 3D come 3D Studio Max o Blender.

Organizzazione di una scena

In *Retained Mode* è possibile immagazzinare, tramite una struttura ad albero, tutte le componenti del mondo 3D compresi oggetti, telecamere, luci e (addirittura) informazioni sulle animazioni.

Ogni elemento della struttura ad albero è costituito da una classe derivata da Node. Tali classi sono

TABELLA 1

	IMMEDIATE	RETAINED
Camera	Imposta la telecamera corrente Graphics3D.setCamera(Camera camera, Transform transform)	Aggiunge una o più telecamere al mondo come nodo figlio e ne attiva una World.addChild(Node child) World.setActiveCamera(Camera camera)
Light	Imposta una fonte di luce Graphics3D.addLight(Light light, Transform transform) Graphics3D.setLight(int index, Light light, Transform transform) Graphics3D.resetLights()	Aggiunge una fonte di luce al mondo come nodo figlio World.addChild(Node child)
Transformation	Passa un oggetto trasformazione alla funzione di rendering Graphics3D.render(Node node, Transform transform)	Imposta una trasformazione ad un determinato nodo dell'albero che compone il mondo Node.setTransform(Transform transform)
Rendering	Effettua il rendering di una scena Graphics3D.render(Node node, Transform transform) Graphics3D.render(VertexBuffer vertices, IndexBuffer triangles, Appearance appearance, Transform transform, int scope) Graphics3D.render(VertexBuffer vertices, IndexBuffer triangles, Appearance appearance, Transform transform)	Effettua il rendering del mondo compresi tutti i nodi figli Graphics3D.render(World world)

- **Camera**, la quale rappresenta un punto di osservazione all'interno della scena
- **Mesh**, definisce un oggetto costituito da un insieme di triangoli con associate le proprietà dei materiali
- **Sprite3D**, definisce un oggetto costituito da un'immagine 2D che può occupare una qualunque posizione all'interno dello spazio 3D. L'immagine risulta sempre orientata con la faccia in direzione del punto d'osservazione.
- **Light**, definisce la posizione, l'orientamento, il colore e gli attributi di una fonte di luce
- **Group**, serve da radice per creare un sottoinsieme di nodi

Ogni nodo definisce un sistema di coordinate locali che può essere trasformato in relazione a quello dell'eventuale nodo genitore. Già questo concetto mostra chiaramente quale può essere l'utilità di avere una struttura ad albero per memorizzare gli oggetti.

Utilizzando, ad esempio, una classe Group è possibile organizzare una serie di oggetti che vogliamo far muovere nello spazio con la semplice trasformazione delle coordinate di tale classe.

Esiste anche una classe speciale derivata da Node, ossia la classe World. Questa rappresenta la radice di qualunque mondo 3D si voglia creare.

In **Figura 1** viene presentato un esempio di strutturazione di un mondo 3D. Osservate come, tramite una tale struttura, per effettuare il rendering di una scena diventa sufficiente passare come argomento l'oggetto World al metodo Render()

della classe Graphics3D.

Osserviamo ora le principali differenze tra le due modalità. A questo scopo viene riportato, in **Tabella 1**, uno specchietto riepilogativo che compara le chiamate ai metodi nell'una e nell'altra modalità.

Immediate Mode e *Retained Mode* non sono mutuamente esclusive; infatti è sempre possibile mischiarle utilizzando opportunamente i metodi Render().

Finalmente un po' di codice

Mettiamo in pratica i concetti appena visti utilizzando parte del codice degli esempi proposti negli articoli precedenti. È il momento di realizzare la scena secondo la struttura in **Figura 1**.

La modalità "Retained" è anche molto utilizzata per la possibilità di caricare agevolmente oggetti e, addirittura, intere scene modellate con tool tipo 3DS Max e Blender

Innanzitutto, all'interno del metodo Init() della classe GameCanvas è necessario scrivere tutte le istruzioni che generano la scena nell'oggetto World da renderizzare. Il codice del metodo è riportato per intero nel **Listato 1**.

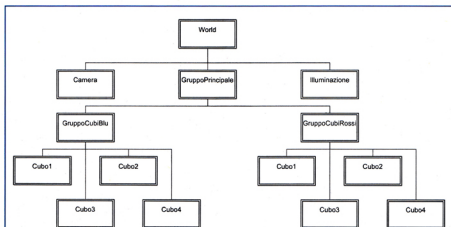


FIGURA 1 Lo schema della struttura ad albero che costituirà la scena

LISTATO 1

Il codice del metodo `init()` necessario ad inizializzare tutta la struttura della scena

```
// Creo un nuovo nodo di tipo World
world = new World();

// Creo una nuova telecamera con tanto di correzione prospettiva
Camera camera = new Camera();
float aspectRatio = (float) getWidth() / (float) getHeight();
camera.setPerspective(30.0f, aspectRatio, 1.0f, 1000.0f);

// Riposiziono la telecamera
camera.setTranslation(0.0f, 0.0f, 10.0f);

// Aggiungo la camera come nodo figlio del nodo World
world.addChild(camera);

// Imposto la telecamera come attiva
world.setActiveCamera(camera);

// Creo una nuova fonte di luce di tipo OMNI
Light light = new Light();
light.setMode(Light.OMNI);

// Riposiziono la fonte di luce
light.setTranslation(0.0f, 0.0f, 3.0f);

// Aggiungo la fonte di luce come nodo figlio del nodo World
world.addChild(light);

// Creo un nuovo VertexBuffer per memorizzare i vertici del cubo Blu
VertexBuffer datiVerticiCuboBlu = new VertexBuffer();

// Imposto il colore dei vertici a Blu
datiVerticiCuboBlu.setDefaultColor(0x0000FF);

// Creo un nuovo VertexBuffer per memorizzare i vertici del cubo Rosso
VertexBuffer datiVerticiCuboRosso = new VertexBuffer();

// Imposto il colore dei vertici a Rosso
datiVerticiCuboRosso.setDefaultColor(0xFF0000);

// Creo il VertexArray che conterrà le coordinate dei vertici del cubo
VertexArray posizioniVertici = new VertexArray
    (POSIZIONI_VERTICI.length/3, 3, 1);

// Memorizzo i dati nel VertexArray
posizioniVertici.set(0, POSIZIONI_VERTICI.length/3, POSIZIONI_VERTICI);

// Imposto i dati dei vertici su ognuno dei due VertexBuffer
datiVerticiCuboBlu.setPositions(posizioniVertici, 1.0f, null);
datiVerticiCuboRosso.setPositions(posizioniVertici, 1.0f, null);

// Creo ora un nuovo VertexArray per memorizzare i dati delle normali
VertexArray normaliVertici = new VertexArray
    (NORMALI_VERTICI.length/3, 3, 1);

// Memorizzo i dati delle normali
normaliVertici.set(0, NORMALI_VERTICI.length/3, NORMALI_VERTICI);

// Inserisco i dati delle normali nel VertexBuffer dei due cubi
datiVerticiCuboBlu.setNormals(normaliVertici);
datiVerticiCuboRosso.setNormals(normaliVertici);

// Creo la catena di triangoli che costituisce il cubo

TriangleStripArray triangoliCubo = new TriangleStripArray
    (INDICI_TRIANGOLI, LUNGHEZZE_TRIANGOLI);

// Creo un nuovo materiale
Material material = new Material();
material.setVertexColorTracking(enable(true));

// Creo un nuovo oggetto Appearance per impostare il materiale creato
Appearance appearance = new Appearance();
appearance.setMaterial(material);

// Creo la struttura gerarchica degli oggetti nella scena tramite un gruppo
// generale al quale saranno agganciati i due sottogruppi di cubi
Group tuttiCubi = new Group();
tuttiCubi.setUserID(ID_TUTTI_I_CUBI);

// Aggiungo il gruppo come nodo figlio dell'oggetto World
world.addChild(tuttiCubi);

// Creo il sottogruppo dei cubi Blu
Group cubiBlu = new Group();
cubiBlu.setUserID(ID_CUBI_BLU);

// Aggiungo il sottogruppo come nodo figlio del gruppo tuttiCubi
tuttiCubi.addChild(cubiBlu);

// Creo il sottogruppo dei cubi Rossi
Group cubiRossi = new Group();
cubiRossi.setUserID(ID_CUBI_ROSSI);

// Aggiungo il sottogruppo come nodo figlio del gruppo tuttiCubi
tuttiCubi.addChild(cubiRossi);

// Creo i nodi Mesh dei cubi disposti a cerchio
for (int i=0; i<8; i++)
{
    Mesh meshCubo = null;

    // Genero alternativamente un cubo Blu e uno Rosso
    if ((i%2) == 0)
    {
        // Creo un nuovo oggetto Mesh con le impostazioni dei cubi Blu
        meshCubo = new Mesh(datiVerticiCuboBlu, triangoliCubo, appearance);
        // Aggiungo l'oggetto Mesh come nodo figlio del sottogruppo cubiBlu
        cubiBlu.addChild(meshCubo);
    }
    else
    {
        // Creo un nuovo oggetto Mesh con le impostazioni dei cubi Rossi
        meshCubo = new Mesh(datiVerticiCuboRosso, triangoliCubo, appearance);
        // Aggiungo l'oggetto Mesh come nodo figlio del sottogruppo cubiRossi
        cubiRossi.addChild(meshCubo);
    }
}

// Applico le opportune trasformazioni al cubo appena creato
meshCubo.setTranslation(1.5f * (float) Math.cos(i*Math.PI/4), 1.5f *
    (float) Math.sin(i*Math.PI/4), 0.0f);

meshCubo.setScale(0.4f, 0.4f, 0.4f);
}
```

Per prima cosa creiamo un nuovo nodo di tipo World

```
world = new World();
```

procediamo, ora, aggiungendo una telecamera alla scena, impostando anche la correzione prospettiva

```
Camera camera = new Camera();
float aspectRatio = (float) getWidth() / (float) getHeight();
camera.setPerspective(30.0f, aspectRatio, 1.0f, 1000.0f);
```

alla telecamera è necessario applicare una trasformazione di traslazione per allontanarla dal centro della scena in modo da poter visualizzare tutti gli oggetti che la comporranno

```
camera.setTranslation(0.0f, 0.0f, 10.0f);
```

la telecamera deve essere aggiunta alla scena come nodo alla radice World

```
world.addChild(camera);
```

dopo questa operazione, tramite la chiamata successiva, tale telecamera, diventa quella attiva. Ricordo che una telecamera attiva determina la posizione e l'orientamento del punto d'osservazione.

```
world.setActiveCamera(camera);
```

Aggiungiamo anche una fonte di luce di tipo OMNI

```
Light light = new Light();
light.setMode(Light.OMNI);
```

applico una traslazione anche alla fonte di luce, allontanandola dal centro della scena, in direzione dell'asse Z.

```
light.setTranslation(0.0f, 0.0f, 3.0f);
```

Anche la fonte di luce appena creata diventa un nodo aganciato direttamente alla radice World

```
world.addChild(light);
```

Il "Retained Mode" è un metodo di gestione delle scene 3D molto più comodo dell' "Immediate Mode", specialmente nel caso di mondi molto complessi

Ora procediamo con il creare la struttura gerarchica dei vari gruppi che conterranno due serie di cubi orientati in cerchio rispetto al centro della scena. Il primo gruppo che

viene creato conterrà due sottogruppi; uno per una serie di cubi Blu, mentre l'altro per i cubi Rossi.

```
Group tuttiCubi = new Group();
```

associamo un ID al gruppo principale. L'ID è necessario per poter recuperare, successivamente, il riferimento di un particolare oggetto nodo.

```
tuttiCubi.setUserID(ID_TUTTI_I_CUBI);
```

come ogni altro oggetto, il gruppo principale, va aggiunto alla struttura della scena

```
world.addChild(tuttiCubi);
```

Ad ogni nodo della struttura è possibile applicare trasformazioni che si riflettono su tutti i nodi figlio

Creiamo un sottogruppo, al quale saranno agganciati tutti i cubi di colore blu, e gli associamo uno specifico ID

```
Group cubiBlu = new Group();
cubiBlu.setUserID(ID_CUBI_BLU);
```

a questo punto il gruppo appena creato deve diventare un nodo figlio del gruppo principale

```
tuttiCubi.addChild(cubiBlu);
```

effettuiamo gli stessi passi con il gruppo per i cubi rossi

```
Group cubiRossi = new Group();
cubiRossi.setUserID(ID_CUBI_ROSSI);
tuttiCubi.addChild(cubiRossi);
```

È ora di creare gli oggetti cubo. Il procedimento è identico a quello seguito nei precedenti articoli; ossia si crea il solito oggetto VertexBuffer, in questo caso per il cubo di colore blu

```
VertexBuffer datVerticiCubiBlu = new VertexBuffer();
```

la seguente istruzione permette di specificare un colore di default per tutti i vertici specificati

```
datVerticiCubiBlu.setDefaultColor(0x000000FF);
```

creiamo anche il VertexBuffer per i dati del cubo di colore rosso

```
VertexBuffer datVerticiCubiRosso = new VertexBuffer();
```

anche in questo caso impostiamo il colore di default per i vertici

```
datiVerticiCuboRosso.setDefaultColor(0x00FF0000);
```

Tramite le due linee successive viene creato un `VertexArray` nel quale vengono memorizzati i valori dei vertici

```
VertexArray posizioniVertici = new
    VertexArray(POSIZIONI_VERTICI.length/3, 3, 1);
posizioniVertici.set(0, POSIZIONI_VERTICI.length/3, POSIZIONI_VERTICI);
```

a questo punto i dati contenuti nell'array sono passati ai `VertexBuffer` dei due cubi

```
datiVerticiCuboBlu.setPositions(posizioniVertici, 1.0f, null);
datiVerticiCuboRosso.setPositions(posizioniVertici, 1.0f, null);
```

Allo stesso modo è necessario caricare i dati delle normali di ogni vertice in un `VertexArray`

```
VertexArray normaliVertici = new
    VertexArray(NORMALI_VERTICI.length/3, 3, 1);
normaliVertici.set(0, NORMALI_VERTICI.length/3, NORMALI_VERTICI);
```

e passarli ai rispettivi `VertexBuffer`

```
datiVerticiCuboBlu.setNormals(normaliVertici);
datiVerticiCuboRosso.setNormals(normaliVertici);
```

creiamo un `TriangleStrip` per i triangoli che compongono

il cubo

```
TriangleStripArray triangoliCubo = new
    TriangleStripArray(INDICI_TRIANGOLI,
        LUNGHEZZE_TRIANGOLI);
```

attivando l'illuminazione è d'obbligo creare un oggetto `Appearance` per impostare le caratteristiche del materiale che riflette la luce

```
Material material = new Material();
material.setVertexColorTrackingEnable(true);
Appearance appearance = new Appearance();
appearance.setMaterial(material);
```

in ultimo creiamo un certo numero di cubi in cerchio rispetto al centro della scena. Ogni cubo è un oggetto `Mesh` che viene aggiunto alla struttura nel proprio sottogruppo, a seconda che si tratti di un cubo blu

```
meshCubo = new Mesh(datiVerticiCuboBlu, triangoliCubo, appearance);
cubiBlu.addChild(meshCubo);
```

O FOSCO

```
meshCubo = new Mesh(datiVerticiCuboRosso, triangoliCubo, appearance);
cubiRosso.addChild(meshCubo);
```

Se provassimo ad eseguire il programma otterremmo un'immagine come quella in **Figura2**. Abbiamo però semplicemente realizzato una scena statica. Proviamo ad applicare

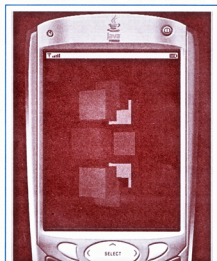


FIGURA 2

L'esempio in esecuzione dove viene applicata una rotazione al gruppo principale

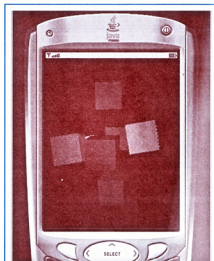


FIGURA 3

L'esempio in esecuzione dove viene applicata una rotazione anche ai due sottogruppi dei cubi colorati

alcune trasformazioni ai vari gruppi di oggetti per vedere cosa accade.

Per ottenere questo risultato, nel metodo `frame()` della classe `GameCanvas`, inseriamo le seguenti linee di codice.

```
Group tuttiCubi = (Group)world.find(ID_TUTTI_I_CUBI);
```

Questa trova il riferimento all'oggetto `Group` che rappresenta il gruppo principale, cercandolo tramite l'ID che precedentemente gli avevamo assegnato.

```
tuttiCubi.postRotate(1.0f, 0.0f, 1.0f, 0.0f);
```

La quale applica la trasformazione al gruppo principale. Lanciando il programma vedremmo ruotare l'intero insieme di cubi intorno all'asse Y, mentre aggiungendo le seguenti linee

```
Group cubiBlu = (Group)world.find(ID_CUBI_BLU);
```

```
cubiBlu.postRotate(1.0f, 0.0f, 0.0f, 1.0f);
```

```
Group cubiRossi = (Group)world.find(ID_CUBI_ROSSI);
```

```
cubiRossi.postRotate(-1.0f, 1.0f, 0.0f, 0.0f);
```

vedremmo i cubi ruotare in funzione alla rotazione del gruppo di appartenenza, come mostrato in **Figura3**.

Conclusioni

Con un semplice esempio abbiamo potuto sperimentare le facilitazioni offerte dal questa modalità di utilizzo delle M3G. Ovviamente, suggerisco di fare sempre riferimento alla documentazione delle classi, per ogni approfondimento in merito.

La documentazione è presente sul sito FTP nel file associato a questo numero.

Bibliografia

- [1] Sun Corporation, "Mobile 3D Graphics API Specification", www.sun.com

Mauro Pesci

Lavora come libero professionista nel settore dello sviluppo del software, in particolar modo, si sta occupando della realizzazione di videogiochi commerciali per telefonini cellulari distribuiti, principalmente, tramite il portale di H3G.

WHO?

WHAT?

YOU!



GRUPPO EDITORIALE
INFOMEDIA
THE SOFTWARE SIDE OF YOUR MIND



Se l'informatica è il tuo "pane quotidiano" e vuoi scrivere un articolo per condividere le tue conoscenze su un argomento mai affrontato sulle pagine della rivista, questa è l'occasione giusta!

Cerchiamo autori in grado di scrivere articoli sulla programmazione in C/C++, C#, Delphi, Visual Basic, VB.NET o nei principali linguaggi di scripting per il Web (ASP.NET, Javascript, Perl, PHP, Python, VBScript). Ma anche sulle tecniche di debugging, di testing, di ottimizzazione e modellazione del codice, su ambienti di sviluppo, database, sistemi operativi, sicurezza, wireless...

**Dimostra la tua abilità:
scrivi un articolo per Dev.**

Invia la tua proposta a dev-proposte@infomedia.it indicando a chi si rivolge, il possibile titolo, una breve descrizione e le caratteristiche dell'eventuale codice presentato.



La rivista che ti insegna a programmare

JSR184 – In Retained Mode

Una caratteristica molto utile del "Retained Mode" è la possibilità di caricare oggetti 3D realizzati con programmi di modellazione

PARTE 2

di Mauro Pesci > mpesci@infomedia.it

Dovendo realizzare un videogioco interamente in 3D o anche una demo, sicuramente, si avrà bisogno di creare e disegnare a video diversi oggetti e non sempre semplici come le primitive (cubi, piramidi, ecc.). È ovvio che diventerebbe un lavoro molto stressante, se non impossibile, specificare a mano, dato per dato, tutte le informazioni necessarie a definire tali oggetti. Se avete caricato e compilato i progetti per NetBeans degli esempi presentati negli articoli precedenti, vi siete sicuramente accorti della quantità di passi necessari alla definizione di un semplice cubo. Per maggiore chiarezza ne riporto un estratto.

```
private static final byte[] POSIZIONI_VERTICI = {
    -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 1, // fronte
    1, -1, -1, -1, -1, -1, 1, 1, -1, 1, -1, 1, // retro
    1, -1, 1, 1, -1, -1, 1, 1, 1, 1, 1, -1, // destra
    -1, -1, -1, -1, 1, -1, 1, -1, -1, 1, 1, 1, // sinistra
    -1, 1, 1, 1, 1, 1, -1, 1, -1, 1, 1, -1, // alto
    -1, -1, -1, 1, -1, -1, -1, 1, 1, 1, 1, -1, // basso
};

private static int[] INDICI_TRIANGOLI = {
    0, 1, 2, 3, // fronte
    4, 5, 6, 7, // retro
    8, 9, 10, 11, // destra
    12, 13, 14, 15, // sinistra
    16, 17, 18, 19, // alto
    20, 21, 22, 23, // basso
};

private static final byte[] NORMALI_VERTICI = {
    0, 0, 127, 0, 0, 127, 0, 0, 127, 0, 0, 127, // fronte
    0, 0, -128, 0, 0, -128, 0, 0, -128, 0, 0, -128, // retro
    127, 0, 0, 127, 0, 0, 127, 0, 0, 127, 0, 0, // destra
    -128, 0, 0, -128, 0, 0, -128, 0, 0, -128, 0, 0, // sinistra
    0, 127, 0, 0, 127, 0, 0, 127, 0, 0, 127, 0, // alto
};
```

```
0, -128, 0, 0, -128, 0, 0, -128, 0, 0, -128, 0, // basso
};

private static int[] LUNGHEZZE_TRIANGOLI = {
    4, 4, 4, 4, 4, 4
};

private static final byte[] COLORI_VERTICI = {
    0, (byte)255, 0, 0, (byte)255, (byte)255,
    (byte)255, 0, 0, (byte)255, 0, (byte)255,
    (byte)255, (byte)255, 0, (byte)255, (byte)255, (byte)255,
    0, 0, (byte)128, 0, 0, (byte)255
};
```

Oltre a questo non dimentichiamo che per creare in memoria ogni oggetto, è necessario creare un `VertexBuffer` per mantenere le informazioni, vari `VertexArray` per caricare i dati, uno o più `TriangleStrip` e almeno un oggetto `Appearance`. In codice

```
datiVerticiCubo = new VertexBuffer();
VertexArray posizioniVertici = new VertexArray(POSIZIONI_VERTICI.length / 3,
3, 1);
posizioniVertici.set(0, POSIZIONI_VERTICI.length / 3, POSIZIONI_VERTICI);
datiVerticiCubo.setPositions(posizioniVertici, 1.0f, null);
triangoliCubo = new TriangleStripArray(INDICI_TRIANGOLI, new int[]
(INDICI_TRIANGOLI.length));
appareanzaCubo = new Appearance();
modalitaPoligoni = new PolygonMode();
appareanzaCubo.setPolygonMode(modalitaPoligoni);
VertexArray coloriVertici = new VertexArray(COLORI_VERTICI.length / 3, 3, 1);
coloriVertici.set(0, COLORI_VERTICI.length / 3, COLORI_VERTICI);
datiVerticiCubo.setColors(coloriVertici);
```

Immaginate di dover reiterare queste operazioni per ogni oggetto da visualizzare e vi renderete conto che avere la possibilità di caricare direttamente in memoria gli oggetti contenuti in un file consente di semplificare enormemente il lavoro.

Utilizzare i file m3g

JSR-184, a questo scopo, mette a disposizione un tipo di file binario, con estensione `.m3g`. Il formato di un file `m3g` è molto complesso, come è possibile vedere dalla documentazione delle API. Per semplicità possiamo dire che, ogni scena in esso contenuta, viene strutturata alla stessa maniera vista nella prima parte di questo speciale.

Il processo di caricamento, infatti, crea un array in memoria dove ogni elemento rappresenta un nodo della struttura. Ovviamente il primo elemento dell'array non è altro che la radice dell'albero, ovvero il nodo di tipo `World`.

Nell'articolo di Maurizio Terzo, parte integrante di questo speciale, viene dettagliatamente spiegato il processo di creazione di un file `.m3g` tramite 3D Studio Max.

Procediamo con un esempio pratico e commentiamo il

codice presente nel **Listato1**. È di una semplicità estrema poiché il fulcro del processo di caricamento è rappresentato dalla linea

```
Object3D[] oggetti = Loader.load("esempio.m3g");
```

la quale si occupa di aprire il file indicato, caricarlo in memoria e ricostruire la struttura ad albero che rappresenta la scena. Nella scena proposta come esempio sono presenti solo 3 oggetti, ossia una telecamera, una fonte di luce ed la Mesh dell'autovettura. Ogni oggetto creato in memoria viene inserito nell'array `Object3D[]` di conseguenza, come già detto, nel primo elemento di tale array sarà presente il riferimento all'oggetto `World`.

```
world = (World)oggetti[0];
```

L'unica ulteriore operazione che si rende necessaria è quella di cambiare le impostazioni della telecamera attiva. Ciò avviene perché la telecamera deve adattare la correzione prospettica alle caratteristiche del dispositivo sul quale il programma è in esecuzione.

Il "Retained Mode" è un metodo di gestione delle scene 3D molto più comodo dell' "Immediate Mode", specialmente nel caso di mondi molto complessi

Per questo dopo aver recuperato il riferimento all'oggetto della telecamera attiva

```
Camera camera = world.getActiveCamera();
```

ricalcoliamo l'*aspect ratio*, ricordo che questo valore è il rapporto tra la larghezza e l'altezza dell'area di di rendering ed è necessario perché la API riesca a far mantenere le stesse proporzioni alle immagini, sottoposte a rendering, su display con dimensioni diverse.

```
float aspectRatio = (float) getWidth() / (float) getHeight();
camera.setPerspective(60.0f, aspectRatio, 1.0f, 1000.0f);
```

Dopo aver impostato i valori per la correzione prospettica, siamo pronti ad eseguire l'applicazione.

Il risultato è visibile in **Figura1**.

Come al solito si tratta di una demo statica, infatti, il codice si occupa solamente di eseguire una procedura di inizializzazione, tramite il metodo `Init()` della classe `GameCanvas`, e disegnare il mondo a video ad ogni passo tramite l'istruzione

```
g3d.render(world);
```

LISTATO 1

Il codice del metodo `Init()` necessario a caricare il file di esempio contenente tutta la struttura della scena e frame() per animare l'oggetto "Hello World!"

```
private World world;

public void Init() {

    try
    {
        // Carico il file m3d
        Object3D[] oggetti = Loader.load("esempio.m3g");

        // Ottengo il riferimento all'oggetto World
        world = (World)oggetti[0];

        // Recupero il riferimento alla telecamera attiva
        Camera camera = world.getActiveCamera();

        // Calcolo l'aspectRatio
        float aspectRatio = (float) getWidth() / (float) getHeight();

        // Attivo la correzione prospettica sulla telecamera
        camera.setPerspective(60.0f, aspectRatio, 1.0f, 1000.0f);
    }
    catch (Exception e)
    {
        // Errore
    }

    initDone = true;
}

void frame()
{
    // Qui viene inserito il codice che effettua
    // le operazioni periodiche di gestione dell'applicazione

    // Recupero l'oggetto Mesh con id 1
    Mesh obj = (Mesh) world.find(1);

    // Effettuo una rotazione sull'asse Y e Z
    obj.postRotate(-2.0f, 0.0f, 1.0f, 1.0f);
}
```

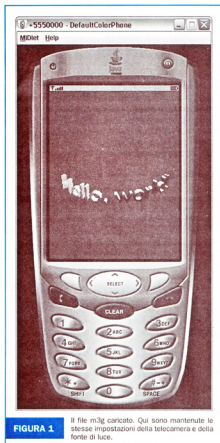


FIGURA 1

Per aggiungere un poco di animazione alla scena, come nell'altro esempio del resto, facciamo ruotare l'oggetto. È sufficiente eseguire due semplici operazioni, ossia, recuperare il riferimento all'oggetto Mesh che intendiamo ruotare

```
Mesh obj = (Mesh) world.find(1);
```

in questo caso come ID all'oggetto è stato assegnato il numero 1. Ottenuto il riferimento possiamo applicare la trasformazione voluta all'oggetto con

```
obj.postRotate(-2.0f, 0.0f, 1.0f, 1.0f);
```

ed ecco, come potete vedere in **Figura2**, l'oggetto ruoterà su se stesso al centro della scena.

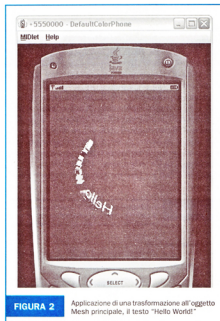


FIGURA 2

Conclusioni

Con questo articolo concludiamo, per il momento, la serie dedicata alla programmazione dei giochi sui cellulari. Probabilmente in futuro ci sarà spazio per ulteriori ampliamenti ed approfondimenti. Comunque spero che il percorso che abbiamo seguito sia stato utile per farvi entrare e muovere i primi passi in questo meraviglioso mondo che, secondo le nostre aspettative, ancora non ha mostrato i suoi veri frutti. Assisteremo a breve, infatti, ad una notevole crescita delle capacità dei dispositivi cellulari che porterà questi ad un progressivo "avvicinamento" se non una vera e propria fusione con le console portatili.

Per delucidazioni, consigli o domande di qualunque genere rimango a vostra disposizione. Continuate a contattarmi.

Bibliografia

- [1] Sun Corporation, "Mobile 3D Graphics API Specification", www.sun.com

Mauro Pesci

Lavora come libero professionista nel settore dello sviluppo del software, in particolare modo, si sta occupando della realizzazione di videogiochi commerciali per telefonini cellulari distribuiti, principalmente, tramite il portale di H3G.