

Java3D

Lezione 7

Texture



Cosa impareremo oggi?

- Come aggiungere texture alle geometrie
- Come caricare una texture con il *TextureLoader*
- Come personalizzare una texture con il *TextureAttributes*
- Come generare automaticamente le coordinate delle texture
- Come applicare texture multiple

Data la loro complessità alcuni argomenti saranno solo accennati

Le texture (1)

- L'aspetto degli oggetti nel mondo reale dipende principalmente dalla loro *texture* (tessitura)
- Per apprezzare il ruolo delle texture nell'aspetto degli oggetti basta pensare ad un tappeto
- Anche se tutte le fibre del tappeto hanno lo stesso colore, il tappeto non apparirà con un colore costante a causa delle interazioni della luce con le geometrie delle fibre

Le texture (2)

- Anche se Java3D è capace di modellare le geometrie delle singole fibre del tappeto, la memoria e la potenza di calcolo richieste per riuscire a modellare un oggetto del genere renderebbero tale modello inutilizzabile
- D'altra parte usare un rettangolo “piatto” di un solo colore come simulazione di un tappeto sicuramente non darebbe un bel risultato
- Le texture permettono di ottenere performance notevoli senza compromettere il risultato visivo

Il processo di texturing (1)

- Un tappeto è probabilmente l'esempio estremo in termini di complessità geometrica di un modello
- Ma non è certamente l'unico oggetto per il quale noi percepiamo una texture: mattoni, cemento, legno, prati, muri
- Proprio come per il tappeto, il costo per rappresentare tali superfici tramite primitive geometriche potrebbe diventare molto proibitivo

Il processo di texturing (2)

- Una possibile alternativa è modellare il tappeto come un poligono piatto con molti vertici ed assegnare ad ogni vertice un colore
- Se i vertici sono sufficientemente vicini si riesce ad ottenere una buona simulazione del tappeto
- Questo metodo richiede molta meno memoria del precedente, ma ancora troppa per i nostri scopi
- L'idea di rappresentare l'immagine di un oggetto su una superficie piatta è alla base del *processo di texturing*

Il processo di texturing (3)

- Il processo di texturing (in gergo *texture mapping*) è un modo per aggiungere “ricchezza visuale” ad una superficie senza aggiungere dettagli geometrici
- La ricchezza visuale è fornita da un'immagine (texture) che da i dettagli dell'aspetto della superficie dell'oggetto
- L'immagine è “mappata” sulla geometria dell'oggetto al momento del rendering (da cui il termine texture mapping)

Texturing – concetti base (1)

- In Java3D il texturing di poligoni è ottenuto:
 - creando un *Appearance*
 - caricando una texture
 - specificando le posizione della texture sulla geometria
 - impostando gli attributi di texturing
- Questi passaggi possono essere noiosi e ripetitivi
 - ci sono comunque le classi di utility per semplificare il processo e di solito non serve modificare le impostazioni di default degli attributi di texturing

Texturing – concetti base (2)

- A causa dell'estrema flessibilità del modello di texturing di Java3D, il numero di opzioni che l'API mette a disposizione potrebbe a prima vista confondere le idee
- In realtà il texturing non è particolarmente complesso, i passaggi chiave sono:
 - preparare la texture
 - caricare la texture
 - impostare la texture nell'*Appearance*
 - specificare le coordinate delle texture

Preparare la texture (1)

- Questo è un passaggio più artistico che tecnico e di norma viene fatto esternamente all'applicazione Java3D anzi la maggior parte delle texture sono preparate prima che il programma inizi
- Ci sono due fattori essenziali nella preparazione di una texture:
 - assicurarsi che l'immagine abbia dimensioni “accettabili”
 - assicurarsi che l'immagine sia salvata in un formato leggibile

Preparare la texture (2)

- Per problemi di efficienza Java3D richiede che le dimensioni delle texture siano una potenza di due (1, 2, 4, 8, 16, ...)
 - se non si rispetta tale regola si otterrà un'eccezione a runtime (dipende dalla piattaforma)
- È ovvio inoltre che la texture deve essere salvata in un formato leggibile da Java3D
 - per non correre rischi è meglio limitarsi a JPEG e GIF, formati letti correttamente dalla classe di utility *TextureLoader*

Caricare la texture

- Le *Texture* possono essere caricate via file o URL
- Il caricamento di una *Texture* può essere fatto con molte linee di codice oppure con due linee di codice che usano la classe *TextureLoader*.

```
TextureLoader TL=new TextureLoader("a.gif",this);  
ImageComponent2D image=TL.getImage();
```

- In ogni caso alla fine si ottiene un'immagine memorizzata in un oggetto *ImageComponent2D*

Impostare la texture nell'Appearance

- Per essere usata come una texture, l'immagine caricata deve essere convertita in un oggetto *Texture* e quindi inserita in un *Appearance*

- Il codice per fare ciò è molto semplice:

```
Texture2D tex=new Texture2D();
```

```
texture.setImage(0,image);
```

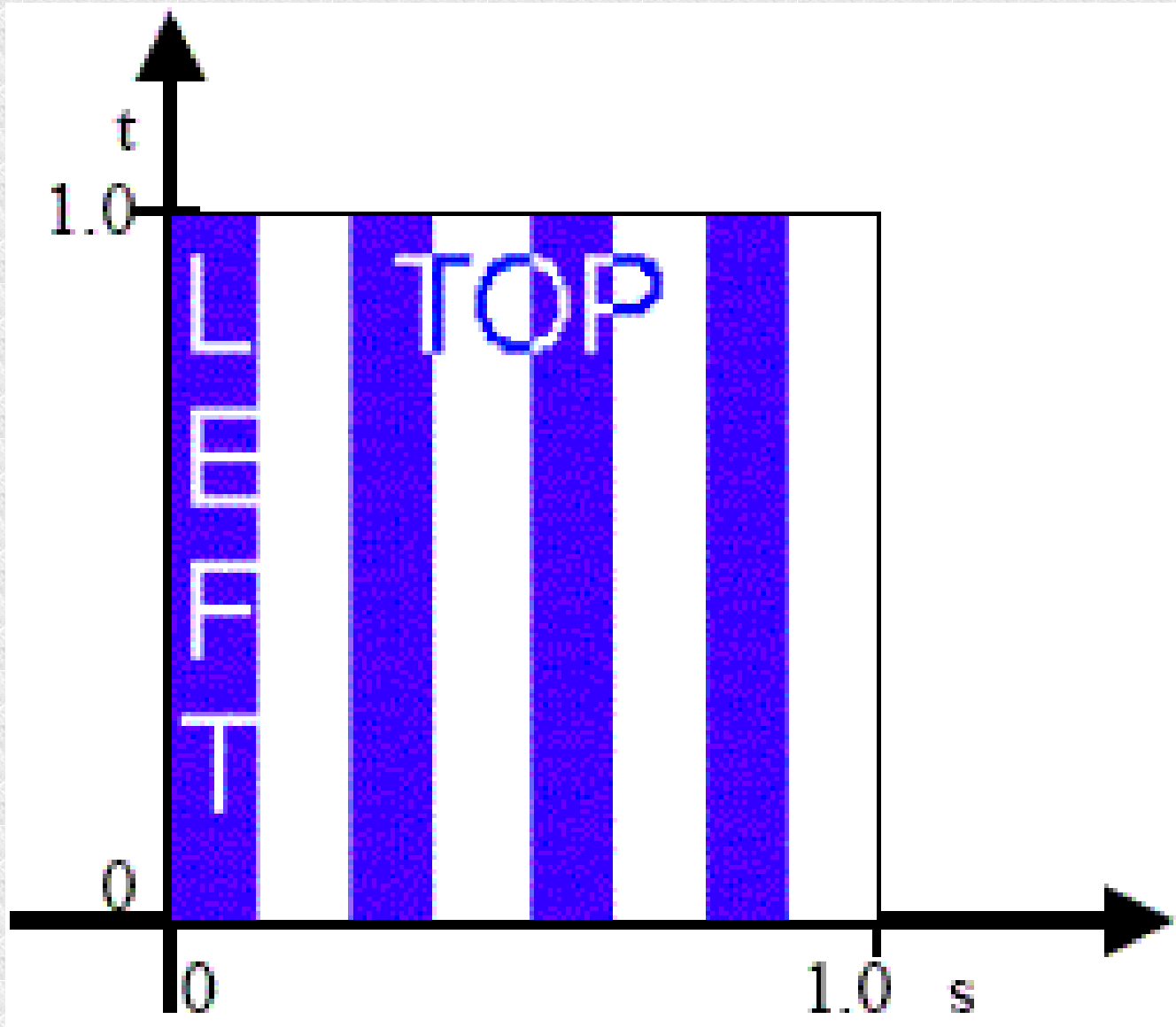
```
Appearance app=new Appearance();
```

```
app.setTexture(tex);
```

Specificare le coordinate delle texture (1)

- Infine bisogna specificare come posizionare la *Texture* sulla geometria attraverso le coordinate delle texture
- Il posizionamento della *Texture* è fatto in base ai vertici: ogni coordinata di texture specifica quale punto della texture deve essere mappata su un vertice
- Le coordinate delle texture sono specificate nelle dimensioni s e t (orizzontale e verticale) nel range $[0.0, 1.0]$

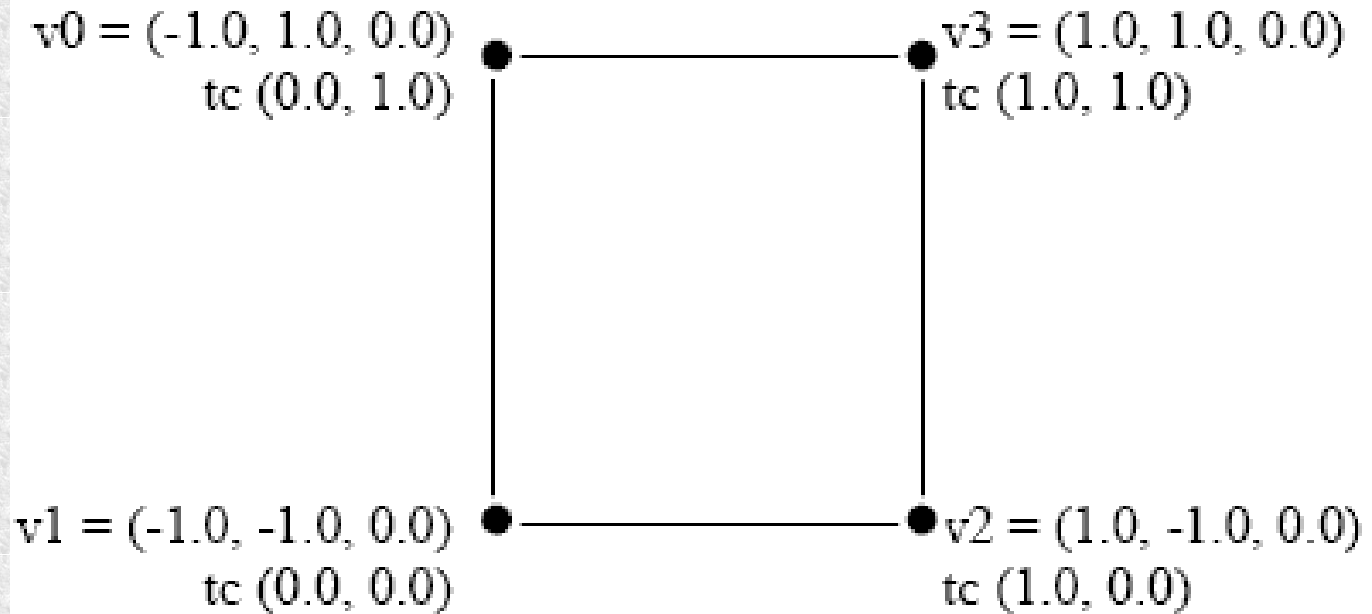
Specificare le coordinate delle texture (2)



Specificare le coordinate delle texture (3)

```
QuadArray plane=new QuadArray(4, QuadArray.COORDINATES|QuadArray.TEXTURE_COORDINATE_2);  
Point3f p=new Point3f();  
p.set(-1,1,0);  
plane.setCoordinate(0,p);  
p.set(-1,-1,0);  
plane.setCoordinate(1,p);  
p.set(1,-1,0);  
plane.setCoordinate(2,p);  
p.set(1,1,0);  
plane.setCoordinate(3,p);
```

```
TexCoord2f q=new TexCoord2f();  
q.set(0,1);  
plane.setTextureCoordinate(0,0,q);  
q.set(0,0);  
plane.setTextureCoordinate(0,1,q);  
q.set(1,0);  
plane.setTextureCoordinate(0,2,q);  
q.set(1,1);  
plane.setTextureCoordinate(0,3,q);
```



Specificare le coordinate delle texture (4)

- Dal codice appena riportato sorge naturale chiedersi a cosa serve il primo parametro del metodo *setTextureCoordinate()*
 - Java3D permette di definire le “multitexture”: la possibilità applicare più texture ad un oggetto
- Per permettere il multitexturing bisogna introdurre un “indice”: il primo parametro del metodo
- In molti casi il multitexturing può tornare molto utile: ad esempio simulare un'ombra applicando una texture sopra un oggetto già “texturizzato”

Specificare le coordinate delle texture (5)

Esempi

SimpleTextureApp_a1

SimpleTextureApp_a2

SimpleTextureApp_a3

Le opzioni di texturing

- Il texturing permette molto più della semplice impostazione delle coordinate:
 - *Texture2D* può avere diversi “comportamenti ai bordi” e filtri di mappatura
 - *TextureAttributes* permette di definire ulteriori attributi per le texture
 - le texture possono essere tridimensionali ed a diversi “livelli di dettaglio” (*MIPmaps*)
 - *TexCoordGeneration* permette la generazione automatica delle coordinate delle texture

I comportamenti ai bordi (1)

- Nei precedenti esempi le texture sono state mappate in modo tale da coprire interamente il piano
- Cosa succede quando le coordinate delle texture sono fuori dal range $[0.0, 1.0]$?
- Le impostazioni del comportamento ai bordi determinano come effettuare la mappatura in questi casi
- Le scelte sono duplicare (default) o prolungare (ripetere il colore ai bordi) la texture

I comportamenti ai bordi (2)

Esempio

BoundaryModeApp

I filtri di mappatura (1)

- Nel calcolo delle coordinate delle texture capita raramente che un pixel si mappa “esattamente” su un *texel* (texture element)
- Di solito un pixel copre più di un texel oppure è più piccolo di un solo texel, in entrambi i casi serve un filtro di zooming per mappare i pixel
- Ovviamente ci sono diverse opzioni per effettuare queste operazioni

I filtri di mappatura (2)

- Nel caso di un ingrandimento ogni texel apparirà su più pixel ed è possibile che il risultato finale sia una “tassellatura” dell'immagine
- Le possibili scelte per l'ingrandimento sono:
 - point sampling (seleziona il texel più vicino)
 - interpolazione (interpola i valori dei texel vicini)
- Come sempre la scelta tra i due dipende se è più importante la qualità o la velocità

I filtri di mappatura (3)

- Nel caso di una riduzione più texel appariranno su un pixel
- In questo caso il problema è che un pixel può avere un solo colore
- Anche in questo caso le scelte sono tra point sampling ed interpolazione e le differenze sono qualitative e di performance

I filtri di mappatura (4)

- Non sempre è chiaro quale filtro sarà usato: una texture potrebbe dover essere ingrandita in una direzione e ridotta in un'altra
- Non c'è modo per il programmatore di prendere decisioni in merito
- È Java3D che decide quale filtro usare per ottenere i risultati migliori

Un po' di esempi vari

Esempi

SimpleTextureApp_b1

SimpleTextureApp_b2

SimpleTextureApp_b3

La classe TextureAttributes

- La classe *TextureAttributes* permette ulteriori personalizzazioni del texturing
- È possibile impostare: modalità di texturing, blend color, modalità di correzione prospettica e texture map transform
- I valori di default sono: REPLACE, nero, FASTEST e NONE, e solitamente vanno bene nella maggior parte dei casi

Generazione automatica delle coordinate (1)

- Assegnare le coordinate delle texture per ogni vertice della geometria è un passo necessario e spesso noioso e complicato
- Le coordinate delle texture sono spesso calcolate con codice specifico per ogni oggetto
- Sarebbe meglio avere un sistema automatico: questo è proprio quello che tenta di fare la classe *TexCoordGeneration*

Generazione automatica delle coordinate (2)

- Per generare automaticamente le coordinate delle texture bisogna specificare alcuni parametri in un oggetto *TexCoordGeneration*
- Le coordinate delle texture sono calcolate a runtime in base ai parametri specificati
- I parametri da specificare sono: formato delle texture (2D o 3D), modalità di generazione (lineare o sferica)

Generazione automatica delle coordinate (3)

Esempio

TexCoordGenApp

La classe MIPmap

- Per capire a cosa serve una *MIPmap* consideriamo un oggetto texturizzato che si muove in una scena
 - a seconda che l'oggetto sia vicino o lontano dall'osservatore la sua texture potrebbe risultare troppo “sgranata” o eccessivamente dettagliata
- La classe *MIPmap* serve a risolvere questo problema proprio come la classe *DistanceLOD* risolve il problema delle geometrie
 - tramite la classe *MIPmap* è possibile definire più texture da utilizzare a diverse distanze