JAVA 3D - TEXTURES

Corso di Immagini e Multimedialità
Lezione 4.1: JAVA3D

cristian.virgili@uniud.it

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

JAVA 3D - TEXTURES

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

ESEMPIO TEXTURE (2)

TEXTURE (3)

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 (1)

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 (2)

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 dà 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 (1)

In Java3D il texturing di poligoni è ottenuto:

- creando un *Appearance*
- caricando una texture
- specificando le posizioni 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 (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 preparare 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)

Prima della versione 1.5 per problemi di efficienza Java3D richiedeva che le dimensioni delle texture fossero di una potenza di due (1, 2, 4, 8, 16, …) ed ancora oggi per alcune schede ci potrebbero essere problemi di manipolazione di immagini con dimensioni generiche (attributo ALLOW_NON_POWER_OF_TWO)

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);
```

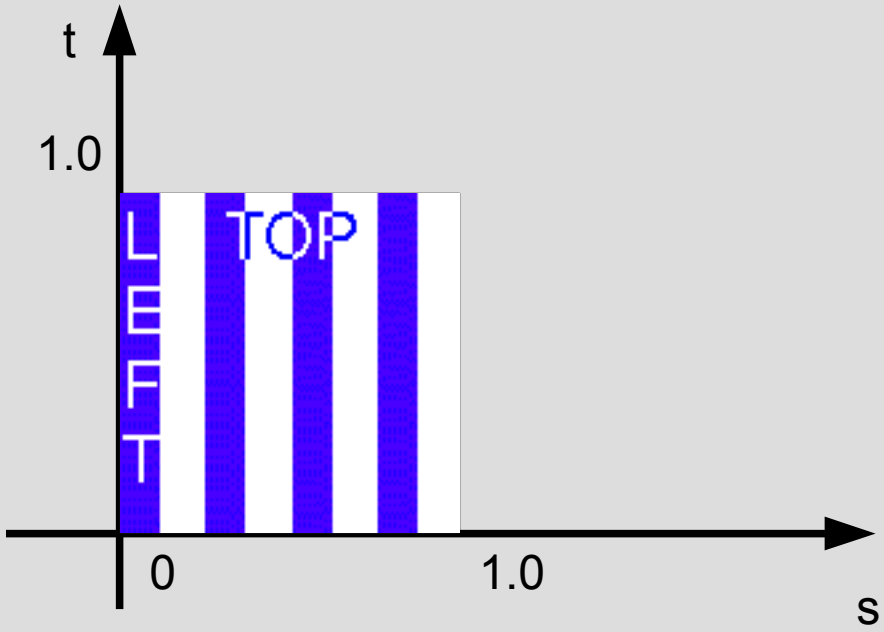
SPECIFICARE LE COORDINATE DELLA TEXTURE (1)

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 DELLA TEXTURE (2)

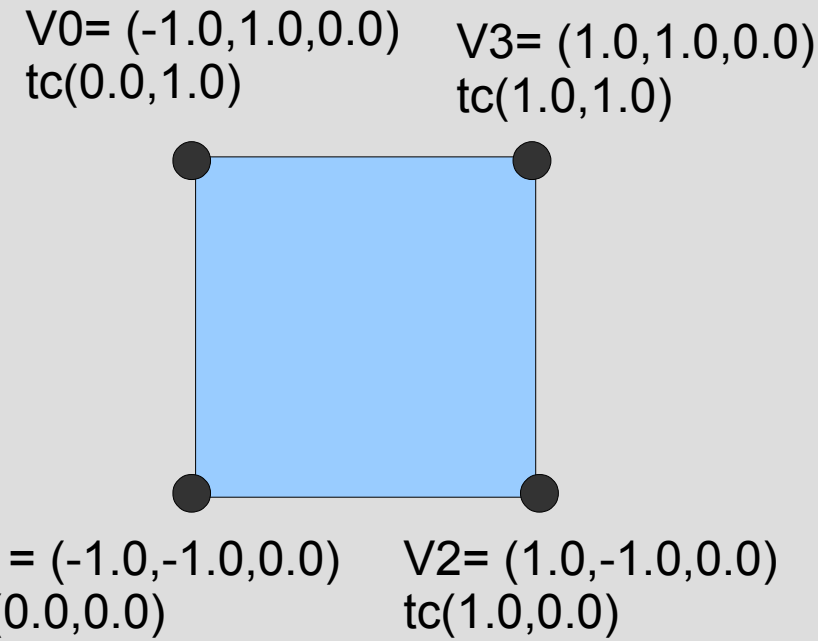


SPECIFICARE LE COORDINATE DELLA TEXTURE (3)

```
QuadArray flat=new QuadArray(4, QuadArray.COORDINATES|
QuadArray.TEXTURE_COORDINATE_2);

Point3f p=new Point3f();
p.set(-1,1,0);
flat.setCoordinate(0,p);
p.set(-1,-1,0); flat.setCoordinate(1,p);
p.set(1,-1,0); flat.setCoordinate(2,p);
p.set(1,1,0); flat.setCoordinate(3,p);

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



Esercizio

Creare un Piano (left bottom -1,-1, top right 1,1) a cui vengono associate le texture coordinates

```
//Crea un piano
    QuadArray plane=new QuadArray(4,GeometryArray.COORDINATES|
    GeometryArray.TEXTURE_COORDINATE_2);
    Point3f p=new Point3f(-1.0f,1.0f,0.0f);
    plane.setCoordinate(0,p);
    p.set(-1.0f,-1.0f,0.0f);
[...]
//Imposta le coordinate della texture
    TexCoord2f q=new TexCoord2f(0.0f,1.0f);
    plane.setTextureCoordinate(0,0,q);
[...]
 //Crea l'aspetto del piano
    Appearance appear=new Appearance();
    //Carica la texture
    TextureLoader loader=new TextureLoader("stripe.gif",this);
    ImageComponent2D image=loader.getImage();
//Crea la texture
    Texture2D texture=new Texture2D(Texture.BASE_LEVEL,
Texture.RGBA,image.getWidth(),image.getHeight());
    texture.setImage(0,image);
    appear.setTexture(texture);
[...]
```

SPECIFICARE LE COORDINATE DELLA 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”

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

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

ESERCIZIO : I COMPORTAMENTI AI BORDI

Verificare i diversi comportamenti ai bordi sul piano.

```
Appearance appear=new Appearance();

    //Carica la texture
    TextureLoader loader=new TextureLoader("stripe.gif",null);
    ImageComponent2D image=loader.getImage();

    //Crea la texture
    Texture2D texture=new Texture2D(Texture.BASE_LEVEL,
                                    Texture.RGBA,image.getWidth(),image.getHeight());
    texture.setImage(0,image);

    //Imposta il comportamento ai bordi
    texture.setBoundaryModeS(bS);//orizzontale
    texture.setBoundaryModeT(bT); //verticale

    //Imposta la texture nell'aspetto
    appear.setTexture(texture);
```

Per bS e bT usare le combinazioni:
- Texture.*WRAP*
- Texture.*CLAMP*

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

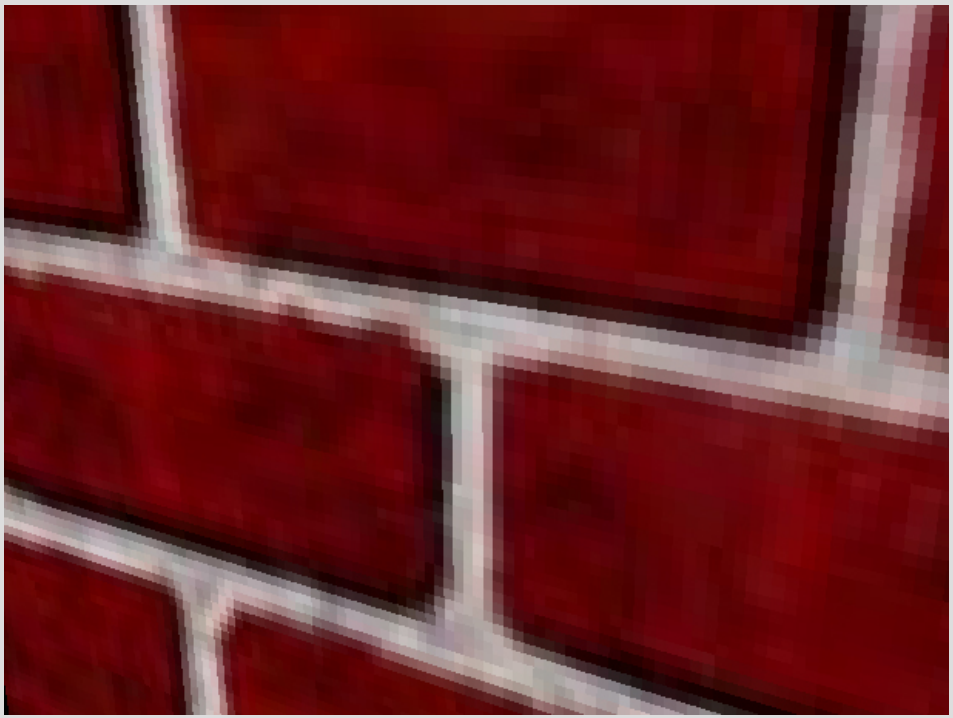
I FILTRI DI MAPPATURA (2)

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

ESMPIO: I FILTRI DI MAPPATURA



Non interpolato

Interpolato

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

TEXTURE ATTRIBUTES

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

Si consiglia la consultazione dei javadoc per i dettagli

http://download.java.net/media/java3d/javadoc/1.5.0/javax/media/j3d/TextureAttributes.html

COSTRUTTORE TEXTURE ATTRIBUTES

- *TextureAttributes()*
- *TextureAttributes(int textureMode, Transform3D transform, Color4f textureBlendColor, int perspCorrectionMode)*

Metodi principali:
- void setColor(Color3f color)

Il default corrisponde a TextureAttributes(TextureAttributes.REPLACE, new Transform3D(), new Color4f(), TextureAttributes.NICEST).

I valori ammessi come textureMode, sono: MODULATE, DECAL, BLEND, REPLACE e COMBINE. *perspCorrectionMode* può essere NICEST o FASTEST.

http://download.java.net/media/java3d/javadoc/1.5.0/javax/media/j3d/TextureAttributes.html

ESEMPIO DI TEXTATTRIBUTES

```
TextureAttributes  textureAttributes  = new  TextureAttributes ( ) ;
// Impostazioni per fondere il colore dell'oggetto con la texture.
textureAttributes.setTextureMode( TextureAttributes .COMBINE) ;
textureAttributes.setCombineRgbSource(0,
                                TextureAttributes.COMBINE_TEXTURE_COLOR);
textureAttributes.setCombineRgbSource(1,
                                TextureAttributes.COMBINE_OBJECT_COLOR);
//Correzione prospettica .
textureAttributes.setPerspectiveCorrectionMode(TextureAttributes.NICEST);
appearance.setTextureAttributes(textureAttributes) ;
```

TexCoordGeneration (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*

TexCoordGeneration (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)

ESERCIZIO

Creare un oggetto sphere utilizzando la generazione automatica delle texture coords ed applicare la texture earth.jpg

```
Appearance appearance = new Appearance ( ) ;
//Caricamento della texture da file.
TextureLoader textureLoader = new TextureLoader("earth.jpg", null);

// Inizializzazione dell 'oggetto Texture.
Texture texture = textureLoader.getTexture();
// Impostazione dell'aspetto .
appearance.setTexture(texture);
// Creazione di una primitiva completa di
// coordinate per la texture.
Sphere earth = new Sphere( 1.0f, Primitive.GENERATE_TEXTURE_COORDS, appearance);
```

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

- tramite la classe MIPmap è possibile definire più texture da utilizzare a diverse distanze

```
// Caricamento della texture con generazione delle MIPMAP.
Texture texture = new TextureLoader("earth.png" ,
TextureLoader.GENERATE_MIPMAP, this).getTexture() ;
```

BACKGROUND (1)

Per default lo sfondo di un mondo Java3D è nero, ma è possibile specificare altri tipi di sfondo

Java3D da la possibilità di specificare come sfondo: un colore, un'immagine, una geometria o una combinazione di questi

Se si specifica un'immagine allora l'eventuale colore specificato non sarà considerato

Se si specifica una geometria allora sarà disegnata sopra l'immagine o il colore specificati

BACKGROUND (2)

La parte difficile nella creazione di uno sfondo potrebbe esse-re la definizione della geometria

Una geometria deve essere specificata come punti
- se la geometria è definita come un *PointArray* (che potrebbe rappresentare stelle lontane anni luce) o un *TriangleArray* (che potrebbe rappresentare montagne sullo sfondo) tutte le coordinate devo essere specificate sulla sfera unitaria

BACKGROUND (3)

Ogni *Background* ha un *bound* che permette di specificare sfondi differenti per differenti regioni del mondo virtuale

Se più *Background* sono attivi allora sarà usato il Background più “vicino all'occhio”

Se nessun *Background* è attivo allora lo sfondo sarà colorato di nero

ESEMPIO BACKGROUND

Aggiungere uno sfondo (stars.jpg) alla terra dell'esempio precedente

```
TextureLoader myLoader = new TextureLoader( "stars.jpg", this );
ImageComponent2D myImage = myLoader.getImage( );
Background myBack = new Background( );
myBack.setImage( myImage );
myBack.setImageScaleMode(Background.SCALE_FIT_MAX);
BoundingSphere myBounds = new BoundingSphere(new Point3d( ), 1000.0 );
myBack.setApplicationBounds( myBounds );
objRoot.addChild(myBack);
```

ESERCIZIO 4.1
RELAZIONE FINALE

Alla “Terra” aggiungere una luce direzionale per creare l'effetto dell'illuminazione solare

ESERCIZIO 4.2
RELAZIONE FINALE

- Applicare una texture agli elementi della colonna dorica creata Precedentemente.

- Riprodurre la facciata del tempio di Poseidone di Pæstum.

ESERCIZIO