Laboratorio di Sistemi Interattivi

Paolo Bottoni

Dipartimento di Informatica

Sapienza Università di Roma

Lezione 12: Note di Java 3D

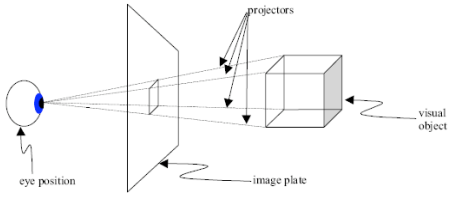
Argomenti

- Grafi della scena
  - Contenuto e appearance
  - Nodi di trasformazione
- Geometria
- Comportamento
- Resa

Concetti fondamentali

- Gerarchia di classi Java che servono da interfaccia a sistema di grafica 3D e resa sonora
- Costrutti di alto livello per creare e manipolare oggetti geometrici 3D.
- Oggetti risiedono in universo virtuale
- Resa dell'universo virtuale in parallelo
- Programma Java3D crea istanze di oggetti 3D e le posiziona in grafo della scena.
- Grafo della scena strutturato ad albero
  - specifica il contenuto dell'universo e come deve essere reso.

Posizioni relative



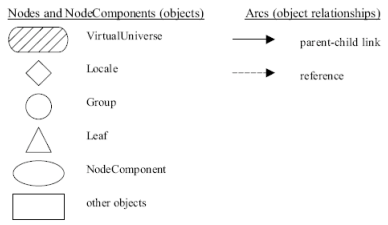
Grafo della scena

- Oggetti Node formano albero
  - Oggetto radice Locale
  - Un solo cammino da radice a ogni foglia
- Archi riferimento legano nodi e NodeComponents
- Informazione di stato include:
  - posizione, orientamento e taglia di oggetti visivi
- Attributi visivi di oggetto dipendono solo dal cammino dalla radice

Resa grafica

- Ordine di resa determinato dal motore grafico per motivi di efficienza
- Programmatore non ha controllo
- Grafi scorretti possono essere compilati, ma non resi graficamente

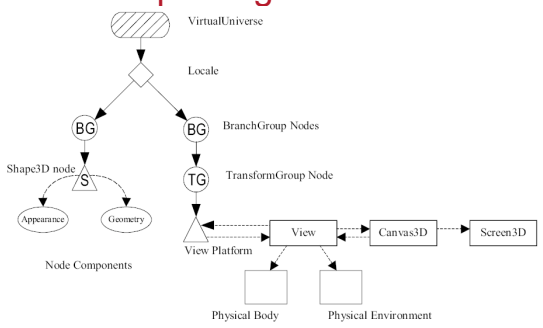
Alfabeto per grafi di scena



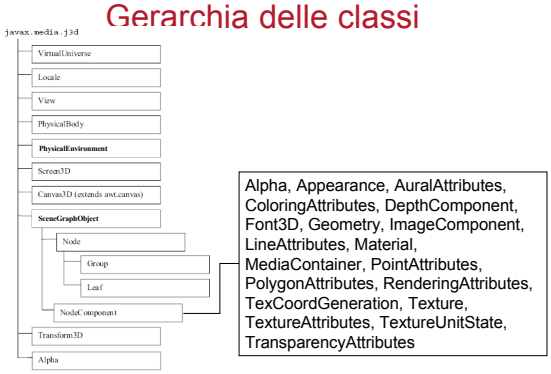
Vincoli sul linguaggio

- Grafo di scena ha un singolo universo virtuale.
- Programma può avere più universi, ma non possono comunicare tra loro.
- Due categorie di sottografi: *view branch graph* e *content branch graph*.

Esempio di grafo di scena



Gerarchia delle classi



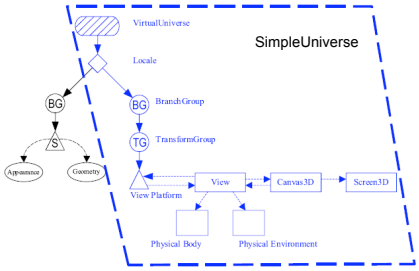
Ricetta per costruire programmi 3D

1. Creare oggetto Canvas3D
2. Creare oggetto VirtualUniverse
3. Creare oggetto Locale e attaccarlo al VirtualUniverse
4. Costruire grafo BranchView
   a) Creare oggetto View
   b) Creare oggetto ViewPlatform
   c) Creare oggetto PhysicalBody
   d) Creare oggetto PhysicalEnvironment
   e) Attaccare oggetti ViewPlatform , PhysicalBody, PhysicalEnvironment e Canvas3D all'oggetto View
5. Costruire i contenuti dei grafi BranchView
6. Compilare i rami dei grafi BranchView
7. Inserire i sottografi nell'oggetto Locale

Semplificazione

- Uso di SimpleUniverse
- Non c'è bisogno di creare grafo branch di View
- Uso di classi Viewer e ViewingPlatform

Esempio di semplificazione



Ricetta semplificata

1. Creare oggetto Canvas3D Object
2. Creare oggetto SimpleUniverse che si riferisca a oggetto Canvas3D
   a) Adattare oggetto SimpleUniverse
3. Costruire ramo dei contenuti
4. Compilare contenuto del grafo di branch
5. Inserire grafo di branch nel Locale del SimpleUniverse

Ramo di view

- Oggetto SimpleUniverse crea grafo di ramo di view per universo virtuale
- Grafo contiene un *image plate*, rettangolo concettuale su cui è proiettato il contenuto per formare l'immagine da rendere.
- Ruolo svolto da oggetto Canvas3D.
- Oggetti visivi proiettati su image plate.
- Occhio dietro il plate. Oggetti resi sono quelli di fronte all'image plate.
- Centro dell'image plate a (0,0,0).

Resa grafica

Renderer Java3D inizia un ciclo infinito quando ramo con istanza di View diventa vivo in universo virtuale

```
while(true) {
        Process input
        If (request to exit) break
        Perform Behaviors
        Traverse scene graph and render visual objects
}
Cleanup and exit
```


Esempio di creazione

```
public class HelloJava3Da extends Applet {

   public HelloJava3Da() {
        setLayout(new BorderLayout());
        GraphicsConfiguration config =
   SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas3D = new Canvas3D(config);
        add("Center", canvas3D);
        BranchGroup scene = createSceneGraph();
        scene.compile();
```


Esempio di creazione

```
// SimpleUniverse is a Convenience Utility class
        SimpleUniverse simpleU = new
   SimpleUniverse(canvas3D);
// Moves the ViewPlatform back a bit so objects in the
   scene can be viewed.

   simpleU.getViewingPlatform().setNominalViewingTran
   sform();
        simpleU.addBranchGraph(scene);
} // end of HelloJava3Da (constructor)
```

Creazione del ramo di contenuto

```
   public BranchGroup createSceneGraph() {
// Create the root of the branch graph
        BranchGroup objRoot = new BranchGroup();
// Create a simple shape leaf node, add it to the scene graph.
// ColorCube is a Convenience Utility class
        objRoot.addChild(new ColorCube(0.4));
        return objRoot;
   } // end of createSceneGraph method
} // end of class HelloJava3Da
```

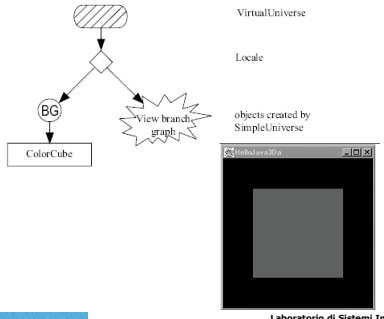

Oggetti 3D predefiniti

- Node
  - Leaf
    - Shape3D
      - ColorCube un colore per ogni faccia
  - Group
    - Primitive
      - Box length, width, height
      - Cone radius, height
      - Cylinder radius, height
      - Sphere radius

Definizione dell'apparenza

- NodeComponent
  - Appearance
- Definisce lo stato di rendering
  - Coloring attributes
  - Line attributes.
  - Point attributes
  - Polygon attributes
  - Rendering attributes
  - Transparency attributes.
  - Material
  - Texture
  - Texture attributes
  - Texture coordinate generation
  - Texture unit state // array per attributi di ogni texture unit

Grafo di scena e immagine



Matrici di trasformazione

```
[ m00 m01 m02 m03 ] [ x ]       [ x' ]
[ m10 m11 m12 m13 ] [ y ]   =   [ y' ]
[ m20 m21 m22 m23 ] [ z ]       [ z' ]
[ m30 m31 32 m33 ] [ w ]        [ w' ]

x' = m00 . x    +   m01 . y   +   m02 . z + m03 . w
y' = m10 . x    +   m11 . y   +   m12 . z + m13 . w
z'  = m20 . x   +   m21 . y   +   m22 . z + m23 . w
w' = m30 . x    +   m31 . y   +   m32 . z + m33 . w
```


Definizioni delle trasformazioni

- ZERO – matrice di tutti zeri
- IDENTITY – 1 sulla diagonale e 0 altrove
- SCALE – matrice di trasformazione di scala. Non ci sono elementi di traslazione o rotazione.
- ORTHOGONAL – vettori di righe ortogonali con scala unitaria
- RIGID – parte superiore di matrice 3 X 3 ortogonale, componente di traslazione. Scala unitaria.
- CONGRUENT – preserva angoli e lunghezze. Può traslare, ruotare e riflettere rispetto a un asse. Può scalare di una quantità uguale in tutte le direzioni.
- AFFINE – può traslare, ruotare, riflettere e scalare anisotropicamente, e flettere. Linee rimangono diritte, linee parallele restano parallele, angolo fra linee può cambiare
- NEGATIVE_DETERMINANT – Matrice ortogonale con determinante negativo definisce rotazioni e riflessioni, positivo solo rotazione

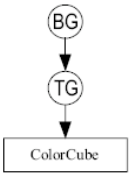
Oggetti di trasformazione

- **Transform3D()**
- **Transform3D**(double[] matrix)
- **Transform3D**(GMatrix m1)
- **Transform3D**(Matrix3d m1, Vector3d t1, double s)
- **Transform3D**(Matrix3f m1, Vector3d t1, double s)
- **Transform3D**(Matrix4d m1)
- **Transform3D**(Quat4d q1, Vector3d t1, double s)
- **Transform3D**(Transform3D t1)

Definizione delle trasformazioni

```
public BranchGroup createSceneGraph() {
// Create the root of the branch graph
    BranchGroup objRoot = new BranchGroup();
// rotate object has composite transformation matrix
    Transform3D rotate = new Transform3D();
    rotate.rotX(Math.PI/4.0d);
    TransformGroup objRotate = new TransformGroup(rotate);
    objRotate.addChild(new ColorCube(0.4));
    objRoot.addChild(objRotate);
    return objRoot;
} // end of createSceneGraph method
```

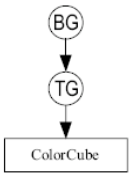

Grafo risultante

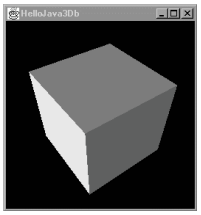


Composizione di trasformazioni

```
public BranchGroup createSceneGraph() {
// Create the root of the branch graph
    BranchGroup objRoot = new BranchGroup();
// rotate object has composite transformation matrix
    Transform3D rotate = new Transform3D();
    Transform3D tempRotate = new Transform3D();
    rotate.rotX(Math.PI/4.0d);
    tempRotate.rotY(Math.PI/5.0d);
    rotate.mul(tempRotate);
    TransformGroup objRotate = new TransformGroup(rotate);
    objRotate.addChild(new ColorCube(0.4));
    objRoot.addChild(objRotate);
    return objRoot;
}
```


Grafo e immagine





Compilazione e dinamica

- Grafo compilato a una rappresentazione interna.
- Fissa la resa da parte di Renderer 3D
- Per permettere modifiche del grafo dopo la compilazione, occorre definire le capacità di accesso.
- Modifica dei bit della capacità
- Metodi setCapability(int bit) e clearCapability(int bit)

Comportamenti

- Possono definire animazione (trascorrere del tempo) o interazione (eventi utente)
- Comportamenti definiti da oggetti legati da archi reference ai Node
- Comportamenti attivi in una scheduling region
- Si applicano solo se l'activation volume della View Platform interseca la scheduling region per il behavior di un oggetto

Creazione di interpolazioni

Creare un oggetto TransformGroup
Attivare la ALLOW_TRANSFORM_WRITE capability
Creare un oggetto Alpha
Specificare parametri temporali per l'oggetto
Creare un oggetto Interpolator
  Creare riferimenti con Alpha e TransformGroup
  Adattare i parametri di comportamento
Specificare una scheduling region
  Assegnare la regione al comportamento
Assgnare comportamento a TransformGroup

Esermpio di creazione

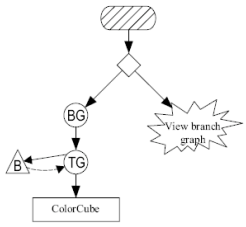
```
public BranchGroup createSceneGraph() {
// Create the root of the branch graph
    BranchGroup objRoot = new BranchGroup();
    TransformGroup objSpin = new TransformGroup();

    objSpin.setCapability(TransformGroup.ALLOW_TR
    ANSFORM_WRITE);
    objRoot.addChild(objSpin);
    objSpin.addChild(new ColorCube(0.4));
// create time varying function to drive the animation
    Alpha rotationAlpha = new Alpha(-1, 4000);
```

Esermpio di creazione

```
// Create a new Behavior object and add it into the
   scene graph.
   RotationInterpolator rotator = new
   RotationInterpolator(rotationAlpha, objSpin);
   BoundingSphere bounds = new BoundingSphere();
   rotator.setSchedulingBounds(bounds);
   objSpin.addChild(rotator);
   return objRoot;
} // end of createSceneGraph method
```


Grafo risultante



Capabilities per Node

static int **ALLOW_AUTO_COMPUTE_BOUNDS_READ**
static int **ALLOW_AUTO_COMPUTE_BOUNDS_WRITE**
static int **ALLOW_BOUNDS_READ**
static int **ALLOW_BOUNDS_WRITE**
static int **ALLOW_COLLIDABLE_READ**
static int **ALLOW_COLLIDABLE_WRITE**
static int **ALLOW_LOCAL_TO_VWORLD_READ**
static int **ALLOW_LOCALE_READ**
static int **ALLOW_PARENT_READ**
static int **ALLOW_PICKABLE_READ**
static int **ALLOW_PICKABLE_WRITE**
static int **ENABLE_COLLISION_REPORTING**
static int **ENABLE_PICK_REPORTING**

Capabilities per Group e TransformGroup

static int ALLOW_CHILDREN_EXTEND
static int ALLOW_CHILDREN_READ
static int ALLOW_CHILDREN_WRITE
static int ALLOW_COLLISION_BOUNDS_READ
static int ALLOW_COLLISION_BOUNDS_WRITE

static int ALLOW_TRANSFORM_READ
static intALLOW_TRANSFORM_WRITE

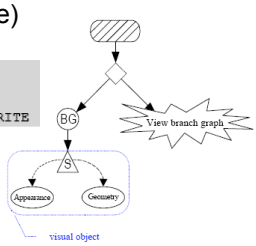
Definizione della geometria

- Basata su primitive o su definizione di shape
- Primitive definisce caratteristiche comuni alla gestione delle forme, e.g. numero di poligoni per rendere una superficie
- Geometria non definisce colore, occorre definire Appearance.

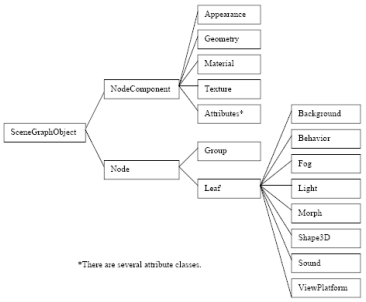
Oggetti Shape3D

- Oggetti visivi definiti da Geometry e Appearance (opzionale)

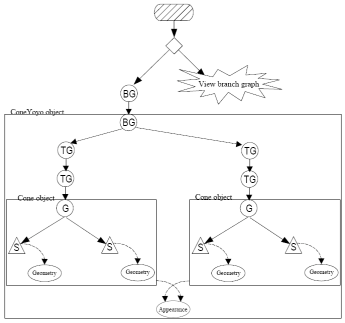
```
ALLOW_GEOMETRY_READ | WRITE
ALLOW_APPEARANCE_READ | WRITE
ALLOW_COLLISION_BOUNDS_READ | WRITE
```



Gerarchia



Composizione di geometrie



Classi matematiche

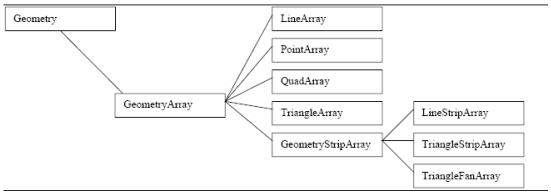
- Da vecmath
- Basate su Tuple* con 2, 3, o 4 coordinate
  - Point definisce singolo punto
  - Vector definisce coppia di punti
  - Color definisce posizione in spazio di colori
  - TexCoord coordinate relative a una texture

Sottoclassi di Geometry

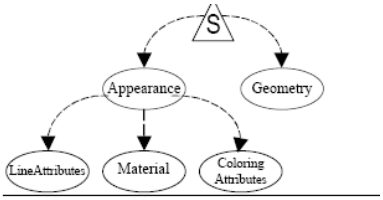
- Tre categorie
  - Geometria basata su vertici non indicizzata
    - Vertici possono essere usati una sola volta per la resa
  - Geometria basata su vertici indicizzata
    - Vertici possono essere riusati nella resa
- Altri oggetti visivi,
  - Raster,
  - Text3D
  - CompressedGeometry)

Array

- Permettono di elencare punti
- Vengono associati a oggetti 3D.



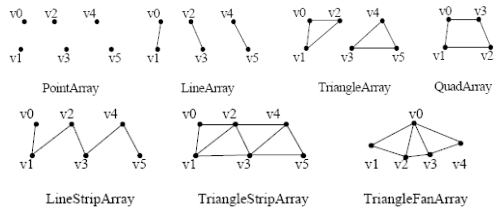
Appearance



Oggetti di utilità

- Primitive di GeometryInfo
  - POLYGON_ARRAY
  - QUAD_ARRAY
  - TRIANGLE_ARRAY
  - TRIANGLE_FAN_ARRAY
  - TRIANGLE_STRIP_ARRAY
- Triangulator
  - void triangulate(GeometryInfo gi) converte oggetti POLYGON_ARRAY in TRIANGLE_ARRAY
- Stripifier
  - void stripify(GeometryInfo gi) converte la geometria in un array di Triangle Strips.

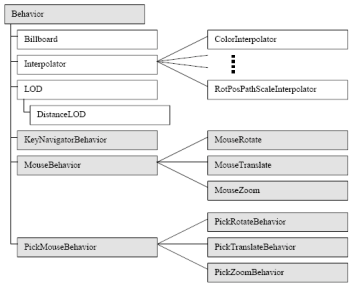
Array



Interazione

- Definita da comportamenti
  - Possono influenzare TransformGroup, Geometry, SceneGraph o View
  - Causati da utente, collisioni, tempo (animazione), collocazione del punto di vista

Gerarchia di comportamenti



Definizione di comportamenti

Scrivere un costruttore
  Mantenere un riferimento all'oggetto da cambiare
Ridefinire public void initialization()
Specificare criteri iniziali di attivazione (trigger)
Ridefinire public void processStimulus()
Decodificare la condizione di trigger (wakeupOn)
Agire in base alla condizione
Reimpostare il trigger come appropriato

Esempio

```
public class SimpleBehaviorApp extends Applet {
  public class SimpleBehavior extends Behavior {
   private TransformGroup targetTG;
   private Transform3D rotation = new
   Transform3D();
   private double angle = 0.0;
   SimpleBehavior(TransformGroup targetTG) {
     this.targetTG = targetTG;
   }

   public void initialize(){
     this.wakeupOn(new
     WakeupOnAWTEvent(KeyEvent.KEY_PRESSE
```

Esempio II

```
public void processStimulus(Enumeration criteria) {
    angle += 0.1;
    rotation.rotY(angle);
    targetTG.setTransform(rotation);
    this.wakeupOn(new
        WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
  }
} // end of class SimpleBehavior
```


Creiteri di wakeup

WakeUpCriterion // un solo criterio possibile per Behavior

- WakeupOnActivation // prima volta volume di attivazione interseca View
- WakeupOnAWTEvent // su occorrenza evento
- WakeupOnBehaviorPost // specifico Behavior pubblica specifico evento
- WakeupOnCollisionEntry // specifico oggetto collide con altro oggetto
- WakeupOnCollisionExit // specifico oggetto non collide più
- WakeupOnCollisionMovement // specifico oggetto si muove mentre collide
- WakeupOnDeactivation // prima volta volume non interseca più View
- WakeupOnElapsedFrames // dopo esecuzione numero di frame

Creiteri di wakeup

- WakeupOnElapsedTime // dopo intervallo
- WakeupOnSensorEntry // intersezione volume sensore (InputDevice)
- WakeupOnSensorExit // fine intersezione volume sensore
- WakeupOnTransformChange // se cambia trasformazione per un gruppo
- WakeupOnViewPlatformEntry // intersezione con ViewPlatform
- WakeupOnViewPlatformExit // fine intersezione con ViewPlatform