

Présentation Java 3D

Jean-Marc Farinone

`farinone@cnam.fr`

(Maître de Conférences CNAM)

Jean-Marc Le Gallic

`legallic@ensg.ign.fr`

Institut Géographique National

École Nationale des Sciences Géographiques

Alexandre Topol

`topol@cnam.fr`

(Maître de Conférences CNAM)

RSX205

Moteur 3D

La programmation d'un univers 3D implique sa transformation en univers 2D (écran)!



Démonstrations (IE)

Des exemples de programmes 3D

Voir de bons cours faits par mes collègues ;-) mais ...

La 3D utilise des mathématiques (géométrie 3D, quaternion, etc.) intéressantes et non triviales. Ainsi que des notions informatiques (double buffering, pipe line 3D, etc.) de même qualité ;-)

Il existe des demos 3D en Java avec leurs sources données avec le J2SE sous :

`REP_INSTALL_J2SE/demo/applets/`

voir les répertoires `WireFrame`, `MoleculeViewer`

Voir les sources (et les mathématiques associées) dans le code des applets.

Des demos ? Bon d'accord (filaires et molécules dans navigateur)

Qu'est-ce-que Java 3D ?

Java 3D est une bibliothèque de classes destinée à créer des scènes 3D (utilisation de formes complexes, d'éclairages, de textures, d'animations, de sons ...)

Le **package J3D** est une extension du langage Java

J3D utilise les bibliothèques graphiques 3D existantes (les standards **OpenGL**, **DirectX** ...) et peut importer des graphes de scènes **VRML**

Indépendant du matériel

Démarrer en Java 3D

- 1. Prérequis**
- 2. Téléchargement**
- 3. Installation**
- 4. Architecture des packages**
- 5. Performances**

Prérequis

Connaître la programmation en Java

Avoir des connaissances en programmation 3D n'est pas obligatoire mais des notions en géométrie 3D facilitent l'apprentissage

Bibliographie

http://java.sun.com/products/java-media/3D/forDevelopers/J3D_1_2_API/j3dguide/index.html
1 : les spécifications de Java 3D

<http://wiki.java.net/bin/view/Javadesktop/Java3DFAQ> :
FAQ Java 3D

http://java.sun.com/products/java-media/3D/forDevelopers/J3D_1_2_API/j3dapi/index.html :
documentation Java 3D en ligne

<http://java.sun.com/developer/onlineTraining/java3d/index.html>: le tutorial Java 3D

Certaines parties du site ne sont accessibles que si on est inscrit : le faire c'est gratuit.

Téléchargement

“The Java 3D API is now a community source project developed on java.net”. Voir à

<https://java3d.dev.java.net/>

Java 3D est actuellement disponible sous Windows, Solaris, Mac OS X et Linux.

Version actuelle (janvier 2007) de Java 3D : 1.5.0

URL de chargement :

<https://java3d.dev.java.net/binary-builds.html>

Prendre `java3d-1_5_0-windows-i586.exe`

L'installation détecte le J2SE (1.5 au moins) et range les diverses parties dans des répertoires fixes.

Installation

Comme Java 3D est une extension de J2SE, il faut avoir installé J2SE.

Des DLL sont installées dans le répertoire **bin** du JRE

Les fichiers **vecmath.jar**, **j3dcore.jar**, **j3daudio.jar** et **j3dutils.jar** dans le répertoire **lib/ext** du JRE

Pour récupérer les exemples, aller à <https://j3d-examples.dev.java.net/> (s'inscrire à la communauté java.net)

il faut utiliser CVS. Voir à

<https://j3d-examples.dev.java.net/servlets/ProjectSource>

Après un peu de bidouille CVS, on récupère les exemples Java 3D sur son disque

Architecture logicielle

Notre programme

API Java proposant des routines 3D

Bibliothèques 3D de base (OpenGL, Direct3D)

Hardware : CPU + entrées (souris 3D)
+ sorties écran carte graphique avancée

Performances

La 3D est grande consommatrice de calculs. Pour accélérer ses traitements, Java 3D utilise les optimisations offertes par le système

Pour **OpenGL** : appels à des méthodes natives de la **DLL Java 3D/OpenGL**

Ces méthodes appellent des fonctions de la **DLL OpenGL** du système qui utilise l'accélérateur graphique disponible sur la carte graphique

La puissance de traitement de Java 3D dépend donc de la puissance du microprocesseur mais aussi de celle de la carte graphique utilisée

LES PRINCIPES 3D

- 1. Construction d'un « univers » 3D**
- 2. Le repère 3D**
- 3. Les transformations 3D**
- 4. Conseils et optimisations**

De quoi a-t-on besoin ?

Un ensemble de formes **géométriques** (simples ou complexes) et un ensemble de **transformations** appliquées aux formes (translation, rotation ...)

→ **La scène 3D**

Une zone de l'écran qui permet de visualiser la scène 3D

→ **Canvas3D** (`javax.media.j3d.Canvas3D`)

Un objet qui permet de relier l'instance de Canvas3D à la scène 3D

→ **SimpleUniverse**

La classe **SimpleUniverse** est une classe qui permet de créer des instances par défaut des classes **VirtualUniverse**, **Locale**, **ViewPlatform**, **View**, **PhysicalBody** et **PhysicalEnvironment**, nécessaires à Java 3D

Représentation d'une scène 3D : arbre

Une scène est représentée par un graphe de type arbre comprenant des nœuds et des feuilles

Deux types de nœuds

BranchGroup

Nœud représentant la racine d'un sous-arbre comprenant d'autres nœuds (la racine d'une scène est une instance de BranchGroup)

TransformGroup

Nœud permettant de gérer un ensemble de transformations appliquées à la racine d'un arbre donc aux objets 3D composant cet arbre.

Les feuilles

représentent les objets : objets graphiques, sons, lumières ...

Pour les objets graphiques il faut décrire la forme (**Geometry**) et l'apparence (**Appearance** - couleur, texture, etc ...)

Arborescence des classes Java3D

Les classes représentant un groupe dérivent des classes **javax.media.j3d.Node** et **javax.media.j3d.Group**

Les classes représentant une feuille dérivent de la classe **javax.media.j3d.Leaf**

Arborescence des classes fondamentales :

java.lang.Object

javax.media.j3d.SceneGraphObject

javax.media.j3d.Node

javax.media.j3d.Group

javax.media.j3d.BranchGroup

javax.media.j3d.TransformGroup

javax.media.j3d.Leaf

javax.media.j3d.Background

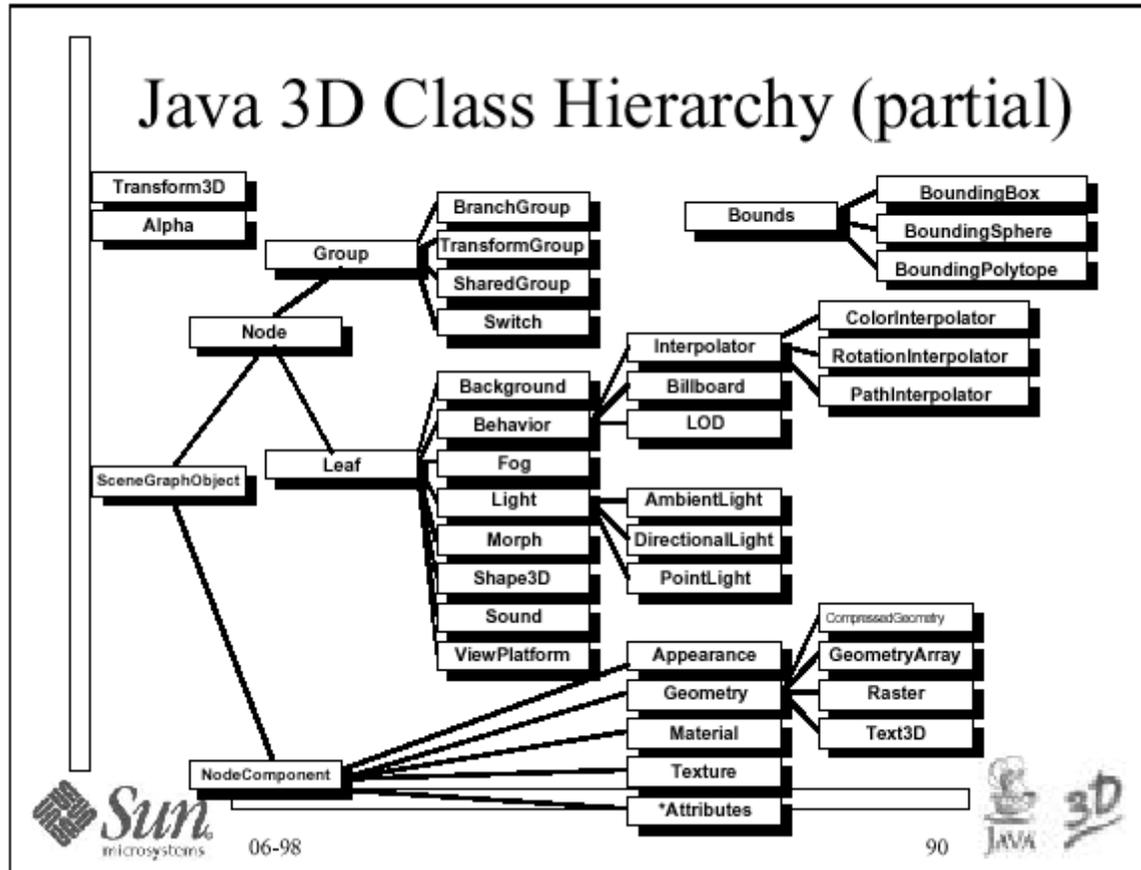
javax.media.j3d.Behavior

javax.media.j3d.Light

javax.media.j3d.Morph

javax.media.j3d.Sound

Les classes de Java 3D



Premier programme Java 3D

On va construire une applet présentant une scène 3D : le cube fondamental de Java 3D. Cette applet construit une instance de la classe `com.sun.j3d.utils.universe.SimpleUniverse`

Un objet de classe **SimpleUniverse** est constitué :

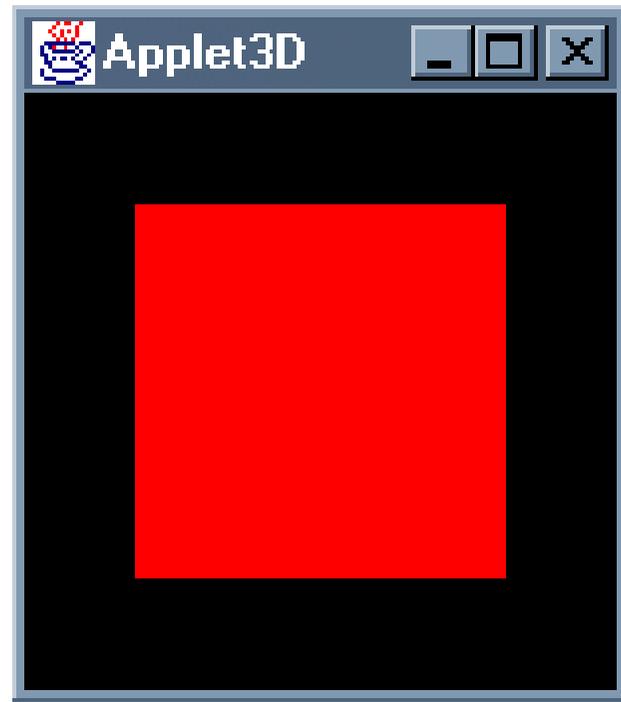
- d'un objet **VirtualUniverse** qui représente la porte d'entrée dans la scène (c'est la racine de l'arbre de l'application)

- d'un objet **Locale** qui est le descendant direct de la racine et qui permet de gérer le système de coordonnées de l'application

- d'un ou plusieurs nœuds de type **BranchGroup** et **TransformGroup**

Il peut y avoir plusieurs objets **VirtualUniverse** et plusieurs objets **Locale** rattachés à un **Simple Universe** un mais très souvent un seul suffit.

Un exemple : le cube

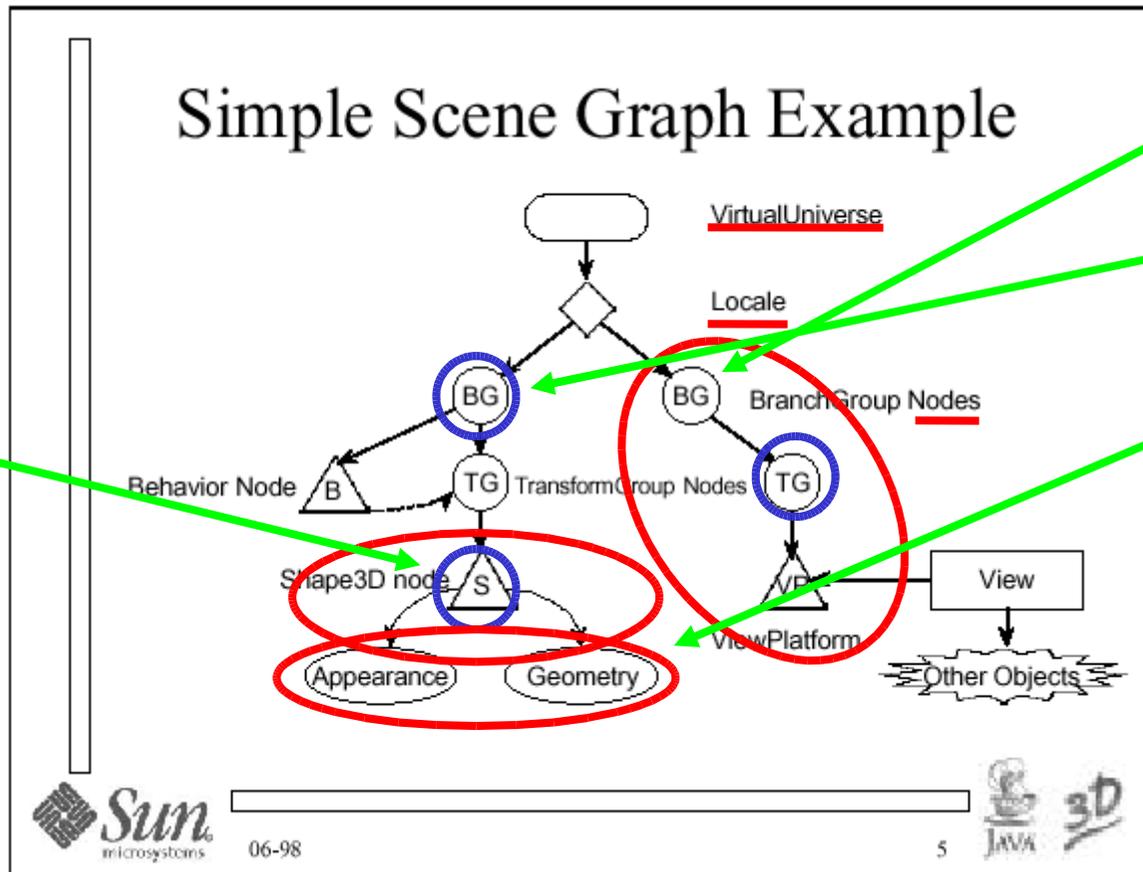


Graphe de scènes

Node

Leaf

- Background
- Behavior
- BoundingLeaf
- Clip
- Fog
- Light
- Link
- Morph
- Shape3D
- Sound
- Soundscape
- ViewPlatform



Group

- BranchGroup
- Java 3D Object Hierarchy
- SceneGraphObject
- Node
- Group
- Leaf
- NodeComponent
- Geometry
- Appearance
- ColoringAttributes
- LineAttributes
- PointAttributes
- PolygonAttributes
- RenderingAttributes
- TextureAttributes
- TransparencyAttributes
- Material
- Texture



06-98

5



Procédure de création d'un « univers 3D »

1. Créer un objet **Canvas3D**
2. Créer un objet **VirtualUniverse**
3. Créer un objet **Locale** rattaché au **VirtualUniverse**
4. Créer un objet **BranchGroup**
5. Créer des branches pour les différentes formes 3D
6. Insérer les branches dans l'objet **Locale**

Les étapes 2, 3 et 4 sont réalisées automatiquement

Un exemple complet : le cube

```
import java.applet.Applet;
import java.awt.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.SimpleUniverse;
import com.sun.j3d.utils.geometry.ColorCube;
```

```
public class Applet3D extends Applet {
    public void init () {
        Canvas3D canvas = new Canvas3D (SimpleUniverse.getPreferredConfiguration ( ));
        setLayout (new BorderLayout ( ));
        add (canvas, BorderLayout.CENTER);
        BranchGroup scene = createSceneTree ( );
        SimpleUniverse universe = new SimpleUniverse (canvas);
        universe.addBranchGraph (scene);
        universe.getViewingPlatform ( ).setNominalViewingTransform ( );
    }
}
```

Création d'un composant de classe Canvas3D permettant de visualiser une scène 3D

Création de la scène 3D à visualiser

Création d'un univers 3D rattaché au composant 3D

Rattachement de la scène 3D à l'univers

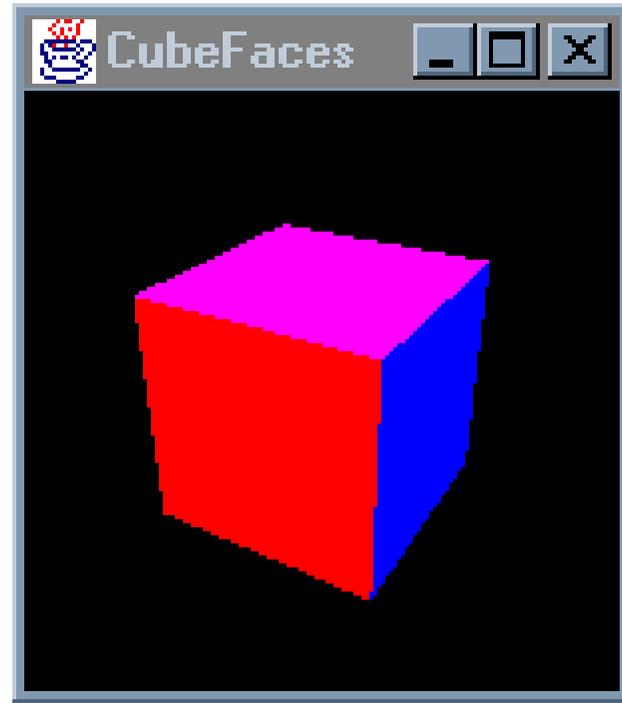
Positionnement pour avoir une vue correcte sur la scène 3D (permet de voir une scène 3D contenue dans un cube d'1 unité de côté et centré sur le centre du repère i.e. recule l'oeil)

Création d'un cube coloré dont deux des sommets opposés sont situés en (-0.5,-0.5,-0.5) et (0.5,0.5,0.5)

Racine de l'arbre des objets représentés dans la scène 3D

Ajout du cube à la racine de l'arbre

Les transformations 3D



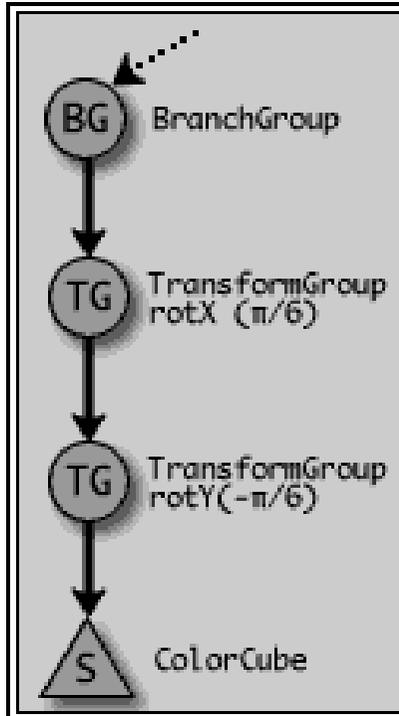
Les transformations 3D

Les **transformations** sont utilisées en 3D pour positionner et animer une forme 3D dans l'espace

Java 3D utilise la classe **Transform3D** pour décrire une opération de translation, de rotation ou d'homothétie

Cette opération est associée à une instance de la classe **TransformGroup** et ajoutée à l'arbre de la scène pour l'appliquer sur une forme

Arbre représentant le cube après rotations



L'arbre représentant la scène 3D du cube après une rotation de $\pi/6$ autour de l'axe des X puis une rotation de $-\pi/6$ autour de l'axe des Y

Un exemple : le cube après rotations

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

```
public class CubeFaces extends Applet3D {
```

```
public BranchGroup createSceneTree () {
```

```
    BranchGroup root = new BranchGroup ();
```

```
    Transform3D rotationXAxis = new Transform3D ();
```

```
    rotationXAxis.rotX (Math.PI/6);
```

```
    TransformGroup rotationXAxisGroup = new TransformGroup (rotationXAxis);
```

```
    Transform3D rotationYAxis = new Transform3D ();
```

```
    rotationYAxis.rotY (-Math.PI/6);
```

```
    TransformGroup rotationYAxisGroup = new TransformGroup (rotationYAxis);
```

```
    ColorCube cube = new ColorCube (0.5);
```

```
    rotationYAxisGroup.addChild (cube);
```

```
    rotationXAxisGroup.addChild (rotationYAxisGroup);
```

```
    root.addChild (rotationXAxisGroup);
```

```
    return root;
```

```
    }  
}
```

Méthode de la classe Applet3D surchargée

Racine de l'arbre des objets représentés dans la scène 3D

Création d'une rotation de $\pi/6$ autour de l'axe X

Création d'une rotation de $-\pi/6$ autour de l'axe Y

Création d'un cube coloré

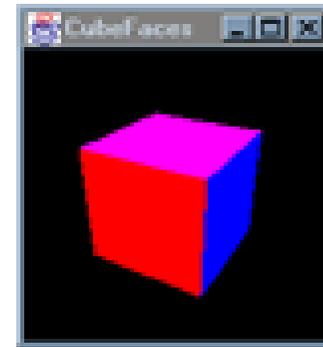
Construction de la branche de l'arbre de la scène

Ordre des transformations

L'ordre dans lequel sont effectuées les transformations a une importance.

Si, dans la méthode **createSceneTree()** de la classe **CubeFaces**, l'enchaînement des rotations est différent, on obtient un résultat différent

```
// Construction de la branche de l'arbre de la scène  
// Rotation autour de l'axe y puis de l'axe x  
rotationYAxisGroup.addChild (cube);  
rotationXAxisGroup.addChild (rotationYAxisGroup);  
root.addChild (rotationXAxisGroup);
```



```
// Construction de la branche de l'arbre de la scène  
// Rotation autour de l'axe x puis de l'axe y  
rotationXAxisGroup.addChild (cube);  
rotationYAxisGroup.addChild (rotationXAxisGroup);  
root.addChild (rotationYAxisGroup);
```



Démonstration (2bisCubeFaces2.bat et 2CubeFaces.bat)

Quelques conseils

Dessiner un arbre quand vous créez vos scènes 3D

permet d'effectuer mentalement l'assemblage des différentes formes d'une scène

documentation plus lisible que le code

Attention à ne pas affecter **deux parents à un nœud**

Java 3D déclenchera une exception

MultipleParentException

Attention à ne pas créer de **branches inutiles**

Java 3D ne signalera aucune erreur à l'exécution mais le résultat ne correspondra pas à vos attentes

Optimisation

Java 3D propose une notion pour améliorer les performances des calculs d'affichage 3D :

La compilation des nœuds d'une scène

Compilation des nœuds

Compilation => ? représentation interne que Java 3D manipule de manière optimum

En fait peu précisé dans les spécifs Java 3D.

La méthode **compile()** de la classe **BranchGroup** optimise tout le sous-arbre de racine ce **BranchGroup**.

La méthode **isCompiled()** de la classe **SceneGraphObject** de savoir si un nœud a été compilé (= fils d'un **BranchGroup** compilé)

La classe **Applet3D** peut être optimisée en ajoutant l'instruction **scene.compile();** dans la méthode **init()**

Nœud vivant, aptitude (capacity) d'un nœud

Ajouter un point de branchement (**BranchGroup**) à un **Locale** le rend *vivant*, ainsi que tous les objets de sa sous-arborescence.

Pour **modifier/lire les caractéristiques d'un objet vivant ou compilé** d'un nœud (changer la transformation pour créer une animation), il faut le préciser **avant** son ajout dans le **Locale** ou **avant** la **compilation**.

setCapability() de la classe `SceneGraphObject` autorise la lecture ou la modification de la propriété correspondante d'un nœud compilé ou vivant

Capacité d'un nœud TransformGroup : exemple

setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE)

Permet d'appeler la méthode **setTransform()** pour changer la transformation

Remarque

Quand la racine de l'arbre d'une scène 3D est rattaché à un univers avec la méthode **addBranchGraph()**, Java 3D vérifie les différentes aptitudes (capacity) de chacun des nœuds en utilisant la méthode **getCapability ()** et optimise la représentation interne des objets 3D.

Capacité d'un nœud : exemple

```
public BranchGroup createSceneGraph() {
    BranchGroup objRoot = new BranchGroup ( );

    objTrans = new TransformGroup ( );
    objTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    objRoot.addChild(objTrans);

    objTrans.addChild(new ColorCube(0.4));

    return objRoot;
}

public void actionPerformed(ActionEvent e ) {
    if (e.getSource()==rotateB){
        angle += Math.toRadians( 10 );
        trans.rotY( angle );
        objTrans.setTransform(trans);
    }
}
```

Animation d'une scène 3D

On utilise les notions suivantes :

Interpolator (sous classe de **Behavior**) :
construit une interpolation du temps entre deux
valeurs

Alpha : précise le nombre de fois et
la vitesse du mouvement

BoundingSphere : indique la zone (sphere)
réellement modifiée (zone de clipping 3D ?!)
par le mouvement

Syntaxe

Les deux premiers sont reliés par :

```
RotationInterpolator(Alpha alpha, TransformGroup target)
```

```
puis void setSchedulingBounds(Bounds region)
```

lancé sur un **Behavior**

Animation d'une scène 3D : code

```
public BranchGroup createSceneTree() {
    // Cree la racine BranchGroup
    BranchGroup objRoot = new BranchGroup();

    // Cree le nœud TransformGroup. Rend cette transformation
    // modifiable, ajoute cette transformation à BranchGroup
    TransformGroup objSpin = new TransformGroup();
    objSpin.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    objRoot.addChild(objSpin);

    // Cree le cube et l'ajoute au graphe de scène.
    objSpin.addChild(new ColorCube(0.4));

    // Cree les caracteristiques de l'animation rotation : la rotation
    // ne s'arrete pas (-1) et le tour est complet en 4 secondes (4000).
    Alpha rotationAlpha = new Alpha(-1, 4000);

    // Cree un Behavior qui est une animation rotation, avec les
    // caracteristiques rotationAlpha.
    // Associe cette animation rotation au TransformGroup objSpin
    RotationInterpolator rotator =
        new RotationInterpolator(rotationAlpha, objSpin);

    // Creer une sphere centrée en (0,0,0), de rayon 1 enveloppant la zone
    // sensible à l'animation et donc tout le cube. L'associe à l'animation.
    BoundingSphere bounds = new BoundingSphere();
    rotator.setSchedulingBounds(bounds);
    objSpin.addChild(rotator);

    return objRoot;
} // fin de la méthode CreateSceneGraph
```

Animation d'une scène 3D : code

Démonstration (`3AnimCube.bat`)

Conclusion

Points forts

Avantages de Java (indépendant de la machine, gratuité, collection d'applets ...)

Facilité de la mise en œuvre

Richesse des bibliothèques graphiques (couleur, éclairage, texture ...)

Points faibles

Très gourmand en mémoire

Assez lent

Bibliographie Java et le multimédia

<http://java.sun.com/javase/technologies/desktop/techoverview.jsp> : Un site sur les technologies Java additionnelles au J2SE entre autre le multimédia (Java 3D, JOGL = Java Bindings for OpenGL, Java Sound, Java Media Framework, Java Speech)

Construire des mondes 3D

- ... à l'aide de modeleurs 3D
- Exemples:
 - 3D Studio Max
 - Lightwave 3D
 - Wavefront obj
 - VRML
 - ...

VRML

```
#VRML V2.0 utf8
DEF T Transform {
children [
  Transform { # top
    translation 0.0 0.4 0.0
    children
    Shape {
      appearance Appearance {
        material Material { diffuseColor 0.8 0.8 0.8 }
      }
      geometry Box { size 1.3 0.15 1.3 }
    }
  }
  Transform { # leg 1
    translation -.5 0.0 -0.4
    children
    DEF Leg Shape {
      appearance Appearance {
        material Material { diffuseColor 1.0 1.0 0.0 }
      }
      geometry Cylinder { height 0.9 radius .075 }
    }
  }
  Transform { # leg 2
    translation .5 0.0 -0.4
    children USE Leg
  }
  Transform { # leg 3
    translation -.5 0.0 0.4
    children USE Leg
  }
  Transform { # leg 4
    translation .5 0.0 0.4
    children USE Leg
  }
]
}
```



Une table en
VRML

Charger des mondes 3D avec Java 3D

- Use a scene loader
- From 3D world file to a Scene object:
method `Scene load(String filename)` of the `Loader` interface
- `com.sun.j3d.loaders.Scene` is a Java interface
- Use the right class which implements this interface
- Pattern Bridge

De la scene aux objets Java 3D

- `public BranchGroup getSceneGroup()`
 - returns the `BranchGroup` containing the overall scene loaded by the loader. All enabled items will be loaded into this `Scene`
- `public Hashtable getNamedObjects()`
 - returns a `Hashtable` which contains a list of all named objects in the file and their associated scene graph objects.
 - For example, DEF names of VRML or filenames of objects in Lightwave 3D.

Charger un monde 3D VRML dans Java 3D

```
// ...
import com.sun.j3d.loaders.Scene;
import org.web3d.j3d.loaders.VRML97Loader;
public class MyLoaderVRML extends JFrame {
    public MyLoaderVRML ( ) {
        // ...
        // change BranchGroup scene = buildScene(); by
        BranchGroup scene = loadScene();
        // ...
    }

    public BranchGroup loadScene ( ) {
        BranchGroup rootBG = new BranchGroup ( );

        VRML97Loader loaderVRML = new VRML97Loader();
        Scene theScene = null;
        // ...
        theScene = loaderVRML.load(nomFichierDeScene);
        // ...
        Node forme3D = theScene.getSceneGroup();
        rootBG.addChild(forme3D);
        return rootBG;
    }

    public static void main(String args[] ) {
        new MyLoaderVRML ( );
    }
}
```

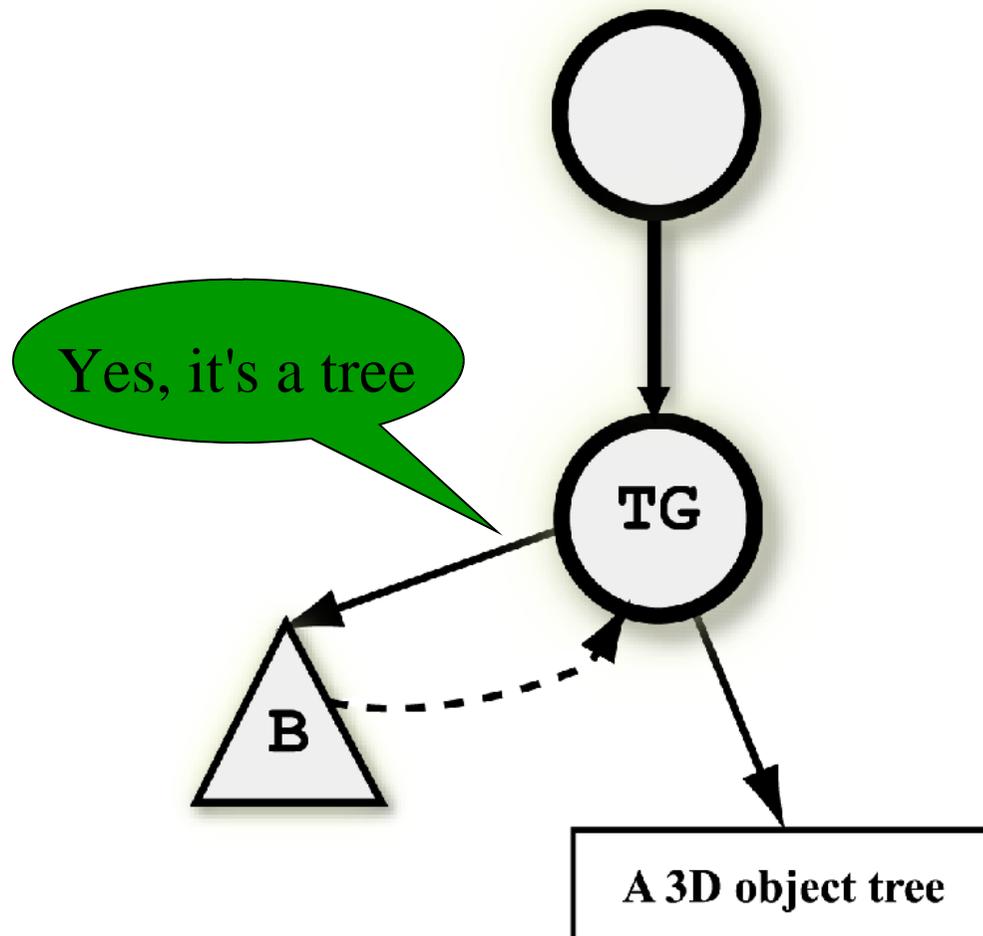
- Download Xj3D at <http://www.web3d.org/TaskGroups/source/xj3d.html> to obtain a VRML loader



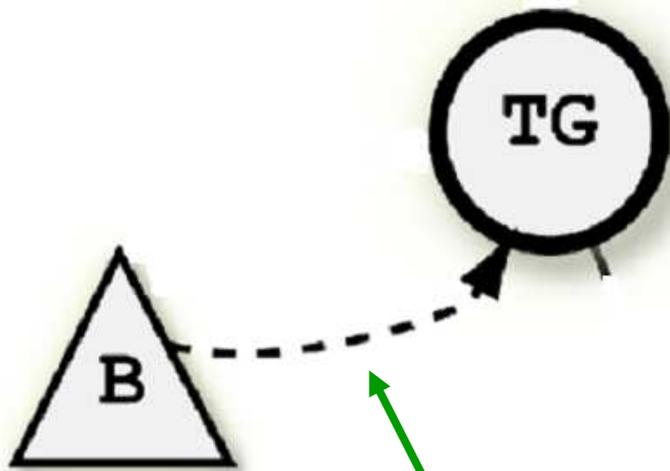
Animation et Interaction

- Describe by the abstract class Behavior
- a behavior = a link between a stimulus and an action
- A lot of behaviors are given by Java 3D:
Interpolator (PositionInterpolator, RotationInterpolator) or MouseRotate, MouseTranslate
- A Java 3D programmer can define a behavior

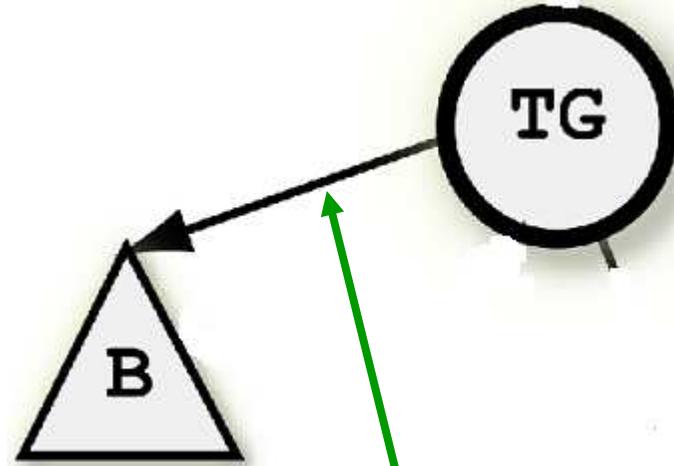
Graphe Java 3D pour interactions et animations



Programmer les interactions



```
import com.sun.j3d.utils.behaviors.mouse.*;  
// ...  
MouseRotate mouseRotationB = new MouseRotate();  
mouseRotationB.setTransformGroup(galleonTG);
```



```
galleonTG.addChild(mouseRotationB);
```

An interaction demo



- Use a WaveFront loader

4Galleon.bat

Conclusion

Points forts

Avantages de Java (indépendant de la machine, gratuité, collection d'applets ...)

Facilité de la mise en œuvre

Richesse des bibliothèques graphiques (couleur, éclairage, texture ...)

Points faibles

Très gourmand en mémoire

Assez lent

Bibliography Java 3D

- Java 3D in the real world
http://java.sun.com/products/java-media/3D/in_action/