L’API Java 3D

UE optionnelle EVC
Master Informatique 2 GL et Mitic

2009-2010

Thierry Duval (d’après la bibliographie !...)
IFSIC / IRISA - Bunraku
thierry.duval@irisa.fr

Introduction

- API Java de Sun destinée à l'affichage 3D sous Java
  - Sun jusque la version 1.3
  - communauté open-source depuis (actuellement 1.5.2)
  - https://java3d.dev.java.net/
- Destiné à l'écriture d'applications et d'Applets
- Conçu dans le but de faciliter la programmation d'environnements 3D
- Justifié par la portabilité de Java qui permet la portabilité des applications
- Fonctionne sur toute plate-forme où Java existe :
  - sur systèmes Solaris, Windows, Linux, MacOS
  - sur architectures Intel, Amd, Sparc

Principales fonctionnalités (1)

- Modélisation et affichage 3D de scènes :
  - ✔ primitives classiques de modélisation
    - ✗ types primitifs, listes de facettes, ...
  - ✔ moteur de rendu : Z-Buffer + ombrage de Gouraud tendu
    - ✗ permet de visualiser en « temps interactif »
  - ✔ gestion d'un graphe de scène
    - ✗ avec ses règles de parcours...
- Importation de géométries à l'aide de « loaders »
  - ✔ à condition que quelqu'un les ait écrits...
- Évolutivité par dérivation de classes

Principales fonctionnalités (2)

- Notion de « Behavior » :
  - ✔ possibilité d'implanter des comportements spécifiques programmés
    - ✗ essentiellement pour l'interaction, la navigation
- Notion d'« Interpolator » :
  - ✔ possibilité de « brancher » l'heure du système sur les variables définissant une scène :
    - ✗ permet de réaliser des animations

Principales fonctionnalités (3)

- Détection des collisions entre objets
- Sélection d’objets
- Gestion de périphériques d'entrée
- Fonctionnalités classiques de l’informatique graphique :
  - ✔ brouillard
  - ✔ antialiasing
  - ✔ ...
- Gestion des sons dans un environnement 3D

Plan

- Généralités sur le graphe de scène
  - ✔ organisation, rendu, univers virtuels, repères, branches graphiques, branche de visualisation
- Comportements
  - ✔ animation, interaction
- Groupes et feuilles
- Géométrie
  - ✔ formes, attributs, chargeurs
- Outils mathématiques
  - ✔ changement de point de vue
- Exemples

Graphe de scène (Scene graph)

- Modèle de programmation de Java 3D permettant la représentation et le rendu de scènes :
  - ✔ description complète de la scène (ou univers virtuel)
  - ✔ données géométriques, divers attributs, informations de visualisation
  - ✔ graphe de scène organisé sous forme arborescente par assemblage d'éléments (objets), liés par des relations de type père ↔ fils

Graphe de scène et POO

- Graphe de scène :
  - ✔ construit par assemblage d'éléments graphiques constitués d'objets Java individuels instanciés à partir de classes Java
- Extensibilité :
  - ✔ possibilité de créer ses propres classes par dérivation de classes existantes

Graphe de scène : présentation

- Une application Java3D doit construire un graphe de scène acyclique...
- Les composants de ce graphe :
  - « Shape3D »
    - « Geometry »
    - « Appearance »
  - « Group »
    - « TransformGroup », « BranchGroup », ...
  - « Light »
    - « AmbientLight », « DirectionalLight », « PointLight »
  - « Fog », « Background »
  - « Sound »
  - « Behavior »
  - « ViewPlatform »

Graphe de scène : exécution (1)

- Le graphe de scène spécifie le contenu, mais pas l'ordre du rendu :
  - ✔ celui-ci est choisi par Java3D
- Pas de restriction de gauche à droite ou du sommet vers le bas :
  - ✔ à l'exception des attributs limités spatialement :
    - ✗ source de lumière
    - ✗ brouillard

Graphe de scène : exécution (2)

- Java3D utilise plusieurs threads, asynchrones et indépendants :
  - ✔ rendu graphique
  - ✔ rendu sonore
  - ✔ animation du comportement
  - ✔ gestion des périphériques
  - ✔ génération des événements (détection de collision)

Hautes performances

- Programmation simplifiée d'applications gérant concurremment des tâches du type :
  - parcours du graphe de scène
  - gestion des changements des variables d'état
  - …
- Support natif de :
  - OpenGL
  - Direct 3D
  - QuickDraw 3D
- Optimisations

Modèles de rendu

- Existence de trois modèles de rendu permettant d'espérer des optimisations conduisant à l'amélioration de la vitesse d'affichage au prix d'une programmation moins flexible :
  - ✔ Immediate mode
  - ✔ Retained mode
  - ✔ Compiled-Retained mode

Immediate mode

- Permet d’utiliser ou non la structure graphe de scène inhérente à l’API Java 3D
- Maximum de flexibilité au prix d’une moins bonne vitesse de rendu

Retained mode

- Tous les objets définis dans le graphe de scène sont accessibles et manipulables (création, destruction, modification, ...) par programme, par sélection, …
- Optimisation de la vitesse de rendu par réalisation automatique d'optimisations par Java3D (opération possible car Java 3D connaît ce que le programmeur a réalisé)

Compiled-Retained mode

- Meilleures performances de rendu par optimisation poussée
- Perte de la flexibilité de programmation liée aux possibilités d'accès et de modification des objets
- Conservation d'un accès aux objets
  - via les « capability flags »

La hiérarchie de classes Java 3D : javax.media.j3d

- « VirtualUniverse »
- « Locale »
- « View »
- « PhysicalBody »
- « PhysicalEnvironment »
- « Screen3D »
- « Canvas3D » (hérite de java.awt.Canvas, non conteneur)
- « SceneGraphObject »
  - « Node » (« Group », « Leaf »)
  - « NodeComponent »
- « Transform3D »

Graphe de scène : terminologie (1)

- « Node » : item dans un graphe de scène
  - ✔ « Leaf » nodes : nœuds sans enfants (feuilles)
    - ✗ « Shape », « Light », « Sound », …
    - ✗ « Behavior » (pour l'interaction, la navigation et l'animation)
  - ✔ Goup nodes : nœuds avec enfants
    - ✗ « Transform », « Switch », …
- « NodeComponent » : attribut de nœud
  - ✔ « Geometry » ou « Color » pour une « Shape3D »
  - ✔ Données à émettre pour un « Sound »
  - ✔ ...

Graphe de scène : terminologie (2)

- Vivant (live)
  - attaché à un graphe de scène
- Compilé (compiled), dans un format optimisé :
  - compiler avant l'attachement au graphe de scène principal
  - ne peut pas être désassemblé
- Certaines actions dépendent de l'état vivant ou compilé :
  - une fois vivant ou compilé, les caractéristiques (capabilities) d'un nœud ne peuvent plus être changées

Univers Virtuel (« VirtualUniverse »)

- Collection de graphes de scène :
  - ✔ généralement un par application
  - ✔ container de plus haut niveau pour les graphes de scène
- Contient un ensemble d’objets « Locale » :
  - ✔ chaque Locale ayant une position « haute-résolution » dans cet univers virtuel définissant ainsi un repère
- Définition des méthodes pour énumérer ses objets « Locale » et les gérer

« SimpleUniverse »

- Définition d’un environnement utilisateur minimum :
  - ✔ pour créer facilement un programme Java3D
- Création des objets nécessaires à la gestion de la partie « view » du graphe de scène
- Partie « géométrie » initialisée à vide

Repère (« Locale »)

- Définition d'une position en haute-résolution dans un « VirtualUniverse » (généralement un par univers) :
  - ✔ méthodes « set » et « get » nécessaires à la configuration des coordonnées haute-résolution
- Container pour un ensemble de sous-graphes enracinés en cette position :
  - ✔ objets dans un objet « Locale » définis en utilisant des coordonnées double-précision relativement à l'origine de cet objet (définition du système de coordonnées du monde virtuel pour cet objet « Locale »)
  - ✔ méthodes pour ajouter, retirer et énumérer des graphes branches (« BranchGraph ») contenus

Branches : terminologie

- Branche de scène :
  - ✔ formes, lumières, etc…
    - ✗ plusieurs par repère
- Branche de visualisation :
  - ✔ généralement une par univers (au moins ?!...)

Graphe de scène

- Constitué de nœuds organisés en arbre au moyen de relations père ↔ fils
- Constituants : super-classe « SceneGraphObject » dérivée vers « Node », elle-même dérivée en deux sous-classes :
  - ✔ « Group » : 1 ou plusieurs nœuds fils (ou même 0), un seul nœud parent
  - ✔ « Leaf » : 0 nœuds fils, un seul nœud parent :
    - ✗ formes géométriques
    - ✗ lumières
    - ✗ brouillard
    - ✗ sons
    - ✗ ...

Parcours du graphe de scène

- Sans notion d'ordre de parcours horizontal :
  - ✔ nœuds fils d'un nœud parcourus par la fonction de rendu de Java 3D dans un ordre arbitraire voire même en parallèle...
- « Héritage » :
  - ✔ un nœud fils hérite de l'ensemble des paramètres d'état de son père
  - ✔ s'il possède un frère, un nœud n'hérite pas de ses paramètres d'état (une seule exception, les nœuds lumière et brouillard)
  - ✔ « héritage » vertical (de la racine vers le nœud) et non horizontal

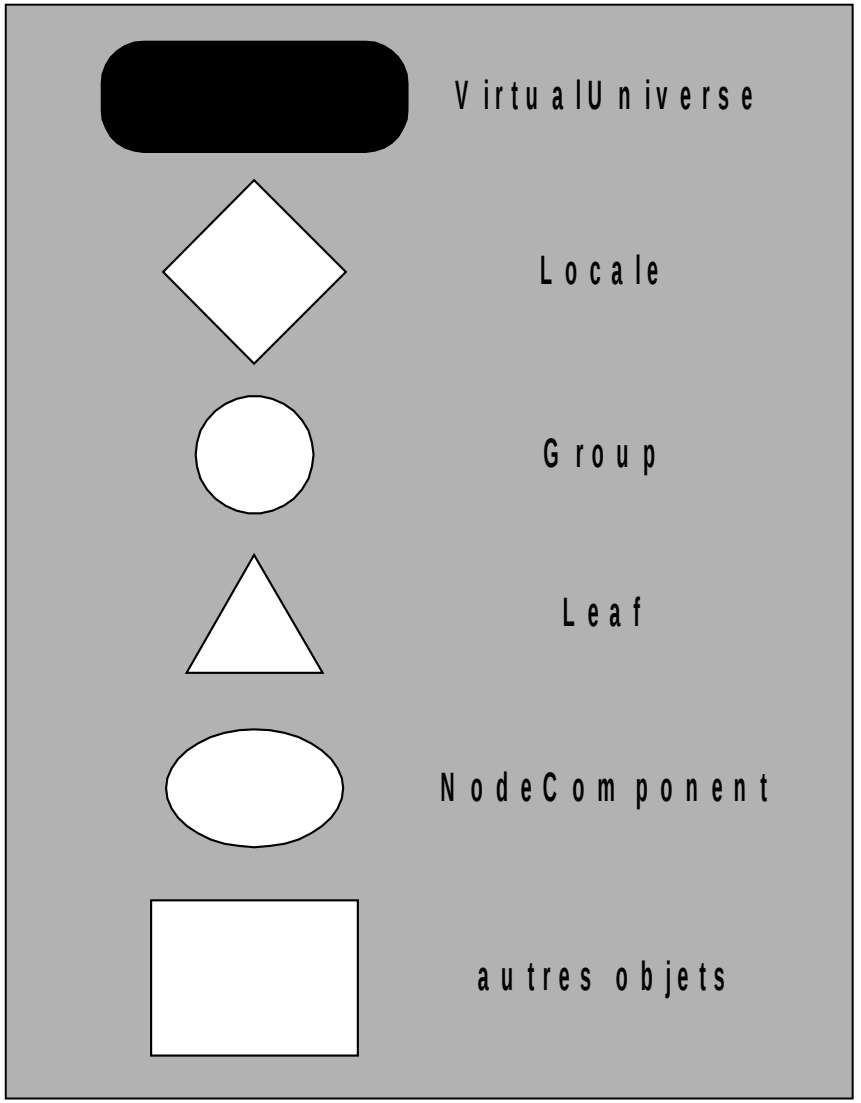
Graphe de scène : les nœuds

- Construits par instanciation de la classe souhaitée
- Accédés via les méthodes get et set existantes et autorisées (pour permettre des optimisations à l'exécution)
- Ont pour super-classe « SceneGraphObject »
- 2 classes dérivées de « SceneGraphObject » :
  - « Node » : classe abstraite pour les nœuds de structuration d'un graphe de scène et les nœuds feuilles d'un tel graphe
  - « NodeComponent » : classe abstraite pour les nœuds définissant certaines composantes des graphes de scène

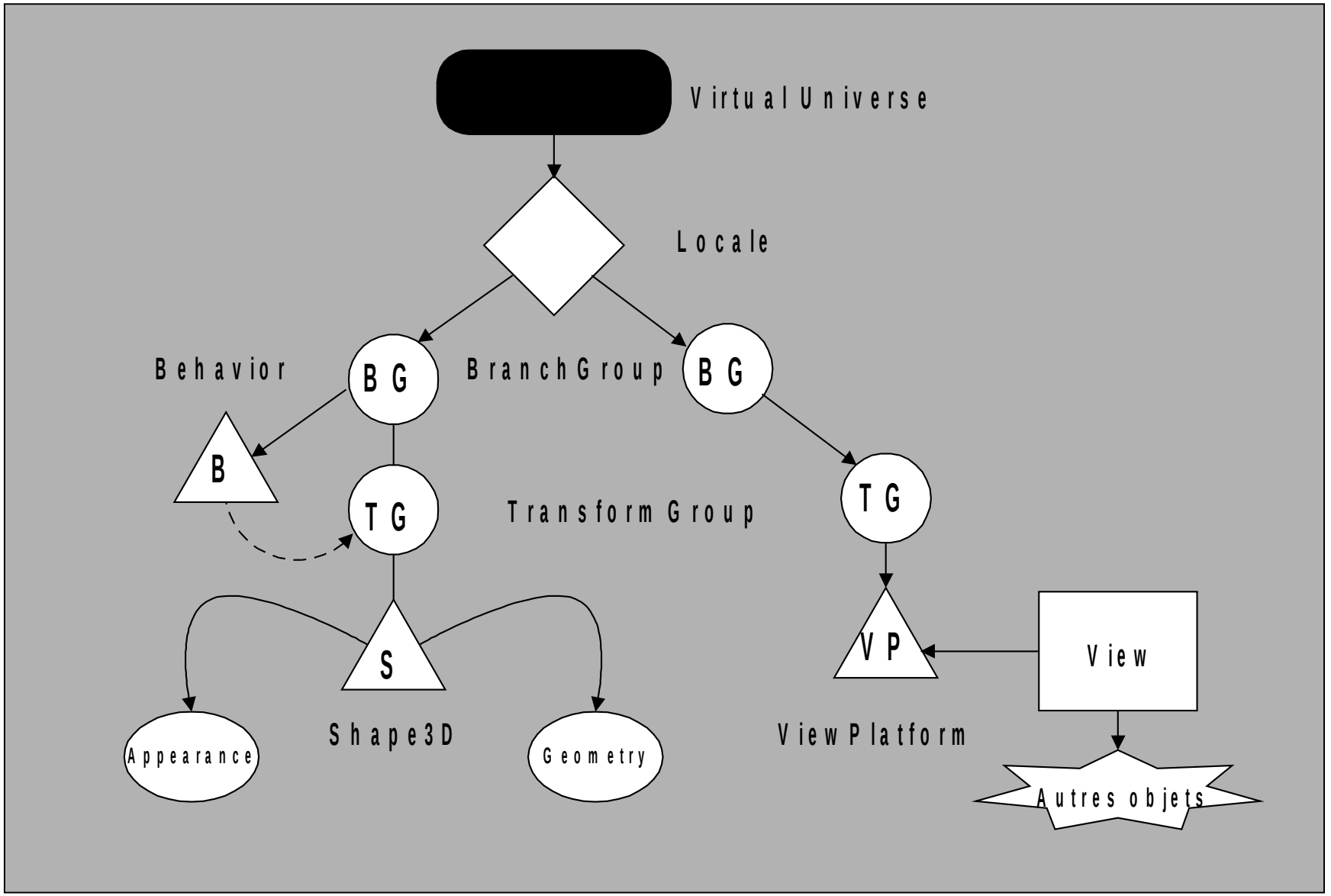
Environnement nécessaire à Java 3D

- Utilisation d'un objet « Canvas3D » comme zone de dessin pour l'application ou l'applet
- Création d'un objet « View » pour définir tous les paramètres nécessaires au rendu d'une scène depuis un point de vue arbitraire
- Création des éléments du graphe de scène :
  - ✔ un objet « VirtualUniverse »
  - ✔ un objet haute-résolution « Locale »
  - ✔ un objet « BranchGroup »
  - ✔ un objet nœud « TransformGroup »
  - ✔ un objet nœud feuille « ViewPlatform » qui définit la position et l'orientation du point de vue sur l'univers virtuel

Graphe de scène : représentation graphique



Graphe de scène : représentation graphique



Construction d’un graphe de scène (1)

- Construire une zone d’accueil 3D :
  - Canvas3D canvas = new Canvas3D (...) ;
- Construire un univers :
  - SimpleUniverse universe = new SimpleUniverse (canvas) ;
- Construire la branche de l’univers graphique :
  - BranchGroup branchGroup = new BranchGroup () ;

Construction d’un graphe de scène (2)

- Construction des nœuds et des groupes :
  - Shape3D sphape1 = new Shape3D (geom1, appear1) ;
  - Shape3D sphape2 = new Shape3D (geom2) ;
  - Group group = new Group () ;
  - group.addChild (shape1) ;
  - group.addChild (shape2) ;
- Modification des instances :
  - shape2.setAppearance (appear2) ;

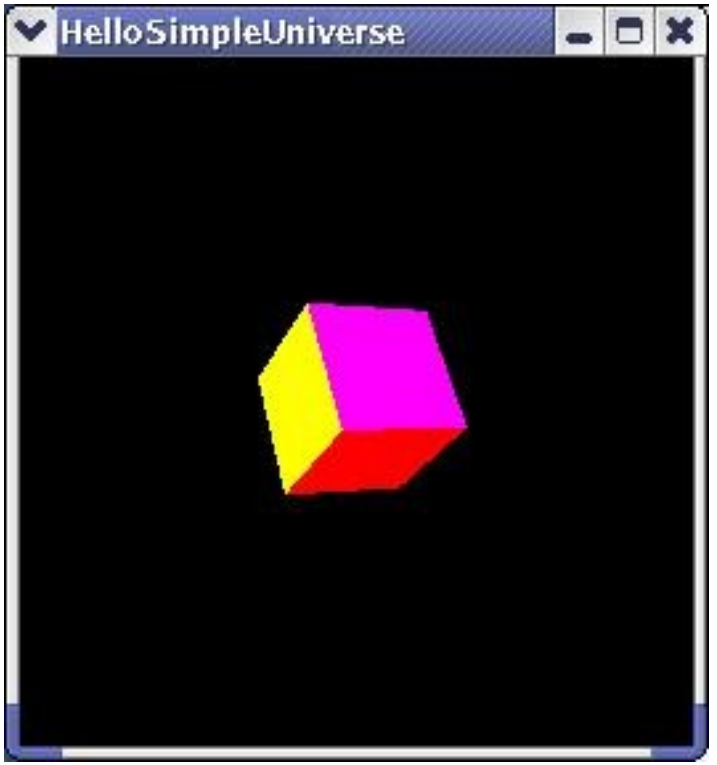
Construction d’un graphe de scène (3)

- Construction de l’arbre :
  - branchGroup.addChild (group) ;
- Compilation des branches graphiques :
  - branchGroup.compile () ;
- Association à un repère :
  - universe.addBranchGraph (branchGroup) ;

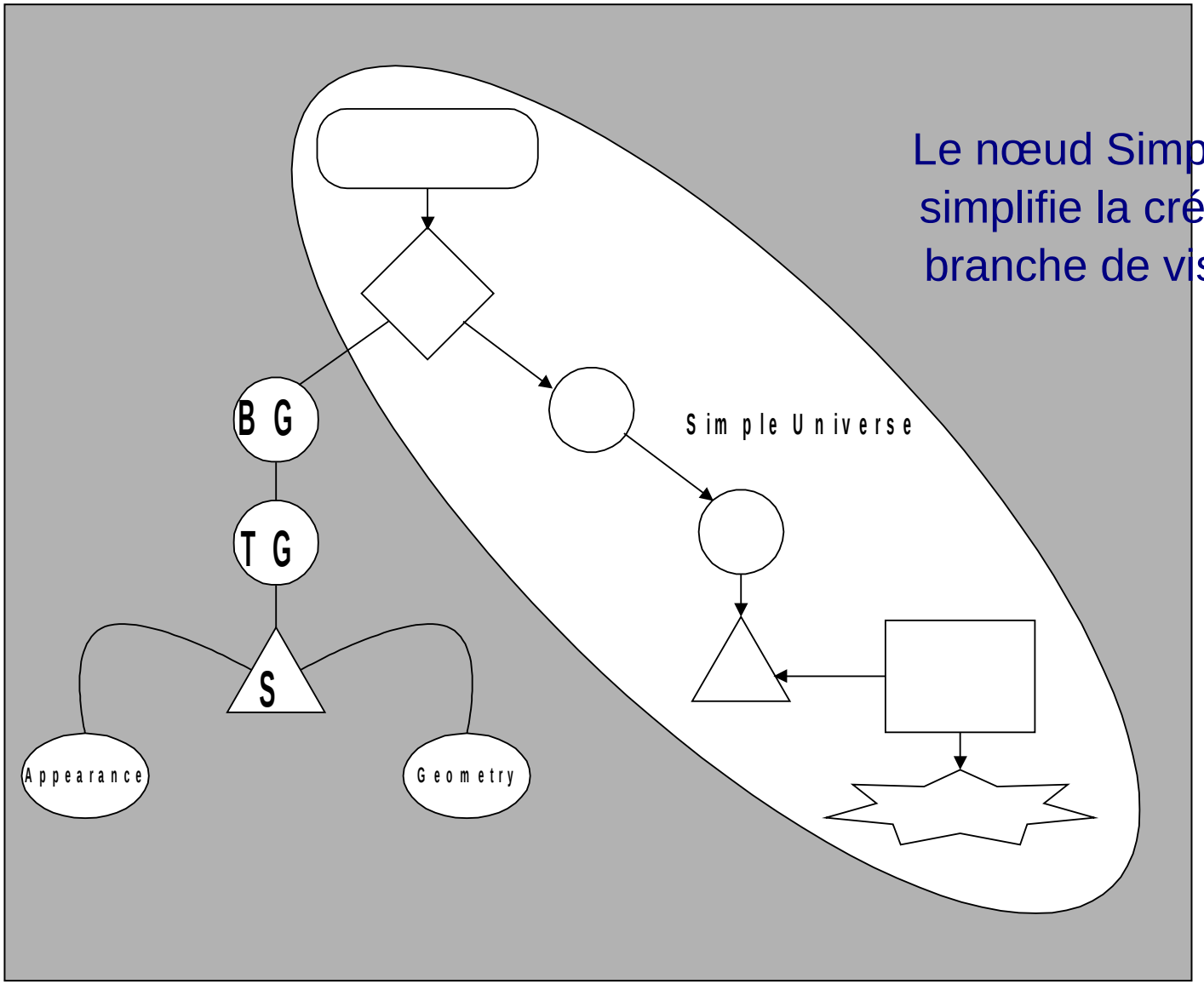
Premiers exemples Java3D

- Exemple élémentaire :
  - ✔ HelloSimpleUniverse
- Exemple avec interpolateur :
  - ✔ AnimatedSimpleUniverse
- Exemple avec navigation :
  - ✔ NavigationalSimpleUniverse
- Exemple avec interaction :
  - ✔ InteractiveSimpleUniverse

HelloSimpleUniverse



HelloSimpleUniverse : le graphe de scène



Le nœud SimpleUniverse simplifie la création de la branche de visualisation

HelloSimpleUniverse.java (1)

```
import java.awt.GraphicsConfiguration ;

import javax.media.j3d.BranchGroup ;
import javax.media.j3d.Canvas3D ;
import javax.media.j3d.Transform3D ;
import javax.media.j3d.TransformGroup ;
import javax.swing.JApplet ;
import javax.vecmath.Vector3d ;

import com.sun.j3d.utils.applet.MainFrame ;
import com.sun.j3d.utils.geometry.ColorCube ;
import com.sun.j3d.utils.universe.SimpleUniverse ;
```

HelloSimpleUniverse.java (2)

```
public class HelloSimpleUniverse extends JApplet {

    private SimpleUniverse u = null ;

    public HelloSimpleUniverse () {

    }

    public static void main (String [] args) {
        new MainFrame (new HelloSimpleUniverse (), 256, 256) ;
    }
```

HelloSimpleUniverse.java (3)

```
public void init () {
    GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration () ;
    Canvas3D c = new Canvas3D (config) ;
    getContentPane ().add (c) ;
    BranchGroup scene = createSceneGraph () ;
    u = new SimpleUniverse (c) ;
    u.getViewingPlatform ().setNominalViewingTransform () ;
    u.addBranchGraph (scene) ;
}

public void destroy () {
    u.removeAllLocales () ;
}
```

HelloSimpleUniverse.java (4)

```
    public BranchGroup createSceneGraph () {
        BranchGroup objRoot = new BranchGroup () ;
        Transform3D orientation = new Transform3D () ;
        orientation.setEuler (new Vector3d (Math.PI / 3, Math.PI / 6, 0)) ;
        TransformGroup objTrans = new TransformGroup (orientation) ;
        objRoot.addChild (objTrans) ;
        objTrans.addChild (new ColorCube (0.2)) ;
        objRoot.compile () ;
        return objRoot ;
    }

}
```

Animations : « Interpolator »

- Interpolation :
  - ✔ entre 2 valeurs
  - ✔ au cours du temps (à l'aide d'un « Alpha »)
  - ✔ sur un objet à transformer

La classe « Alpha » (1)

- Conversion automatique d'une valeur temporelle en une valeur comprise entre 0 et 1
- 5 périodes de temps définissable pour l'Alpha de manière cyclique ou non sur les quatre dernières :
  - ✔ délai initial (Alpha = 0)
  - ✔ Alpha croissant
  - ✔ Alpha à 1
  - ✔ Alpha décroissant
  - ✔ Alpha à 0

La classe « Alpha » (2)

- Paramètres :
  - LoopCount :
    - nombre d'exécution de cet objet Alpha
    - -1 indique qu'il boucle indéfiniment
  - TriggerTime :
    - temps en milli-secondes compté à partir du lancement du système à partir duquel cet objet sera activé pour la première fois
  - PhaseDelayDuration :
    - nombre de milli-secondes attendues apr s le moment de déclenchement avant de démarrer cet Alpha
  - Mode :
    - peut être défini à INCREASING_ENABLE, DECREASING_ENABLE, ou à la valeur des deux
    - INCREASING_ENABLE active les paramètres Alpha croissants
    - DECREASING_ENABLE active les paramètres Alpha décroissants

La classe « Alpha » (3)

- Paramètres croissants Alpha :
  - IncreasingAlphaDuration :
    - période de temps durant laquelle l'Alpha varie de 0 à 1
  - IncreasingAlphaRampDuration
  - AlphaAtOneDuration :
    - période de temps durant laquelle l'Alpha reste à 0
- Paramètres décroissants Alpha :
  - DecreasingAlphaDuration :
    - période de temps durant laquelle l'Alpha varie de 1 à 0
  - DecreasingAlphaRampDuration
  - AlphaAtZeroDuration :
    - période de temps durant laquelle l'Alpha reste à 0

Classes dérivées de « Interpolator » (1)

- « ColorInterpolator » :
  - ✔ interpolation entre deux couleurs pour un objet « Material »
- « KBRotPosScaleSplinePathInterpolator »
- « RotPosScaleTCBSplinePathInterpolator »
- « PositionInterpolator » :
  - ✔ interpolation entre deux positions pour la composante de translation d’un objet « TransformGroup »
- « RotationInterpolator » :
  - ✔ interpolation entre deux valeurs d’angle pour la composante de rotation d'un objet « TransformGroup »

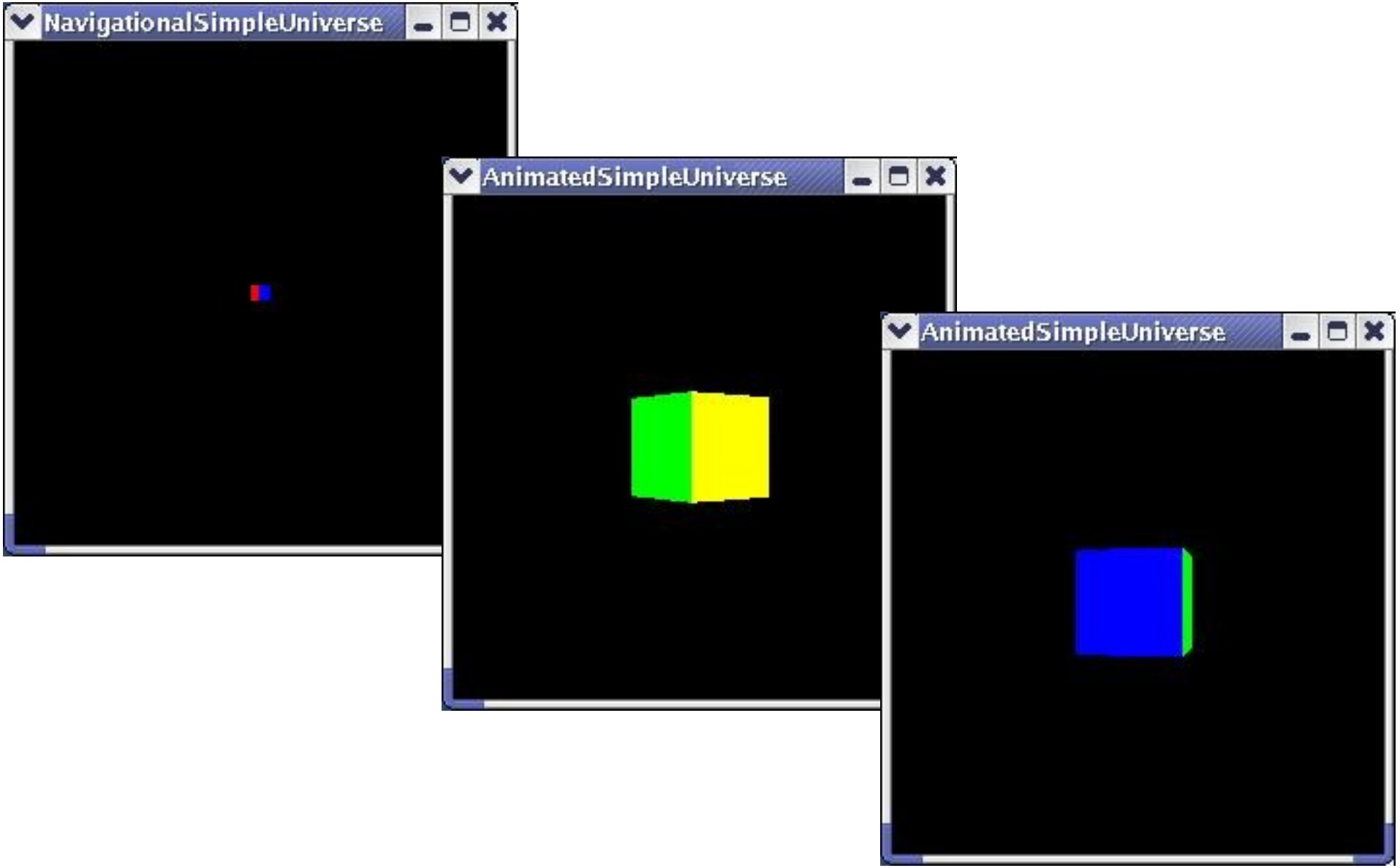
Classes dérivées de « Interpolator » (2)

- « ScaleInterpolator » :
  - interpolation entre deux valeurs de mise à l'échelle pour la composante de mise à l'échelle uniforme d'un objet « TransformGroup »
- « SwitchValueInterpolator » :
  - interpolation entre deux valeurs d'index pour un objet « Switch »
- « TransparencyInterpolator » :
  - interpolation entre deux valeurs de transparence pour un objet « TransparencyAttributes »

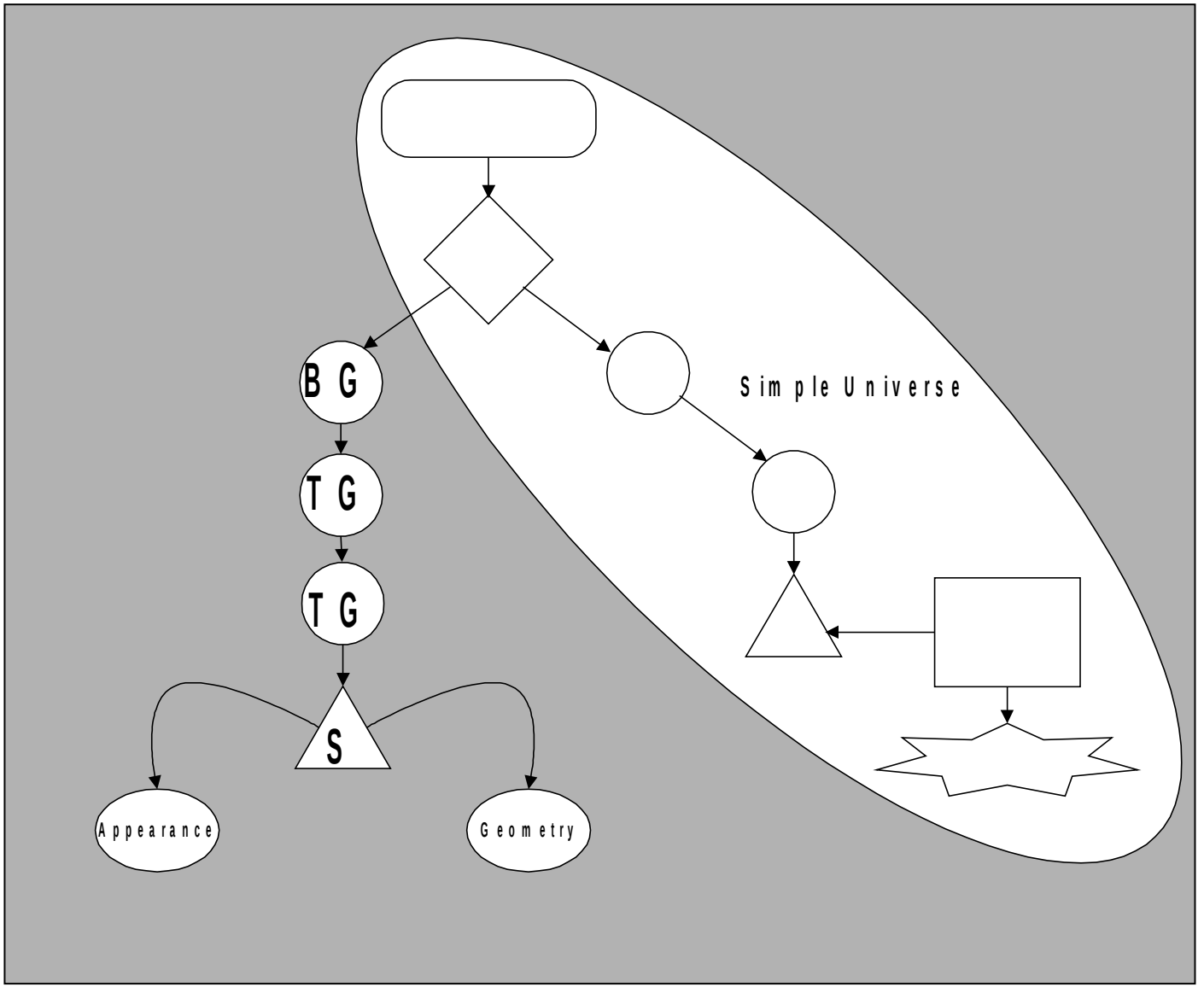
Classes dérivées de « Interpolator » (3)

- « PositionPathInterpolator » :
  - interpolation entre une série de positions pour la composante de translation d'un objet « TransformGroup »
- « RotationPathInterpolator » :
  - idem pour la composante d'orientation
- « RotPosPathInterpolator » :
  - idem pour les 2 composantes
- « RotPosScalePathInterpolator » :
  - idem pour l'échelle en plus …

AnimatedSimpleUniverse



AnimatedSimpleUniverse : le graphe de scène



AnimatedSimpleUniverse.java (1)

```
import java.awt.GraphicsConfiguration ;

import javax.media.j3d.Alpha ;
import javax.media.j3d.BoundingSphere ;
import javax.media.j3d.BranchGroup ;
import javax.media.j3d.Canvas3D ;
import javax.media.j3d.RotationInterpolator ;
import javax.media.j3d.Transform3D ;
import javax.media.j3d.TransformGroup ;
import javax.swing.JApplet ;

import com.sun.j3d.utils.applet.MainFrame ;
import com.sun.j3d.utils.geometry.ColorCube ;
import com.sun.j3d.utils.universe.SimpleUniverse ;
```

AnimatedSimpleUniverse.java (2)

```
public class AnimatedSimpleUniverse extends JApplet {

    private SimpleUniverse u = null ;

    public AnimatedSimpleUniverse () {

    }

    public static void main (String [] args) {
        new MainFrame (new AnimatedSimpleUniverse (), 256, 256) ;
    }
```

AnimatedSimpleUniverse.java (3)

```
public BranchGroup createSceneGraph () {
    BranchGroup objRoot = new BranchGroup () ;
    TransformGroup objSpin = new TransformGroup () ;
    objSpin.setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;
    objRoot.addChild (objSpin) ;
    objSpin.addChild (new ColorCube (0.2)) ;
    Transform3D yAxis = new Transform3D () ;
    Alpha rotationAlpha = new Alpha (-1, 4000) ;
    RotationInterpolator rotator = new
        RotationInterpolator (rotationAlpha, objSpin, yAxis, 0.0f, (float) Math.PI * 2.0f) ;
    BoundingSphere bounds = new BoundingSphere () ; // origine, rayon 1
    rotator.setSchedulingBounds (bounds) ;
    objSpin.addChild (rotator) ;
    objRoot.compile () ;
    return objRoot ;
}
```

AnimatedSimpleUniverse.java (4)

```
    public void init () {
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration () ;
        Canvas3D c = new Canvas3D (config) ;
        getContentPane ().add (c) ;
        BranchGroup scene = createSceneGraph () ;
        u = new SimpleUniverse (c) ;
        u.getViewingPlatform ().setNominalViewingTransform () ;
        u.addBranchGraph (scene) ;
    }

    public void destroy () {
        u.removeAllLocales () ;
    }

}
```

« PositionInterpolator » (1)

```
import javax.media.j3d.BoundingSphere ;
import javax.media.j3d.PositionInterpolator ;
import javax.media.j3d.Transform3D ;
import javax.media.j3d.TransformGroup ;
import javax.vecmath.AxisAngle4d ;
import javax.vecmath.Point3d ;
import javax.vecmath.Vector3d ;

class ObjetBougeant extends TransformGroup {

   public ObjetBougeant (double x, double y, double z) {
      setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;
      // création du décalage et association au noeud Transform
      Transform3D t3d = new Transform3D () ;
      t3d.setTranslation (new Vector3d (x, y, z)) ;
      setTransform (t3d) ;
```

« PositionInterpolator » (2)

```
// ajout d'un TransformGroup pour la première rotation locale :
// autour de l'axe des y (vertical en position normale ...)
TransformGroup tg1 = new TransformGroup () ;
tg1.setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;
addChild (tg1) ;
BoundingSphere bounds =
                              new BoundingSphere (new Point3d (0.0,0.0,0.0), 100.0) ;
Transform3D xyAxis = new Transform3D () ;
xyAxis.setRotation (new AxisAngle4d (0, 0, 1, Math.PI / 4.0)) ;
Alpha translationAlpha1 = new Alpha (-1, 4000) ;
```

« PositionInterpolator » (3)

```
    PositionInterpolator translator1 =
                        new PositionInterpolator (translationAlpha1, tg1, xyAxis, -2.0f, 2.0f) ;
    translator1.setSchedulingBounds (bounds) ;
    tg1.addChild (translator1) ;
    // ajout d'une géométrie
    tg1.addChild (new ColorCube (0.2)) ;
  }
}
```

« RotationInterpolator » (1)

```
import javax.media.j3d.Alpha ;
import javax.media.j3d.BoundingSphere ;
import javax.media.j3d.RotationInterpolator ;
import javax.media.j3d.Transform3D ;
import javax.media.j3d.TransformGroup ;
import javax.vecmath.AxisAngle4d ;
import javax.vecmath.Point3d ;
import javax.vecmath.Vector3d ;

import com.sun.j3d.utils.geometry.ColorCube ;

class ObjetTournant extends TransformGroup {
```

« RotationInterpolator » (2)

```
public ObjetTournant (double x, double y, double z) {
   setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;

   // création du décalage et association au noeud Transform
   Transform3D t3d = new Transform3D () ;
   t3d.setTranslation (new Vector3d (x, y, z)) ;
   setTransform (t3d) ;

   // ajout d'un TransformGroup pour la première rotation locale :
   // autour de l'axe des y (vertical en position normale ...)
   TransformGroup tg1 = new TransformGroup () ;
   tg1.setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;
   addChild (tg1) ;
```

« RotationInterpolator » (3)

```
BoundingSphere bounds =
                        new BoundingSphere (new Point3d (0.0, 0.0, 0.0), 100.0) ;

Transform3D yAxis = new Transform3D () ;
Alpha rotationAlpha1 = new Alpha (-1, 6000) ;
RotationInterpolator rotator1 =
    new RotationInterpolator (rotationAlpha1, tg1, yAxis, 0.0f, (float)Math.PI * 2.0f) ;
rotator1.setSchedulingBounds (bounds) ;
tg1.addChild (rotator1) ;

// ajout d'un TransformGroup pour la seconde rotation locale :
// on fait tourner l'objet de pi/2 autour de l'axe X :
// pour que la suite tourne autour de l'axe Z ...
TransformGroup tg2 = new TransformGroup () ;
tg1.addChild (tg2) ;
tg2.setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;
```

« RotationInterpolator » (4)

```
    Transform3D zAxis = new Transform3D () ;
    zAxis.setRotation (new AxisAngle4d (1, 0, 0, Math.PI / 2.0)) ;
    Alpha rotationAlpha2 = new Alpha (-1, 3000) ;
    RotationInterpolator rotator2 =
        new RotationInterpolator (rotationAlpha2, tg2, zAxis, 0.0f, (float)Math.PI * 2.0f) ;
    rotator2.setSchedulingBounds (bounds) ;
    tg2.addChild (rotator2) ;

    // ajout d'une géométrie
    tg2.addChild (new ObjetGraphique (0, 0, 0)) ;
  }

}
```

« PositionPathInterpolator » (1)

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

class ObjetPassant extends TransformGroup {

   public ObjetPassant (double x, double y, double z) {
      setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;

      // création du décalage et association au noeud Transform
      Transform3D t3d = new Transform3D () ;
      t3d.setTranslation (new Vector3d (x, y, z)) ;
      setTransform (t3d) ;
```

« PositionPathInterpolator » (2)

```
// ajout d'un TransformGroup pour la première rotation locale :
// autour de l'axe des y (vertical en position normale ...)
TransformGroup tg1 = new TransformGroup () ;
tg1.setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;
addChild (tg1) ;

BoundingSphere bounds =
                              new BoundingSphere (new Point3d (0.0,0.0,0.0), 100.0) ;
Transform3D xyAxis = new Transform3D () ;
Alpha translationAlpha1 = new Alpha (-1, 4000) ;
float noeuds [] = { 0.0f, 0.2f, 0.5f, 1.0f } ;
Point3f positions [] = {  new Point3f (0, 0, 0),
            new Point3f (1, 0, 0),
            new Point3f (1, 1, 0),
            new Point3f (0, 1, 0) } ;
```

« PositionPathInterpolator » (3)

```
    PositionPathInterpolator translator1 =
      new PositionPathInterpolator (translationAlpha1, tg1, xyAxis, noeuds, positions) ;
    translator1.setSchedulingBounds (bounds) ;
    tg1.addChild (translator1) ;

    // ajout d'une géométrie
    tg1.addChild (new ColorCube (0.2)) ;
  }

}
```

« RotPosPathInterpolator » (1)

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

class ObjetDeambulant extends TransformGroup {

   public ObjetDeambulant (double x, double y, double z) {
      setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;

      // création du décalage et association au noeud Transform
      Transform3D t3d = new Transform3D () ;
      t3d.setTranslation (new Vector3d (x, y, z)) ;
      setTransform (t3d) ;
```

« RotPosPathInterpolator » (2)

```
// ajout d'un TransformGroup pour la première rotation locale :
// autour de l'axe des y (vertical en position normale ...)
TransformGroup tg1 = new TransformGroup () ;
tg1.setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;
addChild (tg1) ;

BoundingSphere bounds =
    new BoundingSphere (new Point3d (0.0,0.0,0.0), 100.0) ;

Transform3D axis = new Transform3D () ;
Alpha translationAlpha1 = new Alpha (-1, 10000) ;
float noeuds [] = { 0.0f, 0.25f, 0.5f, 0.75f, 1.0f } ;
```

« RotPosPathInterpolator » (3)

```
Quat4f orientations [] = {  new Quat4f (0.0f, 1.0f, 0.0f, 0.0f),
                new Quat4f ((float)(Math.PI / 2.0f), 0.0f, 1.0f, 0.0f),
                new Quat4f (0.0f, 0.0f, 0.0f, 1.0f),
                new Quat4f ((float)(Math.PI / 2.0f), 0.0f, 1.0f, 0.0f),
                new Quat4f (0.0f, 1.0f, 0.0f, 0.0f) } ;
Point3f positions [] = {  new Point3f (0, 0, 0),
              new Point3f (-1, 0, 0),
              new Point3f (-1, 1, 0),
              new Point3f (0, 1, 0),
              new Point3f (0, 0, 0) } ;
```

« RotPosPathInterpolator » (4)

```
    RotPosPathInterpolator translator1 = new RotPosPathInterpolator
                                (translationAlpha1, tg1, axis, noeuds, orientations, positions) ;
    translator1.setSchedulingBounds (bounds) ;
    tg1.addChild (translator1) ;

    // ajout d'une géométrie
    tg1.addChild (new ColorCube (0.2)) ;
  }

}
```

Les comportements (« Behavior »)

- Ont besoin d’un objet sur lequel agir :
  - souvent un « TransformGroup »
- Doivent être vivants :
  - ajoutés dans le graphe de scène
- Ne sont actifs que lorsque leur « scheduling bounds » intersecte le volume d’activation d’un « ViewPlatform » :
  - attention à bien placer le « Behavior » dans le repère local de l’objet qu’il doit manipuler !
  - sinon il risque d'être perdu...

Les types de comportements prédéfinis

- « Behavior »
  - ✔ « KeyNavigatorBehavior »
  - ✔ « MouseBehavior »
    - ✗ « MouseRotate »
    - ✗ « MouseTranslate »
    - ✗ « MouseZoom »
    - ✗ « MouseWheelZoom »
  - ✔ « PickMouseBehavior »
    - ✗ « PickRotateBehavior »
    - ✗ « PickTranslateBehavior »
    - ✗ « PickZoomBehavior »

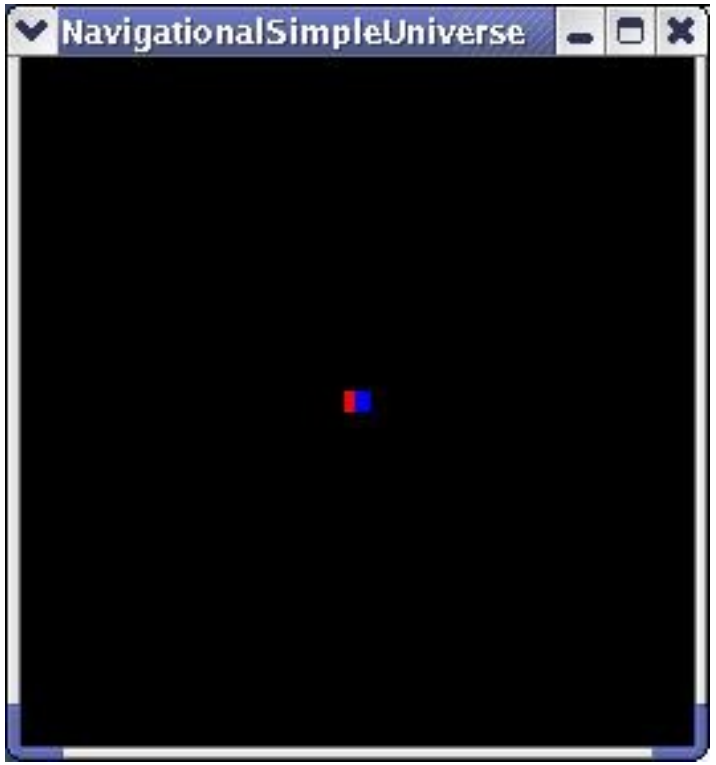
Autoriser la modification d'un objet

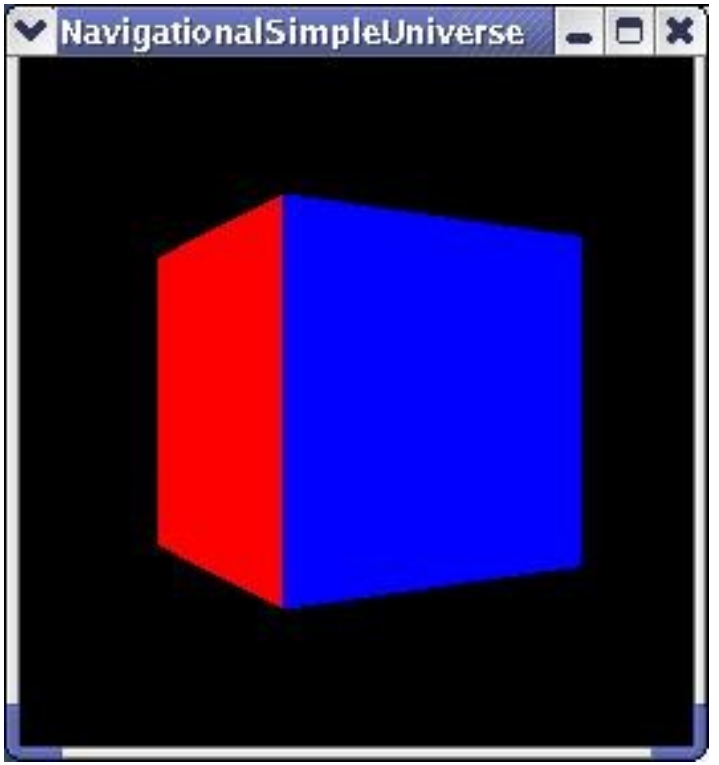
```
class ObjetGraphique extends TransformGroup {
    public ObjetGraphique (double x, double y, double z) {
        setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;
        setCapability (TransformGroup.ALLOW_TRANSFORM_READ) ;
        setCapability (TransformGroup.ENABLE_PICK_REPORTING) ;
        // création du décalage et association au nœud Transform
        Matrix4d matDécalage = new Matrix4d () ;
        matDécalage.setIdentity () ;
        matDécalage.setTranslation (new Vector3d (x, y, z)) ;
        setTransform (new Transform3D (matDécalage)) ;
        // ajout d'une géométrie
        ColorCube cc = new ColorCube (0.2) ;
        cc.getGeometry ().setCapability (Geometry.ALLOW_INTERSECT) ;
        addChild (cc) ;
    }
}
```

Donner des comportements à des objets

```
class GroupeInteractif extends BranchGroup {
    Canvas3D c ;
    ObjetGraphique o ;
    public GroupeInteractif (Canvas3D c, double x, double y, double z) {
        this.c = c ;
        o = new ObjetGraphique (x, y, z) ;
        addChild (o) ;
        addChild (new PickRotateBehavior (this, c,
                                          new BoundingSphere (new Point3d (x, y, z), 0.2))) ;
        addChild (new PickTranslateBehavior (this, c,
                                          new BoundingSphere (new Point3d (x, y, z), 0.2))) ;
        addChild (new PickZoomBehavior (this, c,
                                          new BoundingSphere (new Point3d (x, y, z), 0.2))) ;
    }
}
```

NavigationalSimpleUniverse





NavigationalSimpleUniverse.java (1)

```
import javax.swing.* ;
import java.awt.* ;
import com.sun.j3d.utils.applet.* ;
import com.sun.j3d.utils.geometry.* ;
import com.sun.j3d.utils.universe.* ;
import javax.media.j3d.* ;
import javax.vecmath.* ;
import com.sun.j3d.utils.behaviors.keyboard.KeyNavigatorBehavior ;
```

NavigationalSimpleUniverse.java (2)

```
public class NavigationalSimpleUniverse extends JApplet {

    private SimpleUniverse u = null ;

    public NavigationalSimpleUniverse () {

    }

    public static void main (String [] args) {
        new MainFrame (new NavigationalSimpleUniverse (), 256, 256) ;
    }
```

NavigationalSimpleUniverse.java (3)

```
public BranchGroup createSceneGraph () {
    BranchGroup objRoot = new BranchGroup () ;
     TransformGroup objSpin = new TransformGroup () ;
     objSpin.setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;
     objRoot.addChild (objSpin) ;
     objSpin.addChild (new ColorCube (0.2)) ;
     Transform3D yAxis = new Transform3D () ;
     Alpha rotationAlpha = new Alpha (-1, 4000) ;
     RotationInterpolator rotator = new RotationInterpolator
                                    (rotationAlpha, objSpin, yAxis, 0.0f, (float) Math.PI * 2.0f) ;
     BoundingSphere bounds = new BoundingSphere () ; // origine, rayon 1
     rotator.setSchedulingBounds (bounds) ;
     objSpin.addChild (rotator) ;
     return objRoot ;
 }
```

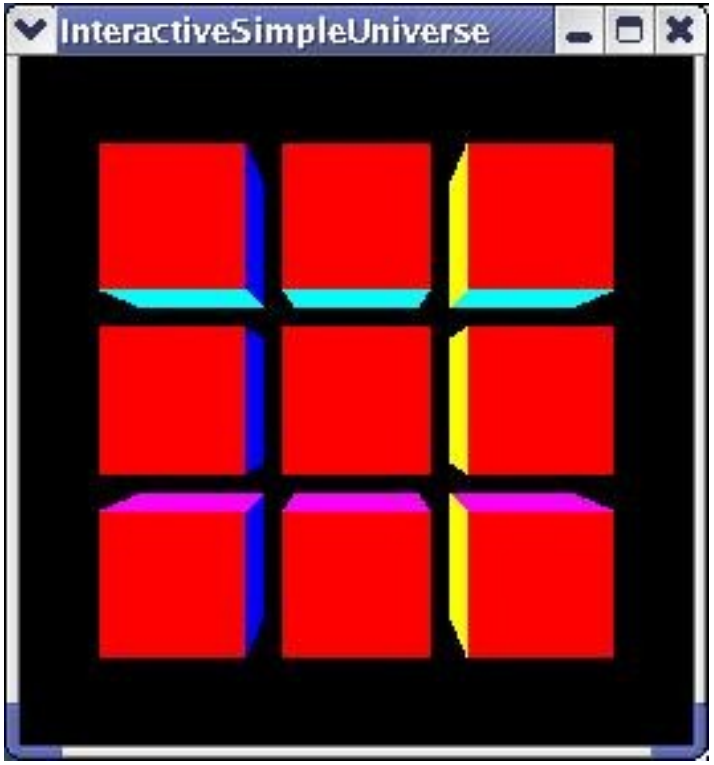
NavigationalSimpleUniverse.java (4)

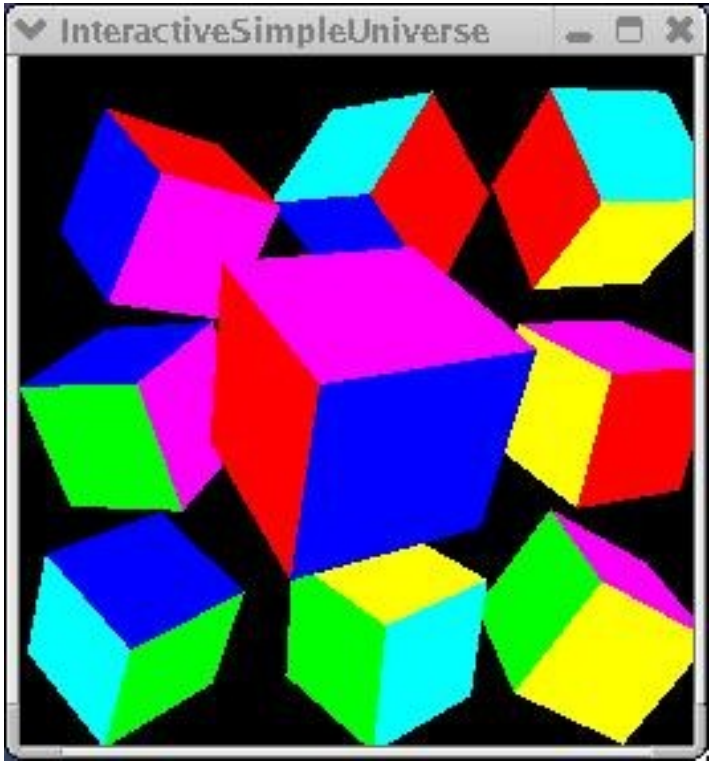
```
 public void destroy () {
    u.removeAllLocales () ;
 }

public void init () {
    GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration () ;
    Canvas3D c = new Canvas3D (config) ;
    getContentPane ().add (c) ;
    u = new SimpleUniverse (c) ;
    BranchGroup scene = createSceneGraph () ;
```

NavigationalSimpleUniverse.java (5)

```
        TransformGroup vpTrans =
            u.getViewingPlatform ().getViewPlatformTransform () ;
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior (vpTrans) ;
        keyNavBeh.setSchedulingBounds (new BoundingSphere (new Point3d (),
                                                           1000.0)) ;
        scene.addChild (keyNavBeh) ;
        scene.compile () ;
        u.addBranchGraph (scene) ;
        u.getViewingPlatform ().setNominalViewingTransform () ;
    }

}
```

InteractiveSimpleUniverse





InteractiveSimpleUniverse.java (1)

```
import javax.swing.* ;
import java.awt.* ;
import com.sun.j3d.utils.applet.* ;
import com.sun.j3d.utils.geometry.* ;
import com.sun.j3d.utils.universe.* ;
import javax.media.j3d.* ;
import javax.vecmath.* ;
import com.sun.j3d.utils.picking.* ;
import com.sun.j3d.utils.picking.behaviors.* ;
```

InteractiveSimpleUniverse.java (2)

```
public class InteractiveSimpleUniverse extends JApplet {

    private SimpleUniverse u = null ;
    private Canvas3D c = null ;

    public InteractiveSimpleUniverse () {

    }

    public static void main (String [] args) {
        new MainFrame (new InteractiveSimpleUniverse (), 256, 256) ;
    }
```

InteractiveSimpleUniverse.java (3)

```
public TransformGroup createColorCube (Vector3d v3d) {
    Transform3D translation = new Transform3D () ;
    translation.setTranslation (v3d) ;
    TransformGroup objTrans = new TransformGroup (translation) ;
    objTrans.setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;
    objTrans.setCapability (TransformGroup.ALLOW_TRANSFORM_READ) ;
    objTrans.setCapability (TransformGroup.ENABLE_PICK_REPORTING) ;
    ColorCube cc = new ColorCube (0.2) ;
    cc.getGeometry ().setCapability (Geometry.ALLOW_INTERSECT) ;
    objTrans.addChild (cc) ;
    return (objTrans) ;
}
```

InteractiveSimpleUniverse.java (4)

```
public BranchGroup createSceneGraph () {
    BranchGroup objRoot = new BranchGroup () ;
    objRoot.addChild (createColorCube (new Vector3d (0, -0.5, 0))) ;
    objRoot.addChild (createColorCube (new Vector3d (-0.5, -0.5, 0))) ;
    objRoot.addChild (createColorCube (new Vector3d (0.5, -0.5, 0))) ;
    objRoot.addChild (createColorCube (new Vector3d (0, 0, 0))) ;
    objRoot.addChild (createColorCube (new Vector3d (-0.5, 0, 0))) ;
    objRoot.addChild (createColorCube (new Vector3d (0.5, 0, 0))) ;
    objRoot.addChild (createColorCube (new Vector3d (0, 0.5, 0))) ;
    objRoot.addChild (createColorCube (new Vector3d (-0.5, 0.5, 0))) ;
    objRoot.addChild (createColorCube (new Vector3d (0.5, 0.5, 0))) ;
    return objRoot ;
}
```

InteractiveSimpleUniverse.java (5)

```
public void enableInteraction (BranchGroup objRoot) {
    BoundingSphere bounds =  new BoundingSphere (new Point3d (0, 0, 0), 100) ;
    PickRotateBehavior prb = new PickRotateBehavior (objRoot, c, bounds) ;
    prb.setMode (PickTool.GEOMETRY) ;
    prb.setTolerance (0.0f) ;
    objRoot.addChild (prb) ;
    PickTranslateBehavior ptb = new PickTranslateBehavior (objRoot, c, bounds) ;
    ptb.setMode (PickTool.GEOMETRY) ;
    ptb.setTolerance (0.0f) ;
    objRoot.addChild (ptb) ;
    PickZoomBehavior pzb = new PickZoomBehavior (objRoot, c, bounds) ;
    pzb.setMode (PickTool.GEOMETRY) ;
    pzb.setTolerance (0.0f) ;
    objRoot.addChild (pzb) ;
}
```

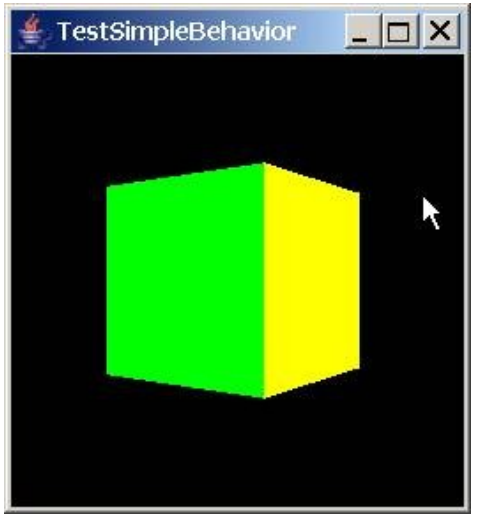
InteractiveSimpleUniverse.java (6)

```
    public void init () {
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration () ;
        c = new Canvas3D (config) ;
        getContentPane ().add (c) ;
        u = new SimpleUniverse (c) ;
        BranchGroup scene = createSceneGraph () ;
        enableInteraction (scene) ;
        u.getViewingPlatform ().setNominalViewingTransform () ;
        scene.compile () ;
        u.addBranchGraph (scene) ;
    }

    public void destroy () {
        u.removeAllLocales () ;
    }
}
```

Créer un « Behavior »

- Écrire un constructeur :
  - ✔ qui stocke une référence sur l'objet à modifier
- Redéfinir la méthode « initialize » :
  - ✔ spécifier le critère initial de déclenchement
- Redéfinir la méthode « processStimulus » :
  - ✔ décoder la condition de déclenchement
  - ✔ agir conformément à cette condition
  - ✔ réinitialiser correctement la condition de déclenchement

TestSimpleBehavior



TestSimpleBehavior.java (1)

```
import java.awt.GraphicsConfiguration ;
import java.awt.event.KeyEvent ;
import java.util.Enumeration ;

import javax.media.j3d.Behavior ;
import javax.media.j3d.BoundingSphere ;
import javax.media.j3d.BranchGroup ;
import javax.media.j3d.Canvas3D ;
import javax.media.j3d.Transform3D ;
import javax.media.j3d.TransformGroup ;
import javax.media.j3d.WakeupOnAWTEvent ;
import javax.swing.JApplet ;

import com.sun.j3d.utils.applet.MainFrame ;
import com.sun.j3d.utils.geometry.ColorCube ;
import com.sun.j3d.utils.universe.SimpleUniverse ;
```

TestSimpleBehavior.java (2)

```
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 () {
        wakeupOn (new WakeupOnAWTEvent (KeyEvent.KEY_PRESSED)) ;
    }
    public void processStimulus (Enumeration criteria) { // examiner criteria …
        angle += 0.1 ;
        rotation.rotY (angle) ;
        targetTG.setTransform (rotation) ;
        wakeupOn (new WakeupOnAWTEvent (KeyEvent.KEY_PRESSED)) ;
    }
}
```

TestSimpleBehavior.java (3)

```
public class TestSimpleBehavior extends JApplet {

    private SimpleUniverse u = null ;

    public TestSimpleBehavior () {

    }

    public static void main (String [] args) {
      new MainFrame (new TestSimpleBehavior (), 256, 256) ;
    }
```

TestSimpleBehavior.java (4)

```
public BranchGroup createSceneGraph () {
    BranchGroup objRoot = new BranchGroup () ;
    TransformGroup objRotate = new TransformGroup () ;
    objRotate.setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;
    objRoot.addChild (objRotate) ;
    objRotate.addChild (new ColorCube (0.4)) ;
    SimpleBehavior myRotationBehavior = new SimpleBehavior (objRotate) ;
    myRotationBehavior.setSchedulingBounds (new BoundingSphere ()) ;
    objRoot.addChild (myRotationBehavior) ;
    objRoot.compile () ;
    return (objRoot) ;
}
```

TestSimpleBehavior.java (5)

```
  public void init () {
      GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration () ;
      Canvas3D c = new Canvas3D (config) ;
      getContentPane ().add (c) ;
      BranchGroup scene = createSceneGraph () ;
      u = new SimpleUniverse (c) ;
      u.getViewingPlatform ().setNominalViewingTransform () ;
      u.addBranchGraph (scene) ;
   }

   public void destroy () {
      u.removeAllLocales () ;
   }

}
```

Déclenchement d'un « Behavior »

- Sur l’occurrence d’un ou plusieurs stimuli
- « WakeupCondition »
  - ✔ « WakeupOr », « WakeupAnd »
  - ✔ « WakeupAndOfOrs », « WakeupOrOfAnds »
  - ✔ « WakeupCriterion »
    - ✗ « WakeupOnActivation », « WakeupOnAWTEvent »
    - ✗ « WakeupOnBehaviorPost », « WakeupOnCollisionEntry »
    - ✗ « WakeupOnCollisionExit », « WakeupOnCollisionMovement »
    - ✗ « WakeupOnDeactivation », « WakeupOnElapsedFrames »
    - ✗ « WakeupOnElapsedTime », « WakeupOnSensorEntry »
    - ✗ « WakeupOnSensorExit », « WakeupOnTransformChange »
    - ✗ « WakeupOnViewPlatformEntry », « WakeupOnViewPlatfornExit »

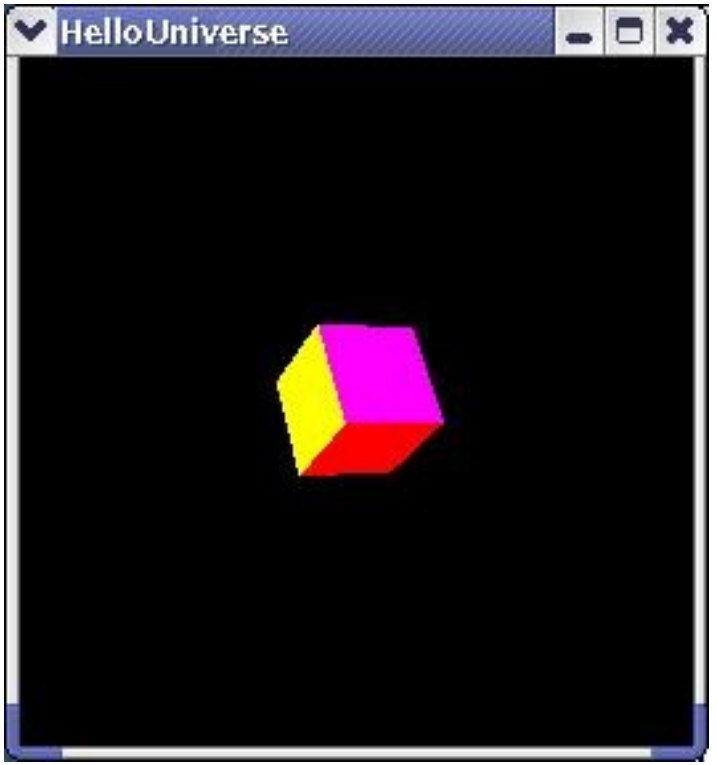
Construction d’un graphe de scène (4)

- Construire une zone d’accueil 3D :
  - Canvas3D canvas = new Canvas3D (...) ;
- Construire un univers :
  - VirtualUnivers universe = new VirtualUniverse () ;
- Construire un repère :
  - Locale locale = new Locale (universe) ;

Construction d'un graphe de scène (5)

- Construire au moins une visualisation :
  - ✔ créer un objet « View »
  - ✔ créer un objet « ViewPlatform »
    - ✗ le nœud du graphe de scène rattaché à la vue
  - ✔ créer un objet « PhysicalBody »
    - ✗ les caractéristiques de la tête de l'utilisateur...
  - ✔ créer un objet « PhysicalEnvironment »
    - ✗ les caractéristiques des dispositifs d'entrées 3D (tracking 3D...)
  - ✔ attacher les objets « ViewPlatform », « PhysicalBody », « PhysicalEnvironment » et « Canvas3D » à l'objet « View »…
  - ✔ créer un objet « ViewingPlatform » (une branche de graphe de scène) et lui associer l'objet « ViewPlatform »
- Associer a visualisation et la scène à un repère de l'univers
  - ✔ locale.addBranchGraph (viewingPlatform) ;
  - ✔ locale.addBranchGraph (sceneBranchGroup) ;

HelloUniverse



HelloUniverse.java (1)

```
import java.awt.GraphicsConfiguration ;
import javax.media.j3d.BranchGroup ;
import javax.media.j3d.Canvas3D ;
import javax.media.j3d.PhysicalBody ;
import javax.media.j3d.PhysicalEnvironment ;
import javax.media.j3d.Transform3D ;
import javax.media.j3d.TransformGroup ;
import javax.media.j3d.View ;
import javax.media.j3d.ViewPlatform ;
import javax.media.j3d.VirtualUniverse ;
import javax.swing.JApplet ;
import javax.vecmath.Vector3d ;
import com.sun.j3d.utils.applet.MainFrame ;
import com.sun.j3d.utils.geometry.ColorCube ;
import com.sun.j3d.utils.universe.SimpleUniverse ;
import com.sun.j3d.utils.universe.ViewingPlatform ;
```

HelloUniverse.java (2)

```
public class HelloUniverse extends JApplet {

    private VirtualUniverse universe = null ;

    public HelloUniverse () {

    }

    public static void main (String [] args) {
        new MainFrame (new HelloUniverse (), 256, 256) ;
    }
```

HelloUniverse.java (3)

```
public BranchGroup createSceneGraph () {
    BranchGroup objRoot = new BranchGroup () ;
    Transform3D orientation = new Transform3D () ;
    orientation.setEuler (new Vector3d (Math.PI / 3, Math.PI / 6, 0)) ;
    TransformGroup objTrans = new TransformGroup (orientation) ;
    objRoot.addChild (objTrans) ;
    objTrans.addChild (new ColorCube (0.2)) ;
    objRoot.compile () ;
    return objRoot ;
}
```

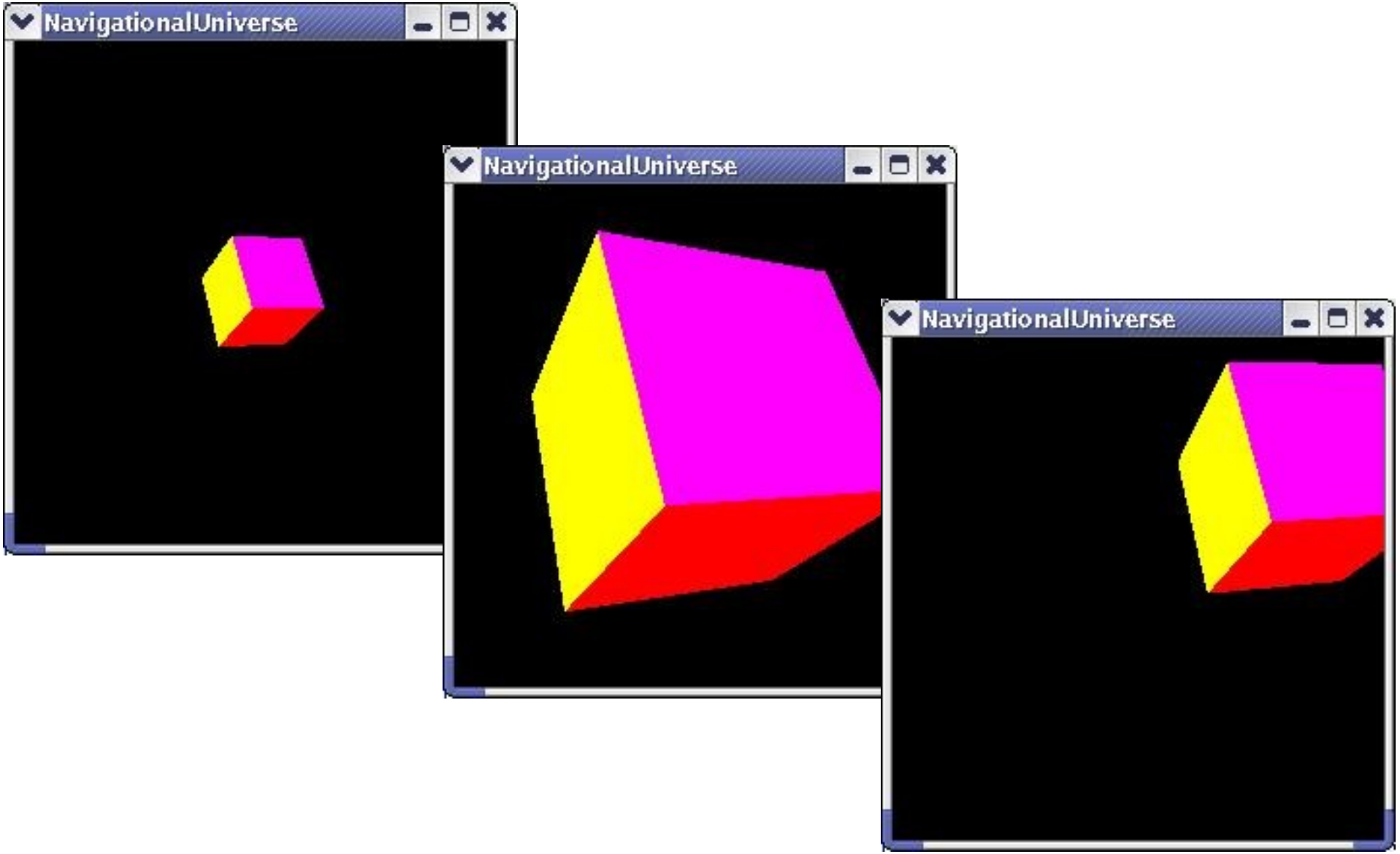
HelloUniverse.java (4)

```
public void init () {
    GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration () ;
    Canvas3D canvas3D = new Canvas3D (config) ;
    getContentPane ().add (canvas3D) ;
    universe = new VirtualUniverse () ;
    javax.media.j3d.Locale locale = new javax.media.j3d.Locale (universe) ;
    ViewPlatform viewPlatform = new ViewPlatform () ;
    PhysicalBody physicalBody = new PhysicalBody () ;
    PhysicalEnvironment physicalEnvironment = new PhysicalEnvironment () ;
    View view = new View () ;
    view.addCanvas3D (canvas3D) ;
    view.setPhysicalBody (physicalBody) ;
    view.setPhysicalEnvironment (physicalEnvironment) ;
    view.attachViewPlatform (viewPlatform) ;
```

HelloUniverse.java (5)

```
        ViewingPlatform viewingPlatform = new ViewingPlatform () ;
        viewingPlatform.setViewPlatform (viewPlatform) ;
        viewingPlatform.setNominalViewingTransform () ;
        viewingPlatform.compile () ;
        BranchGroup scene = createSceneGraph () ;
        locale.addBranchGraph (viewingPlatform) ;
        locale.addBranchGraph (scene) ;
    }

    public void destroy () {
        universe.removeAllLocales () ;
    }

}
```

NavigationalUniverse



NavigationalUniverse.java (1)

```
import java.awt.GraphicsConfiguration ;

import javax.media.j3d.* ;
import javax.swing.JApplet ;
import javax.vecmath.Point3d ;
import javax.vecmath.Vector3d ;

import com.sun.j3d.utils.applet.MainFrame ;
import com.sun.j3d.utils.behaviors.keyboard.KeyNavigatorBehavior ;
import com.sun.j3d.utils.geometry.ColorCube ;
import com.sun.j3d.utils.universe.SimpleUniverse ;
import com.sun.j3d.utils.universe.ViewingPlatform ;
```

NavigationalUniverse.java (2)

```
public class NavigationalUniverse extends JApplet {

    private VirtualUniverse universe = null ;

    public NavigationalUniverse () {

    }

    public static void main (String [] args) {
        new MainFrame (new NavigationalUniverse (), 256, 256) ;
    }
```

NavigationalUniverse.java (3)

```
public BranchGroup createSceneGraph () {
    BranchGroup objRoot = new BranchGroup () ;
    Transform3D orientation = new Transform3D () ;
    orientation.setEuler (new Vector3d (Math.PI / 3, Math.PI / 6, 0)) ;
    TransformGroup objTrans = new TransformGroup (orientation) ;
    objRoot.addChild (objTrans) ;
    objTrans.addChild (new ColorCube (0.2)) ;
    return objRoot ;
}
```

NavigationalUniverse.java (4)

```
public void init () {
    GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration () ;
    Canvas3D canvas3D = new Canvas3D (config) ;
    getContentPane ().add (canvas3D) ;
    universe = new VirtualUniverse () ;
    javax.media.j3d.Locale locale = new javax.media.j3d.Locale (universe) ;
    ViewPlatform viewPlatform = new ViewPlatform () ;
    PhysicalBody physicalBody = new PhysicalBody () ;
    PhysicalEnvironment physicalEnvironment = new PhysicalEnvironment () ;
    View view = new View () ;
    view.addCanvas3D (canvas3D) ;
    view.setPhysicalBody (physicalBody) ;
    view.setPhysicalEnvironment (physicalEnvironment) ;
    view.attachViewPlatform (viewPlatform) ;
```

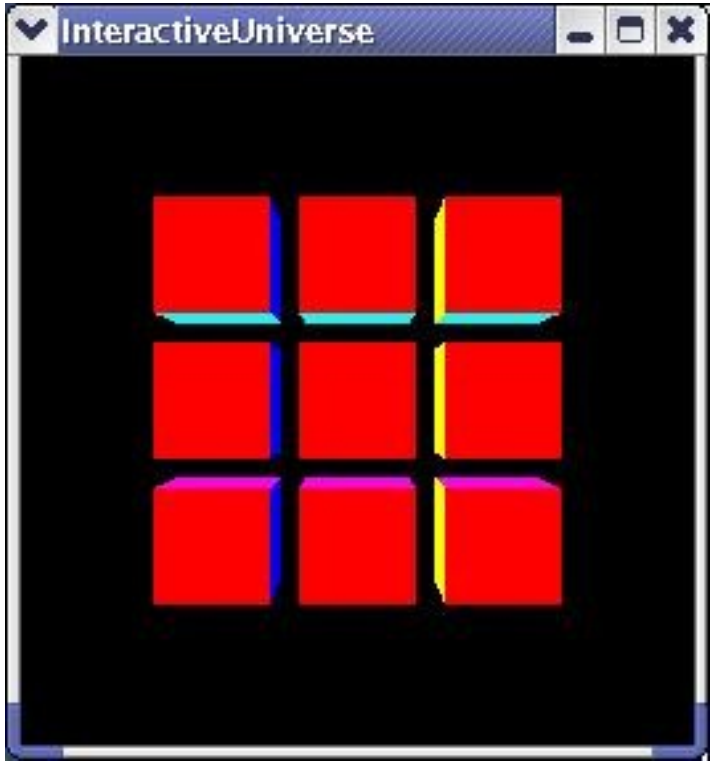
NavigationalUniverse.java (5)

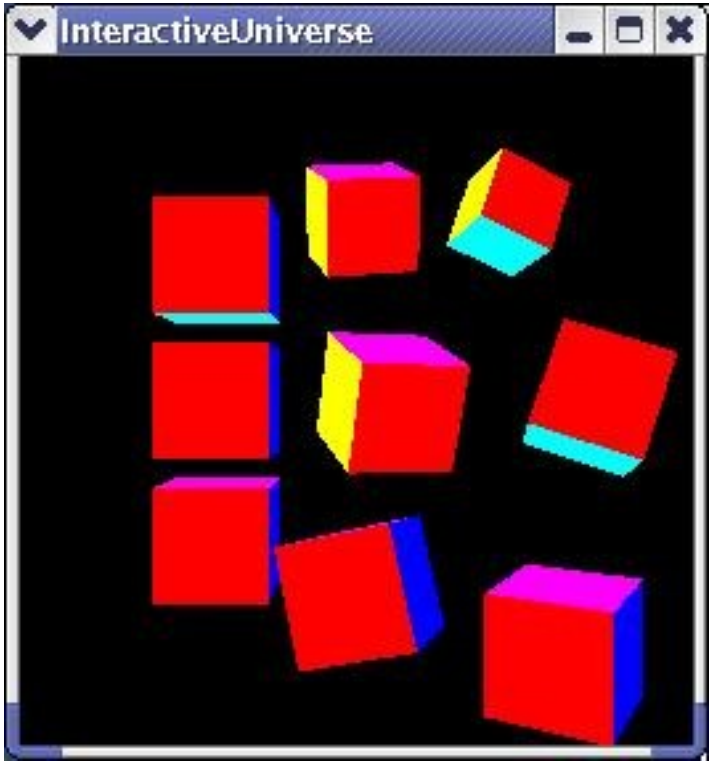
```
ViewingPlatform viewingPlatform = new ViewingPlatform () ;
viewingPlatform.setViewPlatform (viewPlatform) ;
TransformGroup viewPointTransform =
                                        viewingPlatform.getViewPlatformTransform () ;
viewPointTransform.setCapability
                                    (TransformGroup.ALLOW_TRANSFORM_READ) ;
viewPointTransform.setCapability
                                   (TransformGroup.ALLOW_TRANSFORM_WRITE) ;
viewingPlatform.setNominalViewingTransform () ;
viewingPlatform.compile () ;
```

NavigationalUniverse.java (6)

```
        BranchGroup scene = createSceneGraph () ;
        KeyNavigatorBehavior keyNavBeh =
                                        new KeyNavigatorBehavior (viewPointTransform) ;
        keyNavBeh.setSchedulingBounds (new BoundingSphere (new Point3d (),
                                                            1000.0)) ;
        scene.addChild (keyNavBeh) ;
        scene.compile () ;
        locale.addBranchGraph (viewingPlatform) ;
        locale.addBranchGraph (scene) ;
    }

    public void destroy () {
        universe.removeAllLocales () ;
    }

}
```

InteractiveUniverse





InteractiveUniverse.java (1)

```
import java.awt.GraphicsConfiguration ;

import javax.media.j3d.* ;
import javax.swing.JApplet ;
import javax.vecmath.Point3d ;
import javax.vecmath.Vector3d ;

import com.sun.j3d.utils.applet.MainFrame ;
import com.sun.j3d.utils.geometry.ColorCube ;
import com.sun.j3d.utils.picking.PickTool ;
import com.sun.j3d.utils.picking.behaviors.PickRotateBehavior ;
import com.sun.j3d.utils.picking.behaviors.PickTranslateBehavior ;
import com.sun.j3d.utils.picking.behaviors.PickZoomBehavior ;
import com.sun.j3d.utils.universe.SimpleUniverse ;
import com.sun.j3d.utils.universe.ViewingPlatform ;
```

InteractiveUniverse.java (2)

```
public class InteractiveUniverse extends JApplet {

    private VirtualUniverse universe = null ;

    public InteractiveUniverse () {

    }

    public static void main (String [] args) {
        new MainFrame (new InteractiveUniverse (), 256, 256) ;
    }
```

InteractiveUniverse.java (3)

```
public TransformGroup createColorCube (Vector3d v3d) {
    Transform3D translation = new Transform3D () ;
    translation.setTranslation (v3d) ;
    TransformGroup objTrans = new TransformGroup (translation) ;
    objTrans.setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;
    objTrans.setCapability (TransformGroup.ALLOW_TRANSFORM_READ) ;
    objTrans.setCapability (TransformGroup.ENABLE_PICK_REPORTING) ;
    ColorCube cc = new ColorCube (0.2) ;
    cc.getGeometry ().setCapability (Geometry.ALLOW_INTERSECT) ;
    objTrans.addChild (cc) ;
    return (objTrans) ;
}
```

InteractiveUniverse.java (4)

```
public BranchGroup createSceneGraph () {
    BranchGroup objRoot = new BranchGroup () ;
    objRoot.addChild (createColorCube (new Vector3d (0, -0.5, 0))) ;
    objRoot.addChild (createColorCube (new Vector3d (-0.5, -0.5, 0))) ;
    objRoot.addChild (createColorCube (new Vector3d (0.5, -0.5, 0))) ;
    objRoot.addChild (createColorCube (new Vector3d (0, 0, 0))) ;
    objRoot.addChild (createColorCube (new Vector3d (-0.5, 0, 0))) ;
    objRoot.addChild (createColorCube (new Vector3d (0.5, 0, 0))) ;
    objRoot.addChild (createColorCube (new Vector3d (0, 0.5, 0))) ;
    objRoot.addChild (createColorCube (new Vector3d (-0.5, 0.5, 0))) ;
    objRoot.addChild (createColorCube (new Vector3d (0.5, 0.5, 0))) ;
    return objRoot ;
}
```

InteractiveUniverse.java (5)

```
public void enableInteraction (BranchGroup objRoot, Canvas3D c) {
    BoundingSphere bounds =  new BoundingSphere (new Point3d (0, 0, 0), 100) ;
    PickRotateBehavior prb = new PickRotateBehavior (objRoot, c, bounds) ;
    prb.setMode (PickTool.GEOMETRY) ;
    prb.setTolerance (0.0f) ;
    objRoot.addChild (prb) ;
    PickTranslateBehavior ptb = new PickTranslateBehavior (objRoot, c, bounds) ;
    ptb.setMode (PickTool.GEOMETRY) ;
    ptb.setTolerance (0.0f) ;
    objRoot.addChild (ptb) ;
    PickZoomBehavior pzb = new PickZoomBehavior (objRoot, c, bounds) ;
    pzb.setMode (PickTool.GEOMETRY) ;
    pzb.setTolerance (0.0f) ;
    objRoot.addChild (pzb) ;
}
```

InteractiveUniverse.java (6)

```
public void init () {
    GraphicsConfiguration config =
      SimpleUniverse.getPreferredConfiguration () ;
    Canvas3D canvas3D = new Canvas3D (config) ;
    getContentPane ().add (canvas3D) ;
    universe = new VirtualUniverse () ;
    javax.media.j3d.Locale locale = new javax.media.j3d.Locale (universe) ;
    ViewPlatform viewPlatform = new ViewPlatform () ;
    PhysicalBody physicalBody = new PhysicalBody () ;
    PhysicalEnvironment physicalEnvironment = new PhysicalEnvironment () ;
```

InteractiveUniverse.java (7)

```
View view = new View () ;
view.addCanvas3D (canvas3D) ;
view.setPhysicalBody (physicalBody) ;
view.setPhysicalEnvironment (physicalEnvironment) ;
view.attachViewPlatform (viewPlatform) ;
ViewingPlatform viewingPlatform = new ViewingPlatform () ;
viewingPlatform.setViewPlatform (viewPlatform) ;
TransformGroup viewPointTransform =
                                        viewingPlatform.getViewPlatformTransform () ;
viewPointTransform.setCapability
                                    (TransformGroup.ALLOW_TRANSFORM_READ) ;
viewPointTransform.setCapability
                                   (TransformGroup.ALLOW_TRANSFORM_WRITE) ;
viewingPlatform.setNominalViewingTransform () ;
viewingPlatform.compile () ;
```

InteractiveUniverse.java (8)

```
        BranchGroup scene = createSceneGraph () ;
        enableInteraction (scene, canvas3D) ;
        scene.compile () ;
        locale.addBranchGraph (viewingPlatform) ;
        locale.addBranchGraph (scene) ;
        viewPointTransform.setTransform (recul) ;
    }

    public void destroy () {
        universe.removeAllLocales () ;
    }

}
```

« SceneGraphObject » : les méthodes

- final boolean getCapabilities (int bit) ;
- final void setCapability (int bit) ;
- final void clearCapability (int bit) ;
- final boolean isCompiled () ;
- final boolean isLive () ;
- void setUserData (Object userData) ;
- Object getUserData (Object userData) ;

La classe « Node » : les méthodes

- final void setBounds (Bounds region) ;
- final void setBoundsAutoCompute (boolean autocompute) ;
- final void getLocalToWorld (…) ;
- Node cloneTree (…) ;
- Node duplicateNode (Node original, boolean forceDuplicate) ;
- void setPickable (boolean pickable) ;

La classe « Node » : caractéristiques (capabilities)

- ALLOW_BOUNDS_READ | WRITE
- ALLOW_AUTO_COMPUTE_BOUNDS_READ | WRITE
- ENABLE_PICK_REPORTING
- ALLOW_PICKABLE_READ | WRITE
- ALLOW_COLLISION_REPORTING
- ALLOW_COLLIDABLE_READ | WRITE
- ALLOW_LOCALE_TO_VWORLD_READ

La classe « Group » : caractéristiques et méthodes

- ALLOW_CHILDREN_READ autorise :
  - final Node getChild (int index) ;
  - final int numChildren () ;
- ALLOW_CHILDREN_WRITE autorise :
  - final void setChild (Node child, int index) ;
  - final void insertChild (Node child, int index) ;
  - final void removeChild (int index) ;
- ALLOW_CHILDREN_EXTEND autorise :
  - final void addChild (Node child) ;
  - final void moveTo (BranchGroup branchGroup) ;

Quelques dérivés de « Group »

- « BranchGroup » :
  - branche d'un graphe de scène, à insérer dans une « Locale »
- « OrderedGroup » :
  - les fils sont parcourus dans l'ordre de définition
- « SharedGroup » :
  - destiné à être partagé par plusieurs nœuds feuille « Link »
- « Switch » :
  - contrôle celui ou ceux des fils qui seront affichés
- « TransformGroup » :
  - contient une transformation appliquée aux nœuds fils

Quelques dérivés de « Leaf » (1)

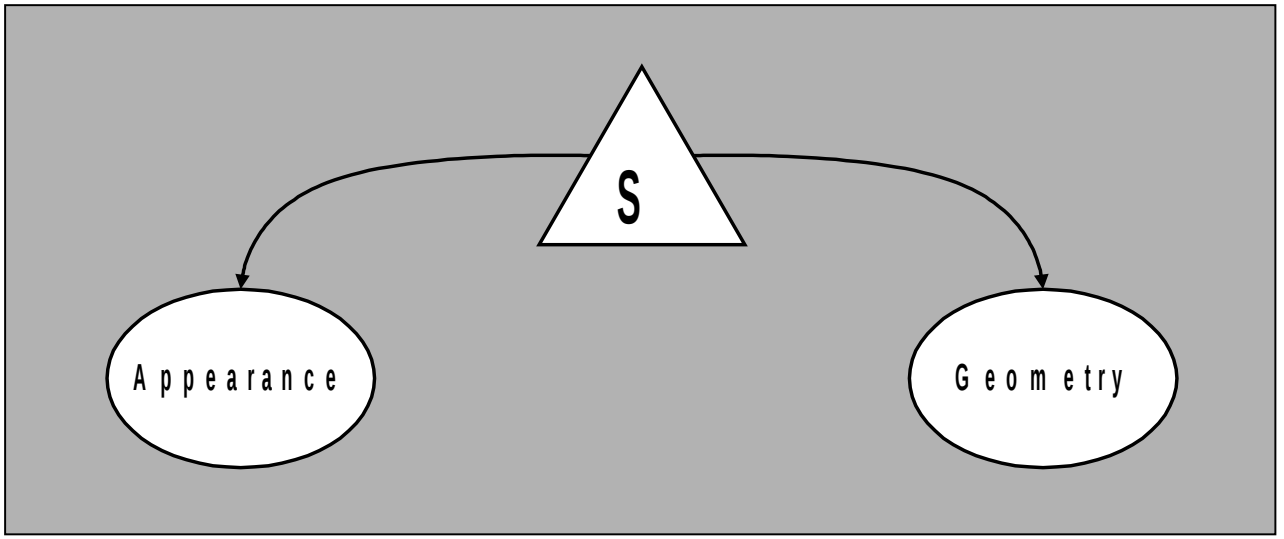
- « Background » :
  - couleur ou image de fond
- « Behavior » :
  - les comportement d’animation ou d’interaction
- « BoundingLeaf » :
  - région englobante pouvant être référencée par d’autres noeuds
- « Clip » :
  - définition des distances de clipping avant et arrière dans un univers
- « Light »
  - « AmbientLight » « DirectionalLight » « PointLight » « SpotLight »
- « Fog »

Quelques dérivés de « Leaf » (2)

- « Link » :
  - pour référencer un graphe partagé
- « Shape3D » :
  - tous les objets géométriques
- « Sound » :
  - « BackgroundSound » « PointSound » « ConeSound »
  - définition des propriétés des sources sonores
- « ViewPlatform » :
  - contrôle des paramètres de l'observateur :
    - position
    - orientation
    - zoom

Géométrie : « Shape3D »

- Le nœud feuille « Shape3D » permet de construire l'ensemble des nœuds géométriques

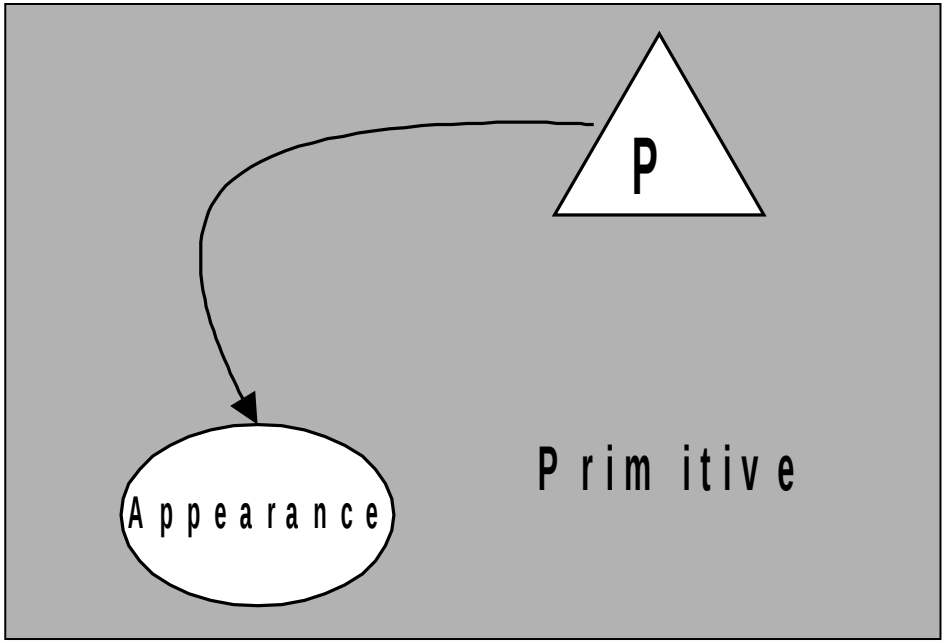


« Shape3D »

- Géométrie (« Geometry »)
  - ✔ les coordonnées
  - ✔ la construction
  - ✔ les classes
- Apparence (« Appearance »)
  - ✔ les couleurs
  - ✔ les matériaux
  - ✔ les attributs

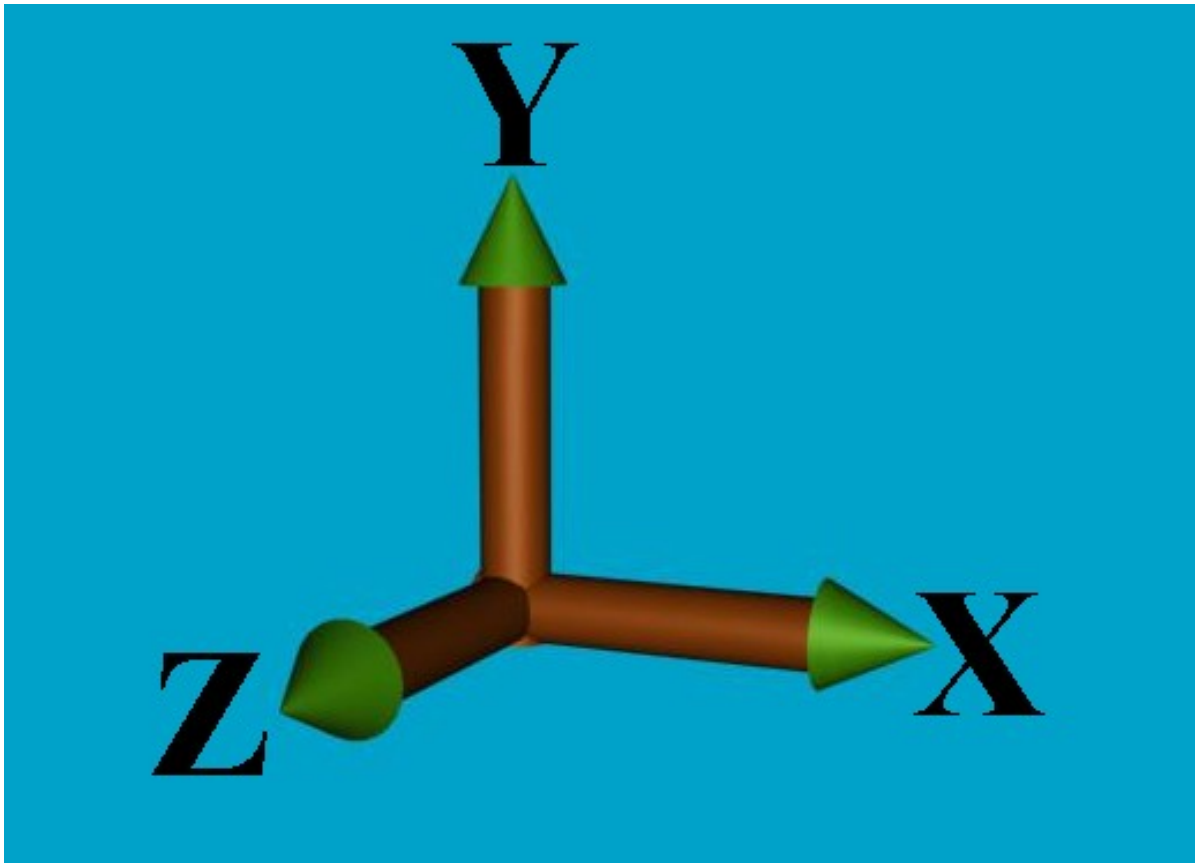
« Primitive »

- Une collection de primitives permet la création de formes simples :
  - ✔ « Cylinder »
  - ✔ « Cone »
  - ✔ « Box »
  - ✔ « Sphere »



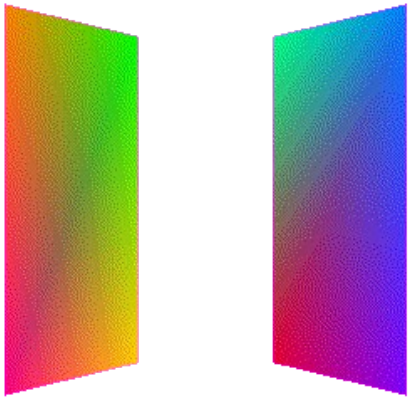
Les coordonnées

- Repère direct
  - X gauche_droite
  - Y bas-haut
  - Z arrière-devant
- Unités
  - mètre
  - sur 256 bits



Géométrie : la construction (1)

- Utiliser les classes « Shape3D » :
  - ✔ « Geometry »
    - ✗ « GeometryArray »
    - ✗ « IndexedGeometryStripArray »
      - « Point », « Line », « Triangle », « Quad »
  - ✔ « Appearance »
    - ✗ Référencent d'autres objets attributs
- Utiliser les chargeurs :
  - ✔ 3DS, DEM, DXF, LWS, OBJ, WRL...



Géométrie : la construction (2)

- Simples (simple geometry)
  - « PointArray », « LineArray », « TriangleArray », « QuadArray »
- En bandes (strip geometry)
  - « LineStripArray », « TriangleStripArray », « TriangleFanArray »
- Simples indexées (indexed geometry)
  - « IndexedPointArray », « IndexedLineArray »,
    « IndexedTriangleArray », « IndexedQuadArray »
- En bandes indexées (indexed strip geometry)
  - « IndexedLineStripArray », « IndexedTriangleStripArray »,
    « IndexedTriangleFanArray »

Géométrie : les points (1)

- Un point (vertex)
  - ✔ coordonnées spatiales
  - ✔ couleur
  - ✔ coordonnées de texture
  - ✔ vecteur normal

Geometry : les points (2)

Geometry : les points (3)

```
public class GroupeDePoints extends Shape3D {
    public GroupeDePoints () {
        float vert [] = new float [300000] ;
        float color [] = new float [300000] ;

        for (int i = 0 ; i < 100000 ; i += 3) {
            vert [i] =   (float)Math.random () - .5f ;
            vert [i+1] = (float)Math.random () - .5f ;
            vert [i+2] = (float)Math.random () - .5f ;
            color [i]   = (float)Math.random () ;
            color [i+1] = (float)Math.random () ;
            color [i+2] = (float)Math.random () ;
        }
```

Geometry : les points (4)

```
        PointArray point =
           new PointArray (100000, PointArray.COORDINATES | PointArray.COLOR_3) ;
        point.setCoordinates (0, vert) ;
        point.setColors (0, color) ;
        this.setGeometry (point) ;
    }
}
```

« Appearance » : définition (1)

- L’apparence contrôle les attributs de couleur et transparence :
  - ✔ Colorisation :
    - ✗ constante ou dégradée (shading)
  - ✔ Matériaux :
    - ✗ couleurs ambiante, diffuse, émissive et spéculaire
    - ✗ Éclairement
  - ✔ « GeometryArray » et « IndexedGeometryArray » :
    - ✗ couleurs par coordonnées
  - ✔ « TransparencyAttributes »
    - ✗ quantité et mode de transparence

« Appearance » : définition (2)

- Rendu
  - ✔ « PointAttributes »
    - ✗ taille et anticrénelage
  - ✔ « LineAttributes »
    - ✗ largeur, pattern et anticrénelage
  - ✔ « PolygonAttributes »
    - ✗ culling et style de tracé
  - ✔ « RenderingAttributes »
    - ✗ utilisation des tampons de profondeur et de transparence ou d'opacité (alpha)

« Appearance » : définition (3)

- « Texture »
  - « Texture2D », « Texture 3D »
  - « TextureAttributes »
  - « TexCoordGeneration »

« Appearance » : « ColoringAttributes »

- Couleur de l’objet lorsque la lumière est éteinte
- Mode de lissage (à plat ou Gouraud)
- Utiliser ces attributs lorsqu’une forme n’est pas lissée :
  - ✔ réduit les calculs de rendu

« Appearance » : « Material »

- Contrôle :
  - ✔ les couleurs ambiante, diffuse, émissive et spéculaire
  - ✔ le facteur de brillance (shininess)
- Utiliser un « Material » lorsque la forme est lissée
- Surdéfini les « ColoringAttributes » lorsque la scène est éclairée

« Appearance » : les attributs (1)

- Transparence
  - ✔ intensité
  - ✔ mode
    - ✗ SCREEN_DOOR, BLENDED, NONE, FASTEST, NICEST
- Attributs de rendu
  - ✔ utilisation du Depth buffer
  - ✔ utilisation du Alpha buffer

« Appearance » : les attributs (2)

- Attributs de points
  - ✔ « PointAttributes »
    - ✗ taille des points
    - ✗ anticrénelage
- Attributs de lignes
  - ✔ « LineAttributes «
    - ✗ épaisseur
    - ✗ tiretée/pointillée
    - ✗ anticrénelage

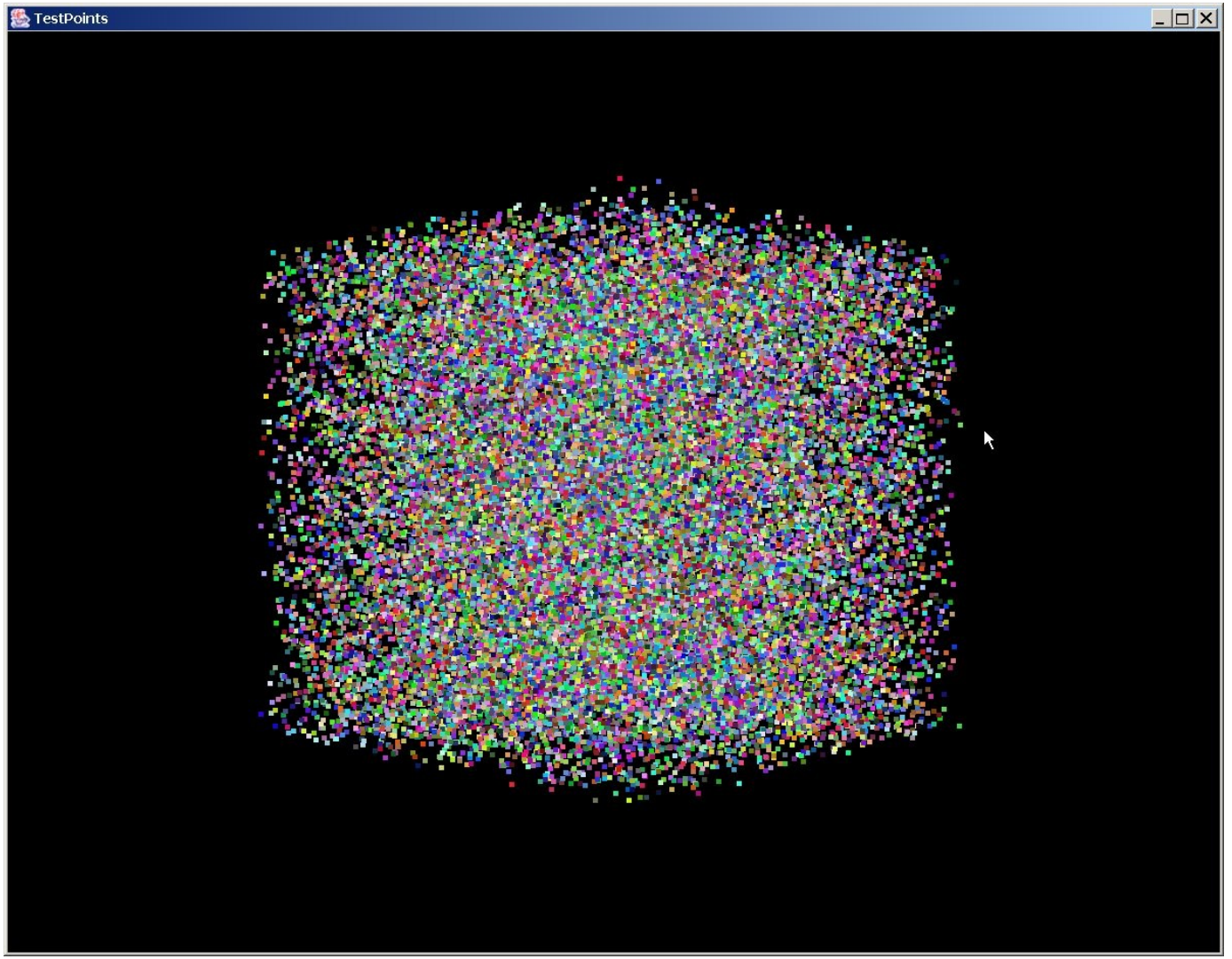
« Appearance » : les attributs (3)

- Attributs de polygones
  - ✔ Face culling
    - ✗ devant, arrière, aucun
  - ✔ Mode de remplissage
    - ✗ point, ligne, continu
  - ✔ Z offset

« Appearance » : application

- Instancier l’objet « Geometry »
- Instancier un objet « Appearance »
  - ✔ instancier un objet « Material »
  - ✔ instancier le ou les objets « XXXAttributes »
    - ✗ initialiser les objets « Material » et « XXXAttributes »
  - ✔ modifier l’objet « Appearance »
- Référencer les objets « Geometry » et « Appearance » auprès de l’objet « Shape3D »

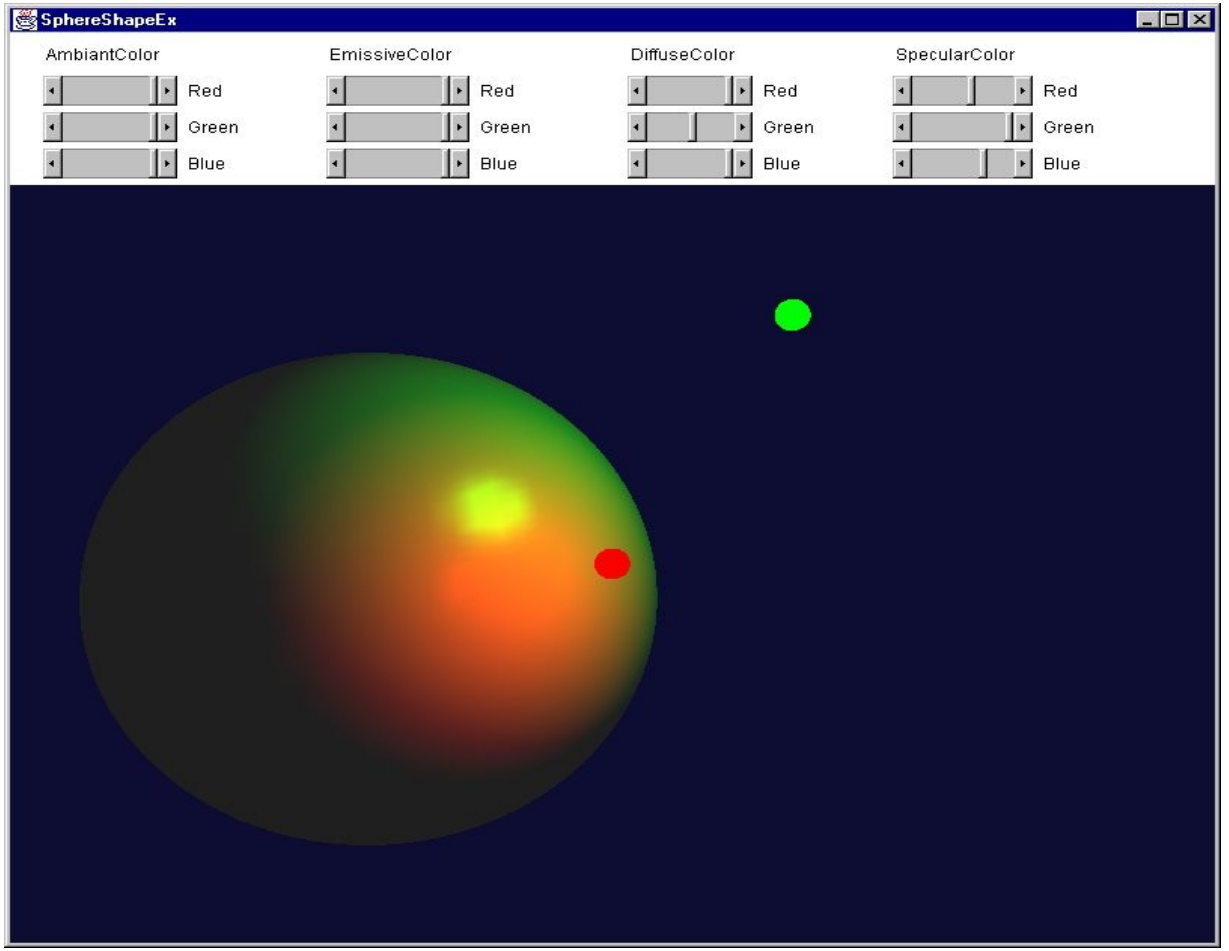
« Appearance » : les points



« Appearance » : les points

```
PointArray point = new PointArray (100000,
                                   PointArray.COORDINATES | PointArray.COLOR_3 ) ;
point.setCoordinates (0, vert) ;
point.setColors (0, color) ;

Appearance ap = new Appearance () ;
PointAttributes pta = new PointAttributes () ;
pta.setPointSize (5.0f) ;
ap.setPointAttributes (pta) ;
this.setAppearance (ap) ;
this.setGeometry (point) ;
```

« Appearance » : les attributs (1)



« Appearance » : les attributs (2)

```
public SphereShape1 (float radius, Canvas3D canvas) {
    super (radius, Sphere.ENABLE_APPEARANCE_MODIFY |
                   Sphere.GENERATE_NORMALS, 45) ;
    this.canvas = canvas ;
    Appearance ap = this.getAppearance () ;

    material = new Material () ;
    material.setCapability (Material.ALLOW_COMPONENT_WRITE) ;
    material.setAmbientColor (1.0f, 1.0f, 0.3f) ;
    material.setDiffuseColor (1.0f, 1.0f, 0.3f) ;
    material.setEmissiveColor (1.0f, 0.5f, 0.0f) ;
    material.setSpecularColor (.0f, 1.0f, 1.0f) ;
    material.setShininess (0.4f) ;
    material.setLightingEnable (true) ;
    ap.setMaterial (material) ;
```

« Appearance » : les attributs (3)

```
    TransparencyAttributes ta = new TransparencyAttributes () ;
    ta.setTransparency (0.9f) ;
    ta.setTransparencyMode (TransparencyAttributes.BLENDED) ;

    ap.setTransparencyAttributes (ta) ;

    this.setAppearance (ap) ;
}
```

Géométrie : les chargeurs existants (1)

- 3DS 3DStudio
- COB Caliguri trueSpace
- DEM Digital Elevation Map
- DXF Autocad
- IOB Imagine
- LWS Lightwave Scene Format
- NFF WorldToolkit NFF format

Géométrie : les chargeurs existants (2)

- OBJ Wavefront
- PDB Protein Data Bank
- PLAY PLAY
- SLD Solid Works
- VRT Superscape VRT
- VTK Visual Toolkit
- WRL Virtual Reality Modeling Language

Géométrie : utilisation des chargeurs

- Rechercher un chargeur ou l'écrire
- Importer les classes nécessaires
- Déclarer une variable de type Scene
- Instancier un chargeur
- Charger le fichier dans un bloc try catch
- Assigner le résultat à la variable scène
- Insérer la scène dans le graphe de scène

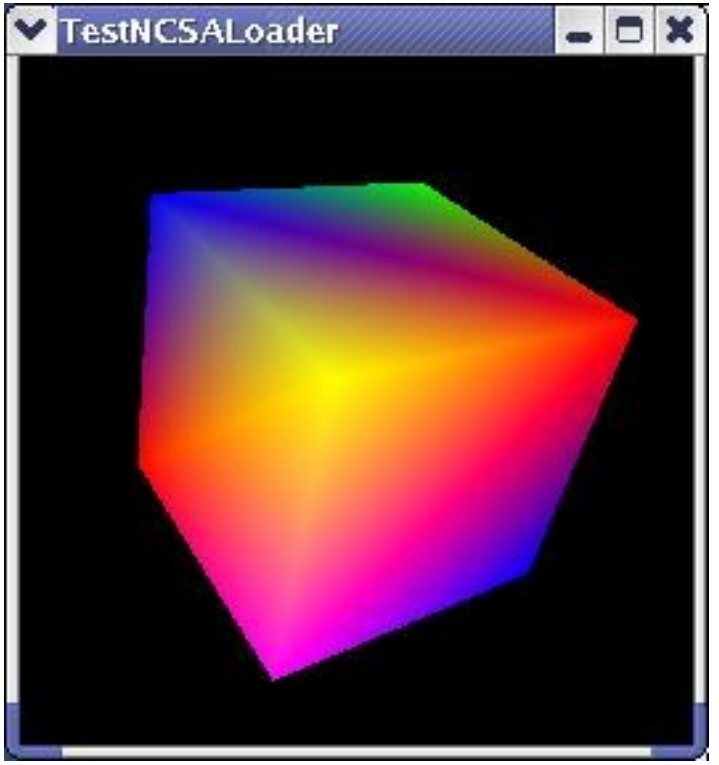
Les chargeurs : exemple

```
import org.jdesktop.j3d.loaders.vrml97.VrmlLoader ;
    …

public BranchGroup createSceneGraph () {
   BranchGroup objRoot = new BranchGroup () ;
   VrmlLoader loader = new VrmlLoader () ;
   try {
      Scene scene = loader.load (fileName) ;
      objRoot = scene.getSceneGroup () ;
      } catch (FileNotFoundException e) {
         e.printStackTrace () ;
      }
   return objRoot ;
}
```

Chargeurs et interaction (1)

- Il faut pouvoir déterminer des « capabilities »
  - il faut donc connaître la structure des objets chargés
  - c'était indispensable avec Java3D 1.3
  - ça ne l'est plus avec Java3D 1.5.2
- L’exemple suivant présente le cas du chargement d’un nœud « IndexedFaceSet » VRML

Chargeurs et interaction (2)



TestIndexedFaceSetVrmlLoader.java (1)

```
import com.sun.j3d.loaders.Scene ;
import java.io.FileNotFoundException ;
import javax.media.j3d.* ;

import org.jdesktop.j3d.loaders.vrml97.VrmlLoader ;

import java.awt.GraphicsConfiguration ;
import javax.media.j3d.Canvas3D ;
import javax.swing.JApplet ;
import javax.vecmath.Point3d ;
import com.sun.j3d.utils.applet.MainFrame ;
import com.sun.j3d.utils.picking.PickTool ;
import com.sun.j3d.utils.picking.behaviors.PickRotateBehavior ;
import com.sun.j3d.utils.picking.behaviors.PickTranslateBehavior ;
import com.sun.j3d.utils.picking.behaviors.PickZoomBehavior ;
import com.sun.j3d.utils.universe.SimpleUniverse ;
```

TestIndexedFaceSetVrmlLoader.java (2)

```
public class TestVrmlLoader extends JApplet {

   private static final long serialVersionUID = 1L ;
   private String fileName ;
   private SimpleUniverse u ;

   public TestVrmlLoader (String fileName) {
      this.fileName = fileName ;
   }

   public static void main (String [] args) {
      new MainFrame (new TestVrmlLoader (args [0]), 256, 256) ;
   }
```

TestIndexedFaceSetVrmlLoader.java (3)

```
public void destroy () {
   u.removeAllLocales () ;
}

public void init () {
   GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration () ;
   Canvas3D c = new Canvas3D (config) ;
   getContentPane ().add (c) ;
   BranchGroup scene = createSceneGraph () ;
   u = new SimpleUniverse (c) ;
   u.getViewingPlatform ().setNominalViewingTransform () ;
   enableInteraction (scene, c) ;
   scene.compile () ;
   u.addBranchGraph (scene) ;
}
```

TestIndexedFaceSetVrmlLoader.java (4)

```
public BranchGroup createSceneGraph () {
   BranchGroup objRoot = new BranchGroup () ;
   TransformGroup objTrans = new TransformGroup () ;
   objTrans.setCapability (TransformGroup.ALLOW_TRANSFORM_READ) ;
   objTrans.setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;
   objTrans.setCapability (TransformGroup.ENABLE_PICK_REPORTING) ;
   objRoot.addChild (objTrans) ;
   VrmlLoader loader = new VrmlLoader () ;
   try { // cas d'un IndexedFaceSet
      Scene scene = loader.load (fileName) ;
      objTrans.addChild (scene.getSceneGroup ()) ;
      Shape3D shape = (Shape3D)
                        (((Group)scene.getSceneGroup ().getChild (0)).getChild (0)) ;
      shape.getGeometry ().setCapability (Geometry.ALLOW_INTERSECT) ;
   } catch (FileNotFoundException e) { e.printStackTrace () ; }
   return objRoot ;
}
```

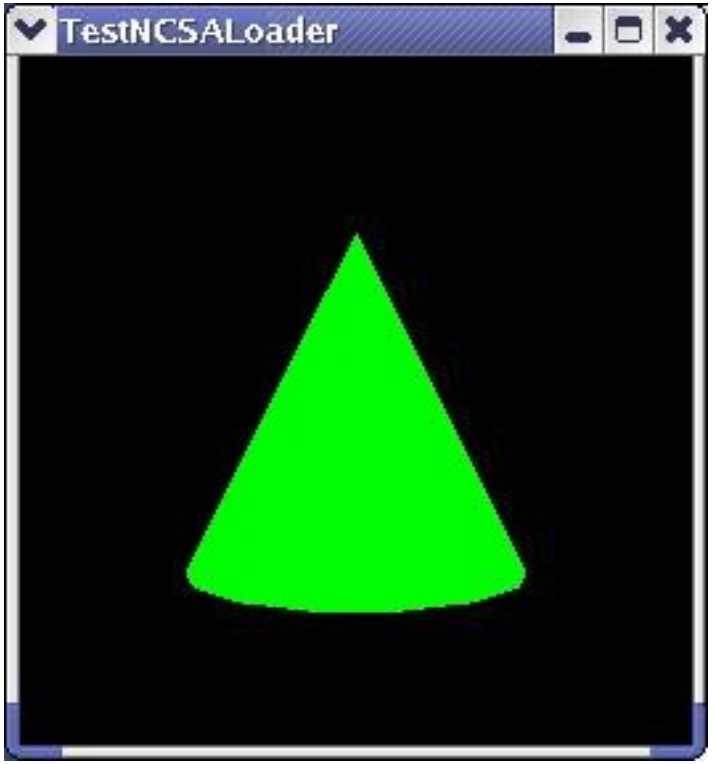
TestIndexedFaceSetVrmlLoader.java (5)

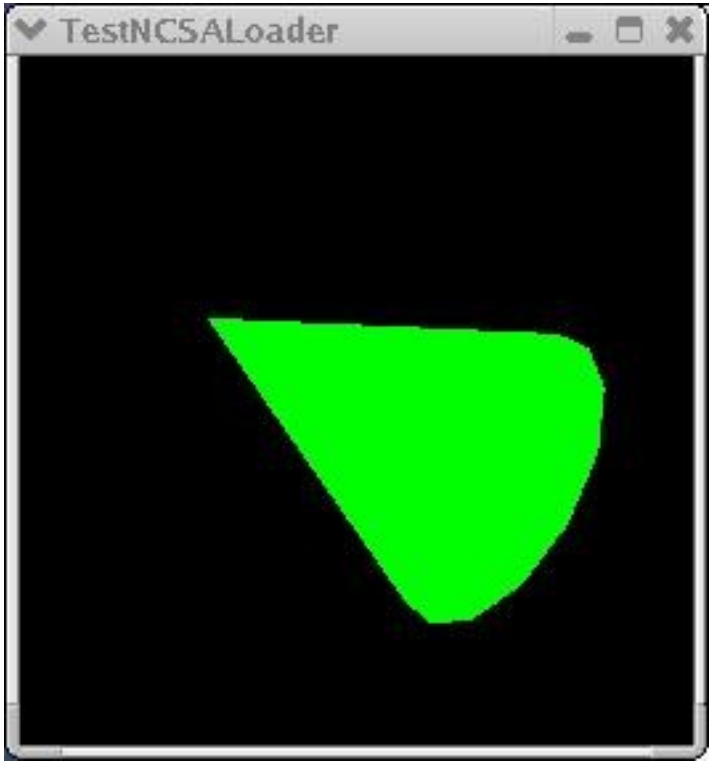
```
public void enableInteraction (BranchGroup objRoot, Canvas3D c) {
   BoundingSphere bounds = new BoundingSphere (new Point3d (0, 0, 0), 100) ;
   PickRotateBehavior prb = new PickRotateBehavior (objRoot, c, bounds) ;
   prb.setMode (PickTool.GEOMETRY) ;
   prb.setTolerance (0.0f) ;
   objRoot.addChild (prb) ;
   PickTranslateBehavior ptb = new PickTranslateBehavior (objRoot, c, bounds) ;
   ptb.setMode (PickTool.GEOMETRY) ;
   ptb.setTolerance (0.0f) ;
   objRoot.addChild (ptb) ;
   PickZoomBehavior pzb = new PickZoomBehavior (objRoot, c, bounds) ;
   pzb.setMode (PickTool.GEOMETRY) ;
   pzb.setTolerance (0.0f) ;
   objRoot.addChild (pzb) ;
}
```

Chargeurs et interaction (3)

```
// Ce qu'il faudrait changer pour charger un nœud contenant une primitive de base :

// cas d'une primitive de base
Scene vrml_scene = ldr.load ("coneVert.wrl") ;
objTrans.addChild (vrml_scene.getSceneGroup ()) ;
Primitive prim = (Primitive)(vrml_scene.getSceneGroup ().getChild (0)) ;
prim.getShape (0).getGeometry ().setCapability (Geometry.ALLOW_INTERSECT) ;
```

Chargeurs et interaction (4)





Classes mathématiques pour l’Infographie

- Définies dans le package java.vecmath
- Stockage des valeurs caractéristiques des objets
- Implantation des fonctions membres pour les opérations mathématiques classiques sur ces objets utiles pour l'infographie

Les classes vecteur (1)

- « Tuple2f « :
  - coordonnées flottantes simple précision en 2D
  - sous-classes : « Point2f » « TexCoord2f » « Vector2f »
- « Tuple3b » :
  - coordonnées entières sur un octet signé en 3D
  - sous-classe : « Color3b »
- « Tuple3d » :
  - coordonnées réelles double précision en 3D
  - sous-classes : « Point3d » « Vector3d »

Les classes vecteur (2)

- « Tuple3f » :
  - coordonnées réelles simple précision en 3D
  - sous-classes : « Color3f » « Point3f » « TexCoord3f » « Vector3f »
- « Tuple4b » :
  - coordonnées entières sur un octet signé en 4D
  - sous-classe : « Color4b »
- « Tuple4d » :
  - coordonnées flottantes double précision en 4D
  - sous-classes : « Point4d » « Quat4d » « Vector4d »

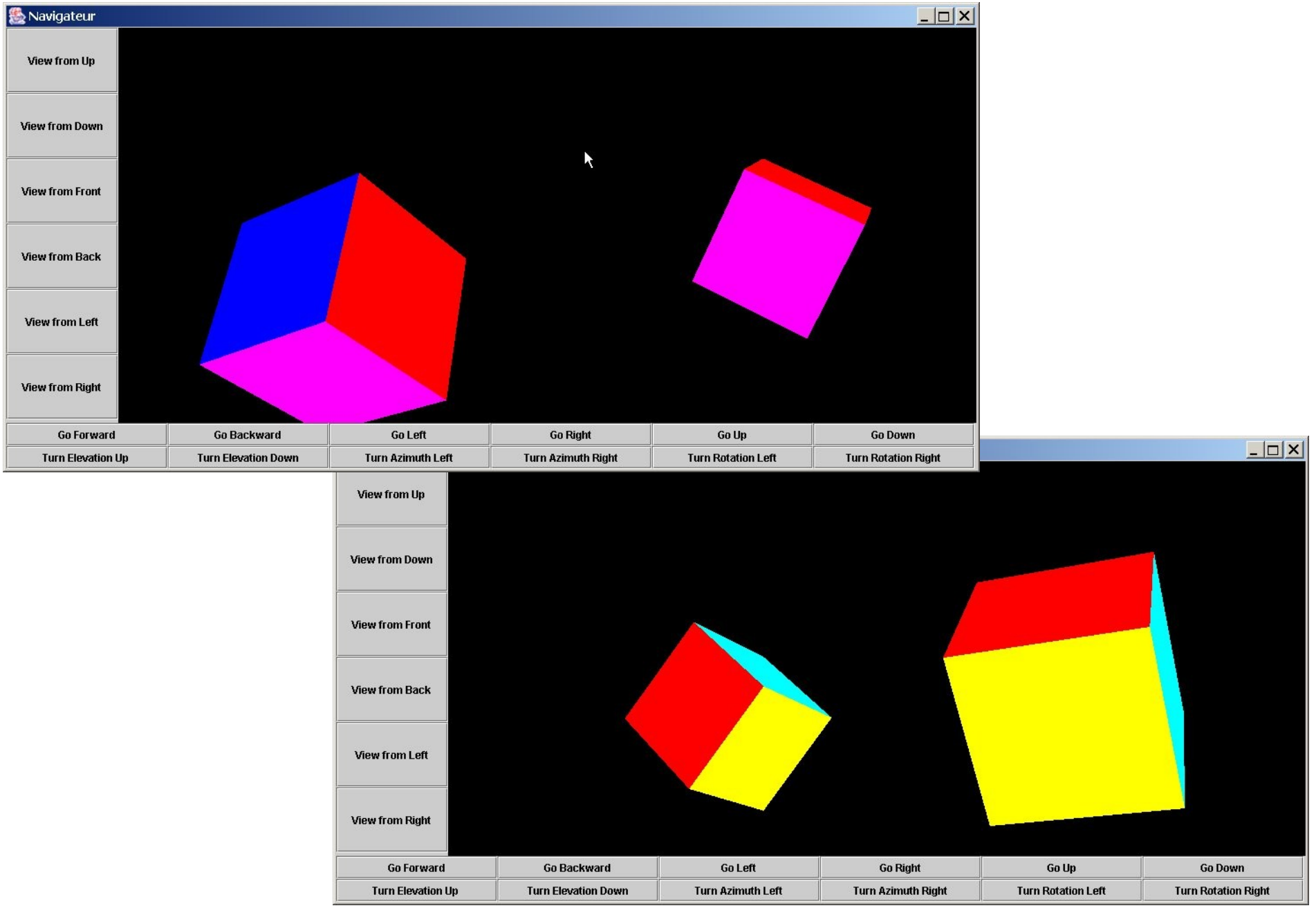
Les classes vecteur (3)

- « Tuple4f » :
  - ✔ coordonnées flottantes simple précision en4D
  - ✔ sous-classes : « Color4f » « Point4f » « Quat4f » « Vector4f »
- « AxisAngle4d » :
  - ✔ coordonnées flottantes double précision
    - ✗ valeurs d'une rotation a en radian autour d'un axe (x, y, z) en 3D
- « AxisAngle4f » :
  - ✔ coordonnées flottantes simple précision
    - ✗ valeurs d'une rotation a en radian autour d'un axe (x, y, z) en 3D
- « GVector » :
  - ✔ vecteur de dimension arbitraire pour stocker des valeurs double précision

Les classes matrice

- « Matrix3f » :
  - matrice 3x3 de réels simple précision
- « Matrix3d » :
  - matrice 3x3 de réels double précision
- « Matrix4f » :
  - matrice 4x4 de réels simple précision
- « Matrix4d » :
  - matrice 4x4 de réels double précision
- « GMatrix » :
  - matrice arbitraire MxN (M lignes, N colonnes) de réels double précision

Exemple de navigation



Point de vue : placement absolu

```
SimpleUniverse u ;

…

   void goThere (double x, double y, double z) {
      TransformGroup vpTrans ;
      vpTrans = u.getViewingPlatform ().getViewPlatformTransform () ;
      Vector3d translate = new Vector3d () ;
      translate.set (x, y, z) ;
      Transform3D t3d = new Transform3D () ;
      vpTrans.getTransform (t3d) ; // récupération de position et orientation
      t3d.setTranslation (translate) ; // on modifie seulement la position
      vpTrans.setTransform (t3d) ; // on met à jour …
   }
```

Point de vue : orientation absolue

```
void lookThatWay (double h, double p, double r) {
    TransformGroup vpTrans ;
    vpTrans = u.getViewingPlatform ().getViewPlatformTransform () ;
    Vector3d rotate = new Vector3d () ;
    rotate.set (h, p, r) ;
    Transform3D t3d = new Transform3D () ;
    vpTrans.getTransform (t3d) ;
    Vector3d translate = new Vector3d () ;
    // on récupère la translation pour la réintroduire ensuite
    t3d.get (translate) ;
    // on fixe l'orientation, mais cela écrase la translation
    t3d.setEuler (rotate) ;
    // on réintroduit la translation, cela ne perturbe pas l'orientation
    t3d.setTranslation (translate) ;
    vpTrans.setTransform (t3d) ;
}
```

Point de vue : déplacement relatif

```
void goLocallyInThisDirection (double dx, double dy, double dz) {
    TransformGroup vpTrans ;
    vpTrans = u.getViewingPlatform ().getViewPlatformTransform () ;
    Transform3D oldT3D = new Transform3D () ;
    vpTrans.getTransform (oldT3D) ;
    Vector3d translate = new Vector3d () ;
    translate.set (dx, dy, dz) ;
    Transform3D localT3D = new Transform3D () ;
    localT3D.setTranslation (translate) ;
    Transform3D newT3D = new Transform3D () ;
    newT3D.mul (oldT3D, localT3D) ;
    vpTrans.setTransform (newT3D) ;
}
```

Point de vue : orientation relative

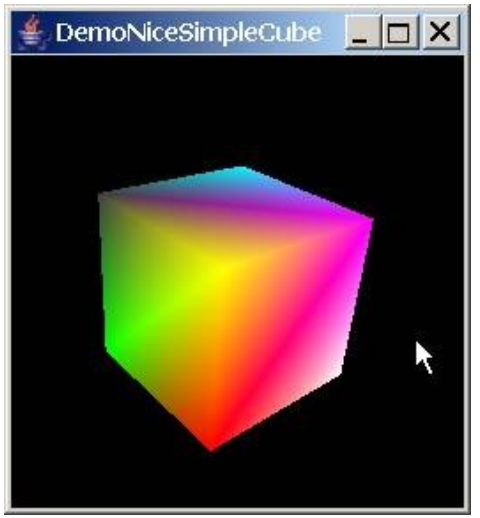
```
void lookLocallyInThisWay (double h, double p, double r) {
    TransformGroup vpTrans ;
    vpTrans = u.getViewingPlatform ().getViewPlatformTransform () ;
    Transform3D oldT3D = new Transform3D () ;
    vpTrans.getTransform (oldT3D) ;
    Vector3d rotate = new Vector3d () ;
    rotate.set (h, p, r) ;
    Transform3D localT3D = new Transform3D () ;
    localT3D.setEuler (rotate) ;
    Transform3D newT3D = new Transform3D () ;
    newT3D.mul (oldT3D, localT3D) ;
    vpTrans.setTransform (newT3D) ;
}
```

Création d’une forme 3D

- Utilisation d’un nœud « Shape3D »
  - par exemple par dérivation …
- Lui associer une géométrie
  - par construction ou via la méthode « setGeometry »
- Éventuellement lui associer une apparence
  - par construction ou via la méthode « setAppearance »

Création de la partie géométrie

- Utilisation d'un nœud dérivé de « Geometry »
- Exemple : les « QuadArray », ...



NiceSimpleCube.java (1/4)

```
import javax.media.j3d.* ;
import javax.vecmath.* ;
import java.awt.Color ;

class NiceSimpleCubeGeometry extends QuadArray {

    public static final Point3d pfhr = new Point3d (1.0, 1.0, 1.0) ;
    public static final Point3d pfhl = new Point3d (-1.0, 1.0, 1.0) ;
    public static final Point3d pflr = new Point3d (1.0, -1.0, 1.0) ;
    public static final Point3d pfll = new Point3d (-1.0, -1.0, 1.0) ;
    public static final Point3d pbhr = new Point3d (1.0, 1.0, -1.0) ;
    public static final Point3d pbhl = new Point3d (-1.0, 1.0, -1.0) ;
    public static final Point3d pblr = new Point3d (1.0, -1.0, -1.0) ;
    public static final Point3d pbll = new Point3d (-1.0, -1.0, -1.0) ;
```

NiceSimpleCube.java (2/4)

```
final static Point3d [] points = {
    pfhr, pfhl, pflr, pfll, pbhr, pbhl, pblr, pbll
} ;

Point3d [] faces = {
    points [0], points [1], points [3], points [2], // front face
    points [5], points [4], points [6], points [7], // back face
    points [0], points [2], points [6], points [4], // right face
    points [1], points [5], points [7], points [3], // left face
    points [1], points [0], points [4], points [5], // top face
    points [3], points [7], points [6], points [2] // bottom face
} ;
```

NiceSimpleCube.java (3/4)

```
public static final Color3b red = new Color3b (Color.red) ;
public static final Color3b green = new Color3b (Color.green) ;
public static final Color3b blue = new Color3b (Color.blue) ;
public static final Color3b yellow = new Color3b (Color.yellow) ;
public static final Color3b magenta = new Color3b (Color.magenta) ;
public static final Color3b cyan = new Color3b (Color.cyan) ;
public static final Color3b darkGray = new Color3b (Color.darkGray) ;
public static final Color3b white = new Color3b (Color.white) ;

protected static final Color3b [] colors = {
    red, green, blue, white,
    darkGray, yellow, magenta, cyan,
    red, white, magenta, yellow,
    green, darkGray, cyan, blue,
    green, red, yellow, darkGray,
    blue, cyan, magenta, white
} ;
```

NiceSimpleCube.java (4/4)

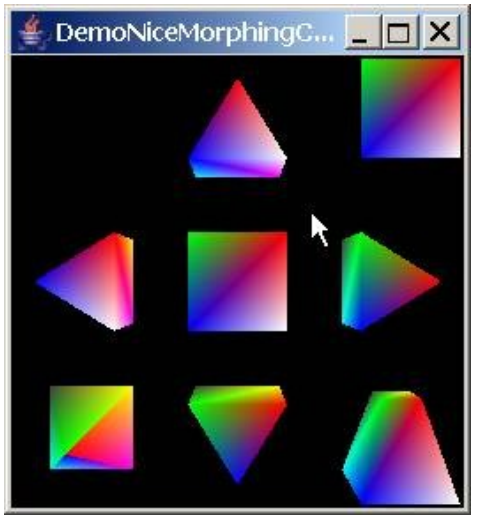
```
    public NiceSimpleCubeGeometry () {
        super (24, QuadArray.COORDINATES | QuadArray.COLOR_3) ;
        setCoordinates (0, faces) ;
        setColors (0, colors) ;
    }

}

public class NiceSimpleCube extends Shape3D {

    public NiceSimpleCube () {
        super (new NiceSimpleCubeGeometry ()) ;
    }

}
```

Utilisation du NiceSimpleCube

```
…
public TransformGroup createNiceSimpleCube (Vector3d v3d) {
    Transform3D translation = new Transform3D () ;
    translation.setTranslation (v3d) ;
    translation.setScale (0.2d) ;
    TransformGroup objTrans = new TransformGroup (translation) ;
    objTrans.setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;
    objTrans.setCapability (TransformGroup.ALLOW_TRANSFORM_READ) ;
    objTrans.setCapability (TransformGroup.ENABLE_PICK_REPORTING) ;
    NiceSimpleCube nc = new NiceSimpleCube () ;
    nc.getGeometry ().setCapability (Geometry.ALLOW_INTERSECT) ;
    objTrans.addChild (nc) ;
    return (objTrans) ;
}
…
```

Déformation d'objet : le nœud « Morph »

- Une seule apparence
- Un tableau de géométries :
  - ✔ toutes du même type !
  - ✔ avec le même format de points
  - ✔ avec le même nombre de points
  - ✔ …
- Un tableau de poids associés :
  - ✔ pour les nœuds de chaque géométrie
  - ✔ la somme des poids d'un nœud doit être égale à 1
  - ✔ calcul des valeurs par interpolations linéaires
- Actuellement « deprecated » sans remplaçant ?!?



Évolution de la déformation

- Avec un nœud « Alpha » :
  - ✔ pour le calcul des poids en fonction du temps
- Avec un « Behavior » adapté :
  - ✔ déclenché par un « WakeupOnElapsedFrames (0) » :
    - ✗ toutes les modifications effectuées sur les objets du graphe de scène à ce moment seront effectives pour l'image à dessiner
  - ✔ à chaque nouvelle image dessinée (frame), on recalcule la nouvelle déformation

Exemple de déformation

- Création de plusieurs géométries :
  - dérivées d'une même classe ancêtre
- Création d'une classe pour la déformation :
  - paramétrée par le tableau de géométries utilisé pour les déformations
  - devra autoriser les accès sur toutes les géométries pour d'éventuelles interactions
  - sera associée à un « Behavior » et à un « Alpha »
  - le « Behavior » calculera les poids en fonction de l'« Alpha »

GeometryForMorphing.java (1/4)

```
import javax.media.j3d.* ;
import javax.vecmath.* ;
import java.awt.Color ;

public abstract class GeometryForMorphing extends QuadArray {

    public static final Color3b red = new Color3b (Color.red) ;
    public static final Color3b green = new Color3b (Color.green) ;
    public static final Color3b blue = new Color3b (Color.blue) ;
    public static final Color3b yellow = new Color3b (Color.yellow) ;
    public static final Color3b magenta = new Color3b (Color.magenta) ;
    public static final Color3b cyan = new Color3b (Color.cyan) ;
    public static final Color3b darkGray = new Color3b (Color.darkGray) ;
    public static final Color3b white = new Color3b (Color.white) ;
```

GeometryForMorphing.java (2/4)

```
public static final Point3d pfhr = new Point3d (1.0, 1.0, 1.0) ;
public static final Point3d pfhl = new Point3d (-1.0, 1.0, 1.0) ;
public static final Point3d pflr = new Point3d (1.0, -1.0, 1.0) ;
public static final Point3d pfll = new Point3d (-1.0, -1.0, 1.0) ;
public static final Point3d pbhr = new Point3d (1.0, 1.0, -1.0) ;
public static final Point3d pbhl = new Point3d (-1.0, 1.0, -1.0) ;
public static final Point3d pblr = new Point3d (1.0, -1.0, -1.0) ;
public static final Point3d pbll = new Point3d (-1.0, -1.0, -1.0) ;
public static final Point3d cfront = new Point3d (0.0, 0.0, 1.0) ;
public static final Point3d cback = new Point3d (0.0, 0.0, -1.0) ;
public static final Point3d cright = new Point3d (1.0, 0.0, 0.0) ;
public static final Point3d cleft = new Point3d (-1.0, 0.0, 0.0) ;
public static final Point3d ctop = new Point3d (0.0, 1.0, 0.0) ;
public static final Point3d cbottom = new Point3d (0.0, -1.0, 0.0) ;

protected abstract Point3d [] getPoints () ;
```

GeometryForMorphing.java (3/4)

```
protected static final Color3b [] colors = {
    red, green, blue, white,
    darkGray, yellow, magenta, cyan,
    red, white, magenta, yellow,
    green, darkGray, cyan, blue,
    green, red, yellow, darkGray,
    blue, cyan, magenta, white
} ;
```

GeometryForMorphing.java (4/4)

```
    public GeometryForMorphing () {
        super (24, QuadArray.COORDINATES | QuadArray.COLOR_3) ;
        Point3d [] faces = {
            getPoints () [0], getPoints () [1], getPoints () [3], getPoints () [2], // front face
            getPoints () [5], getPoints () [4], getPoints () [6], getPoints () [7], // back face
            getPoints () [0], getPoints () [2], getPoints () [6], getPoints () [4], // right face
            getPoints () [1], getPoints () [5], getPoints () [7], getPoints () [3], // left face
            getPoints () [1], getPoints () [0], getPoints () [4], getPoints () [5], // top face
            getPoints () [3], getPoints () [7], getPoints () [6], getPoints () [2] // bottom face
        } ;
        setCoordinates (0, faces) ;
        setColors (0, colors) ;
    }


}
```

NiceCube.java

```
import javax.media.j3d.* ;
import javax.vecmath.* ;

class NiceCubeGeometry extends GeometryForMorphing {
    final static Point3d [] points = { pfhr, pfhl, pflr, pfll, pbhr, pbhl, pblr, pbll } ;
    protected Point3d [] getPoints () { return points ; }
}

public class NiceCube extends Shape3D {
    public NiceCube () {
        super (new NiceCubeGeometry ()) ;
    }
}
```

PyramidFront.java

```
import javax.media.j3d.* ;
import javax.vecmath.* ;

class PyramidFrontGeometry extends GeometryForMorphing {
    final static Point3d [] points = {cfront, cfront, cfront, cfront, pbhr, pbhl, pblr, pbll } ;
    protected Point3d [] getPoints () { return points ; }
}

public class PyramidFront extends Shape3D {
    public PyramidFront () {
        super (new PyramidFrontGeometry ()) ;
    }
}
```

MorphingCube.java (1/5)

```
import javax.media.j3d.* ;
import javax.vecmath.* ;
import java.util.Enumeration;

public class MorphingCube extends TransformGroup {

    Alpha morphingAlpha ;
    Morph morph ;

    public MorphingCube (Vector3d v3d, GeometryForMorphing [] ga) {
        Transform3D translation = new Transform3D () ;
        translation.setTranslation (v3d) ;
        translation.setScale (0.2d) ;
        setTransform (translation) ;
        setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;
        setCapability (TransformGroup.ALLOW_TRANSFORM_READ) ;
        setCapability (TransformGroup.ENABLE_PICK_REPORTING) ;
```

MorphingCube.java (2/5)

```
    for (int i = 0 ; i < ga.length ; i ++) {
        ga [i].setCapability (Geometry.ALLOW_INTERSECT) ;
    }
    morph = new Morph (ga) ;
    morph.setCapability (Morph.ALLOW_WEIGHTS_READ) ;
    morph.setCapability (Morph.ALLOW_WEIGHTS_WRITE) ;
    addChild (morph) ;
    morphingAlpha = new Alpha (-1, ga.length * 2000) ;
    MorphingBehavior mb = new MorphingBehavior () ;
    BoundingSphere bounds = new BoundingSphere (new Point3d (0.0,0.0,0.0),
                                                100.0) ;
    mb.setSchedulingBounds (bounds) ;
    addChild (mb) ;
}
```

MorphingCube.java (3/5)

```
class MorphingBehavior extends Behavior {

    double weights [] ;
   int plageCourante ; // de 0 à weights.length - 1
    double tempsPlage ;

    WakeupOnElapsedFrames w = new WakeupOnElapsedFrames (0) ;

    public void initialize () {
        morphingAlpha.setStartTime (System.currentTimeMillis ()) ;
        wakeupOn (w) ;
    }

    public MorphingBehavior () {
        weights = morph.getWeights () ;
        tempsPlage = 1.0 / weights.length ;
    }
```

MorphingCube.java (4/5)

```
public void processStimulus (Enumeration criteria) {
    double val = morphingAlpha.value () ;
    // pour la première géométrie : v2 < v1 ...
    double v1 = (weights.length - 1) * tempsPlage ;
    double v2 = tempsPlage ;
    if ((val >= v2) && (val <= v1)) {
        weights [0] = 0 ;
    } else {
        if (val <= tempsPlage) {
            weights [0] = (tempsPlage - val) * weights.length ;
        } else {
            weights [0] = (val - (weights.length - 1) * tempsPlage) * weights.length ;
        }
    }
```

MorphingCube.java (5/5)

```
        // pour toutes les autres, v1 < v2, c'est plus simple
        for (int i = 1 ; i < weights.length ; i ++) {
            v1 = (i - 1) * tempsPlage ;
            v2 = (i + 1) * tempsPlage ;
            if ((val < v1) || (val > v2)) {
                weights [i] = 0 ;
            } else {
                if (val < i * tempsPlage) {
                    weights [i] = (val - (i - 1) * tempsPlage) * weights.length ;
                } else {
                    weights [i] = ((i + 1) * tempsPlage - val) * weights.length ;
                }
            }
        }
        morph.setWeights (weights) ;
        wakeupOn (w) ;
} } }
```

Utilisation du MorphingCube (1/2)

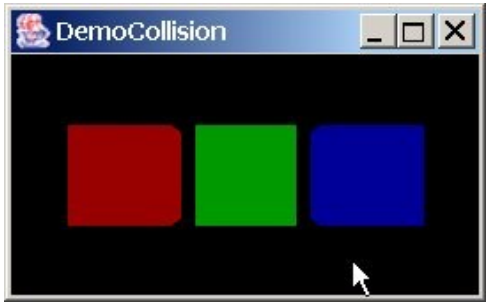
```
…
public BranchGroup createSceneGraph () {
    // Create the root of the branch graph
    BranchGroup objRoot = new BranchGroup () ;
    objRoot.addChild (placeShape (new Vector3d (0, 0, 0), new NiceCube ())) ;
    objRoot.addChild (placeShape (new Vector3d (-0.7, -0.7, 0),
                        new PyramidFront ())) ;
    objRoot.addChild (placeShape (new Vector3d (0.7, 0.7, 0),
                        new PyramidBack ())) ;
    objRoot.addChild (placeShape (new Vector3d (0.7, 0, 0),
                        new PyramidRight ())) ;
    objRoot.addChild (placeShape (new Vector3d (-0.7, 0, 0),
                        new PyramidLeft ())) ;
    objRoot.addChild (placeShape (new Vector3d (0, 0.7, 0), new PyramidTop ())) ;
    objRoot.addChild (placeShape (new Vector3d (0, -0.7, 0),
                        new PyramidBottom ())) ;
    GeometryForMorphing [] ga = new GeometryForMorphing [12]
```

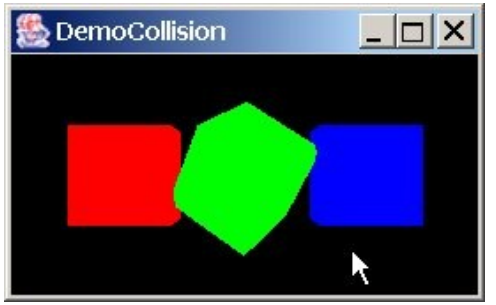
Utilisation du MorphingCube (2/2)

```
    ga [0] = (GeometryForMorphing)new NiceCube ().getGeometry () ;
    ga [1] = (GeometryForMorphing)new PyramidFront ().getGeometry () ;
    ga [2] = (GeometryForMorphing)new NiceCube ().getGeometry () ;
    ga [3] = (GeometryForMorphing)new PyramidBack ().getGeometry () ;
    ga [4] = (GeometryForMorphing)new NiceCube ().getGeometry () ;
    ga [5] = (GeometryForMorphing)new PyramidRight ().getGeometry () ;
    ga [6] = (GeometryForMorphing)new NiceCube ().getGeometry () ;
    ga [7] = (GeometryForMorphing)new PyramidLeft ().getGeometry () ;
    ga [8] = (GeometryForMorphing)new NiceCube ().getGeometry () ;
    ga [9] = (GeometryForMorphing)new PyramidTop ().getGeometry () ;
    ga [10] = (GeometryForMorphing)new NiceCube ().getGeometry () ;
    ga [11] = (GeometryForMorphing)new PyramidBottom ().getGeometry () ;
    objRoot.addChild (new MorphingCube (new Vector3d (0.7, -0.7, 0), ga)) ;
    return objRoot ;
}
…
```

Détection de collision

- À l'aide d'un « Behavior », qui s'active sur :
  - ✔ « WakeupOnCollisionEntry »
  - ✔ « WakeupOnCollisionExit »
- Exemple (fortement inspiré des exemples Sun) :
  - ✔ changer la couleur d'un objet lors d'une collision





DemoCollision.java (1/7)

```
import javax.swing.JApplet ;
import java.awt.* ;
import com.sun.j3d.utils.applet.* ;
import com.sun.j3d.utils.geometry.* ;
import com.sun.j3d.utils.universe.* ;
import javax.media.j3d.* ;
import javax.vecmath.* ;
import com.sun.j3d.utils.picking.* ;
import com.sun.j3d.utils.picking.behaviors.* ;

public class DemoCollision extends JApplet {

    private SimpleUniverse u = null ;
    private Canvas3D c = null ;
```

DemoCollision.java (2/7)

```
public TransformGroup createCube (Vector3d v3d, Color3f color,
                                  Color3f collisionColor) {
    Transform3D translation = new Transform3D () ;
    translation.setTranslation (v3d) ;
    translation.setScale (0.2d) ;
    TransformGroup objTrans = new TransformGroup (translation) ;
    objTrans.setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;
    objTrans.setCapability (TransformGroup.ALLOW_TRANSFORM_READ) ;
    objTrans.setCapability (TransformGroup.ENABLE_PICK_REPORTING) ;
    Cube box = new Cube () ;
    box.getGeometry ().setCapability (Geometry.ALLOW_INTERSECT) ;
    Appearance app =  new Appearance () ;
    box.setAppearance (app) ;
    objTrans.addChild (box) ;
```

DemoCollision.java (3/7)

```
    ColoringAttributes ca = new ColoringAttributes () ;
    ca.setColor (color) ;
    app.setCapability (Appearance.ALLOW_COLORING_ATTRIBUTES_WRITE) ;
    app.setColoringAttributes (ca) ;
    CollisionDetector cd = new CollisionDetector (box, collisionColor) ;
    BoundingSphere bounds = new BoundingSphere (new Point3d (0.0, 0.0, 0.0),
                                                                    100.0) ;
    cd.setSchedulingBounds (bounds) ;
    objTrans.addChild (cd) ;
    return (objTrans) ;
}
```

DemoCollision.java (4/7)

```
public BranchGroup createSceneGraph () {
    BranchGroup objRoot = new BranchGroup () ;
    objRoot.addChild (createCube (new Vector3d (-0.5, 0.0, 0.0),
                                  new Color3f (0.6f, 0.0f, 0.0f), new Color3f (1.0f, 0.0f, 0.0f))) ;
    objRoot.addChild (createCube (new Vector3d (0.0, 0.0, 0.0),
                                  new Color3f (0.0f, 0.6f, 0.0f), new Color3f (0.0f, 1.0f, 0.0f))) ;
    objRoot.addChild (createCube (new Vector3d (0.5, 0.0, 0.0),
                                  new Color3f (0.0f, 0.0f, 0.6f), new Color3f (0.0f, 0.0f, 1.0f))) ;
    return objRoot ;
}
```

DemoCollision.java (5/7)

```
public void enableInteraction (BranchGroup objRoot) {
    BoundingSphere bounds =  new BoundingSphere (new Point3d (0, 0, 0), 100) ;
    PickRotateBehavior prb = new PickRotateBehavior (objRoot, c, bounds) ;
    prb.setMode (PickTool.GEOMETRY) ;
    prb.setTolerance (0.0f) ;
    objRoot.addChild (prb) ;
    PickTranslateBehavior ptb = new PickTranslateBehavior (objRoot, c, bounds) ;
    ptb.setMode (PickTool.GEOMETRY) ;
    ptb.setTolerance (0.0f) ;
    objRoot.addChild (ptb) ;
    PickZoomBehavior pzb = new PickZoomBehavior (objRoot, c, bounds) ;
    pzb.setMode (PickTool.GEOMETRY) ;
    pzb.setTolerance (0.0f) ;
    objRoot.addChild (pzb) ;
}
```

DemoCollision.java (6/7)

```
public DemoCollision () {
}

public void init () {
    GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration () ;
    c = new Canvas3D (config) ;
    getContentPane ().add (c) ;
    u = new SimpleUniverse (c) ;
    BranchGroup scene = createSceneGraph () ;
    enableInteraction (scene) ;
    u.getViewingPlatform ().setNominalViewingTransform () ;
    scene.compile () ;
    u.addBranchGraph (scene) ;
}
```

DemoCollision.java (7/7)

```
    public void destroy () {
        u.removeAllLocales () ;
    }
    public static void main (String [] args) {
        new MainFrame (new DemoCollision (), 256, 256) ;
    }
}
```

CollisionDetector.java (1/3)

```
import javax.media.j3d.* ;
import javax.vecmath.* ;
import com.sun.j3d.utils.picking.behaviors.* ;
import java.util.Enumeration ;

public class CollisionDetector extends Behavior {

    private ColoringAttributes highlight ;
    private boolean inCollision = false ;
    private Shape3D shape ;
    private ColoringAttributes shapeColoring ;
    private Appearance shapeAppearance ;

    private WakeupOnCollisionEntry wEnter ;
    private WakeupOnCollisionExit wExit ;
```

CollisionDetector.java (2/3)

```
public CollisionDetector (Shape3D s, Color3f c) {
    shape = s ;
    highlight = new ColoringAttributes (c, ColoringAttributes.SHADE_GOURAUD) ;
    shapeAppearance = shape.getAppearance () ;
    shapeColoring = shapeAppearance.getColoringAttributes () ;
    inCollision = false ;
}

public void initialize () {
    wEnter = new WakeupOnCollisionEntry (shape,
                                         WakeupOnCollisionEntry.USE_GEOMETRY) ;
    wExit = new WakeupOnCollisionExit (shape,
                                         WakeupOnCollisionEntry.USE_GEOMETRY) ;
    wakeupOn (wEnter) ;
}
```

CollisionDetector.java (3/3)

```
    public void processStimulus (Enumeration criteria) {
        inCollision = ! inCollision ;
        if (inCollision) {
            shapeAppearance.setColoringAttributes (highlight) ;
            wakeupOn (wExit) ;
        } else {
            shapeAppearance.setColoringAttributes (shapeColoring) ;
            wakeupOn (wEnter) ;
        }
    }

}
```

Cube.java (1/3)

```
class CubeGeometry extends QuadArray {

    public static final Point3d pfhr = new Point3d (1.0, 1.0, 1.0) ;
    public static final Point3d pfhl = new Point3d (-1.0, 1.0, 1.0) ;
    public static final Point3d pflr = new Point3d (1.0, -1.0, 1.0) ;
    public static final Point3d pfll = new Point3d (-1.0, -1.0, 1.0) ;
    public static final Point3d pbhr = new Point3d (1.0, 1.0, -1.0) ;
    public static final Point3d pbhl = new Point3d (-1.0, 1.0, -1.0) ;
    public static final Point3d pblr = new Point3d (1.0, -1.0, -1.0) ;
    public static final Point3d pbll = new Point3d (-1.0, -1.0, -1.0) ;

    final static Point3d [] points = {
        pfhr, pfhl, pflr, pfll, pbhr, pbhl, pblr, pbll
    } ;
```

Cube.java (2/3)

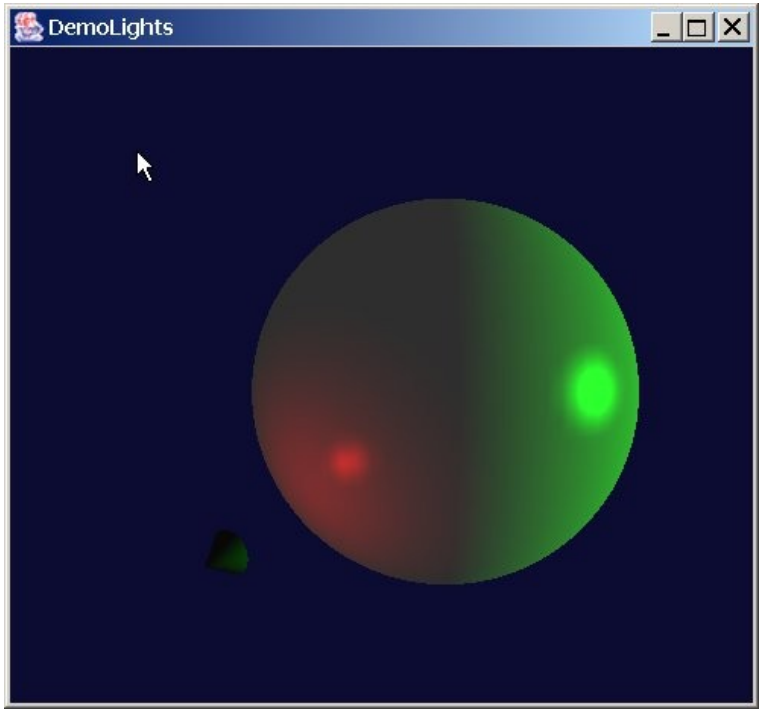
```
    Point3d [] faces = {
        points [0], points [1], points [3], points [2],
        points [5], points [4], points [6], points [7],
        points [0], points [2], points [6], points [4],
        points [1], points [5], points [7], points [3],
        points [1], points [0], points [4], points [5],
        points [3], points [7], points [6], points [2]
    } ;

    public CubeGeometry () {
        super (24, QuadArray.COORDINATES) ;
        setCoordinates (0, faces) ;
    }

}
```

Cube.java (3/3)

```
class Cube extends Shape3D {
    public Cube () {
        super (new CubeGeometry ()) ;
        setAppearance (new Appearance ()) ;
    }
}
```

Lumières

- « Light » (couleur, zone d'influence)
  - ✔ « AmbientLight »
  - ✔ « DirectionalLight » (direction)
  - ✔ « PointLight » (position, atténuation)
    - ✗ « SpotLight » (direction, concentration)



DemoLights.java (1/10)

```
import javax.swing.JApplet ;
import java.awt.* ;
import com.sun.j3d.utils.applet.* ;
import com.sun.j3d.utils.geometry.* ;
import com.sun.j3d.utils.universe.* ;
import javax.media.j3d.* ;
import javax.vecmath.* ;
import com.sun.j3d.utils.picking.* ;
import com.sun.j3d.utils.picking.behaviors.* ;

public class DemoLights extends JApplet {

    private SimpleUniverse u = null ;
    private Canvas3D c = null ;
    private BoundingSphere bounds =
                                new BoundingSphere (new Point3d (0.0, 0.0, 0.0), 100.0) ;
```

DemoLights.java (2/10)

```
public TransformGroup createSphere (Vector3d v3d) {
    Transform3D translation = new Transform3D () ;
    translation.setTranslation (v3d) ;
    TransformGroup objTrans = new TransformGroup (translation) ;
    objTrans.setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;
    objTrans.setCapability (TransformGroup.ALLOW_TRANSFORM_READ) ;
    objTrans.setCapability (TransformGroup.ENABLE_PICK_REPORTING) ;
```

DemoLights.java (3/10)

```
    Color3f objectColor = new Color3f (0.6f, 0.6f, 0.6f) ;
    Color3f emissiveColor = new Color3f (0.0f, 0.0f, 0.0f) ;
    Color3f specularColor = new Color3f (1.0f, 1.0f, 1.0f) ;
    Material m = new Material (objectColor, emissiveColor, objectColor,
                                        specularColor, 100.0f);
    Appearance a = new Appearance( ) ;
    m.setLightingEnable (true) ;
    a.setMaterial (m) ;
    Sphere sphere = new Sphere (0.5f, Sphere.GENERATE_NORMALS, 80, a) ;
    sphere.getShape ().getGeometry ().setCapability
                                                (Geometry.ALLOW_INTERSECT) ;
    objTrans.addChild (sphere) ;
    return (objTrans) ;
}
```

DemoLights.java (4/10)

```
public TransformGroup createSpotLight (Vector3d v3d) {
    Transform3D translation = new Transform3D () ;
    translation.setTranslation (v3d) ;
    translation.setScale (0.5d) ;
    TransformGroup objTrans = new TransformGroup (translation) ;
    objTrans.setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE) ;
    objTrans.setCapability (TransformGroup.ALLOW_TRANSFORM_READ) ;
    objTrans.setCapability (TransformGroup.ENABLE_PICK_REPORTING) ;
    Color3f spotLightColor = new Color3f (1.0f, 0.0f, 0.0f) ;
    Cone cone = new Cone (0.1f, 0.15f) ;
    cone.getShape (0).getGeometry ().setCapability
                                                    (Geometry.ALLOW_INTERSECT) ;
    cone.getShape (1).getGeometry ().setCapability
                                                    (Geometry.ALLOW_INTERSECT) ;
    objTrans.addChild (cone) ;
```

DemoLights.java (5/10)

```
    Point3f position = new Point3f (0.0f, 0.0f, 0.0f) ;
    Point3f attenuation = new Point3f (1.0f, 1.0f, 1.0f) ;
    Vector3f direction = new Vector3f (0.0f, -1.0f, 0.0f) ;
    float spreadAngle = (float)Math.PI/2 ;
    float concentration = 0.0f ;
    SpotLight spot = new SpotLight (true, spotLightColor, position, attenuation,
                                                direction, spreadAngle, concentration) ;
    spot.setInfluencingBounds (bounds) ;
    objTrans.addChild (spot) ;
    return (objTrans) ;
}
```

DemoLights.java (6/10)

```
public BranchGroup createSceneGraph () {
    BranchGroup objRoot = new BranchGroup () ;
    Color3f directionalColor = new Color3f (0.0f, 1.0f, 0.0f) ;
    Color3f ambientColor = new Color3f (0.3f, 0.3f, 0.3f) ;
    Color3f backgroundColor = new Color3f (0.05f, 0.05f, 0.2f) ;
    DirectionalLight dl = new DirectionalLight (true, directionalColor,
                                                new Vector3f (-1.0f, 0.0f, 0.0f)) ;
    dl.setInfluencingBounds (bounds) ;
    objRoot.addChild (dl) ;
```

DemoLights.java (7/10)

```
    AmbientLight ambientLight = new AmbientLight (true, ambientColor) ;
    ambientLight.setInfluencingBounds (bounds) ;
    objRoot.addChild (ambientLight) ;
    Background bg = new Background (backgroundColor) ;
    bg.setApplicationBounds (bounds) ;
    bjRoot.addChild (bg) ;
    objRoot.addChild (createSphere (new Vector3d (-0.5, 0.0, 0.0))) ;
    objRoot.addChild (createSpotLight (new Vector3d (0.0, 0.0, 0.0))) ;
    return objRoot ;
}
```

DemoLights.java (8/10)

```
public void enableInteraction (BranchGroup objRoot) {
    PickRotateBehavior prb = new PickRotateBehavior (objRoot, c, bounds) ;
    prb.setMode (PickTool.GEOMETRY) ;
    prb.setTolerance (0.0f) ;
    objRoot.addChild (prb) ;
    pickTranslateBehavior ptb = new PickTranslateBehavior (objRoot, c, bounds) ;
    ptb.setMode (PickTool.GEOMETRY) ;
    ptb.setTolerance (0.0f) ;
    objRoot.addChild (ptb) ;
    PickZoomBehavior pzb = new PickZoomBehavior (objRoot, c, bounds) ;
    pzb.setMode (PickTool.GEOMETRY) ;
    pzb.setTolerance (0.0f) ;
    objRoot.addChild (pzb) ;
}
```

DemoLights.java (9/10)

```
public DemoLights () {
}

public void init () {
    GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration () ;
    c = new Canvas3D (config) ;
    getContentPane ().add (c) ;
    u = new SimpleUniverse (c) ;
    BranchGroup scene = createSceneGraph () ;
    enableInteraction (scene) ;
    u.getViewingPlatform ().setNominalViewingTransform () ;
    scene.compile () ;
    u.addBranchGraph (scene) ;
}
```

DemoLights.java (10/10)

```
    public void destroy () {
        u.removeAllLocales () ;
    }

    public static void main (String [] args) {
        new MainFrame (new DemoLights (), 256, 256) ;
    }

}
```

Bibliographie :

- Serge Morvan :
  - « Bibliothèques 3D Java3D », cours ENIB module Réalité Virtuelle
- Nicolas JANEY :
  - LIFC, Département Informatique Université de Franche Comté : http://raphaello.univ-fcomte.fr/IG/Java3D/Java3D.htm
- Dennis J. Bouvier :
  - « Getting Started with the Java 3D API », Sun
- Le site de la communauté Java3D :
  - https://java3d.dev.java.net/
- Le tutorial Java3D :
  - http://java.sun.com/developer/onlineTraining/java3d/