Jean-Marc LE GALLIC

Ingénieur TGCE à l'IGN depuis 1986

Enseignant à l'ENSG depuis 1999 – 2004

Service de la Recherche janvier 2005

jean-marc.le-gallic@ign.fr

Tél : 01 43 98 80 00 poste 7715

Java 3D

Présentation générale de Java 3D

- API destinée à la création d'univers en 3 dimensions
- Extension de Java 2 (à installer dans le répertoire du JDK ou JRE)
- Formes complexes, éclairages, textures, animations …
- Utilise les concepts d'OpenGL et de DirectX et en particulier la notion de « **graphe de scène** »
- Peut importer des scènes 3D issues d'autres logiciels (VRML)
- Bénéficie de tous les avantages de Java (indépendant du matériel, exécution dans un butineur …)

Le graphe de scène

Représente les objets de « l'univers 3D » sous forme d'arbres

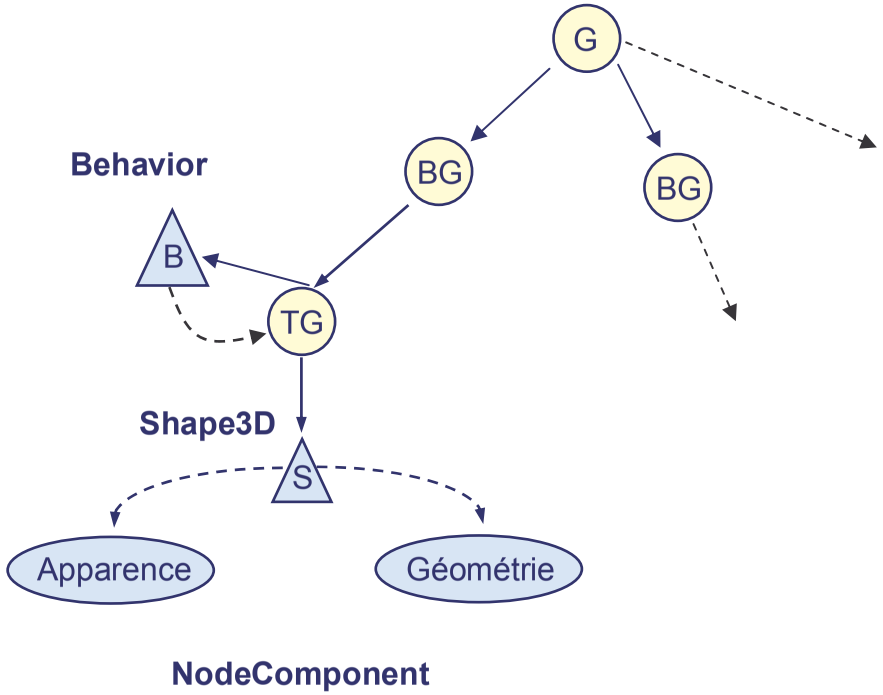
Les « Nœuds »

- Group
- BranchGroup
- TransformGroup

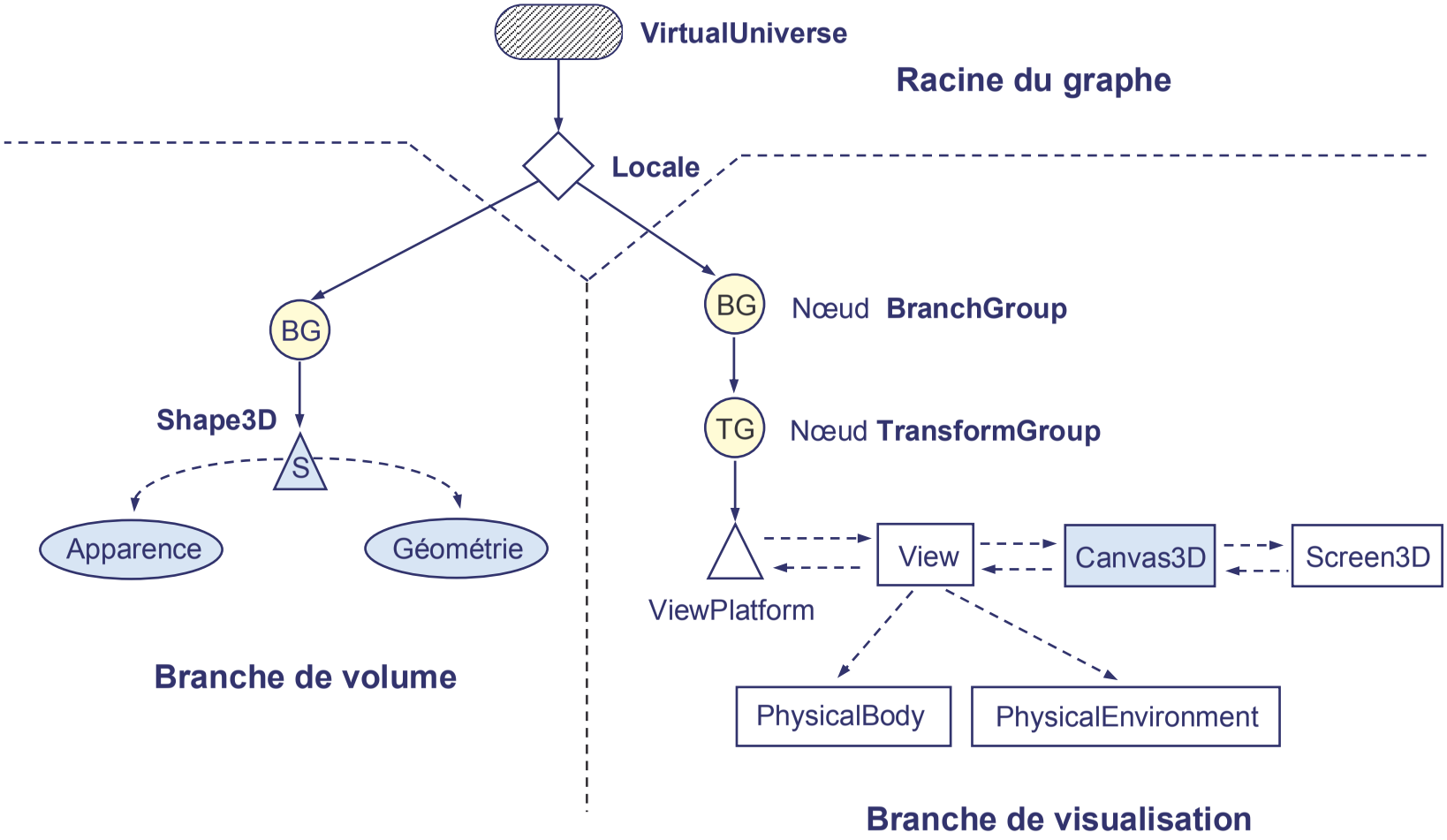
Les « Feuilles »

- Shape3D
- Light
- Behavior

Les Relations



Le graphe de scène d'une application



La racine du graphe

Virtual Universe

- Contient l'ensemble des scènes

Locale

- Point de départ de la scène
- Définit les caractéristiques de la scène (origine des coordonnées, branche(s) de volume, branche de visualisation)
- Détermine la position des objets visuels dans l'univers 3D

La branche de visualisation

ViewPlatform

- Détermine la position et l'orientation du point de vue de l'utilisateur

View

- Regroupe toutes les caractéristiques du processus de rendu (utilisation ou non de la perspective, politique de réaffichage …)
- Possède une référence sur « **Canvas3D** », « **PhysicalBody** » et « **PhysicalEnvironment** »

La branche de volume

Le « Group »

- Regrouper des nœuds
- Possède les méthodes d'ajout et de suppression des nœuds

Le « BranchGroup »

- Classe dérivée de Group
- Départ d'une branche de l'arbre
- Possède la méthode « `detach()` »

Le « TransformGroup »

- Classe dérivée de Group
- Associé à un objet « `Transform3D` » qui définit la transformation

L'optimisation du graphe de scène

La compilation des branches

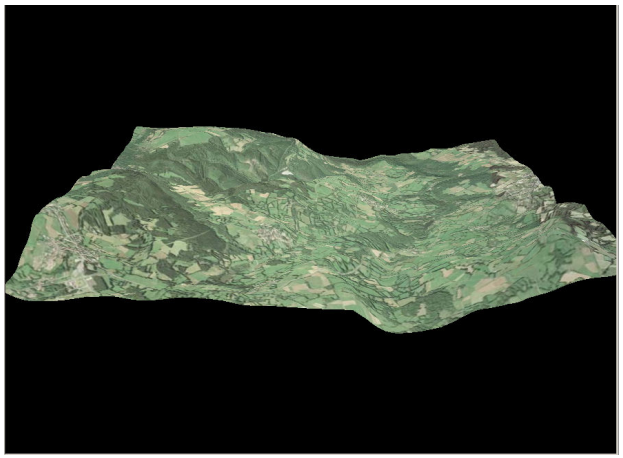
- Optimise la gestion de la branche compilée
- Méthode `compile` appliquée au BranchGroup

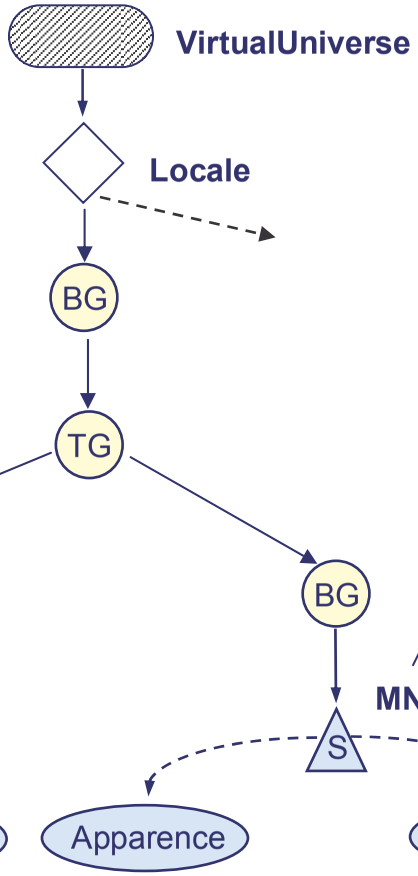
Les « bits d'aptitude » ou capacités d'un nœud

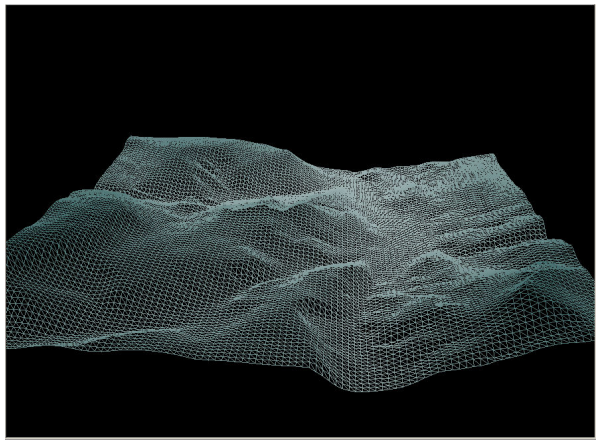
- Associe un droit (en lecture, écriture, maj …) à un nœud
- Nécessaire si le nœud est « vivant » ou compilé

Les limites d'influence

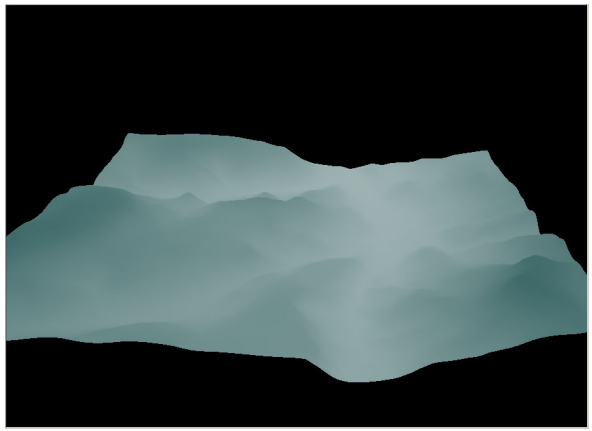
Modification d'un objet 3D



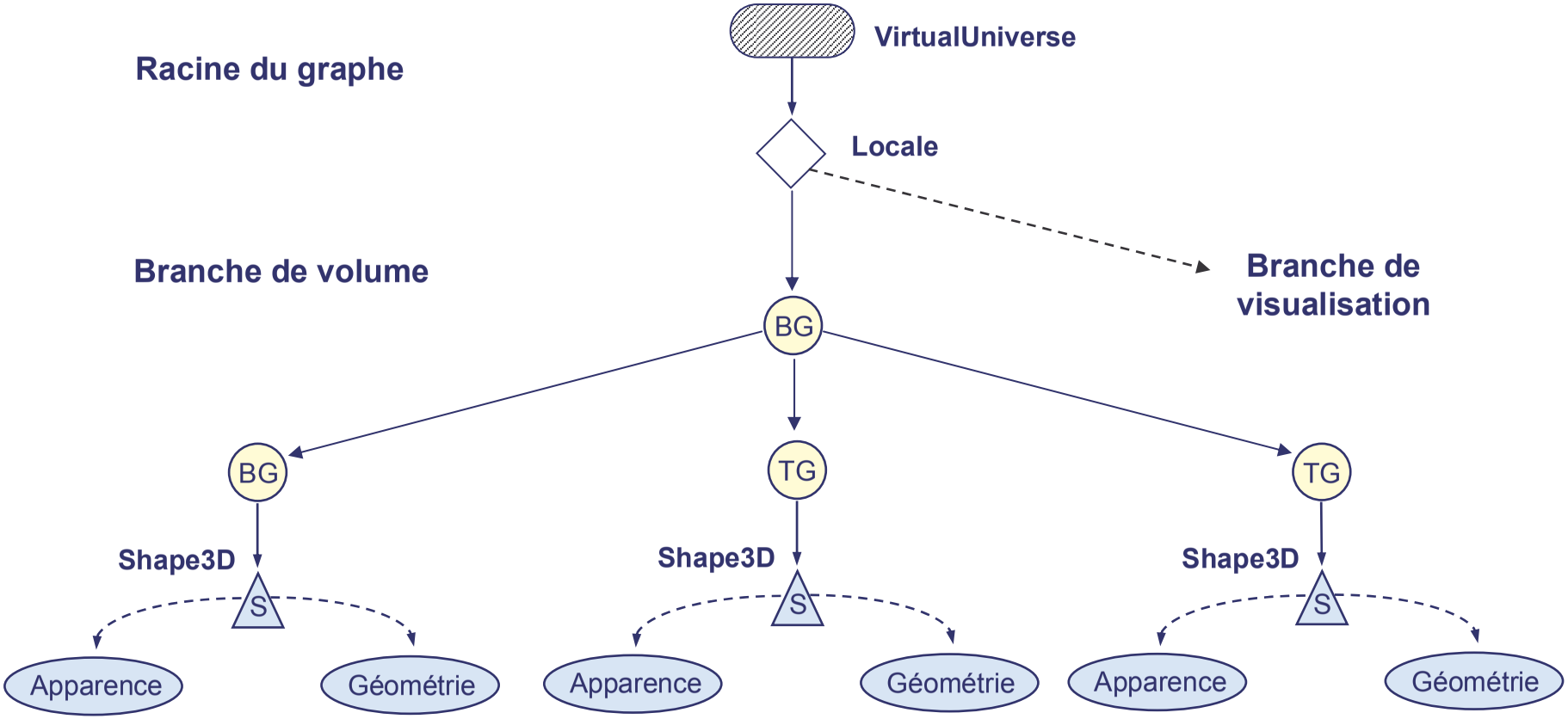




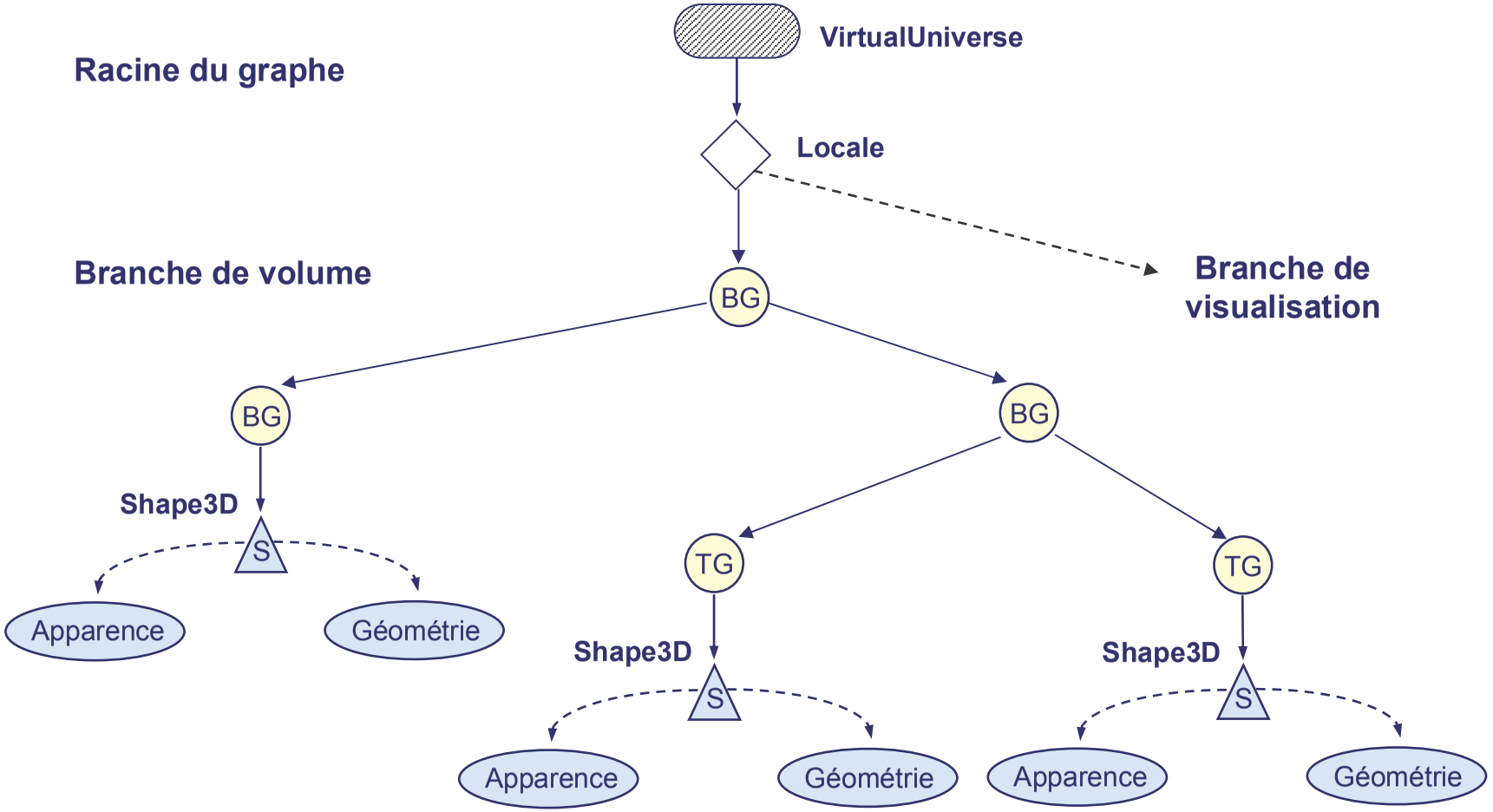
MNT



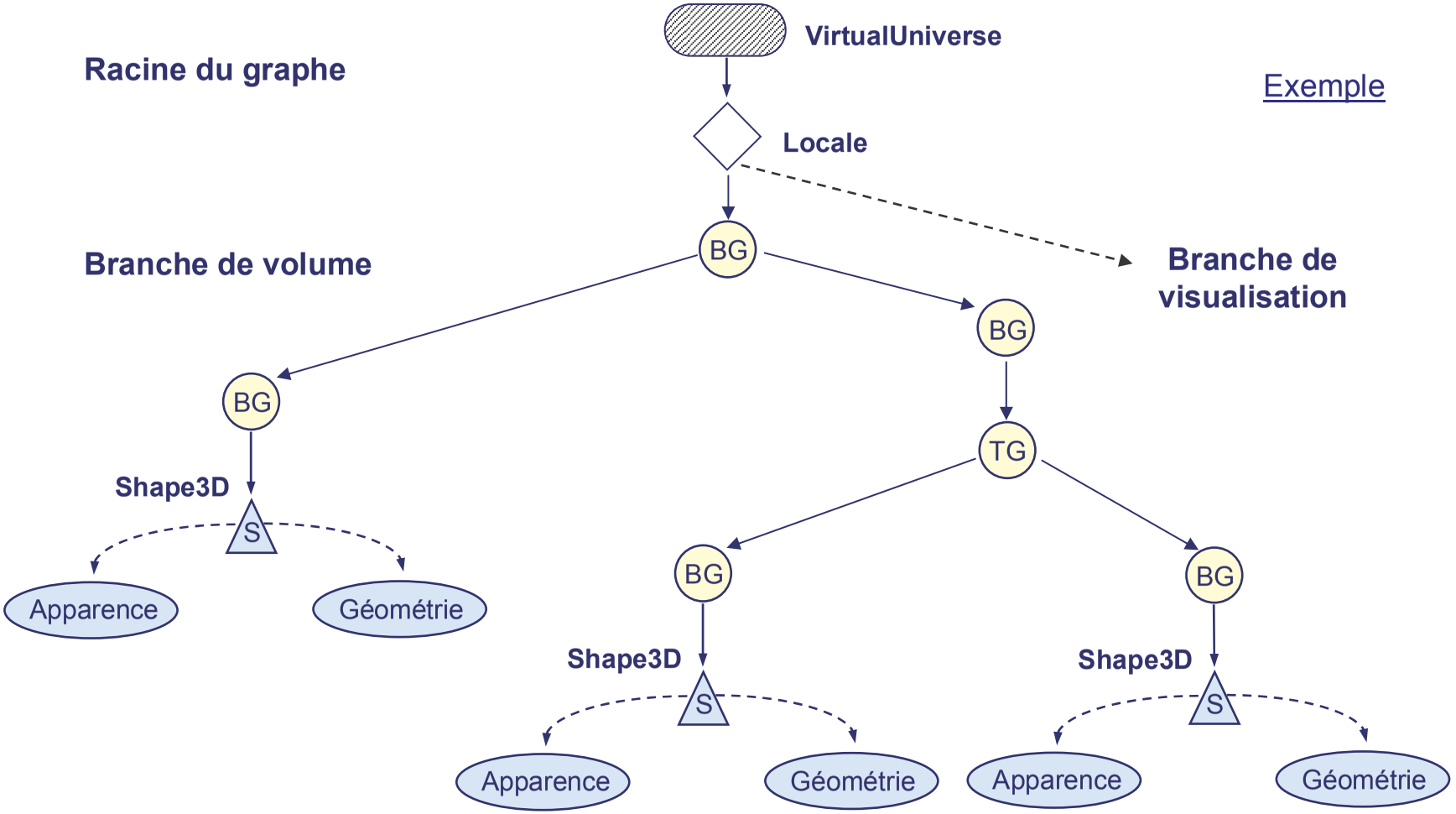
Exemple de graphe de scène



Exemple de graphe de scène



Exemple de graphe de scène



Règles de construction d'un univers 3D

8 étapes

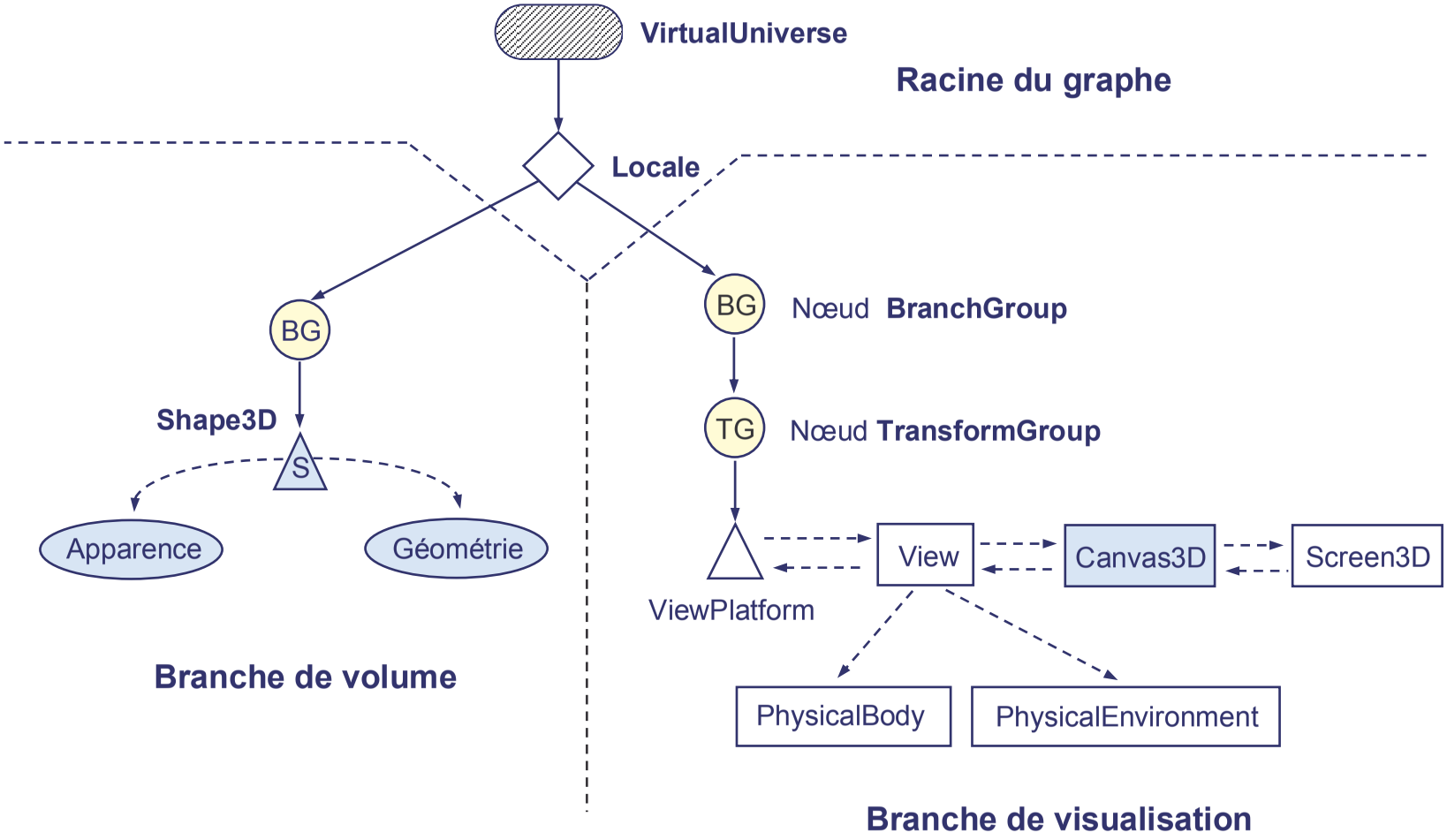
- Créer une classe qui hérite de « JFrame » ou « JApplet »
- Créer un objet Canvas3D et l'ajouter au conteneur de la frame
- Créer un objet « VirtualUniverse »
- Créer un objet « Locale » lié au « VirtualUniverse »
- **Créer la branche de visualisation et l'attacher à l'objet « Locale »**
- Construire la scène 3D (créer la branche de volume)
- Attacher la scène 3D à l'univers via l'objet « Locale »
- Compiler l'arborescence de la scène

Règles de construction d'un univers 3D

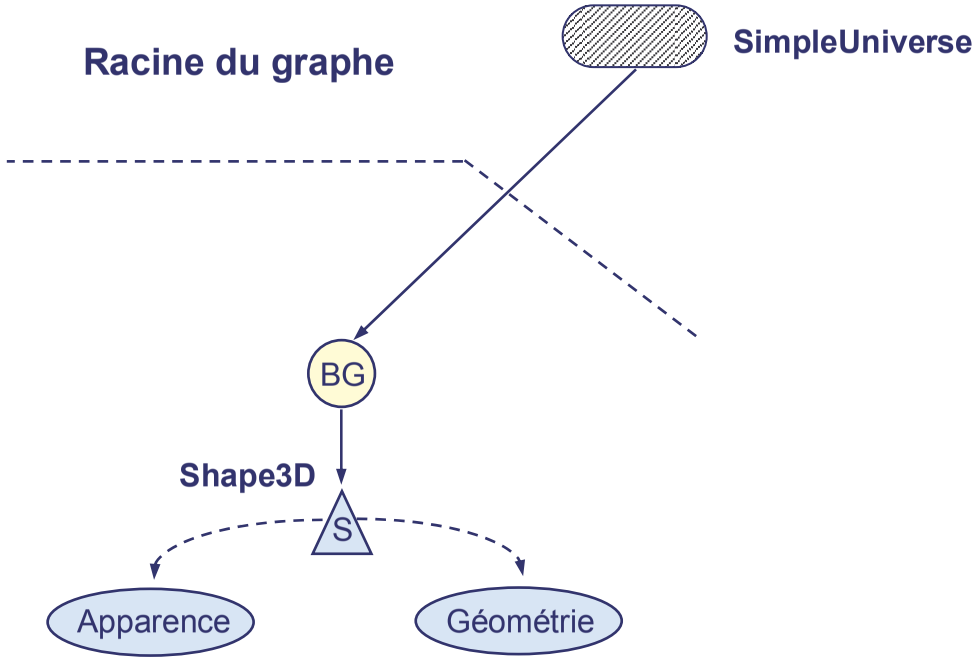
Créer la branche de visualisation et « l'attacher » à l'objet Locale

- Créer un objet « ViewPlatform » et l'attacher au « TransformGroup »
- Créer un objet « View »
- Créer un objet « PhysicalBody » et un objet « PhysicalEnvironment»
- Attacher « PhysicalBody », « PhysicalEnvironment », « ViewPlatform » et « Canvas3D » à l'objet « View »

Le graphe de scène d'une application

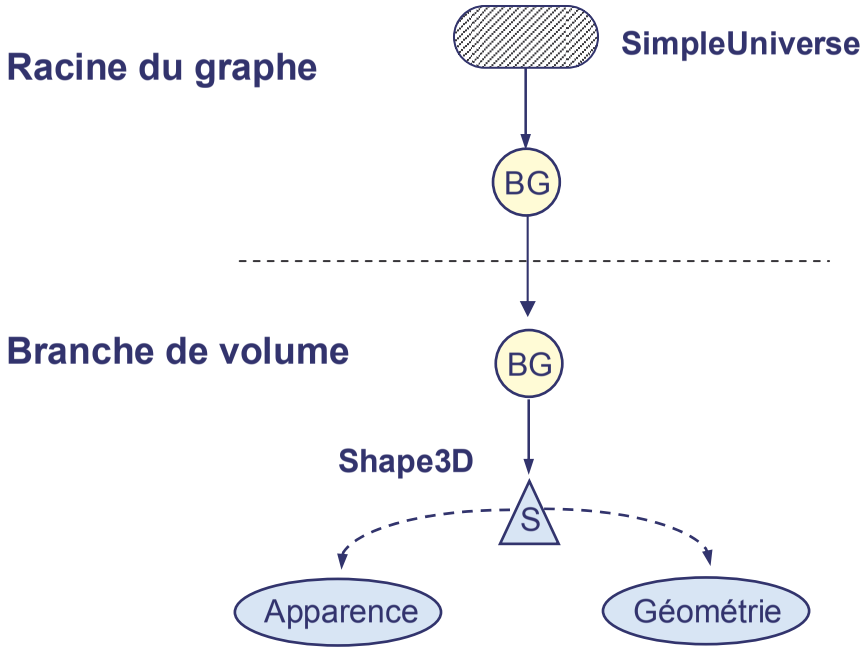


Le graphe de scène d'une application



Branche de volume

Le SimpleUniverse



Règles de construction simplifiée

7 étapes

- Créer une classe qui hérite de « JFrame » ou « JApplet »
- Créer un objet « Canvas3D » et l'ajouter au conteneur de la frame
- Créer un objet « SimpleUniverse » avec référence sur « Canvas3D »
- Créer un « branchement » entre le « SimpleUniverse » et la scène
- Construire la scène 3D (branche de volume)
- Attacher la scène à l'univers via le « branchement »
- Compiler l'arborescence de la scène

Un exemple simple : le ColorCube

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

public class Scene3D extends JFrame
{
      private SimpleUniverse univers;
      private BranchGroup racine;

      public Scene3D (int l, int L )   {
            setSize ( l, L ) ;

            Container conteneur = this.getContentPane ( ) ;
            conteneur.setLayout ( new BorderLayout ( ) ) ;
```

Un exemple simple : le ColorCube

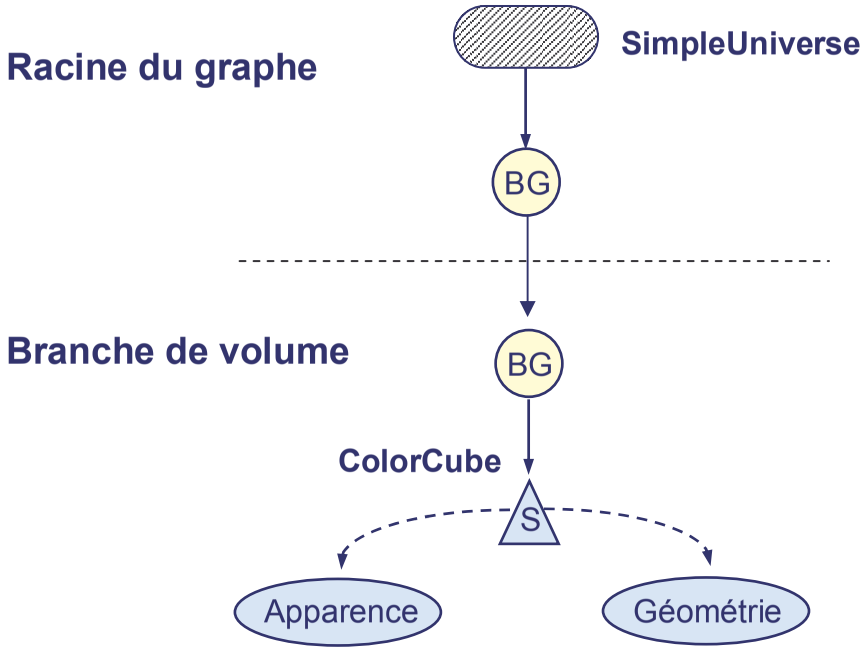
```
    GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration ( ) ;
    Canvas3D c = new Canvas3D ( config ) ;
    conteneur.add ( "Center",c ) ;

    univers = new SimpleUniverse ( c ) ;
    univers.getViewingPlatform ( ).setNominalViewingTransform ( ) ;

    racine = new BranchGroup ( ) ;

    racine.addChild ( createBrancheVolume ( ) ) ;
    univers.addBranchGraph ( racine ) ;

    setVisible ( true ) ;
}
```

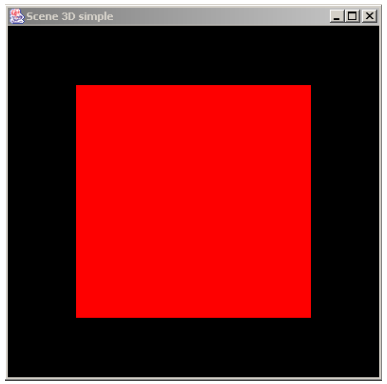
Paramétrage par défaut de la ViewPlatform

Un exemple simple : le ColorCube

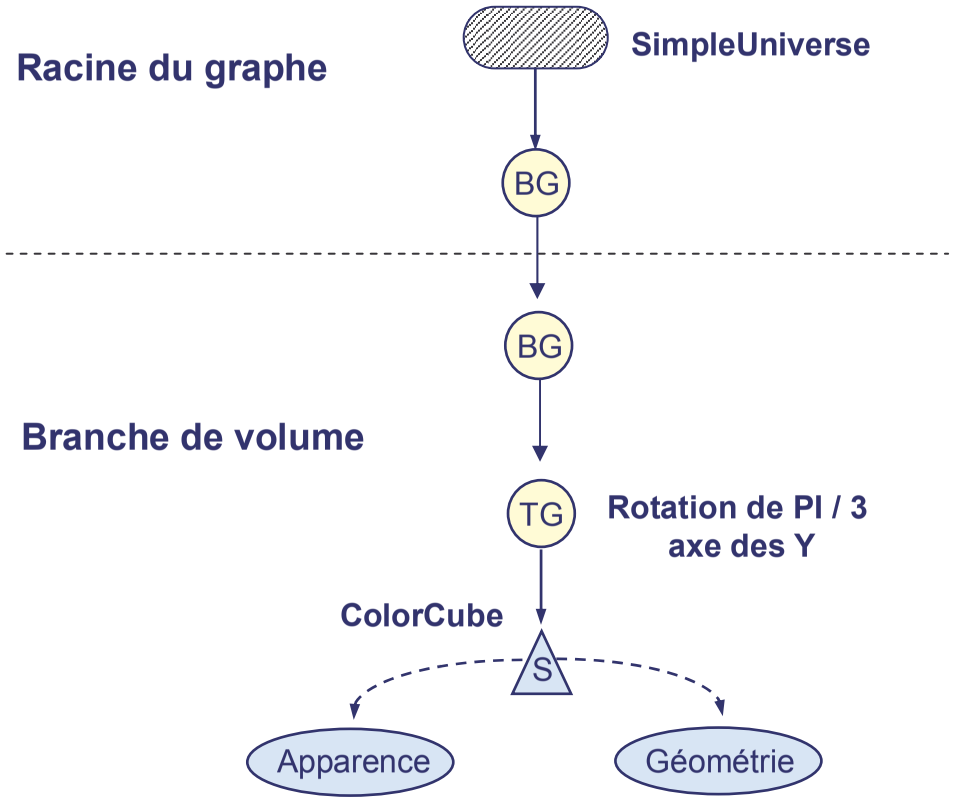
```
    private BranchGroup createBrancheVolume ( )
    {
        BranchGroup  racine = new BranchGroup ( ) ;

        racine.addChild ( new ColorCube (0.5) ) ;
        racine.compile ( ) ;
        return racine ;
    }

    public static void main (String s [] ) {
        Scene3D s3D = new Scene3D ( 400,400 ) ;
    }
}
```

Un exemple simple : le ColorCube





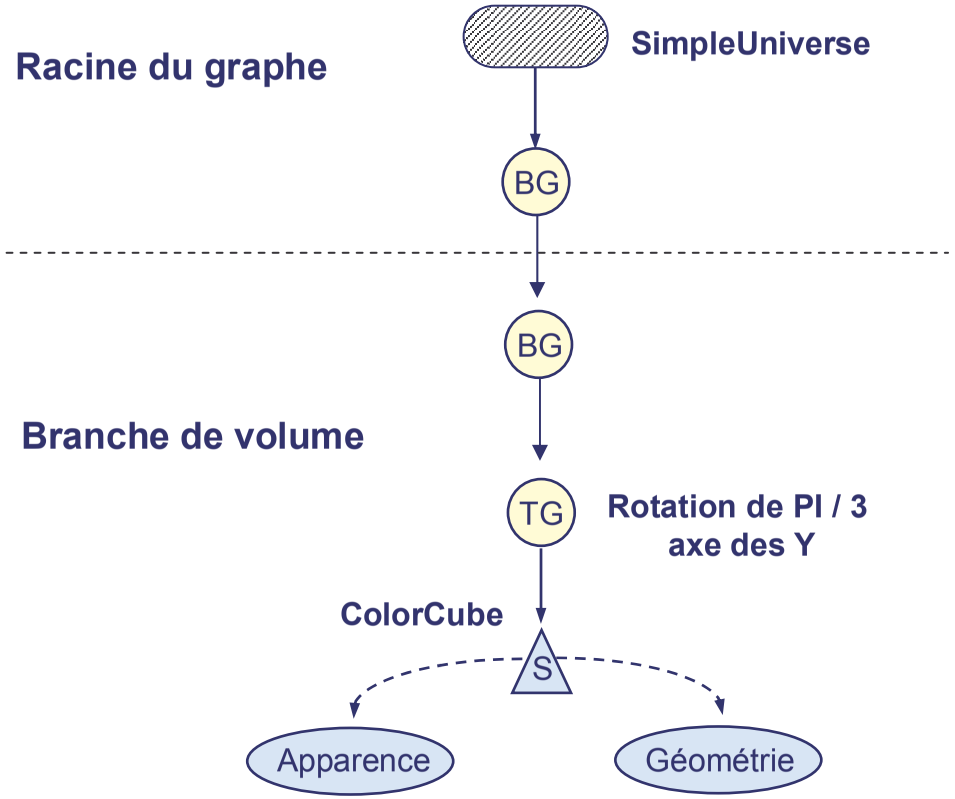
Les transformations 3D

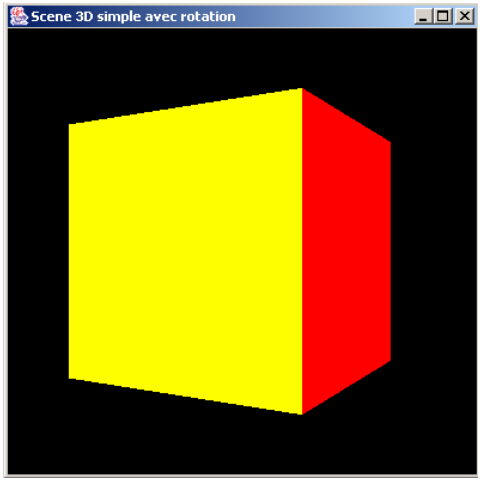


Les transformations 3D

```
private BranchGroup createBrancheVolume ( )
{
    BranchGroup racine = new BranchGroup ( ) ;

    TransformGroup transform = new TransformGroup ( ) ;
    Transform3D rotationY = new Transform3D ( ) ;

    rotationY.rotY ( Math.PI / 3.0d ) ;
    transform.setTransform ( rotationY ) ;

    racine.addChild ( transform ) ;
    transform.addChild ( new ColorCube (0.5) ) ;

    racine.compile ( ) ;
    return racine ;
}
```

Les transformations 3D

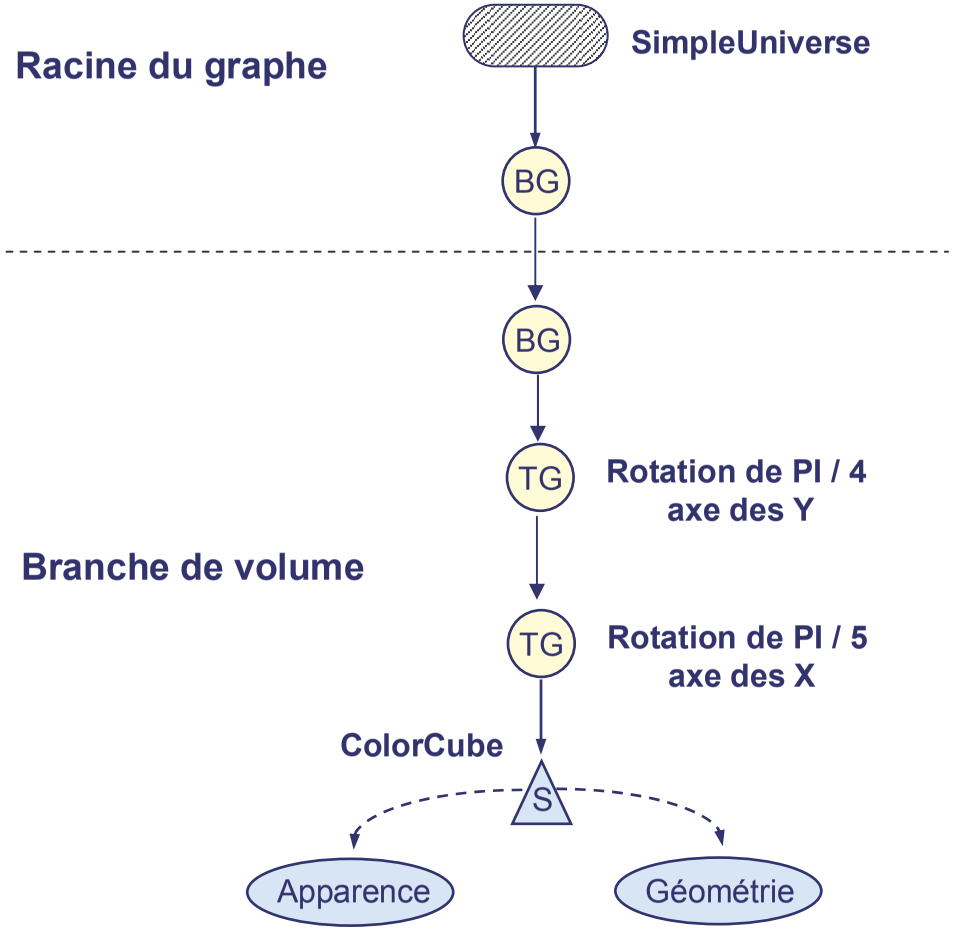


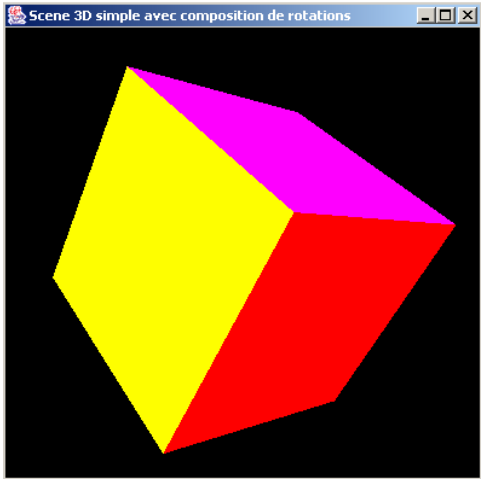


Les transformations 3D

```
BranchGroup racine = new BranchGroup ( ) ;
TransformGroup transform = new TransformGroup ( ) ;

Transform3D rotationY = new Transform3D ( ) ;
Transform3D rotationX = new Transform3D ( ) ;

rotationY.rotY ( Math.PI / 4.0d ) ;
rotationX.rotX ( Math.PI / 5.0d ) ;

rotationY.mul ( rotationX ) ;
transform.setTransform ( rotationY ) ;

racine.addChild ( transform ) ;
```

Les transformations 3D





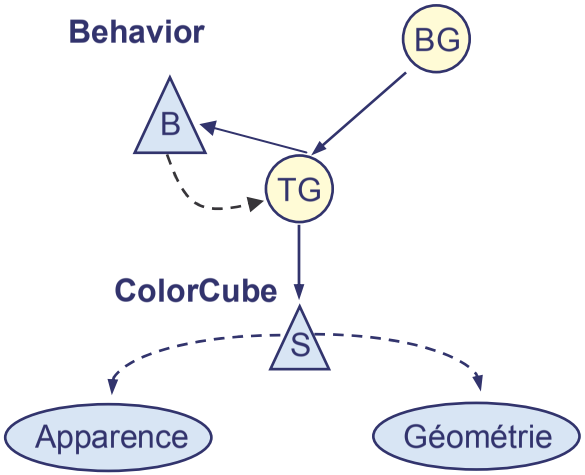
Les interactions et les animations

- **Interactions** : l'utilisateur agit sur les objets de la scène
  (souris – clavier)

- **Animations** : les comportements sur les objets de la scène sont programmés en fonction d'un temps écoulé

La classe Behavior

- La classe « Behavior » gère les interactions et les animations
- Ajouté à un « TransformGroup »



Comportements et stimuli

- **Comportement** : l'association d'un **stimulus** (événement) et d'une réponse à ce stimulus. La réponse génère un changement du graphe de scène
- Les stimuli sont définis dans la classe « WakeupCriterion »
- Un comportement possède trois propriétés :
  - un état (active - désactive le comportement) : méthode `setEnable`
  - une zone d'influence : classes `Bounds` ou `BoundingLeaf`
  - une condition d'activation : méthode `WakeupOn`

Les classes de stimuli prédéfinis

Méthodes	Actions
WakeupAWTEvent	Réagit à un événement AWT
WakeupElaspsedTime	Réagit à l'écoulement d'une durée
WakeupOnCollisionEntry WakeupOnCollisionMovement WakeupOnCollisionExit	Réagit à la collision d'objets
WakeupOnTransformChange	Réagit à la modification du TransformGroup
WakeupOnActivation WakeupOnDeactivation	Réagit à l'activation/désactivation d'un comportement
WakeupOnSensorEntry WakeupOnSensorExit	Réagit à l'entrée/sortie d'une zone de la scène 3D
...	

Création d'une classe de comportements

- Relier l'objet définissant le comportement à l'objet sur lequel doit s'appliquer ce comportement (rôle du constructeur)

- Ecrire la méthode « `initialize` » dans laquelle on appelle « `wakeUpOn` » en précisant, en paramètre, quels sont les stimuli auxquels cette classe veut réagir

- Ecrire la méthode « `processStimulus` » décrit les actions

Exemple

Interactions à l'aide de la souris

- La classe « MouseBehavior »
  - « MouseRotate », « MouseTranslate », « MouseZoom »

Mouvement	Action
Rotation	Le bouton gauche de la souris
Translation	Le bouton droit de la souris
Zoom	Le bouton gauche de la souris + touche ALT du clavier

Interactions à l'aide de la souris

```
TransformGroup transform = new TransformGroup ( ) ;

transform.setCapability ( TransformGroup.ALLOW_TRANSFORM_READ ) ;
transform.setCapability ( TransformGroup.ALLOW_TRANSFORM_WRITE ) ;

MouseTranslate mouseTr = new MouseTranslate ( transform ) ;

BoundingSphere boundsTr = new BoundingSphere ( new Point3d ( ) ,100.0 ) ;
mouseTr.setSchedulingBounds ( boundsTr ) ;

transform.addChild ( mouseTr ) ;
```

Exemple

Interactions à l'aide du clavier

- La classe « KeyNavigatorBehavior »

Mouvement	Action
Vers la gauche	La flèche gauche
Vers la droite	La flèche droite
Vers le haut	Page up
Vers le bas	Page down
Vers l'avant	La flèche haut
Vers l'arrière	La flèche bas

Interactions à l'aide du clavier

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

KeyNavigatorBehavior clavier = new KeyNavigatorBehavior (
            this.univers.getViewingPlatform ( ).getViewPlatformTransform ( ) ) ;

clavier.setSchedulingBounds ( bounds ) ;

transform.addChild ( clavier ) ;
```

Les animations

- Une **animation** est un type de comportement qui modifie une scène 3D de manière prédéterminée en fonction d'un temps écoulé
- **Stimuli** : la classe « WakeupElaspedTime »
- **Interpolateur :** une sous-classe de la classe « `Behavior` » qui manipule les paramètres des objets en fonction d'un argument temporel

Les classes « Interpolator »

- « PositionInterpolator »
- « RotationInterpolator »
- « ColorInterpolator »
- « TransparencyInterpolator »

Les classes d'interpolateurs fournissent également des méthodes permettant de combiner les actions

La classe « Alpha »

- Un objet « `Interpolator` » modifie un objet « `TransformGroup` » par une action spécifique qui répond à la valeur d'un objet de la classe « `Alpha` »
- Les objets de la classe « `Alpha` » sont utilisés pour créer une fonction de variation temporelle qui produit une valeur comprise entre 0 et 1
- Les constructeurs de la classe « `Alpha` » peuvent avoir jusqu'à 10 paramètres permettant de gérer des mouvements très complexes (accélération, amortissement, oscillation pendulaires …)

Le « RotationInterpolator »

```
TransformGroup transform = new TransformGroup ( ) ;
transform.setCapability ( TransformGroup.ALLOW_TRANSFORM_WRITE ) ;

Alpha rotationAlpha = new Alpha ( -1, 4000 ) ;

RotationInterpolator rotator = new RotationInterpolator ( rotationAlpha, transform ) ;

BoundingSphere bounds = new BoundingSphere ( ) ;
rotator.setSchedulingBounds ( bounds ) ;

transform.addChild ( rotator ) ;
transform.addChild ( new ColorCube(0.5) ) ;

racine.addChild ( transform ) ;
```

Exemple

Les animations plus complexes

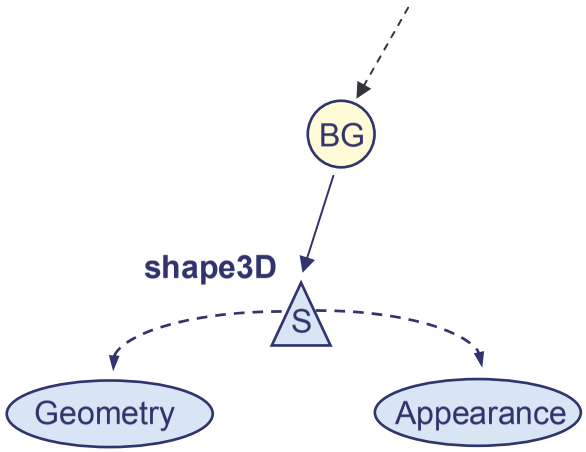
```
// Accélération , stabilisation, freinage, arrêt 2 secondes
Alpha rotor1Alpha1 = new Alpha (-1, Alpha.INCREASING_ENABLE ,0 ,0, 10000,
                                2000, 2000,10000,2000,0 ) ;
// 5 tours avant le freinage
RotationInterpolator rotator1 = new RotationInterpolator ( rotor1Alpha1, tg3 ,
                        new Transform3D ( ) , 0.0f, 2.0f*( float )Math.PI*5.0f);

BoundingSphere bounds = new BoundingSphere ( ) ;
rotator.setSchedulingBounds ( bounds ) ;

transform.addChild ( rotator ) ;
transform.addChild ( new ColorCube(0.5) ) ;

racine.addChild ( transform ) ;
```

Exemple 1

Exemple 2

Les formes 3D

- La classe « shape3D »
  - « Geometry »
  - « Appearance »



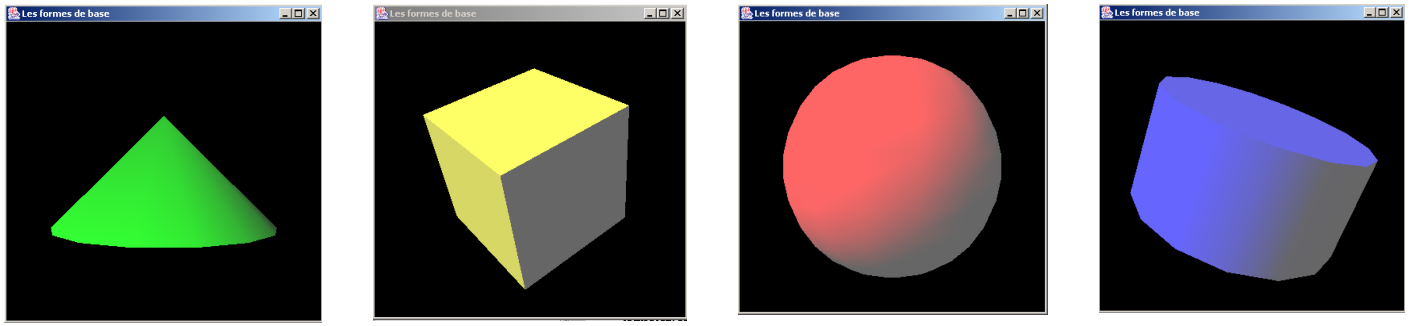
- Les méthodes « setGeometry » et « setAppearance »
  « getGeometry » et « getAppearance »

Création de formes 3D

- Le « ColorCube »
- Les formes de base (cylindre, cône, sphère, parallélépipède) utilisées seules ou assemblées
- Construction point par point des formes en modifiant les nœuds « Geometry » et « Appearance »
- Importer des scènes créées par d'autres logiciels (Lightwave, Wavefront, VRML …)

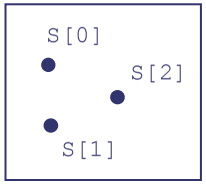
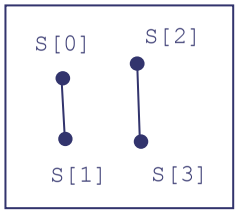
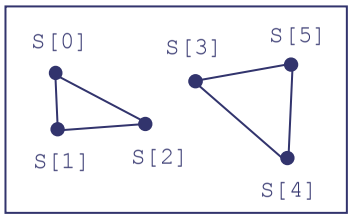
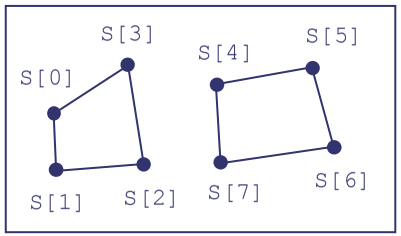
La géométrie des formes 3D

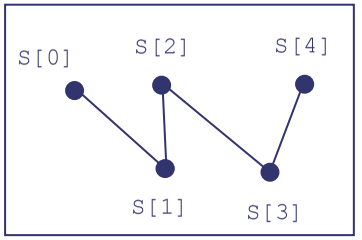
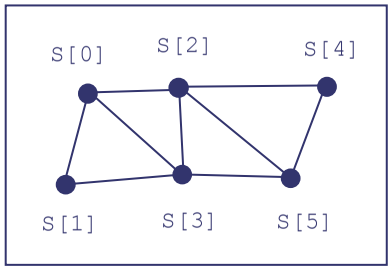
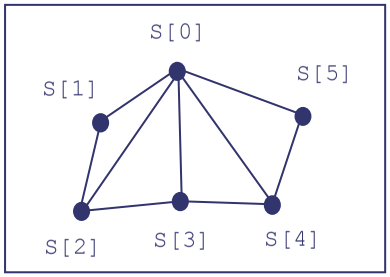
- La classe « Geometry »
  - Les formes de base (cylindre, cône, sphère, parallélépipède et le « ColorCube ») ont leur géométrie prédéfinie



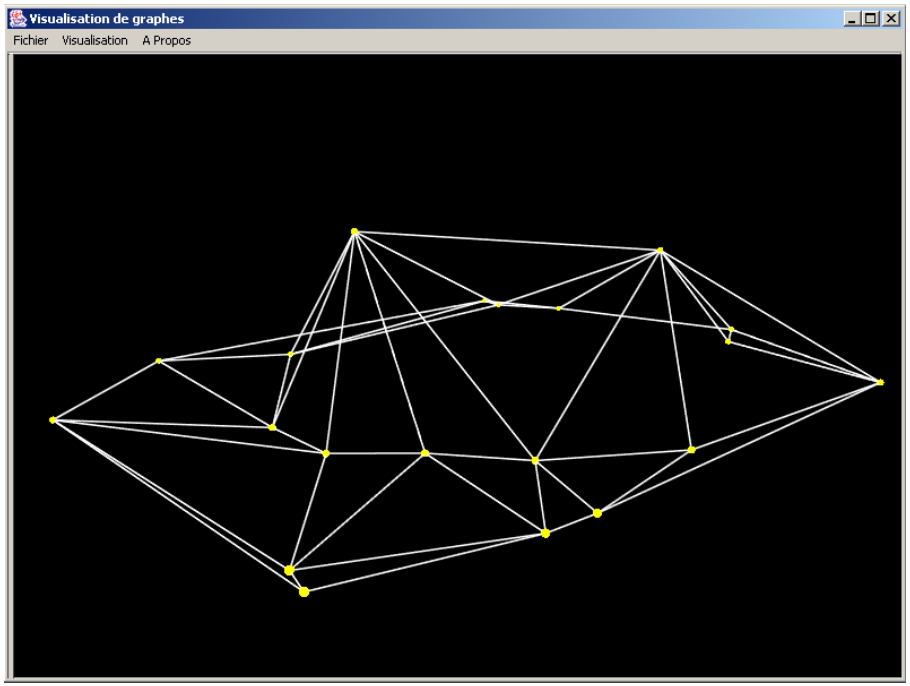
  - Les formes complexes créées à partir de la classe « GeometryArray » et « IndexedGeometryArray »

La classe « GeometryArray »

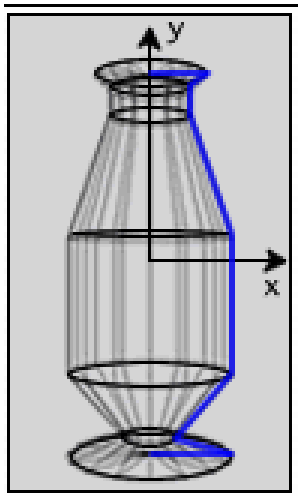
	PointArray
	LineArray
	TriangleArray
	QuadArray

	LineStripArray
	TriangleStripArray
	TriangleFanArray

Exemples de formes 3D complexes



Classe « LineStripArray »



Classe « NormalGenerator » et méthode « generateNormals »

L'apparence des formes 3D

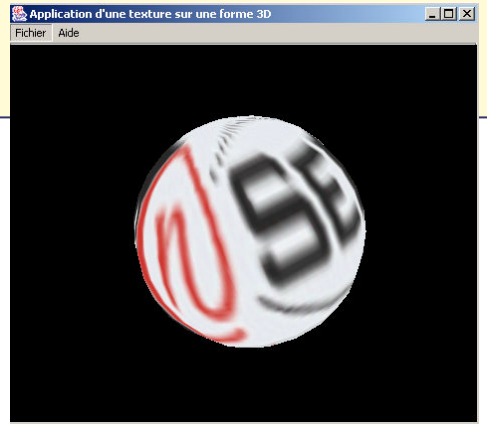
- La classe « Appearance »
  - La couleur (« ColoringAttributes » et « Material »)
  - Le mode d'affichage des facettes (« PointAttributes », « LineAttributes » et « polygonAttributes »)
  - La transparence et le rendu (« TransparencyAttributes » et RenderingAttributes »)
  - L'application de textures (« TextureAttributes et « Texture »)

Changer la couleur d'une forme 3D

```
Appearance BoxApp = new Appearance ( ) ;

 BoxApp.setColoringAttributes ( new ColoringAttributes (1, 0, 0,
                                        ColoringAttributes.SHADE_GOURAUD ) ) ;
 box.setAppearance ( BoxApp ) ;

 Appearance SphereApp = new Appearance ( ) ;
 SphereApp.setColoringAttributes ( new ColoringAttributes (0, 1, 0,
                                         ColoringAttributes.SHADE_GOURAUD ) ) ;
sphere.setAppearance ( SphereApp ) ;
```

Appliquer une texture à une forme 3D

```
Appearance appSphere = new Appearance ( ) ;

Sphere sphere = new Sphere ( 2.0f ) ;

TextureLoader TextureLogo = new TextureLoader ( " LogoPushnsee.gif " , frame) ;
Texture texture = TextureLogo.getTexture ( ) ;

appSphere.setTexture ( texture ) ;

sphere.setAppearance ( appSphere ) ;
```



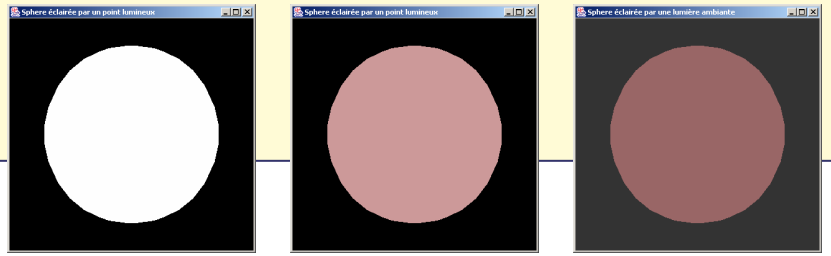
L'éclairage d'une scène 3D

- La classe « Material »
  - La lumière émise (« emissiveColor »)
  - La lumière spéculaire (« specularColor »)
  - La lumière diffuse (« diffuseColor »)

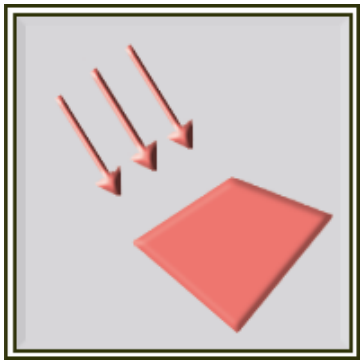
- La lumière ambiente (« AmbientLight »)
- La lumière directionnelle (« DirectionalLight »)
- Le point lumineux (« PointLight »)
- Le « spot » lumineux (« SpotLight »)

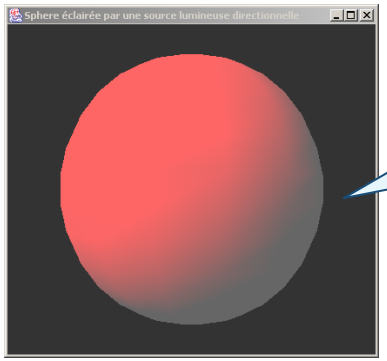
« Material » et « AmbientLight »

```
Color3f lightColor = new Color3f (1.0f, 0.0f, 0.0f ) ;
light = new AmbientLight ( lightColor ) ;
racine.addChild ( light ) ;

Sphere sphere = new Sphere ( 0.7f ) ;
Material matSphere = new Material ( new Color3f ( 0.2f , 0.2f ,0.2f ) ,   // ambiante
                                    new Color3f ( 1.0f, 1.0f, 1.0f ) ,    // diffuse
                                    new Color3f ( 1.0f, 1.0f, 1.0f ) ,    // spéculaire
                                    new Color3f ( 0.0f, 0.0f , 0.0f ) ,   // émise
                                                                64 ) ;  // brillance
 Appearance appSphere = new Appearance ( ) ;
 appSphere.setMaterial ( matSphere ) ;
 sphere.setAppearance ( appSphere ) ;
```



La classe « DirectionalLight »

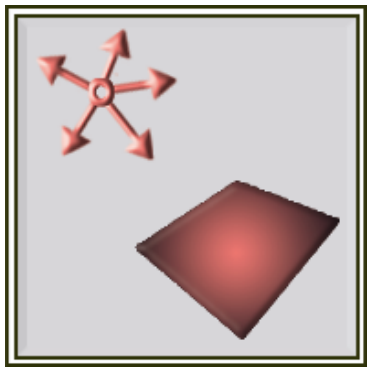


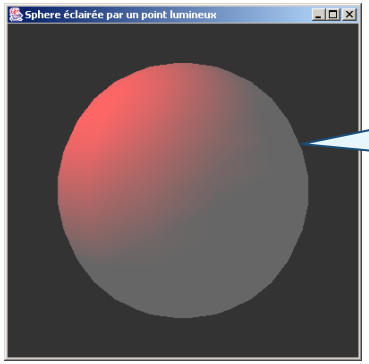


« jouer » sur la couleur « diffuse » et la couleur « émise »

```
Color3f lightColor = new Color3f (1.0f, 0.0f, 0.0f);
light = new DirectionalLight ( lightColor , new Vector3f  ( 1, -1, -1 ) ) ;

racine.addChild (light);
```

La classe « PointLight »

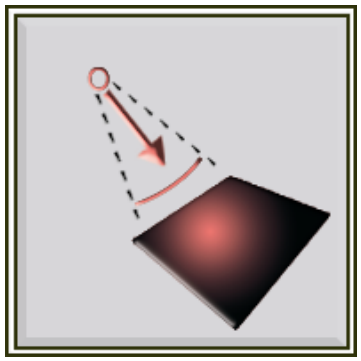


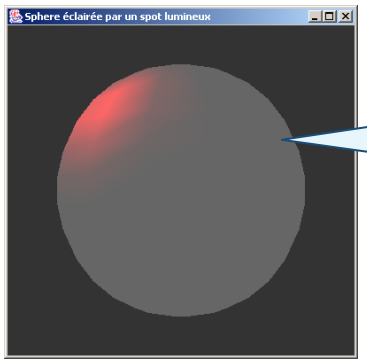


« jouer » sur la couleur « diffuse » et la couleur « émise »

```
Color3f lightColor = new Color3f (1.0f, 0.0f, 0.0f);
light = new PointLight ( lightColor,  new Point3f ( -1, 1, 1 ) ,        // position
                                      new Point3f ( 1f, 0.5f, 0 ) ) ;  // attenuation

racine.addChild (light);
```

La classe « SpotLight »





« jouer » sur la couleur « diffuse » et la couleur « émise »

```
Color3f lightColor = new Color3f (1.0f, 0.0f, 0.0f);
light = new SpotLight ( lightColor,    new Point3f ( -1, 1, 1 ) ,      // position
                                       new Point3f (1, 0, 0 ) ,        // atténuation
                                       new Vector3f (1, -1, -1 ) ,     // direction
                                       ( float ) Math.PI / 4 ,         // angle
                                        50 ) ;                         // concentration
racine.addChild (light);
```