Java 3D

Laboratório de Computação Gráfica e Interfaces
Jorge Barbosa

2001

Java 3D

- API para desenvolver aplicações 3D, com capacidade de descrever ambientes virtuais complexos
- Fornece um conjunto de construções de alto nível para:
  – Descrever objectos 3D (geometria, aparência, transformação, comportamento)
  – Construir o grafo necessário para efectuar *rendering* da cena criada

Objectivos

- **Desempenho:** utiliza outros API's a um nível inferior que implementam funções gráficas optimizadas e adaptados a cada arquitectura. Ex: DirectX ou OpenGL.
- **Fácil utilização:** programação OO de alto nível.
- **Compatibilidade:** existe suporte para vários formatos de dados: programas CAD específicos, VRML, etc.

Utilização

- Modelo de programação:
  - Um programa consiste em criar instâncias de classes do Java3D, ligando-as posteriormente numa estrutura em árvore, a qual se designa por ***Scene Graph***.
- Modelo de execução:
  - Na execução, o Java 3D inicia um ciclo infinito, percorrendo os nós do *scene graph;* efectua os comportamentos descritos e o *rendering* dos objectos visíveis.

packages

javax.media.j3d - classes principais

javax.vecmath - classes úteis para definir a geometria dos objectos na cena (Point*, Color*, Vector*, TexCoord*, etc)

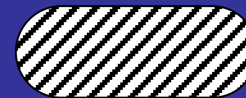
(* : 3b, 3f, 3d, 4b, 4f, 4d, etc) [Tutorial 2-16]

com.sun.j3d.utils - 4 categorias: loaders, classes de ajuda na criação da cena, geometria (Box, Sphere, ColorCube) e outros utilitários (e.g. KeyNavigatorBehavior, etc)

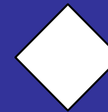
Classes Símbolo no grafo

javax.media.j3d

VirtualUniverse



Locale



View

PhysicalBody

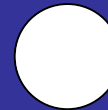
PhysicalEnvironment

Screen3D

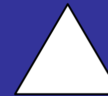
Canvas3D (extends awt.Canvas)

SceneGraphObject

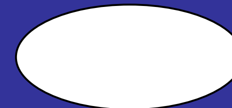
- Node
  - Group



  - Leaf



  - NodeComponent



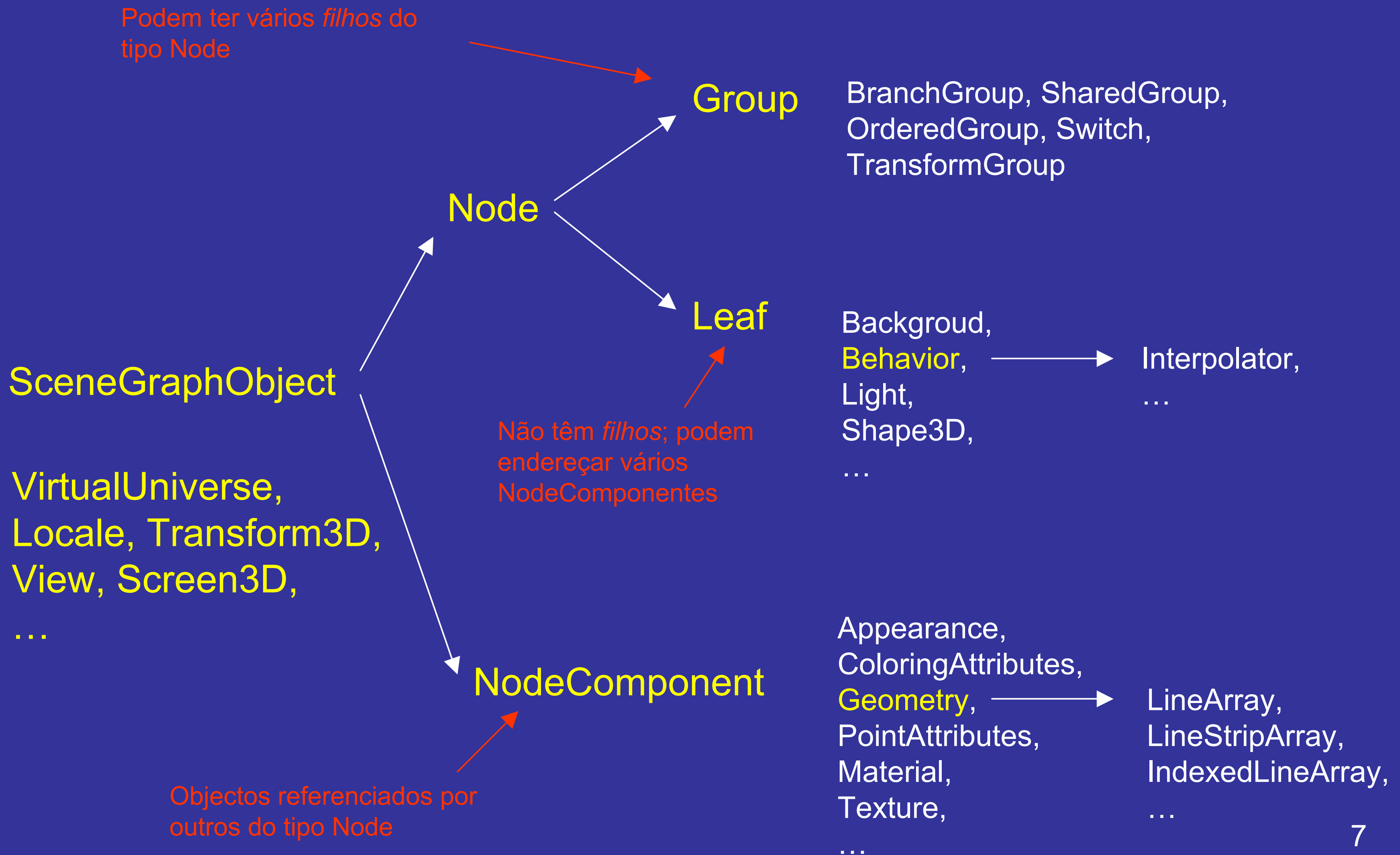
    - *Various component objects*

Transform3D

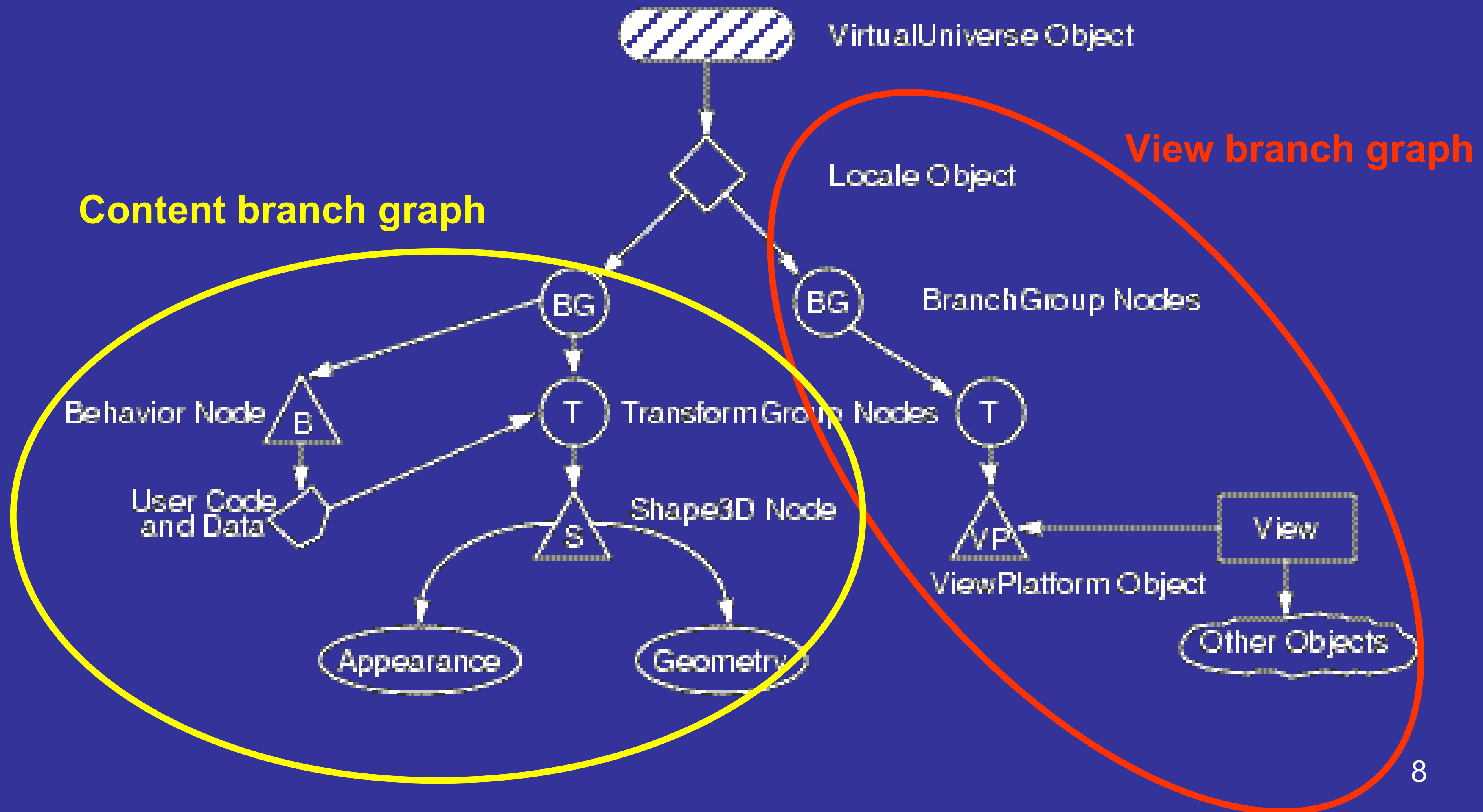
Classes para definir o *content graph*

Outros

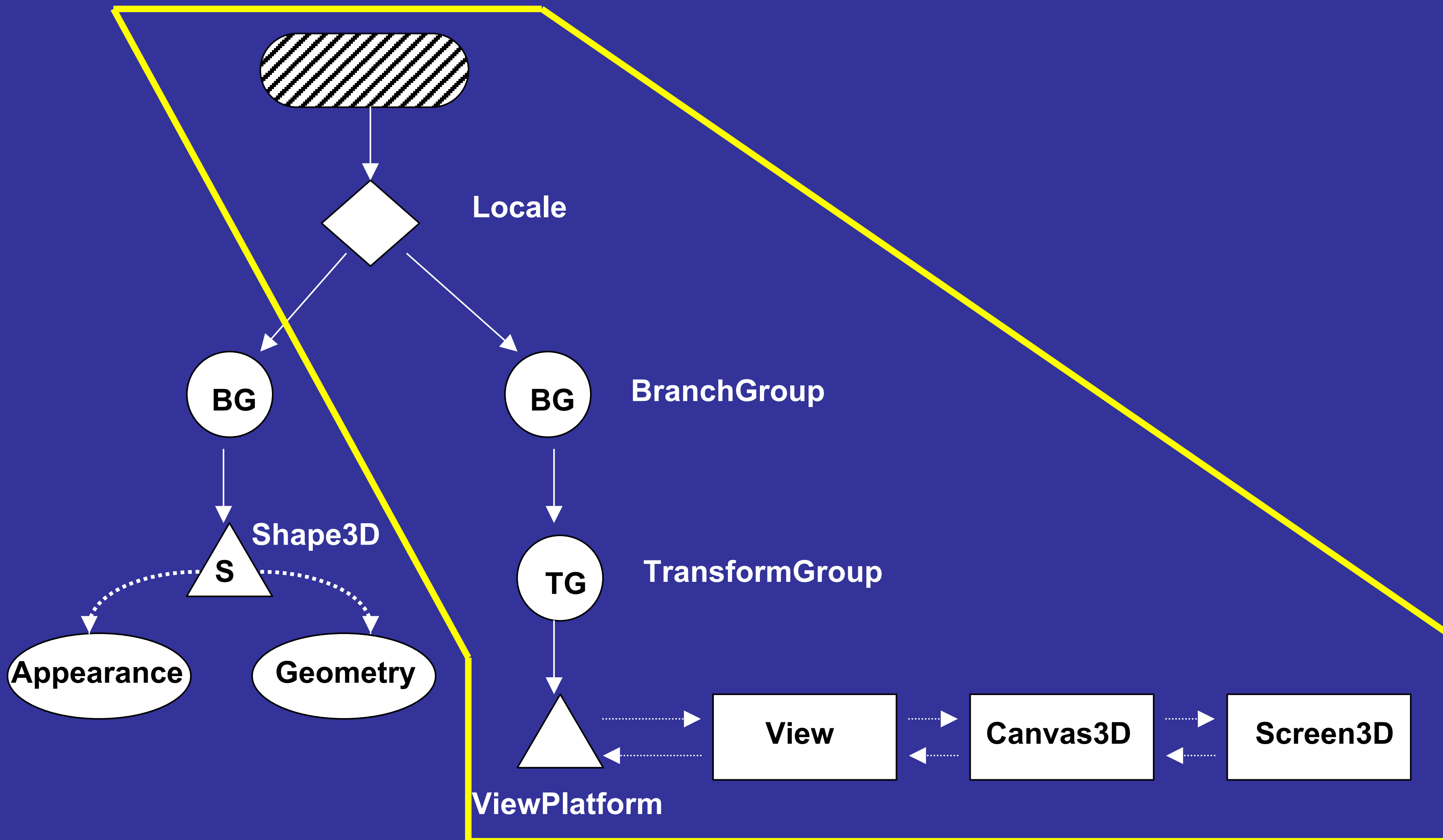
Javax.media.j3d



Exemplo de um grafo



SimpleUniverse

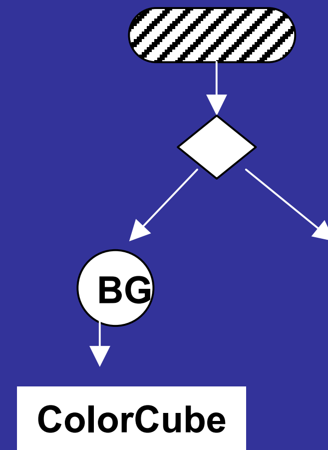


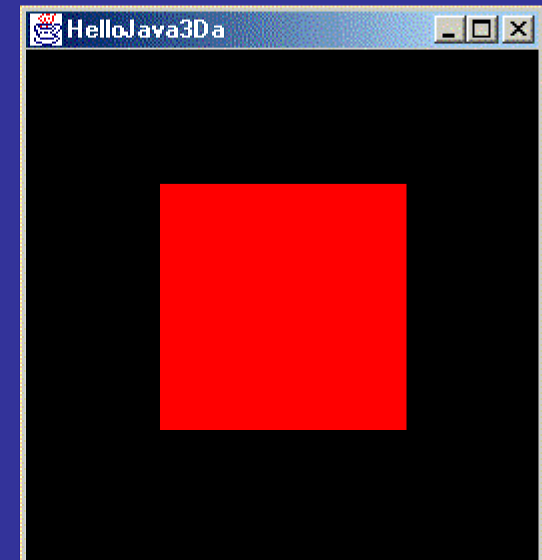
Construção de um programa

1. Criar um objecto Canvas3D
2. Criar um objecto SimpleUniverse o qual referencia Canvas3D
    a. Configurar o SimpleUniverse
3. Construir o Content Branch Graph
4. Compilar o Content Branch Graph
5. Inserir Content Branch Graph no objecto Locale do SimpleUniverse

Exemplo: HelloJava3Da

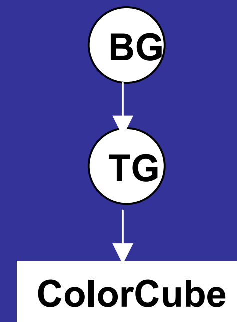
```java
public HelloJava3Da() {
      setLayout(new BorderLayout());
      Canvas3D canvas3D = new Canvas3D(null);
      add("Center", canvas3D);

      BranchGroup scene = createSceneGraph();

      // SimpleUniverse is a Convenience Utility class
      SimpleUniverse simpleU = new SimpleUniverse(canvas3D);

      // This will move the ViewPlatform back a bit so the
      // objects in the scene can be viewed.
      simpleU.getViewingPlatform().setNominalViewingTransform();

      simpleU.addBranchGraph(scene);
   } // end of HelloJava3Da (constructor)

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

      objRoot.addChild(new ColorCube(0.4));

      return objRoot;
   } // end of CreateSceneGraph method of HelloJava3Da
```

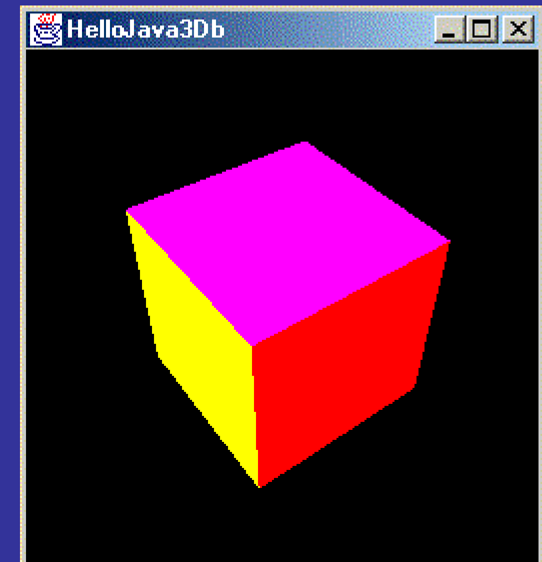




Rotação do cubo

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

  // rotate object has composited transformation matrix
  Transform3D rotate = new Transform3D();
  Transform3D tempRotate = new Transform3D();

  rotate.rotX(Math.PI/4.0d);
  tempRotate.rotY(Math.PI/5.0d);
  rotate.mul(tempRotate);       // rotate=rotate*tempRotate
  TransformGroup objRotate = new TransformGroup(rotate);

  objRoot.addChild(objRotate);
  objRotate.addChild(new ColorCube(0.4));
  // Let Java 3D perform optimizations on this scene graph.
  objRoot.compile();

  return objRoot;
} // end of CreateSceneGraph method of HelloJava3Db
```





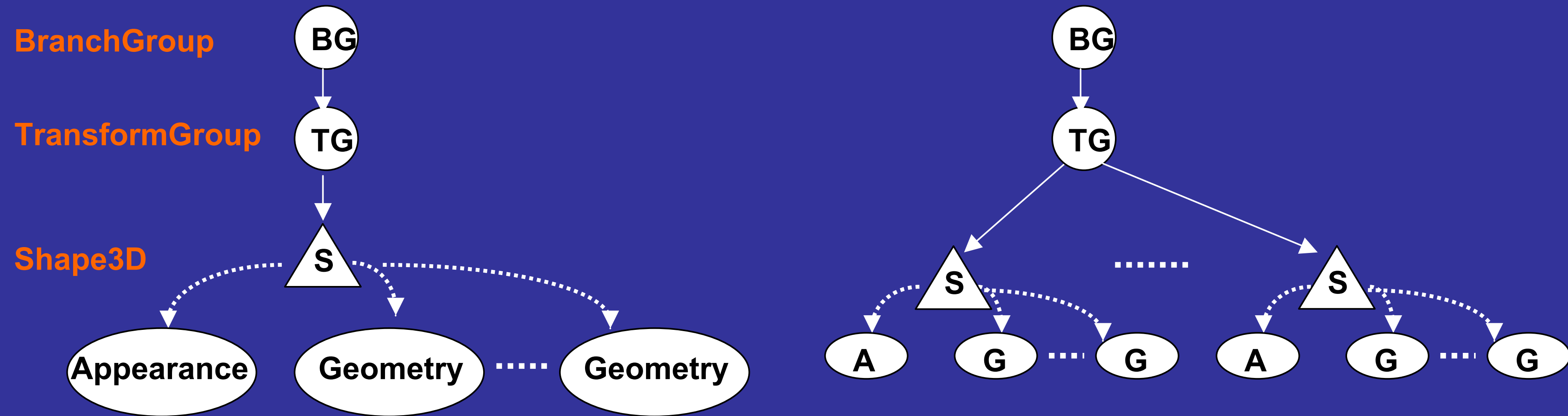
Classes usadas no exemplo

- BranchGroup
  - Usada para criar subgrafos. As suas instâncias são os únicos objectos que podem ser ligados ao Locale
  - compile() : o Java3D efectua optimizações em todo o subgrafo, mesmo que inclua outros BG
  - Objectos BG podem ser retirados ou colocados no grafo em run time se ALLOW_DETACH for true
  - Quando adicionado ao grafo, o subgrafo diz-se *Vivo*, i.e. os objectos passam a ser visualizados (*rendering)*

Classes usadas no exemplo

- Transform3D
  - Representam transformações 3D como translações, rotações e escalamentos
- TransformGroup
  - Classe usada na construção do grafo para implementar as transformações necessárias nos nós que lhe estão ligados
  - Construtor: TransformGroup(Transform3D t1)

Especificação de um Objecto na cena



Objectos endereçáveis por objectos Appearance (descrevem atributos):

ColoringAttributes

Texture

TransparencyAttributes

LineAttributes

Material

PolygonAttributes

PointAttributes

TextureAttributes

RenderingAttributes

Especificação da Geometria do Objecto

- Todos os objectos da cena são visualizados com base na representação das coordenadas dos seus vértices.

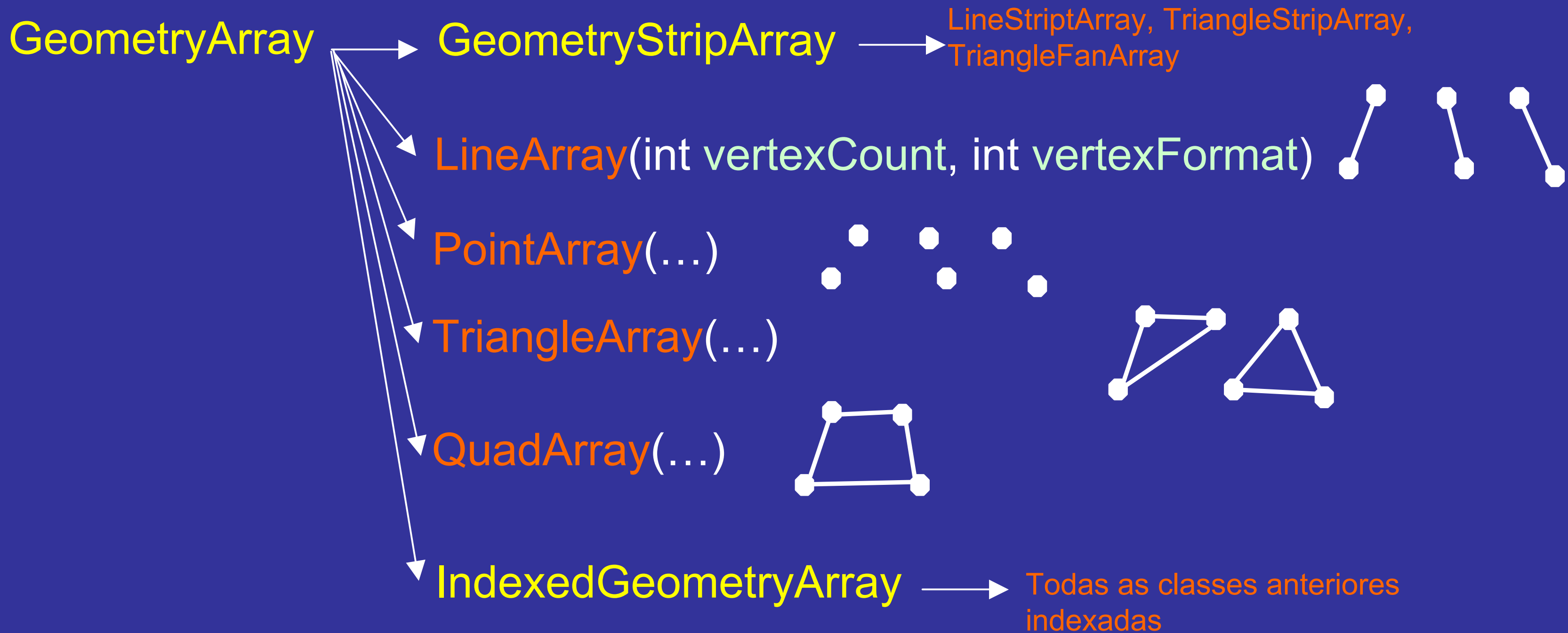
Métodos da classe Shape3D:

```
Void addGeometry(Geometry geometry)
Void SetAppearance(Appearance appearance)
```

Geometry: superclasse de um conjunto de classes para definir a geometria de várias formas

Geometria

Geometry



Cada vértice pode especificar até 4 parâmetros (indicado por vertexFormat):

- Coordenadas
- Cor
- Normais à superfície (necessário para calculo de iluminação)
- Coordenadas de textura

Geometria

- A principal diferença entre as classes de especificação de geometria está no número de vértices guardados.
- Nos *Arrays* básicos o mesmo vértice pode aparecer mais do que uma vez. Maior consumo de memória mas obtém-se *rendering* mais eficiente.
- Com *Arrays* indexados cada vértice aparece apenas uma vez. Um nível mais de redireccionamento => maior complexidade no *rendering.*

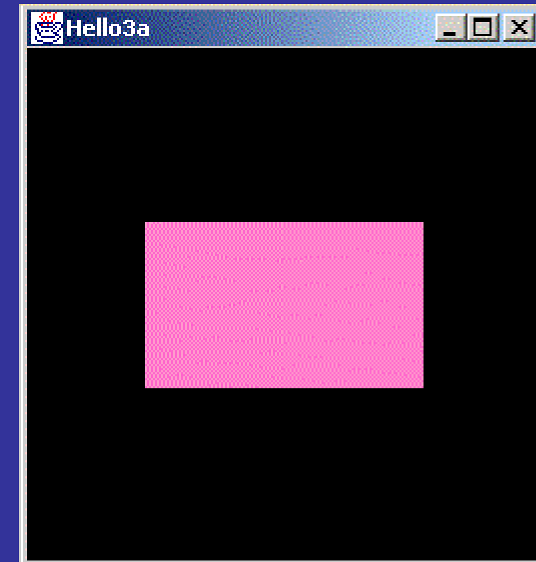
Recomendação: usar StripArrays sempre que possível

(ver exemplos: AxisApp.java, YoyoApp.java e Axis.java)

com.sun.j3d.utils.geometry.*

Classes para representar primitivas geométricas:

- Box
- Cone
- Cylinder
- Sphere
- ColorCube



Exemplo:

```
Appearance app = new Appearance();
app.setColoringAttributes(new
      ColoringAttributes(1.f,0.5f,0.8f,ColoringAttributes.FASTEST));
objRoot.addChild(new Box(0.5f,0.3f,0.2f, app));
```

// ver método getShape() e setAppearance() de Shape3D para atribuír cores diferentes às faces

(ver exemplo: ConeYoyoApp.java e Axis.java)

Interacção e Animação

Interacção: a acção ocorre em resposta a estimulos provocados pelo utilizador

Animação: a acção ocorre pela passagem do tempo

Behavior class: classe abstracta que fornece os mecanismos necessários para responder a eventos possibilitando a alteração do grafo em *run time*

Interacção e Animação

Exemplos de estimulos: teclado, rato, colisão de objectos, tempo, combinação de vários eventos,...

Exemplos de Acções: adicionar/remover objectos da cena, mudar atributos de objectos, lançar *Threads*,...

As subclasses de Behavior têm de definir:

- Método **`initialize()`** - define o evento que activa esse behavior
- Método **`processStimulus(Enumeration c)`** – método invocado pelo sistema quando ocorre o evento correspondente. A última instrução deve definir novamente a nova condição de activação.
- *Scheduling Region* : especifica a região do espaço onde o *behavior* é válido. Restringe a região onde são verificadas as condições de activação. Melhora o desempenho do sistema.

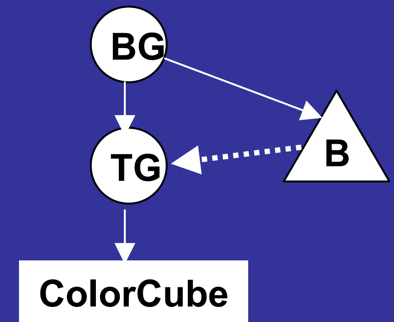
Interacção e Animação

Tipos de *scheduling region*:

- BoundingSphere
- BoundingBox
- BoundingPolytope
  - permite definir regiões a partir de equações de planos, pela reunião das regiões definidas por um conjunto de objectos, etc

Exemplo: teclado

```
public class SimpleBehavior extends Behavior{

        private TransformGroup targetTG;
        private Transform3D rotation = new Transform3D();
        private double angle = 0.0;

        // create SimpleBehavior
        SimpleBehavior(TransformGroup targetTG){
            this.targetTG = targetTG;
            //targetTG.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        }

        // initialize the Behavior  -  set initial wakeup condition
        // called when behavior beacomes live
        public void initialize(){
            this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
        }

        // called by Java 3D when appropriate stimulus occures
        public void processStimulus(Enumeration criteria){
            // decode event,   do what is necessary
            angle += 0.1;
            rotation.rotY(angle);
            targetTG.setTransform(rotation);
            this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
        }
} // end of class SimpleBehavior
```



(ver exemplo: SimpleBehavior.java)

Descodificação

```
public void processStimulus (Enumeration criteria) {
    WakeupCriterion wakeup;
    AWTEvent[] event;
    int id, i;

    while (criteria.hasMoreElements()) {
      wakeup = (WakeupCriterion) criteria.nextElement();
      if (wakeup instanceof WakeupOnAWTEvent) {
          event = ((WakeupOnAWTEvent)wakeup).getAWTEvent();

          for(i=0; i < event.length; i++)
          {
                id = event[i].getID();
                if(id==KeyEvent.KEY_PRESSED) {
                    if (((KeyEvent) event[i]).GetKeyCode() == KeyEvent.VK_S){
                        // processamento do evento
                        targetTG.getTransform(transl);
                        transl.mul(offsetRight);
                        targetTG.setTransform(transl);
                    }
                }

          }
      }
    }
    this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
}
```

Interacção e Animação

Interpolator class: classe abstracta que extende a classe Behavior e fornece vários métodos usados por subclasses de interpolação

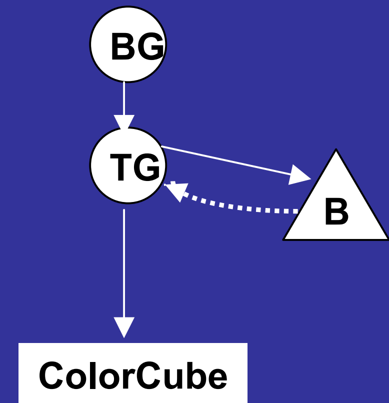
Interpolator

- ColorInterpolator
- PathInterpolator

- PositionInterpolator
- RotationInterpolator
- ScaleInterpolator
- SwitchValueInterpolator
- TransparencyInterpolator

Exemplo: HelloJava3Dc

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

         // Create the transform group node and initialize it to the
         // identity. Add it to the root of the subgraph.
         TransformGroup objSpin = new TransformGroup();
         objSpin.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
         objRoot.addChild(objSpin);

         // Create a simple shape leaf node, add it to the scene graph.
         // ColorCube is a Convenience Utility class
         objSpin.addChild(new ColorCube(0.4));

         // Create a new Behavior object that will perform the desired
         // operation on the specified transform object and add it into
         // the scene graph.
         Alpha rotationAlpha = new Alpha(-1, 4000);

         RotationInterpolator rotator =
             new RotationInterpolator(rotationAlpha, objSpin);

         // a bounding sphere specifies a region a behavior is active
         // create a sphere centered at the origin with radius of 100
         BoundingSphere bounds = new BoundingSphere();
         rotator.setSchedulingBounds(bounds);
         objSpin.addChild(rotator);

         return objRoot;
} // end of CreateSceneGraph method
```



Documentação

Tutorial

1. Introdução
2. Geometria
3. Criação fácil de conteúdos
4. Interacção
5. Animação
6. Iluminação
7. Texturas