Java 3D™ API

Profa. Isabel Harb Manssour

(http://www.inf.pucrs.br/~manssour/Java3D)

XVI SIBGRAPI – 12-15 de Outubro de 2003

2. Criando Universos Virtuais

Criando Universos Virtuais

- **Grafo de Cena**
- Classes Principais
- Geometrias
- Texto e *Background*

Criando Universos Virtuais

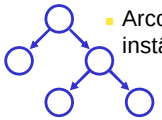
Grafo de Cena

- Aplicação Java 3D
  - Projetada a partir de um grafo de cena
- Grafo de cena
  - Possibilita a criação de um universo virtual
  - Simples de construir
    - Não é necessário ter experiência em programação 3D
  - Contém objetos gráficos
    - Geometria, luz, som, entre outros

[Bicho 2002]

Criando Universos Virtuais
Grafo de Cena

- Universo virtual
  - Composto por um ou mais grafos de cena
- Grafo de cena
  - Consiste em objetos Java 3D (*nodes*) organizados em uma estrutura do tipo árvore, composta de:
    - Nodos (ou vértices): instâncias das classes Java 3D
    - Arcos (ou arestas): relacionamento entre as instâncias

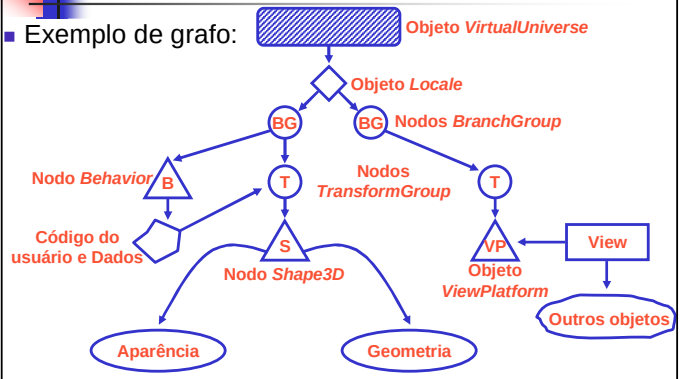


Criando Universos Virtuais
Grafo de Cena

- Relacionamentos
  - Pai-Filho
    - "Nodo do tipo grupo"
      - Pode ter um ou mais filhos, mas apenas um pai
      - Representado por um círculo 
    - "Nodo do tipo folha"
      - Não pode ter filhos e tem apenas um pai
      - Representado por um triângulo 
  - Referência
    - Associa um objeto com o grafo de cena

Criando Universos Virtuais
Grafo de Cena

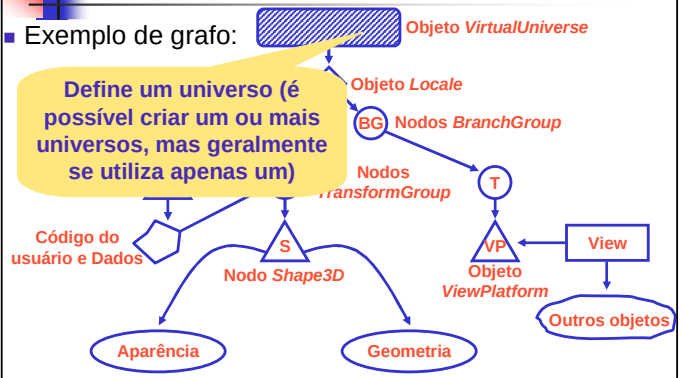
- Exemplo de grafo:



[Sowizral 1998]

Criando Universos Virtuais
Grafo de Cena

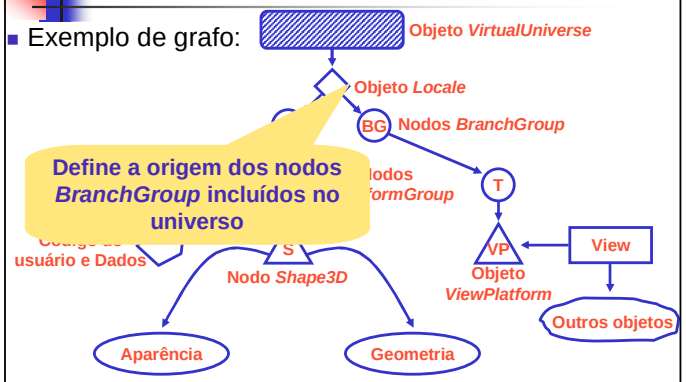
- Exemplo de grafo:



Criando Universos Virtuais

Grafo de Cena

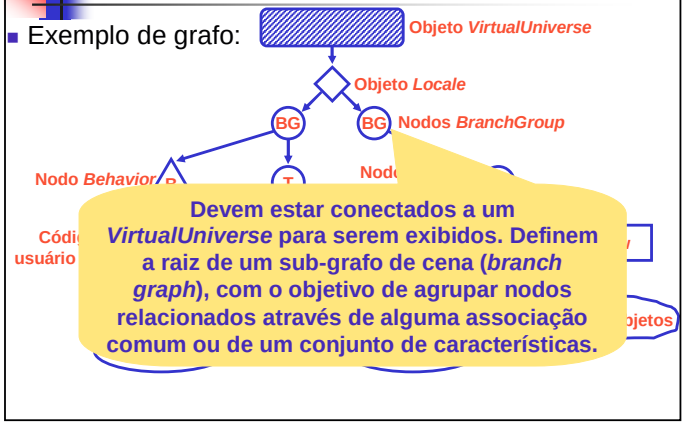
- Exemplo de grafo:



Criando Universos Virtuais

Grafo de Cena

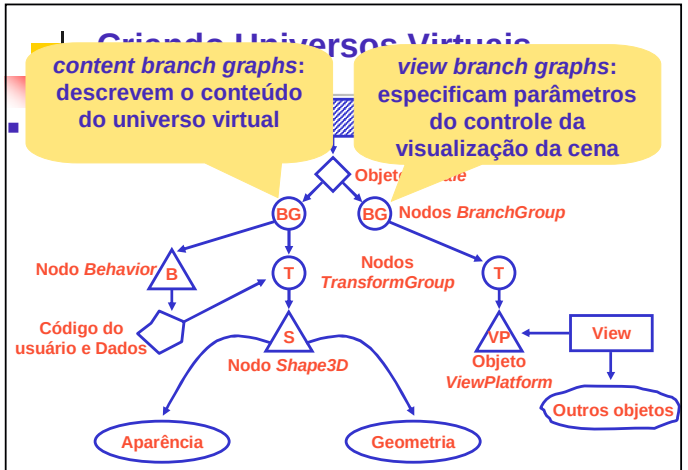
- Exemplo de grafo:



Criando Universos Virtuais

Grafo de Cena

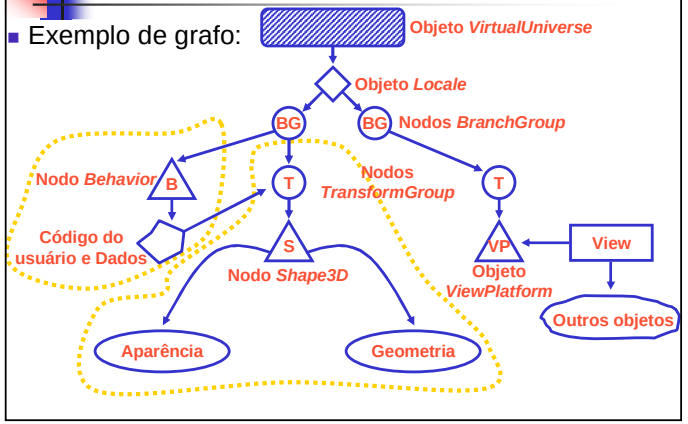
- Exemplo de grafo:



Criando Universos Virtuais

Grafo de Cena

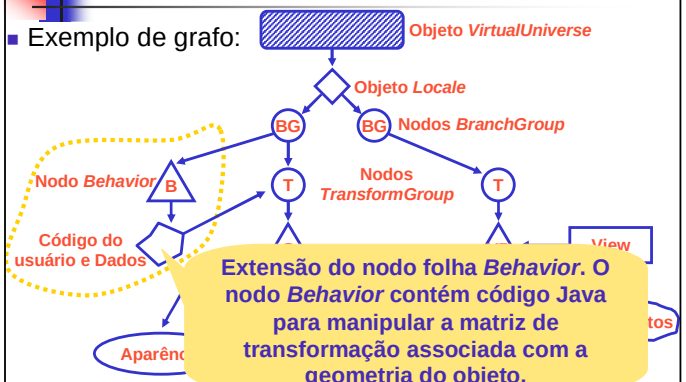
- Exemplo de grafo:



Criando Universos Virtuais

Grafo de Cena

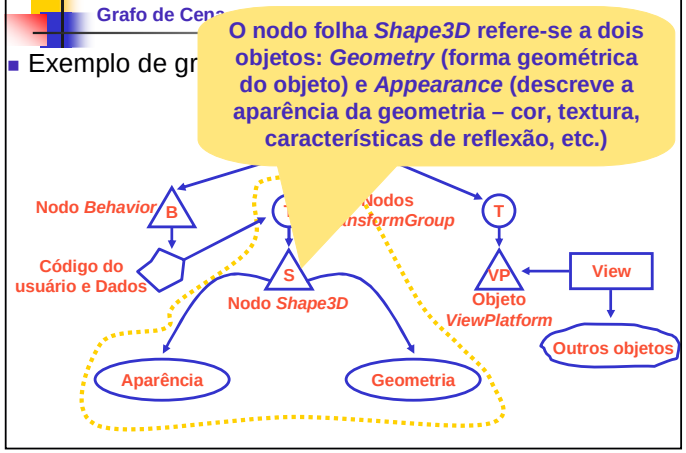
- Exemplo de grafo:



Criando Universos Virtuais

Grafo de Cena

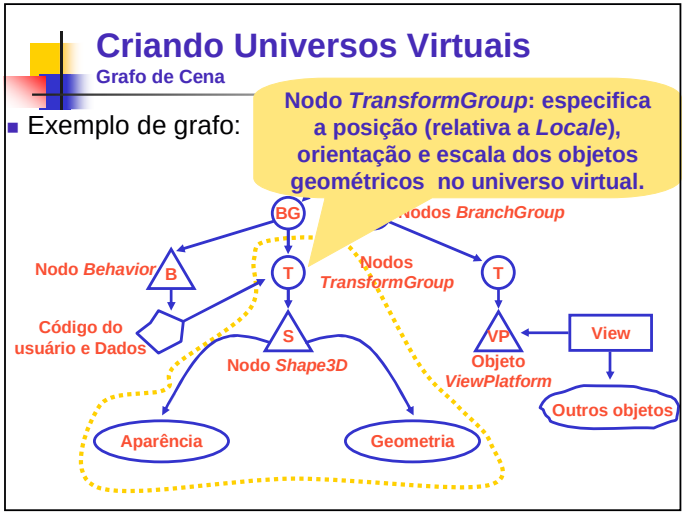
- Exemplo de gr



Criando Universos Virtuais

Grafo de Cena

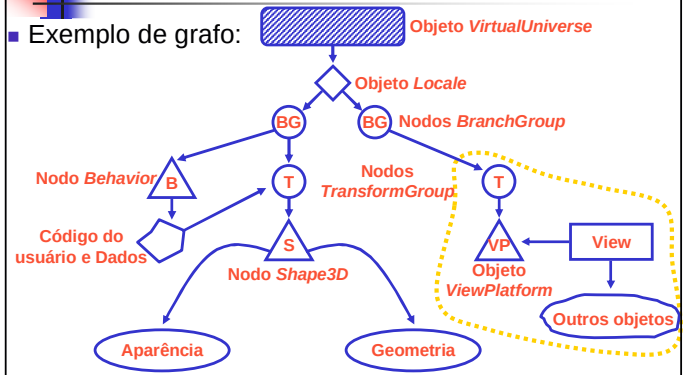
- Exemplo de grafo:



Criando Universos Virtuais

Grafo de Cena

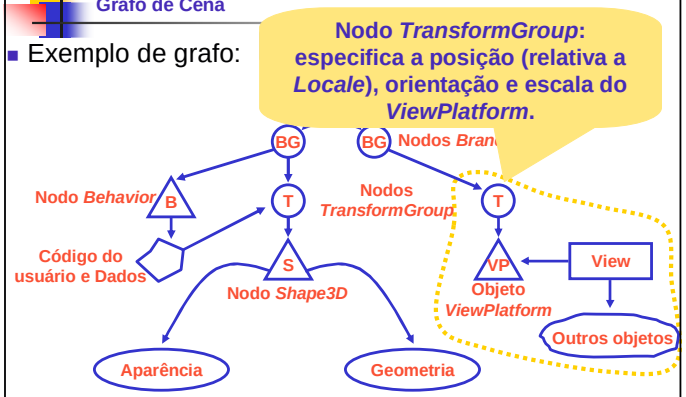
- Exemplo de grafo:



Criando Universos Virtuais

Grafo de Cena

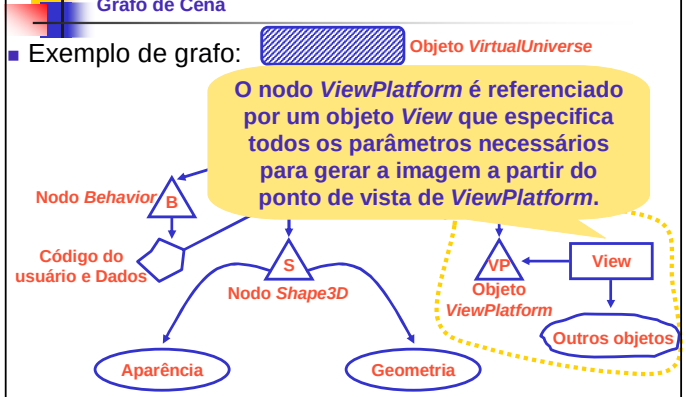
- Exemplo de grafo:



Criando Universos Virtuais

Grafo de Cena

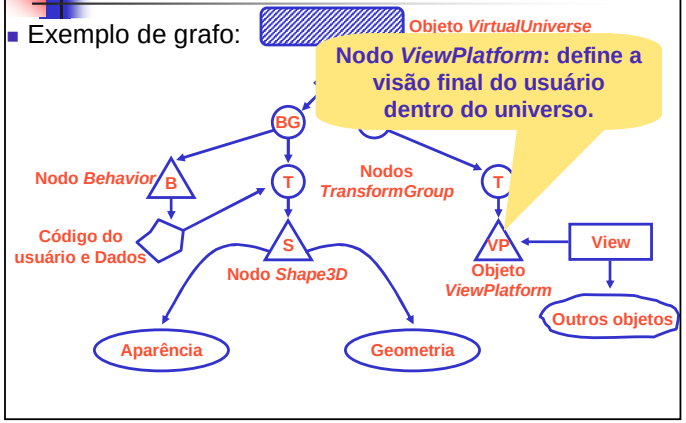
- Exemplo de grafo:



Criando Universos Virtuais

Grafo de Cena

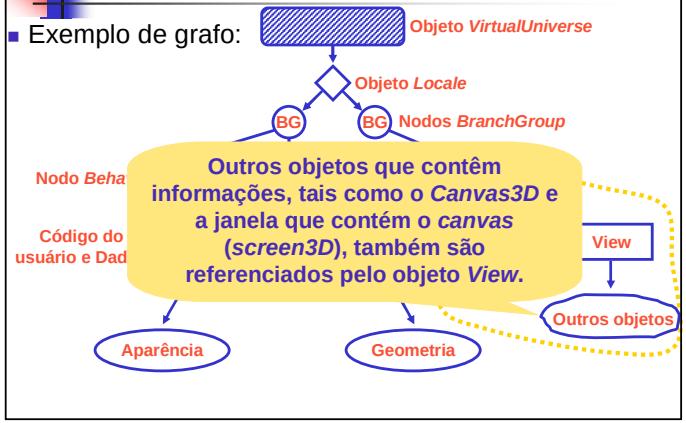
- Exemplo de grafo:



Criando Universos Virtuais

Grafo de Cena

- Exemplo de grafo:



Criando Universos Virtuais
Grafo de Cena

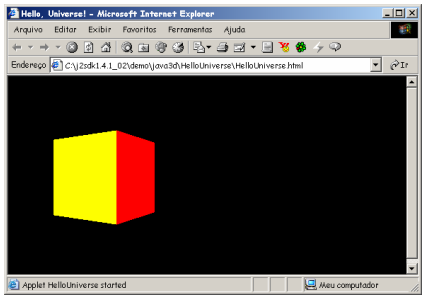
- Exemplo:
  - Aplicação para o projeto de um escritório
    - Grupo de elementos da arquitetura
      - Chão, paredes e portas
      - Estáticos
    - Grupo dos móveis
      - Mesa, cadeira, luminária, etc
      - Podem ser trocados de lugar

[Barrilleaux 2001]

Criando Universos Virtuais
Grafo de Cena

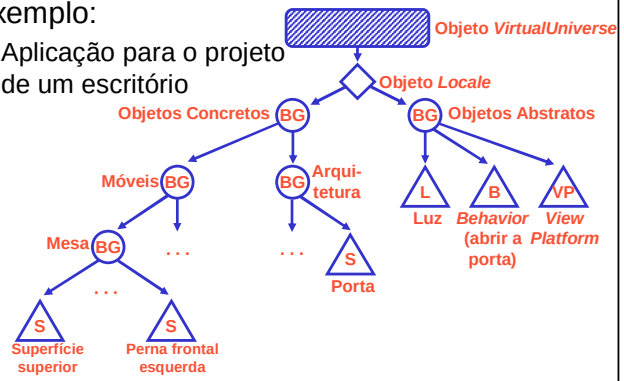
- Primeiro programa Java 3D [Sowizral 1998, SUN 2003]

HelloUniverse.java



Criando Universos Virtuais
Grafo de Cena

- Exemplo:
  - Aplicação para o projeto de um escritório



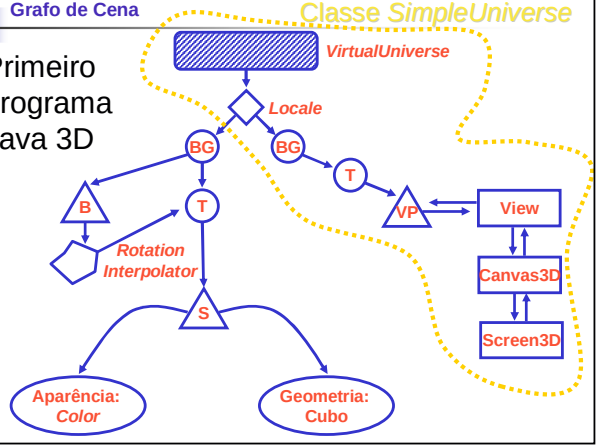
Criando Universos Virtuais
Grafo de Cena

- Primeiro programa Java 3D
  - Passos para a criação do grafo de cena [Sun 2003]
    1. Criar um objeto *GraphicsConfiguration*
    2. Criar um objeto *Canvas3D*
    3. Construir e compilar pelo menos um sub-grafo de conteúdo
    4. Criar um objeto *SimpleUniverse*, que referencia o objeto *Canvas3D* criado e automaticamente cria os objetos *VirtualUniverse* e *Locale*, e constrói o sub-grafo de visualização
    5. Inserir o sub-grafo de conteúdo no universo virtual

Criando Universos Virtuais

Grafo de Cena

- Primeiro programa Java 3D



Criando Universos Virtuais

Grafo de Cena

```
 :
 * Neither the name of Sun Microsystems, Inc. or the names of
 * contributors may be used to endorse or promote products derived
 * from this software without specific prior written permission.
 */

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
import java.awt.GraphicsConfiguration;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.ColorCube;
import com.sun.j3d.utils.universe.*;
import javax.media.j3d.*;
import javax.vecmath.*;
```

Criando Universos Virtuais

Grafo de Cena

- Primeiro programa Java 3D

```
/*
 *   @(#)HelloUniverse.java 1.55 02/10/21 13:43:36
 *
 * Copyright (c) 1996-2002 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * - Redistributions of source code must retain the above copyright
 *   notice, this list of conditions and the following disclaimer.
 *
 * - Redistribution in binary form must reproduce the above copyright
 *   notice, this list of conditions and the following disclaimer in
 *   the documentation and/or other materials provided with the
 *   distribution.
 *
 :
```


Criando Universos Virtuais

Grafo de Cena

```
public class HelloUniverse extends Applet
{

    private SimpleUniverse u = null;

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

       // Create the TransformGroup node and initialize it to the
       // identity. Enable the TRANSFORM_WRITE capability so that
       // our behavior code can modify it at run time. Add it to
       // the root of the subgraph.
       TransformGroup objTrans = new TransformGroup();
       objTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
       objRoot.addChild(objTrans);

       // Create a simple Shape3D node; add it to the scene graph.
       objTrans.addChild(new ColorCube(0.4));
```

Criando Universos Virtuais

Grafo de Cena

```
   // Create a new Behavior object that will perform the
   // desired operation on the specified transform and add
   // it into the scene graph.
   Transform3D yAxis = new Transform3D();
   Alpha rotationAlpha = new Alpha(-1, 4000);

   RotationInterpolator rotator =
          new RotationInterpolator(rotationAlpha, objTrans, yAxis,
                                   0.0f, (float) Math.PI*2.0f);
   BoundingSphere bounds =
                new BoundingSphere(new Point3d(0.0,0.0,0.0), 100.0);
   rotator.setSchedulingBounds(bounds);
   objRoot.addChild(rotator);

    // Have Java 3D perform optimizations on this scene graph.
    objRoot.compile();

   return objRoot;
} // end of createSceneGraph()
```

Criando Universos Virtuais

Grafo de Cena

```
    public HelloUniverse()
    {
    }
    public void init()
    {
      setLayout(new BorderLayout());
      GraphicsConfiguration config =
                    SimpleUniverse.getPreferredConfiguration();
      Canvas3D c = new Canvas3D(config);
      add("Center", c);

      // Create a simple scene and attach it to the virtual universe
      BranchGroup scene = createSceneGraph();
      u = new SimpleUniverse(c);

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

      u.addBranchGraph(scene);
   } // end of init()
```


Criando Universos Virtuais

Grafo de Cena

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

    //
    // The following allows HelloUniverse to be run as an application
    // as well as an applet
    //
    public static void main(String[] args)
      {
       new MainFrame(new HelloUniverse(), 256, 256);
    }

} // end of class HelloUniverse
```

Exercício 1

Objetivo: familiarizar-se com o ambiente, analisar e executar um exemplo de programa Java 3D

Criando Universos Virtuais

- Grafo de Cena
- **Classes Principais**
- Geometrias
- Texto e *Background*

Criando Universos Virtuais
Classes Principais

- Classes importantes para a criação de grafos de cena que apareceram no exemplo
  - *SimpleUniverse*
  - *GraphicsConfiguration*
  - *Canvas3D*
  - *BranchGroup*
  - *TransformGroup*
  - *Transform3D*
  - *Alpha*
  - *RotationInterpolation*
  - *BoundingSphere*
  - *MainFrame*

Criando Universos Virtuais
Classes Principais

- *SimpleUniverse*
  - Configura um ambiente mínimo para executar um programa Java 3D
  - Fornece as funcionalidades necessárias para a maioria das aplicações
  - Sub-grafo de visualização

Criando Universos Virtuais
Classes Principais

- *SimpleUniverse*
  - Cria todos os objetos necessários para um sub-grafo de visualização (valores *default*)
    - *Locale*
      - Define uma posição no *VirtualUniverse* e serve como um *container* para uma coleção de grafos de cena
    - *ViewingPlatform*
      - Usada para especificar a *view*; contém um nodo *MultiTransformGroup* que permite a combinação de uma série de transformações

Criando Universos Virtuais
Classes Principais

- *SimpleUniverse*
  - Cria todos os objetos necessários para um sub-grafo de visualização (valores *default*)
    - *Viewer*
      - Contém todas informações que descrevem uma "presença" física e virtual no universo Java 3D
      - Física: *Canvas3D*, *PhysicalEnvironment* (características do hardware), *PhysicalBody* (características físicas e preferências pessoais da pessoa que está visualizando a cena)
      - Virtual: *View* e *ViewerAvatar* (representa a pessoa que está visualizando o universo)

Criando Universos Virtuais
Classes Principais

- *GraphicsConfiguration*
  - Faz parte do pacote ***awt***
  - Descreve as características do dispositivo gráfico (impressora ou monitor)
  - A estrutura varia de plataforma para plataforma
  - No *Microsoft Windows*, *GraphicsConfigurations* representa os *PixelFormats* disponíveis de acordo com resolução do dispositivo

Criando Universos Virtuais
Classes Principais

- *GraphicsConfiguration*
  - Alguns métodos
    - *public abstract GraphicsDevice getDevice()*
      - Retorna o *GraphicsDevice* associado com o *GraphicsConfiguration*
    - *public abstract Rectangle getBounds()*
      - Retorna os limites do *GraphicsConfiguration* nas coordenadas do dispositivo
    - *public abstract ColorModel getColorModel()*
      - Retorna o *ColorModel* associado com o *GraphicsConfiguration*

Criando Universos Virtuais
Classes Principais

- *Canvas3D*
  - Fornece um *canvas* para fazer o *rendering* 3D
  - Extensão da classe *Canvas* da *awt*
    - Inclui informações 3D, tais como tamanho e localização do *canvas*
  - Contém uma referência para um objeto *Screen3D*
    - Define o tamanho do *pixel*
    - Pode converter o tamanho do *Canvas3D* (em *pixels*) para metros

Criando Universos Virtuais
Classes Principais

- *Canvas3D*
  - Alguns métodos
    - *Canvas3D(java.awt.GraphicsConfiguration graphicsConfiguration)*
    - *java.awt.Dimension getSize()*
    - *GraphicsContext3D getGraphicsContext3D()*
      - Retorna o contexto gráfico 3D associado com o *Canvas*
    - *java.awt.Point getLocation()*
      - Retorna a localização do componente
    - *double getPhysicalHeight()*
      - Retorna a altura do *canvas* em metros
    - *double getPhysicalWidth ()*
      - Retorna a largura do *canvas* em metros
    - *Screen3D getScreen3D ()*
      - Retorna o objeto *Screen3D* associado ao *Canvas3D*
    - *View getView()*
      - Retorna a *View* deste *Canvas3D*

Criando Universos Virtuais
Classes Principais

- *BranchGroup*
  - Serve como ponteiro para a raiz de um sub-grafo de cena
  - Únicos objetos que podem ser inseridos em *Locale*
  - Um grafo de cena (ou sub-grafo) que tem um *BranchGroup* como raiz pode ser considerado como uma *compile unit*

Criando Universos Virtuais
Classes Principais

- *BranchGroup*
  - Pode ser:
    - Compilado através do método *compile* (incluindo os seus descendentes)
    - Inserido em um universo virtual, associando-o com *Locale*
    - Desassociado em tempo de execução, configurando as opções apropriadas e se fizer parte de outro *BranchGroup*

Criando Universos Virtuais
Classes Principais

- *BranchGroup*
  - Alguns métodos
    - *public BranchGroup()*
    - *void compile()*
      - Compila o *BranchGroup* associado com o objeto e cria e armazena um grafo de cena compilado
    - *void detach()*
      - Desassocia o *BranchGroup* do seu pai
    - *SceneGraphPath[] pickAll (PickShape pickShape)*
      - Retorna um vetor com todos os itens do *BranchGroup* que são *pickable* e que se interseccionam com *PickShape*
    - *SceneGraphPath[] pickClosest (PickShape pickShape)*
      - Retorna um *SceneGraphPath* que referencia o item *pickable* mais próximo da origem de *PickShape*

Criando Universos Virtuais
Classes Principais

- *TransformGroup*
  - Especifica uma transformação, através de um objeto *Transform3D*, que pode transladar, rotacionar ou trocar a escala de todos os seus filhos
  - Usado como nodo pai de *ViewPlatform*
  - Os efeitos das transformações num grafo de cena são cumulativos

Criando Universos Virtuais
Classes Principais

- *TransformGroup*
  - Alguns métodos
    - *public TransformGroup ()*
    - *public TransformGroup (Transform3D t)*
    - *void setTransform (Transform3D t)*
    - *void getTransform (Transform3D t)*
  - Constantes
    - ALLOW_TRANSFORM_READ
      - Especifica que o nodo permite acesso às informações de transformação dos objetos
    - ALLOW_TRANSFORM_WRITE
      - Especifica que o nodo permite escrever as informações de transformação dos objetos

Criando Universos Virtuais
Classes Principais

- *Transform3D*
  - Objetos representam uma matriz 4x4 (*float*)
  - Usados para aplicar transformações geométricas
  - Alguns métodos
    - *rotX(double ang)*, *rotY(double ang)* e *rotZ(double ang)*
    - *setRotation(AxisAngle4d a1) – AxisAngle4d(ang,x,y,z)*
    - *setScale(double scale)*
    - *setScale(Vector3d scale)*
    - *setTranslation(Vector3d trans)* – ou *Vector3f*
    - *setIdentity()*
    - *mul(Transform3D t1)*

Criando Universos Virtuais
Classes Principais

- *Alpha*
  - Fornece métodos para converter um valor de tempo em um valor *alpha* (entre 0 e 1)
    - f(t) = [0,1]
  - Útil para fornecer valores *alpha* para interpolações (*Interpolator behaviors*)
  - Constantes
    - INCREASING_ENABLE
    - DECREASING_ENABLE

Criando Universos Virtuais
Classes Principais

- *Alpha*
  - Alguns construtores
    - *Alpha ()*
      - Cria o objeto com os parâmetros *default*
    - *Alpha (int loopCount, long increasingAlphaDuration)*
      - Especifica os parâmetros *loopCount* e *increasingAlphaDuration*, e usa os valores *default* para os outros parâmetros
      - *loopCount*: número de vezes para executar o objeto (-1 = *loop*)
      - *increasingAlphaDuration*: período de tempo durante o qual *Alpha* vai de zero para um

Criando Universos Virtuais
Classes Principais

- *RotationInterpolation*
  - Esta classe define um comportamento (*interpolator behavior*)
  - Modifica o componente de rotação do *TransformGroup* através de uma interpolação linear entre um par de ângulos especificados (usando o valor gerado por um objeto *Alpha*)
  - O ângulo interpolado é usado para gerar uma transformação de rotação sobre o eixo Y

Criando Universos Virtuais
Classes Principais

- *RotationInterpolation*
  - Construtores
    - *RotationInterpolator (Alpha a, TransformGroup target)*
      - Cria um *rotation interpolator* trivial com o *TransformGroup* especificado, com valores *default* para um eixo de transformação (identidade), ângulo mínimo (0.0) e ângulo máximo (2*PI)
    - *RotationInterpolator (Alpha a, TransformGroup target, Transform3D axisOfTransform, float minimumAngle, float maximumAngle)*
      - Cria um novo *rotation interpolator* que varia o componente de rotação do nodo de transformação

Criando Universos Virtuais
Classes Principais

- *BoundingSphere*
  - Define uma região limitada por uma esfera, que é definida por um ponto central e um raio
  - Alguns construtores
    - *BoundingSphere()*
      - Cria e inicializa uma *BoundingSphere* com raio=1 em (0,0,0)
    - *BoundingSphere(Point3d center, double radius)*
      - Cria e inicializa uma *BoundingSphere* a partir do centro e raio fornecidos

Criando Universos Virtuais
Classes Principais

- *MainFrame*
  - Subclasse de java.awt.Frame
    └ com.sun.j3d.utils.applet.MainFrame
  - Implementa várias interfaces (java.applet.AppletContext)
  - Permite que o programa seja executado como uma *applet* ou como uma aplicação
  - Construtores
    - MainFrame(java.applet.Applet applet, int width, int height)
    - MainFrame(java.applet.Applet applet, java.lang.String[] args)
    - MainFrame(java.applet.Applet applet, java.lang.String[] args, int width, int height)

Criando Universos Virtuais
Classes Principais

- Existem várias outras classes e métodos

- Consultar a API!!!

  c:\Arquivos de programas\j2sdk1.4.1_05\docs\j3dapi

Criando Universos Virtuais

- Grafo de Cena
- Classes Principais
- **Geometrias**
- Texto e *Background*

Criando Universos Virtuais
Geometrias

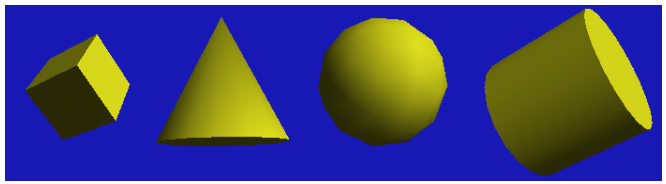
- No exemplo foi utilizado o *ColorCube*
- Entretanto, muitas outras formas podem ser utilizadas
- Em Computação Gráfica
  - Modelos são utilizados para representar entidades físicas ou abstratas e fenômenos no computador
  - Modelagem consiste em todo o processo de descrever um modelo, objeto ou cena, de forma que se possa desenhá-lo

Criando Universos Virtuais
Geometrias

- Um modelo em Java 3D é composto por
  - Uma **geometria**
    - Define a "forma geométrica" do modelo
    - Formas de definir uma geometria
      - Primitivas gráficas
      - Lista de vértices e lista de arestas
      - *Loader*
  - Uma **aparência**
    - Define a cor, transparência, textura, etc, da geometria

Criando Universos Virtuais
Geometrias

- Formas de definir a **geometria** de um modelo
  - Primitivas gráficas
    - *Box*, *Cone*, *Sphere* e *Cylinder*



Criando Universos Virtuais
Geometrias

- Primitivas gráficas
  - *Box*
    - *Box ()*
      - *Box default* com todas as dimensões em 1.0
    - *Box (float xdim, float ydim, float zdim, Appearance ap)*
      - Passa as dimensões e a aparência do Box

Criando Universos Virtuais
Geometrias

- Primitivas gráficas
  - *Cone*
    - *Cone ()*
      - *Cone default* com raio = 1.0 e altura = 2.0
    - *Cone (float radius, float height)*
      - Passa o raio e a altura do cone que está sendo criado
    - *Cone (float radius, float height, Appearance ap)*
      - Passa o raio, a altura e a aparência do cone

Criando Universos Virtuais
Geometrias

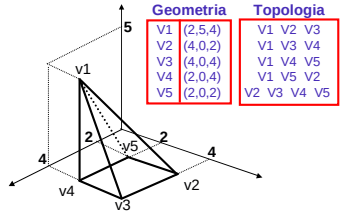
- Primitivas gráficas
  - *Sphere*
    - *Sphere ()*
      - Esfera *default* com raio = 1.0
    - *Sphere(float radius)*
      - Passa o raio da esfera que está sendo criada
    - *Sphere(float radius, Appearance ap)*
      - Passa o raio e a aparência da esfera
    - *Sphere(float radius, int primflags, int divisions, Appearance ap)*
      - Passa o raio, o número de divisões e a aparência da esfera

Criando Universos Virtuais
Geometrias

- Primitivas gráficas
  - *Cylinder*
    - *Cylinder ()*
      - Cilindro *default* com raio = 1.0 e altura = 2.0
    - *Cylinder (float radius, float height)*
      - Passa o raio e a altura do cilindro que está sendo criado
    - *Cylinder (float radius, float height, Appearance ap)*
      - Passa o raio, a altura e a aparência do cilindro
    - *Cylinder (float radius, float height, int primflags, int xdivision, int ydivision, Appearance ap)*
      - Passa o raio, a altura, a resolução (x e y) e a aparência do cilindro

Criando Universos Virtuais
Geometrias

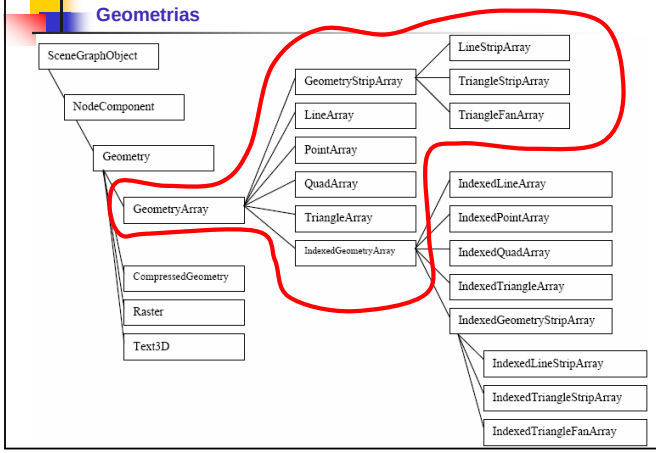
- Formas de definir a **geometria** de um modelo
  - Lista de vértices e lista de arestas
  - Malha de polígonos representa uma superfície composta por faces planas
    - Triângulos
    - Quadrados



Criando Universos Virtuais
Geometrias

- Instância da classe *Shape3D*
  - Possui dois objetos
    - *Geometry* (define a forma geométrica do modelo)
    - *Appearance* (define a cor, transparência, textura, etc, da geometria)
- *Geometry*
  - Classe abstrata
  - Subclasses usadas para especificar pontos, linhas, polígonos preenchidos e texto

Criando Universos Virtuais
Geometrias

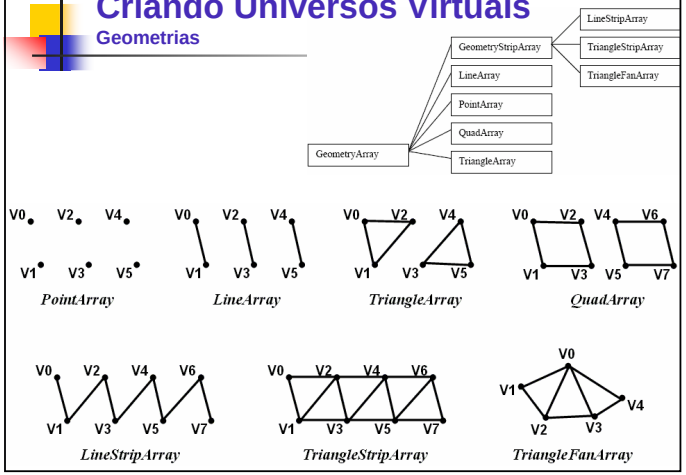


Criando Universos Virtuais
Geometrias

- Formas de definir a geometria de um modelo
  - Importar dados geométricos criados por outras aplicações
    - Importar arquivo de formato padrão através de um *loader*
    - VRML (.wrl), Wavefront (.obj), AutoCAD (.dfx), 3D Studio (.3ds), etc
    - Implementações da interface definida no pacote com.sun.j3d.loaders

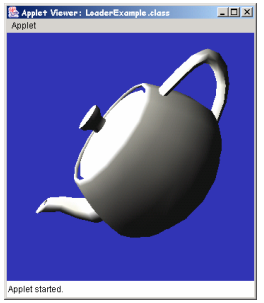
```
coordIndex [              # lista de Faces        coord Coordinate {
    4, 3, 2, 1  -1, # Base (vértices                point [          # lista de Vértices
                      4, 3,2 e 1)                      0  10  0,   # vértice 0
    0, 1, 2, -1,  # Frontal                          -5   0  5,   # vértice 1
    0, 2, 3, -1,  # Direita                           5   0  5,   # vértice 2
    0, 3, 4, -1,  # Traseira                          5   0 -5,   # vértice 3
    0, 4, 1, -1   # Esquerda                         -5   0 -5,   # vértice 4
]                                                  ]
```

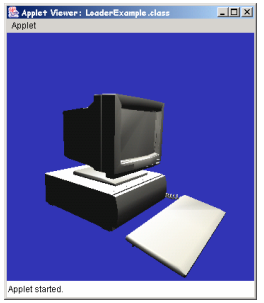
Criando Universos Virtuais
Geometrias



Criando Universos Virtuais
Geometrias

- Visualização de arquivos *obj* utilizando um *loader*





Criando Universos Virtuais
Geometrias

- **Aparência** de uma geometria
  - Nodo *Appearance*
    - Permite especificar as suas propriedades
      - Textura, transparência, cor, estilo de linha, material, entre outras
    - Faz referência a vários outros objetos (ex.: *Material*)
    - Alguns métodos
      - *void setColoringAttributes (ColoringAttributes coloringAttributes)*
      - *void setLineAttributes (LineAttributes lineAttributes)*
      - *void setTexture (Texture texture)*
      - *void setMaterial (Material material)*
      - *Material getMaterial ()*

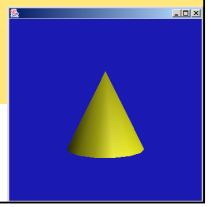
Criando Universos Virtuais
Geometrias

- Aparência de uma geometria
  - Nodo *Material*
    - Define a aparência de um objeto considerando que existem fontes de luz
    - Propriedades que são especificadas
      - Cor ambiente refletida da superfície do material – *default* (0.2, 0.2, 0.2)
      - Cor difusa: cor do material quando iluminado – *default* (1.0, 1.0, 1.0)
      - Cor especular do material – *default* (1.0, 1.0, 1.0)
      - Cor emissiva: cor da luz que o material emite – *default* (0.0, 0.0, 0.0)
      - *Shininess*: concentração do brilho do material que varia entre 1 e 128, sendo 128 o brilho máximo – *default* 64

Criando Universos Virtuais
Geometrias

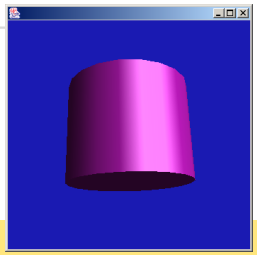
- Exemplo 1:

```
Appearance app = new Appearance();

//Parâmetros:ambColor,emissiveColor,diffuseColor,specColor,shininess
Material material = new Material(new Color3f(0.8f,0.8f,0.1f),
                                 new Color3f(0.0f,0.0f,0.0f),
                                 new Color3f(0.8f,0.8f,0.1f),
                                 new Color3f(1.0f,1.0f,1.0f), 100.0f);

app.setMaterial(material);

Cone cone = new Cone(0.4f, 0.8f);
cone.setAppearance(app);
```



Criando Universos Virtuais
Geometrias

- Exemplo 2:



```
Appearance app = new Appearance();

//Parâmetros:ambColor,emissiveColor,diffuseColor,specColor,shininess
Material material = new Material(new Color3f(0.7f,0.1f,0.7f),
                                 new Color3f(0.0f,0.0f,0.0f),
                                 new Color3f(0.7f,0.1f,0.7f),
                                 new Color3f(1.0f,1.0f,1.0f), 60.0f);

app.setMaterial(material);

Cylinder cilindro = new Cylinder(0.5f, 0.8f, 1, 20, 10, app);
```

Exercício 2

Objetivo: criar e alterar a aparência de diferentes objetos usando primitivas, instâncias de *Geometry* e *loader*

Criando Universos Virtuais

- Grafo de Cena
- Classes Principais
- Geometrias
- **Texto e *Background***

Criando Universos Virtuais

Texto e *Background*

- Existem duas maneiras de adicionar texto em uma cena Java 3D
  - Classe *Text2D*
    - Polígonos retangulares com o texto aplicado como textura
  - Classe *Text3D*
    - Objetos geométricos 3D

Criando Universos Virtuais

Texto e *Background*

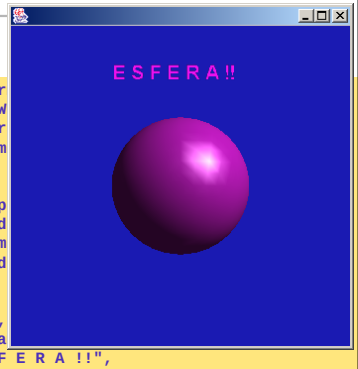
- Exemplo: *Text2D*

```
// Cria nodo TransformGroup e permite que ele possa ser alterado em
// tempo de execução (TRANSFORM_WRITE). Adiciona-o no sub-grafo.
TransformGroup textTrans = new TransformGroup();
textTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
objRaiz.addChild(textTrans);

// Cria um novo objeto que irá aplicar as transformações
// geométricas sobre texto e o adiciona no sub-grafo.
Transform3D trans = new Transform3D();
trans.setTranslation(new Vector3d(-0.4,0.6,0.0));
textTrans.setTransform(trans);

// Text2D (java.lang.String text, Color3f color,
//        java.lang.String fontName, int fontSize, int fontStyle)
Text2D text2D = new Text2D("E S F E R A !!",
                           new Color3f(0.9f, 0.1f, 0.9f),
                           "Helvetica", 30, Font.BOLD);
textTrans.addChild(text2D);
```

Criando Universos Virtuais

Texto e *Background*

- Exemplo: *Text2D*




```
// Cria nodo TransformGroup e per
// tempo de execução (TRANSFORM_W
TransformGroup textTrans = new Tr
textTrans.setCapability(Transform
objRaiz.addChild(textTrans);

// Cria um novo objeto que irá ap
// geométricas sobre texto e o ad
Transform3D trans = new Transform
trans.setTranslation(new Vector3d
textTrans.setTransform(trans);

// Text2D (java.lang.String text,
//        java.lang.String fontNa
Text2D text2D = new Text2D("E S F E R A !!",
                           new Color3f(0.9f, 0.1f, 0.9f),
                           "Helvetica", 30, Font.BOLD);
textTrans.addChild(text2D);
```


Criando Universos Virtuais

Texto e *Background*

- Exemplo: *Text3D*


```
TransformGroup textTrans = new Tra
textTrans.setCapability(Transform(
objRaiz.addChild(textTrans);

Transform3D trans = new Transform3
Transform3D t1 = new Transform3D(
t1.rotX(Math.toRadians(-10.0));
trans.mul(t1);
trans.setScale(0.3);
trans.setTranslation(new Vector3d(
textTrans.setTransform(trans);

Font3D font3d = new Font3D(new Fo
                           new Fo
Text3D textGeom = new Text3D(font3d, new String("ESFERA 3D !"),
                             new Point3f(-1.0f, 0.0f, 0.0f));
Shape3D textShape = new Shape3D(textGeom);
textShape.setAppearance(app);
textTrans.addChild(textShape);
```

Criando Universos Virtuais

Texto e *Background*

- Exemplo: *Text3D*

```
TransformGroup textTrans = new TransformGroup();
textTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
objRaiz.addChild(textTrans);

Transform3D trans = new Transform3D();
Transform3D t1 = new Transform3D();
t1.rotX(Math.toRadians(-10.0));
trans.mul(t1);
trans.setScale(0.3);
trans.setTranslation(new Vector3d(-0.6,0.6,0.0));
textTrans.setTransform(trans);

Font3D font3d = new Font3D(new Font("Helvetica", Font.PLAIN, 1),
                           new FontExtrusion());
Text3D textGeom = new Text3D(font3d, new String("ESFERA 3D !"),
                             new Point3f(-1.0f, 0.0f, 0.0f));
Shape3D textShape = new Shape3D(textGeom);
textShape.setAppearance(app);
textTrans.addChild(textShape);
```

Criando Universos Virtuais

Texto e *Background*

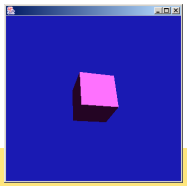
- *Background*
  - *Default*: cor sólida preta
  - API Java fornece uma maneira simples de especificar
    - Cor sólida
    - Imagem (sobrescreve a cor sólida)
    - Geometria (desenhada sobre a cor ou imagem)
    - Combinação das anteriores

Criando Universos Virtuais
Texto e *Background*

- *Background*
  - "Receita" para a especificação [Sun 2003]
    - Criar um objeto *Background* especificando uma cor ou imagem
    - Adicionar geometria (opcional)
    - Fornecer uma *Application Boundary* ou *BoundingLeaf*
    - Adicionar o objeto *Background* no grafo de cena

Criando Universos Virtuais
Texto e *Background*

- *Background*: exemplo 1



```
// Cria um "bounds" para o background
BoundingSphere bounds =
   new BoundingSphere(new Point3d(0.0,0.0,0.0), 100.0);

// Especifica um background azul e o adiciona no grafo
Color3f bgColor = new Color3f(0.1f, 0.1f, 0.7f);
Background bg = new Background(bgColor);
bg.setApplicationBounds(bounds);
objRaiz.addChild(bg);
```

Criando Universos Virtuais
Texto e *Background*

- *Background*: exemplo 2

```
// Cria um "bounds" para o background
BoundingSphere bounds =
          new BoundingSphere(new Point3d(0.0,0.0,0.0), 100.0);

// objeto para abrir imagens
Toolkit toolkit = Toolkit.getDefaultToolkit();

TextureLoader texturaBg =
           new TextureLoader(toolkit.getImage("parede.png"), this);
Background bg = new Background(texturaBg.getImage());
bg.setApplicationBounds(bounds);
bg.setImageScaleMode(Background.SCALE_REPEAT);
objRaiz.addChild(bg);
```

Criando Universos Virtuais
Texto e *Background*

- *Background*: exemplo 2

```
// Cria um "bounds" para o backgr
BoundingSphere bounds =
          new BoundingSphere(new

// objeto para abrir imagens
Toolkit toolkit = Toolkit.getDefa

TextureLoader texturaBg =
           new TextureLoader(too
Background bg = new Background(te
bg.setApplicationBounds(bounds);
bg.setImageScaleMode(Background.S
objRaiz.addChild(bg);
```

Exercício 3

Objetivo: verificar como criar textos (2D e 3D) e colocar *backgrounds* em aplicações Java 3D

Referências

[Bicho 2002] A. L. Bicho, L. G. da Silveira Jr, A. J. A. da Cruz e A. B. Raposo. **Programação Gráfica 3D com OpenGL, Open Inventor e Java 3D**. REIC - Revista Eletrônica de Iniciação Científica. v. II, n. I, março, 2002. http://www.sbc.org.br/reic/edicoes/2002e1/tutoriais/ProgramacaoGrafica3DcomOpenGLOpenInventoreJava3D.pdf

[Barrilleaux 2001] J. Barrilleaux. **3D User Interfaces with Java 3D**. Manning Publications Co. 2001. 499 p.

[Brown 1999] K. Brown. **Ready-to-run Java 3D**. New York, NY: John Wiley & Sons, 1999. 400 p.

[Lathrop 1997] O. Lathrop. **The Way Computer Graphics Works**. Wiley Computer Publishing, 1997.

Referências

[Sowizral 1998] H. Sowizral, K. Rushforth, M. Deering. **The Java™ 3D API Specification**. 2nd Edition. Addison-Wesley. 1998. 482 p.

[Sun 2003] Sun Microsystems Java 3D Engineering Team. **Java 3D API Tutorial**. Disponível em http://developer.java.sun.com/developer/onlineTraining/java3d/. Acesso: setembro/2003.

[Tori 2002] R. Tori, R. Nakamura. **Desenvolvimento de Jogos para Aprendizagem de Java 3D: Um Estudo de Caso**. Workshop de Jogos, 2002.

[Walsh 2002] A. E. Walsh. **Java 3D: API Jump-start**. Upper Saddle River, NJ: Prentice Hall, 2002. 245 p.