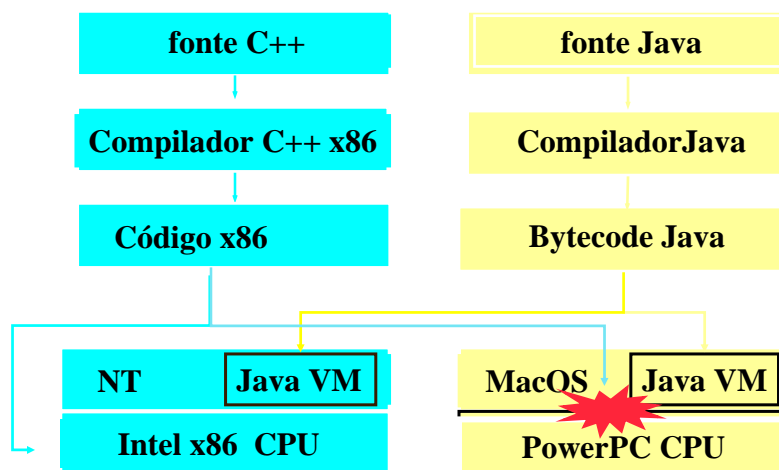


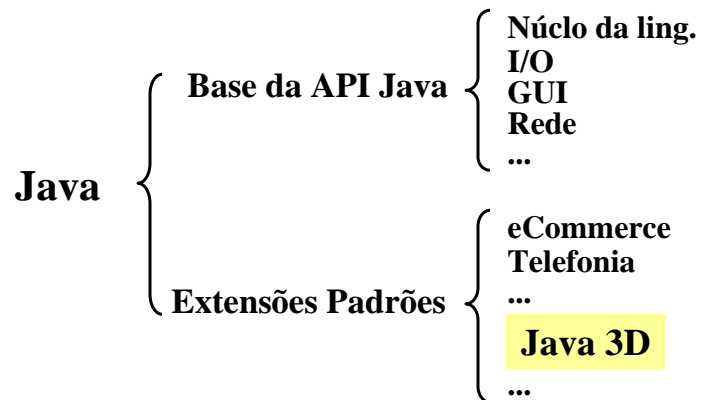
# Java 3D

Adailton J. A. da Cruz  
Alberto B. Raposo

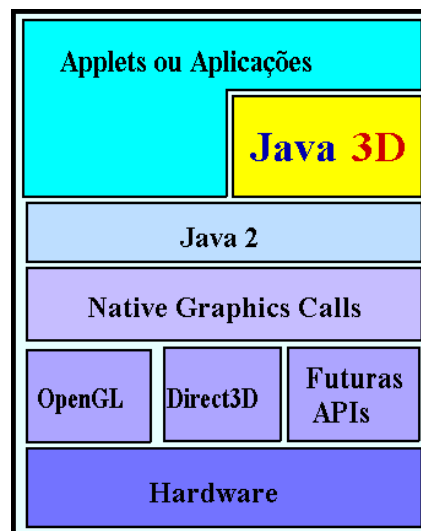
## Java: Independência de plataforma



## Localizando a API Java 3D



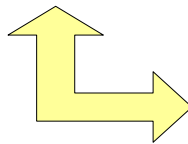
## Componentes



# Java 3D

- A Java 3D é uma API baseada em grafo de cenas que possibilita que programadores usem a tecnologia Java para aplicações 3D

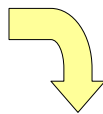
**3D para Java**



**Java para 3D**

## Facilidades

**Java para 3D**

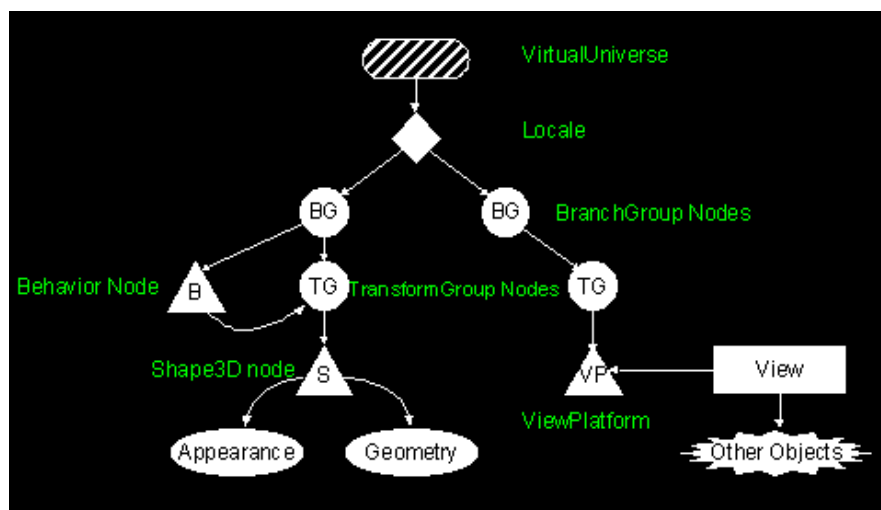


- portabilidade de aplicação
  - “Write once, run anywhere”
- independência de Hardware
  - o usuário decide que hardware é mais apropriado para ele
- 3D para rede
  - projetado para trabalhar na rede
  - ambientes colaborativos

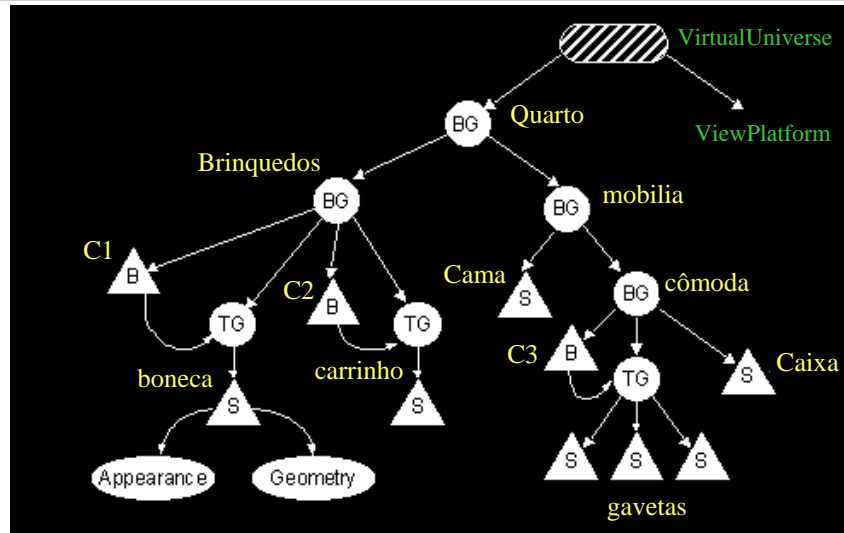
## Grafo de cena

- O grafo de cena é uma estrutura de dados do tipo árvore, usada para descrever a cena.
- Descrição de alto nível
  - objetos
  - posição
  - grupo

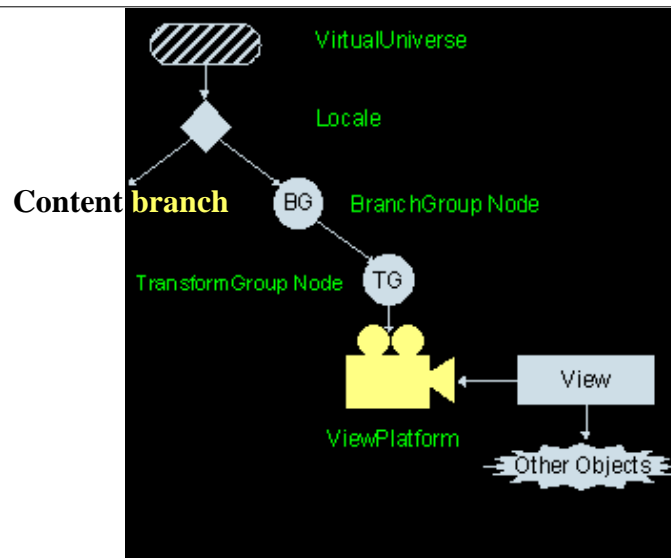
## Grafo de cena



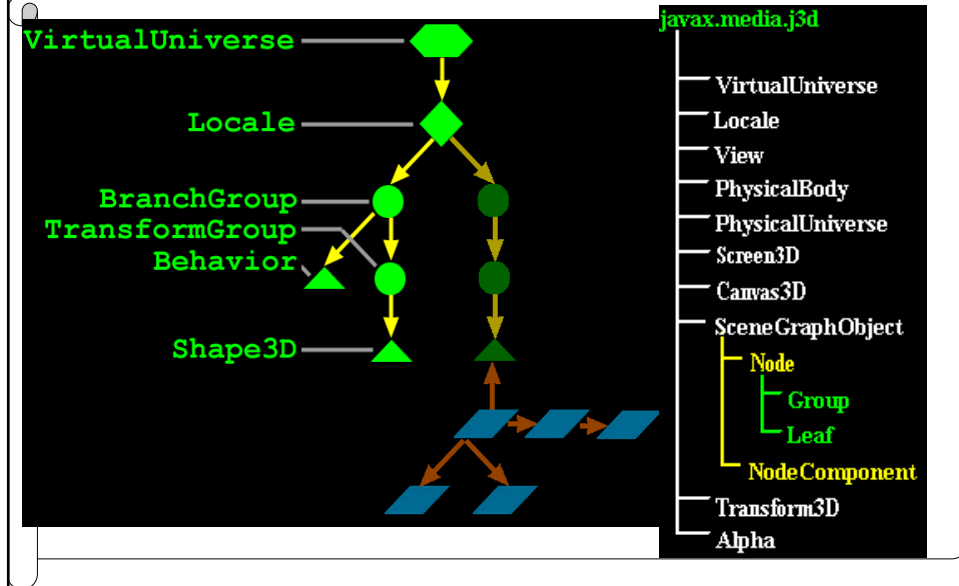
## Content branch



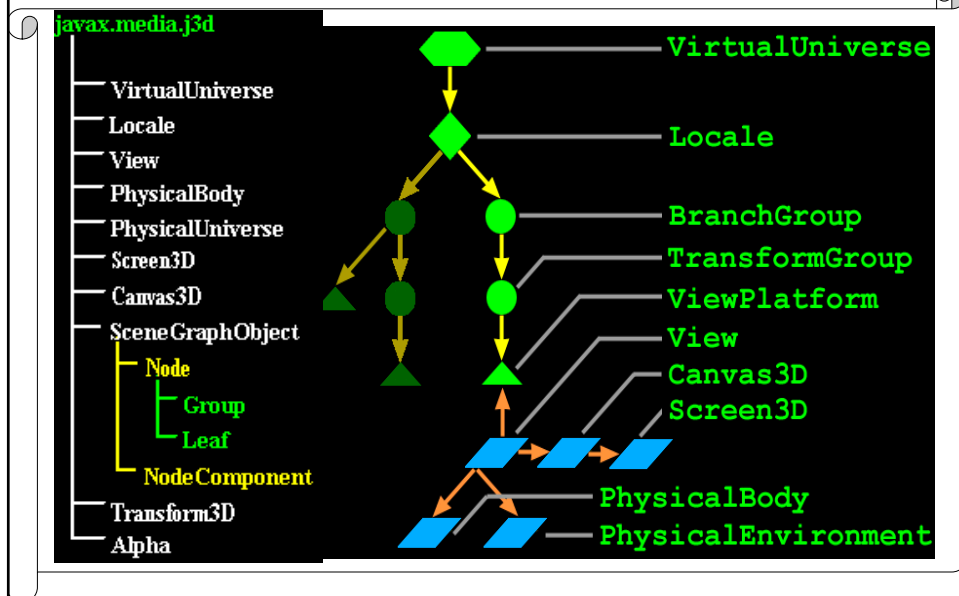
## View branch



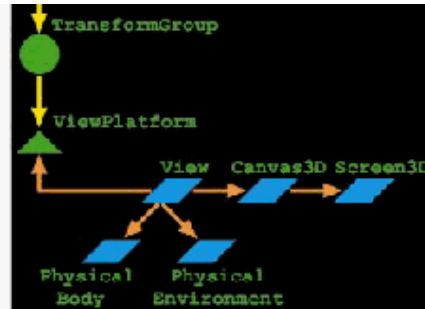
## Content branch - nós



## View branch - nós

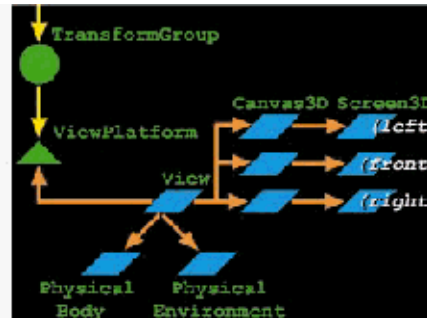
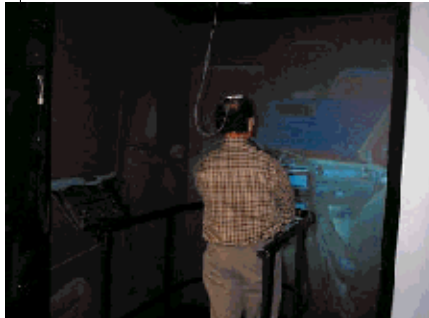


## Display (1)



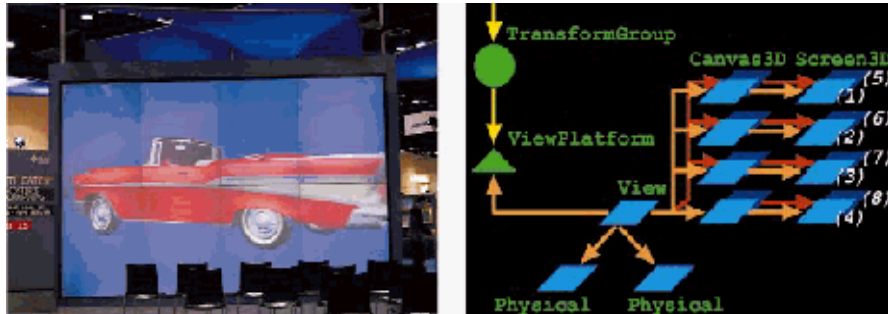
- Aplicação em RV
  - View branch : 1 canvas3D e 1 Screen3D

## Display (2)



- Quarto imerssivo com 3 paredes de projeção
  - View branch : 3 canvas3D e 3 Screen3D

## Display (3)



- Tela composta de 8 projetores
  - View branch : 8 canvas3D e 8 Screen3D

## Terminologia(1)

- Live
  - anexar um nó BG a um nó Locale torna todos objetos daquele branch “Live”
    - sujeitos a serem renderizados
    - converte o BG para um formato otimizado
- Compiled
  - converte BG para um formato otimizado
    - torna o processo de renderização mais eficiente
    - método `compile()`



## Terminologia(2)

- **Capability**

- Lista de parâmetros de um objeto que pode ser acessada
- objetos live ou compiled não podem ter seus parâmetros alterados, a menos que tenham sido previamente configurados

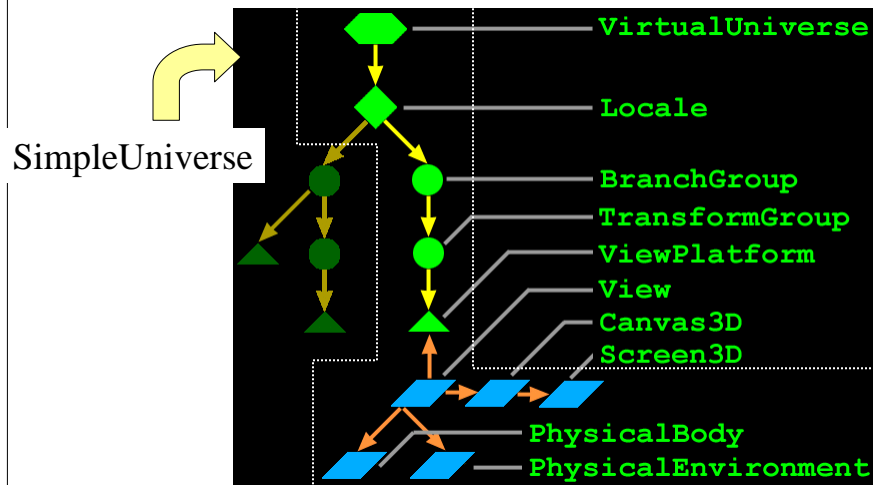


`setCapability( capability type)`

## Um programa Java3D

1. Criar um objeto **Canvas3D**
2. Criar um objeto **VirtualUniverse**
3. Criar objeto **Locale** e anexa-lo ao **VirtualUniverse**
4. Construir um grafo **view branch**
  - 4.1 Criar um objeto **View**
  - 4.2 Criar um objeto **ViewPlatform**
  - 4.3 Criar um objeto **PhysicalBody**
  - 4.4 Criar um objeto **PhysicalEnvironment**
  - 4.5 anexar objetos criados em 4.2, 4.3 e 4.4 ao **View**
5. Construir um (ou mais) grafo **content branch**
6. Compilar o(s) grafo(s) branch
7. Inserir os subgrafos no nó **Locale**

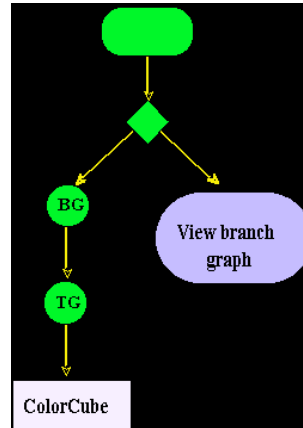
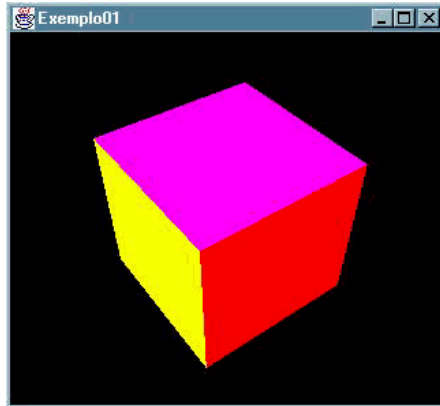
# SimpleUniverse



## Programa - versão simplificada

1. Criar um objeto **Canvas3D**
2. Criar um objeto **SimpleUniverse**
  - 2.1 Configurar o **SimpleUniverse**
3. Construir um (ou mais) grafo **content branch**
4. Compilar o(s) grafo(s) **content branch**
5. Inserir o(s) subgrafo(s) no nó **Locale** do **SimpleUniverse**

## Programa Exemplo01



## Exemplo 01

```
public class Exemplo01 extends Applet {  
    public Exemplo01 () {  
        .....  
        Canvas3D canvas3D = new Canvas3D(null); // P1  
        add("Center", canvas3D);  
        BranchGroup s = ConstroiContentBranch(); // P3  
        s.compile(); // P4  
        SimpleUniverse su = new SimpleUniverse(canvas3D); // P2  
        su.getViewingPlatform().setNominalViewingTransform(); // P2.1  
        su.addBranchGraph(s); // P5  
    }  
}
```

## Exemplo 01 (cont.)

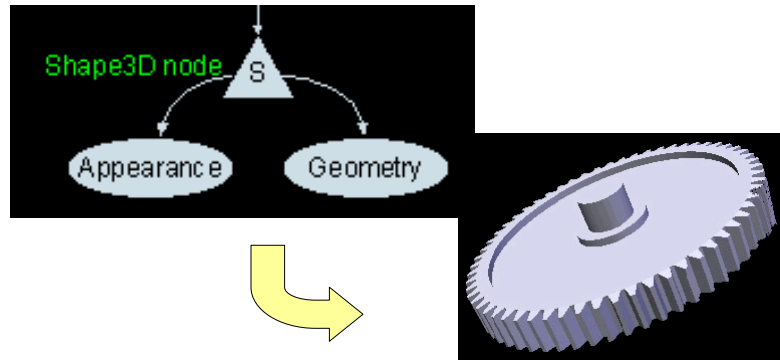
```
public BranchGroup ConstroiContentBranch( ) {  
    1. BranchGroup objRoot = new BranchGroup( );  
    2. Transform3D rotate1 = new Transform3D( );  
    3. Transform3D rotate2 = new Transform3D( );  
    4. rotate1.rotX(Math.PI/4.0d);  
    5. rotate2.rotY(Math.PI/5.0d);  
    6. rotate1.mul(rotate2);  
    7. TransformGroup objRotate = new TransformGroup(rotate1);  
    8. objRoot.addChild(objRotate);  
    9. objRotate.addChild(new ColorCube(0.4));  
    10. return objRoot;  
}
```

## Modelagem

- Objeto 3D = geometria + aparência
  - geometria descreve a forma ou estrutura
    - inclui coordenadas, normal a superfície, ...
  - aparência descreve os atributos
    - cor, transparência, tipo de material, textura entre outros

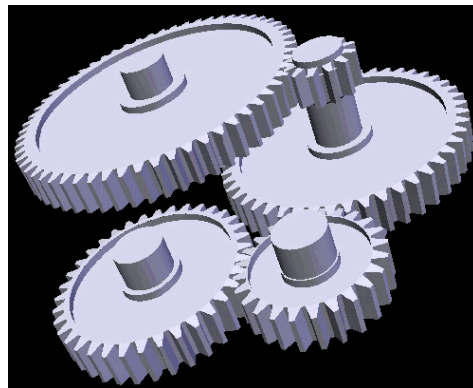
## Nó Shape3D

- Shape3D( Geometry g, Appearance a);
  - constrói e inicializa um objeto 3D



## Classe Objeto3D(1)

- definindo uma classe Objeto 3D
  - Engrenagem3D = new Engrenagem3D( )



## Classe Objeto3D (2)

```
public class Objeto3D extends Shape3D{
    private Geometry g;
    private Appearance a;
    public Objeto3D( ){
        g = constroiGeometria( );
        a = constroiAparencia( );
        this.setGeometry(g);
        this.setAppearance(a);
    }
    private Geometry constroiGeometria ( ){
        // inserir o código que cria a geometria desejada }
    private Appearance constroiAparencia ( ){
        // inserir o código que cria a aparência desejada } }
}
```

## Criando a geometria

- Opções na descrição da geometria
  - Primitivas
    - Haltere = esfera + cilindro + esfera
  - Descrição baseada em vértices
    - vértices    ➡ triângulos    ➡ objetos
  - Loaders
    - .wrl    ➡ VRML
    - .obj    ➡ Wavefront
    - .dxf    ➡ AutoCAD
    - .3ds    ➡ 3D Studio
    - .lws    ➡ LightWave

## Definindo a geometria (1)

- Primitivas

- `com.sun.utils.geometry.Primitive.*`  
adiciona funcionalidades geométrica

- classes disponíveis

- Box
    - Cone
    - Cylinder
    - Sphere

- ```
Appearance a = new Appearance();
Primitive caixa = new Box (X, Y, Z, a);
```

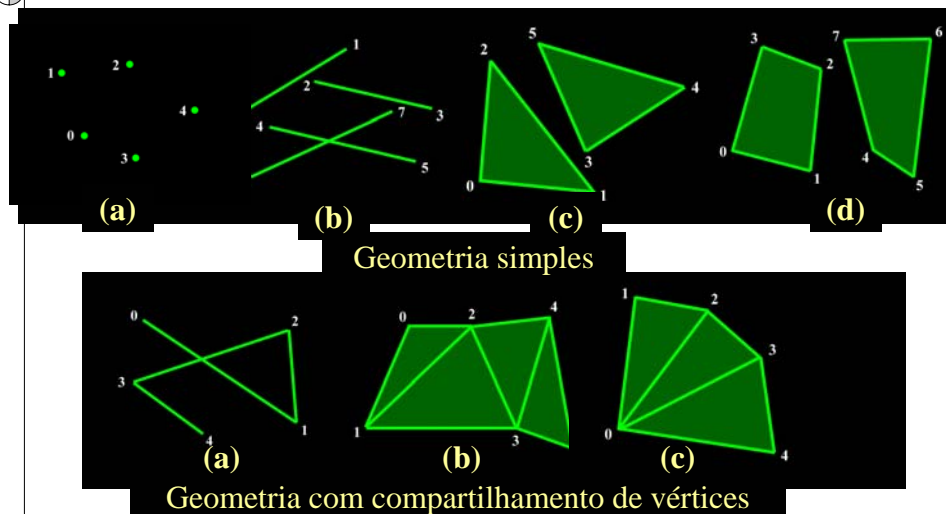
## Definindo a geometria (2)

- Descrição baseada em vértices

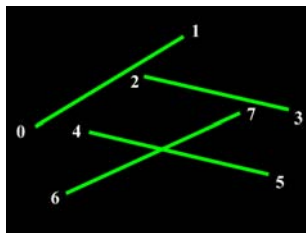
- As subclasses de **GeometryArray**  
formam a base para construção de pontos,  
linhas, triângulos e quadriláteros.

| <code>GeometryArray</code>    | <code>GeometryArray</code>         |
|-------------------------------|------------------------------------|
| └─ <code>LineArray</code>     | └─ <code>GeometryStripArray</code> |
| └─ <code>PointArray</code>    | └─ <code>LineStripArray</code>     |
| └─ <code>QuadArray</code>     | └─ <code>TriangleStripArray</code> |
| └─ <code>TriangleArray</code> | └─ <code>TriangleFanArray</code>   |

## Subclasses de GeometryArray(1)



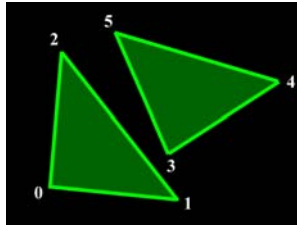
## Construindo um vetor de linhas



```
LineArray lines = new LineArray(  
    vertexCount,  
    GeometryArray.COORDINATES );  
lines.setCoordinates( 0, coords );
```

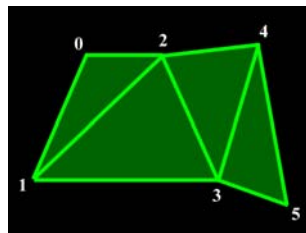


## Construindo um vetor de triângulos(1)



```
TriangleArray tris = new TriangleArray(  
    vertexCount,  
    GeometryArray.COORDINATES |  
    GeometryArray.NORMALS );  
tris.setCoordinates( 0, coords );  
tris.setNormals( 0, normals );
```

## Construindo um vetor de triângulos(2)



```
TriangleStripArray strips = new TriangleStripArray(  
    vertexCount,  
    GeometryArray.COORDINATES |  
    GeometryArray.NORMALS,  
    stripVertexCounts[] );  
strips.setCoordinates( 0, coords );  
strips.setNormals( 0, normals );
```

## Método constroiGeometria

```
private Geometry constroiGeometria (){  
    private static final float[] vertice = {  
        1.0f, -1.0f, 1.0f,  
        1.0f, 1.0f, 1.0f,  
        -1.0f, 1.0f, 1.0f,  
        -1.0f, -1.0f, 1.0f,  
        ( .. )  
    }  
    QuadArray cubo = new QuadArray( 24, COORDINATES);  
    quadrado.setCoordinates( 0, vertice);  
    return cubo;  
}
```

## Definindo a geometria (3)

- Loaders : Construir ou usar
  - abrir um dado formato de arquivo
  - Converter o conteúdo do arquivo em objetos Java 3D
  - Inserir os objetos no mundo virtual

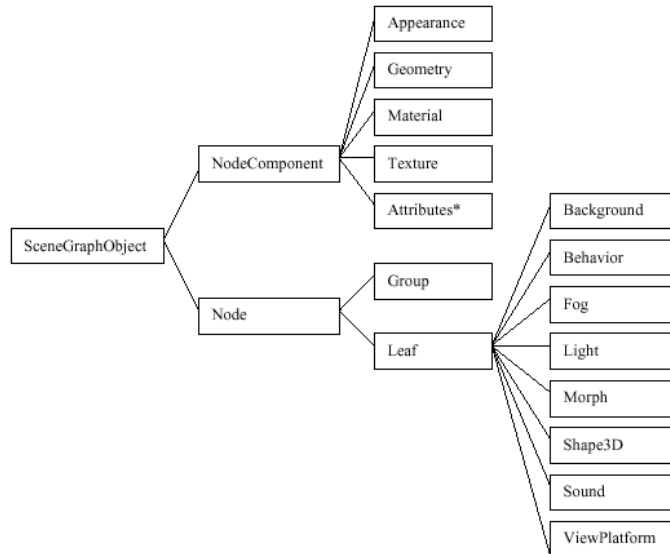
## Usando um Loader

```
public BranchGroup ConstroiContentBranch( ) {  
    BranchGroup objRaiz = new BranchGroup( );  
    ObjectFile f= new ObjectFile( );  
    Scene s = null;  
    try{  
        s = f.load("cena01.obj");  
    } catch ( FileNotFoundException e) {  
        system.out.println("Não foi possível abrir arquivo . . .")  
        system.exit(1);  
    }  
    objRaiz.addChild (s.getSceneGroup( ) );  
}
```

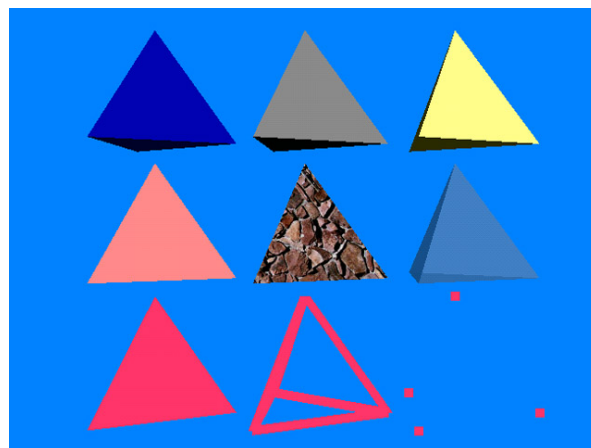
## Definindo a Aparência

- Controla como o J3D “mostra” a geometria
  - cor
  - Transparência
  - entre outros
- Estes controles são encapsulados em um objeto Appearance
  - Estas informações são mantidas em objetos NodeComponent

## Hierarquia de classes



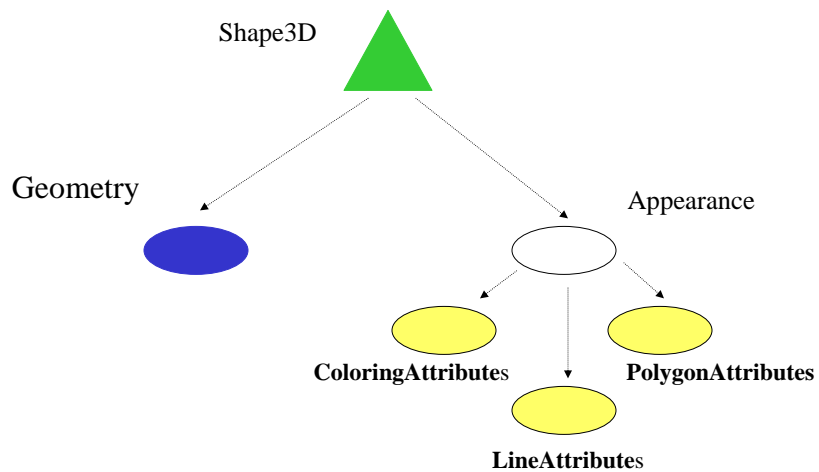
## Exemplo de atributos



## Método constroiAparência

```
private Appearance constroiAparencia ( ){  
    Appearance ap = new Appearance();  
    LineAttributes al = new LineAttributes();  
    al.setLineWidth(2.0f);  
    al.setLinePattern(PATTERN_SOLID);  
    ap.setColoringAttributes(al);  
    ColoringAttributes aCor = new ColoringAttributes();  
    aCor.setColor(new Color3f(1.0f, 0.0f, 0.0f));  
    ap.setColoringAttributes(aCor);  
    PolygonAttributes pa = new PolygonAttributes();  
    pa.setPolygonMode(PolygonAttributes.Polygon_FILL);  
    ap.setPolygonAttributes(pa);  
    return ap;  
}
```

## Grafo de cenas




## Interação

- Mudanças efetuadas no mundo virtual em função da ação direta do usuário
- As interações são especificadas em J3D usando **Behaviors**.



Mecanismo usado para adicionar movimento e ação ao mundo virtual

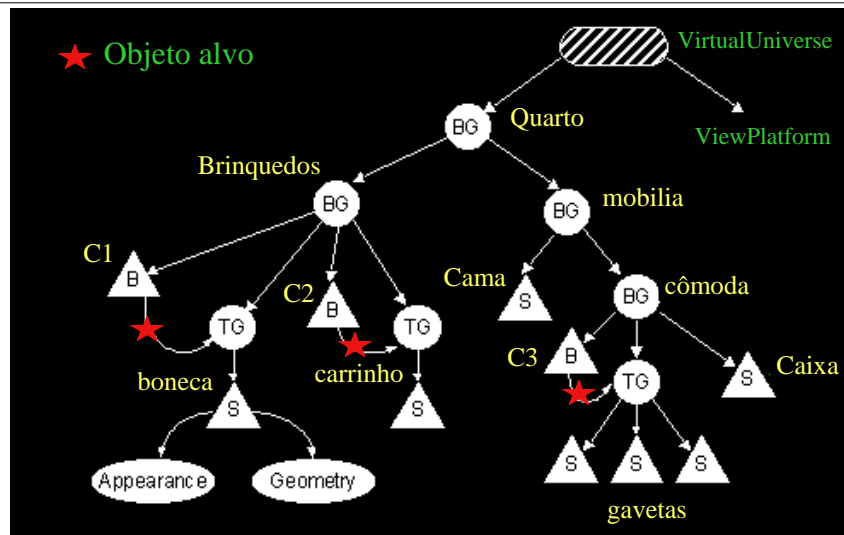
## Behavior

- Os Behaviors são chamados para executar uma ação somente quando ocorrer os eventos associados a ele
- 
- As WakeupCondition são formadas por um conjunto de objetos denominados WakeupCriterion

## Especificando eventos

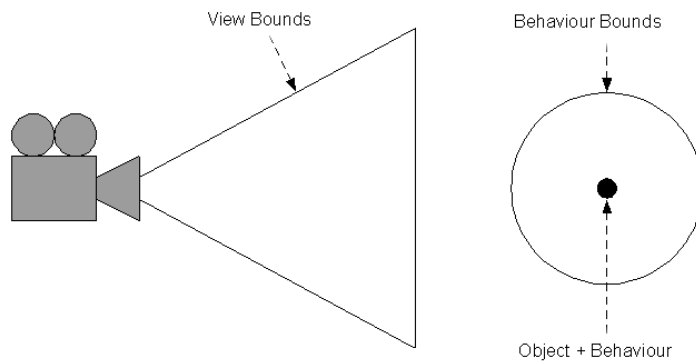
- WakeupCriterion objetos J3D usados para especificar um evento ou uma combinação lógica de eventos
  - pressionar teclas do mouse/teclado
  - ocorrência de um intervalos de tempo
  - alteração dos parâmetros de uma transformação
  - entre outros mais

## Nó Behavior

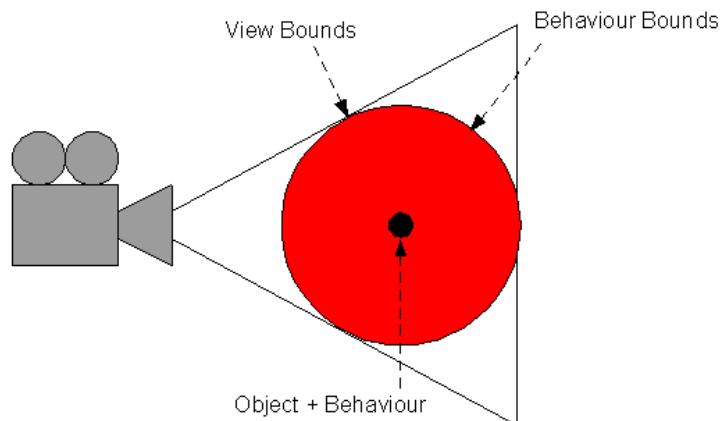


## Região de influência do Behavior

- scheduling bound
  - melhora performance do sistema



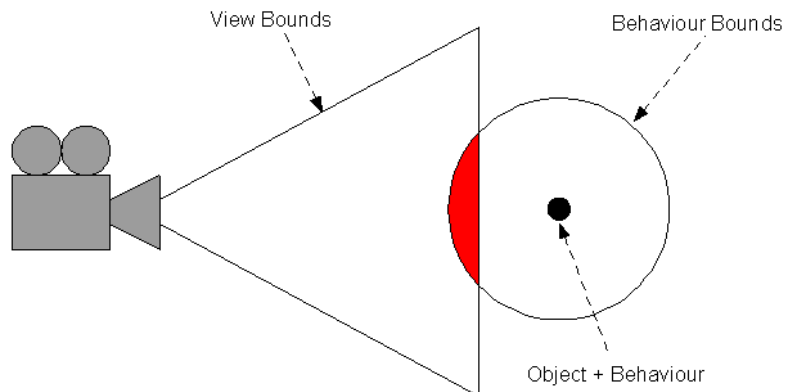
## Behavior ativos(1)





## Behavior ativos(2)

Problema: Objeto não está visível



## Estrutura de um Behavior

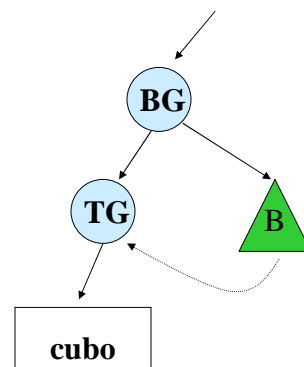
- Todo behavior contém
  - um método construtor
  - um método initialize( )
    - chamado quando o behavior torna-se "live"
    - valor inicial da WakeupContition
  - um método processStimulus()
    - chamado quando a condição de disparo ocorrer e o behavior correspondente estiver ativo
    - efetua as mudanças especificadas no método
    - define a próxima WakeupContition

## Criando um Behavior

```
public class Rotaciona extends Behavior {  
    private TransformGroup alvo;  
    private Transform3D r = new Transform3D();  
    private double angulo = 0.0;  
    Rotaciona ( TransformGroup alvo){  
        this.alvo = alvo;    }  
    public void initialize() {  
        this.wakeupOn ( new  
            WakeupOnAWTEvent(KeyEvent.KEY_PRESSED)); }  
    public void processStimulus( Enumeration criteria) {  
        angulo += 0.1;  
        r.rotY (angulo);  
        alvo.setTransform(r);  
        this.wakeupOn( new  
            WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));}  
}
```

## Usando um behavior

- Criar objeto alvo
- Criar um Behavior e efetuar as referências necessárias
- definir a região de influência
- configurar as capacidades do objeto alvo



## Programa exemplo

```
public BranchGroup ConstroiContentBranch() {  
    BranchGroup objRoot = new BranchGroup();  
    TransformGroup objRotate = new TransformGroup();  
    objRotate.setCapability  
        (TransformGroup.ALLOW_TRANSFORM_WRITE);  
    objRoot.addChild(objRotate);  
    objRotate.addChild(new ColorCube(0.4));  
  
    Rotacional RotacionalBehavior = new Rotacional(objRotate);  
    RotacionalBehavior.setSchedulingBounds (new  
        BoundingSphere());  
    objRoot.addChild(RotacionalBehavior);  
    objRoot.compile();  
    return objRoot;}  

```

## Conclusão

- Alto nível X Flexibilidade
- Maior produtividade
- Uso da VRML no processo de modelagem
- Documentação para programação
- Referências para Java 3D