Java 3D

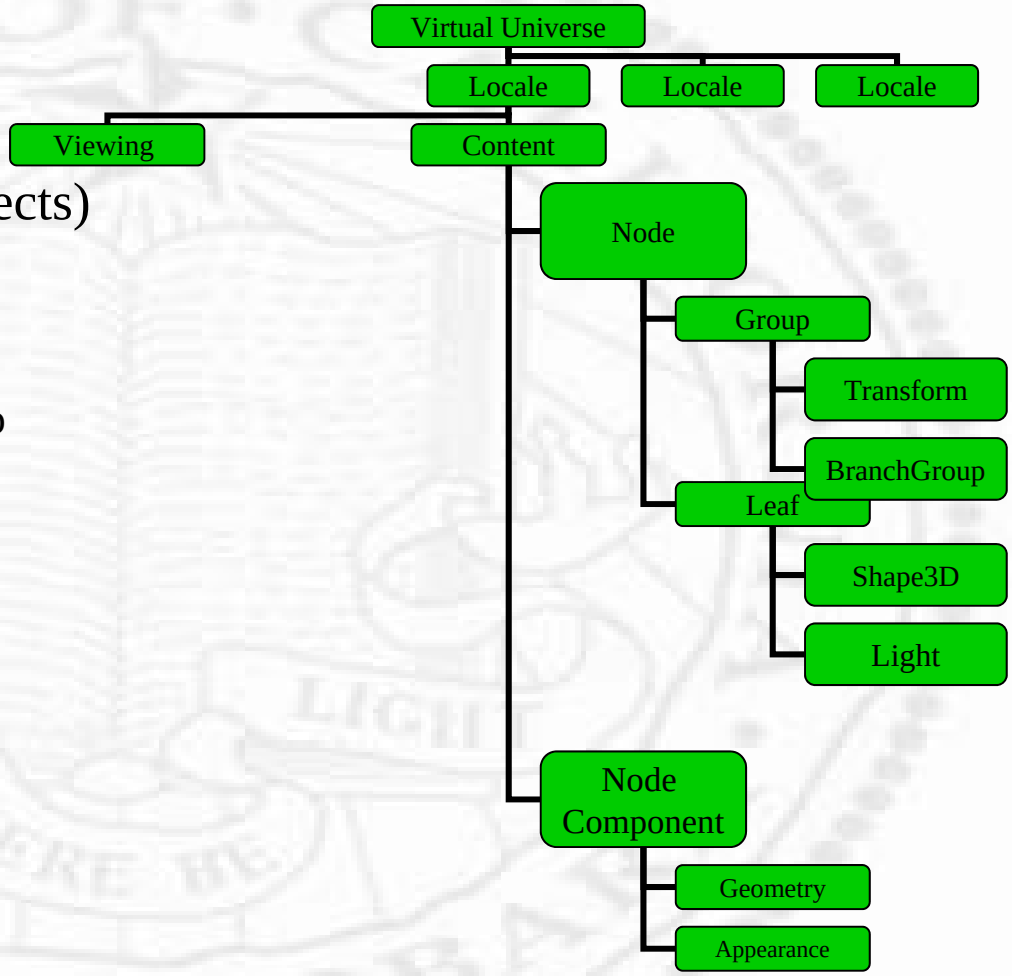
A *very* short summary

High-Level Scene Description

- Represent scenes as graphs
  - Concentrate on structure and content, not rendering details
  - Parallel traversal and rendering by JVM
  - Compiled down to either OpenGL or DirectX
- Advantage
  - Multiple platforms
  - Multiple display environments
  - Multiple input devices
  - *Write once, run anywhere in 3D*

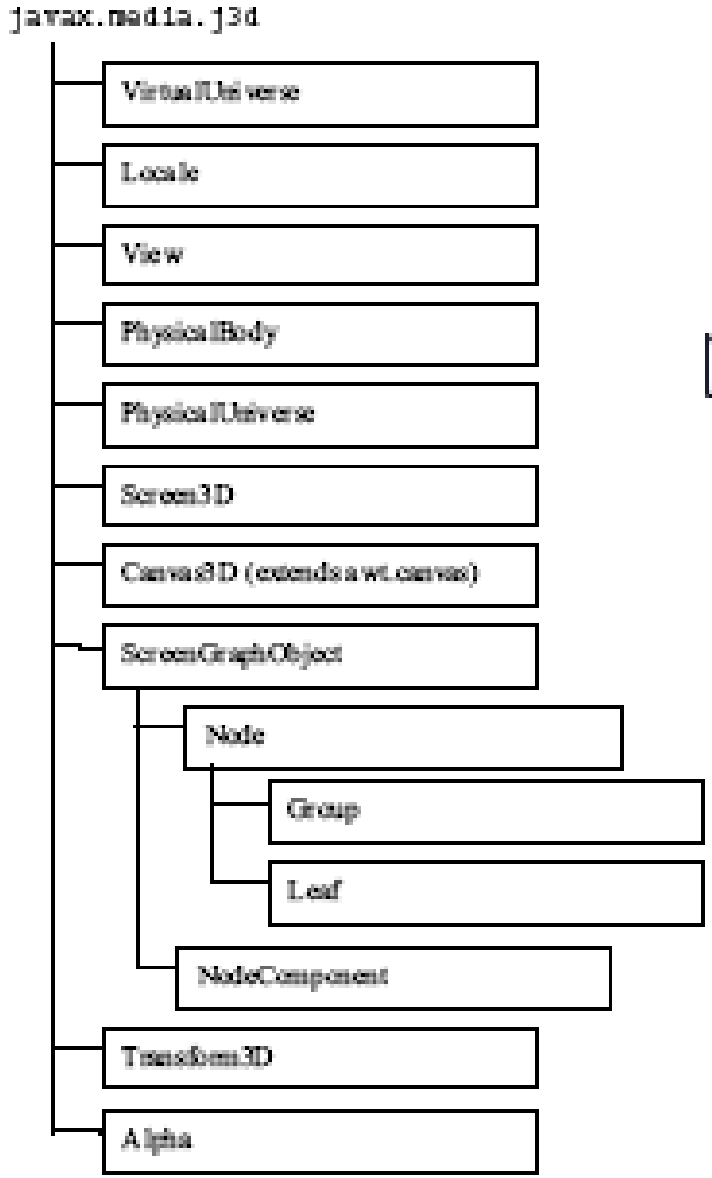
Scene Graphs

- Virtual Universe
- Locale
  - Viewing platform
  - Content (SceneGraphObjects)
    - Node
      - Group
        - Branchgroup
        - Transformgroup
      - Leaf
        - Shape
        - Light
        - Behavior
        - Sound
    - NodeComponent
      - Geometry
      - Color
      - Etc.



(Abbreviated) Class Hierarchy

- java.lang.object
  - javax.media.j3d.SceneGraphObject
    - javax.media.j3d.Node
      - javax.media.j3d.Group
        - javax.media.j3d.BranchGroup
        - javax.media.j3d.TransformGroup
      - javax.media.j3d.Leaf
        - javax.media.j3d.Behavior
        - javax.media.j3d.BoundingLeaf
        - javax.media.j3d.Light
        - javax.media.j3d.Shape3D
    - javax.media.j3d.NodeComponent
      - javax.media.j3d.Appearance
      - javax.media.j3d.ColorAttributes
      - javax.media.j3d.Material
      - javax.media.j3d.Geometry
        - javax.media.j3d. TriangleArray
  - javax.media.j3d.Alpha
  - Javax.media.j3d.Transform3D



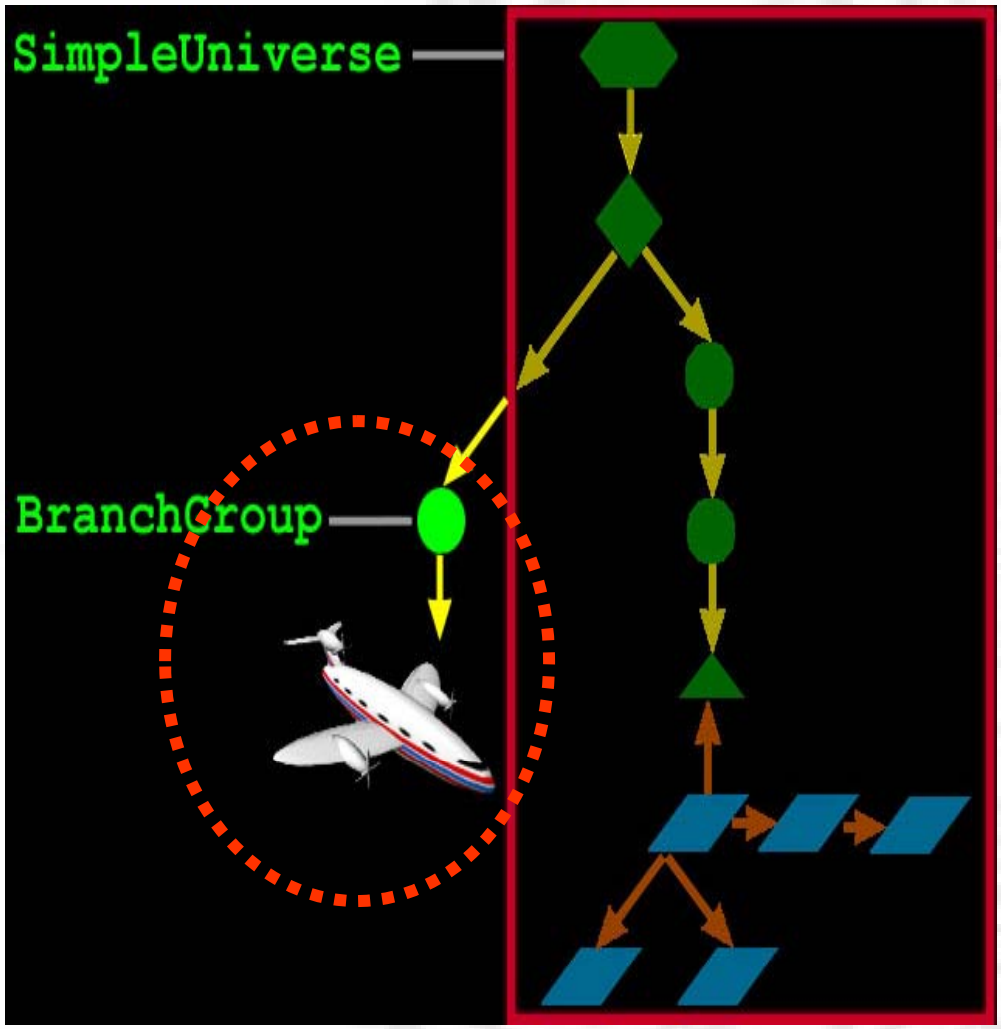
Cook Book Procedure

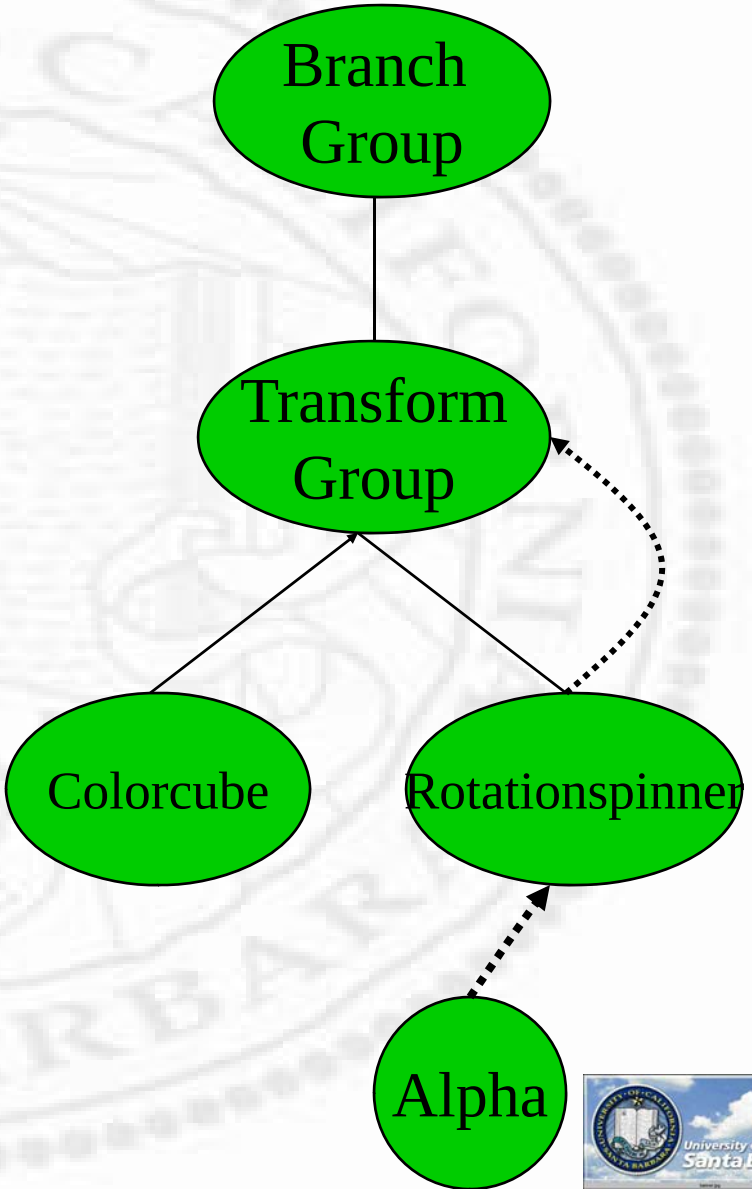
1. Create a Canvas3D object
2. Create a VirtualUniverse object
3. Create a Locale object, attaching it to the VirtualUniverse object
4. Construct a view branch graph
   a. Create a View object
   b. Create a ViewPlatform object
   c. Create a PhysicalBody object
   d. Create a PhysicalEnvironment object
   e. Attach ViewPlatform, PhysicalBody, PhysicalEnvironment, and Canvas3D objects to View object
5. Construct content branch graph(s)
6. Compile branch graph(s)
7. Insert subgraphs into the Locale

Use simpleUniverse solves steps 2-4 above
Concentrate on “content” not “rendering”

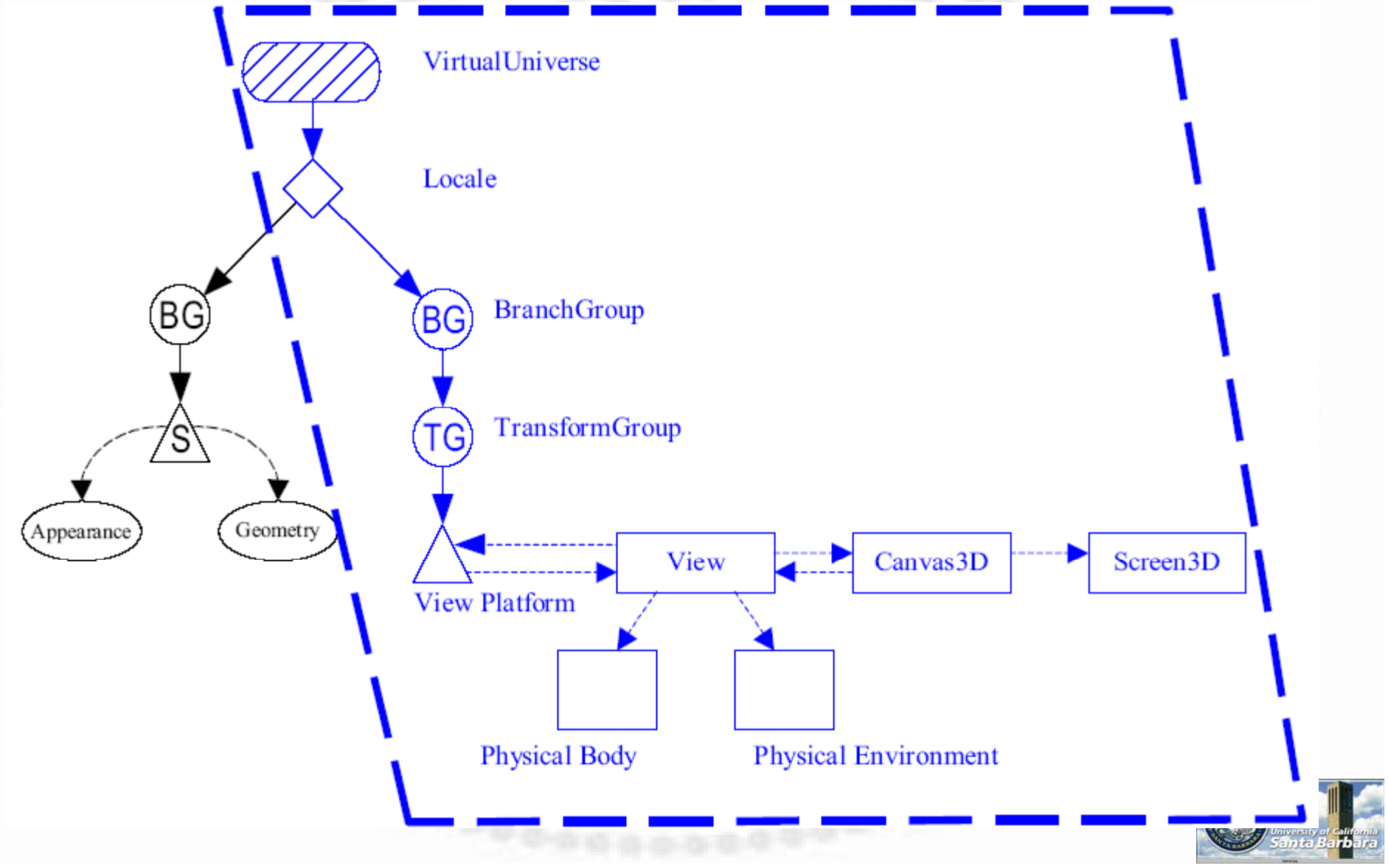
1. Create a Canvas3D Object
2. Create a SimpleUniverse object which references the earlier Canvas3D object
   a. Customize the SimpleUniverse object
3. Construct content branch
4. Compile content branch graph
5. Insert content branch graph into the Lccale of the SimpleUniverse

Simple Universe





More Detail


VirtualUniverse
Locale
BG
BG
BranchGroup
S
TG
TransformGroup
Appearance
Geometry
View
Canvas3D
Screen3D
View Platform
Physical Body
Physical Environment

Sample Program

```
import javax.media.j3d.*;
import javax.vecmath.*;
import java.applet.*;
import java.awt.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.applet.MainFrame; // for application +
    applet
```

Sample Program (cont.)

```
public class HelloWorld
{
    public static void main( String[] args ) {
        Frame frame = new Frame( );
        frame.setSize( 640, 480 );
        frame.setLayout( new BorderLayout( ) );
        GraphicsConfiguration config =
                SimpleUniverse.getPreferredConfiguration();
        Canvas3D canvas = new Canvas3D( config );
        frame.add( "Center", canvas );

        // set up the top structure and view branch
        SimpleUniverse univ = new SimpleUniverse( canvas );
        univ.getViewingPlatform( ).setNominalViewingTransform( );
        // create, add and compile content
        BranchGroup scene = createContent( );
        scene.compile( );
        univ.addBranchGraph( scene );

        frame.show( );
    }
```

Frame

Canvas3D

SimpleUniverse

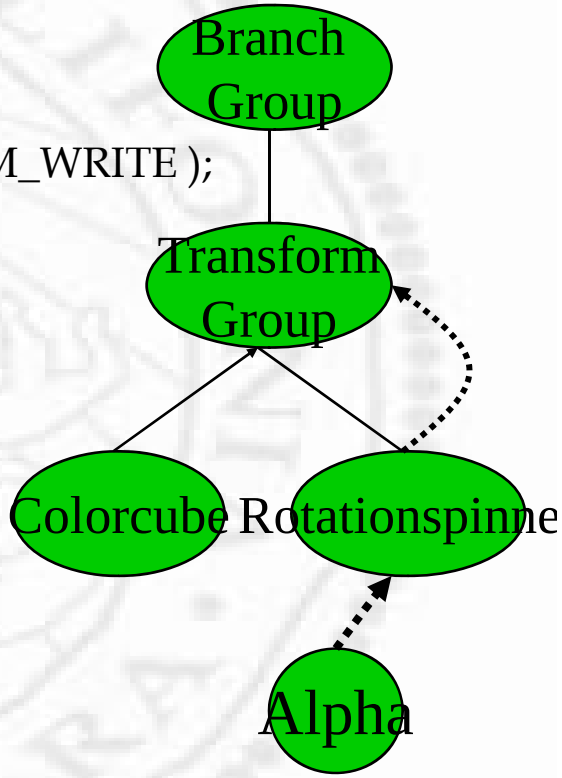
BranchGroup

Skeleton

- Layout
- Frame
- Canvas3D
  - Use a simple layout mechanism
- Content
- Universe
  - Use SimpleUniverse
- BranchGroup for content
  - Scene definitions

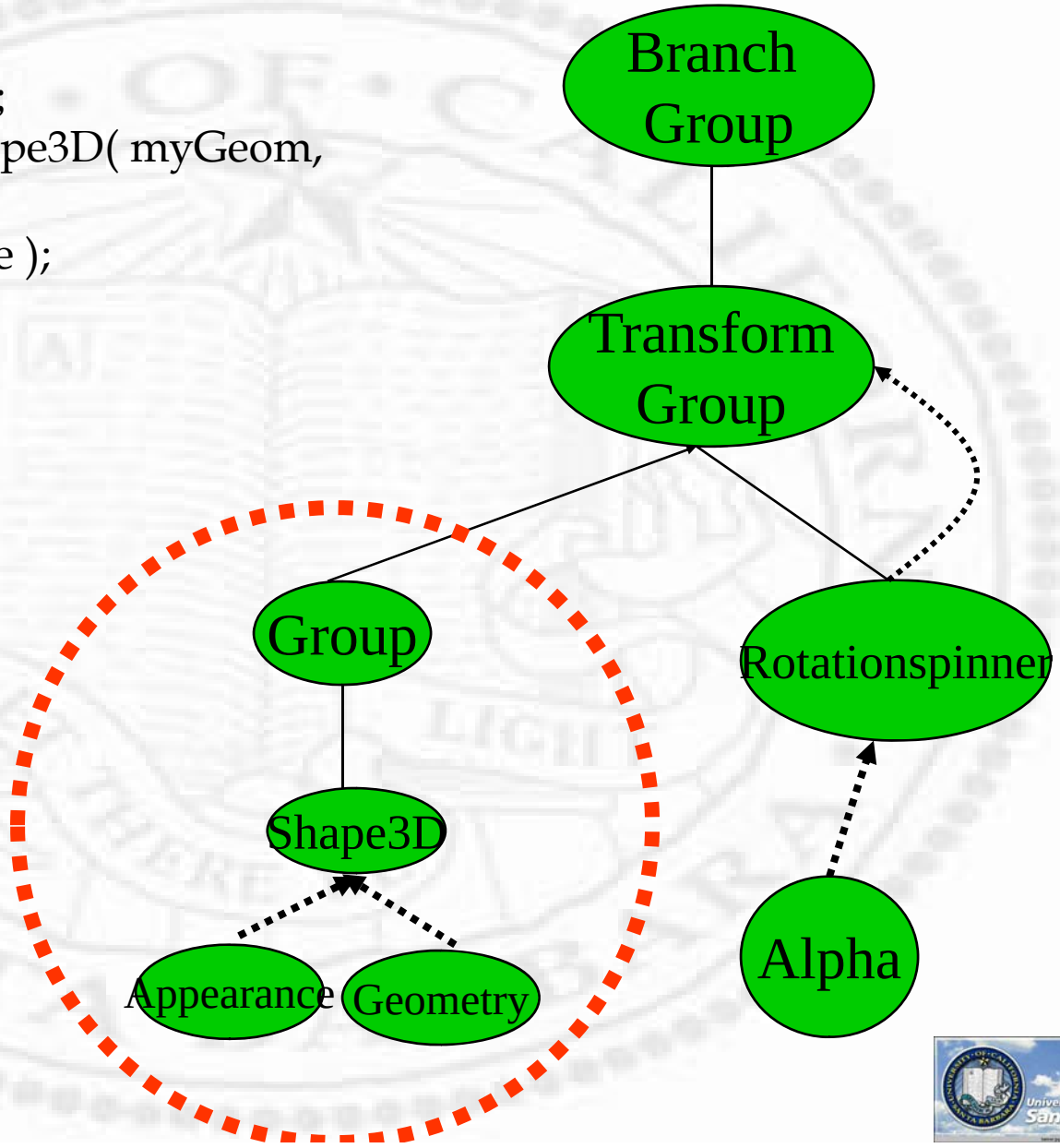
Sample Program (cont.)

```
private static BranchGroup createContent( )
    {
        // Make a scene graph branch
        BranchGroup branch = new BranchGroup( );
        // Make a changeable 3D transform
        TransformGroup trans = new TransformGroup( );
        trans.setCapability( TransformGroup.ALLOW_TRANSFORM_WRITE );
        branch.addChild( trans );
        // Make a shape
        ColorCube demo = new ColorCube( 0.4 );
        trans.addChild( demo );
        // Make a behavor to spin the shape
        Alpha spinAlpha = new Alpha( -1, 4000 );
        RotationInterpolator spinner =
                new RotationInterpolator( spinAlpha, trans );
        spinner.setSchedulingBounds(
                new BoundingSphere( new Point3d( ), 1000.0 ) );
        trans.addChild( spinner );
        return branch;
    }
}
```

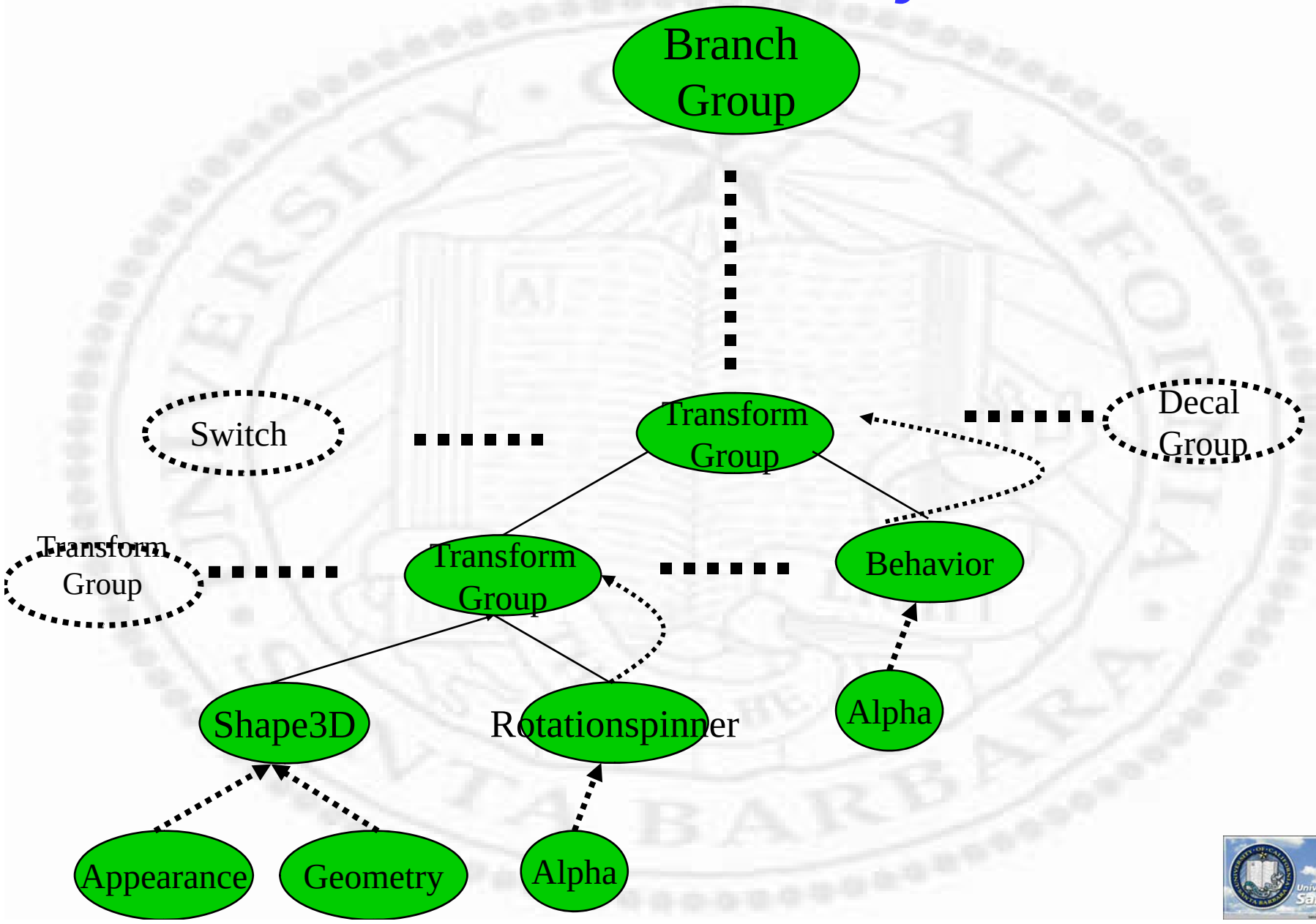


More generally

```
Group myGroup = new Group( );
Shape3D myShape = new Shape3D( myGeom,
myAppear );
myGroup.addChild( myShape );
trans.addChild( myGroup );
```



Most Generally



Sample Program – Applet Version

Applet in place of frame

```
1. public class HelloJava3Da extends Applet {
2.     public HelloJava3Da() {
3.         setLayout(new BorderLayout());
4.         GraphicsConfiguration config =
5.            SimpleUniverse.getPreferredConfiguration();
6.         Canvas3D canvas3D = new Canvas3D(config);
7.         add("Center", canvas3D);
8.
9.         BranchGroup scene = createSceneGraph();
10.        scene.compile();
11.
12.        // SimpleUniverse is a Convenience Utility class
13.        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
14.
15.        // This moves the ViewPlatform back a bit so the
16.        // objects in the scene can be viewed.
17.        simpleU.getViewingPlatform().setNominalViewingTransform();
18.
19.        simpleU.addBranchGraph(scene);
20.    } // end of HelloJava3Da (constructor)
```

Sample Program – Applet or Application

```
1.      //  The following allows this to be run as an application
2.      //  as well as an applet
3.
4.      public static void main(String[] args) {
5.          Frame frame = new MainFrame(new HelloJava3Da(), 256, 256);
6.      } // end of main (method of HelloJava3Da)
```

OpenGL vs. Java3D

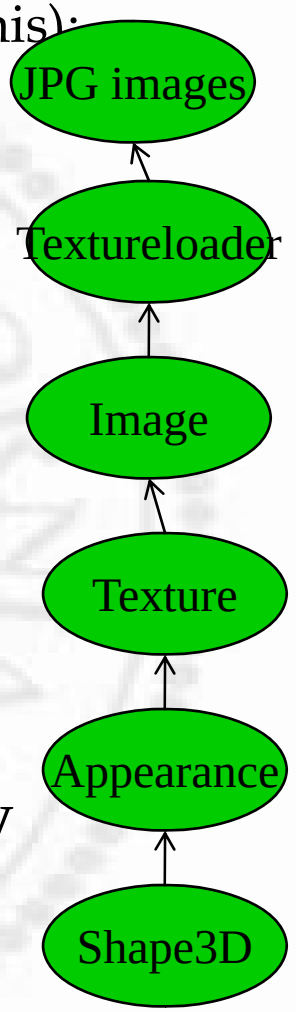
- Java3D is much easier to use and program
- However, for speed consideration, you probably want to program in OpenGL or DirectX
- Java3D is good for
  - Fast prototyping
  - Engineering and scientific applications
  - Cross platform, web-enabled applications
  - Caveat: web-integration is non-trivial (Flash is much better)

Example: Texture Mapping

```
TextureLoader myLoader = new TextureLoader("brick.jpg", this);
ImageComponent2D myImage = myLoader.getImage();

Texture2D myTex = new Texture2D ();
myTex.setImage(0,myImage);
myTex.setEnable(true);

Appearance myAppear = new Appearance();
myAppear.setTexture(myTex);
Shape3D myShape = new Shape3D(myGeom, myAppear);
```



- Comparing to OpenGL, Java3D code is usually *much* simpler

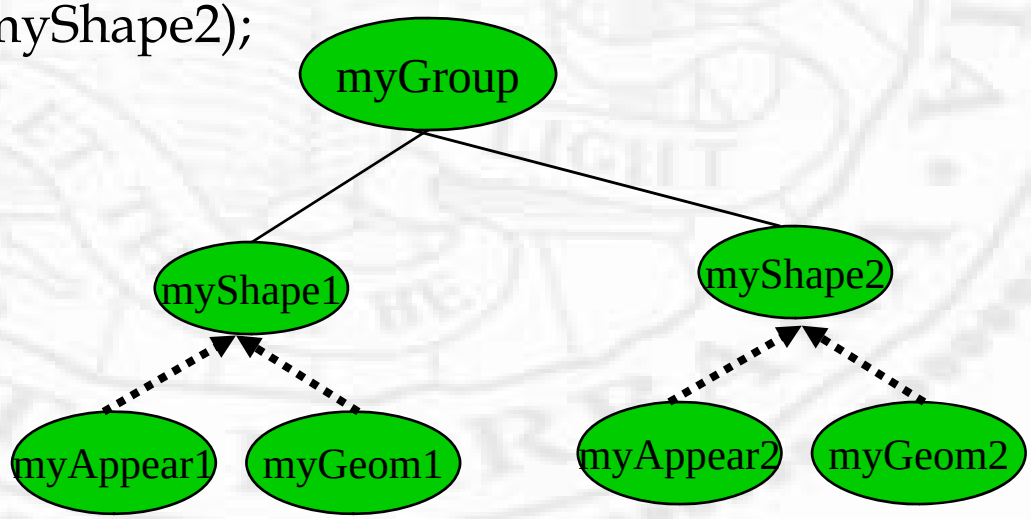
Nitty-Gritty Details

- There are still many, but to do the programming assignments, there are just a few

Most Basics

- 3D Shape + Appearance + Geometry

```
Shape3D myShape1 = new Shape3D (myGeom1, myAppear1);
Shape3D myShape2 = new Shape3D (myGeom2);
myShape2.setAppearance(newAppear2);
Group myGroup = new Group();
myGroup.addChild(myShape1);
myGroup.addChild(myShape2);
```



Controlling Access

- Read/Write permission at run time
- Don’t allow write permission for optimization

```
Shape3D myShape = new Shape3D (myGeom, myAppear);
myShape.setCapability(Shape3D.ALLOW_APPEARANCE_WRITE);
myShape.setAppearance(newAppearance); //ok
myShape.setGeometry(newGeom); //error
```

Fill in Contents - Geometry

- Various types of array objects
  - javax.media.j3d.GeomtryArray
    - setCoordinates (int index, * coorindate);
    - setNormals (int index, * normal);
    - setColors (int index, * color);
    - setTextureCoordinates (int index, * texCood);

Geometry

```
Point3f[] myCoords = {
        new Point3f (0.0f, 0.0f, 0.0f);
        …
}
Vector3f[] myNormals = {
        new Vector3f (0.0f, 1.0f, 0.0f);
        …
}
TriangleArray myLines = new TriangleArray (
        myCoords.length,
        GeometryArray.COORDINATES |
        GeometryArray.NORMALS);
myLines.setCoordinates(0, myCoords);
myLines.setNormals(0,myNormals);
Shape3D myShape = new Shape3D(myLines, myAppar);
```

Fill in Contents - Appearance

- Point, line style
- Intrinsic color (no lighting)
- Material (with lighting)
- Texture
- Etc.
  - javax.media.j3d.Appearance
    - setColoringAttributes
    - setMaterial
    - setRenderingAttributes
    - Etc.

Appearance

```
ColoringAttributes myCA = new ColoringAttributes();
myCA.setColor(1.0f, 1.0f, 0.0f);

myCA.setShadeModel(
   coloringAttributes.SHADE_GOURAUD);

Appearance myAppear = new Appearance();
myApepar.setColoringAttributes(myCA);

Shape3D myShape = new Shape3D( myGeom, myApppear);
```

Appearance - Texture

- Similar to OpenGL
  - Texture coordinates go from 0 to 1 for both *s*- and *t*- direction
  - Texture image size must be 2^m by 2^n

Texture Coordinate Mapping

```
Point3f[] myCoords = {
      new Point3f( 0.0f, 0.0f, 0.0f );
      …
}
Vector3f[] myNormals = {
    new Vector3f( 0.0f, 1.0f, 0.0f );
    …
}
Point2f[] myTexCoords = {
    new Point2f (0.0f, 0.0f );
    …
}
```

```
QuadArray myQuads = new
    QuadArray { myCoods.length,
GeometryArray.COORDINATES |
GeometryArray.NORMALS |
Geometry.TEXTURE_COORDINATE_2
    );
myQuads.setCoordinates(0, myCoords);
myQuads.setNormals(0, myNormals);
myQuads.setTextureCoordinates (0,
    myTexCoords);
Shape3D myShape = new Shape3D(
    myQuads, myAppear);
```

Other Texture Details

- Wrap or Clamp at Boundary

```
myTex.setBoundaryModeS (Textuer.WRAP);
myTex.setBoundaryModeT (Textuer.WRAP);
```

Other Texture Details

- Texture mode controls mapping

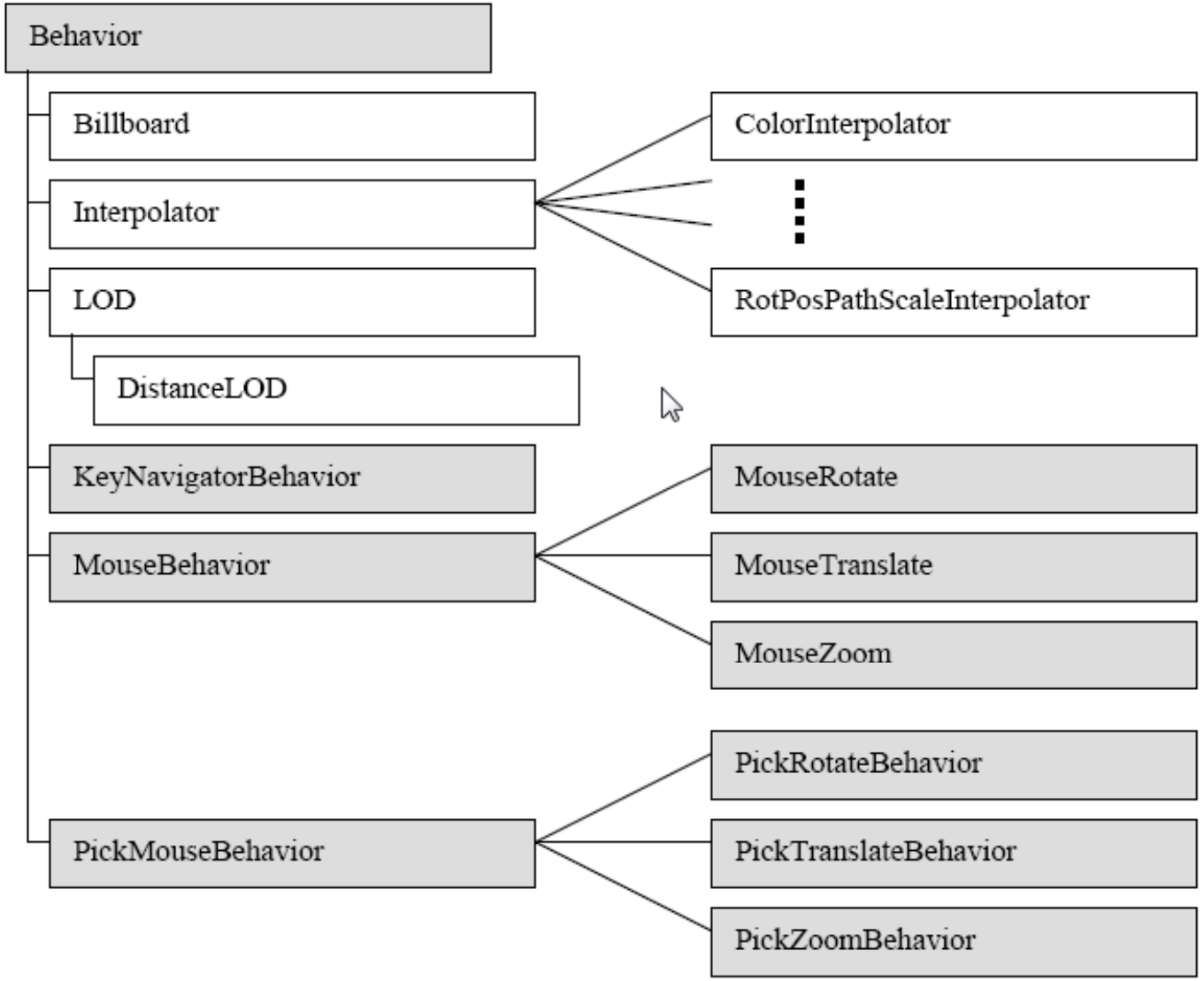
```
TextureAttributes myTA = new TextureAttributes();
myTA.setTextureMode ( Texture.MODULATE);
Appearance myAppear = new Appearance();
myAppear.setTextureAttributes ( myTA);
```

Mode	**Result color**	**Result transparency**
REPLACE	T_{rgb}	T_a
DECAL	$S_{rgb}*(1-T_a)+T_{rgb}*T_a$	S_a
MODULATE	$S_{rgb}*T_{rgb}$	S_a*T_a
BLEND	$S_{rgb}*(1-T_{rgb})+B_{rgb}*T_{rgb}$	S_a*T_a

S_{rgb} is the color of the shape being texture mapped
S_a is the alpha of the shape being texture mapped
T_{rgb} is the texture pixel color
T_a is the texture pixel alpha
B_{rgb} is the shape blend color
B_a is the shape blend alpha

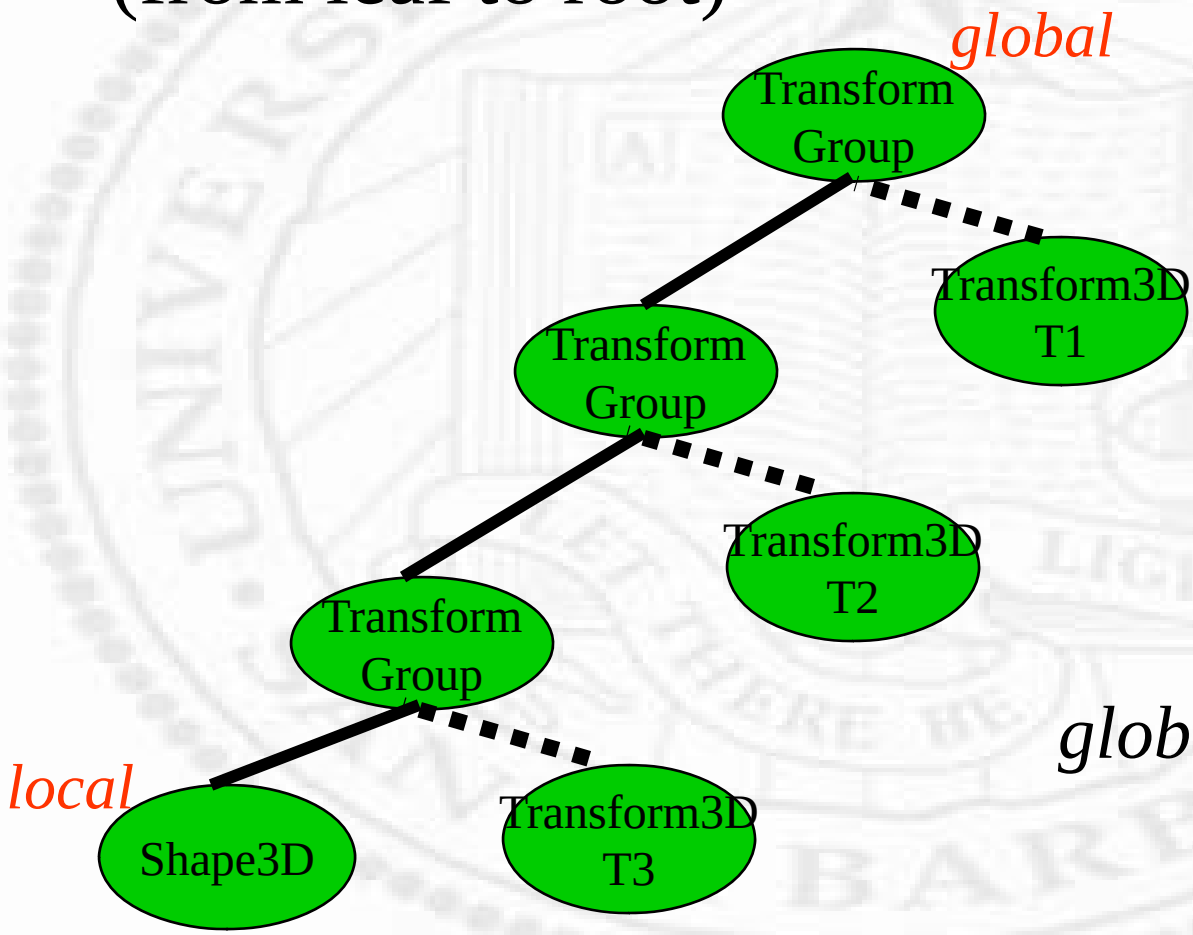
Behavior + Animation

- Static
  - Transformation (position & orientation an object in space)
- Dynamic
  - Animation (e.g., rotating a cube in space, indexed by time)
    - Different inerpolators
  - Behavior (e.g., respond to mouse or keyboard inputs)
    - Hard way: implement your own behavior classes
    - Easy way: use existing behavior classes


Behavior
Billboard
Interpolator
LOD
DistanceLOD
KeyNavigatorBehavior
MouseBehavior
PickMouseBehavior
ColorInterpolator
RotPosPathScaleInterpolator
MouseRotate
MouseTranslate
MouseZoom
PickRotateBehavior
PickTranslateBehavior
PickZoomBehavior

Transformation

- Similar to OpenGL, applied in reverse order (from leaf to root)

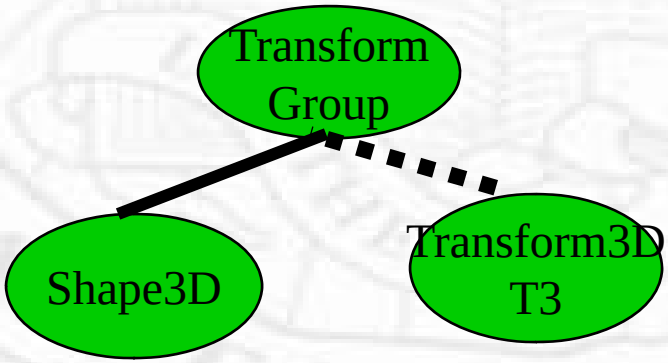


$$global = T1 \cdot T2 \cdot T3 \cdot local$$

Transform Example

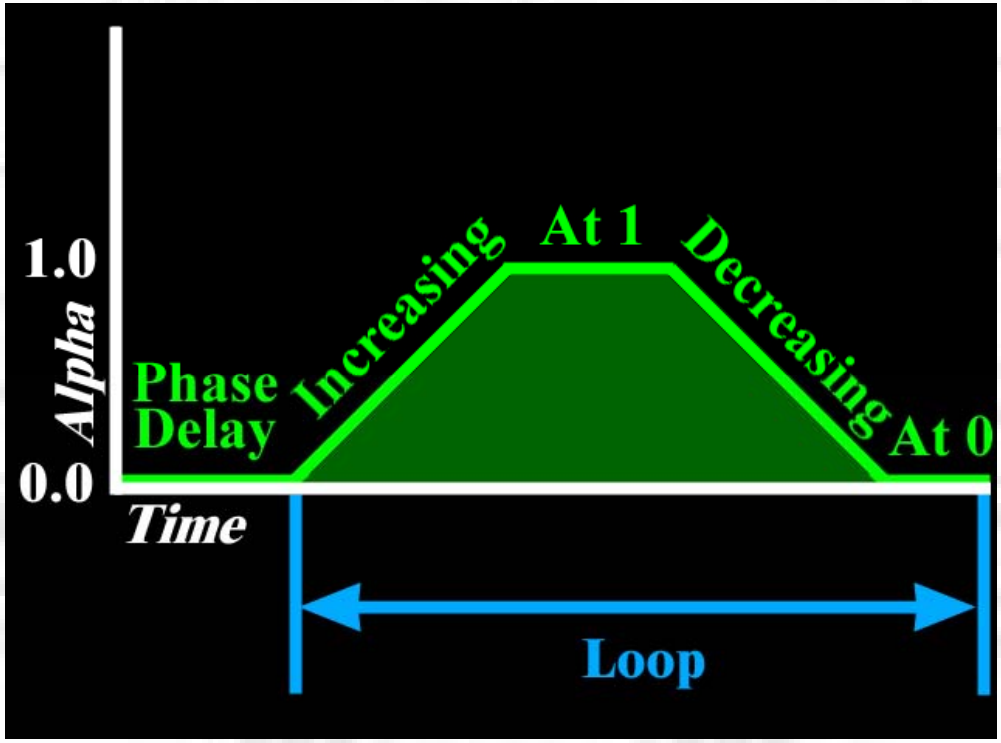
```
Shape3D myShape = new Shape3D (myGeom, myAppear);
Transform3D myTrans3D = new Transform3D();
myTrans3D.set (new Vector3D(1.0, 0.0, 0.0) );
TransformGroup = new TransformGroup();
myGroup.setTransform (myTrans3D);
myGroup.addChild (myShape);
```

Constant translation



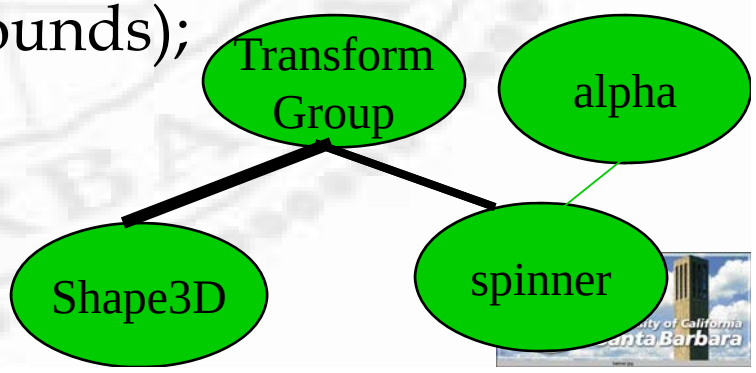
Animation

- *Canned* transformations
- Time to alpha
- Alpha to value (e.g., position, angle, color)



Behavior Example

```
TransformGroup myGroup = new TransformGroup();
Alpha upRamp = new Alpha();
upRamp.setIncreasingAlphaDuration(10000);
upRamp.setLoopCount(-1);
RotationInterpolator mySpinner = new
    RotationInterpolator(upRamp, myGroup);
mySprinner.setAxisofRotation(new Transform3D());
mySpinner.setMinimumAngle(0.0f);
mySpinner.setMaximumAngle((float)(Math.PI *2I));
mySpinner.setScheduleingBounds(bounds);
myGroup.addChild(spinner);
```



Customized Behavior – Hard Way

1. write (at least one) constructor
   store a reference to the object of change
2. override `public void initialization()`
   specify initial wakeup criteria (trigger)
3. override `public void processStimulus()`
   decode the trigger condition
   act according to the trigger condition
   reset trigger as appropriate

```
1.   import java.awt.event.*;
2.   import java.util.Enumeration;
3.
4.   //   SimpleBehaviorApp renders a single, rotated cube.
5.
6.   public class SimpleBehaviorApp extends Applet {
7.
8.      public class SimpleBehavior extends Behavior{
9.
10.        private TransformGroup targetTG;
11.        private Transform3D rotation = new Transform3D();
12.        private double angle = 0.0;
13.
14.        // create SimpleBehavior - set TG object of change
15.  ❶     SimpleBehavior(TransformGroup targetTG){
16.            this.targetTG = targetTG;
17.        }
18.
19.        // initialize the Behavior
20.        //      set initial wakeup condition
21.        //      called when behavior becomes live
22.  ❷     public void initialize(){
23.            // set initial wakeup condition
24.            this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
25.        }
26.
27.        // called by Java 3D when appropriate stimulus occurs
28.  ❸     public void processStimulus(Enumeration criteria){
29.            // do what is necessary in response to stimulus
30.            angle += 0.1;
31.            rotation.rotY(angle);
32.            targetTG.setTransform(rotation);
33.            this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
34.        }
35.
36.     } // end of class SimpleBehavior
```

Define behavior class

Provide constructor. Store reference
To a TransformGroup

Override initialization method

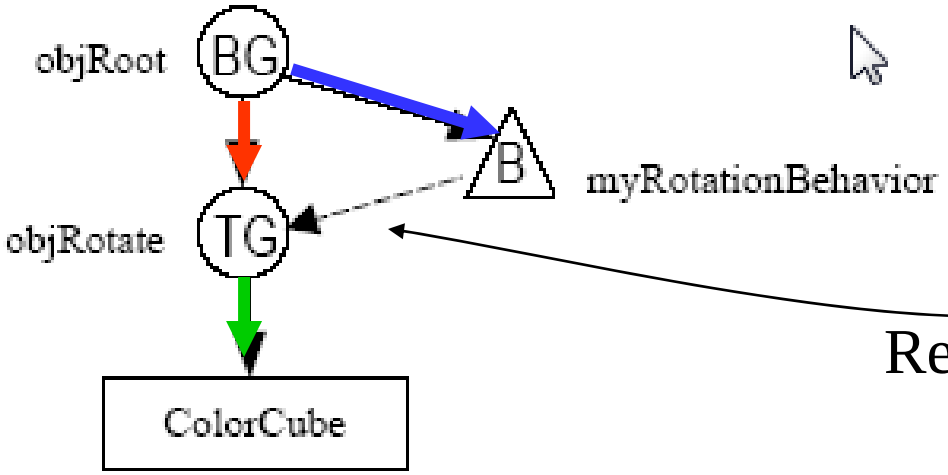
Process stimulus

Reset behavior for repetition

Customized Behavior – easy way

1. prepare the scene graph (by adding a TransformGroup or other necessary objects)
2. insert behavior object in the scene graph, referencing the object of change
3. specify a scheduling bounds (or SchedulingBoundingLeaf)
4. set write (and read) capabilities for the target object (as appropriate)

```
37. public BranchGroup createSceneGraph() {
38.     // Create the root of the branch graph
39.     BranchGroup objRoot = new BranchGroup();
40.
41.     TransformGroup objRotate = new TransformGroup();
42. ❹ objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
43. ❶
44.     objRoot.addChild(objRotate);
45.     objRotate.addChild(new ColorCube(0.4));
46.
47.     SimpleBehavior myRotationBehavior = new SimpleBehavior(objRotate);
48. ❸ myRotationBehavior.setSchedulingBounds(new BoundingSphere());
49. ❷ objRoot.addChild(myRotationBehavior);
50.
51.     // Let Java 3D perform optimizations on this scene graph.
52.     objRoot.compile();
53.
54.     return objRoot;
55. } // end of CreateSceneGraph method of SimpleBehaviorApp
```



Reference link

Canned Behavior

1. provide read and write capabilities for the target transform group
2. create a MouseBehavior object
3. set the target transform group
4. provide a bounds (or BoundingLeaf) for the MouseBehavior object
5. add the MouseBehavior object to the scene graph

```
1.  public class MouseRotateApp extends Applet {
2.
3.      public BranchGroup createSceneGraph() {
4.          // Create the root of the branch graph
5.          BranchGroup objRoot = new BranchGroup();
6.
7.          TransformGroup objRotate = new TransformGroup();
8.      ❶   objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
9.      ❶   objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
10.
11.         objRoot.addChild(objRotate);
12.         objRotate.addChild(new ColorCube(0.4));
13.
14.     ❷   MouseRotate myMouseRotate = new MouseRotate();
15.     ❸   myMouseRotate.setTransformGroup(objRotate);
16.     ❹   myMouseRotate.setSchedulingBounds(new BoundingSphere());
17.     ❺   objRoot.addChild(myMouseRotate);
18.
19.         // Let Java 3D perform optimizations on this scene graph.
20.         objRoot.compile();
21.
22.         return objRoot;
23.     } // end of CreateSceneGraph method of MouseRotateApp
```

Lights

- Javax.media.j3d.Light
  - javax.media.j3d.AmbientLight
  - javax.media.j3d.DirectionLight
  - javax.media.j3d.PointLight
    - javax.media.j3d.SpotLight
  - setEnable();
  - setColor();
  - setInfluenceBounds(); // must set to have any effect

Light - Example

```
DirectionalLight myLight = new DirectionalLight();
myLight.setEnable( true );
myLight.setColor( new Color3f(1.0f, 1.0f, 1.0f));
myLight.setDirection( new Vector3f (1.0f, 0.0f, 0.0f) );
BoundingSphere myBounds = new BoundingSphere (
    new Point3d(), 1000.0);
myLight.setInfluencingBounds( myBounds);
```

Light - Example

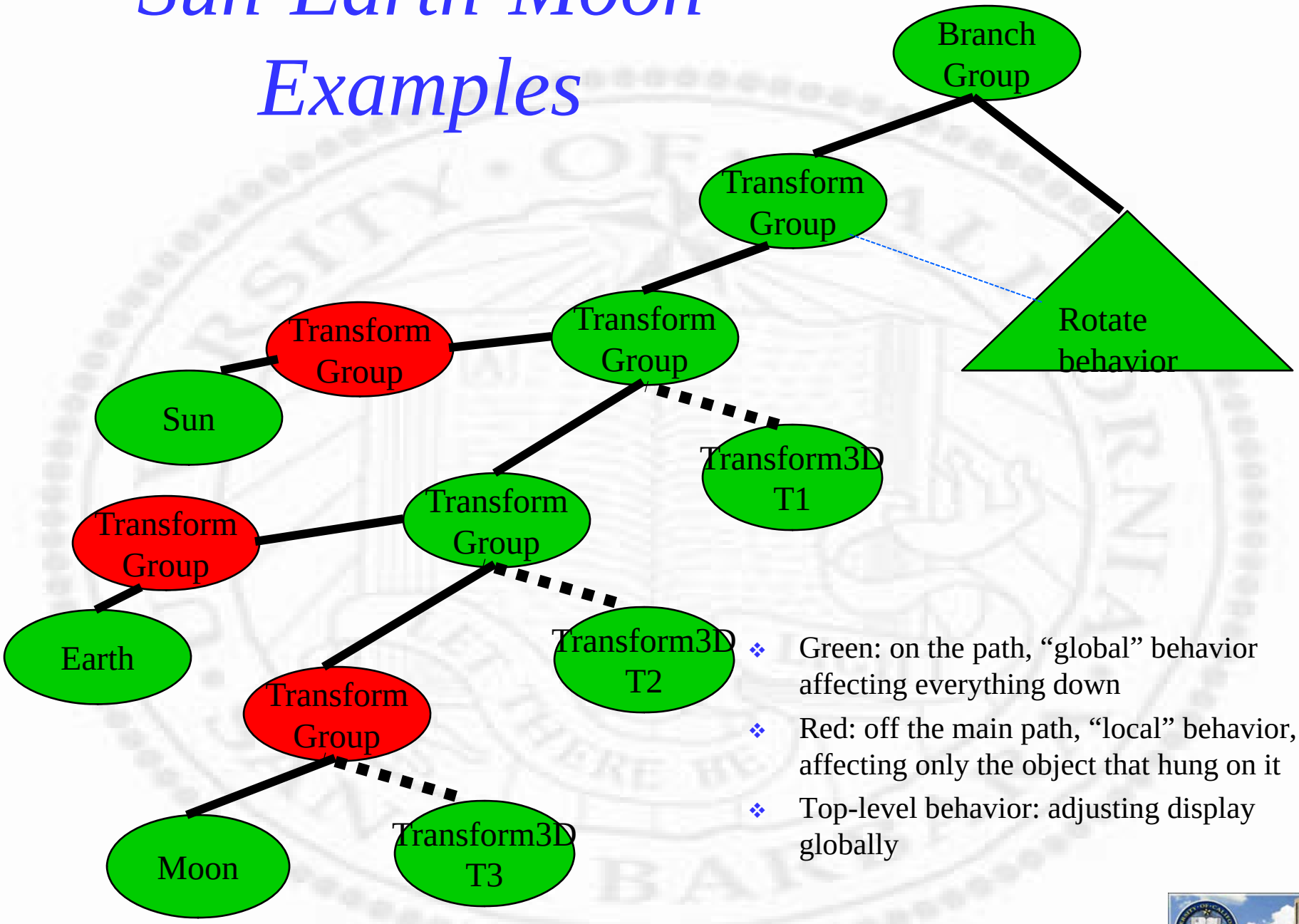
- Relative to light's coordinate system:
  - Light moves, so does the volume

```
PointLight myLight = new PointLight();
myLight.setInfluencingBounds (myBounds);
```

- Relative to a bounding leaf's coordinate system
  - Light moves, *not* the volume
  - If bounding leaf moves, so does the volume

```
TransformGroup myGroup = new TransformGroup();
BoundingLeaf  myLeaf = newBoundingLeaf (myBounds);
myGroup.addChild (myLeaf);
myLight.setInfluencingBounds(myLeaf);
```

Sun-Earth-Moon Examples



- Green: on the path, “global” behavior affecting everything down
- Red: off the main path, “local” behavior, affecting only the object that hung on it
- Top-level behavior: adjusting display globally