Computer Game Technologies

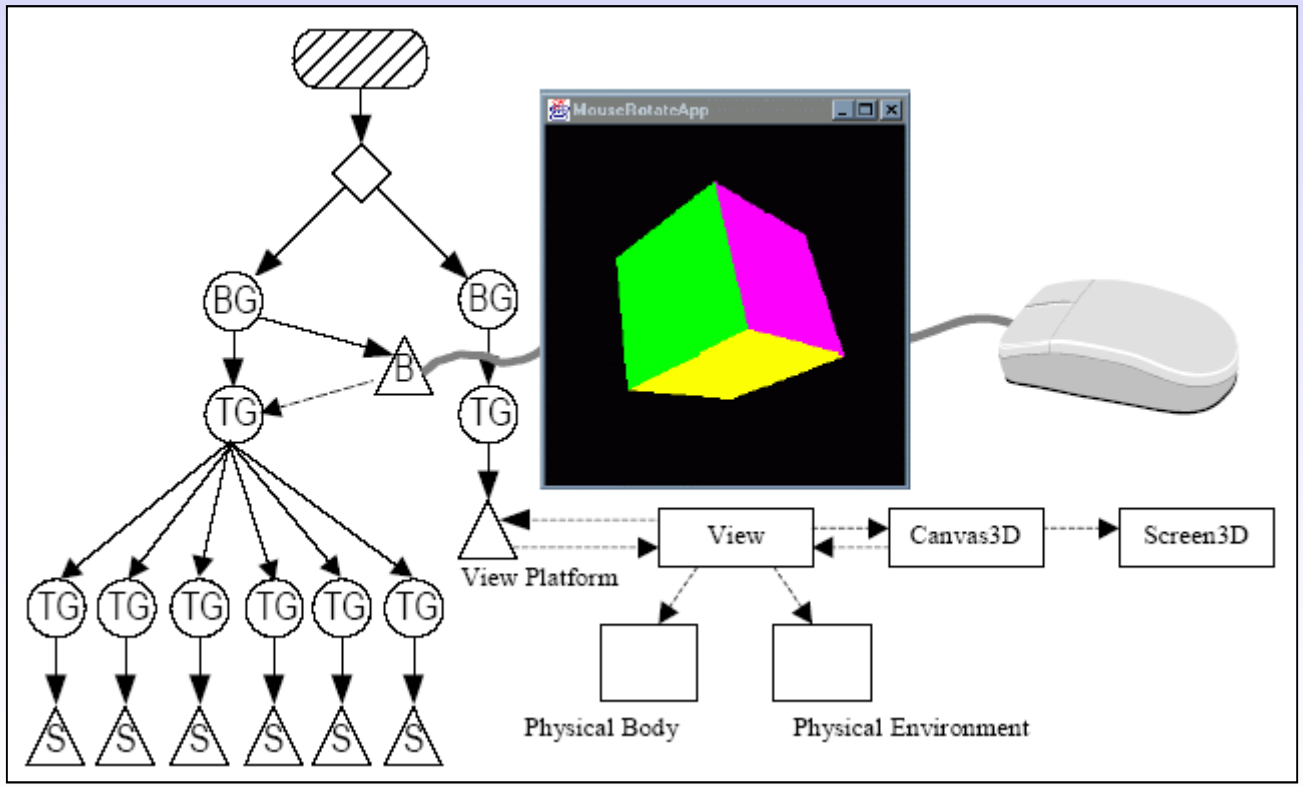
Interaction and Animation in Java 3D

Interaction and Animation in Java 3D

- *Interaction* is a change in the scene in response to user action
  - Key press
  - Mouse click or movement
- *Animation* is a change without any direct user action
  - Passage of time
  - Collisions
- Interaction and animation result in changes to the scene graph
- Implemented in Java 3D using *Behavior* objects

Behaviors in Java 3D

- *A behavior* is a link between a *stimulus* and an *action*



(Java Tutorial Chapt. 4)

Applications of Behaviors

Table 4-1 Applications of Behavior Categorized by Stimulus and Object of Change

stimulus (reason for change)	**object of change**			
	TransformGroup (visual objects change orientation or location)	Geometry (visual objects change shape or color)	Scene Graph (adding, removing, or switching objects)	View (change viewing location or direction)
user	interaction	application specific	application specific	navigation
collisions	visual objects change orientation or location	visual objects change appearance in collision	visual objects disappear in collision	View changes with collision
time	animation	animation	animation	animation
View location	billboard	level of detail (LOD)	application specific	application specific

(Java Tutorial Chapt. 4)

Inbuilt Behaviors

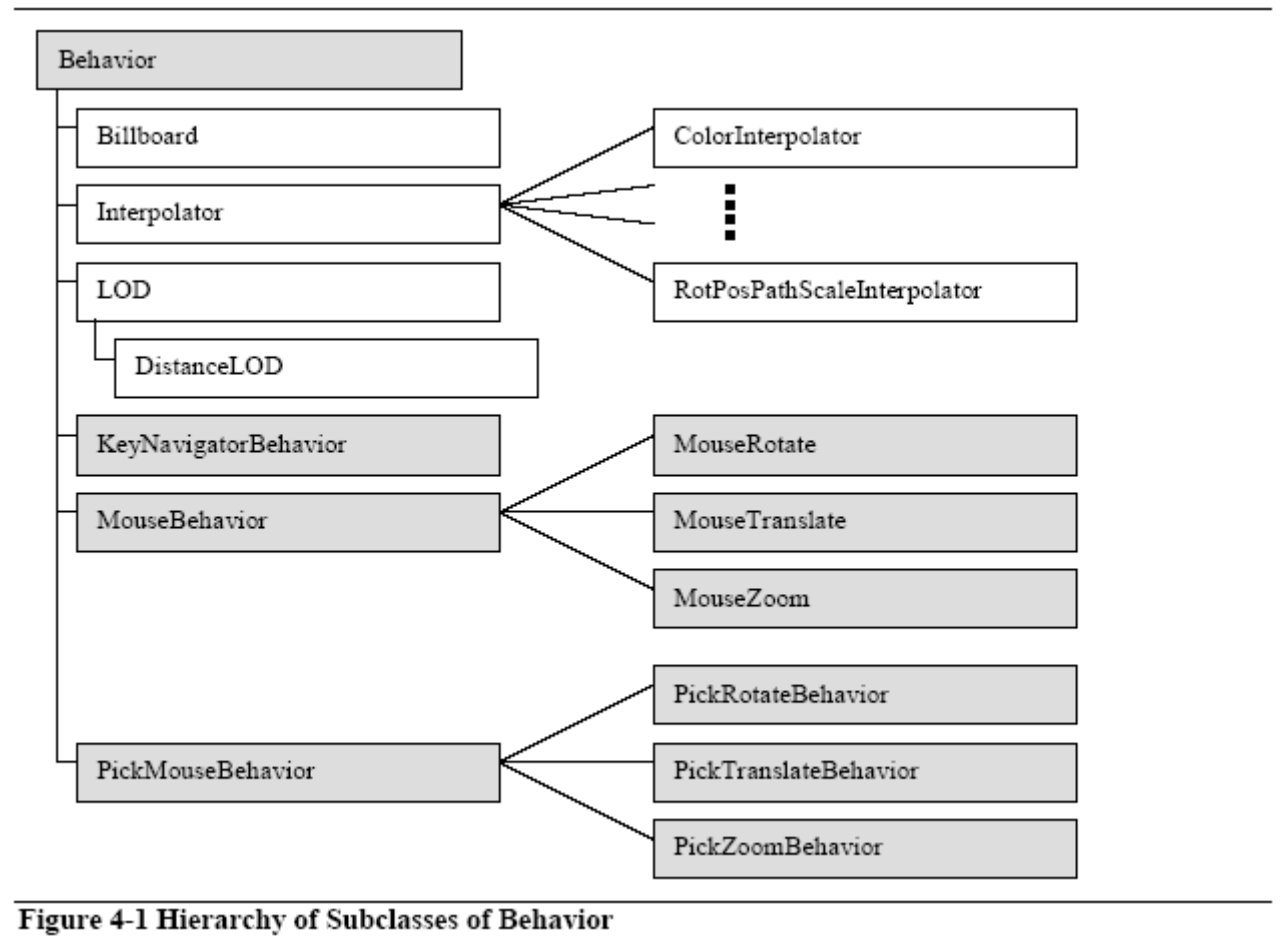


Figure 4-1 Hierarchy of Subclasses of Behavior

Writing a Behavior Class

- Behavior is an abstract class
- Custom Behavior class implements
  - The class constructor
  - initialize() method
  - processStimulus() method
- Behavior typically has an *object of change*
  - E.g. visible object; view platform
  - A reference to the object of change is usually set up in the Behavior constructor

Writing a Behavior Class (2)

- initialize() method
  - Executed when behavior first goes "live" (ie when added to scene graph)
  - Sets initial *trigger* event for the behavior
  - Trigger is a (combination of) WakeupCondition object
  - Also sets initial values of any behavior state variables
- processStimulus() method
  - Invoked when the trigger event occurs
    - E.g. key press; passage of time
  - Carries out action in response to the event
    - Manipulates object of change
    - Usually results in a change to the scene graph
  - Resets the trigger (if necessary)

Example Custom Behavior

- Make object rotate in response to a key press
  - Needs reference to a Transform Group for the object
  - The angle of rotation
- When any key is pressed
  - The angle of rotation is incremented
  - New rotation transformation around the Y axis is created
  - Rotation is added to the Transform Group
- No information about the object itself is required as only the transform group is changed

Custom Behavior Code

```
public class SimpleBehavior extends Behavior{

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

  // create SimpleBehavior - set TG object of change
  SimpleBehavior(TransformGroup targetTG){
    this.targetTG = targetTG;
  }

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

Custom Behavior Code (2)

```
  // called by Java 3D when appropriate stimulus occurs
  public void processStimulus(Enumeration criteria){
    // do what is necessary in response to stimulus

    angle += 0.1;
    rotation.rotY(angle);
    targetTG.setTransform(rotation);

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

} // end of class SimpleBehavior
```

Using a Behavior

1. Add transform group to scene graph above the object of change
2. Insert Behavior object into scene graph, with reference to the object of change
   - via the transform group
3. Specify the scheduling bounds for the behavior
   - Behavior only active when current view intersects the scheduling bounds
   - Behavior is never active if bounds not specified
4. Set write (and maybe read) capabilities on the transform group
   - So it can be changed at run time

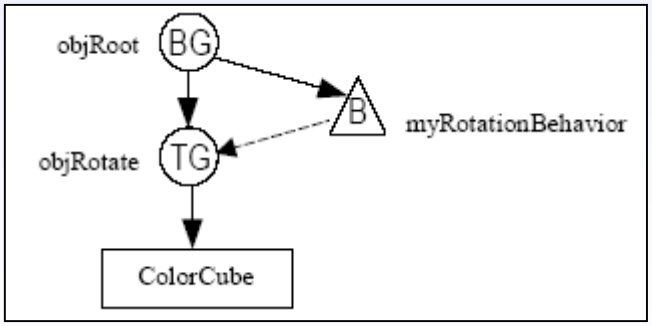
Example Using Behavior Code

```
public BranchGroup createSceneGraph() {
  BranchGroup objRoot = new BranchGroup();

  TransformGroup objRotate = new TransformGroup();
  objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

  objRoot.addChild(objRotate);
  objRotate.addChild(new ColorCube(0.4));

  SimpleBehavior myRotationBehavior = new
                              SimpleBehavior(objRotate);
  myRotationBehavior.setSchedulingBounds(new BoundingSphere());
  objRoot.addChild(myRotationBehavior);

  objRoot.compile();

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

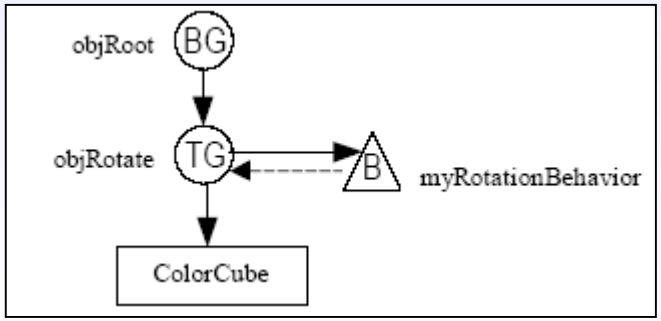
Adding Behavior to Scene Graph

- A behavior can go anywhere in the scene graph
- Possible considerations are:
  - Code maintenance
  - Effect on scheduling bounds

A. Scheduling bounds independent of cube location



B. Scheduling bounds moves with the cube



Wakeup Conditions

- Behaviors are triggered by the occurrence of one (or more) specified wakeup conditions

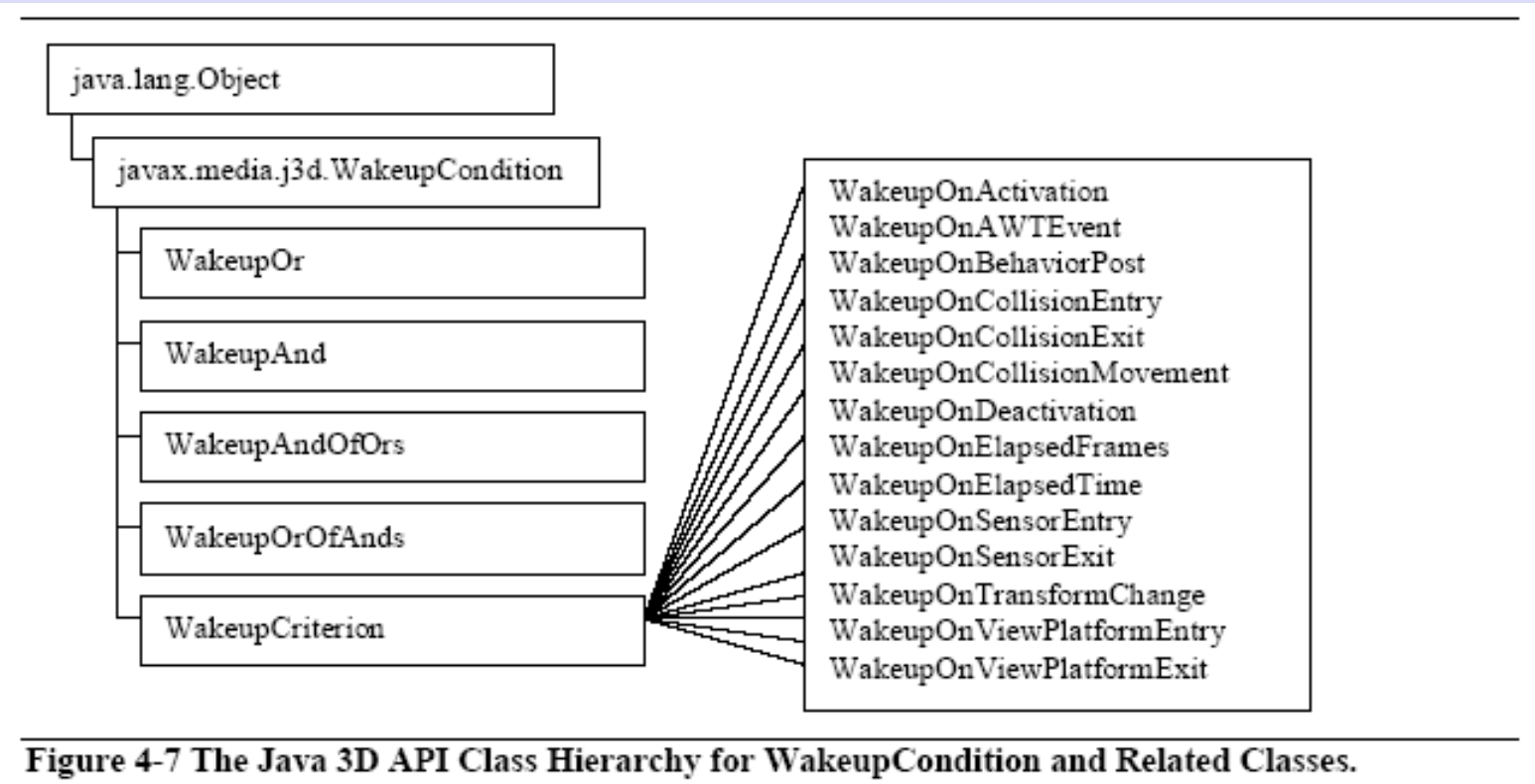


Figure 4-7 The Java 3D API Class Hierarchy for WakeupCondition and Related Classes.

Specific Wakeup Conditions

- WakeupOnAWTEvent
  - Triggered when an AWT event occurs
  - e.g. key pressed, key released, mouse clicked, mouse dragged
  - processStimulus() method of behavior may need to decode the event to find if the trigger is appropriate

Checking for Key Presses

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

   while( criteria.hasMoreElements() ) {
     wakeup = (WakeupCriterion) criteria.nextElement();
     if( wakeup instanceof WakeupOnAWTEvent ) {
       event = ((WakeupOnAWTEvent)wakeup).getAWTEvent();
       for( int i = 0; i < event.length; i++ ) {
         if( event[i].getID() == KeyEvent.KEY_PRESSED )
           processKeyEvent((KeyEvent)event[i]);
       }
     }
   }
   wakeupOn( keyPress );
 } // end of processStimulus()
```

Specific Wakeup Conditions (2)

- WakeupOnElapsedTime
  - Wakeup when a specific number of milliseconds has elapsed (timing not guaranteed to be entirely accurate)
- WakeuponElapsedFrames
  - Wakeup when a number of frames have elapsed
- These are useful conditions for creating animations
  - Movement of sprites

Specific Wakeup Conditions (3)

- WakeupOnCollisionEntry
  - Wakeup when object of change first collides with any other object in the scene graph
- WakeuponCollisionExit
  - Wakeup when object first no longer collides with another object
- Not as useful for collision detection as they might seem
  - Collisions detected only after they have occurred
  - Will look strange if a ball bouncing off a wall first enters the wall!
  - Need to predict collisions in many cases

Inbuilt Behavior Utility Classes

- KeyNavigatorBehavior
  - Provides precoded object movement actions in response to certain key presses
  - Pass it the Transform Group of the object to be controlled

Key	MOVEMENT	Alt-key movement
←	rotate left	lateral translate left
→	rotate right	lateral translate right
↑	move forward	
↓	move backward	
PgUp	rotate up	translation up
PgDn	rotate down	translation down
+	restore back clip distance (and return to the origin)	
-	reduce back clip distance	
=	return to center of universe	

Example: Viewer Navigation

```
public BranchGroup createSceneGraph(SimpleUniverse su) {
// Create the root of the branch graph
  TransformGroup vpTrans = null;
  Vector3f translate = new Vector3f();
  Transform3D T3D = new Transform3D();
  BranchGroup objRoot = new BranchGroup();
  objRoot.addChild(createLand()); // create other content

  vpTrans = su.getViewingPlatform().getViewPlatformTransform();
  translate.set( 0.0f, 0.3f, 0.0f); // 3 meter elevation
  T3D.setTranslation(translate); // set as translation
  vpTrans.setTransform(T3D); // used for initial position
  KeyNavigatorBehavior keyNavBeh =
        new KeyNavigatorBehavior(vpTrans);
  keyNavBeh.setSchedulingBounds(new BoundingSphere(
        new Point3d(),1000.0));
  objRoot.addChild(keyNavBeh);

  objRoot.compile();
  return objRoot;
} // end
```

Behavior Utility Classes (2)

- Mouse interaction
  - Translating, zooming and rotating visual objects
  - Again, provide transform group of visual object to behavior

Table 4-4 Summary of Specific MouseBehavior Classes

MouseBehavior class	Action in Response to Mouse Action	Mouse Action
MouseRotate	rotate visual object in place	left-button held with mouse movement
MouseTranslate	translate the visual object in a plane parallel to the image plate	right-button held with mouse movement
MouseZoom	translate the visual object in a plane orthogonal to the image plate	middle-button held with mouse movement

Example: Object Rotation

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

  TransformGroup objRotate = new TransformGroup();
  objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
  objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);

  objRoot.addChild(objRotate);
  objRotate.addChild(new ColorCube(0.4));

  MouseRotate myMouseRotate = new MouseRotate();
  myMouseRotate.setTransformGroup(objRotate);
  myMouseRotate.setSchedulingBounds(new BoundingSphere());
  objRoot.addChild(myMouseRotate);

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

Behavior Utility Classes (3)

- Picking
  - Selecting a visual object with a mouse click
  - Selection with a "pick ray"

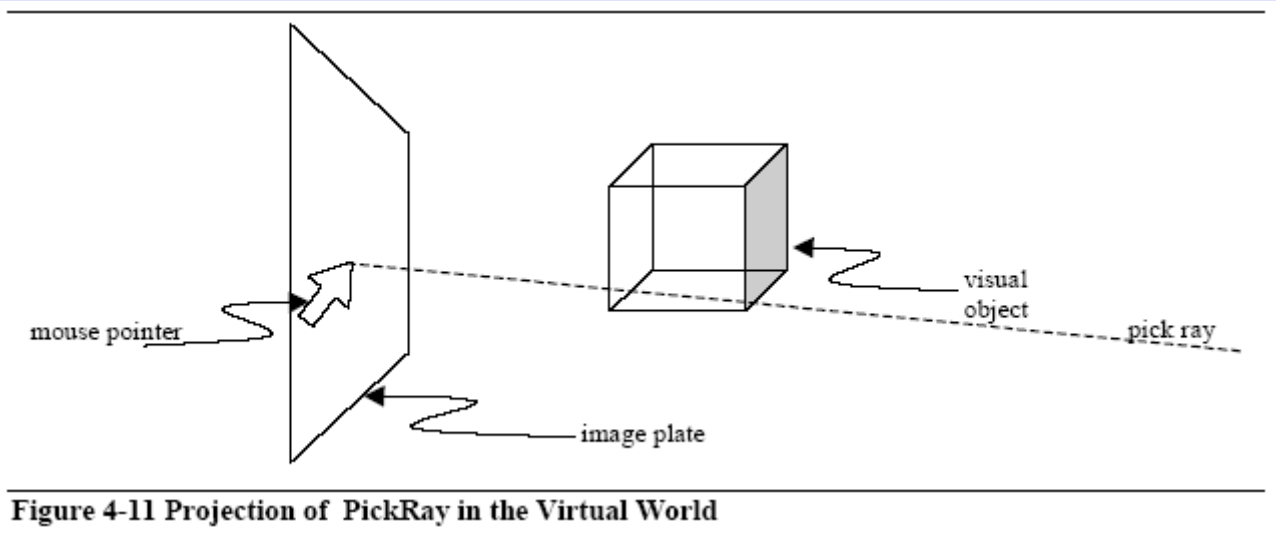


Figure 4-11 Projection of PickRay in the Virtual World

The End