Behaviours

- Java 3D uses the `Behavior` class to facilitate interaction and animation
- This class, and its descendants, are links to user code which can change the graphics and sounds of the virtual universe
- The purpose of a `Behavior` object in a scene graph is to change the scene graph, or objects in the scene graph, in response to some stimulus

Capabilities and Scheduling Bounds

- Scene graph objects possess capability bits which permit modifications to be made to them after they become *live* (are added to the scene graph)
  - E.g. ALLOW_TRANSFORM_WRITE
- Nearly all of the behaviours we shall meet will require capability bits to be set or they will have no effect whatsoever
- Behaviours can also be excessively demanding of processor time unless kept in check
  - Scheduling bounds are used to limit their operation to parts of the scene graph

Behaviour Stimuli and Effects

- A stimulus can be the press of a key, a mouse movement, the collision of objects, the passage of time, some other event, or combinations of them
- Changes produced include adding objects to the scene graph, removing objects from the scene graph, changing attributes of objects in the scene graph, rearranging objects in the scene graph, or combinations of these

Behaviour Applications

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

NB *Picking* (not listed above) is also implemented using behaviours

Interactions

- It is important to realise that interaction in Java 3D (via behaviours) is very different to interaction in 2D (via awt)
- The `Behavior` abstract class has two abstract methods
  - `initialize()`
  - `processStimulus()`
- All user-defined classes derived from `Behavior` must provide implementations of these two methods

Initialize() & ProcessStimulus()

- Initialize()
  - Called when a `Behavior` object is created
  - Defines the type(s) of event that trigger the `Behavior`
  - Triggers are specified with `wakeup` classes
- ProcessStimulus()
  - Called when an event of the type(s) to which the `Behavior` responds occurs
  - Contains the code to be executed when the event(s) occur

wakeupOn() Method

- In order for a behaviour to be triggered wakeup criteria must be specified for it

  `b.wakeupOn(WakeupCondition criteria)`

  Defines the wakeup criteria for behaviour `b`

- **Both** initialize() **and** processStimulus() will normally need to call this method

Wakeup Conditions

- Active behaviours are triggered by one or more wakeup stimuli
- The wakeup stimuli for a behaviour are specified via the abstract class `WakeupCondition`
- Five classes extend `WakeupCondition`
  - The abstract class `WakeupCriterion`
  - Four utility classes which allow multiple wakeup criteria to be combined into one wakeup condition

    `WakeupOr` `WakeupOrOfAnds`
    `WakeupAnd` `WakeupAndOfOrs`

Wakeup Condition Methods

The `WakeupCondition` class has two methods -

`Enumeration allElements()`

Returns an enumeration of all `WakeupCriterion` objects in this condition

`Enumeration triggeredElements()`

Returns an enumeration of all triggered `WakeupCriterion` objects in this condition

Wakeup Criterion

- The `WakeupCriterion` abstract class supports 14 specific wakeup criterion classes (see next slide)
- It provides one (rarely needed) method -

`boolean hasTriggered()`

Returns true if this criterion triggered the wakeup

Wakeup Criteria

Wakeup Criterion	Trigger
WakeupOnActivation	on first detection of a ViewPlatform's activation volume intersecting with this object's scheduling region.
WakeupOnAWTEvent	when a specific AWT event occurs
WakeupOnBehaviorPost	when a specific behavior object posts a specific event
WakeupOnCollisionEntry	on the first detection of the specified object colliding with any other object in the scene graph
WakeupOnCollisionExit	when the specified object no longer collides with any other object in the scene graph
WakeupOnCollisionMovement	when the specified object moves while in collision with any other object in the scene graph
WakeupOnDeactivation	when a ViewPlatform's activation volume no longer intersects with this object's scheduling region
WakeupOnElapsedFrames	when a specific number of frames have elapsed
WakeupOnElapsedTime	when a specific number of milliseconds have elapsed
WakeupOnSensorEntry	on first detection of any sensor intersecting the specified boundary
WakeupOnSensorExit	when a sensor previously intersecting the specified boundary no longer intersects the specified boundary
WakeupOnTransformChange	when the transform within a specified TransformGroup changes
WakeupOnViewPlatformEntry	on first detection of a ViewPlatform activation volume intersecting with the specified boundary
WakeupOnViewPlatformExit	when a View activation volume no longer intersects the specified boundary

Special Wakeup Criteria Triggers

- Some WakeupCriterion classes trigger on first detection
  - These criteria will trigger only once for the event
  - A WakeupOnActivation object will trigger only upon first detection of the intersection of a ViewPlatform activation volume with the scheduling region of the associated behavior
  - The WakeupCondition will not trigger again until Java 3D has detected that the volumes ceased to intersect at some point and have just started to intersect again
- Some WakeupCriterion classes form matched pairs
  - Entry/Exit or Activation/Deactivation
  - These criteria only trigger in strict alternation beginning with the Entry or Activation criterion

Key Navigator Behaviours

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	

`KeyNavigatorBehavior`

- This class invokes `KeyNavigator` to modify the view platform transform
- Needs

```
import com.sun.j3d.utils.behaviors.keyboard
```

- Constructor

`KeyNavigatorBehavior(TransformGroup targetTG)`

Constructs a new key navigator behavior node that operates on the specified transform group

Mouse Behaviours

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

Mouse Behaviour Utilities

- The 3 specific mouse behaviour classes
  - MouseRotate
  - MouseTranslate
  - MouseZoom

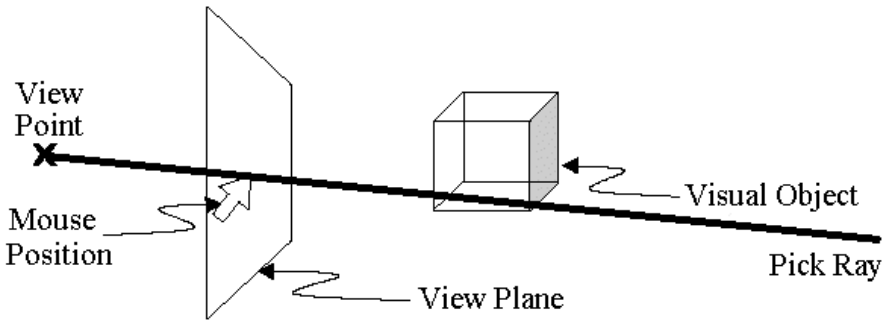
  are extensions of the `MouseBehavior` abstract class
- They implement the `MouseCallback` interface
- They need

```
import com.sun.j3d.utils.behaviors.mouse
```

Picking Behaviours

- Interactively selecting, or *picking*, visual objects is normally achieved with a mouse
  - Note that it doesn't have to be
- Picking is implemented by a `Behavior`
  - Three utility classes are provided by Java 3D
    - PickTranslateBehavior, PickRotateBehavior, PickZoomBehavior
- The user places the mouse pointer over the visual object and presses a mouse button
  - The `Behavior` is triggered by the button press and begins the picking operation

Mouse Picking Operation



- A *pick ray* is projected into the virtual world from the view point through the mouse position on the view plane
- The visual objects intersected by the ray are determined
- By default the visual object nearest to the viewer is selected
  - Other selections exist such as forming an array of all objects intersected

Mouse Picking Utility Classes

- The mouse picking utility classes set up some commonly desired functions and require three parameters
  - Branch Group - only visual objects in this branch can be selected
  - Canvas
  - Bounds - only visual objects which intersect these bounds can be selected
- Need `import com.sun.j3d.utils.picking.behaviors.*;`

`PickTranslateBehavior(branch,canvas,bounds);`

Holding right button down and dragging translates selected visual object

`PickRotateBehavior(branch,canvas,bounds);`

Holding left button down and dragging rotates selected visual object

`PickZoomBehavior(branch,canvas,bounds);`

Holding middle button down and dragging zooms in on selected visual object

Mouse Pick Behavior Capabilities

- Checking for intersections between a pick ray and many complex visual objects could become computationally prohibitive
- By default no *internal* nodes (branch groups or transform groups) are pickable
  - But leaf nodes, such as Shape3D nodes, are
- To make an internal node pickable use - `internalNode.setCapability(Node.ENABLE_PICK_REPORTING);`
- To change the “pickability” of a leaf node use - `leafNode.setPickable(false);leafNode.setPickable(true);`

Further Mouse Pick Behaviors

- So far we have used a line (the `PickRay`) to select visual objects
  - We can use a more complicated `PickShape` than this

```
PickRay,           PickSegment,
PickConeRay,       PickConeSegment,
PickCylinderRay,   PickCylinderSegment,
PickPoint,         PickBounds
```

- Nor do we have to settle for the default of selecting the visual object nearest the viewpoint
  - The `PickTool` class provides a number of alternative methods

```
PickAll(),PickAllSorted(),PickAny(),PickClosest()
```

Choosing a PickShape

application	geometry		
	polygons appear large	polygons appear small	points and/or lines
general	PickRay	PickRay, PickBounds, PickCone, PickCylinder	PickBounds, PickCone, PickCylinder
accuracy	PickRay, PickSegment, PickPoint	PickRay, PickSegment, PickPoint,	PickRay, PickSegment, PickPoint
speed	PickRay	PickRay	PickRay

- A PickRay was an infinitely long line segment projecting from the view point through the entire virtual universe
- A PickSegment is finite and only projects a fixed distance into the virtual universe

PickTool Methods

- PickTool permits an increased level of precision
- It has two modes -
  `pTool.setMode(PickTool.BOUNDS);`
  Visual objects selected if picking shape intersects their volume
  `pTool.setMode(PickTool.GEOMETRY);`
  Visual objects selected only if picking shape intersects a rendered part of them - E.g. an edge on a wire-frame display
  [Needs `iNode.setCapability(Geometry.ALLOW_INTERSECT);`]
- The various PickTool methods return their selections as `PickResult` objects

PickTool Usage

```
// Create a line segment between 2 endpoints
PickSegment pSegment = new PickSegment(p[0],p[1]);
// Create a pick tool whose domain is pBranch
PickTool pTool = new PickTool(pBranch);
// Set the geometry mode for precision
pTool.setMode(PickTool.GEOMETRY);
// Set the pick segment for the shape
pTool.setShape(pSegment, p[0]);
// Obtain all the picked objects sorted by
// distance from the view point
PickResult [] pResults = pTool.pickAllSorted();
```

Now,

```
(Shape3D)pResults[i].getObject()
```

will yield the visual objects as Shape3D objects

Billboards

- Natural things, such as trees, take a tremendous amount of geometry to represent accurately
- The *billboard* approach uses textured polygons instead of the detailed geometry
- Behaviours can be used to automatically orientate the textured polygon orthogonal to the viewer such that only the front textured face is viewed
- This orienting behaviour is called *billboard behaviour*

Level of Detail (LOD)

- *LOD* represents visually complex objects with multiple visual objects of varying levels of detail
- The visual object representation with the least detail is used when the viewer is far away and the most detailed representation is used when the viewer is close
- The LOD behaviour automatically switches between the representations based on the distance of the objects from the viewer