Locators and Context

Module 9

Autodesk

Autodesk Maya Python API Training

Kristine Middlemiss, *Senior Developer Consultant*
Autodesk Developer Network (ADN)

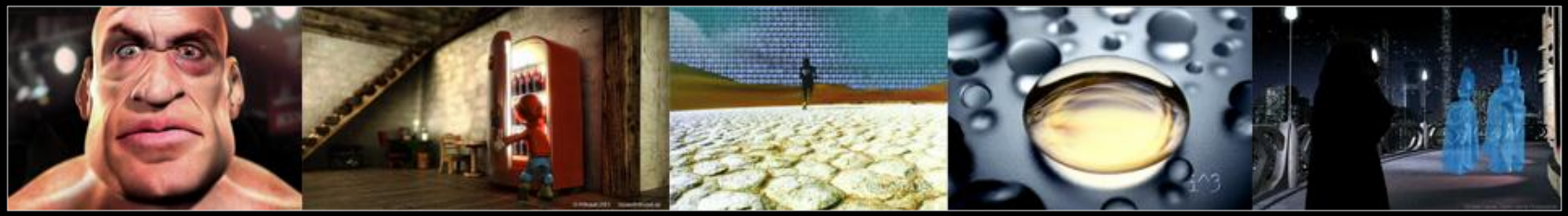
Autodesk

Homework Review:

UI Tools

Locators

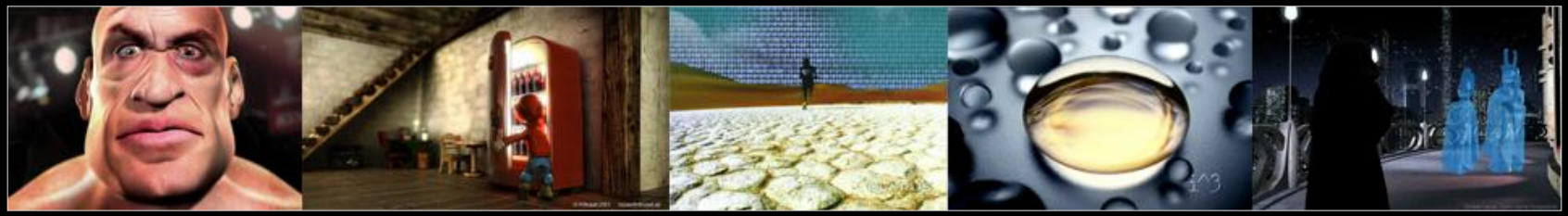
Context / Tools



Autodesk®

What is a Locator?

A Maya locator is a small icon like an x-y-z axis that marks a point in space.



MPxLocatorNode

A class that allows users to draw 3D graphical elements in the Maya scene, which can be manipulated using Maya standard manipulators.

MPxLocatorNode Support

- Drawing
- Selection

Only in Maya viewport, NOT renderable

MPxLocatorNode

Override the following methods:

- draw()
- IsBounded()
- boundingBox()

Methods are called during

- Refresh
- Selection

MPxLocatorNode::draw()

- void MPxLocatorNode:: draw (M3dView & view, const MDagPath & path, M3dView::DisplayStyle style, M3dView::DisplayStatus status)

Two usage of this function:

  - draw the node in current viewport
  - Maya determine whether the node is selected

- It should leave OpenGL in exactly the same state it was before this function is called. (use glPushAttrib())

- Draw in local object space

MPxLocatorNode

Example Class Declaration:

```
class arrowLocator(OpenMayaMPx.MPxLocatorNode):
        def __init__(self):
                OpenMayaMPx.MPxLocatorNode.__init__(self)

        def compute(self, plug, data):
                return OpenMaya.kUnknownParameter

        def draw(self, view, path, style, status):
                OpenMayaMPx.MPxManipContainer.draw(self, view, path, style, status)

        def isBounded(self):
                return True

        def boundingBox(self):
                # code ...

def arrowLocator_creator():
        return OpenMayaMPx.asMPxPtr( arrowLocator() )

def arrowLocator_initialize():
        # code ...
```

OpenGL Basics

OpenGL: Open Graphics Library, developed by SGI(Silicon Graphics Inc.) in 1992

A standard specification defining an API for writing applications that produce 2D and 3D computer graphics.

Widely used in video games, virtual reality, CAD, etc…

OpenGL Basics: Draw Primitives

```
glColor3f(1.0, 1.0, 1.0);
```

```
glColor3i (1, 1, 1);
glColor3d (1.0, 1.0, 1.0);
```

Multiple commands for the same operation with different types of arguments

State machine

```
glBegin(GL_POLYGON);

    glVertex3f(0.0, 0.0, 1.0);
    glVertex3f(0.0, 3.0, 1.0);
    glVertex3f(3.0, 3.0, 2.0);
    glVertex3f(4.0, 1.5, 2.0);
    glVertex3f(3.0, 0.0, 0.0);

glEnd();
```

Draw calls in between

OpenGL Basics

- State Machine

- Stack-based
  - Matrix Mode: GL_MODELVIEW, GL_PROJECTION

```
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
…..
glPopMatrix();
```

  - State Attribute

```
glPushAttrib(GL_CURRENT_BIT)
…….
glPopAttrib()
```

- Double Buffer Drawing
  - swapBuffers

MPxLocatorNode::draw()

- Store any OpenGL drawing code, call M3dView::beginGL() and push the current OpenGL state:

```
view.beginGL();
glPushAttrib(GL_CURRENT_BIT);
```

- Call OpenGL drawing code, e.g. glVertex3f, etc… Can create variations depending on the drawing style and node status (which are passed into the method)

- Once drawing is finished, pop the previous OpenGL state and call M3dView::endGL():

```
glPopAttrib();
view.endGL();
```

MPxLocatorNode::boundingBox()

- If boundingBox() is overridden, isBounded() must return true

- Highly recommended to implement these. Without them, Maya will have difficulty determining the exact size of the locator, e.g. “Frame All” and “Frame Selected” will not zoom correctly.

- boundingBox() must be efficient since it is called each time the window refreshes.

MPxLocatorNode Registration

Specify kLocatorNode when registering the node in initializePlugin

```
plugin.registerNode("myLocator", myLocator::id,  myLocator_creator,  myLocator_initialize,
OpenMayaMPx.MPxNode.kLocatorNode )
```

Maya Context (Tools)

- Contexts defines what happens when interactive events occur within an interactive panel in Maya.

- Examples:
  - moveTool
  - CV CurveTool

- Context Information:
  - currentCtx command
  - contextInfo command

Features of “CV CurveTool”

- Menu button clicked:
  - mouse turned into '+' cursor, entering context
  - help line display help text
- Mouse pressed: draw a point in 3d space
- Hit “Enter”:
  - a CV curve is created
  - command executed in script editor
- Ctrl-Z and Z: undo/redo
- Tool property sheet

Custom Context

- Core classes:
  - MPxContext, MPxSelectionContext
  - MPxContextCommand
  - MPxToolCommand
- Related classes: MEvent, MCursor

MPxContext

Initialize and Cleanup:

virtual void MPxContext::toolOnSetup(MEvent & event)

virtual void MPxContext::toolOffCleanup(Mevent & event)

Help String and Graphical information:

MStatus MPxContext::setTitleString (const MString &str)

MStatus MPxContext::setHelpString (const MString &str)

MStatus MPxContext::helpStateHasChanged(MEvent & event)

MStatus MPxContext::setCursor (const MCursor & newCursor)

…

Custom Context Event:

Mouse Event:

- virtual Mstatus MPxContext::doPress(MEvent & event)
- Virtual Mstatus MPxContext::doRelease (MEvent &event)
- virtual Mstatus MPxContext::doDrag(MEvent &event)
- virtual Mstatus MPxContext::doHold(MEvent &event)
- virtual Mstatus MPxContext::doEnterRegion (MEvent &event)

Keyboard Event:

- void MPxContext::deleteAction( )
- void MPxContext::completeAction( )
- void MPxContext::abortAction( )

MEvent

System Event information:

- MEvent::getPosition()

Modifiers:

- shift key
- control key
- second mouse event

MEvent::isModifierMiddleMouseButton()

MEvent::isModifierLeftMouseButton()

MCursor

MPxContext::setCursor (const Mcursor & newCursor)

defaultCursor : Maya default cursor, the left arrow

crossHairCursor : '+' cursor

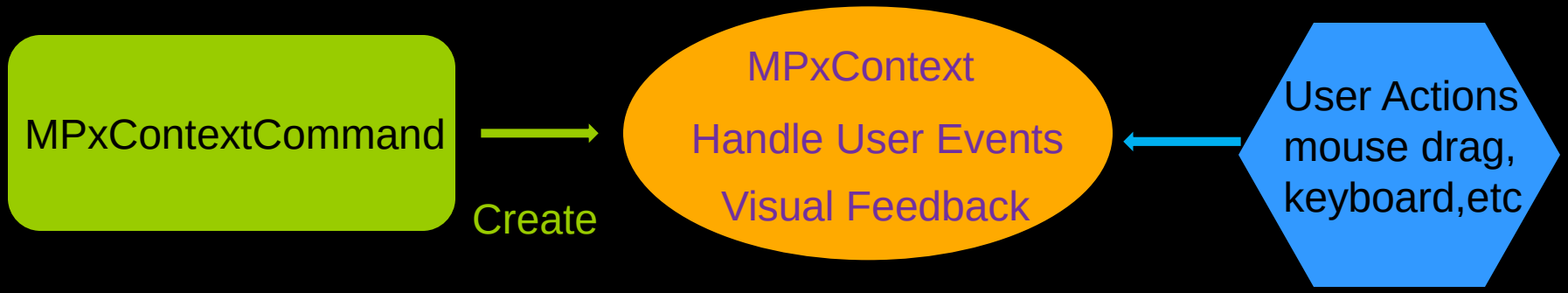
doubleCrossHairCursor : '+' cursor with double lines

editCursor : Wedge-shaped arrow pointing left

pencilCursor : Pencil shape

handCursor : Open hand shaped cursor

MPxContextCommand



MPxContextCommand

- Create instances of user context
- Add custom syntax (flags and arguments)
- Edit and query context properties
- Not derived from MPxCommand
- Not Undoable

MPxContextCommand

Register Context :

```
def initializePlugin( mobject ):
    mplugin = OpenMayaMPx.MFnPlugin(mobject)
     try:
      mplugin.registerContextCommand( "simpleMarqueeContext",
            simpleMarqueeContextCmd_creator )
    except:
      raise
}
```

Example: simpleMarqueeTool

Class Declaration:

```
class simpleMarqueeContextCmd (OpenMayaMPx.MPxContextCommand):
        def __init__(self):
          OpenMayaMPx.MPxContextCommand.__init__ (self)
        def  makeObj(self):
          # code …
```

Create Context :

```
def makeObj(self):
    return OpenMayaMPx.asMPxPtr( simpleMarqueeContext() )
```

Example: simpleMarqueeTool

```
def simpleMarqueeContext (OpenMayaMPx.MPxContext):
    def __init__(self):
        OpenMayaMPx.MPxContext.__init__(self)
        self.__view = OpenMayaUI.M3dView()
        self.__start_x = 0
        self.__start_y = 0
        self.__last_y = 0
        self.__last_y = 0
        self.__listAdjustment = None

    def toolOnSetup(self, event):
    def doPress(self, event):
    def doDrag(self, event):
    def doRelease(self, event):
    def doEnterRegion(self, event):
```

Example: simpleMarqueeTool

Code Structure:

```
def doPress(event):
    #check if modifier keys are pressed
    #record event position as start position of the marquee

def doDrag(event):
    #retrieve current event position
    #draw visual feedback  in real time in the viewport

def doRelease(event):
    #retrieve current event position as end position of marquee
    #select objects within the range defined by the marquee
```

Activate Custom Context

setToolTo Command:

```
$myContext = `simpleMarqueeContext`;
setToolTo $myContext;
```

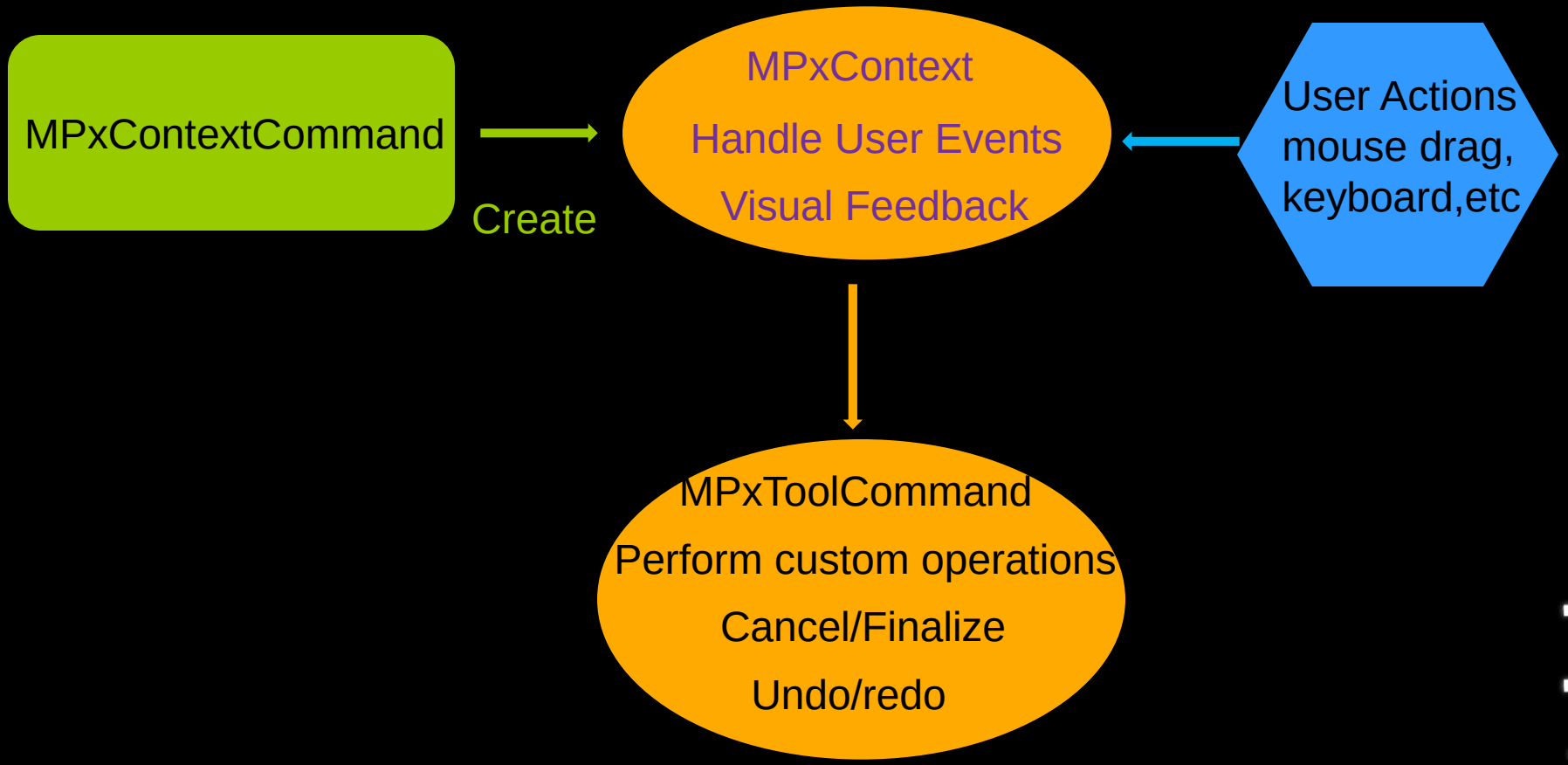
Add to Maya tool Shelf:

```
simpleMarqueeContext simpleMarqueeContext1;
setParent Custom;
toolButton  -cl  toolCluster   -t simpleMarqueeContext1
            -i1 "simpleMarqueeTool.xpm" simpleMarqueeTool1;
```

MPxToolCommand

- Commands executed from within a context
- Can also be executed from Maya command line
- Derived from MPxCommand
- Support Undo / Redo / Journalling

MPxToolCommand



MPxToolCommand

Derived from MPxCommand, special functions:

- MPxToolCommand.cancel()
- MPxToolCommand.finalize()
- MPxToolCommand.doFinalize(MArgList)

Register MPxToolCommand:

```
def initializePlugin( obj )
    plugin = OpenMayaMPx.MFnPlugin(obj)
    try:
        plugin.registerContextCommand( “helixToolContext”, helixContextCmd_creator,
    “helixToolCmd”, helixToolCmd_creator, helixToolCmd_newSyntax);
    except:
        raise
```

Example: helixTool.py

```
def helixContext (OpenMayaMPx.MPxContext):
    def __init__(self):
        OpenMayaMPx.MPxContext.__init__(self)

    def toolOnSetup(self, event):
    def doPress(self, event):
    def doDrag(self, event):
    def doRelease(self, event):
    def doEnterRegion(self, event):

    def getClassName(name) :
    def setNumCVs (newNumCVs):
    def numCVs ():

    … other functions or variable definitions….
```

Example: helixTool.py

```
def doRelease(self,event):

    # Clear the overlay plane & restore from overlay drawing
    self.__view.clearOverlayPlane()
    self.__view.endOverlayDrawing()
    self.__view.endGL()

    cmd = self._newToolCommand()
    cmd.setPitch( height/numCV )
    cmd.setRadius( radius )
    cmd.setNumCVs( numCV )
    cmd.redoIt()
    cmd.finalize()
```

MPxContext::newToolCommand()

Example: helixTool.py

```
class helixToolCmd (OpenMayaMPx.MPxToolCommand):

        def __init__(self):
            OpenMayaMPx.MPxToolCommand.__init__(self)

        def doIt(self, args):
        def parseArgs(self, args):

        def redoIt(self):
        def undoIt(self):
        def isUndoable(self):

        def finalize(self):

        #Other help functions etc….
        def setRadius(self, newRadius):
        def setPitch(self, newPitch):
        …….

def helixToolCmd_newSyntax ():
```

Example: helixTool.py

```
def finalize(self):

    command = OpenMaya.MArgList()
    command.addArg(self.commandString())
    command.addArg(self.radius)
    command.addArg(self.pitch)
    command.addArg(self.numCV)
    try:
        OpenMayaMPx.MPxToolCommand._doFinalize(self, command)
    except:
        pass
```

```
helixToolCmd -r 5  -p 10  -ncv 30;
```

Tool Representation

Tool Property Sheet MEL Scripts:

- namePropertiesUI.mel — helixPropertiesUI.mel
- nameValues.mel — helixValues.mel

Image File: XPM file

Example: helixTool.py

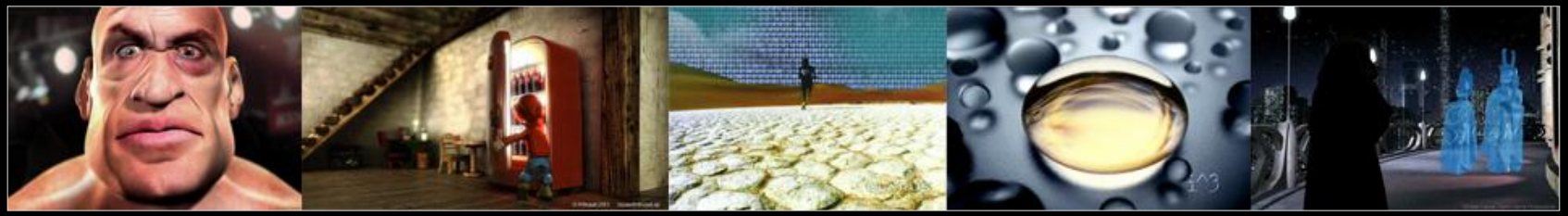
```
class helixContextCmd (OpenMayaMPx. MPxContextCommand):

    def __init__(self):
        OpenMayaMPx. MPxContextCommand.__init__(self)
    def doEditFlags(self)
    def doQueryFlags(self)
    def makeObj(self)
    def appendSyntax(self)

def helixContextCmd _creator():
    return OpenMayaMPx.asMPxPtr (helixContextCmd ())
```

work with tool property sheet

Workshop Session



Autodesk®

Examples: arrowLocator

In this project, we will implement a custom locator, it has a unit attribute “windDirection”. This locator is drawn as a big arrow in the Maya viewport. You can change the direction of the locator by retrieving its “windDirection” attribute and rotating corresponding angles when drawing the locator node with OpenGL calls.

Autodesk