Manipulators

Module 10

Autodesk

Autodesk Maya Python API Training

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

Autodesk

Maya Manipulators

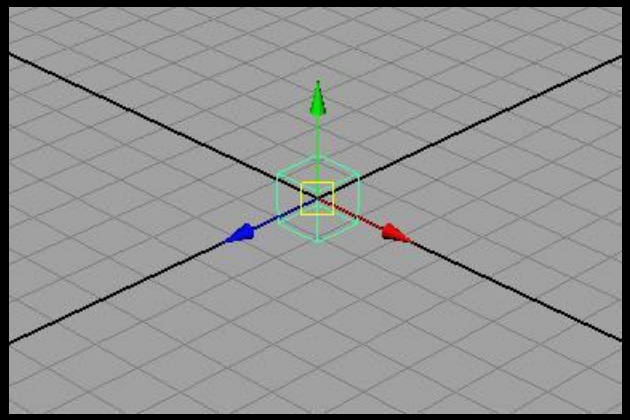
- What is a manipulator?
  - A node that draws itself using 3D graphical elements that respond to user events. An intuitive way to change a node's attribute.
- Examples: swissArmyManip

Base Manipulators & API class:

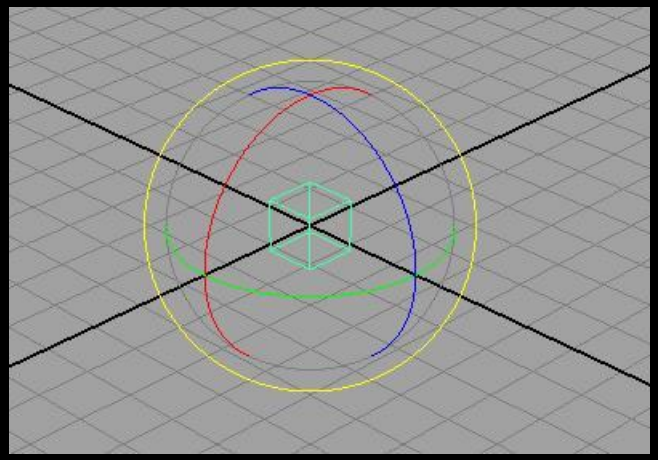
Maya Standard Manipulators: MFnManip3D

- FreePointTriadManip: MFnFreePointTriadManip
- RotateManip: MFnRotateManip
- ScaleManip: MFnScaleManip
- DirectionManip: MFnDirectionManip
- DistanceManip: MFnDistanceManip
- DiscManip: MFnDiscManip
- PointOnCurveManip: MFnPointOnCurveManip
- PointOnSurfaceManip: MFnPointOnSurfaceManip
- CircleSweepManip: MFnCircleSweepManip
- ToggleManip: MFnToggleManip
- StateManip: MFnStateManip
- CurveSegmentManip: MFnCurveSegmentManip

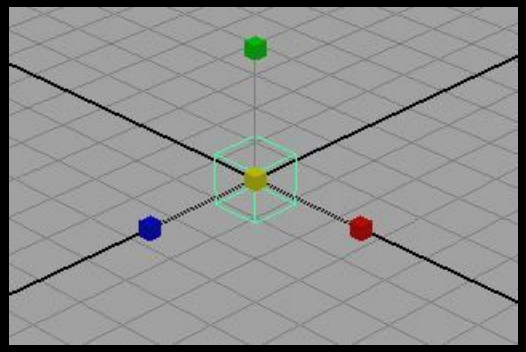
FreePointTriadManip



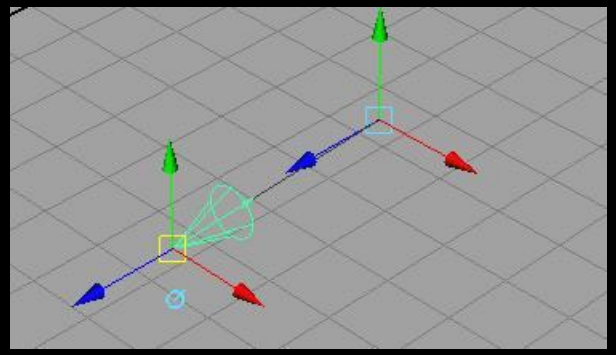
RotateManip



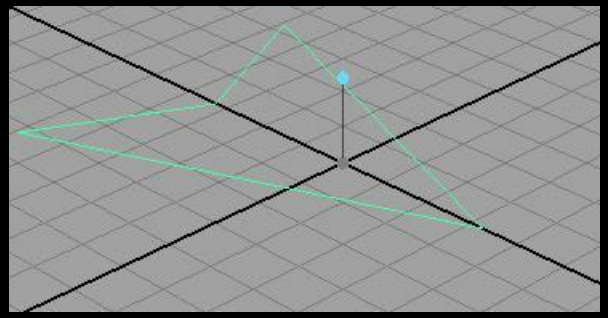
ScaleManip



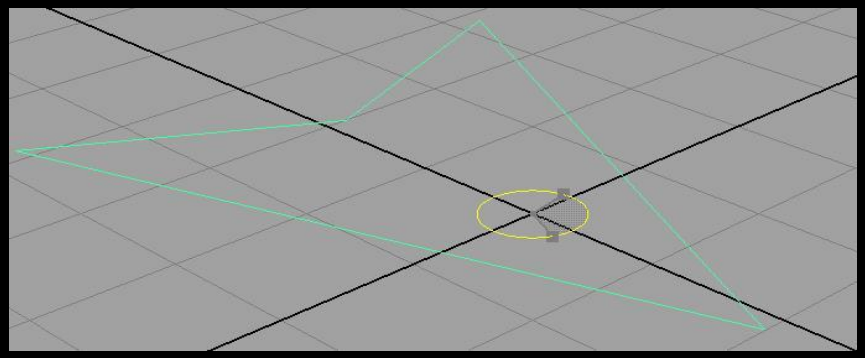
DirectionManip



DistanceManip



DiscManip



Custom Manipulator

- Maya Standard Manipulators

- Derived from MPxManipContainer
  - Container rather than a proxy manipulator
  - Can contain more than one base manipulator
  - Basic workflow:
    - add base manipulators to container
    - Set up associations between base manipulators and attributes

MPxManipContainer

- Specify kManipContainer when registering the node in initializePlugin

```
def initializePlugin(obj):
  ...
  plugin.registerNode( "arrowLocatorManip", arrowLocatorManip::id, arrowLocatorManip_creator,
        arrowLocatorManip_initialize, OpenMayaMPx.MPxNode::kManipContainer )
 ...
```

Example: arrowLocatorManip.py

```
Class arrowLocatorManip (OpenMayaMPx.MPxManipContainer):

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

    def createChildren(self):
    def connectToDependNode(self, dependNode):
    def draw(self, view, path, style, status):


fDiscManip = OpenMaya.MDagPath()
```

MPxManipContainer

- Key functionalities:
  - add base manipulators:

    MPxManipContainer::createChildren()

  - make associations between manipulators and attributes on nodes:

    MPxManipContainer::connectToDependNode()

  - customize the drawing of your manipulator:

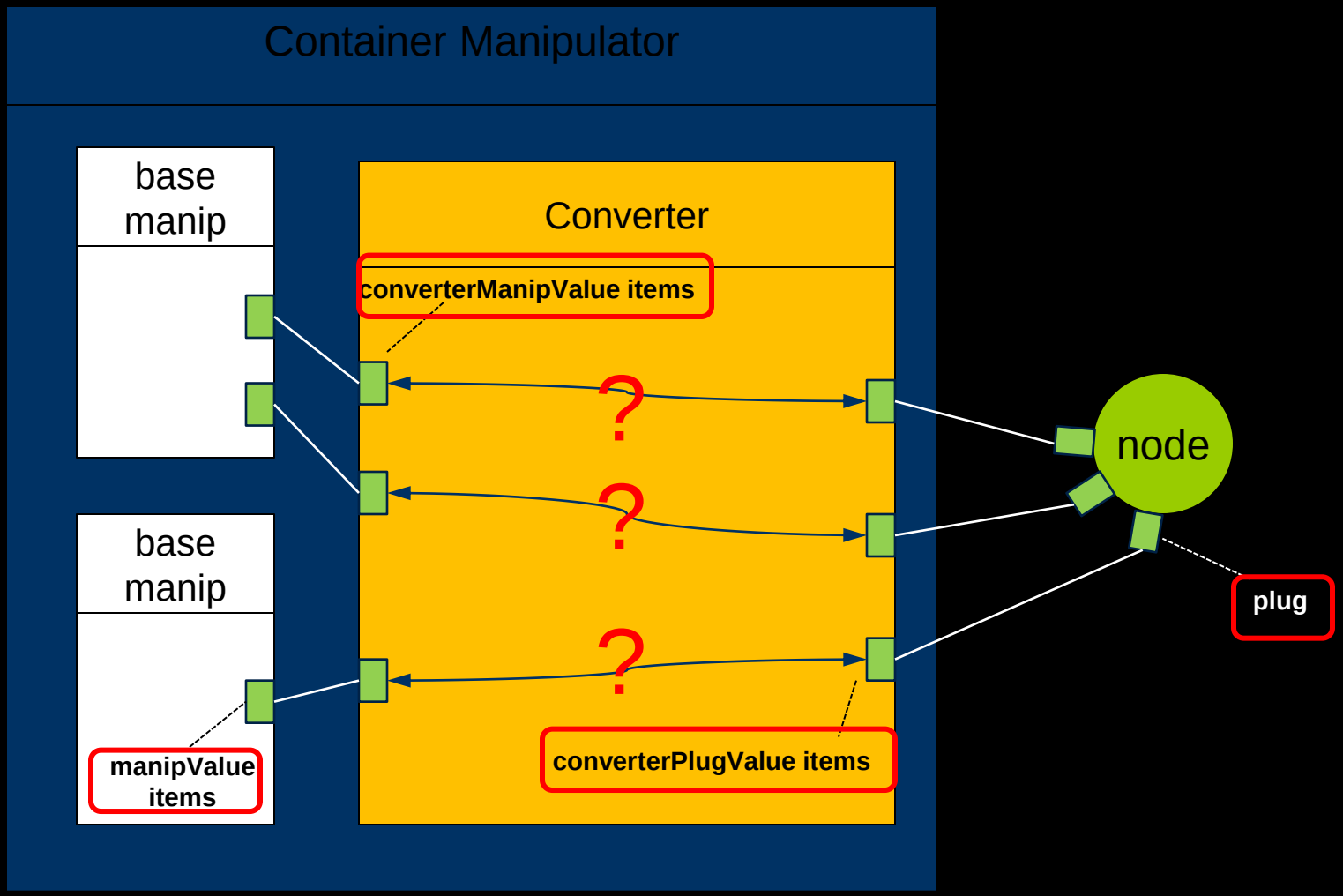
    MPxManipContainer::draw()

Add Base Manipulators

MPxManipContainer.createChildren()

```
def createChildren(self):
    ...
    manipName ="angleManip"
    angleName ="yRotation"
    fDiscManip = self.addDiscManip(manipName,angleName)

    startPoint = OpenMaya.MPoint(0,0,0)
    startAngle = OpenMaya.MAngle(0.0,OpenMaya.MAngle.kDegrees)
    fnDisc = OpenMayaUI.MFnDiscManip (fDiscManip)
    fnDisc.setCenterPoint(startPoint)
    fnDisc.setAngle(startAngle)
    ...
```

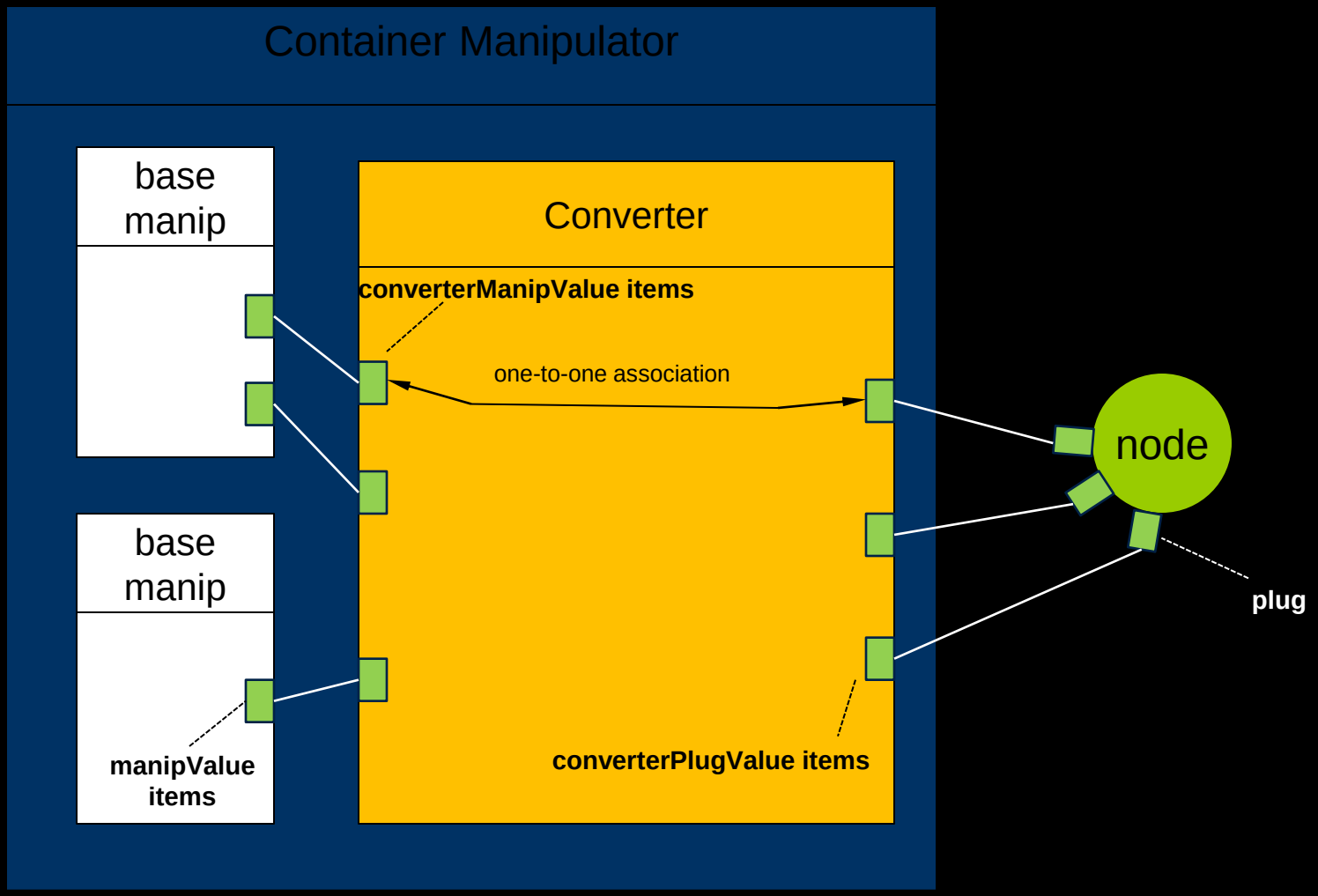
Communications between Manipulators and Nodes



Communications between Manipulators and Nodes

- Two types:
  - One-to-one association
  - Conversion functions
- MPxManipContainer::connectToDependNode()

Communications between Manipulators and Nodes



One-to-one Association

- One-to-one associations are established through methods on the manipulator classes derived from MFnManip3D
- Every standard manipulator has functions called “connectTo**Plug”
  - MFnFreePointTriadManip::connectToPointPlug
  - MFnDirectionManip::connectToDirectionPlug
  - MFnDistanceManip::connectToDistancePlug
  - MFnPointOnCurveManip::connectToCurvePlug
  - …
  - ….
  - MFnScaleManip::connectToScaleCenterPlug

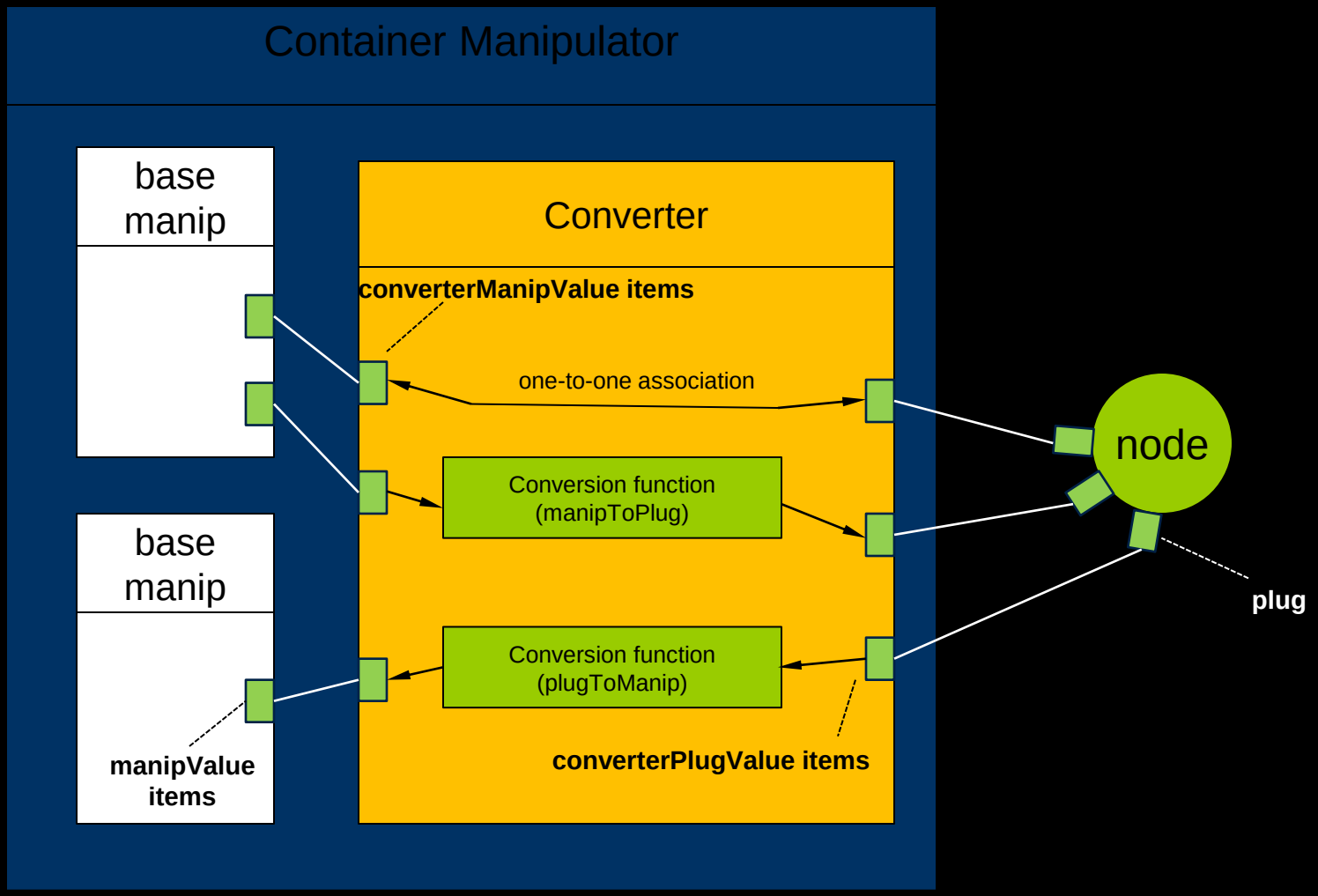
Example: arrowLocatorManip.py

```
def connectToDependNode(self, dependNode)

    #Connect the plug with manip
    fnDepNode = OpenMaya.MFnDependencyNode (dependNode)
    rotationPlug = fnDepNode.findPlug("windDirection”)

    fnDisc = OpenMayaUI.MFnDiscManip (fDiscManip)
    fnDisc.connectToAnglePlug(rotationPlug)

    OpenMayaMPx.MPxManipulatorNode.finishAddingManips(self)
    OpenMayaMPx.MPxManipulatorNode.connectToDependNode(self, dependNode)
```

Example: arrowLocatorManip.py

In this project, you will need to create a manipulator on the arrowLocator and set up the one-to-one relationship between the manipulator item value and the plug value on your node.

Communications between Manipulators and Nodes



Conversion Functions

- Convert between manipulator values and plug values
- Implemented as callback methods
- Two kinds of conversion callbacks:
  - manipToPlug
    - addManipToPlugConversion (unsigned int manipIndex)

      converterManipValues → converterPlugValues
  - plugToManip
    - addPlugToManipConversion (unsigned int manipIndex)

      converterPlugValues → converterManipValues
  - The converter must be implemented in the manipToPlugConversion() and plugToManipConversion virtual method of these classes
- MManipData

Example:arrowLocatorManip.py

```
def connectToDependNode(self, dependNode):
    ....
    fnDepNode = OpenMaya.MFnDependencyNode (dependNode)
    rotationPlug = fnDepNode.findPlug("windDirection")

    fnDisc = OpenMayaUI.MFnDiscManip (fDiscManip)
    fnDisc.connectToAnglePlug(rotationPlug)

    centerPointIndex = fnDisc.centerIndex()
    OpenMayaMPx.MPxManipulatorNode.addPlugToManipConversion (centerPointIndex)
    ....
```

Example:arrowLocatorManip.py

```
def plugToManipConversion( self, manipIndex ):
    try:
        #Get parent transform node of the locator node
        parentTransform = self.fNodePath.transform()
        #Get the transform node DAG path
        transformPath = OpenMaya.MDagPath()
        OpenMaya.MDagPath.getAPathTo(parentTransform,transformPath)
        #Retrieve world space translation
        fnTrans = OpenMaya.MFnTransform(transformPath)
        translation = OpenMaya.MVector()
        translation = fnTrans.getTranslation(OpenMaya.MSpace.kWorld)
        numData = OpenMaya.MFnNumericData()
        numDataValue = numData.create(OpenMaya.MFnNumericData.k3Double)
        status = numData.setData3Double(translation.x,translation.y,translation.z)
        manipData = OpenMayaUI.MManipData(numDataValue)
    except:
        sys.stderr.write("ERROR: arrowManip.plugToManipConversion\n")
        raise
    return manipData
```

MPxManipContainer::draw()

```
def draw(self, view, path, style, status):
    OpenMayaMPx.MPxManipContainer.draw(self, view, path, style, status)
    view.beginGL()
    textPos = OpenMaya.MPoint (0, 0, 0)
    distanceText = "Stretch Me!"
    view.drawText(distanceText, textPos, OpenMayaUI.M3dView.kLeft)
    view.endGL()
```

Invoke manipulators

- Show Manipulator tool
  - Manipulator named after node type name

    arrowLocator → arrowLocatorManip
  - In custom node's initialize(), call
    MPxManipContainer::addToManipConnectTable(MTypeId &id)

- Custom Context

Manipulators and Contexts

MPxManipContainer::newManipulator (const MString & manipName,
MObject & manipObject, MStatus * ReturnStatus = NULL)

MPxContext::addManipulator(const MObject &manipulator)

MPxContext::deleteManipulators()

MPxContext::toolOnSetup()

MPxContext::toolOffCleanup()

Example: moveManip.py (devkit)

```
Id1 = OpenMaya.MCallbackId()

def toolOnSetup(self, event):
    ....
    updateManipulators(self)
    id1 = OpenMaya.MModelMessage.addCallback(OpenMaya.MModelMessage.kActiveListModified,
    updateManipulators,  self)
    ....

def toolOffCleanup(self):
    ...
    OpenMaya.MModelMessage.removeCallback(id1)
    ...
```

Example: moveManip.py (devkit)

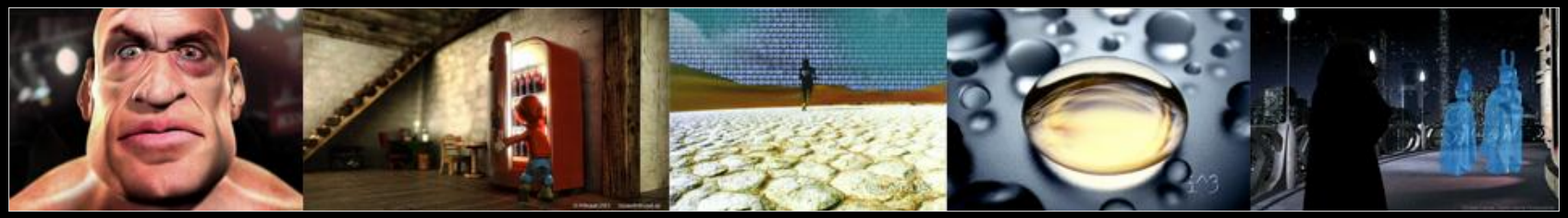
```
def updateManipulators(data)
    ...
    clientData.deleteManipulators()  ...
    # for each object selected
    manipName ="moveManip"
    manipObject = OpenMaya.MObject()
    manipulator = OpenMayaMPx.MPxManipContainer.newManipulator(manipName,manipObject)
    ...
    if manipulator is not None:
        clientData.addManipulator(manipObject)
        manipulator.connectToDependNode(dependNode)
    ...
```

Major Limitation

Cannot create a custom manipulator that:

- Plug-in draws itself
- Maya performs selection on manipulator components

MPxManipulatorNode: openGL draw and selection



Autodesk®

Autodesk