Maya Python Introduction

The Maya Environment

- When maya starts up it reads a file called Maya.env this contains a number of environment variables for Maya.
- On a mac this is located in ~Library/Preferences/Autodesk/maya/[Version]
- Under Linux ~/maya/version/
- We are going to use this to setup some directories to use for the next few weeks Lectures

Basic Setup

```
cd $HOME
mkdir MayaScripts MayaPlugs
```

Add this to the Maya.env (you may need both versions of Maya)

```
MAYA_PLUG_IN_PATH=/home/jmacey/MayaPlugs
MAYA_SCRIPT_PATH=/home/jmacey/MayaScripts
PYTHONPATH=/home/jmacey/MayaScripts
```

A simple Mel Script Test

```
global proc foo()
{
  print("foo\n");
}
```

- Save the file in the scripts directory as test.mel
- in the Mel window source “test.mel”
- then call foo()

A Simple Python Script

```
def foo()
   print "foo!"
```

- Again save this script as foo.py in the Scripts directory
- We need to import the module before we use it so the following is needed

```
import foo

foo.foo()
```

reload

- To help the development cycle, we can easily modify the script and call
- reload (Test)
- to reload the module from the source file, otherwise this will be the same module for the whole of the session.
- Also note that each tab of the script editor is also a different instance so each module is unique to the tab imported

Command Mode

- The simplest way to use Python in maya is with the command module
- This is very similar to the mel scripting language and allows access to most of the basic maya features
- To load the module we use the following import code

```
1 import maya.cmds as cmds
```

- We then use cmds.[function] to access the objects

A simple example

```
1  # let's create a sphere
2  import maya.cmds as cmds
3
4  cmds.sphere(radius=2, name='Sphere1')
5
6  # query the radius of the sphere named Sphere1
7  radius = cmds.sphere('Sphere1', query=True, radius=True)
8  print 'The sphere radius =',radius
9
10 # modify the radius
11 print 'Increment the sphere\'s radius by one unit'
12 cmds.sphere('Sphere1', edit=True, radius=radius+1)
13 radius = cmds.sphere('Sphere1', query=True, radius=True)
14 print 'The new sphere radius =', radius
```

Create a sphere called Sphere1

Use Query mode to get the radius

Use Edit mode to change the radius

moving objects

```
1  # let's create a sphere
2  import maya.cmds as cmds
3
4  # let's delete all objects
5  cmds.select(all=True)
6  cmds.delete()
7  #let's create a new torus
8  cmds.torus(r=4, hr=0.5, name='Torus1')
9  cmds.move(0, 0, 0, 'Torus1')
10 cmds.scale(0.5, 0.5, 0.5, 'Torus1')
11 cmds.rotate(0, '45deg', 0, 'Torus1')
```

select all and delete

Create a torus

use the affine transform commands

Setting Attributes

```
1  # let's create a sphere
2  import maya.cmds as cmds
3
4  # lets delete all objects
5  cmds.select(all=True)
6  cmds.delete()
7  #lets create a nurb's cube
8  cmds.nurbsCube(w=3, name='Cube1')
9  cmds.getAttr('Cube1.tx')
10 cmds.getAttr('Cube1.ty')
11 cmds.getAttr('Cube1.tz')
12
13 # let's make sure 'Cube1' is selected
14 cmds.select('Cube1', replace=True)
15 #let's change its translation attributes
16 cmds.setAttr('Cube1.tx', 1)
17 cmds.setAttr('Cube1.ty', 2)
18 cmds.setAttr('Cube1.tz', 3)
```

select all and delete

get attributes

set attributes

keyframes

```
1  import maya.cmds as cmds
2
3  cmds.polySphere(name='ball', radius=2)
4
5  g = -0.04
6  yVelocity = 0.8
7  xVelocity = 0.2
8  initialYPos = 6
9  initialXPos = 0
10 cmds.move(initialXPos,initialYPos,0,"ball")
11 posy=initialYPos
12 timeMult=4
13 for bounces in range(0,6):
14   time=0
15   posy=initialYPos
16   while posy>0 :
17     posy = initialYPos + yVelocity*(time-1) + g*(time-1)*(time-1)/2
18     posx = initialXPos + xVelocity*((time) + time-1)
19     cmds.setKeyframe( 'ball', attribute='translateY', value=posy, t=time*timeMult )
20     cmds.setKeyframe( 'ball', attribute='translateX', value=posx, t=time*timeMult )
21     time+=1
22   yVelocity-=0.1
```

Accessing API Values

- The Maya C++ api uses pass by reference to return variable values
- As python has no real way to access these values we need to use the MScriptUtil classes
- The MScriptUtil class is Utility class for working with pointers and references in Python

In Python parameters of class types are passed by reference but parameters of simple types, like integers and floats, are passed by value, making it impossible to call those API methods from Python. The MScriptUtil class bridges this gap by providing methods which return pointers to values of simple types and which can extract values from such pointers. These pointers can also be used wherever an API method requires a reference to a simple type or an array of a simple type.

This class is admittedly cumbersome to use but it provides a way of building parameters and accessing return values for methods which would not normally be accessible from Python.

MImage

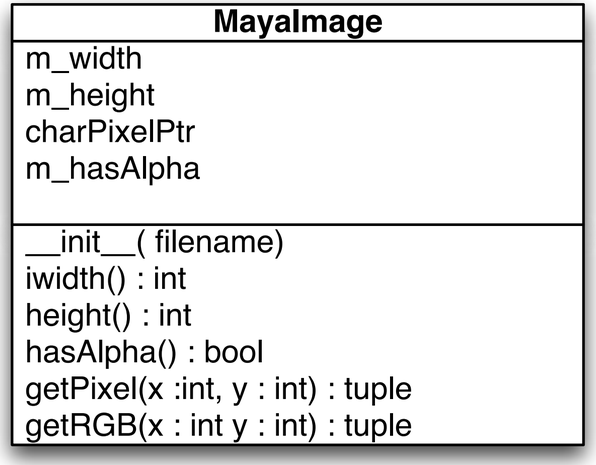
- MImage is part of the OpenMaya API
- It provides access to some of Maya's image manipulation functionality.
- It has methods for loading and resizing image files in any Maya-supported raster format, including IFF, SGI, Softimage (pic), TIFF (tif), Alias PIX (als), GIF, RLA, JPEG (jpg).
- The image is stored as an uncompressed array of pixels, that can be read and manipulated directly.
- For simplicity, the pixels are stored in a RGBA format (4 bytes per pixel).

Public Member Functions

	MImage ()
	~MImage ()
MStatus	**create** (unsigned int width, unsigned int height, unsigned int channels=4, **MPixelType** type=kByte)
MStatus	**readFromFile** (**MString** pathname, **MPixelType** type=kByte)
MStatus	**readFromTextureNode** (const **MObject** &fileTextureObject, **MPixelType** type=kByte)
MStatus	**getSize** (unsigned int &width, unsigned int &height) const
MPixelType	**pixelType** () const
unsigned char *	**pixels** () const
float *	**floatPixels** () const
void	**setPixels** (unsigned char *pixels, unsigned int width, unsigned int height)
void	**setFloatPixels** (float *pixels, unsigned int width, unsigned int height, unsigned int channels=4)
unsigned int	**depth** () const
MStatus	**getDepthMapSize** (unsigned int &width, unsigned int &height) const
MStatus	**setDepthMap** (float *depth, unsigned width, unsigned height)
MStatus	**setDepthMap** (const **MFloatArray** &depth, unsigned width, unsigned height)
float *	**depthMap** (**MStatus** *ReturnStatus=NULL) const
MStatus	**readDepthMap** (**MString** pathname)
MStatus	**resize** (int width, int height, bool preserveAspectRatio=true)
MStatus	**filter** (**MImageFilterFormat** sourceFormat, **MImageFilterFormat** targetFormat, double scale=1.0, double offset=0.0)
MStatus	**writeToFile** (**MString** pathname, **MString** outputFormat=**MString**("iff")) const
MStatus	**writeToFileWithDepth** (**MString** pathname, **MString** outputFormat=**MString**("iff"), bool writeDepth=false) const
MStatus	**release** ()
void	**verticalFlip** ()
void	**setRGBA** (bool rgbaFormat)
bool	**isRGBA** () const
bool	**haveDepth** () const
MStatus	**convertPixelFormat** (**MPixelType** type, double scale=1.0, double offset=0.0)

A Python Wrapper Class

- Most of the work in this class will be done in the ctor
- We will store values for width and height as well as a flag to indicate if we have alpha
- A pointer to the image data is also stored and we can access the pixel values using an index operator

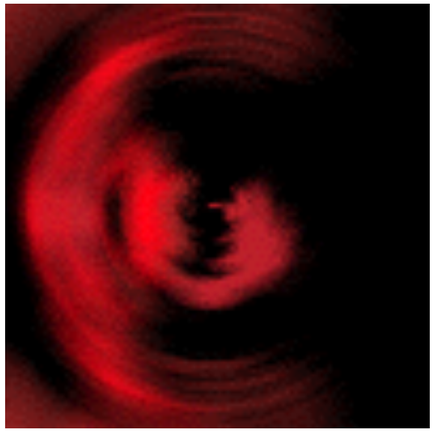


http://jonmacey.blogspot.com/2011/04/using-maya-mscriptutil-class-in-python.html

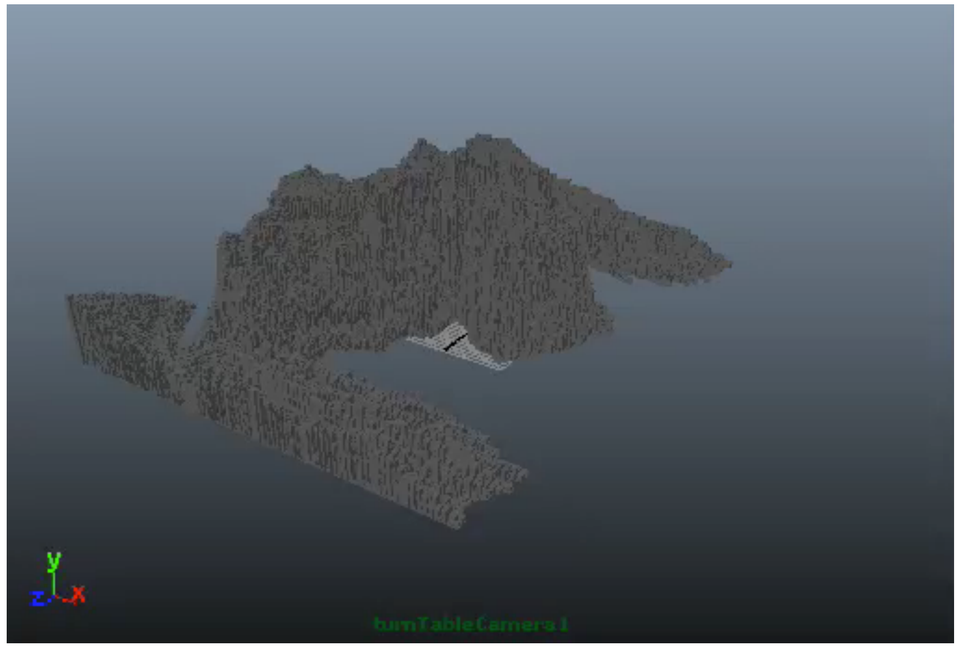
```
1  import maya.OpenMaya as om
2  import sys
3
4  class MayaImage :
5    """ The main class, needs to be constructed with a filename """
6    def __init__(self,filename) :
7      """ constructor pass in the name of the file to load (absolute file
           name with path) """
8      # create an MImage object
9      self.image=om.MImage()
10     # read from file MImage should handle errors for us so no need to check
11     self.image.readFromFile(filename)
12     # as the MImage class is a wrapper to the C++ module we need to access
           data
13     # as pointers, to do this use the MScritUtil helpers
14     self.scriptUtilWidth = om.MScriptUtil()
15     self.scriptUtilHeight = om.MScriptUtil()
16
17     # first we create a pointer to an unsigned in for width and height
18     widthPtr = self.scriptUtilWidth.asUintPtr()
19     heightPtr = self.scriptUtilHeight.asUintPtr()
20     # now we set the values to 0 for each
21     self.scriptUtilWidth.setUint( widthPtr, 0 )
22     self.scriptUtilHeight.setUint( heightPtr, 0 )
23     # now we call the MImage getSize method which needs the params passed
           as pointers
24     # as it uses a pass by reference
25     self.image.getSize( widthPtr, heightPtr )
26     # once we get these values we need to convert them to int so use the
           helpers
27     self.m_width = self.scriptUtilWidth.getUint(widthPtr)
28     self.m_height = self.scriptUtilHeight.getUint(heightPtr)
29     # now we grab the pixel data and store
30     self.charPixelPtr = self.image.pixels()
31     # query to see if it's an RGB or RGBA image, this will be True or False
32     self.m_hasAlpha=self.image.isRGBA()
33     # if we are doing RGB we step into the image array in 3's
34     # data is always packed as RGBA even if no alpha present
35     self.imgStep=4
36     # finally create an empty script util and a pointer to the function
37     # getUcharArrayItem function for speed
38     scriptUtil = om.MScriptUtil()
39     self.getUcharArrayItem=scriptUtil.getUcharArrayItem
```

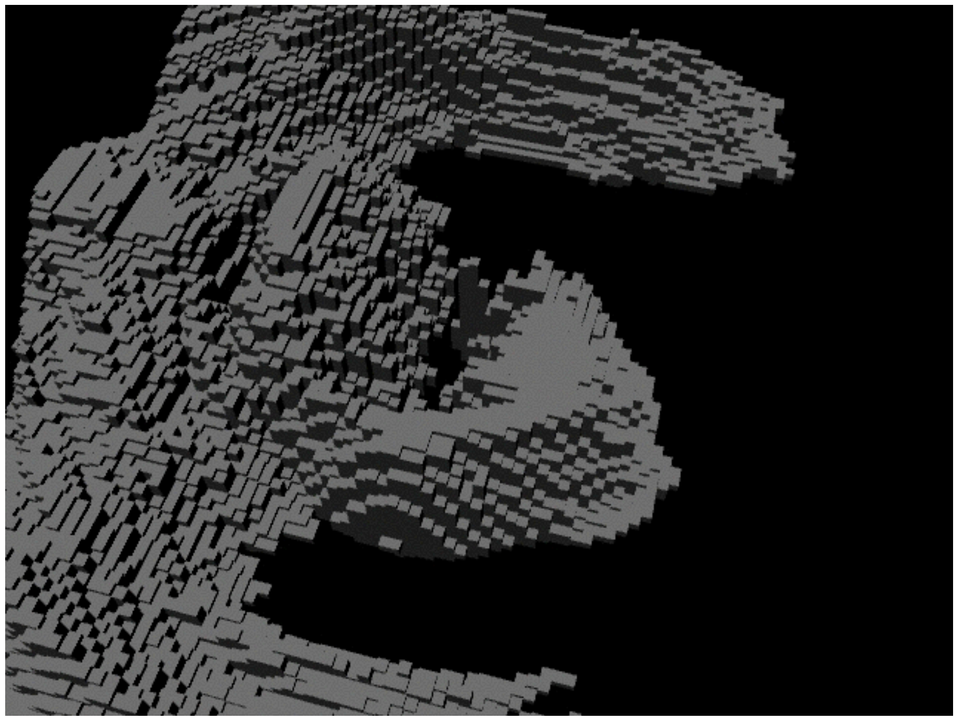
```
def getPixel(self,x,y) :
  """ get the pixel data at x,y and return a 3/4 tuple
      depending upon type """
  # check the bounds to make sure we are in the correct area
  if x<0 or x>self.m_width :
    print "error x out of bounds\n"
    return
  if y<0 or y>self.m_height :
    print "error y our of bounds\n"
    return
  # now calculate the index into the 1D array of data
  index=(y*self.m_width*4)+x*4
  # grab the pixels
  red = self.getUcharArrayItem(self.charPixelPtr,index)
  green = self.getUcharArrayItem(self.charPixelPtr,index+1)
  blue = self.getUcharArrayItem(self.charPixelPtr,index+2)
  alpha=self.getUcharArrayItem(self.charPixelPtr,index+3)
  return (red,green,blue,alpha)

def getRGB(self,x,y) :
  r,g,b,a=getPixel(x,y)
  return (r,g,b)
```

```
def width(self) :
    """ return the width of the image """
    return self.m_width
def height(self) :
  """ return the height of the image """
  return self.m_height

def hasAlpha(self) :
  """ return True is the image has an Alpha channel """
  return self.m_hasAlpha
```



```
1  import maya.OpenMaya as om
2  import maya.cmds as cmds
3
4
5  basicFilter = "*.*"
6
7  imageFile=cmds.fileDialog2(caption="Please select imagefile",
8                             fileFilter=basicFilter, fm=1)
9
10 img=MayaImage(str(imageFile[0]))
11 print img.width()
12 print img.height()
13 xoffset=-img.width()/2
14 yoffset=-img.height()/2
15
16 for y in range (0,img.height()) :
17   for x in range(0,img.width()) :
18     r,g,b,a=img.getPixel(x,y)
19     if r > 10 :
20       cmds.polyCube(h=float(r/10))
21       cmds.move(xoffset+x,float(r/10)/2,yoffset+y)
```



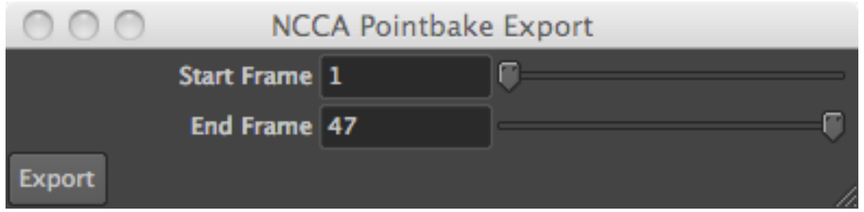


PointBake Animation

- The following examples show how to export mesh data into our own point baked format
- Point baking takes an original mesh (usually exported as an obj) and stores each vertex value for each frame as a relative value
- In this case we are going to write the data out as an XML file as this is easier to parse using python for the loader

```
<?xml version="1.0" encoding="UTF-8" ?>
<NCCAPointBake>
  <MeshName> pCube1 </MeshName>
  <NumVerts> 8 </NumVerts>
  <StartFrame> 0 </StartFrame>
  <EndFrame> 100 </EndFrame>
  <NumFrames> 100 </NumFrames>
  <TranslateMode> absolute </TranslateMode>
  <Frame number="0">
    <Vertex number="0" attrib="translate"> -7.478646 3.267710 5.018191 </Vertex>
    <Vertex number="1" attrib="translate"> -4.407852 3.267710 5.018191 </Vertex>
    <Vertex number="2" attrib="translate"> -7.478646 6.199014 5.018191 </Vertex>
    <Vertex number="3" attrib="translate"> -4.407852 6.199014 5.018191 </Vertex>
    <Vertex number="4" attrib="translate"> -7.478646 6.199014 2.284239 </Vertex>
    <Vertex number="5" attrib="translate"> -4.407852 6.199014 2.284239 </Vertex>
    <Vertex number="6" attrib="translate"> -7.478646 3.267710 2.284239 </Vertex>
    <Vertex number="7" attrib="translate"> -4.407852 3.267710 2.284239 </Vertex>
  </Frame>
</NCCAPointBake>
```

User Interface



- The user interface for the exported is created using the cmds.window() method
- The code on the next page show how the components are added to the window and callbacks are associated to the buttons
- The whole UI is wrapped into a single class with logic to ensure that things are selected before the system will run.

```
class PointBakeExport() :

  # @brief ctor

  def __init__(self) :
    # get the currently selected objects and make sure we have only one object
    selected = OM.MSelectionList()
    OM.MGlobal.getActiveSelectionList(selected)
    self.selectedObjects = []
    selected.getSelectionStrings(self.selectedObjects)
    if len(self.selectedObjects) == 0 :
      cmds.confirmDialog( title='No objects Selected', message='Select a Mesh Object',
          button=['Ok'], defaultButton='Ok', cancelButton='Ok', dismissString='Ok' )
    elif len(self.selectedObjects) > 1 :
      cmds.confirmDialog( title='Select One Object', message='Only One Mesh mat be
          exported at a time', button=['Ok'], defaultButton='Ok', cancelButton='Ok',
          dismissString='Ok' )
    # now we have the correct criteria we can proceed with the export
    else :
      # get the start and end values for our UI sliders
      anim=OMA.MAnimControl()
      minTime=anim.minTime()
      maxTime=anim.maxTime()
      self.m_start=int(minTime.value())
      self.m_end=int(maxTime.value())
      # now we create a window ready to populate the components
      self.m_window = cmds.window( title='NCCA Pointbake Export' )
      # create a layout
      cmds.columnLayout()
      # create two sliders for start and end we also attach methods to be called when
          the slider
      # changes
      self.m_startSlider=cmds.intSliderGrp( changeCommand=self.startChanged,field=True,
          label='Start Frame', minValue=self.m_start, maxValue=self.m_end, fieldMinValue
          =self.m_start, fieldMaxValue=self.m_end, value=self.m_start )
      self.m_endSlider=cmds.intSliderGrp( changeCommand=self.endChanged ,field=True,
          label='End Frame', minValue=self.m_start, maxValue=self.m_end, fieldMinValue=
          self.m_end, fieldMaxValue=self.m_end, value=self.m_end )
      # create a button and add the method called when pressed
      cmds.button( label='Export', command=self.export )
      # finally show the window
      cmds.showWindow( self.m_window )
```

```
# @brief export method attached ot the button, this will be executed once every time
# the button is pressed
# @param *args the arguments passed from the button

def export(self,*args) :
  # get the file name to save too
  basicFilter = "*.xml"
  file=cmds.fileDialog2(caption="Please␣select␣file␣to␣save",fileFilter=basicFilter, dialogStyle
      =2)
  # check we get a filename and then save
  if file !="" :
    if self.m_start >= self.m_end :
      cmds.confirmDialog( title='Range␣Error', message='start␣>=␣end', button=['Ok'],
          defaultButton='Ok', cancelButton='Ok', dismissString='Ok' )
    else :
      NCCAPointBake(file,self.selectedObjects[0],self.m_start,self.m_end)
      # finally remove the export window
      cmds.deleteUI( self.m_window, window=True )

# @brief this is called every time the slider is changed (i.e. a new value)
# @param *args the arguments passed from the button [0] is the numeric value

def startChanged(self, *args) :
  self.m_start=args[0]

# @brief this is called every time the slider is changed (i.e. a new value)
# @param *args the arguments passed from the button [0] is the numeric value

def endChanged(self, *args) :
  self.m_end=args[0]
```

```python
def NCCAPointBake(_fileName,_name,_startFrame,_endFrame) :

  # grab the selected object
  selected = OM.MSelectionList()
  obj=OM.MObject( )
  selected.add(_name)
  selected.getDependNode(0,obj)
  # get the parent transform
  fn = OM.MFnTransform(obj)
  Mesh=""
  oChild = fn.child(0)
  # check to see if what we have is a mesh
  if(oChild.apiTypeStr()=="kMesh") :
    print "got Mesh"
    # get our mesh
    Mesh=OM.MFnMesh(oChild)
  else :
    print "Didn't get mesh ", oChild.apiType()
    return

  # now we try and open the file for writing
  try :
    file=open(str(_fileName[0]),'w')
  # if this fails catch the error and exit
  except IOError :
    print "Error opening file",str(_fileName)
    return

  # set the frame to start
  print "PB get anim control"
  currFrame=OM.MTime()
  anim=OMA.MAnimControl()
  # as these can take time to process we have an interupter to allow for the process to be
  # stopped
  interupter=OM.MComputation()
  # set the start of the heavy computation
  interupter.beginComputation()
  # now we set the tab level to 0 for the initial write to the file
  tabIndent=0

  # now we get the mesh number of points
  numPoints = cmds.polyEvaluate( _name, v=True)
  # write the xml headers
  file.write("<?xml version=\"1.0\" encoding=\"UTF-8\" ?>\n")
  file.write("<NCCAPointBake>\n")
  # up the tab leve
  tabIndent=tabIndent+1
  # write the initial header data
  WriteData(file,tabIndent,"<MeshName> %s </MeshName>" %(_name))
  WriteData(file,tabIndent,"<NumVerts> %d </NumVerts>" %(numPoints))
  WriteData(file,tabIndent,"<StartFrame> %s </StartFrame>" %(_startFrame))
  WriteData(file,tabIndent,"<EndFrame> %s </EndFrame>" %(_endFrame))
  WriteData(file,tabIndent,"<NumFrames> %s </NumFrames>" %(_endFrame-_startFrame))
  WriteData(file,tabIndent,"<TranslateMode> %s </TranslateMode>" %("absolute"))
```

```python
  # now for every frame write out the vertex data
  for frame in range(_startFrame,_endFrame) :
    print "Doing frame %04d" %(frame)
    # move to the correct frame
    currFrame.setValue (frame)
    anim.setCurrentTime(currFrame)
    # write out the frame tag
    WriteData(file,tabIndent,"<Frame number=\"%d\">" %(frame))
    tabIndent=tabIndent+1
    for vertex in range(0,numPoints) :
      # now the actual vertex data for the current mesh index value
      data = cmds.xform( (_name+ ".vtx["+str(vertex)+"]"), q=True, ws=True, t=True )
      WriteData(file,tabIndent,"<Vertex number=\"%d\" attrib=\"translate\"> %f %f %f </Vertex>"
          %(vertex,data[0],data[1],data[2]))
    # now un-indent as we have ended the frame
    tabIndent=tabIndent-1
    WriteData(file,tabIndent,"</Frame>")
    # if we have interupted exit and finish
    if interupter.isInterruptRequested()  :
      file.write("</NCCAPointBake>\n")
      file.close()
      print "File export interrupted ";
      return
  # now finish
  file.write("</NCCAPointBake>\n")
  # and close the file
  file.close()
```

```python
def WriteData(_file,_nTabs,
    _data) :
  for i in range(0,_nTabs) :
    _file.write("\t")
  _file.write(_data)
  _file.write("\n")
```

Maya PointBake Import

- This script imports both the obj and the pointbake file by poping up a dialog box for selection of the files
- It also demonstrates the input dialog box for text input
- As well as the xml.sax xml parser

```
class PointBakeImport() :
  # ctor
  def __init__(self) :

    # create a promptDialog for the base group name of our mesh this will help to
    # avoid name conflicts, may be good to modify this at some stage to check if mesh
    # exists and prompt to replace data / key
    result = cmds.promptDialog(
                              title='Name',
                              message='Enter␣Name␣for␣import',
                              button=['OK', 'Cancel'],
                              defaultButton='OK',
                              cancelButton='Cancel',
                              dismissString='Cancel')

    # if ok was pressed lets process the data
    if result == 'OK':
      # first we get the text entered by the user
      self.m_text = cmds.promptDialog(query=True, text=True)
      # now get the obj file to import
      basicFilter = "*.obj"
      self.m_objFileName=cmds.fileDialog2(caption="Please␣select␣obj␣file␣to␣import",
          fileFilter=basicFilter, fm=1)

      cmds.file(self.m_objFileName,i=True,type="OBJ",ns=self.m_text)
      # now the xml file
      basicFilter = "*.xml"
      self.m_pointBakeFile=cmds.fileDialog2(caption="Please␣select␣xml␣file␣to␣import",
          fileFilter=basicFilter, fm=1)
      # select the object imported
      print self.m_pointBakeFile
      cmds.select("%s:Mesh"%(self.m_text))
      # and pass control back to the parser
      parser = xml.sax.make_parser()
      parser.setContentHandler(ParseHandler("%s:Mesh"%(self.m_text)))
      parser.parse(open(str(self.m_pointBakeFile[0]),"r"))
```

```
class ParseHandler(xml.sax.ContentHandler):

  ## @brief ctor for the class passing in the houdini channel we wish to load the
  ## PB data into
  ## @param[in] _selectedText the mesh the data is to be loaded too
  def __init__(self,_selectedText):
    ## @brief the object selected to load the data too.
    self.m_selectedObject=_selectedText
    ## @brief the Character Data stored as part of parsing
    self.m_charData=""
    ## @brief the m_meshName extracted from the PointBake file
    self.m_meshName=""
    ## @brief number of vertices in the mesh, we will check this against the number of
        points in
    ## the mesh / obj loaded as a basic compatibility check
    self.m_numVerts=0
    ## @brief the Start frame for the data loaded
    self.m_startFrame=0
    ## @brief m_endFrame of the data loaded
    self.m_endFrame=0
    ## @brief number of frames stored in file not used in this example
    self.m_numFrames=0
    ## @brief the Offset into the vertex list for the current data to be set too
    self.m_offset=None
    ## @brief the Current frame to be stored / keyed
    self.m_currentFrame=0
    # the maya time control
    self.m_anim=OMA.MAnimControl()
    # a point array structure, we will load each frame's worth of data into this then
    # load it to the mesh point data each frame, once this is done we need to clear this
        data for
    # the next frame
    self.m_vertData=OM.MFloatPointArray()
    # grab the object ready to set the point data
    selected = OM.MSelectionList()
    obj=OM.MObject( )
    selected.add(self.m_selectedObject)
    selected.getDependNode(0,obj)

    fn = OM.MFnTransform(obj)
    self.m_mesh=""
    oChild = fn.child(0)

    if(oChild.apiTypeStr()=="kMesh") :
      print "got Mesh"
      # get our mesh
      self.m_mesh=OM.MFnMesh(oChild)
    # set the frame to start
```

```
    def __del__(self) :
      print "done"
    ## @brief here we trigger events for the start elements In this case we grab the Offset
        and Frame
    ## @param[in] _name the name of the tag to process
    ## @param[in] _attrs the attribute associated with the current tag
    def startElement(self, _name, _attrs):
      # this is important the characters method may be called many times so we
      # clear the char data each start element then append each time we read it
      self.m_charData=""
      # if we have a vertex start tag process and extract the offset
      if _name == "Vertex" :
        self.m_offset=int(_attrs.get("number"))
      # if we have the Frame we grab the number attribute
      elif _name == "Frame" :
        # set the frame here
        self.m_currentFrame=int(_attrs.get("number"))
        self.m_anim.setCurrentTime(OM.MTime(self.m_currentFrame))
        # we have a new frame so re-set the vertexPoint data ready to be filled
        # with the new dara
        self.m_vertData.clear()

    ## @brief trigger method if we have data between the <> </> tags, copy it to the class
        m_charData so
    ## we can re-use it later
    ## \param[in] _content the character string passed from the parser.
    def characters(self,_content):
      # here we append the content data passed into the method, we need to append
      # as this function may be called more than once if we have a long string
      self.m_charData += _content

```

```
  ## @brief most of the hard processing is done here. Once an end tag is encountered we
  ## process the current char data and add it to the channel created. This does
  ## rely on the order of the data but this is always machine generated so we should
  ## be safe if it does go wrong it will be this data ordering
  ## @brief[in] _name the name of the end element tag
  def endElement(self, _name):
    # extract the m_meshName and save it
    if _name == "MeshName":
      self.m_meshName=self.m_charData
    # get the number of vertices and set this to the channel
    elif _name == "NumVerts" :
      # store value
      self.m_numVerts=int(self.m_charData)

    # parse and sel the m_startFrame
    elif _name == "StartFrame" :
      self.m_startFrame=int(self.m_charData)
      # set the time control to this value
      self.m_anim.setMinTime(OM.MTime(self.m_startFrame))
    ## found an end frame value
    elif _name == "EndFrame" :
      self.m_endFrame=int(self.m_charData)
      # set the end animation time
      self.m_anim.setMaxTime(OM.MTime(self.m_endFrame))

    ## found the number of frames
    elif _name == "NumFrames" :
      self.m_numFrames=int(self.m_charData)
    ## found the vertex
    elif _name =="Vertex" :
      self.m_charData=self.m_charData.strip()
      data=self.m_charData.split("␣")
      ## now we check to see if there are enough values to parse
      if len(data) == 3 :
        # append the vertex data to the array for later loading into the mesh
        self.m_vertData.append(float(data[0]),float(data[1]),float(data[2]))
    elif _name=="Frame" :
      # now we have the end of the frame we should have all the vertex data in the array
      # so we can set this point position for our mesh
      self.m_mesh.setPoints(self.m_vertData)
      # once we have done this we can set this as a keyframe
      cmds.setKeyframe(breakdown=0, hierarchy="none",controlPoints=0 ,shape=0,attribute="
          vtx[*]")
      # now we clear the point data ready for the next frame to load hte data in
      self.m_vertData.clear()
```

References

- http://download.autodesk.com/us/maya/2011help/API/class_m_script_util.html
- http://download.autodesk.com/us/maya/2011help/API/class_m_image.html