Miscellaneous Classes and Tools

Module 8

Autodesk Maya Python API Training

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

Homework Review:

Agenda

- DG / DAG Operations
- Maya Callback System
- Software Shader
- Custom Translator

DG/ DAG Operations

MDGModifier

Used to create remove, and edit nodes in the DG.

This class automatically provides undo and redo for all it’s operations, opposed to implementing all the undo your self.

When each of the functions for editing the DG is called, a record of it is stored from this class.

Important functions in this class:

- doIt()
- undoIt()

DG/DAG Operations

MDGModifier / MDagModifier:

- Facilitate the creation or deletion of nodes and connections
- Provides undo/redo support
- MDagModifier: dag node creation/parenting
- Holds a list of operations. Operations are queued as they are called.
- Does not perform these operations until MDGModifier::doIt() call is issued.

MDGModifier code structure

```
import maya.OpenMaya as OpenMaya
dgMod = OpenMaya.MDGModifier()
dgMod.createNode ( “transform”)
dgMod.commandToExecute(“sphere -n sphere1 -r 1;”)
dgMod.connect(.....)
.....

dgMod.doIt()
```

All these operations won’t get executed until dgMod.doIt()

To undo all these operations, dgMod.undoIt()

Maya Callback System

Maya callbacks allow the user to register functions against specific Maya events.

Maya callback classes:

MMessage: base class, remove callback, query callbacks

MDGMessage	- node added, removed, connected
MNodeMessage	- attribute callbacks
MSceneMessage	- before/after: file open, import, export, etc.
MUiMessage	- UI objects
MEventMessage	- idle, timeChanged, undo, redo, etc.
MConditionMessage	- specific conditions
MModelMessage	- model related messages

MMessage and Callback Functions

OpenMaya.MSceneMessage.addCallback (OpenMaya.MSceneMessage,
MMessage_MBasicFunction , clientData)

OpenMaya.MSceneMessage.kAfterOpen

def MMessage_MBasicFunction (args)

OpenMaya.MSceneMessage.addCallback (OpenMaya.$\boxed{\text{MSceneMessage.kAfterOpen,}}$
afterFileOpenCallback)

MMessage and Callback Functions

OpenMaya.MSceneMessage.addCallback (OpenMaya.MSceneMessage, Mmessage_MBasicFunction , clientData)

```
Id = OpenMaya.MCallbackId()

def afterFileOpenCallback(clientData):
    print "Come to this  afterFileOpenCallback function\n"

def MyCmd_addCallback():
   id =  OpenMaya.MSceneMessage.addCallback(OpenMaya.MSceneMessage.kAfterOpen, afterFileOpenCallback)
```

```
def MyCmd_removeCallback():
    OpenMaya.MMessage.removeCallback( id )
```

Abort operations

OpenMaya.MCallbackId OpenMaya.MSceneMessage addCheckCallback (OpenMaya.MSceneMessage, MMessage_MCheckFunction, clientData)

OpenMaya.MSceneMessage.kBeforeOpenCheck

def MMessage_MCheckFunction (retCode, clientData)

retCode: Result of the function. Provide programmer with options to abort current operations by return false through this variable.

In Python use OpenMaya.MScriptUtil.setBool(retCode, True) since python can't handle pointer or reference.

Abort Operations

```
def beforeOpenCheckCallback(retCode, clientData):
    #Do custom operations, for example, check file versions...
    print “Error: file version is not correct, abort opening operations\n”
    OpenMaya.MScriptUtil.setBool(retCode, False)

def MyCmd_addCallback():
    OpenMaya.MSceneMessage.addCheckCallback(OpenMaya.MSceneMessage.kBeforeOpenCheck, 
beforeOpenCheckCallback)
```

Shading Network and Software Shader

- Different types of software shaders
- Shading group
- Custom Software Shader

Software Shading Node

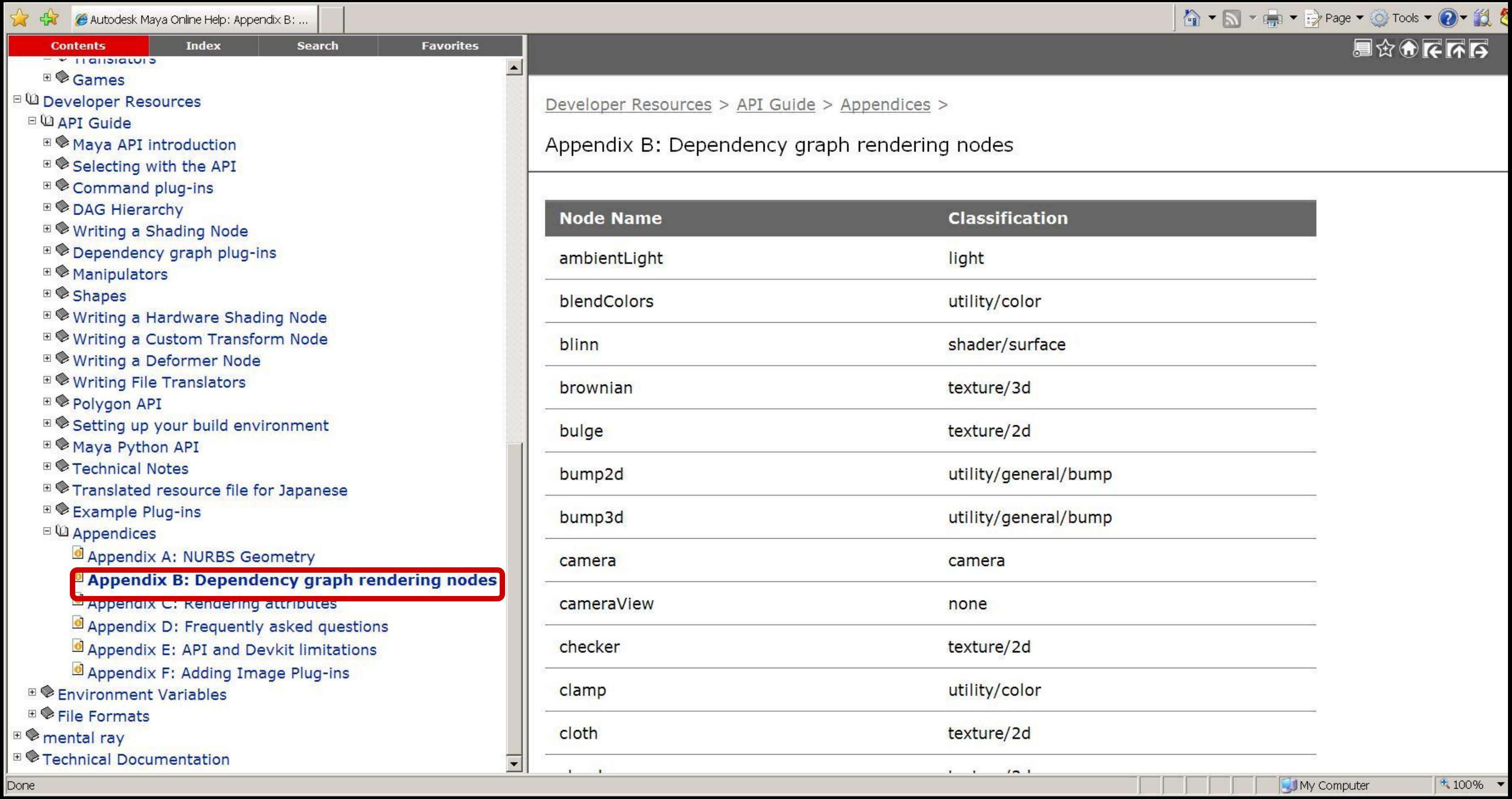
DG Nodes to form Shading Networks

Different Types of Software Shading Nodes:

Type	Frame
Textures	2D Textures, 3D Textures Environment Textures
Materials	Surface Materials, Volumetric Materials Displacement Materials
Lights	Lights
Utilities	General Utilities Color Utilities Particle Utilities Image Planes Glow

Shading Nodes List

Developer Resources -- API Guide -- Appendices – Appendix B: Dependency Graph rendering Nodes



Developer Resources > API Guide > Appendices >

Appendix B: Dependency graph rendering nodes

Node Name	Classification
ambientLight	light
blendColors	utility/color
blinn	shader/surface
brownian	texture/3d
bulge	texture/2d
bump2d	utility/general/bump
bump3d	utility/general/bump
camera	camera
cameraView	none
checker	texture/2d
clamp	utility/color
cloth	texture/2d

Shading Group

- An Object Set
  - A logical grouping of an arbitrary collection of objects, attributes or component of objects
  - Membership is defined by connections:

    whole object is in set:

    node.instObjGroups → objectSet.dagSetMembers

    A part of components are in set:

    node.objectGroups → objectSet.dagSetMembers

- MEL command for “objectSet” node

```
sets -e -add blinn1SG pCubeShape2;

sets –q  -size blinn1SG;
```

Shading Group (Renderable Sets)

- Only renderable elements can be added into Shading Group

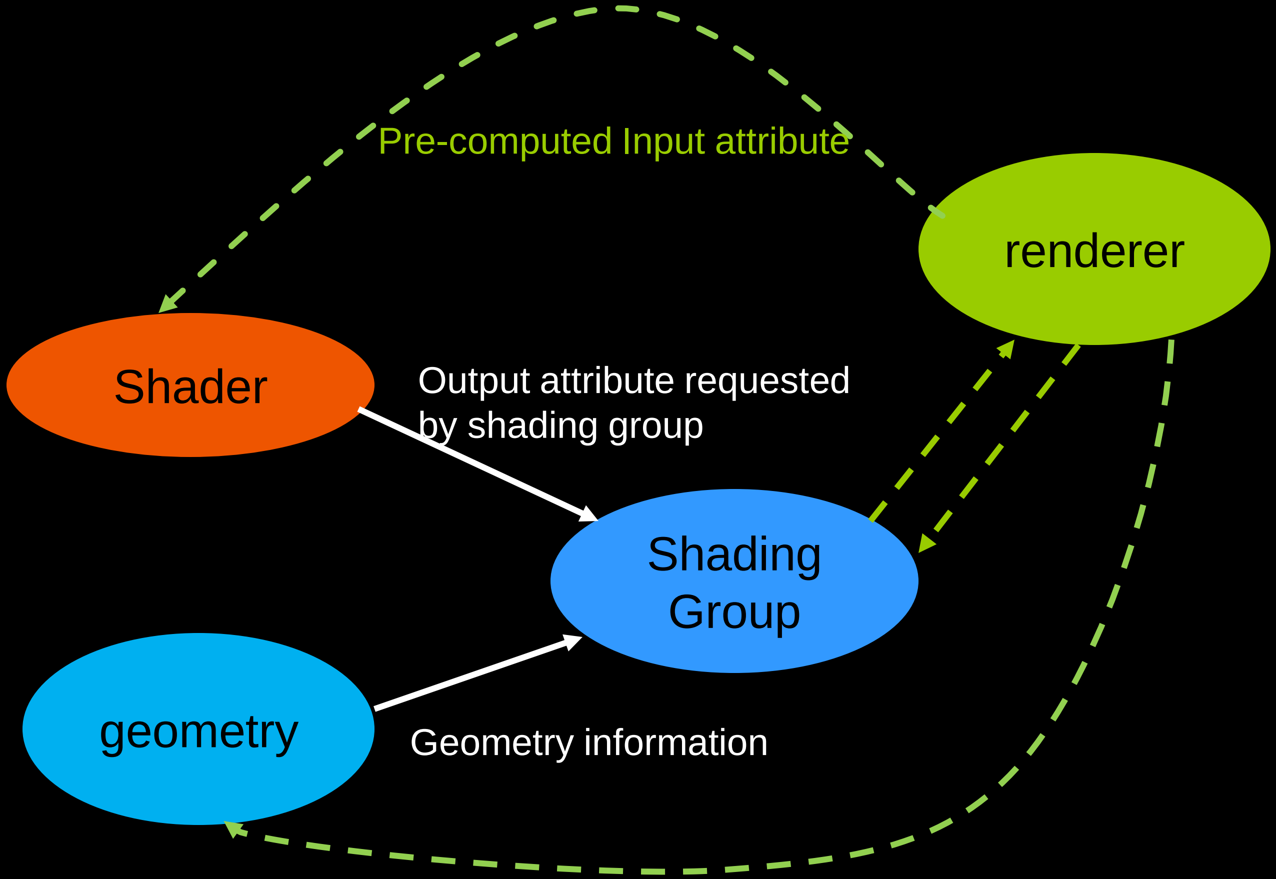
```
sets –q –renderable blinn1SG; //always return true
```

- Every shader has to be connected with a shading group

```
shadingNode -asShader blinn;
// Result: blinn1 //
sets -renderable true -noSurfaceShader true -empty -name blinn1SG;
// Result: blinn1SG //
connectAttr -f blinn1.outColor blinn1SG.surfaceShader;
// Result: Connected blinn1.outColor to blinn1SG.surfaceShader. //
```

- Connection point between geometry and shader

Rendering Network


Pre-computed Input attribute
renderer
Shader
Output attribute requested by shading group
Shading Group
geometry
Geometry information

Custom Shading Node

Registration:

MFnPlugin::registerNode (const MString & typeName, const MTypeId & typeId, MCreatorFunction creatorFunction, MInitializeFunction initFunction, MPxNode::Type type = MPxNode::kDependNode, const MString * classification)

Type	Frame	Classification String
Textures	2D Textures, 3D Textures Environment Textures	“texture/2d” “texture/3d” “texture/environment”
Materials	Surface Materials, Volumetric Materials Displacement Materials	“shader/surface” “shader/volume” “shader/displacement”
Lights	Lights	“light”
Utilities	General Utilities Color Utilities Particle Utilities Image Planes Glow	“utility/general” “utility/color” “utility/particle” “imageplane” “postprocess/opticalFX”

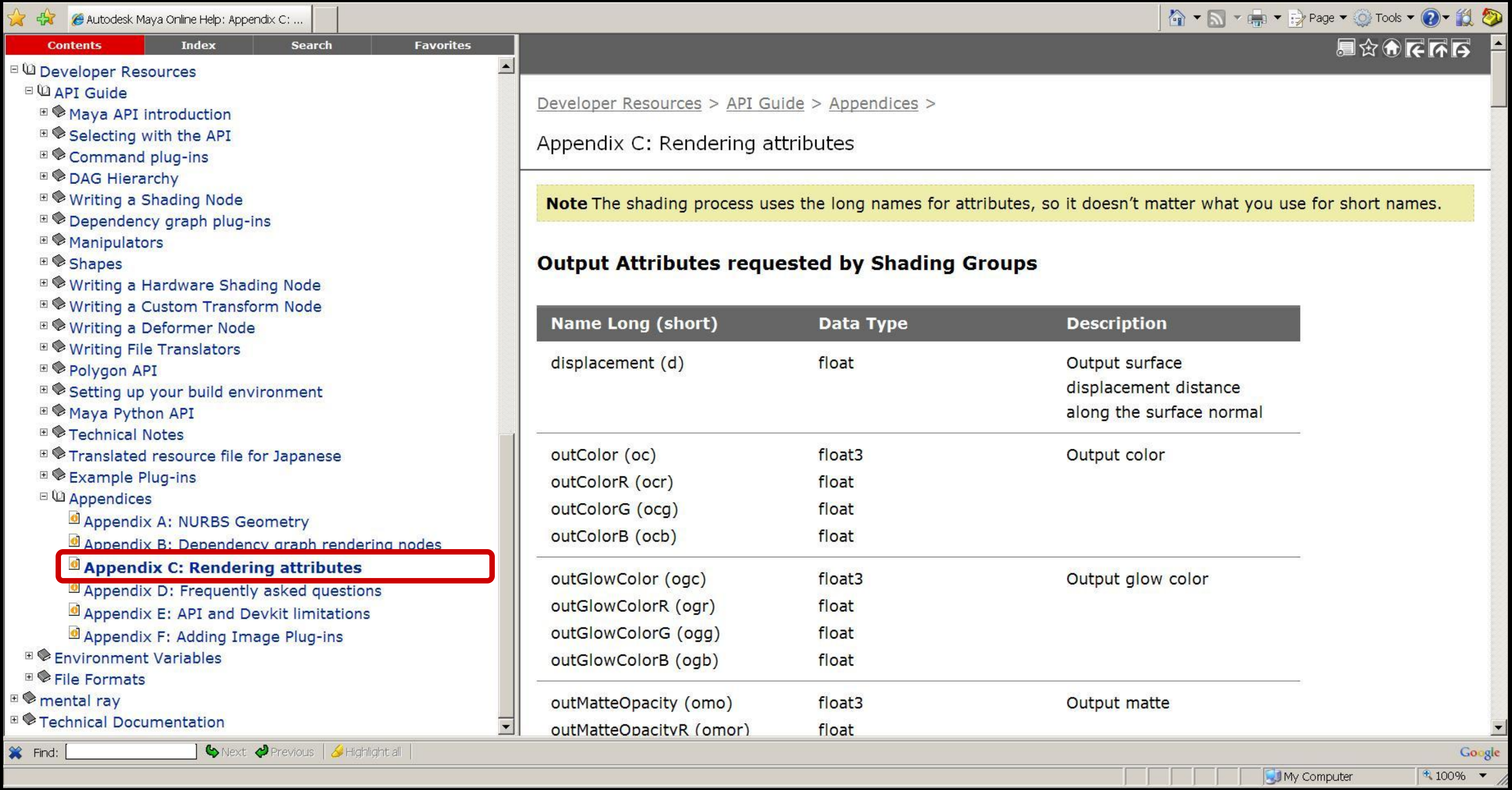
Custom Shading Node

Code Structure:

```
myShaderId = OpenMaya.MTypeId(0x00001)

class myShader (OpenMayaMPx::MPxNode):
   def __init__(self):
      OpenMayaMPx.MPxNode.__init__(self)

   def compute(self,plug,data):
      # code ...

def myShader_creator():
   return OpenMayaMPx.asMPxPtr( myShader() )

def myShader_initialize():
    #Input attributes
    # …….
    #Output attributes
    # …….
```

Software Shading Node

Rendering Attributes



Autodesk Maya Online Help: Appendix C: ...

Contents | Index | Search | Favorites

Developer Resources > API Guide > Appendices >

Appendix C: Rendering attributes

Note The shading process uses the long names for attributes, so it doesn't matter what you use for short names.

Output Attributes requested by Shading Groups

Name Long (short)	Data Type	Description
displacement (d)	float	Output surface displacement distance along the surface normal
outColor (oc)	float3	Output color
outColorR (ocr)	float	
outColorG (ocg)	float	
outColorB (ocb)	float	
outGlowColor (ogc)	float3	Output glow color
outGlowColorR (ogr)	float	
outGlowColorG (ogg)	float	
outGlowColorB (ogb)	float	
outMatteOpacity (omo)	float3	Output matte
outMatteOpacityR (omor)	float	

Find: Next Previous Highlight all

My Computer 100%

Shading node icons for Hypershade

XPM format: 32x32

Icon name: preface "render_".

lambertShader.mll: render_lambertShader.xpm

XBMLANGPATH: Put the icons in one of the directories specified in your XBMLANGPATH.

Custom Translator

- Transmitting data in your production pipeline
- Define custom file format
- Decide what contents you want to export
- Plug-in vs. Standalone Application

Custom Translator Plug-in

Derived from MPxFileTranslator :

- Maya consistent UI (File->Export, File->Import)
- register your extension with Maya
- implement a reader and writer in the same plug-in

Register MSceneMessage callbacks to execute operations before/after import/export

- MSceneMessage::kBeforeExport
- MSceneMesage::kAfterExport
- MSceneMessage::kBeforeImport
- MSceneMessage::kAfterImport

Devkit Example: lepTranslator (C++)

Adds the new file format “Lep” to the file manipulation dialogs

An “Lep” file is an ASCII file with a first line of “<LEP>”. The remainder of the file contains MEL commands that create one of the primitives: nurbsSphere, nurbsCone and nurbsCylinder, as well as move commands to position them.

MPxFileTranslator

- Custom file format:

```
static MString myfileExt("lep");

MString MPxFileTranslator::defaultExtension()
        return custom file format

MPxFileTranslator::identifyFile()
        determine whether it is the type supported by the translator
```

MPxFileTranslator: read & write

MPxFileTranslator::canBeOpened()

decide whether it is an importer or exporter

bool MPxFileTranslator::haveReadMethod()

MStatus MPxFileTranslator::reader (const MFileObject & file, const MString & optionsString, MPxFileTranslator::FileAccessMode mode)

Mode: the method used to read the file – open or import

bool MPxFileTranslator::haveWriteMethod()

MStatus MPxFileTranslator::writer (const MFileObject & file, const MString & optionsString, MPxFileTranslator::FileAccessMode mode)

Mode: the method used to write the file - save, export, or export active

Example: lepTranslator

```
magic =MString (“”)

class LepTranslator (OpenMayaMPx.MPxFileTranslator):
        def __init__ (self):
           OpenMayaMPx.MPxFileTranslator.__init__(self)
        def reader ( mfileobject, optionsString, fileAccessMode):
           #code …
        def writer ( mfileobject, optionsString, fileAccessMode):
           #code …
        def haveReadMethod ():
           return True
        def haveWriteMethod ():
           return True
        def defaultExtension ():
        def canBeOpened():
        def identifyFile (fileName, buffer, size):
        def getPosition(transform, tx, ty, tz ):
```

MPxFileTranslator

Register custom file translator:

```
def initializePlugin( mobject ):
    mplugin = OpenMaya.MFnPlugin (mobject)
    try:
        mplugin.registerFileTranslator( "Lep", "lepTranslator.rgb", LepTranslator_creator,
                "lepTranslatorOpts", "showPositions=1", True )
    except:
        raise
```

UI for MPxFileTranslator

Export Options: a text string with format:

varName1=value1;varName2=value2;...

```
lepTranslatorOpts.mel

global proc int lepTranslatorOpts ( )
{
   ……
}
```

In lepTranslator example, the option string is “showPositions = 1” or “showPositions = 0”

Workshop Session

Example: setUpTransCircle

transCircle node

setUpTransCircle: In this example, we create a custom command, and simulate the same functionality of the MEL operations we used in “transCircleNode” project, which set up the transCircle node.

Here are the commands you need to simluate:

```
createNode transCircle -n circleNode1;
sphere -n sphere1 -r 1;
sphere -n sphere2 -r 2;
connectAttr sphere2.translate circleNode1.inputTranslate;
connectAttr circleNode1.outputTranslate sphere1.translate;
connectAttr time1.outTime circleNode1.input;
```

Example: sceneMsgCmd

sceneMsgCmd: this example register several callbacks for scene messages such as MSceneMessage::kBeforeOpen and MSceneMessage::kAfterNew, it also shows how to abort the operation by setting retCode.

Autodesk