Python MAYA API

Preproduction Techniques

Samantha Comeau, sjcomeau@wpi.edu
Jason Abel, jabel@wpi.edu
Matthew Schueler, mjschueler@wpi.edu
Yo Karita, ykarita@wpi.edu
David Tang, dytang@wpi.edu

28 April, 2018

Table of Contents

Introduction

The goal of this project was to use the Maya Python API to create a IK/FK blend plugin for artists to use during the pre-production process. Maya is an Autodesk product that has two methods of interaction: the Maya Commands or the Maya API. For this project we chose to use the Maya Python API to develop our prototype because it was the easiest for beginners to understand. An overview of how Maya is layout is shown below.

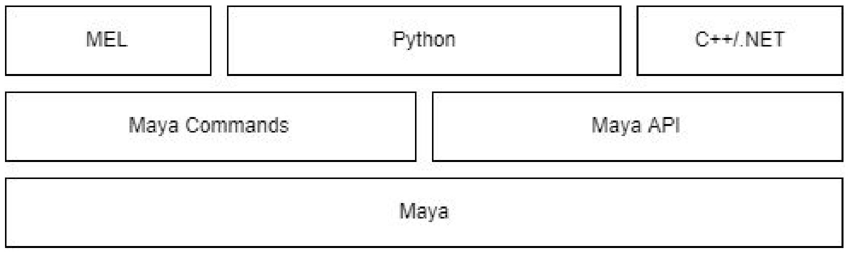

Figure 1. Overview of Maya interactions.

Acknowledgements and Logistics

Our project is stored on GitHub at https://github.com/sjcomeau43543/MayaPlugins. The core development team consisted of Samantha Comeau, Jason Abel and Matthew Schueler. We'd like to thank Chayan Vinayak Goswami for the starter code and tutorials that got us off the ground with this project.

Maya Python API

The Maya Python API is based on the C++ API and allows the API to be used through the Python scripting language [1]. It is important to understand the difference between a plug-in and a script. For Maya, a plug-in is required if you are planning to extend Maya's capabilities. Scripts are solely used for one time operations, or development of plug-ins.

For information on the descriptions of functions the Maya help source is linked here: https://help.autodesk.com/view/MAYAUL/2017/ENU/?guid=__files_API_Introduction_htm . This document also covers the basics on integrating with Maya and provides basic definitions of key terms.

Key Terms

DAG

DAG stands for Directed Acyclic Graph. As a graph is an interconnection of nodes which each contain a set of information, so a DAG is a graph where the connections between two nodes have a direction (directed) and by following these connections, there is no way to return to a previous node (acyclic).

A directed tree is a good example of this. A directed tree is a hierarchical data structure where a series of nodes branch out from a single node, and each of these nodes can have their own set of nodes, much like a tree.

Within Maya we use two types of nodes: transform (the outer node, which defines the translation, rotation and scale) and shape (the inner node, which defines the shape or dimentry)

DAG Path

A path from the root to the object and a node by following the directed connections defined. Within Maya, we can get this by using maya.OpenMaya.mDagPath(). Print using fullPathName(). A DAG path is the path of nodes from the top of the DAG tree. This shows the hierarchy of nodes.

DG

A Maya DG is a dependency graph (directed cyclic graph).

MObject

MObject is short for Model object, which is a special handle to allow access to Maya's internal modeling, lighting and rendering. It can be referenced using maya.OpenMaya.MObject.

MSelectionList

Specialized selection list of objects, where objects in the scene can be selected, then added or removed from the list. As with other lists in programming, the indexes (numbers used to represent the items in the list) start from 0, and increments by 1 for each item. This can be referenced using maya.OpenMaya.MSelectionList. Add one object using .add() and | signifies the tree structure.

MFN

MFN Stands for Model Function Set. A MFN is a class that provides a set of functions based on the specified input type. MFnMesh is used for for meshes and can be printed using fullPathName(). A MFnDependencyNode is used for dependency nodes and you can access the name of the dependency nodes using name(). To get the nodes that a node is connected to use getConnections(MPlugArray) which will return an array of plugs.

MPlug

An MPlug offers functions for operating on plug type data. Plugs are handles to create, modify, or access attributes of nodes. Plugs can be divided into two categories: Network plugs are plugs of dependency nodes and Non network plugs are user-defined plugs. To find the plugs that it's connected to use connectedTo(destMPlugArray, asDst, asSrc) which returns you an MPlugArray. To get the node that exists in it use yourObject.node(). To get the dependency node - OpenMaya.MfnDependencyNode(mplug.node()) which returns the dependency node. To find the plug within a scene use findPlug("width").asInt() to get width or subdivisionWidth.

MPlugArray

MPlugArray is a class that provides methods for operating on an array of plugs.

Callbacks

Maya callbacks are important to understand for creating plugins in Maya. It is a way of linking a specific function to a specific events. It allows us to watch the occurrence of the event at all times. For example, if someone clicks on a node in the scene, that can generate a callback which will run the respective function. It also allows us to registers functions to specific events. A good comparison is a Java Listener.

An event callback is used for a selection change and a DG callback is used for the addition, deletion and creation of DGnodes.

The base classes that are used for creating callbacks are as follows. All lie under the struction MMessage which links an event to a function.

MMessage

A MMessage is the function that links events to functions. It can be created or removed at any point.

MAnimMessage - Used for animation curve editing, keyframe editing, baked result change
MCommandMessage - Can be used for MEL command creation
MDGMessage - Dependency graph messages, node added/removed, correction established/removed
MDAGMessage - For dependency graphs, node switching or making as children/parent/instance edited or removed
EventMessage - Similar to a scriptJob command
LockMessage - Watch for locked messages, plug or node are locked or unlocked, plug - values can’t be changed, node - reparented, renamed, deleted
NodeMessage - Dependency node message , attribute - added/removed, plug of a node is dirty, name of a node is changed
SceneMessage - Runs before or after: New scene, existing scene, reference is loaded/unloaded, maya is initialized / existed

IK/FK Plugin Development

How to run

In order to import a plugin, you need the python file containing the plugin (gui_script.py). All of our work and version control can be found on our github page:

https://github.com/sjcomeau43543/MayaPlugins

Github Page

This is where you can download our files. Once you have the file, then you will want to go to Windows > Settings/Preferences > Plug-in Manager. From there, a new popup window will appear. You should then click browse and find the python file of the plugin and open it in the plug-in manager.

Once the plugin is loaded in, a new popup window should appear with a button that says "Update List" and a list of all the IKHandles currently in the scene with a checkbox next to each one.

How To Use

If you want a certain arm to be in IK/FK blend mode, you will need to check the box next to the name of the IKHandle that is connected to that arm. This will automatically make the IKHandles work in IK and the joints work in FK. In order to turn this off for any arm, make sure the IKHandle attached to that arm is unchecked in the plugin popup window. If you add any arm later on after importing the plugin, you will need to hit the "Update List" button in order for the IKHandle for the new arm to appear in the pop-up window.

Depending on which part of an arm is selected right before unchecking a box will determine what mode the arm will be in after. If you click on a joint first and then uncheck the IKHandle in the pop-up window, the whole arm will stay in FK mode. If you clicked on the IKHandle before unchecking it in the pop-up window, it will keep the arm in IK mode.

Development of the IK/FK Blend Plugin

For the IK/FK blend part of the plugin, we used the OpenMaya, OpenMayaMPx and sys packages in order to get the objects so we could modify them in the scene. This part of the plugin is always running in the background by using a callback on "SelectionChanged" which will call a function every time you select something new in the maya scene. For example, if you click on a joint, the function will be called. If you then click on something else, it will call again.

Once the function is called, it uses the function getActiveSelectionList() which will produce a list of MObjects of the current nodes that are selected in the Maya scene. From there, it will go through all the objects that are connected to the one selected and figure out what the type of object is that is selected and will change the "mode" of the arm based on what is selected. If the IK effector, the control curve, or if what is selected is connected to the IK effector, the mode becomes IK, otherwise the mode will be in FK.

Once the mode is set, if the mode is IK, it will set the blend attribute to 0 and will set the visibility of the control curve to false. If the mode is FK, it will set the blend attribute to 1 and set the visibility of the control curve to true.

Development of a UI Widget

To make the UI for this plugin, we used PySide2 (a wrapper for Qt, which was developed for C) and shiboken2 (a wrapper for sip). These two libraries gave us all the components to create the UI used by the plugin. It uses a VBoxLayout to arrange everything vertically, and fills the layout with a checkbox and label for each IKHandle that is found in the scene. The locations of the contents are handled automatically by the layout, which allows the window to be resized and all the components will move relative to each other to stay evenly spaced. This also allows for the integrity of the window to be consistent across different monitor resolutions. For example, the window will look the same on a 4K monitor as it would on a 1920x1080 monitor.

We chose to have an “update” button so that there would not constantly be a script running in the background. The IK/FK blend is activated for each handle as soon as the box is checked. We chose to do it this way because it is more intuitive for the user to simply check a box indicating whether they want the plugin to be active for different nodes in the scene.

Once the Plugin is initialized, it will immediately run a function to acquire all the IKHandles in the scene. This uses MItDependencyNodes() to iterate through all the dependency nodes in the scene. If any of them are IKHandles, we put the the name of the IKHandle into a global array that will be used by the UI to print it out in the pop-up window. Additionally, anytime the “Update List” button is pressed, it will re-run the function to gather all the IKHandle names.

To set what handles are to use the IK/FK blend through the UI we use the checkbox to indicate whether it should be used or not. Once a user changes the value of a checkbox a function is run. Based on whether the checkbox is checked (indicated by a 2) or the checkbox is unchecked (0), the corresponding node is either added or deleted from a global list of active IK/FK blend nodes.

Once the active IK/FK blend nodes list is updated, the IK/FK part of the plugin will additionally check to see if the active IKHandle name matches any of the selected IKHandles. If any names do match, it will apply the IK/FK blend to the arm. Otherwise it will leave the arm as it is.

Works Cited

[1]. “AutoDesk MAYA Help 2017.” Maya 2017, Autodesk, help.autodesk.com/view/MAYAUL/2017/ENU/?guid=__files_API_Introduction_htm.
[2] chayanvinayak. “Chayan Vinayak.” YouTube, YouTube, www.youtube.com/channel/UCwtWCujRCWK9ba3f474sTOQ.
[3] “Cult of Rig » Rigging by First Principles.” Cult of Rig, www.cultofrig.com/.
[4] “QCheckBox Class Reference[QtGui Module].” QCheckBox Class Reference, Riverbank Computing LD, pyqt.sourceforge.net/Docs/PyQt4/qcheckbox.html.
[5] Maya. “Creating a Simple UI in Maya Using PyQt.” Dan Ostrov, danostrov.com/2012/10/27/creating-a-simple-ui-in-maya-using-pyqt/.