Maya API Overview – Day 2

Autodesk Maya Python API Training

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

Autodesk

Day 1 Questions:

1. Where does PyMEL a true OOP version of Mel and Python fit in all as we go forward?
2. Can you provide us with PyQt for compiled for Maya 2013 x64?
3. Can you tell us if these classes will be available for download or for staggered viewing (for those of us in a troublesome time zone)?
4. Is there a way to set up default imports for the script editor? Every time I open a new python tab I have to do a series of imports.
5. Also does userConfig.py do the same thing as userSetup.py?
6. where can I get tutorials on how to start using Maya API with c++
7. Is it ok to use the 2012 educational version since the Autodesk education site does not have version 2013 yet?

userConfig.py or userSetup.py File

- Create a userConfig.py or userSetup.py file and add it to somewhere on your path (PYTHONPATH)
- You can add imports and Python code you would like
- Changes will show up in top level context

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


Autodesk®

Scripts Path

- **PYTHONPATH** defines search path for scripts
- Includes standard Maya script paths
- Update at startup via Maya.env file
- Update from Python:

```
import sys
sys.path.append( "C:/scripts" )
```

Maya Environment Variable

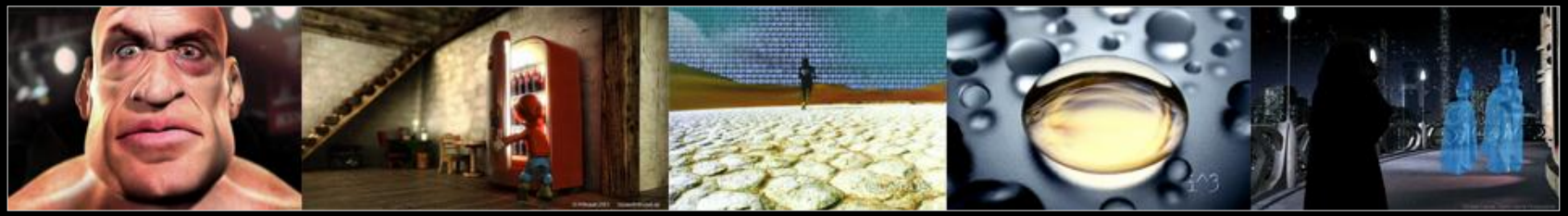
- Text file: Maya.env
- Recommended approach of setting environment variable
- Syntax:
  - PYTHONPATH = C:\Users\wengn\Documents\test
- Save it to
  - C:\Users\wengn\Documents\maya\2012\Maya.env
-

Other Useful Modules (externally)

os	**process management, environment variables, etc**
os.path	file path routines: join, split, etc.
sys	shell arguments (argv), python globals
re	regular expressions
urllib2	URL access
pickle	object archiving
getopt	command line argument parsing
math	assorted math routines and constants

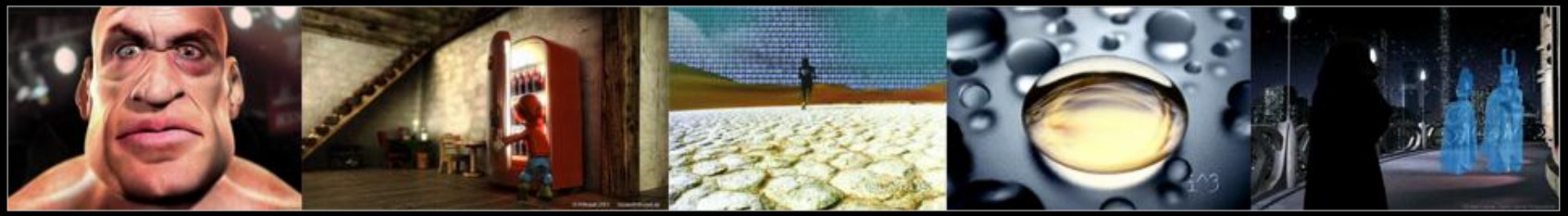
Agenda

- Maya Architecture Overview
- Maya API Introduction
- Plug-in Development



Autodesk®

Maya Architecture Overview



Autodesk®

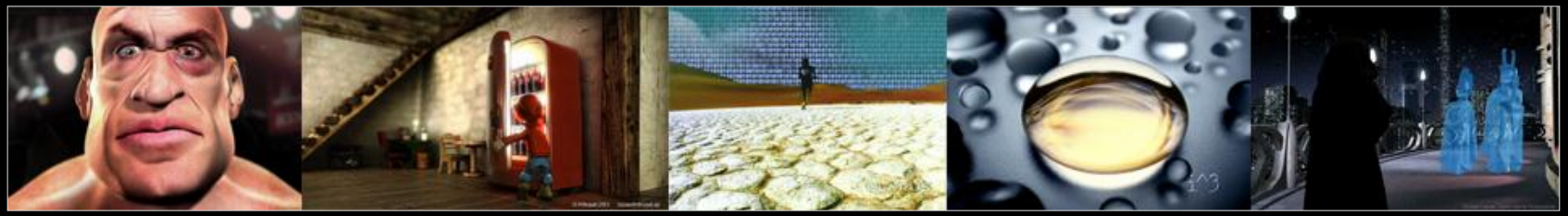
Maya Architecture Overview

Two vital concepts of Maya architecture

1. Dependency Graph

2. Command Architecture

Dependency Graph

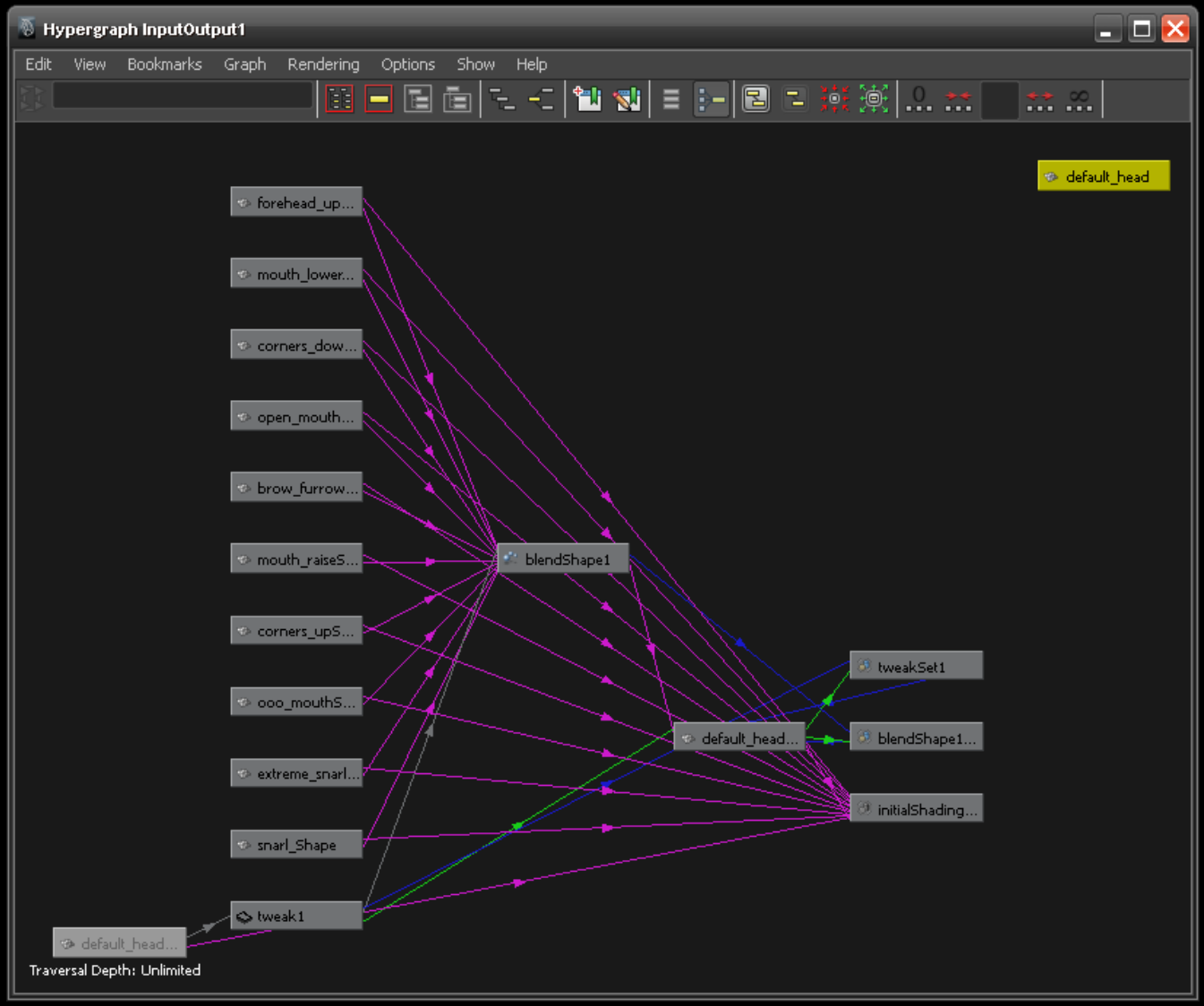


Autodesk®

Dependency Graph

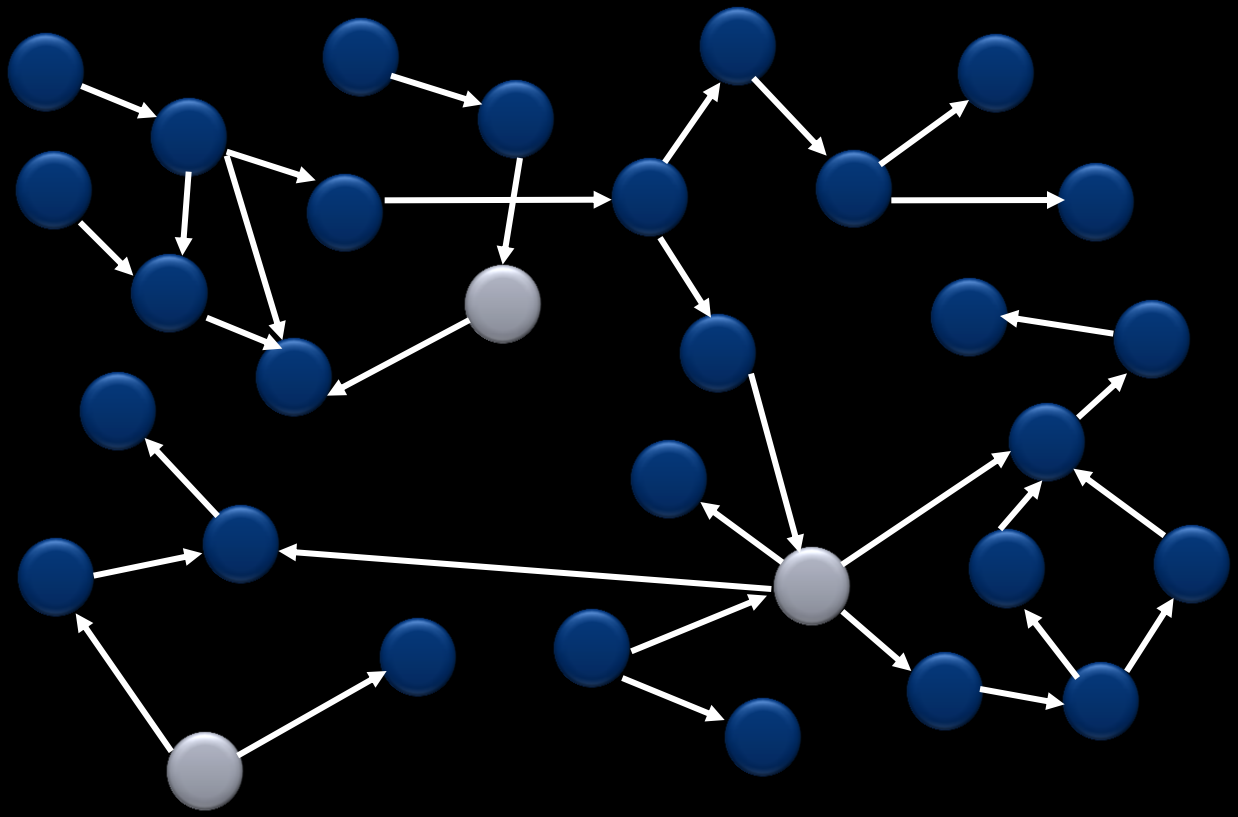
- Control system for Maya
- Glue that holds together disparate operations and lets them work together seamlessly
- Foundation of the Maya file format

Maya Hypergraph



Autodesk®

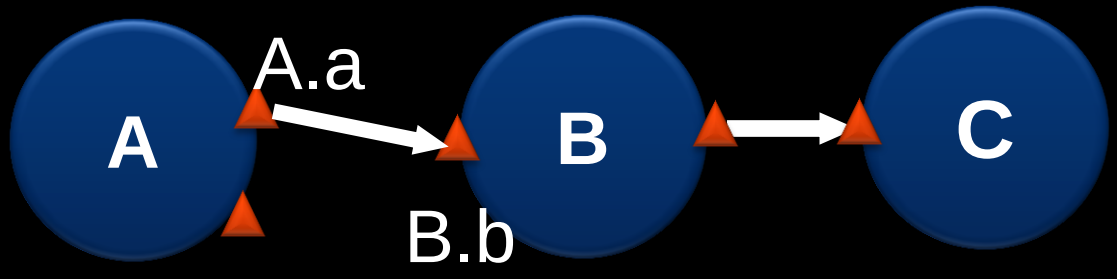
Dependency Graph



Autodesk®

Dependency Graph

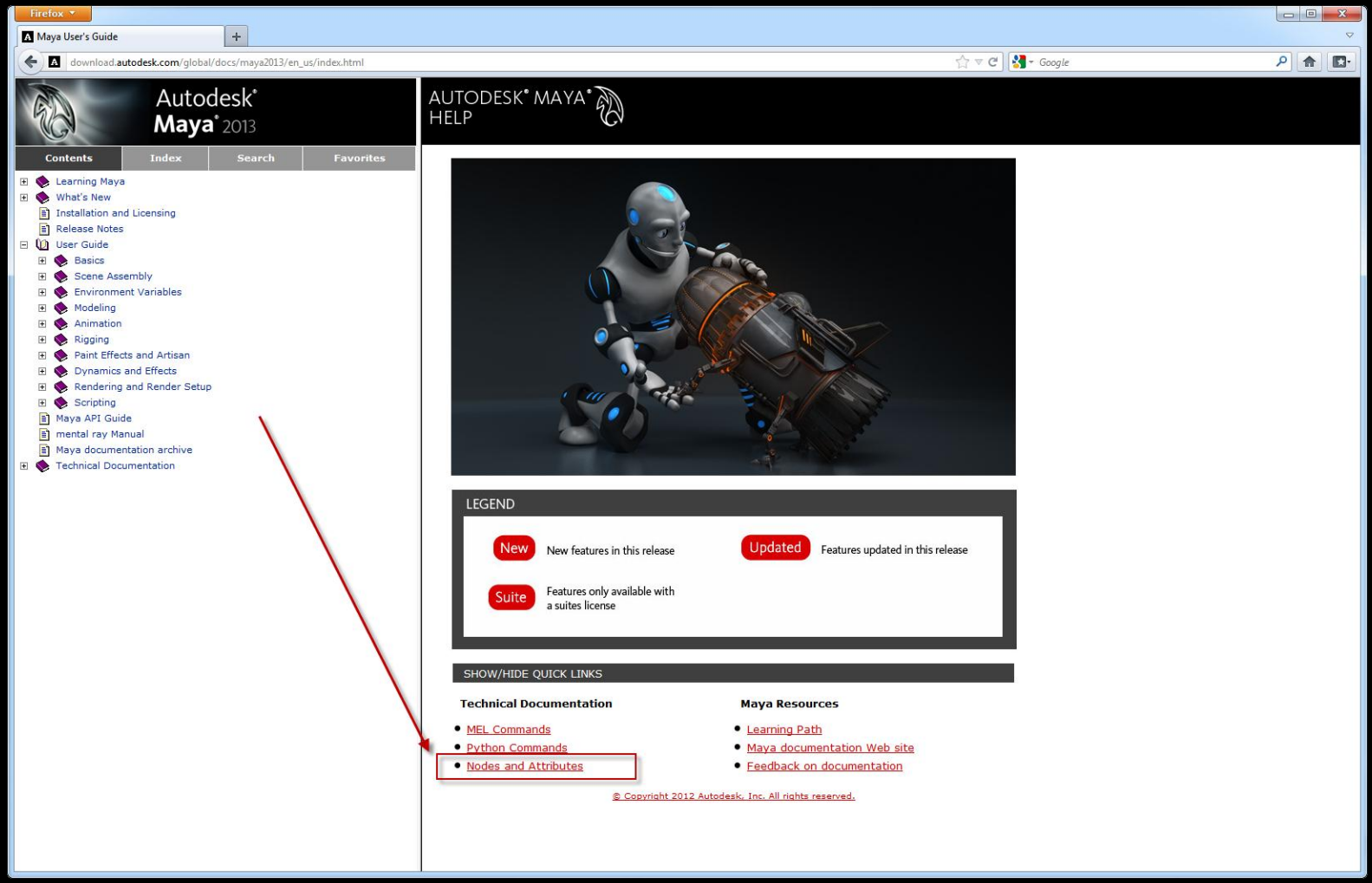
- A collection of nodes that transmit data through connected attributes



Dependency Graph

- Example: revolving surface

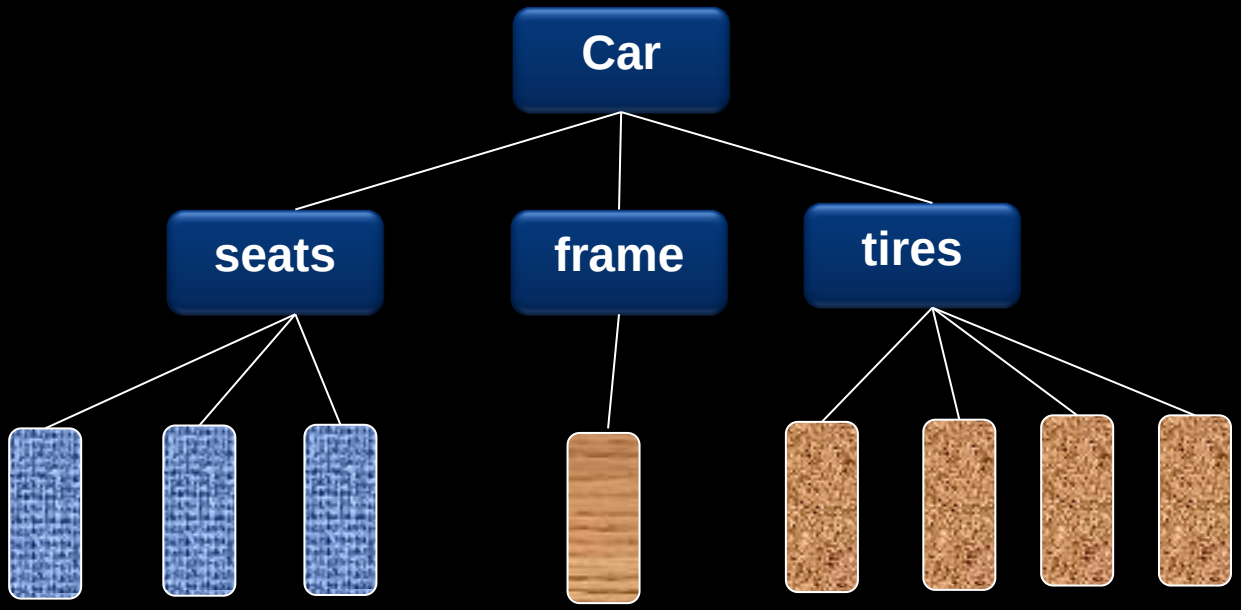
Maya Node Documentation



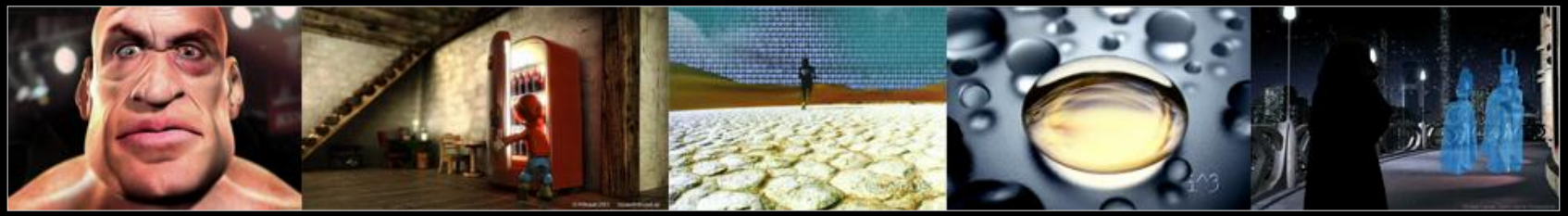
Autodesk

DAG (Directed Acyclic Graph)

- DAG nodes are special dependency graph nodes that form a scene hierarchy (parenting).

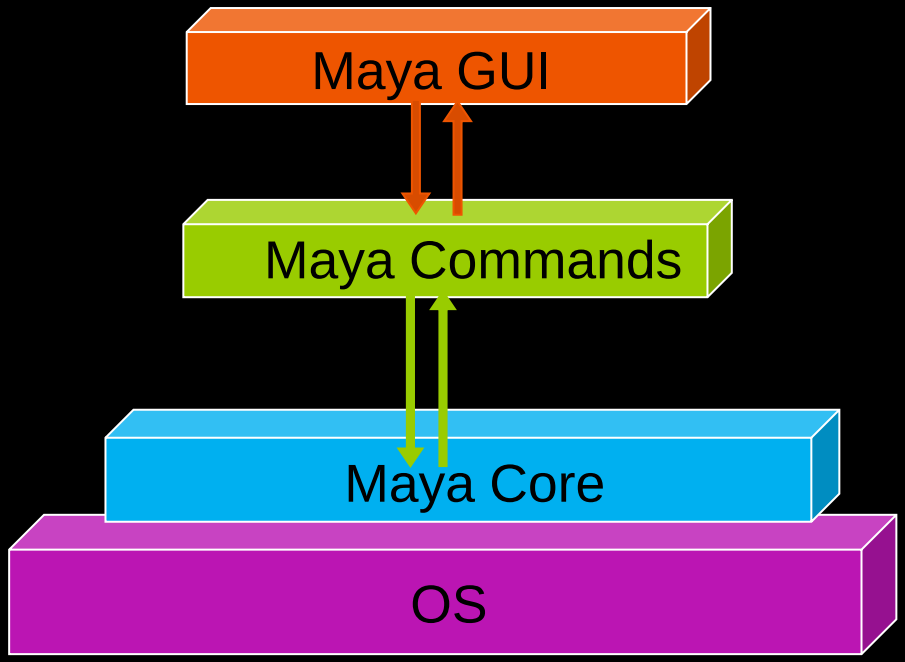


Command Architecture



Autodesk®

Maya Command Architecture



Maya Embedded Language

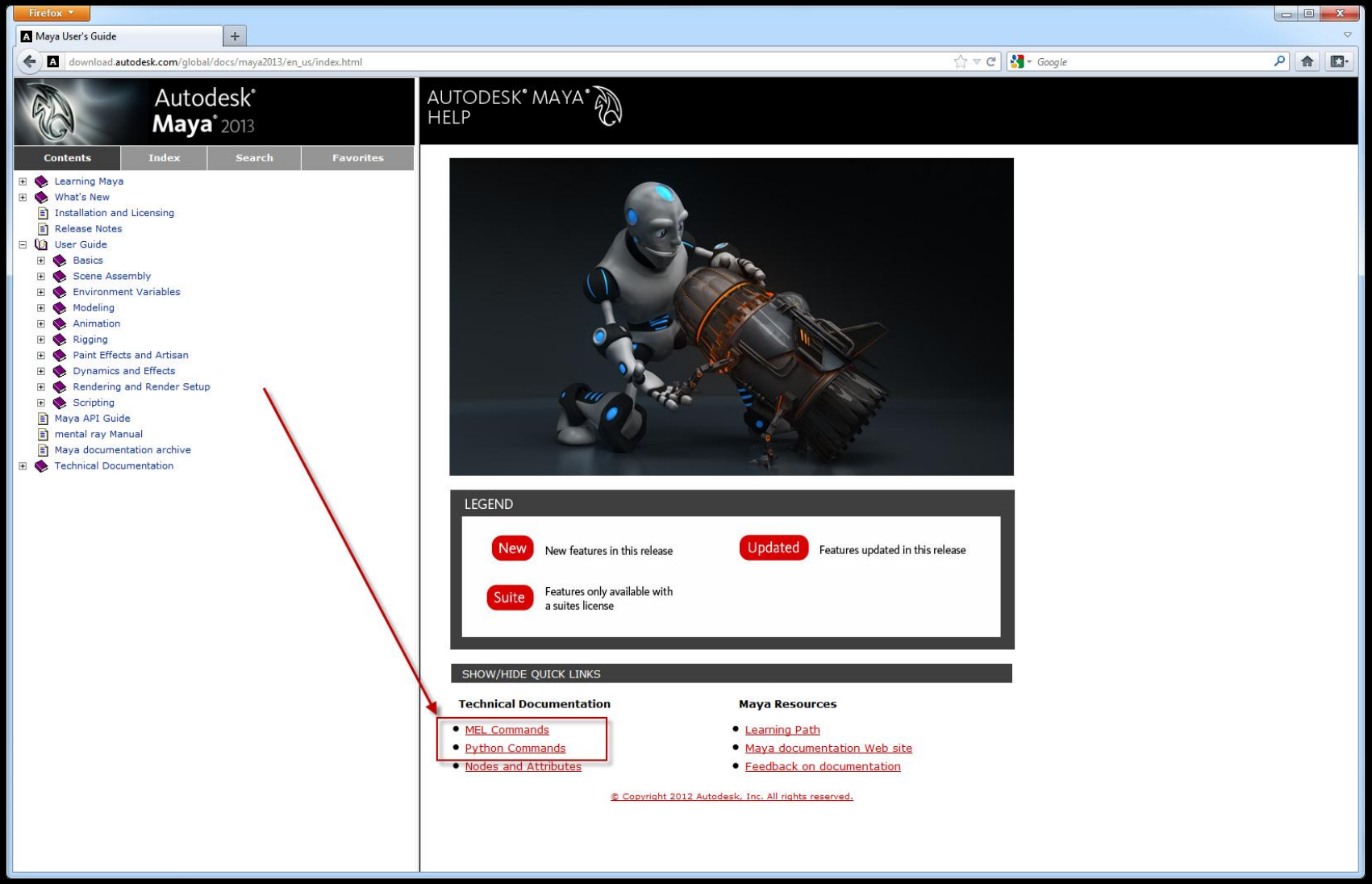
- MEL is the language of the Command Architecture
- MEL is deeply embedded in the Maya Architecture
- Maya Command Examples

```
select –replace myCone;  // Replaces selection list with myCone.
connectAttr myCone.ty mySphere.ty;   // Connects translateY attributes
headsUpDisplay –s 0 –b 0 –label “myHUD” myHUD;
```

- To view the syntax for a given command use `help`

```
help headsUpDisplay;
// Result:
// Synopsis: headsUpDisplay [flags] String
// Flags:
//  -e   -edit
//  -q   -query
…
```

Maya Command Documentation



Autodesk®

Create, Query and Edit Mode

- Create Mode

```
string $window = `window -title “TestWindow"
                 -iconName “testWnd" -widthHeight 200 55`;
showWindow $window;
```

- Query Mode

```
window -query -widthHeight $window;
```

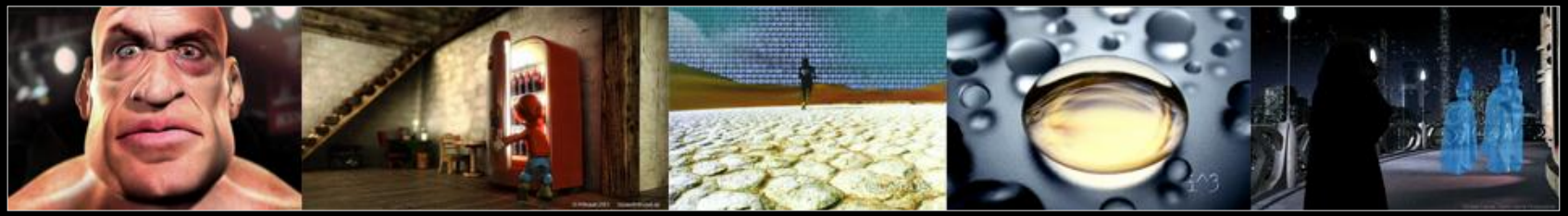
- Edit Mode

```
window -edit -widthHeight 100 100 $window;
```

Execute MEL commands

- Command Line & Script Editor
- Record MEL commands
- Script Files: *.mel
  - Script Path: MAYA_SCRIPT_PATH
  - whatIs: to find a global proc or internal built-in command
  - source: let Maya know a script has updated
  - rehash: let Maya rescan the script paths

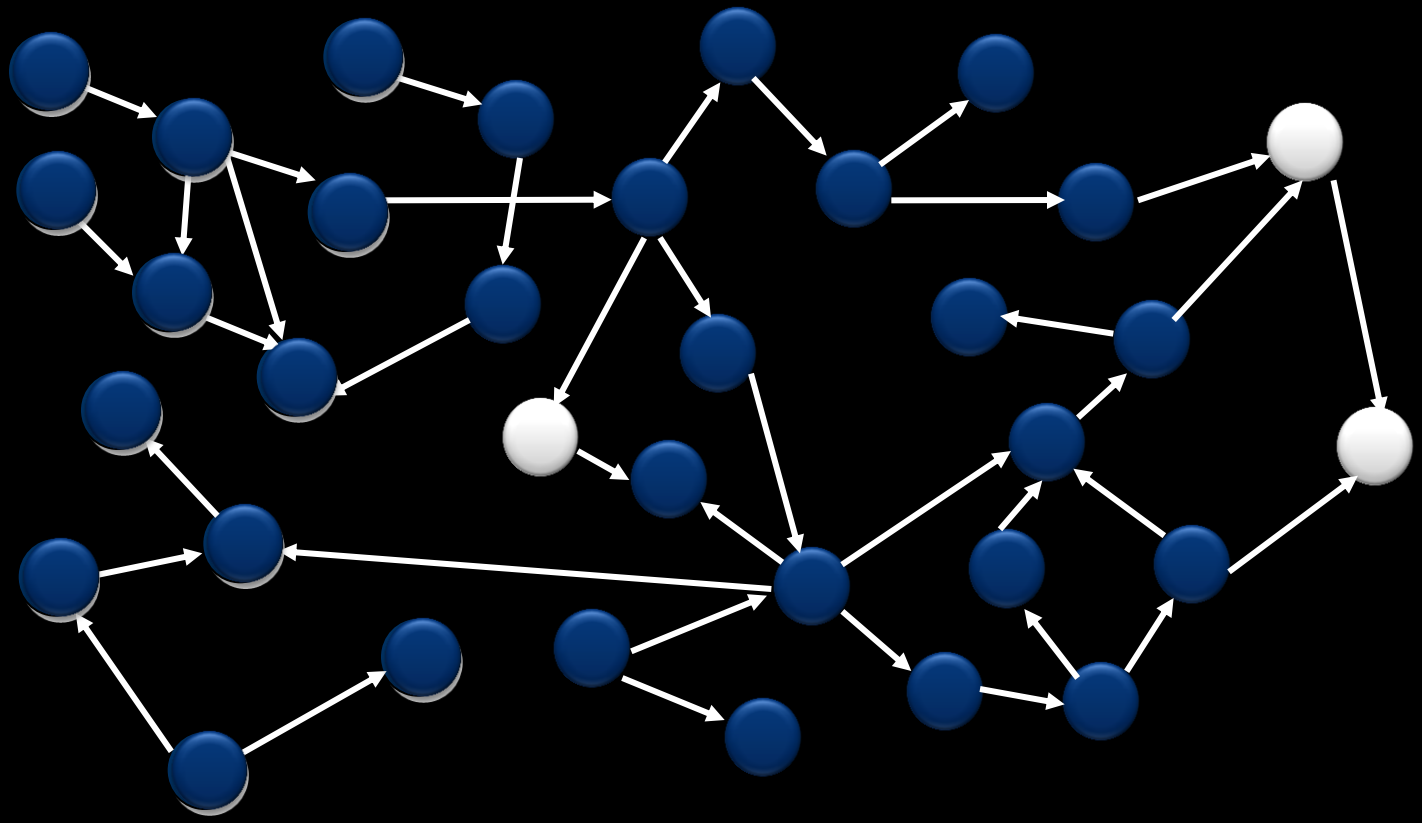
Customizing Vital Concepts



Autodesk®

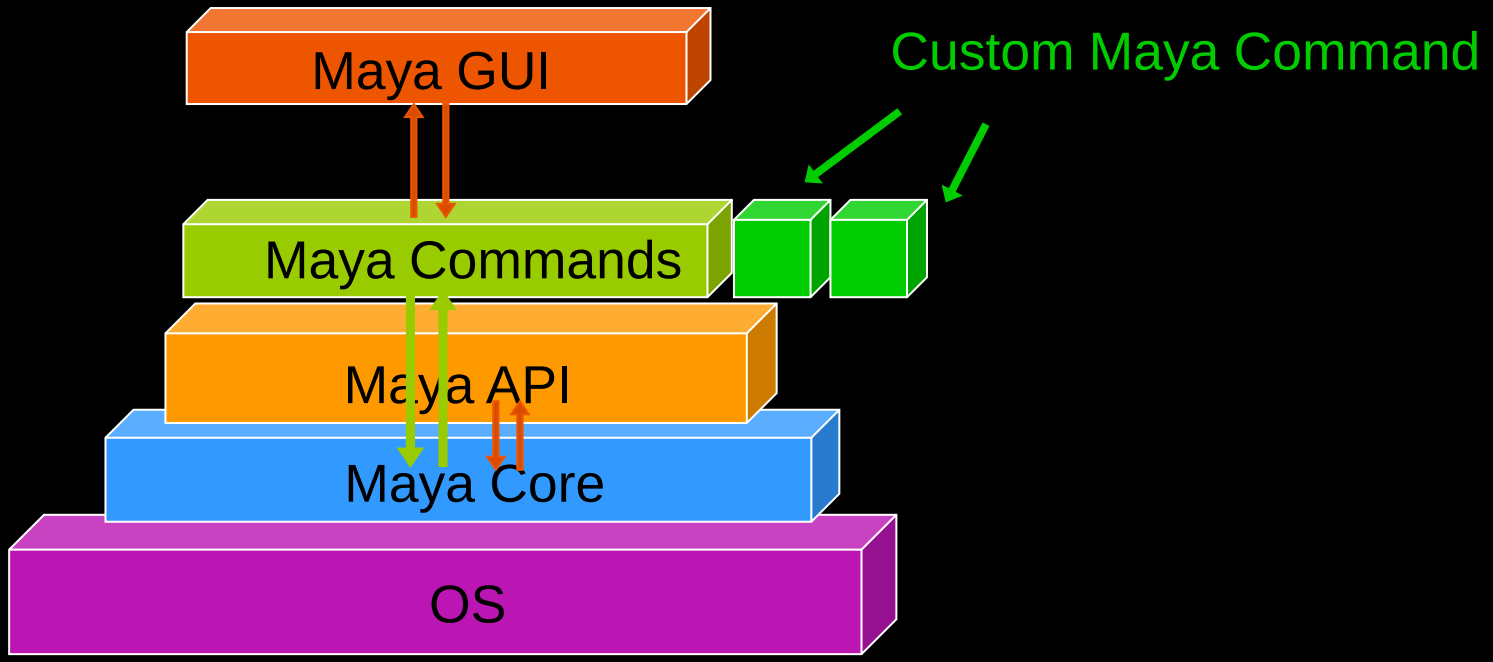
What do we need API for?

New Custom Node

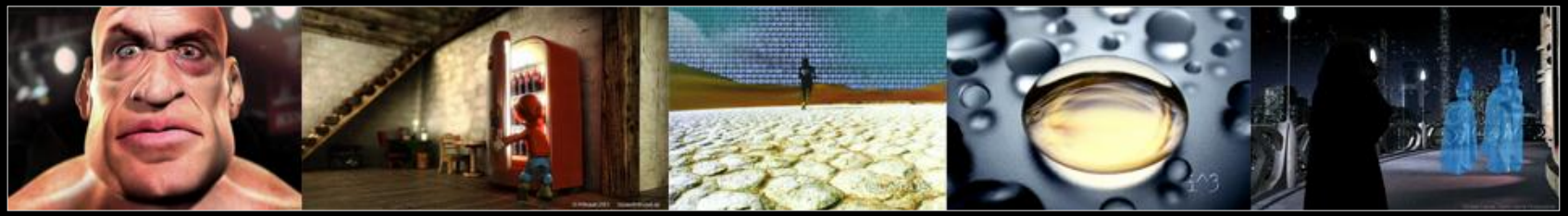


Autodesk®

What do we need API for?



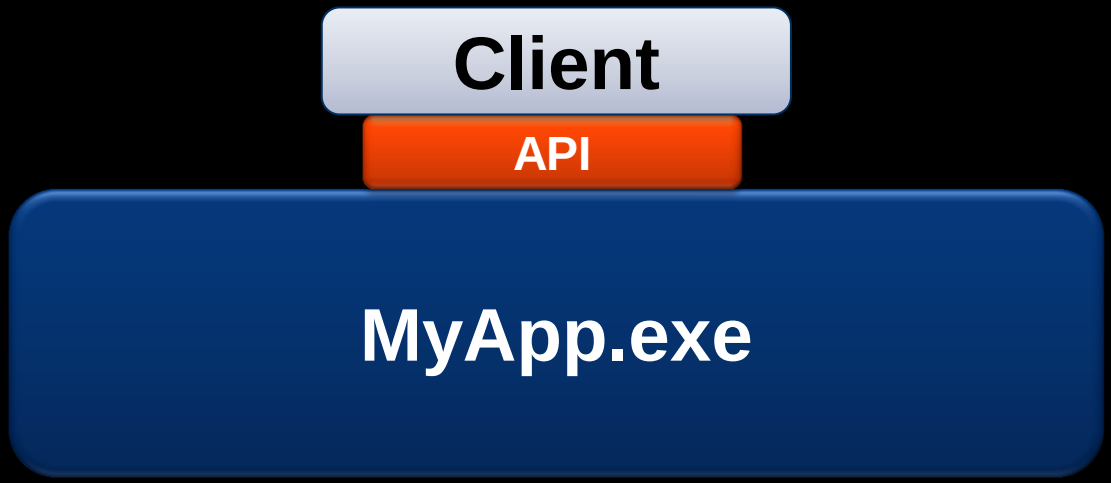
Maya API Introduction



Autodesk®

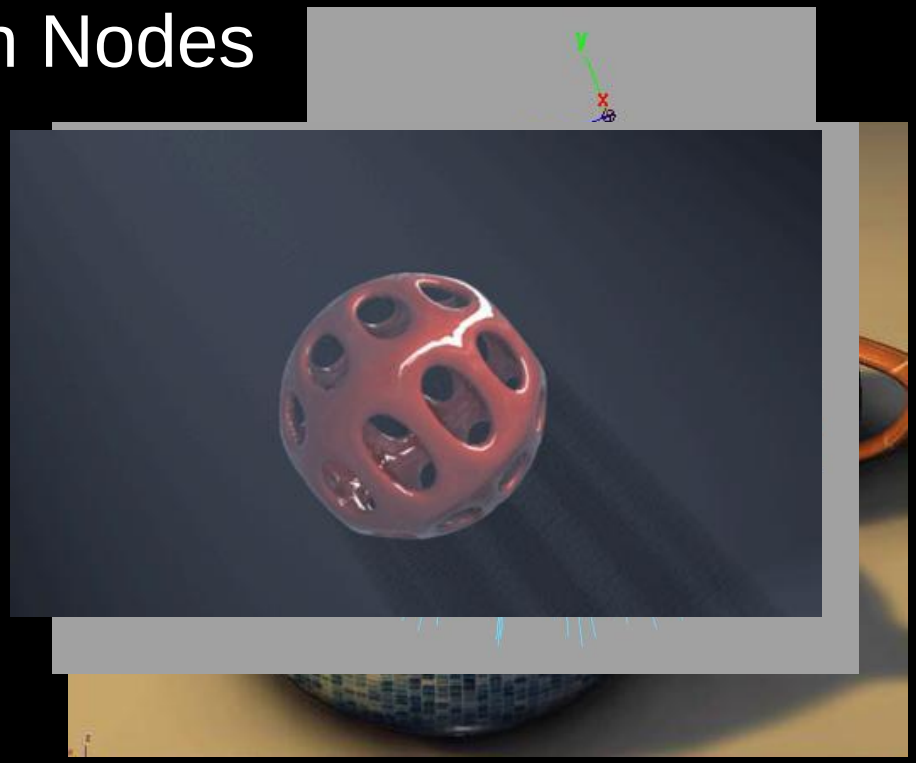
What is an API?

- Application Programming Interface
- It is a contract between the server application exposing the API and the client using that API.



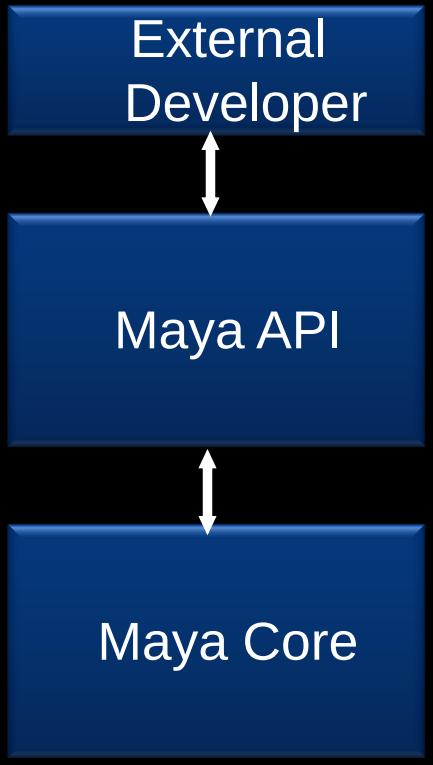
But really, what can you do?

- Commands
- Dependency Graph Nodes
  - Deformers
  - Shaders
  - Manipulators
  - Shapes
  - Etc.
- Tools /Contexts
- File Translators
- Automation

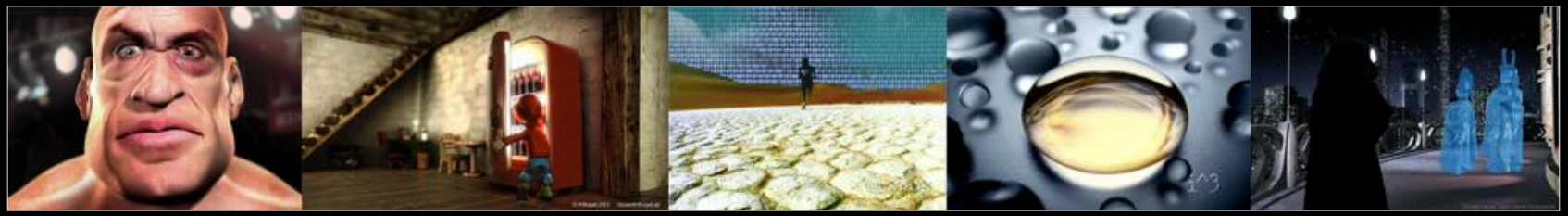


API Design Overview

- A tight wrapper around Maya's internal architecture
- An abstract layer, which separates Maya internal code from external plug-in developers



Plug-in Development



Autodesk®

Plug-in Development Environment

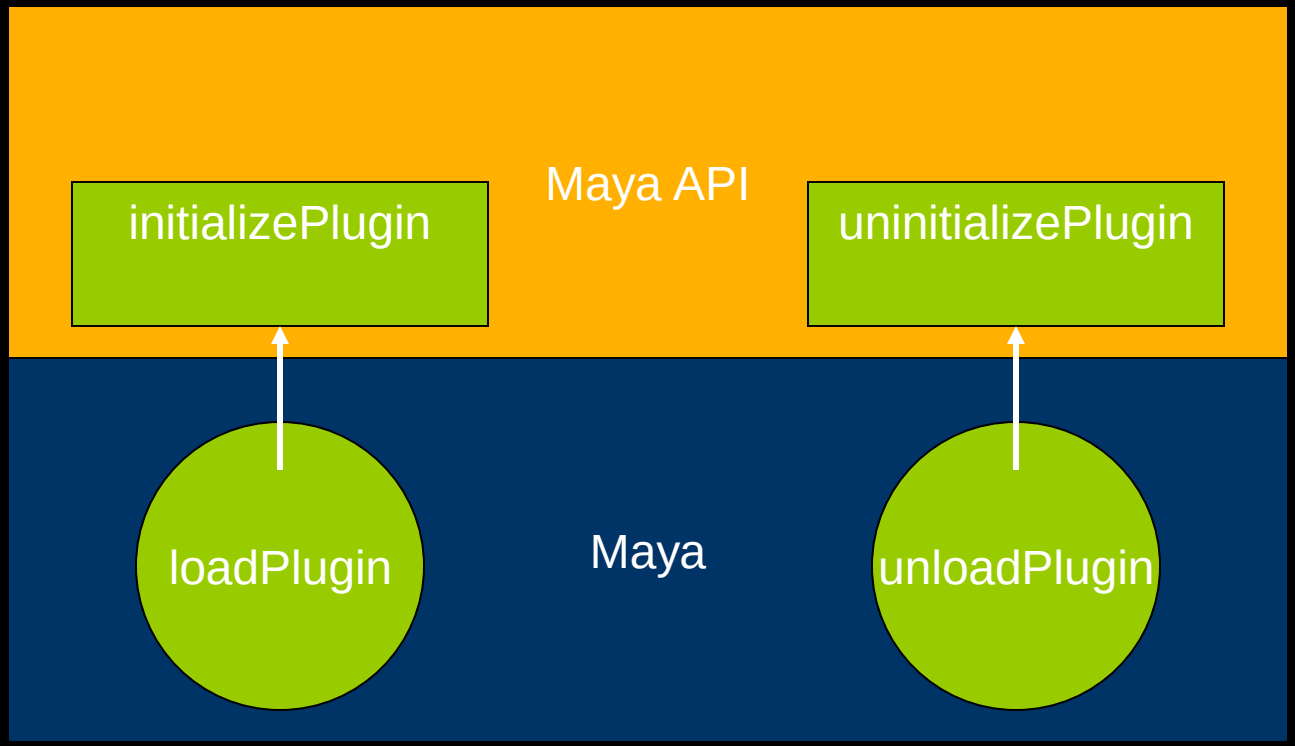
Operating System	Plug-in Type	Compiler
Windows	.mll	Visual Studio 2010 with Service Pack 1
Linux	.so	gcc 4.1.2
Mac	.bundle	XCode 3.2.1 with gcc 4.0.1 Universal build binaries: powerPC, IntelMac
all platforms	.py	Python 2.6 kernel

Maya Python API Plug-ins

- Load/Unload Plug-in commands extended for Python
  - Python plug-in read into a dictionary
  - Searches for entry points
  - Action runs scripts
- Required entry points
  - initializePlugin
  - uninitializePlugin

Maya Plug-in Architecture

- initializePlugin() and uninitializePlugin() as entry point and exit point



Maya Python API Plug-ins

Define a Python module containing initializePlugin(), uninitializePlugin() functions

1. import necessary modules
2. def initializePlugin()
3. def uninitializePlugin()
4. class implementations…
5. def creator()

1. Import Necessary Modules

Classically, we import what we want at the top of the module

```
import maya.OpenMaya as OpenMaya
import maya.OpenMayaMPx as OpenMayaMPx
import sys
```

2. initializePlugin()

Scripted plug-in initialization

```
# Initialize the script plug-in
def initializePlugin(mobject):
    mplugin = OpenMayaMPx.MFnPlugin(mobject)
    try:
        mplugin.registerCommand( “foo”, cmdCreator )
    except:
        sys.stderr.write( "Failed to register command: %s\n" %
kPluginCmdName )
        raise
```

3. uninitializePlugin()

Scripted plug-in uninitialization

```
def uninitializePlugin(mobject):
        mplugin = OpenMayaMPx.MFnPlugin(mobject)
        try:
                mplugin.deregisterCommand( “foo” )
        except:
                sys.stderr.write( "Failed to unregister command: %s\n"
% kPluginCmdName )
                raise
```

4. Class Implementations

Class Implementation: Similar setup to C++ plug-ins

```
class scriptedCommand(OpenMayaMPx.MPxCommand):
    def __init__(self):
        OpenMayaMPx.MPxCommand.__init__(self)
    def doIt(self,argList):
        print "Hello World!"
```

- Parameter: **self** same as **this** in C

5. creator()

Creator Functions is similar to C++ plug-ins

- Pass Python function to MFnPlugin methods

```
…
mplugin.registerCommand( “foo”, cmdCreator )
…
```

- Ownership is important

5. creator()

OpenMayaMPx.asMPxPtr() transfer ownership of newly-created command or node objects to Maya

```
class scriptedCommand(OpenMayaMPx.MPxCommand):
  ….
def cmdCreator():
  return OpenMayaMPx.asMPxPtr( scriptedCommand() )
```

Deployment of a Plug-in

- Maya Plug-in Manager
- Environment Variable: MAYA_PLUG_IN_PATH
- Put your plug-ins into:

  C:\My Documents\maya\2012\plug-ins
- Add your custom plug-in path

  putenv MAYA_PLUG_IN_PATH $destPluginPath;

```
string $currentPluginPath = `getenv MAYA_PLUG_IN_PATH`;
string $destPluginPath = $currentPluginPath + ";C:/My Documents";
putenv MAYA_PLUG_IN_PATH $destPluginPath;
```

- Maya.env or userSetup.mel

How to Load a Plug-in

- Load from Plug-in Manager or Maya commands:

```
maya.cmds.loadPlugin( "foo.py" )

maya.cmds.unloadPlugin("foo.py" )
```

Re-iterate: Generic Plug-in Algorithm

- Define initializePlugin and uninitializePlugin functions
- Register and unregister the proxy classes (MPxCommand, MPxNode, etc…) within these functions. ie register the custom commands and custom nodes being defined by the plugin
- Implement creator and initialize methods (as required) which Maya calls to build the proxy classes
- Implement the required functionality of the proxy classes. This requires importing the necessary modules

Summarize Main Methods

- initializePlugin allows Maya to load a plugin.
- uninitializePlugin allows Maya to unload a plugin
- Both must be defined

- Creator functions are used to instantiate a derivative of a proxy class (MPxNode, MPxCommand) and transfer its ownership to Maya

- Node initialization functions are used for derivatives of MPxNode in order to define their attributes and initialize their values

- Use try … except statements in order to trap errors and avoid crashing Maya. As a rule of thumb use them around API functions that return MStatus object

Autodesk