1. Introduction

Hey there. My name is Cyrille Fauvel, working for the developer technical services here at Autodesk in Europe. Welcome to the introduction to Maya dependency graph programming presentation. This presentation is translated to Japanese by Akira Kudo, developer consultant at the Autodesk Developer Network.

2. About the presenter

This is just quick meet the presenter's slide. you ‘ll recognize me by the picture next time will be in a conference. Feel free to come to me to have a chat about anything and I’ll happy to assist you.

3. Agenda

Ok, so here is agenda for today. The Maya architecture is based on nodes and dependency graph, where nodes are very simple C++ objects. The dependency graph is the real brain of Maya. The dependency graph is a collection of nodes. Nodes can be from very basic operations in Maya to more advanced. But the dependency graph controls how nodes are working together. During the presentation we will make demo in C++ or Python to demonstrate everything we are going to show you today.

4. Working with Node

In the Maya concept, nodes are very simple C++ object, usually derived from the MPxNode's base class in the Maya API. What is the important is that a node is like a black box operation for Maya. It means that Maya does not really know what happens in a node neither that a node knows what happens in Maya. What is important is that a node behaves correctly towards the dependency graph. Data exchange between Maya and nodes, or between nodes, is done through attributes. But attributes are really definitions of the data a node cannot accept or produce. Attributes can be easier static or dynamic and are defined when the node definition is created. A node itself does not hold any data. The MDataBlock class holds all the data for each node instance. A MDataBlock could be unique for a time instance or node instance or shared, depending of its usage. Finally, nodes are interconnected together through plugs and connections.

Here is a quick example of how a node is working. First a node receives from other nodes or Maya, some attributes which has converted into something different and then output to other nodes.

When we are talking about attributes in Maya, we should distinguish two categories. The first one is the attribute's definition. This is the static number of the node class. And an attribute definition is defined only once all instance of the same node type. This one does not all any data, but it is only the definition which will be shared class or nodes of the same type. Derived nodes inherit from their base class. it means that when you derive a new node class from a given node class, this one would inherited from its parent and would exhibits the same behavior. Derived nodes should never try to change an attribute behavior inherited by its parent. The second type is the attribute data. This would be specific to one instance of a node. On this one would hold the data.

5. Attribute definition

This is the short list of different attribute types that you can define for your node. Each attribute types correspond to specific data type that Maya would be able to fill in/fill out from a Maya file. However two of these attributes do not host data directly. The first one is the message attribute. This does not host any data at all and is used to message between nodes in Maya. The second one is the compound attribute which is there to group a set of other attributes to make one new attribute available to your node.

6. The Maya plug-in Wizard for Visual Studio

To illustrate what we are talking about I want to show you some C++ and Python code. But before that I also want to introduce you the plug-in wizard which is available for Visual Studio 2005. If we develop on window platform using Visual Studio 2005, you can use it very handy plug-in wizard to create your plug-in code's skeleton. You would find the plug-in into the Maya developer kit under the plug-in wizard folder. In that folder you would find a zip files and the instruction to install the wizard. You would extract files from the zip file into the VCWizards/Maya plug-in wizard directory on the Visual Studio 8/VC/VCWizards. All the files in that directory are html / javascript and text file which we are used to create project skeleton. The other directory you would need to create is under the VCProjects subfolder and still in Visual Studio 8 directory. In that directory you would find three files. However the .vsz file contains an error that you would need to correct for Visual Studio 2005. When you open that file after having extracted it from the zip file, you would find here VsWizardEngine version as 7.1 where for Visual Studio 2005 it should be 8.0. So you would need to change that manually to get the wizard to work properly. Then you save that file and ready to run the wizard in Visual Studio. 7.1 was OK for Visual Studio 2003. But 8.0 is needed it for 2008.

7. Your first C++ Dependency Graph Node plug-in

Ok, now we have the Maya plug-in wizard installed on Visual Studio 2008. You can access using the file, new project. You would find the Maya plug-in wizard under the Visual C++ section where you can now run the Maya plug-in wizard. You can give a name to your project whatever you like and then you press ok. And you have to provide vendor name. So let's put Autodesk for me. And you can remove comments and leave them if you want into the code generated. And with the wizard you can create three kinds of plug-ins, Mel command, Mel command with undo/redo and for us, dependency graph node plug-in. So this what we want to select. I will also rename this class to "myNode". So I would have a new created node class named "myNode". I can additionally have some other includes which I do not need today. But if you need them, you can automatically have that in your project from the wizard. Then you can press finish and the wizard is now creating source code on the project for Maya with all the setting you need. If you go straight into the solution explorer, you would find all the files on the project generated by the wizard. So let have a quick look to myNode.h file. This is what is generated by the wizard as you can see it creates a new class myNode derived from MPxNode. You can change that if you want to derived from another Node type in Maya. But for our demonstration, we just need that. What is more important is that it creates by default, two attributes definition of type MObject. Because here we are just keep reference to the real object behind input and output variables. And also there is a static MTypeid, a static variable into the class which should be the unique identifier for our new node type. This would be let Maya recognize your C++ class has being a node into a Maya file. Then you have a set of three functions which are compute, creator and initialize. Compute, we are going to talk about that in a few slides. This is a function which is called by Maya when an output attributes need to be re-evaluated. The creator function is static and is the creator of an instantiate of this class. Because Maya does not know your node class. It needs to have common function to create a new instance of that class and the create function is there for that. Initialize, it is we are going to spend most of time on this demo. It is the place where you would instance it and define attributes for your node. So let's have a look now to the C++ implementation of your node.

First thing is you have to provide an id for your class. We’ll come back on these later into this presentation, but this id should be unique and assigned to your class. Then because attributes definitions are static to the class we need to define them. And we have a constructor and destructor for our class. We have the compute method. Let's go through quickly. And we have to create a function we are talking about. This is where you just say "new myNode();" and return the pointer to Maya.

The initialize function now. So this is where you would instantiate and defined your attribute for your node. So first attribute was input. So you declare MFn new numeric attribute which is a function set to work with numerical attributes. So you say, from the function set to create a new input attribute with a short name "in" and you declare this to be a float. And default value for the float will be zero. You say it would be storable meaning that will be saving Maya file. And you also tell it is keyable, so it will show up in the channel box. Then you do the same for the output attributes. Except this time, you may not want to save it into to Maya file. Because you would recompute it anyway. And you would also say that it is not writable. Because it should be evaluated by the node and not let Maya or any users set the value. Once you have created the two attributes definition for your node. All you need it do now is add them to node definition. To do that you would use the addAttribute() method of the MPxNode class providing input and output attributes. That would add the two attribute definitions into your node class.

The last thing you need to do now is tell Maya how is dirty flag should be propagate through the node. To do that you would use the attributeAffects() function providing the link to the input and the link to output attribute definition. Then Maya would know that whenever someone is changing the value for the input attribute definition. The dirty flag will be set and propagated to the output attribute. If a connection is made to another node from the output attribute, then the dirty flag will propagate to the next node. On that’s all you need to do during initialize function.

8. Your first Python Dependency Graph Node plug-in

Ok, let's do the same thing using the Python language now. Since we are writing a dependency graph node, we would need to write Python plug-in versus Python script. For this we would need to use the Python library and to import the Python library into our plug-in we would need to use the import keyword. We need to import two libraries in our plug-in. The first one would be the maya.OpenMaya library. And I would use the keyword "as" to simplify the notation into the script., "as OpenMaya". And the second library would be the "maya.OpenMayaPx as OpenMayaPx". To implement our Maya node, we would implement a new class using Python. So we would create a class of type myNode derived from the "OpenMayaMPx.MPxNode". Because we want to derive from the MPxNode class. And we will also define the two member variables for that class which will define our attributes. So the input which will be of type "OpenMaya.MObject". And the same thing for the output attribute.

To have a function node class we still implement two functions. The first one would be the constructor for our class, "def \_\_init\_\_ (self)". In this constructor, the only thing we need to do is super class message to the MPxNode class, "OpenMayaMpx.MpxNode.\_\_init\_\_(self)". The second method is the compute() method. We do not want yet to fully implement it. But we still need to implement it has minimum to have a functional node. "def compute()" note here that, they will be additional parameters for python which will be the self pointer and will get the plug and datablock parameter. Since we do not want to yet to fully implement it, we just tell Maya that we do not know what to do with any of the pass and in. And to do that we have to return OpenMaya.kUnknown parameter.

We now need to implement the creator function for Maya to instantiate a new instance of this node. But unlike C++, python do not support static member function. We would need to create that function as a global function in python. So the creator function would be implemented as global scope, "nodeCreator()". And the only thing we need to do in that function is return MPxPointer of the new instance, "return OpenMayaMPx.asMPxPtr(myNode())".

On now the function we want to look into more detail is the initialize() function. So it would be defined as well as global scope. Because it was a static function in C++. And the first attribute is the input attribute. I would need to use the MFnNumericAttribute function set. Because we want to have our input attribute as float. So I need to instantiate an OpenMaya.MFnNumericAttribute() function set. And from that I can say to myNode.input variable that it is now a float. So I would give it a long name and short name. Then tell that it is a float using kFloat value and putting the default value to zero. Since we want that attribute to be saved it into the Maya file. Then we will tell our function set that it is storable giving the value 1. Because you want also this attribute to appear in the channel box. We will define the nAttr.setKeyable() function passing one as well.

For the output attribute, it is almost same thing. So I will just copy and paste this case on change only what I need to change. So, for example here. It is now the output attribute. It is name is now "outname" and short name is "out". It is also float and the default value is also zero. But this time, I do not want to store this attribute into the Maya file. Because it is re-evaluated every time. So it puts the zero value. And I also do not want user to be able to change that value directly. So I would tell our function set that set writeable flag should be zero.

Now we have successfully defined our two attributes. We need to have them to our node. To do that we will use the addAttribute() function of the node class to has these attributes definition. So "myNode.addAttribute(myNode.input)" and I had as well the output attribute. Now the last thing we need to do is to setup all the dirty flag propagation. To do this, we will use again a function of the node class which is the "attributeAffects" where we teach Maya that whenever the "myNode.input" attribute is changing it should affect the "myNode.output" attribute. And that is it.

9. MDataBlock and MDataBlock Array

So if node attribute does not know all data where are the data stored, it is stored into the MDataBlock or MDataBlockArray. These classes are specialized in holding the data for a node or an instance of a node. But the MDataBlock may be duplicated for each time instance or node instance.

A MDataBlock could easier be shared or unique for node or several nodes, or both. It does not really matter, Maya will decide for you considering performance versus memory. A MDataBlock could also be cashed depending if we contain light or heavy data. So as you can see on the right of the screen, we have two instances of the same node which could either contain their own MDataBlock or capsulated into the node array or sharing the same MDataBlock instance. What you can also know here that we discuss a few slides ago is that both nodes instance have only one reference to the attribute definition.

10. MPlug and Connections

Finally, in order to get node to fully work into the dependency graph of Maya. A plug is the object which will allow you to connect a node into the dependency graph or two other nodes. When I say connect to a node, in fact it is connected one attribute of a node to another attribute of a node. A plug is a link to an attribute data of a specific node. So you would connect to the data into the MDataBlock of an instance of a node for a given context. You can retrieve a plug for a node using the MFnDependencyNode::findPlug() function. But what is important is when you query a set of value through a plug, do not forget that you would either force a plug to re-evaluate the attribute value or propagate the dirty flag. So when you are inside your compute method of a node, you should never set or get a value from a plug instead you should use the method of the MDataBlock.

11. MDataHandle and MArrayDataHandle

From your compute method when defining your node, you would not call the plug get or set value function instead you would use the MDataHandle or MArrayDataHandle classes. They are used to access the data of your datablock and is more efficient.

12. C++ compute() method implementation

Now we have the tools implemented the compute() function. Let see how it is done in the C++. So our node class definition implements the compute() method. Passing has a parameter to the plug which is been requested and the associated datablock. The first thing we would need to do during this evaluation is verify that first it is the plug we can evaluate and that everything we need. To verify that it is the plug we can work with you would these test. If that plug passed as the parameter is equal to our output attribute definition. If it is then, we will evaluate the data. But if it is not, then we return the kUnkownParameter flag to Maya. That will tell Maya is that either we do not know to evaluate the plug or that maybe our parent is able to evaluated it. So passing the kUnkownParameter will force Maya to call our parent class and ask to evaluate this plug. So now we are verified that plug is something we can evaluate. We would first access the input data using MDataHandle. So here we have only one input attribute. So we would say, "MDataHandle inputs attributes equal our datablock input value for our input attribute definition". Because we know it is a float. Then we can ask the data handle to return the value contain to that handle has a float on get our result vairiable initialized. To set the value to our output attribute, we would also ask for the datablock to return us an handle on the output value. And then we would set the value to whatever value we want and we would clean the dirty flag at the same time. So next time someone needs that value , the dirty flag will be reset and Maya will not ask our node to re-evaluate that value. So result value could be whatever you want as long it is a float because our output value is float. So we just past value from input/output has we do here or we can do something like 1.1, and then we will multiply the input value by factor of 1.1 every time it is evaluated.

13. Python compute() method implementation

The implementation of the compute method() in Python is not much different from the C++ sample . The first thing we need to do is to verify that plug Maya passes parameter is of a type that we can evaluate. So "myNode.output". If it is then we return after evaluation "OpenMaya.kSuccess", otherwise we return kUnknownParameter on that we force Maya to call our parent class to evaluate the plug. Now what we need to is to read the input attributes for our node. And to do that we need a handle. So the "inputHandle". We get it from our datablock parameter "inputValue" for "myNode.input". And we will set our result variable to the float value of this "inputhandle" as float. Now we also need to have a handle onto the output value of our datablock. "datablock.outputValue" for "myNode.output". Now we can say to our handle to set the value has a float for this result value. The last thing we should not forget to do is since we have evaluated the new value for our output attribute. We need to set the dirty flag to be clean. So Maya will stop asking us to evaluate that attribute. "setClean" of this plug. Like in the C++ sample now, we are passing in the input value to the output value of the changing it. But we can again change that value and multiply it by factor of 1.1. So every time the plug would be evaluated. The result would be multiply it by factor of 1.1.

14. Connections rule

The last thing about the plugs are connections. Nodes are interconnected by connections thought plugs. And you can connect two plugs together, if they are of the same type or could be cast two a different types. Fan out is ok. It means that you can connect one output plug to several input values of different nodes. But you cannot fan in meaning that you cannot plug two output attributes to one input attribute of a node.

15. Node diagram

There is a diagram of summary for node implementation into Maya. What you have to remember is that the attribute definition is unique for a node type. The MDataBlock maybe single or unique for a node on contain the attribute value. The compute() method takes input value to evaluate the output value. And the compute() method is only called when a plug is requested. A plug could be easier expose to the outside or internal usage only of a node. If they are exposed outside, then it means other nodes could be connected to this plug. One last thing which is important we have not yet talked about is the dirty flag. So an output value would be evaluated only if it is marked dirty. If it is not marked dirty it means the attribute is up to date and the compute() method will not be called for that plug or value. If it is marked dirty, then the compute() method will be called. Another important things to know is while the data flow one way, the dirty flag is flowing the other way. So for the data when you request an output value to be evaluated. The output attribute will use the compute() method which then pull the data from input attributes. For the dirty flag, it is when you change input attribute, that dirty flag will be propagated throught your node using the affects behavior that you defined for your attribute message. So the dirty flag will fly from the input to the output attribute using the link you have defined.

16. The MPxNode class

This is the quick summary of a few previous slides. When you implement the node, you derive the class from the base MPxNode. And then you give two that classes unique you identifier which will tell Maya about your node and would tell Maya when it reads a Maya file if it contains one your node instance. Then you would declare an attribute definition on me. And this is into the initialize function that you would instantiate this attribute definition. This is where you define the type for a given attribute and which one interactive another one. Finally to let Maya creates an instance of your class you would implement the creator() function where you would instance a new C++ instance of your object. The last thing you have to do is the compute() method where you can see the two arguments are of type MPlug and MDataBlock. The MPlug would describe which plug is been requested and you have the associated MDataBlock which contains the value for this node instance on time instance.

17. A unique identifier

Node id , unique identifier are use for filing purpose. This number is 32 bits integer. So from 0x00000000 to 0x00007fff are reserving for testing purpose. If you wish to have your own id, you would need to replace this id to hotmail@alias.com. If you are ADN developer use the ADN DHO service to reserve blocks of IDs. When you replace the reserved id of block of nodes, you would provide your company name, contact name for an e-mail on the number of id you want to reserve.

18. initialize()

The static initialize() function is the place where you would create the attribute and set the attribute's flags. Then add attribute to the node definition and define the attribute's relationships. You would then declare to Maya how your node is working regarding attributes on data.

19. compute()

The compute method() is called by Maya whenever someone is asking for a dirty output flag value. So Maya would call this compute() method passing in the plug and the corresponding datablock. On the compute() method you would be responsible for evaluating the plug value. If you do not know how to handle the plug, then you would return the kUnkownParameter flag. That will tell Maya that is you do not know how to handle that and will force Maya to call your parent node class to know if this one can evaluate the plug for you. If you know how to evaluate the data, then you et the value and then you update the dirty flag accordingly. So the plug would be marked clean and the compute() method would be not called again. While in the compute() method, you should never do something which you would send a dirty message to the dependency graph. And remember that a node should be consider has a black box operation. So you should never look outside of your node to get the data on only get the data from input attributes.

20. Registering Dependency Graph Node into Maya

When your node class is ready, so when you have defined your node id and class definition, do not forget to register it for Maya uses it. You would then provide to Maya the node id's the creator() and initialize function. And for sure if you register a node into Maya during the initialization of your plug-in, do not forget to unregister your node during the uninitialization. Otherwise Maya will crash soon on later.

21. Registering the Node using C++

Before we can register this node into Maya, we need to assign a unique id to that node. You can reserve id for your node using the Autodesk services or you can use development reserved id. So this is what I am going to do here today. So I would use the id 0x87000 which is used for development only. And now that I have made this I can remove this #error to avoid the compiler to fail compiling. And the last thing I need to do now is to go into plug-in main in cpp file where I have the initialize plug-in and uninitialize plug-in function that Maya calls when it closes and unloads your plug-in into Maya. And this is where this id will be used during the registerNode function call where you provide a name for your node that you provide the id will just define the creator() function to instanciate/uninstance of this node class and initialize() method which will create an instaciate attribute for your node. If the registerNode fails, then you load return your node, otherwise just fine. During uninitialization, you need to unregister your node giving the id you have assigned your node. And that is all you need to do in C++ since everything else is made automatically by wizards.

22.Registering the Node using Python

So let's the how to finish our python plug-in now. So we have implemented a node which has the compute(), creator() and initialized function already. We still need to implement "myNodeId" which should be the unique identifier for our node. So "OpenMaya.MTypeId" which I am going to initialize with the same value we have use in the C++ sample. One that is done, Now I need to register our node into Maya. So Maya can start using it.

When we register our node during the initialize function which is called by Maya when you load your plug-in. So "def initializePlugin" is coming with a parameter. And we initialize the mplugin of type, "OpenMaya.MFnPlugin" which is the function set for any Maya plugin passing in the parameter object. And using that variable will call the registerNode function which will register our node. We pass in the node name, then the node id, the node creator function and the node initialize function. Now the only thing we need to do now is to unregister our node when Maya unloads our plug-in and we will do that from the uninitializePlugin which is implemented the same way has a initializePlugin. But here we will just deregisterNode and the only parameter we need to pass in this time is the node id. And our sample is now completed and ready to work.

23.The Maya Dependency Graph

Until now, we have talked a lot about nodes. But the real of Maya is the dependency graph. This is control system for Maya on the foundation of the Maya file format. The dependency graph is making usage of plugs and connections to connect your node on get your node to work.

24.The dirty flag propagation

On this slide, we are going to demonstrate our node is working inside the dependency graph. When I say how our node is working it is not really truth. It is how the dependency graph itself is working. First take a step back and look at the node we have implemented in the previous slide. So we have in these context a node, it is compute() method and datablock. And every input and output attributes are clean meaning that all attributes are up to date is the correct value. Each symbols represent the value the datablock contains for that instance on time instance. When someone set a new value to one of the three attributes it generates a dirty flag propagation. The dirty flag propagation is going through the links between attributes and gets to next the node through the output attribute's plug. But what we have seen here is that the output to attribute data does not change. Only the dirty flag has been propagated through input three, output two and then outside the node to the next node is the plug is connected. Now let's see what will happen if a node is requesting one of the out attribute data. First case, you ask for the data of the output one node. Since that node is clean, then the data go slide the next node. Second case, now you request the data for an output node which is marked dirty. And because we have links between input two , input three to output two. Then the system will ask for data of the two input attributes. The one which is marked clean will go straight and the one which is marked dirty will then go on be recomputed to the output instance. The new value will then be outputed to the next node by cleaning on his way the dirty flag. And you would then get the same node with clean data again.

We have take a look to one instance of the node of the dependency graph. Here we are going to take a look two bigger thing where you have multiple nodes playing a role. So you set the data on the second node, second input attribute. This is our scene role after the dirty flag propagation. Because as we discuss, the dirty flag propagation is following these relations inside the node to propagate the dirty flag to these output attributes. And because these two output attributes are connected to input attributes of the nodes. The dirty flag propagation go through them. And again, we propagate the dirty flag to those nodes. Now Maya or another node may request a value of an output node which is marked dirty. So let's take the example here where Maya is requesting. The first attribute of the third node, Maya will then evaluate all these attributes which are marked dirty. A boll that attribute. But as you seen, because it does not force the value propagation to other nodes. So attributes are still marked dirty, because we are not evaluated for that request.

25.Further learning

If you want to learn more about the Maya API, you have various resources available for you. You have the online help on the developer's sample coming with Maya release. You also have the developer center on the Autodesk web site as well as some discussion group about the API, Python and MEL Script. There is also the ADN Developer Network that you may want to join if you want to have access to the knowledge base, developer support and training classes.

26.Thank you!

I hope this was useful information for you. And thank you for joining my presentation today.