Week 01 Introduction

- What is MEL and what is MEL for?
- What is PyMEL?
- The Maya Script Editor
- The basic of MEL syntax (part I): variables and object attributes

What is MEL for?

- Bypass Maya's user interface and create quick shortcuts.
  - Example: you can create shelf buttons with customized function

- Create procedural modeling, animation, dynamics, and rendering.

Rock generated by MEL | Phone cord generated by MEL

Shading generated by expression | Expression controlled particle
• Customize Maya's interface.
  ■ You can create GUI for your own workflow. For example:

  ![GUI for a RockGen MEL script](image)

• Access advanced features and hidden features.
  ■ Example: there are several particle attributes that can only be accessed through MEL, but not from the user-interface.
The relationship between MEL, Maya interface, and the node-based architecture

**Maya has a node-based architecture.** Internally, every part of a scene (whether it is 3D geometry, animation, a light, or a texture) is represented as one or more nodes. These nodes are called **DG nodes**. These nodes are connected together to form a dependency graph, or the scene-graph.

Each node has a set of attributes, each of which stores a characteristic of the thing the node represents. For example, a “transform” node contains the x, y, and z position of the “shape” node that it is manipulating.

Most of the MEL commands are used to create and manipulate these DG nodes.

Maya’s user interface, which includes all buttons and menus, is built using MEL. When you click a button or select a menu item, Maya executes a MEL command specific to that menu item; some of these MEL commands are human-readable scripts that are installed when you install Maya (e.g. in C:\Program Files\Alias\Maya2009\scripts), others are built-in commands that come as part of Maya’s executable.

Therefore, you may think that Maya contains 3 layers:

One major usage of MEL scripting is to by-pass the graphical user-interface, and manipulate the dependency graph directly.

(Note that Maya’s node-based architecture will be covered in the course “Computer Animation Workshop II” and will not be covered in this course.)

**Plug-ins vs. MEL scripts**

To extend the graphical user-interface, Maya allows programmer to write two kinds of program: MEL scripts and plug-ins. Since this course will only focus on MEL scripts, we will not look at any plug-ins related stuff. I include a quick comparison between these two approaches:

<table>
<thead>
<tr>
<th></th>
<th>MEL/Python script</th>
<th>Plug-ins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>MEL/Python</td>
<td>C++</td>
</tr>
<tr>
<td>Speed</td>
<td>slower</td>
<td>faster</td>
</tr>
<tr>
<td>Compatibility</td>
<td>Platform independent</td>
<td>Platform dependent</td>
</tr>
</tbody>
</table>

- MEL is the scripting language within Maya. Currently Maya supports two scripting languages. The other one is Python. The choice between the two languages is actually a personal taste,
IMHO. There is one advantage of using Python is it is also available in other animation package like Houdini.

PyMEL scripts

MEL is very powerful as a scripting language. However, it is also very limited in more advanced data type support. For example, you cannot group different types of data together in MEL. It doesn’t support of OOP for developing large applications.

One way for fixing this problem is redesign the language of MEL but Autodesk™ select another approach. They use another language called Python for scripting instead of extending the features for MEL.

The language is called PyMEL as it is a combination of Python and MEL objects (Python + MEL). I will show you how to use this new scripting language later in the lectures. As it can create some really interesting models, otherwise it is very hard to create manually.

Generative models using PyMEL (i.e. Python) Script

The Maya Script Editor

There are several ways to “run” a MEL script:
1. If it is a single-line command, you can type it directly in the Command Line.
2. You can type a longer script in the Script Editor and run it.
3. You can save a script into a MEL file, and later on load it into the Script Editor and run it.
4. You can “store” a script as a shelf button.
5. You can write a “procedure”, store it in a file and use it as a new MEL command (will be discussed later).

### Method 1) Use the Maya Command Line

Type the following in the Command Line, and then press Ctrl+Enter:

```mel
cone;
```

After you type this command in the Command Line, a cone is created (with a default name assigned), and is **selected as the current object**. The “current object” concept is very important for you to write longer MEL script.

Also try using ENTER instead of Ctrl+Enter. What’s the different?

Now try the following:

```mel
move 3 4 6;
```

The **current selected object** is moved to the position (3,4,6).

Each MEL command may have several options. For example, the cone command can be used as:

```mel
cone –axis 0 1 0 –radius 5 –name “myCone”;
```

After you execute this command, another cone, named `myCone`, with axis (0,1,0) and radius 5, is created and selected. Now type the following:

```mel
move –x 2 –y 3;
```

(Which cone is moved?)

Lastly, let’s try the following:

```mel
move –r –x 3;
```

Look at the position of `myCone`. What is the meaning of “-r” in the last command?

### The MEL command categories: create/edit/query

How to change the radius of the cone using script? How to change the axis of the cone using script? Here come to a concept of **MEL command categories**. There are 3 categories:

- Create: these commands create some DG nodes.
- Edit: these commands modify nodes.
- Query: these commands return information about nodes.
Most commands fall into more than one categories. For example, the cone command we tried:

```
cone –axis 0 1 0 –radius 5 –name “myCone”;
```

is used to “create” a cone called “myCone”. We said that we used the command in the create mode. But it can also be used to “edit” a cone. For example, try this:

```
cone –e –radius 8 “myCone”;
```

The option “-e” means using the command cone in edit mode, edit an existing cone called “myCone”. When you look at the MEL manual of any command, it shows you whether a command can be used in create mode, edit mode, or query mode.

The following statement will return the radius of myCone:

```
cone –q –radius “myCone”;
```

The option “-q” means using the command cone in query mode, asking for the information of an existing cone called “myCone”. The result will be shown besides the Command Line, as well as in the Script Editor, which is described in the next section.

Method 2) Use the Maya Script Editor

The Maya Script Editor gives you the full power of writing MEL scripts. Click the lower-right button in Maya to open the Script Editor:

![Script Editor](image)

The Script Editor contains two parts: the lower part is the input area that you can edit your script, and the upper part is the info area – it shows you the result, error and warning after you execute your script.
For example, type the following lines into the input area of the Script Editor:

```mel
cone;
move 3 4 6;
cone –axis 0 1 0 –radius 5 –name “myCone”;
move –x 2 –y 3;
```

Press Ctrl+Enter to execute the script. Look at the result messages shown in the upper area of the Script Editor.

Now type the above script again. Try to high-light one or more statements and then press Ctrl-Enter. What’s the different?

---

**Class exercise: create simple shape from scripts**

Try to write a MEL script which can create the shape shown in the figure.
You can use any command. In particular, the following commands may be helpful:
sphere, cone, move, rotate, scale

**Try using more than one way to create the same shape.**

---

**Method 3) Create a MEL file**

Type the above script into the Script Editor, high-light all statements, and then choose
File > Save Script …. MEL file usually ended with “.mel”. Your own MEL file can be
saved into two folders:

- **My Documents\maya\scripts** if you think that the script can be used
  in other Maya version
- **My Documents\maya\2009\scripts** if you think that the script should only
  be used in Maya 2009

Later on, you can use File > Load Script… to load (without running) an existing MEL
file, or File > Source Script… to run an existing MEL file (without loading it into the
input area).

**Method 4) Save a script as a shelf button**

Select the “Custom” shelf first. Type the above script into the Script Editor, high-light all
statements, and then choose File > Save Script to Shelf…. Give a name to the button.
Now, you have a shelf button containing the 4 lines of MEL scripts. Whenever you click
on the button, these 4 lines will be executed.

**Where is the script saved?** The script is saved to your Maya “shelves preference file”. For example, if your button is put under the “Custom” shelf, then your scripts is saved in:

- **My Documents\maya\2010\prefs\shelves\shelf_Custom.mel**

It is also a MEL file. Open the file using Notepad. You can find your 4 lines of script,
plus some additional MEL scripts which setup the shelf button(s).

You can also add an icon to your shelf button. Prepare a 32x32 BMP image for the
button icon. I will suggest you storing the image file into the “default icons folder”:

- **My Documents\maya\2010\prefs\icons**

Lastly, edit the contents of Shelf items by selecting

**Window > Settings/Preferences > Shelves**

Click “Change Image…” and select your own BMP icon.

**How to learn more:**

1) The **help** command: A useful MEL command is help. Try to type the following:

   *help cone*;
and then look at the info area of the Script Editor. It shows you the options available in the cone command.

2) Select Help > MEL Command Reference. These pages are written in a more detailed way than the help command, but sometimes the explanations are quite hard to understand…

3) Learn from echoes: When you click any button or pick any menu items, from the upper part of the Script Editor you can see the MEL commands being generated by the user interface. This is a good way to learn MEL scripts. You can also copy the MEL scripts, modify it, and to make it as your own tools.

To see more detailed information, you can choose Script > Echo All Commands from the Script Editor's menu bar.

Class exercise: “standard lighting”

Assume that you like to test your objects using something you called “standard lighting” combination: an ambient light, a directional light pointing to the –z axis (i.e. the default direction, no need to rotate it), and a point light placed at (10, 10, 10).

Using the “learn from echoes” technique, try to make a shelf button that can create “standard lighting”.

The Basics of MEL syntax (part 1)

We will learn more and more MEL commands in this course. However, to write a longer and useful script, just using the MEL commands are not enough. We have to use the common programming statements, such as variables, array, if-statements, etc.

MEL script looks like any other scripting language. Let’s look at its basic syntax. At the same time, you can compare to a scripting language that you are familiar with (say, Flash ActionScript).

- **Comments**: MEL use // to start a comment. The remaining parts of the line are ignored. For example:

  ```mel
  sphere; // this remaining part is the comments
  ```
- **Semicolon**: Each MEL statements should end with a semicolon. (Note that in some situation Maya allows you to execute MEL statement without semi-colon. However, to have a better practice, try to include semi-colon at the end of every statement.)

- **Variable**: In MEL, you have to “declare” a variable before you can use it. Variable name should start with a ‘$’. For example:
  ```mel
  int $a;
  float $b;
  $a = 3;
  $b = 4.57;
  ```

There are five basic types in MEL:

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Integer</td>
<td>1, 8, -28</td>
</tr>
<tr>
<td>float</td>
<td>Fractional number</td>
<td>10.12, -3.75</td>
</tr>
<tr>
<td>string</td>
<td>String of characters</td>
<td>“Hello”</td>
</tr>
<tr>
<td>vector</td>
<td>Array of 3 floats</td>
<td>&lt;&lt; 1.7, 2.8, 4.3 &gt;&gt;</td>
</tr>
<tr>
<td>matrix</td>
<td>Array of float array</td>
<td>&lt;&lt; 1.7, 2.8, 4.3; 0.1, 5.6, 7.9 &gt;&gt;</td>
</tr>
</tbody>
</table>

**Declare several variables with the same type**

You can use shorthand to declare several variables with the same type. For example, the following statements:
```mel
int $a;
int $b;
int $c;
``` can be replaced by:
```mel
int $a, $b, $c;
```

**Initial value of a variable**

In MEL, the initial value store inside a variable will usually be zero. However, as a good programming practice, I will suggest you not to assume that a variable has any initial value.

For convenience, you can assign an initial value to a variable during declaration:
```mel
float $balance = 100.78;
string $c = “abcde”;
vector $pos = << 1.7, 2.8, 4.3 >>;
```

**Assign value to variable**: as similar to any scripting language, you can assign value to a variable, i.e. store the value and use it later. For example:
```mel
$x = $y + 10;
```
The right hand side will be evaluated first. Therefore, the value of $y is taken out, add 10, and then the final result will be stored in the variable on the left hand side, i.e. $x.

**Type casting**: you can force the type of a variable/value to change, using the following syntax:

```cpp
float $x;
$x = (float)5;
int $y;
$y = (int)4.89;
string $s;
$s = (string)500;
```

**The print command**: you can use the `print` command to check the value stored in a variable. For example:

```cpp
print ( $x );
print ("x stored the value" + $x);
```

**Object name, attributes, and path to object**

In MEL, we access an object and their attributes by the following format:

```
<Object Name>.<Attribute Name>
```

For example, if we want to access the "scaleX" attribute of a NURBS sphere named "ball", then we use "ball.scaleX" to access this attribute.

Since Maya allows two objects to use the same name if they belong to different groups. As a result, when there is any name conflict, we have to use the “full path name” of the object to access it. For example, if the object “ball” is a child of a node named “Mum”, then its full path name is “[Mum]ball.scaleX”.

**The getAttr, setAttr and listAttr commands**

As mentioned before, in Maya every node has a lot of attributes. One major function of MEL is to manipulate those attributes. Therefore, you have to learn these 3 MEL commands: getAttr, setAttr, and listAttr.

The `getAttr` command returns the attribute value. The parameter inside the () is a string of the attribute name. For example

```cpp
float $x = getAttr( "myCone.translateX" );
```

Note that you can construct the attribute name using “+”. For example:

```cpp
string $s = “myCone”;
float $x = getAttr( $s + “.translateX” );
```

This trick is helpful when the object name ($s in the previous case) should be obtained from somewhere else, say, a GUI (we will see more examples later).

The `setAttr` command set an attribute to a value:
setAttr(“myCone.translateX”, 2.3454);

The `listAttr` command lists all available attributes of a node (the name of the attributes are shown in the info area of Script Editor):

```plaintext
listAttr(“myCone”);
```

** Week 01 End **