

# Principles of Maya Tool Design

---

## CHAPTER OUTLINE

### Tips When Designing for Users 180

Communication and Observation 181

Ready, Set, Plan! 181

Simplify and Educate 183

### Tools in Maya 183

Selection 184

Marking Menus 186

Options Windows 190

### Concluding Remarks 192

---

## BY THE END OF THIS CHAPTER, YOU WILL BE ABLE TO:

- Distinguish form and function in tools.
- Implement some basic strategies for user-centered development.
- Describe the role that selection plays in Maya's tools.
- Create custom marking menus.
- Identify common patterns across marking menus.
- Enumerate some common elements in many Maya GUIs.

Up to this point, we have discussed a number of basic language features of Python, the mechanics of how they actually interact with Maya, and how to create simple programs that execute different operations in Maya. With these basics covered, we can now turn our attention toward the more practical matter of designing Maya tools.

Generally speaking, any tool consists of two fundamental parts. The first part of any tool, on which we have hitherto focused, is the tool's *function*. Function concerns what the tool actually does, the sequence of operations it carries out to produce some desired end result. For instance, a tool may

perform some computation on a selected object and then alter it in a systematic way. This aspect of tool creation is frequently the easiest for many programmers, as it is the focus of much of their professional training, and is often the most exhilarating part of any tool creation task. At bottom, crafting a tool's function is a technical problem-solving exercise.

The second part of every tool is its *form*. The form of a tool is the part that faces the user, whether it is complex and configurable, remarkably simplified, or completely invisible. For example, a tool may take the form of a command that a user invokes via script, it may be a GUI with complex visual controls, or it may be a background process that is silently invoked when some ordinary action is taken, such as saving a file. Thus, the primary purpose of a tool's form is to gather input from the user in some way to pass the data on to the tool's functionality. Therefore, the form of a tool entails not only providing interfaces for gathering all of these data, but also, in most cases, validating the input and providing feedback to the user.

Designing good user interfaces is often the most difficult task for many programmers, as it can often be a psychological problem-solving exercise. It is a delicate art that is very situated in context: The most sensible approach for any particular tool can vary widely depending on its function, how it fits into a larger suite of tools, the person actually using it, and so on. Because of this complexity, and because many good books and articles have already been written on the topic, we won't stray too far into the territory of user interface design generally. However, before we get carried away creating tools and GUIs all willy-nilly, we do want to impress upon you the importance of focusing on your users, especially because the majority of users are rarely developers themselves.

In this chapter, we will briefly discuss some general tips to help get you started thinking about your users and then look at some existing tools in Maya to locate common tool paradigms. This chapter should help serve as a springboard for you to better understand the tools that we cover in the following chapters, as well to develop your own custom tools.

## TIPS WHEN DESIGNING FOR USERS

Without attempting to dictate a single vision for tool design, we want to cover a couple of items briefly, particularly for the benefit of programmers who are just beginning their journey of Maya tool design. There are some general strategies that may help you orient your own development efforts toward your users.

## **Communication and Observation**

One of the most important skills for any tools programmer is communication. Identifying and communicating with your customers during your tool development process is a central task. Whether you are soliciting a tool idea, seeking feedback during development, or reviewing old tools, you always want to involve some of your users to keep yourself from working in a black box and ultimately missing their expectations.

Equally important as communication is observation. Observing your users, either directly or indirectly, can sometimes be more illuminating than an email or even a meeting. Observation can often reveal workflow problems, of which your users may not be aware. Users may be accustomed to a workaround for a broken process, or may be inured to the fact that they have to make several mouse clicks for a single repetitive task.

Observation can also help you clarify communication. Some users may communicate a problem to you that they want solved, yet may be describing a symptom of a larger problem rather than the source problem itself. Users, in fact, don't always necessarily know what they want! For example, suppose an artist requests a tool that validates the naming convention of objects in a scene. You could simply create the tool as requested, but by observing and talking with the artist, you may discover that the artist is using an internal naming convention to sort objects in the scene while working. Thus, the problem concerns sorting and searching for objects, rather than the naming of objects necessarily. In this case, an alternate solution, such as a tagging system, may be more robust and easier to work with. By observing your users directly, you can often help them uncover better solutions to a problem than they originally thought they wanted.

Another useful strategy may be to indirectly observe your users by automatically tracking their input or capturing information when a tool fails that you can submit to a database. While the implementation of such techniques is beyond the scope of this text, it is a relatively trivial task in Maya given the power and flexibility of Python.

## **Ready, Set, Plan!**

Communication and observation are essential for establishing the goals of any tool. However, there is often production pressure to jump right in and start coding once the problem has been established. Although these forces can be difficult to resist, a tool that is developed without sufficient planning can very quickly turn into a mess. Tool developers can follow the same

processes involved in previsualization by planning everything out before beginning work. Thinking through all of the required components and making large changes before any code has been written can save much time and money. Frequently, taking the time to plan can also help developers see how built-in nodes and commands can be leveraged, where existing functionality may otherwise be needlessly recreated.

As you plan, remember to stay focused on the problem! It can often be tempting to expand the feature set of your tools well beyond their original problem spaces, or to overengineer using advanced programming techniques or language features, simply because the joy of solving technical problems becomes so engrossing. Apart from the obvious constraints imposed by production deadlines, wild overengineering can run the risk of resulting in needlessly complex interfaces to negotiate a glut of features. Generally speaking, your central duty is to make the jobs of users easier, not to make your own job more interesting!

At the same time, while the most important task is always solving the problem at hand, you are often required to think ahead. While you have a limited ability to think about the needs of your next project, you may have some perspective that allows you to see problems that will arise on the current project in the coming months. Although it is often wise to avoid overengineering, you can still stay ingratiated to your future self by taking some simple precautions to architect intelligently.

In general, the most important decision you can make in this respect is to separate your form from your functionality in your code. By keeping your user interface separated from the nuts and bolts of your tool, you can more easily alter the interface at a later point. This practice can also make it easier to reuse parts of your code in other places, such as creating additional interfaces for a single tool, or creating common functions that you can share across all of your interfaces. One of the most basic advantages to separating out these elements is that you can more easily access the functionality of your tool via script later, which may help you automate an application of the tool to a large number of files or objects. Fortunately, because of how Maya was architected, you should not need to explicitly create special scripting interfaces for plug-ins, and you only need to organize your scripts intelligently to keep their functions open to other tools later.

In short, although you must always get your job done quickly, a few simple but thoughtful decisions can make your code much more pleasant to work with in the future.

## Simplify and Educate

The central task of a tools programmer is to simplify the lives of others—you are often required to transform a painful and tedious task into something fun and streamlined, or to eliminate it altogether. Your users will often not be as technical as a tools developer, and are unlikely to have much awareness of a tool’s internals. In this respect, you need to maintain heightened awareness of the parts of your tool that face your users. A classic example of this problem concerns the labels that you give your GUI controls. While you may internally use terms like “vector,” “interpolate,” or “initialize,” words such as “direction,” “blend,” and “start” are almost certain to be clearer to a larger number of your users. However, it is equally important to educate your users while still simplifying their work.

Just as many developers overcomplicate their tools, it is easy to fall into the trap of oversimplifying. You may often work hard to hide complex information from the user and either do work behind the scenes or guess at the user’s intentions based on input. While this is useful for streamlining tasks, it can also risk closing off a little too much. You may frequently have to strike a balance between making a tool easy to use while also leaving it open for users to apply in corner cases or even possibly extend for special uses.

Thus, it can also be helpful to educate your users while simultaneously simplifying their work. Creative work in digital art forms, such as films and games, is fundamentally still a technical process. To help your team members grow, you can educate them about some of the technical things that may be going on behind the scenes. Consequently, education can help your users offer clearer and more intelligent feedback for future tools. Your users will become better at expressing their needs and requesting useful ways to manually override certain functionality when needed.

Remember also that someone will break your tools at some point, no matter how airtight you attempt to make them. While raising an exception in the console suffices for most of your work, your artists may rarely look at the Command Line output to figure out what they did wrong. Although our examples in this text are fairly limited due to space requirements, you should always try to find useful and friendly ways to communicate input problems to your users.

## TOOLS IN MAYA

Now that we have discussed some of the higher-level considerations you may want to make when developing tools, we can start looking at some example tools in Maya. While Maya’s tools and interfaces are by no means

the gold standard in all cases, there are some important patterns that can be valuable to follow in many cases.

A good starting point to design for your users, then, is to consider following some of these patterns so that your tools operate predictably. Developing in this manner can save you a good deal of trouble when documenting your tools, since users should generally already know how to use them. In other cases, you may want to break from convention where you can improve an interface, but familiarizing yourself with some of Maya's tools can nonetheless be helpful.

## Selection

A good starting point for examining Maya's tools is to investigate the role that selection plays in the execution of a tool. The most basic observation is of course that operations are applied to currently selected objects. Although this may seem incredibly obvious, it is important to contrast it with other applications where you might enter a tool mode and then start clicking on objects, or where tools exist as part of the objects themselves. The following example illustrates this concept clearly:

1. Open Maya and create a new scene.
2. Create a new polygon cube (**Create** → **Polygon Primitives** → **Cube**).
3. Enter the Polygons menu set by selecting **Polygons** from the dropdown menu in the upper left of the main application window.
4. With the cube still selected, enter the Sculpt Geometry Tool (**Mesh** → **Sculpt Geometry Tool**). You can now see the red Artisan brush move over the surface as you move your cursor over the cube. If you click when this red gizmo appears, then you will apply some deformation to the cube's vertices depending on the brush's current mode.
5. Deselect the cube (**Edit** → **Deselect**).
6. Reenter the Sculpt Geometry Tool (the default hotkey for previous tool is **Y**). You will notice that moving your mouse cursor over the cube no longer displays the red Artisan brush. Likewise, clicking on the cube does nothing.

As you can see, the tool is designed to operate on the current selection, rather than to operate in isolation. Most other tools, such as the Paint Skin Weights Tool, all follow the same principle. Following from this point, however, is the more important issue of how Maya's built-in tools work when multiple objects are concerned. There are many tools that require multiple objects to be selected, and they generally follow a common pattern, as the next example demonstrates.

1. Create a new scene.
2. Enter wireframe display mode by pressing the number **4** or by selecting **Shading → Wireframe** from the menu at the top of your viewport.
3. Create a new polygon cube (**Create → Polygon Primitives → Cube**).
4. Create a new polygon sphere (**Create → Polygon Primitives → Sphere**).
5. Create a new polygon cone (**Create → Polygon Primitives → Cone**).
6. With the cone still selected, hold **Shift** and left-click the sphere and then the cube to add them to your selection. The cube's wireframe should highlight green, while the wireframes for the cone and sphere are white. This color scheme indicates that the cube is the last object that was selected.
7. With the objects still selected, open the Script Editor and execute the following short script in a Python tab.

```
import maya.cmds as cmds;
print(cmds.ls(sl=True));
```

As you can see, order matters! The list that was printed has ordered the items based on the order in which they were selected.


```
[u'pCone1', u'pSphere1', u'pCube1']
```

8. With the objects still selected, use the `parent` command. If you have default hotkeys, you can press the **P** key. If not, you can select the menu item (**Edit → Parent**), or enter it into the Script Editor manually. At this point, the cube is deselected, and only the sphere and cone are selected. If you select the cube, you can see that the sphere and cone are now its children, and will follow it around if you move it.

The important point here is that *the last object in the selection list is semantically important: It corresponds to what the tool is doing*. Specifically, the tool's name, `Parent`, indicates what happens to the final object selected: It becomes the parent of any other objects that are also selected.

Imagine that the tool had a more ambiguous name, such as `Link`. A name like `Link` tells you absolutely nothing about what the result will be: The cube might be “linked” to the sphere, but what does that mean? Which one is the parent and which is the child? Is the tool implying that they both affect one another? `Parent` is a much more useful name as it clearly indicates the direction of the operation.

This same pattern is followed in all of Maya's built-in tools that require multiple selections, whether objects, geometry components, or something else: *The last object in the selection list corresponds to the action indicated by the tool's name.*

9. Select only the cube.
10. **Shift-LMB** click the sphere to add it to the selection list.
11. Open the options window for the Transfer Attributes tool, by selecting the small square icon next to its menu item (**Mesh → Transfer Attributes** ).
12. In the Transfer Attributes Options window, select **Edit → Reset Settings** from its menu.
13. In the Transfer Attributes Options window, turn the Vertex Position radio button to **On** and press **Apply**. You can now see that the sphere's vertices are effectively snapped to the surface of the cube. Because the sphere was the last object selected, it is the object to which the tool's settings were applied.

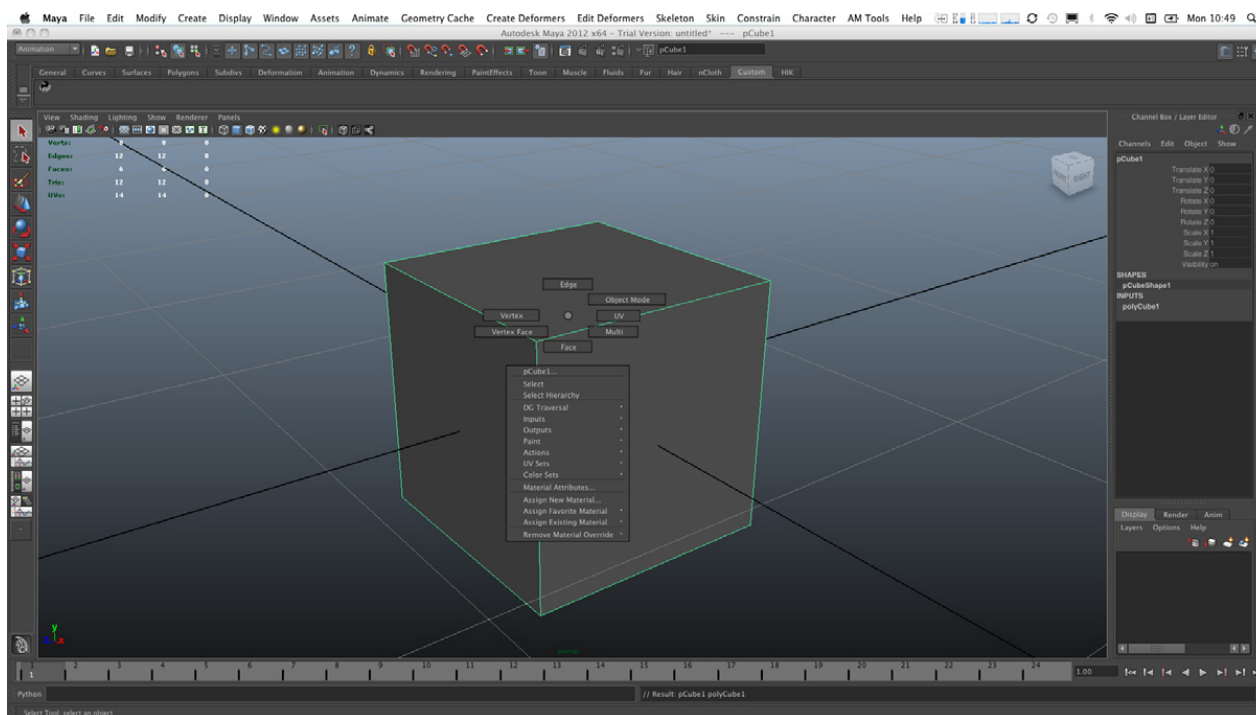
As indicated, you could go through a number of other tools, such as Copy Skin Weights, and you would see the same pattern: The final object selected has the operation applied to it. Whether or not this pattern is objectively the “best” design option, it is important to be aware of it (though it is certainly much easier for users to change what object is the last in their selection lists than to change which object is first, particularly if the selection list is long). In most cases, you will want to try to follow a similar paradigm in any selection-based tools you may create, and should probably have a very good reason for deviating from it lest you risk confusing your users.

## Marking Menus

One of the most unique and important workflow features of Maya's user interface is marking menus, which you saw briefly in the Script Editor in Chapter 1. Marking menus are not only an incredible interface that can outpace hotkeys and standard context-sensitive menus for many users, but are also a valuable example for designing consistency into your tools to leverage a user's muscle memory or spatial understanding. A few brief examples should demonstrate some of the key points regarding marking menus that you may find helpful.

1. Create a new scene.
2. Create a new polygon cube (**Create → Polygon Primitives → Cube**).
3. With the cube selected, move your cursor over the cube and hold the **RMB**. You will see a menu appear, where there is a compass under your cursor and a variety of menu items farther below ([Figure 6.1](#)). While holding the **RMB**, if you move your cursor over the Vertex item in the marking menu's west location (left of the compass center) and release, then you will switch to working on the object in vertex mode.





■ FIGURE 6.1 Basic polygon marking menu.

4. While in vertex mode, move your cursor over the cube. Press and hold the **RMB** and quickly flick the cursor downward, releasing the **RMB** immediately after your downward flick. Performing this action will change the cube to face mode. As you can see, marking menus accommodate flicking as well as a traditional deliberate usage common of right-click context menus.
5. While in face mode, select a face on the cube.
6. While your cursor is over the cube, hold the **Shift** key and **RMB**. **Shift** is the shortcut for performing tool operations in the current selection mode. As you can see, the south item on the compass invokes the Extrude Face tool.
7. With the face still selected, move your cursor over the cube and hold the **Ctrl** key and **RMB**. **Ctrl** is the shortcut for converting one selection to another type. As you can see, in this marking menu, the west item converts the selection to vertices, while the south item converts the selection to faces. If you compare this to the standard **RMB** menu, you will see that the compass items correspond to each component type. In both cases, south corresponds to faces, west to vertices, and so on. In this respect, two different marking menus follow a similar pattern, which can help users

establish muscle memory for the marking menu rather than having to hold the **RMB** and read all of the options individually for a given menu, as is common in many other applications.

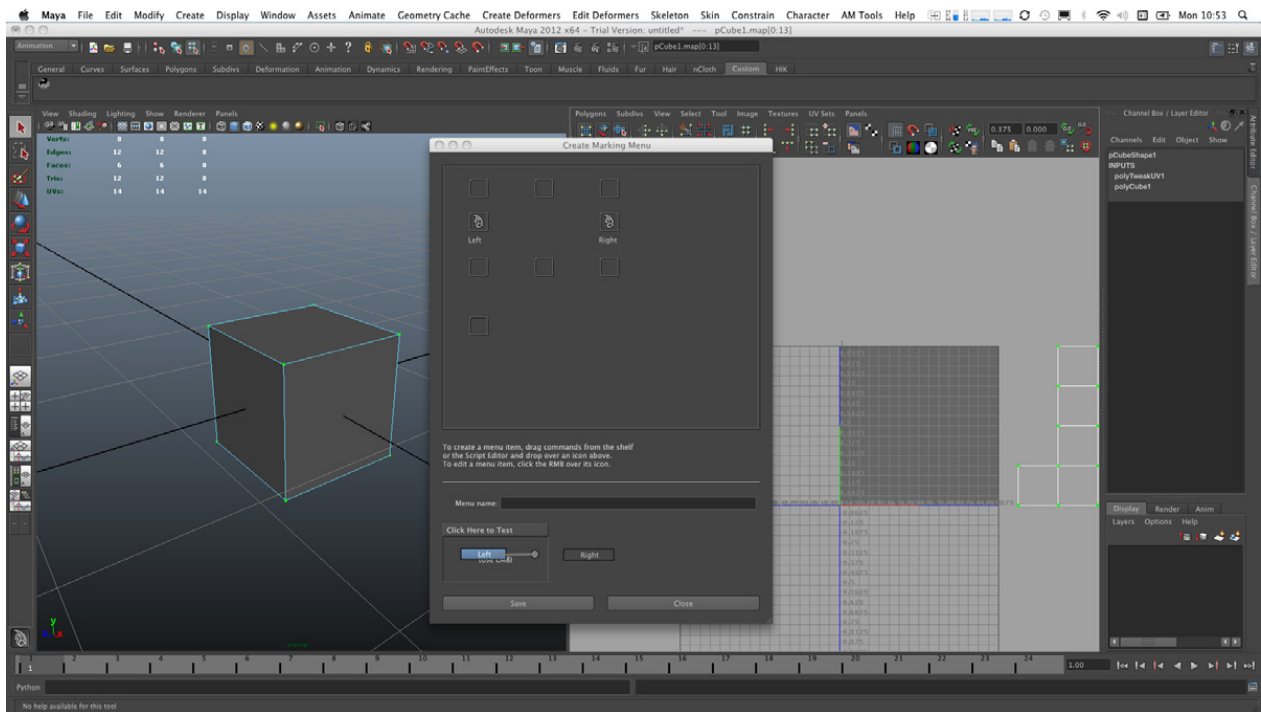
8. Using the **Ctrl + RMB** marking menu, convert the selected face to vertices.
9. Hold **Shift** and **RMB** to look at the tool marking menu for vertices. Just as the Extrude Face tool was south in face mode, the Extrude Vertex tool is south in vertex mode. While not all modes have the same corresponding tool types, each direction on the marking menu compass generally tries to correspond to some equivalent for the other modes where it makes sense.

The key takeaway from this example is the value in establishing a convention to help users predict different actions. Marking menus allow users to work fast not only because they support quick gestures, but also because they provide a means for establishing a visual-spatial relationship with an operation. While you can certainly apply similar principles concerning consistency in other types of GUIs, you can also use this knowledge to help create your own marking menus. The following short example will show you how to create your own simple marking menu.

1. Create a new scene.
2. In the main application window, open the marking menu editor by selecting **Window → Settings/Preferences → Marking Menu Editor**.
3. In the Marking Menus window, click the Create Marking Menu button.
4. In the Create Marking Menu window, the top displays a group of icons representing the different locations in the Marking Menu: north, south, east, west, etc. (Figure 6.2). **RMB** click the icon for the east item and select Edit Menu Item from the context menu.
5. Enter the following lines in the Command(s) input field. Unfortunately, the commands to execute for a marking menu item must be issued in MEL. As you can see, we simply use MEL's `python` command to execute two simple lines of Python code. This particular marking menu item will move the currently selected UVs to the right by one unit.

```
python("import maya.cmds as cmds");
python("cmds.polyEditUV(u=1.0, v=0.0)");
```

6. In the Label field, enter the word "Right" and press the Save and Close button.
7. In the Create Marking Menu window, edit the west menu item to have the following command input. Similar to the command you created in step 5, this marking menu item will move the currently selected UVs one unit to the left.



■ FIGURE 6.2 Testing a custom marking menu in the marking menu editor.

```
python("import maya.cmds as cmds");
python("cmds.polyEditUV(u=-1.0, v=0.0)");
```

8. In the Label field, enter the word “Left” and press the Save and Close button.
9. Keeping the Create Marking Menu window open, create a cube and enter UV editing mode (**RMB + east**).
10. Open the UV Texture Editor window (**Windows → UV Texture Editor**). You should see the default UV layout for the cube.
11. Select all the cube’s UVs.
12. In the Create Marking Menu window, use the **LMB** in the test area (lower left) to try out your new marking menu on the cube’s UVs (Figure 6.2).

The important aspect of what you have done, besides simply creating a marking menu, is leverage the marking menu’s spatial layout to create a gestural control of which the form and function correspond: flicking to the right moves the UVs to the right, while flicking to the left moves them left. As you can see, creating custom marking menus can be a powerful tool for grouping and organizing similar operations. In addition to their speed,

they can also either take advantage of a user's spatial understanding to make controls that are immediately understood, or also create new patterns to reinforce a user's muscle memory. In the case of the default marking menus, for instance, there is nothing inherent about the direction west/left that corresponds with a user's understanding of vertices, but Autodesk has nonetheless created a consistent pattern so users can develop that relationship in the context of Maya.


At this point, you could give your custom marking menu a name, save it, and assign it to a Hotbox quadrant or to a hotkey in the Hotkey Editor (see the dropdown menu in the Marking Menus window labeled "Use marking menu in:"). When you save a marking menu here, it exists as a .mel file in a user's preferences directory. You can leverage this fact to easily share custom marking menus with your users.


## Options Windows

The final aspect of Maya's interface worth discussing at this point is its options windows. As we have reiterated throughout the text, Maya's GUI is, fundamentally, just an interface between the user and the command engine. A majority of the menu items in Maya are simply mechanisms for executing commands.

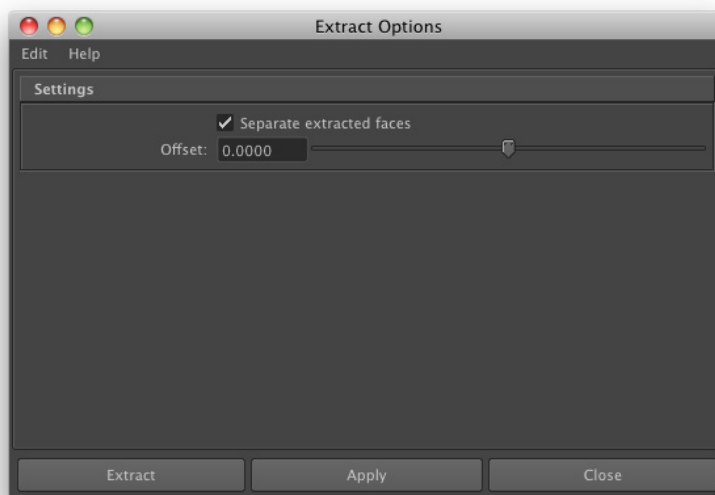
The idea behind this principle is that form and function are almost wholly separated: The functionality of a tool exists in the command, while the form is a GUI control created via script. In many of your own tools, you can simply include your form and functionality in the same module, but will almost certainly want to separate them into different functions at the least. Thus, options windows in Maya are simply interfaces for exposing different command flags to the user to set up arguments.

Just as with marking menus, Maya has set up some conventions that are generally worth following for most simple tools. Take a look at some different options windows in the following example.

1. Enter the Polygons menu set (**F3**).
2. Select the small square icon to the right of the Extract tool in the main application menu (**Mesh** → **Extract** ). Looking at the options window that appears, there are a few items worth noting. First, the title of the window is "Extract Options." Second, the window menu has an Edit submenu and Help submenu. Take a moment to look at the contents of each submenu. The central part of the window exists in a group entitled "Settings." Finally, the bottom of the window has three buttons. From left to right they are Extract, Apply, and Close. The Extract button has the effect

- of performing the operation and closing the window simultaneously, while the Apply button simply performs the operation.
3. Select the small square icon to the right of the Smooth tool in the main application menu (**Mesh** → **Smooth** ). Looking at the options window that appears, you should see the exact same core parts that you saw in the Extract tool, plus additional groups of related items in the center of the window (Figure 6.3).

At this point, you could continue looking through a number of the tools in the main application window and see the same pattern repeated. Apart from simply establishing a consistent pattern across all windows, there are a couple of other important things happening here. First, each window has a menu option to reset it to its default settings, as well as a way to get help. Second, each tool has a button to apply the operation without closing the window (middle), as well as an option to apply the operation and close the window (far left). If you are creating a tool that performs an operation in this way, it can be helpful to provide your user with both options. Moreover, as we will see in the following chapters, some simple organization can allow you to easily reuse code when you implement this or any other common options window patterns. This sort of problem is the perfect example of something that is worth the trouble of architecting well, since you can be reasonably sure you will want to create many different windows with the same menu items and button layouts.



■ **FIGURE 6.3** Options window for the Extract tool.

## **CONCLUDING REMARKS**

Having discussed the higher-level topics related to the creation of tools, such as how to get in the habit of focusing on your users and where to look for starting ideas, we can now move into lower-level topics concerning how to actually create some different interfaces. In the remaining chapters of this part of the book we will create a variety of user interfaces.