CHAPTER

Self-Revealing Gestures

20

The best way to teach somebody something is to have them think they're learning something else.

—Prof. Randy Pausch, "The Last Lecture"

DESCRIPTION

Self-revealing gestures are a philosophy for design of gestural interfaces that posits that the only way to see a behavior in your users is to induce it (*afford* it, for the Gibsonians among us). Users are presented with an interface to which their response is gestural input. This approach contradicts some designers' apparent assumption that a gesture is some kind of "shortcut" that is performed in some ephemeral layer hovering above the user interface. In reality, a successful development of a gestural system requires the development of a *gestural user interface*. Objects are shown on the screen to which the user reacts, instead of somehow intuiting their performance. The trick, of course, is to not overload the user with UI "chrome" that overly complicates the UI, but rather to afford as many suitable gestures as possible with a minimum of extra on-screen graphics. To the user, she is simply operating your UI, when in reality, she is learning a gesture language.

APPLICATION TO NUI

A common immediate reaction to a high-bandwidth, multi-finger input device is to imagine it as a gestural input device. Those of us in the business of multi-touch interface design are often confronted with comparisons between our interfaces and the big-screen version of MIT student John Underkoffler's Ph.D. work: Minority Report. The comparison is fun, but it certainly creates a challenge—how do we design an interface that is as high-bandwidth as has been promised by John and others, but that users are able to immediately walk up to and use? The approach taken by many designers is to try to map a system's functionality onto the set of gestures a user is likely to find intuitive. Of course, the problem with such an approach is immediately apparent: The complexity and vocabulary of the input language are bounded by your least imaginative user.

At a more fundamental level, the goal of providing natural and intuitive gestures that are simultaneously complex and rich seems to contain an inherent contradiction. How can something complex be intuitive? What we have found in practice is that to achieve our goal of an interface that feels natural to its users, we must actually provide them with a UI. The trick, of course, is to do so in a way that is minimally intrusive and that makes it seem to the user as if she is "discovering" the gestures. To this, we will apply many of our design principles, the most salient of which we described in Chapter 10: we will scaffold our user experience.

LESSONS FROM THE PAST: CONTROL VS. ALT HOTKEYS

For a little fun (and perhaps some disillusionment), make an appointment at your local Genius Bar at an Apple store and bring along your OSX-based computer. When it's your turn, kindly ask the genius, "I can't figure out how to use this computer—can you please show me the basics?" As they reach for the trackpad, gently correct them—"Sorry, I meant how to use it using only the keyboard."

It is interesting how devotees to one OS or the other can take on a religious zeal about their choice. In truth, there are very few instances where someone with HCI training can point to a clear winner in the Mac OS vs. Windows debate. Different elements of each have merit. But one instance where Windows is the clear, indisputable winner is in the way hotkeys are designed and taught. In this lesson from the past, we will examine the Windows approach to hotkeys and take away a clear understanding of the merits of the approach.

Many users never notice that, in Microsoft Windows, there are two completely redundant hotkey languages. These languages can be broadly categorized as the *Control* and *Alt* languages. It is from comparing and contrasting these two hotkey languages that we draw some of the most important lessons necessary for self-revealing gestures.

Control Hotkeys and the Gulf of Competence

We consider first the most-used hotkey language: the *Control* language. Although the particular hotkeys are not the same on all operating systems, the notion of the control hotkeys is standard across many operating systems: we assign some modifier key (Function, Control, Apple, Windows), putting the rest of the keyboard into a mode. The user then presses a second (and possibly third) key to execute a function. Many users know a couple of these hotkeys—such as CTRL + X to cut and CTRL + C to copy (APPLE + X on a Mac, but recall we're talking about Windows here). What interests us is how a user learns this key combination.

Control hotkeys generally rely on two mechanisms to allow users to learn them. First, the keyboard keys assigned to their functions are often lexically

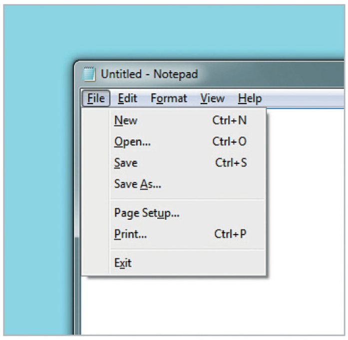

FIGURE 20.1

The Control hotkeys are shown in the File menu in Notepad. Note that the key choices are selected to be intuitive (by matching the first letter of the function name).

intuitive: CTRL + P = print, CTRL + S = save, and so on. Figure 20.1 shows some hotkeys from the Notepad application.

Relying on intuitiveness works well *for a small number of keys*, but it breaks down quickly—if CTRL + C means "copy," then what is the hotkey for "center"? This is roughly parallel to the naïve designer's notion of gesture mappings: we map the physical action to some property in its function (if we want "help," draw a question mark!). However, we quickly learn that this approach does not scale: Frequently used functions may overlap (consider "copy" and "cut"). This gives rise to shortcuts such as CTRL + H for "find next" (CTRL + R is "center", in case you were racking your brain). We also note the use of function keys as CTRL shortcuts—even though they don't actually use the CTRL key, they are still notionally CTRL shortcuts, as we shall see.

Because intuitive mappings can take us only so far, the menu provides the second mechanism for hotkey *learning*: the functions in the menu system are labeled with their hotkey invocation. This approach is a reasonable one. We provide users with an in-place help system labeling functions with a more efficient means of executing them. However, a sophisticated designer must ask themselves, "What does the transition from novice to expert look like?"

In the case of Control shortcuts, the novice-to-expert transition requires a leap on the part of the user: we ask her to first learn the application using the mouse, pointing at menus and selecting functions spatially. To become a power user, she must then make the conscious decision to stop using the menu system and begin to use hotkeys. When the user makes this decision, it will at first come at the cost of a loss of efficiency, as she moves from being an expert in one system, the mouse-based menus, to being a novice in the hotkey system. We term this cost the *gulf of competence*. The graph in Figure 20.3 demonstrates this idea—at the time that the user tries to switch from mouse to keyboard, she slows down.

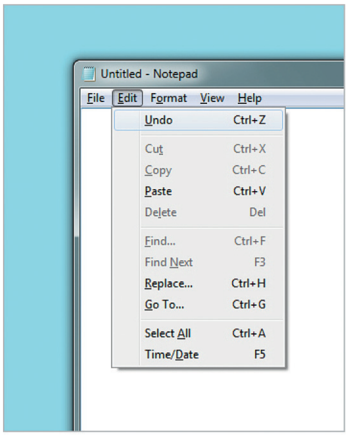

FIGURE 20.2

The Control hotkeys are shown in the Edit menu in Notepad. The first-letter mapping is lost in favor of physical convenience (CTRL + V for paste) or name collisions (F3 for find next—yes, F3 *is* a Control hotkey under our definition, which will be more clear soon).

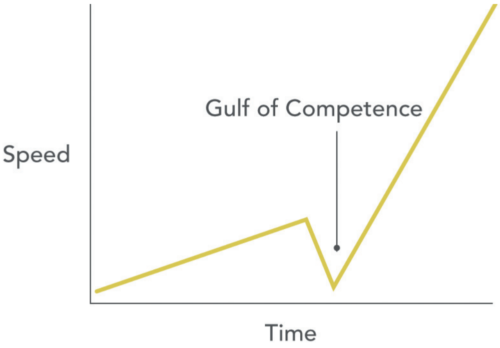

FIGURE 20.3

The learning curve of Control hotkeys: The user first learns to use the system with the mouse. They he must consciously decide to stop using the mouse and begin to use shortcut keys. This decision comes at a cost in efficiency as he begins to learn an all-new system. This cost is the "gulf of competence."

The gulf of competence is easily anticipated by the user: He may know that hotkeys are more efficient, but they will take time to learn. We are asking a busy user to take the time to learn the interface. The gulf of competence is a chasm too far for most users. Only a small set ever progress beyond the most basic control hotkeys, forever doomed to the inefficient world of the WIMP. Thankfully, we have a hotkey system that is far easier to learn: the Alt hotkeys.

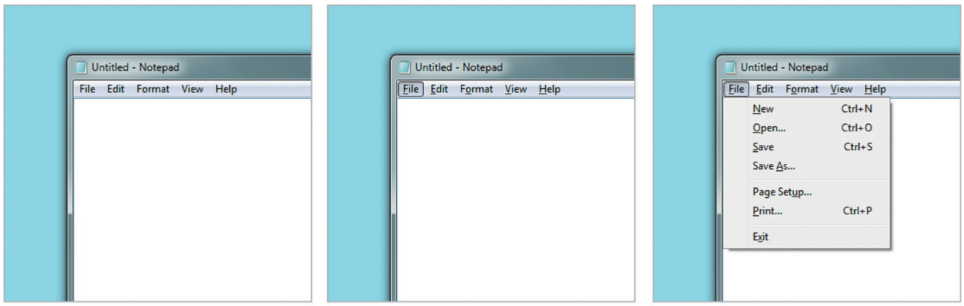

FIGURE 20.4

A novice Alt hotkey user's actions are exactly the same as an expert's: no gulf of competence. On-screen graphics guide the novice user in performing an Alt hotkey operation. Left: The menu system. Center: The user has pressed "Alt." Right: The user has pressed "F" to select the menu.

Alt Hotkeys and the Seamless Novice-to-Expert Transition

While the Control hotkeys rely on either intuition or the willingness to jump the gulf of competence, a far more learnable hotkey system exists in parallel that addresses both of these limitations: the Alt hotkey system. Like any hotkey system, the Alt approach modes the keyboard to provide a hotkey. Unlike the Control keys, however, on-screen graphics guide the user in performing the key combination (Figure 20.4).

Because the Alt hotkeys guide the novice user, there is no need for the user to make an input device change: He doesn't need to navigate menus first with the mouse, then switch to using the keyboard once he has memorized the hotkeys. Nor do we rely on user intuition to help them to "guess" Alt hotkeys.

The Alt hotkey system is a self-revealing interface, because there is no need for a help system or instructions—the actions are simply shown and followed. Better yet, the physical actions of the user are the same as the physical actions of an expert user—both press ALT + F + O to open a file. There is no gulf of competence. In applying this lesson to the gesture space, there is a highly relevant piece of work that should be examined in detail: marking menus.

Marking Menus: The First Self-Revealing Gestures

Marking "menu" is a bit of misnomer—it's not actually a menu system at all. In truth, the marking menu is a system for teaching pen gestures. For those not familiar with them, marking menus are intended to allow users to make gestural "marks" in a pen-based system. The pattern of these marks corresponds to a particular function. For example, the gesture shown in Figure 20.5 (right) leads to an "undo" command. The system does not rely at all on making the marks *intuitive*. Instead, marking menus provide a hierarchical menu system (left in Figure 20.5). Users navigate this menu system by drawing through the selections with the pen.

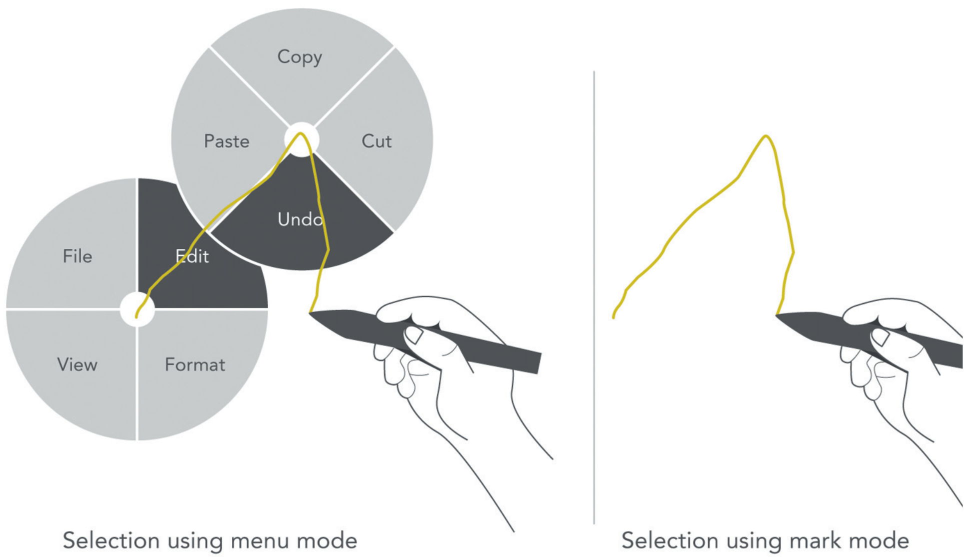

FIGURE 20.5

The marking menu system (left) teaches users to make pen-based gestures (right).

As they become more experienced, users do not rely on visual feedback, and eventually transition to interacting with the system through gestures, and not through the menu. It's important to understand that there is no difference in the software between novice and expert "modes"—the user simply uses the system faster and faster. Because there is a 200 ms delay between the time the pen comes down and when the menu becomes visible, novices declare themselves by doing exactly what comes naturally—hesitating.

Just like the Alt menu system, the physical actions of the novice user are physically identical to those of the expert. There is no gulf of competence, because there is no point where the user must change modalities and throw away all his prior learning. So how can we apply this to multi-touch gestures?

DESIGN GUIDELINES

Self-Revealing Multi-Touch Gestures

So it seems someone else has already done some heavy lifting regarding the creation of a self-revealing gesture system. Why not use that system and call it a day? Well, if we were willing to have users behave with their fingers the way they do with a pen, we'd be done. But the promise of multi-touch is more than a single finger drawing single lines on the screen. For all of the reasons we described in Chapter 18 and

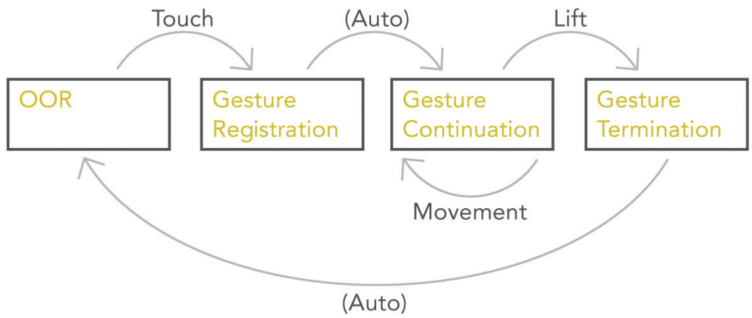

FIGURE 20.6

The three stages of gestural input and the physical actions that lead to them on a pen or touch system, as we described in Chapter 18. OOR is "out of range" of the input device.

throughout the book, we need to do better. We must consider: what would a multi-touch self-revealing gesture system look like?

First, we should recall from Chapter 18 the stages of a gesture. A gesture consists of three stages: *registration*, which sets the type of gesture to be performed, *continuation*, which adjusts the parameters of the gesture, and *termination*, which is when the gesture ends (Figure 20.6).

In the case of pen marks, registration is the moment the pen hits the tablet, continuation happens as the user makes the marks for the menu, and termination occurs when the user lifts the pen off the tablet. Seems simple enough. When working with a pen, the registration action is always the same: the pen comes down on the tablet. The marking menu system kicks in at this point, and guides the user's continuation of the gesture—and that's it. The trick in applying this technique to a multi-touch system is that the registration action varies: it's almost always the hand coming down on the screen, but the posture of that hand is what registers the gesture. On Microsoft Surface, these postures can be any configuration of the hand. Putting a hand down in a Vulcan salute could map to a different function than putting down three fingertips, which is different again from a closed fist. On less-enabled hardware, such as that supported by Windows 7 or the iOS, the variation is limited to some combination of the relative position of multiple fingertip positions. Chapter 25 describes this in detail. Nonetheless, the problem is the same. We now need to provide a self-revealing mechanism to afford a particular initial posture for the gesture, because this initial posture is the registration action that modes the rest of the gesture. Those marking menu guys had it easy, eh?

But wait—it gets even trickier.

In the case of marking menus, the on-screen affordance was needed only for the continuation phase, and it would pop up around the pen following registration. On a multi-touch system, because we have to give affordances before the registration phase, we need to tell the user which posture to go into *before the user touches the screen.* With nearly all of the multi-touch hardware on the market today, the hand is out of range right up until it touches the screen (see Chapter 15).

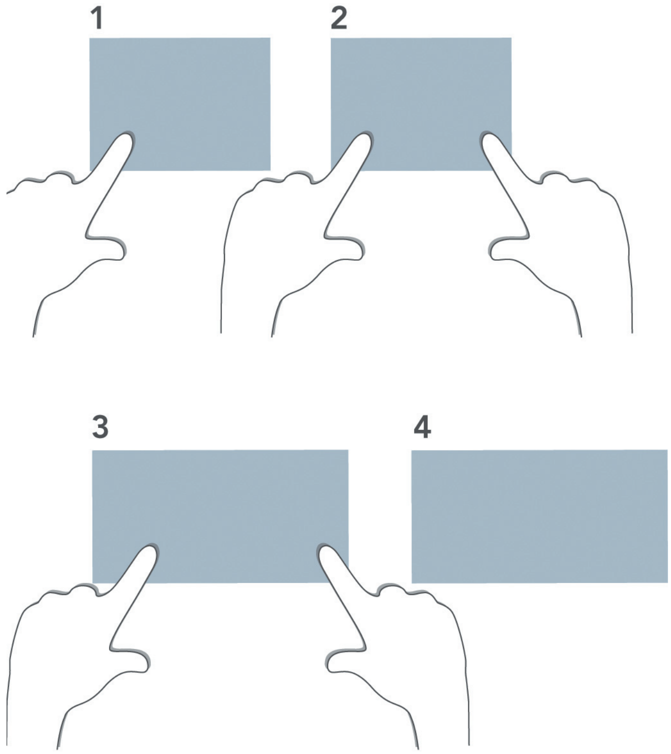

FIGURE 20.7

A theoretical gesture sequence for resizing a photo: Touch with one finger, then add another at the border, pull them apart, and lift.

One must consider affording each of the registration and continuation phases (the termination phase, which is almost always lifting the hand from the device, more or less affords itself). As you will learn in Chapter 27, there is no such thing as a "natural gesture," with the exception of moving things from one place to another, or "direct manipulation." A successful self-revealing gesture system will utilize this to afford actions, similar to the marking menu. Users of marking menus don't need tutorials. It was obvious: select things by tracing over them with the pen. Similarly, physical metaphors (things that slide, things that can be dragged, rolled, etc.) all afford movement.

An approach we advocate is one that we have dubbed "just-in-time chrome," which we present publicly here for the first time. To understand it, let's begin by proposing a gesture to stretch a photo in one of its dimensions. It goes like this: touch the photo with one hand, then touch the border of the photo with a second hand, stretch the hands apart, and lift (Figure 20.7).

This gesture is almost impossible to guess (we tested it with dozens of users). Many had experience enlarging pictures on iPhones, but the idea that they needed to put

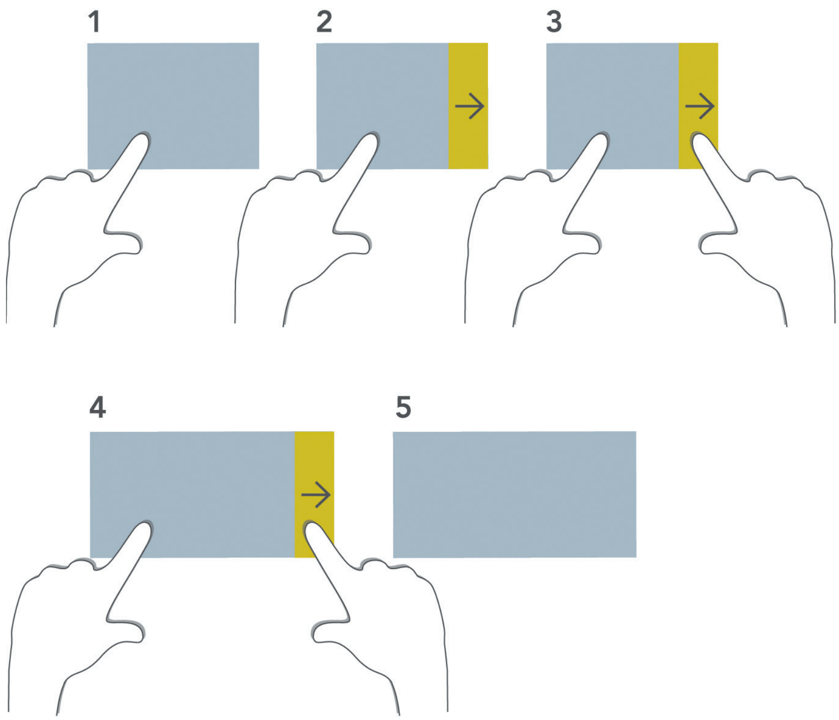

FIGURE 20.8

The same gesture sequence as above, this time with just-in-time chrome to help the user along.

their fingers in a particular location to stretch the photo horizontally wasn't guessed. But if we add just-in-time chrome, the sequence looks like that shown in Figure 20.8.

This gesture, in contrast, is incredibly easy to guess. The participants in our experiments got it right away, almost every time.

Just-in-time chrome begins with the assumption that there is an action that the user can perform that will tell the system where and what she wants to engage with. In the case above, the UI is shown only when the user touches the photo. To avoid the gulf of competence, the gesture must therefore also begin with a single finger touching the photo. The basic intuition here is to let the user touch the screen to tell us where it is she wants to perform the gesture. Next, show on-screen affordances for the available postures and their functions, and allow the user to register the gesture with a second posture, in approximately the same place as the first. From there, have the user perform manipulation gestures with the on-screen graphics, since, as we learned oh-so-many paragraphs ago, manipulation gestures are the only ones that users can learn to use quickly and are the only ones that we have found to be truly "natural."

These affordances are obvious for the continuation phase, but less so for registration. To address registration affordances, we recommend using the hover state of

FIGURE 20.9

UI affordances are shown on tap. The user is told to put down one finger to resize the photo or two fingers to scroll or zoom. Whatever mechanism you use, applying the principle of scaffolding and the lessons of these earlier attempts at self-revealing user interfaces will lead you to far more successful multi-touch and gesture UI's.

your hardware (see Chapter 15), if you've got one. If you don't, then reserve a one-finger tap as a "I need more information" gesture. Thanks to decades of mouse use, this is the first action that users always take when they are trying to figure out what to do. An example of this is shown in Figure 20.9.

Just-in-time chrome is just one method of making your gestural interface self-revealing. The key is to consider affording registration actions as well as continuation actions. An alternative approach was investigated by Freeman and his colleagues at Microsoft: putting a layer of help on top of your application to afford both registration and continuation. While we don't particularly advocate for this approach in general, it is worth considering for certain applications.

Must

- Never rely on an action being "natural" (a.k.a. "guessable"). It's not.
- The only exception to the above is "direct manipulation"—users can and will guess to grab something and move it somewhere else.
- For gestures, present objects on-screen to which users respond.
- Utilize direct manipulation as an on-screen affordance in all cases. Want to afford the user putting their hand down in a Vulcan salute? Put a Vulcan-salute shaped button on the device for them to touch.

Should

- Re-use similar visual affordances to afford the same gestures over and over again. This is commonly known as a "user interface."
- Consider affording both registration and continuation phases of the gesture. This is a "should" only because your gesture system may have only one registration action, such as landing a single finger on the device.

Could

- Use hover capabilities of your input device (if present) to preview available actions before the user actually comes in contact with the display.
- Think about teaching more gestures over time. Consider how to layer your user interface in the same way game designers layer functionality over time.

SUMMARY

The biggest problem with making your gestures self-revealing is getting over the idea that gestures are somehow natural or intuitive. We have seen over and over again that users cannot and will not guess your gesture language. To overcome this, put UI affordances on the screen to which they can react.

UNNATURAL USER INTERFACES

Gord Kurtenbach

Autodesk

I often give a lecture entitled "un-natural user interfaces." This particular title is a setup to make people think I'm going to speak about examples of bad, "unnatural" user interfaces and how we need to design them to be more natural and intuitive. However, the surprise and hopefully entertaining twist of the lecture is that I claim there is no such thing as natural or intuitive interfaces. Effective user interface design is very carefully controlled skill transfer—we design interfaces so users can take their skills from one situation and re-apply them to a new situation. The classic example is the computer desktop. Users who are new to computers transfer their existing skills with the manipulation of real physical files and folders to the computer realm. It can be argued that moving around physical files is "natural," but that too is a learned skill—remember playing with blocks as a child? Consider another more "unnatural" example: Suppose we have software A, which new users find very difficult and unintuitive to learn, but it has been learned and is used by a large population of users. Software B copies A's interface style, hotkeys, etc. The result is that users of A can easily learn to operate B because the interface is familiar. In other words, they transfer their skills with A over to B. Learning software A from scratch did not feel natural or intuitive, but once learned, transferring those skills makes learning B "natural and intuitive." Nice trick!

This chapter describes how this fundamental and powerful concept of skill transfer applies to gesture input. Gesture input holds the potential of being vastly expressive, especially combined with multi-touch. However, without some sort of mechanism to help users learn these complex interactions, these interactions become as difficult as learning a sign language. The authors reveal the secret to successful interface design with gestures: A mechanism must be provided so users can easily learn the gesture set. To accomplish this, skill transfer is used in a powerful way. For example, an interaction technique called "marking menus" is described, where a user's skills with a graphical menu can used to magically teach a vocabulary of arbitrary "zig-zag" gestures. In similar fashion, with a method called "just-in-time chrome," users' skills with interpreting feedback and direct manipulation transfer directly into a rich vocabulary of multi-touch gestures.

Understanding the concept of self-revealing gestures is absolutely critical for the successful use of gestures in a user interface. Simply ask the following question for each gesture in your interface: How will the user learn it? Some gestures reveal themselves because we see others use them, like the ubiquitous “page-turn stroke” and “pinch-zoom.” However, to harness the potential of richer, larger gesture sets the concepts introduced in this chapter are paramount.

Author Biography

Dr. Gordon Kurtenbach is the Director of Research at Autodesk (www.autodeskresearch.com), where he oversees a large range of research including human-computer interaction, graphics and simulation, environment and ergonomics, high-performance computing, and CAD for nanotechnology. Dr. Kurtenbach has published numerous research papers and holds over 40 patents in the field of human-computer interaction. His work on gesture-based interfaces, specifically “marking menus,” has been highly influential in HCI research and practice. In 2005, he received the UIST Lasting Impact Award for his early work on the fundamental issues combining gestures and manipulation.

FURTHER READING

Grossman, T., Dragicevic, P., and Balakrishnan, R. Strategies for accelerating on-line learning of hotkeys, *Proceedings of CHI*, 2007, 1591–1600. In this work, Grossman et al. study various methods for teaching accelerator keys.

Kurtenbach, G. *The Design and Evaluation of Marking Menus*, Ph.D. Thesis. Gord Kurtenbach, working with his advisor, Bill Buxton, at the University of Toronto, developed the marking menu. A series of publications describes the original concept, stages of learning, and how they can be integrated into interfaces. While each was published separately, it is his Ph.D. thesis that describes them all together in great detail.

Freeman, D., Benko, H., Morris, M., and Wigdor, D. ShadowGuides: Visualizations for in-Situ Learning of Multi-Touch and Whole-Hand Gestures, *Proceedings of ACM Tabletop*, 2009. In this work, Freeman and his colleagues make two major contributions. The first is a set of representative gestures that spans the space of possible gestural input to a surface-like device. The second is a teaching method they dub “ShadowGuides,” for teaching gestures with on-screen affordances. In this chapter, we have emphasized that UIs built in to the experience should afford gestures. ShadowGuides, in contrast, provide a visualization that sits on top of the UI. While we don’t recommend this approach in general (it represents earlier thinking in our work), it does nicely break down the idea of providing on-screen affordances for each of the registration, continuation, and termination phases of the gestures they teach.