CHAPTER

Touch versus In-Air Gestures

15

My fellow Americans, I'm pleased to tell you today that I've signed legislation that will outlaw Russia forever. We begin bombing in five minutes.

—Ronald Reagan

DESCRIPTION

As we have described in Chapter 3, technologies inhabit an ecological niche, in that each represents a set of potential uses and markets. Just as those who failed to understand the true utility of touch input predicted the death of the mouse and keyboard, new technologies, such as Xbox Kinect and the Sony EyeToy, that offer touchless gesture input might incorrectly be believed to be a replacement for touch and touch gestures. Certainly, devices that enable users to gesture in air open a seemingly all-new world of interaction potential—and perhaps new ecological niches as well. To the well-trained expert, however, it can be seen that in-air interactions share a great deal with those based on touch. This difference is easily expressed and understood using tools presented in this book, and will enable the quick and easy transfer of design knowledge from touch-based to touchless gesturing.

APPLICATION TO NUI

As we have seen, NUI is not a technology, but rather an experience that can be created using technology. Different sensing technologies are suited to different situations. As one rather well-known researcher is fond of saying, "Everything is best for something, and worst for something else." In-air gesturing, which can be sensed by devices like the Sony EyeToy and the Microsoft Kinect, is suitable in some circumstances where touch input is less so. In living rooms, with digital signage, and in other environments where walking over and touching a screen might detract from the experience, in-air gesturing helps close that gap.

LESSONS FROM THE PAST

The wrong way to think about in-air gesturing is "touch at a distance." The right way to think about it is as a unique input/output paradigm, which must be designed separately and differently from a touch one. That being said, many of the tools we have described elsewhere in this book are applicable to this input methodology as well as to touch input. We will focus on one particular differentiating element: that in-air gestures suffer from a "live mic," similar to the one that Ronald Reagan apocryphally encountered in the early 1980s when he delivered the quote that opens this chapter. In the case of in-air gesturing, this refers to the always-on nature of in-air gesturing and the need to "clutch"—to differentiate physical actions that are intended to drive the computing system from those that are not. In the case of touch computing, the clutch comes when the user lifts her hand from the digitizer. In most cases, when the hand is in the air, the system can't see it. This is true too of the mouse: Lifting your hand from the mouse (or lifting a mouse in the air) causes it to stop sending position change information to the system.

In Chapter 12, we described the state-transition model of input devices. Using this model, we explained that touch is fundamentally different from mouse input in one very important way: Typical touch input has no tracking state, or zone where the touch is registered by the system, but not yet engaged.

We pointed out that modern operating systems actually rely rather heavily on a tracking state and that designing well for touch input would mean designing an all-new UI that does not rely on the tracking state to provide a preview. If you thought that was hard, wait till you see this: in-air gesture systems are typically *one-state* input devices!

You can see now the challenge faced by those designing games and experiences for such input devices: There is no mechanism in the hardware that will differentiate between movements that are intended as gestures to the system and those that are not. When designing a touch application, there is little concern about this—if the user needs to cover his mouth to sneeze, scratch his head, gesture to another person in the room, wring his hands, stretch, or any other of a thousand different non-input actions, there is little worry that your sensor, the touchscreen, will send these to your app as "touches." You can simply assume that these will be filtered out by the simple fact that the user will stop touching the screen while doing them. This is not the case for in-air systems. The sensor will be buzzing away, like the camera and microphone pointed at President Reagan, happily sending all of these events to your application or platform.

This fact makes it easy to encounter errors in recognition of the types described in Chapter 28: where either the user does not intend to perform a gesture but the system recognizes one anyway (*false positive* errors), or the user believes she has performed a gesture but the system does not recognize it (*false negative* errors). These two problems can happen just as easily with touch as with in-air systems, but because of the "live mic" problem, they are likely to happen more often. To understand this, imagine the simple task of pushing a virtual button using both a touch

FIGURE 15.1

Selection using a pigtail gesture.

input and an in-air input device. In the touch case, this is relatively simple to accomplish: The user puts a finger on top of the button and lifts it within its bounds. In the in-air case, it is decidedly more complicated. The naive designer might say, "When the user points at the button, call it 'pushed.'" But this won't work—it's like aiming with a fully automatic rifle with the trigger stuck on "fire." As the user lifts a finger toward the screen, it is pointing the whole way and would be "pushing" every button as the finger is lifted. How then do we distinguish between aiming and firing in a one-state input device? Several approaches have been explored for in-air and other similar contexts that are worth examining.

Reserved Actions

The reserved-action approach takes a particular set of gestural actions and reserves them so that those actions are always designated as either navigation or commanding. For example, Ken Hinckley and his colleagues at Microsoft Research examined the problem of how to distinguish between strokes intended as commands and those intended as drawings in a pen interface. They reserved the "pigtail" gesture for issuing the "invoke menu" command (Figure 15.1).

This approach has the advantage of following an ink-based system without requiring a menu or mode. Want to draw a circle around something? Just go ahead and do it. Want to select something? Draw that same circle, but add a little pigtail to the end, and the system interprets it as a command. The disadvantage of this approach is also equally clear: The user of such a system could never draw a pigtail, because that is a reserved action. False positives are likely, since users of a drawing program are likely to draw strokes that cross themselves fairly often without intending them to invoke the command.

In-air gesturing has also been shown with reserved actions. Grossman and his colleagues invented *Hover Widgets*, a set of gestures performed by the user of a tablet PC with the stylus hovering in the air above the screen (Figure 15.2). Users could use the tracking state on the tablet PC as it was intended for a classic UI, as a preview for what would happen when they transitioned to engaged by touching the pen to the device. If they happened to move the pen in a particular pattern above the device, however, they would invoke commands.

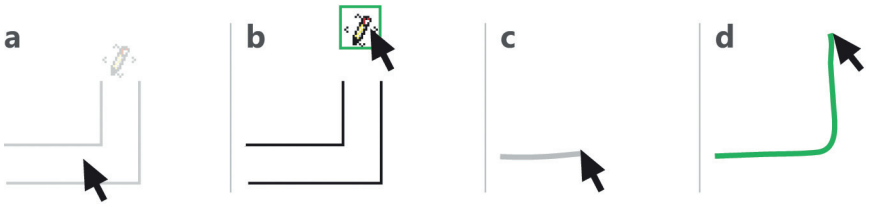

FIGURE 15.2

The Hover Widgets are invoked by moving the stylus in a particular pattern in the air above the device.

In this case, false negatives are more likely than false positives. This is because the "hover" zone above a tablet PC is rather small, and it is likely that the user will exit that zone during the gesture. However, the principle is sound: This reserved action is unlikely to be performed in the normal course of pointing, and is thus on the right track.

Reserving a Clutch

Similar to but different from a reserved action is a reserved clutch. This is a special class of action dedicated to creating a pseudo-tracking state. Similar to the clutch in a car, which connects or disconnects the engine from the wheels, a gestural clutch is an indicator that you want to start or stop recognition of the gesture. An obvious method for such a clutch would be an invisible plane in front of the user. When the user moves her hands in the air, they are tracked. To engage, they must push past that invisible plane.

The advantage of a clutch over a reserved action is immediately apparent: It provides an *a priori* indicator to the recognizer that the action to follow is either intended to be treated as a *gesture* or not. As we will learn in Chapter 18, the earlier your recognizer can differentiate gestures from non-gestures, the more accurate and positive your user experience will be. Another nice effect of a clutch is that it enables the complete gesture space to be used. In a system using Hover Widgets, for example, what if the user just happened to want to move the pen over the surface of the digitizer, but didn't want the Hover Widget action to happen? A clutch provides an obvious mechanism to differentiate the two situations, without having to set aside whole classes of actions.

Unfortunately, a virtual clutch may also cause errors. In the example of the invisible plane, if the plane is too close to the user, it's likely that he'll cross it unintentionally and frequently (false positive error). If the plane is too far away, it's likely that he'll fail to cross it when he intends to (false negative error). Unfortunately, there can be little doubt that these "too close" and "too far" zones will not border opposite sides of a "just right" zone, but rather that they will overlap and be different for different users, or even the same user over time.

A better example of a reserved clutch is the use of a "pinch." When the hand is in a regular pose, it is tracked. When the user pinches the finger and thumb together, it causes a transition to the engaged state. This approach is less likely to be subject to false positive errors, since this action is one that is unlikely to occur in the natural course. It is also less likely to be subject to false negative errors, since it is fairly easily detected and differentiated from other actions. It does, however, have the obvious disadvantage that you remove the pinch (or whichever gesture you choose) from the set of gestures you could otherwise use for other things in your system.

Despite the reduced probability of false negatives or positives, there may well be occasions in which a reserved action or clutch is not feasible. An alternative is to make use of multi-modal input.

Multi-Modal Input

Another solution to the live mic problem is multi-modal input. To understand this solution, we'll turn to the iPhone.

A question frequently asked of us by designers of touch applications is "Why does the iPhone have a button?" An engineer would point out that it allows the system to turn off the touch sensor, which saves power. Aside from this reason, there would still be a need for one that may now obvious to you in light of the live mic problem: without the button, how could the user be guaranteed to always be able to exit an application and return to the home screen? A reserved action might work (say, any time the user were to draw the Apple logo, or slide 5 fingers together on the screen, the application would exit and return to the home screen), but this would be problematic for the same reasons we have outlined above. Instead, multi-modal input is used: touch input is sent to the application, while hardware buttons control universal parameters (e.g., volume) and basic navigation (e.g., go home, open music controls).

Another example of multi-modal input commonly attempted with in-air gesture systems is to use speech input in combination with gesture. The "put that there" system was developed at MIT in the late 1970s and early 1980s. In it, the user could point at a screen, and the point was tracked. To transition between tracking and engaged states, the user issued speech commands. For example, to move an object from one place on the screen to another, the user would point at it, then say "Put that," which moved the system into "engaged." The user would then move her finger to point at the new location, and say "there." The advantage of multi-modal input is also obvious—it does not reduce the vocabulary of the primary modality the way that a reserved action or clutch do.

Another example of multi-modal input is the use of the keyboard to mode mouse clicks or drags. Drag an icon in windows and you will move it from one place to another. Hold down the CTRL key while you drag it, and it will make a copy instead of moving the original. Using input devices and methods in combination with one another may on its face seem more complex, but it can in actuality

greatly simplify the problem of how to differentiate inputs and solve the live mic problem.

DESIGN GUIDELINES

Understand the live mic problem and how you will need to design for it. Consider the lessons of this chapter and the live mic problem. Aside from this, consider all of the lessons contained elsewhere in this book: for the most part, they apply equally well to both in-air gesturing and touch.

Must

- Understand the *live mic* problem as it applies to in-air gestures and design your system accordingly.
- Include mechanisms to differentiate between those actions that the system should recognize and those it should not.

Should

- Consider the solutions we have proposed here: reserving actions or clutches, or using multi-modal input to solve the live mic problem.
- Carefully consider and study your live mic solution. Do not assume that an action that you can perform easily will also be performed easily by your users.
- Consider the problems of both false positives and false negatives in defining your solution.

Could

- Design your system so that there is no need to solve the live mic problem by completely redesigning the UI from the ground up, and taking this issue into account.

SUMMARY

While touch and in-air gesturing may at first seem quite different from one another, there is only one significant subtlety that differentiates them: the live mic problem. Fully understanding and addressing it will allow you to apply the other lessons from this book to both touch and touchless gestural interaction.

FURTHER READING

Hinckley, K., Baudisch, P., Ramos, G., and Guimbretiere, F. Design and analysis of delimiters for selection-action pen gesture phrases in scriboli. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Portland, OR, April 2–7, 2005). CHI '05. ACM, New York, NY, pp. 451–460. DOI=http://doi.acm.org/10.1145/1054972.1055035. In this work, Hinckley and his colleagues examine various methods for telling the system "The stroke I just entered wasn't intended as ink, but rather as a command." They consider a variety of mechanisms, and conclude that the pigtail method is clearly superior.

Grossman, T., Hinckley, K., Baudisch, P., Agrawala, M., and Balakrishnan, R. Hover widgets: Using the tracking state to extend the capabilities of pen-operated devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Montréal, Québec, Canada, April 22–27, 2006). R. Grinter et al., eds. CHI '06. ACM, New York, NY, pp. 861–870. DOI=http://doi.acm.org/10.1145/1124772.1124898. In this work, Grossman and others at Microsoft Research build a set of simple hover-based gestures to complement those built-in to the tablet PC. The Hover Widget gestures differentiate themselves from other actions performed in the hover zone of the PC by reserving specific physical movements to invoke them. These are distinct from other types of in-air gestures in that they are meant to complement a pen input system, where touching the pen to the display performs other actions.