

# Interaction Elements

# 3

Interaction occurs when a human performs a task using computing technology of some sort. The “performs a task” aspect forms the interaction. The task is often goal-oriented, such as sending e-mail, burning a CD, programming a thermostat, or entering a destination in a global positioning system (GPS). But sometimes there is no specific goal. Just browsing the web or chatting with friends on a social networking site is sufficient to qualify as a task. If the user is engaged in any activity with computing technology, interaction is taking place.

In this chapter, elements of human-computer interaction are examined. The topics do not comprehensively span the field of HCI. Instead, the topics are selective and focus on low-level aspects of interaction. Most of the tasks are situated in the cognitive band in Newell’s *Time Scale of Human Action* presented in Chapter 2. The human systems at work are deliberate acts ( $\approx 100$  ms), operations ( $\approx 1$  s), and unit tasks ( $\approx 10$  s). With durations ranging from 100 ms to 10 s, the tasks are well suited to empirical research. In fact, most experimental research in HCI is directed at tasks of this sort. Due to the brevity of the tasks, extraneous behaviors are easy to control. Thus, experiments with high *internal validity* are possible. (Internal validity is discussed in the next chapter.) Although it is possible to experimentally study higher-level tasks—those in the rational social bands—difficulties arise due to the duration of the tasks. Operations in these bands take from minutes to months (see Figure 2.1), and therefore often include extraneous behaviors such as pausing, task switching, pondering, decision-making, and secondary and unrelated tasks. Although such behaviors are germane to the HCI experience, they present challenges for experimental study; therefore, we focus on interaction elements in the cognitive band.

At a stripped-down level, interaction involves humans using their sensors and responders to monitor and control devices, machines, and systems that include computing technology. We saw a typical human factors view of this in the Chapter 2 (see Figure 2.2). The internal states of both the human and the machine interact in a closed-loop system through controls and displays (the machine side) and motor-sensory behaviors (the human side). By convention, the terms *input* and *output* are with respect to the machine; so inputs, or input devices, are inputs to the machine that are controlled or manipulated by human outputs. Most commonly, human

outputs are limbs—the fingers, hands, arms, legs, and feet—but speech, articulated sounds, eye motions, blinks, breath, electrical body signals, and so on can also serve as human outputs. Some of these communication channels are extremely important for disabled users.

There is a long history of human factors research following the classical view that focuses on “knobs and dials” (Chapanis, 1965; Van Cott and Kinkade, 1972). Using responders, the human operator manipulates controls or knobs, perhaps with a goal in mind, while engaging senses to perceive the status of a system on displays or dials.<sup>1</sup> This work is important in HCI, too. Relevant examples are numerous, particularly from tasks when things don’t quite go as planned: terminating an application on a computer while intending only to minimize a window, putting food in a microwave oven and heating it for three seconds instead of three minutes, or terminating a call on a mobile phone while intending only to adjust the volume. These are all knobs and dials issues, and they perplex and frustrate us, even today.

We begin with the basic properties of human input to computers (the knobs or controls) and the responses produced (the dials or displays). The conventional desktop computer system is a convenient input/output system in which to frame these discussions. Although the issues have been highly researched over the years, the interactions in desktop computing are relevant in newer domains, such as mobile computing, virtual environments, surface computing, tangible computing, wearable computing, gaming, and so on.

---

### 3.1 Hard controls and soft controls

Before computers infiltrated every aspect of our homes and workplaces, displays and controls tended to be physical single-purpose devices. Controls like joysticks, switches, arrays of switches, push buttons, keys, keyboards, mice, steering wheels, handles, knobs, levers, and so on arrived through an involved process of design, engineering, and manufacturing. Once built, they were fixed for life. This meant their behaviors were constrained to a small set of relationships with the displays they accompanied. Even an aircraft’s cockpit had a relatively small number of controls and displays. Because they were physical and single-purpose, there was a limited set of relationships that existed between them.

The advent of computers with software, graphical displays, and point-and-click interactions changed this. Through *soft interfaces*, the way humans interact with technology changed, and changed everywhere. The malleability of a display, through software, brings unlimited possibilities to a relatively small physical space. The result is *soft controls*. A soft control can be born out of little more than a programmer’s whim—a few lines of computer code and a quick round or two of code-compile-test—all done between the morning coffee break and lunch. Before you

---

<sup>1</sup>*Displays* as used here is any form of computer output. This includes visual as well as auditory and tactile displays.

know it, there is a new button on a toolbar, a new “Options . . .” popup dialog, or a new drawing mode invoked with CTRL-*something*. The possibilities are limitless.<sup>2</sup>

Human-computer interfaces contain lots of controls. Most are soft controls manipulated by physical controls (the mouse or keyboard). Desktop computer applications, for example, have thousands of control-display arrangements. Every conceivable action one might make with a soft control that produces a displayed response on the system is an example. Controls invoke different responses (displays) depending on the context, so the combinations multiply rapidly.

The distinction between controls and displays is blurred with soft controls. Since a soft control is rendered on a display through software, its size, shape, or appearance can change dynamically to convey information to the user. Thus, soft controls often have properties of displays. Toolbar buttons and widgets are typical examples. Figure 3.1 shows an example from the word processor used to create this chapter. The image was captured with the cursor, or I-beam, positioned inside *this*. The image contains three combo boxes and seven buttons. Each is both a control and a display. The combo boxes reveal that the style is Body Text, the font is Times, and the font size is 12 point. The buttons are toggle buttons, revealing the state of the text at the insertion point. The text is italicized, underlined, left justified.

A scrollbar slider, or elevator, is another example<sup>3</sup> (See Figure 3.2.). The slider is both a control and a display. As a control, it is moved to change the view in the document. As a display, its size reveals the magnitude of the view relative to the entire document and its position reveals the position of the view within the document. Most soft controls are also displays.

Soft controls need little space. Figure 3.3 shows a small area of a GUI application. It is only 30 square centimeters, but look: it contains a bewildering array of soft controls—more than 20. Or are they displays? Each is a world unto itself. Each identifies or suggests its purpose by its shape, its position, a label, or an icon. Each of these soft controls is manipulated by a hard control, by clicking. Or is double-clicking? Or is it SHIFT-clicking? Or is it dragging? What about right-clicking? What about keyboard access? And there are presumptions that somehow it all makes sense to the user. Each control is also precariously close to a neighbor. Many users of this application, Microsoft Word, will invest thousands of hours working this

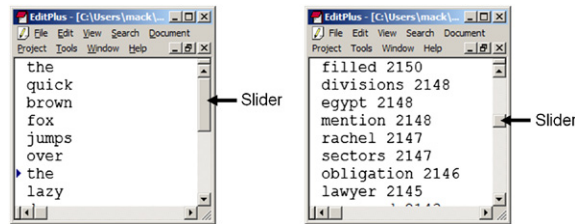


FIGURE 3.1

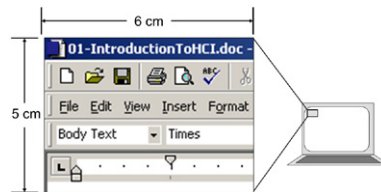
Many soft controls are also displays.

<sup>2</sup>Of the programmer's whimsical role in creating overly complex and unusable interfaces, Cooper is even less kind (Cooper, 1999, pp. 1–246). Others also note the undue influence of technical staff. In one case, management's inability to challenge the judgment of engineering staff is cited as contributing to the failure of an entire project (D. K. Smith and Alexander, 1988, p. 160).

<sup>3</sup>The scrollbar slider has a variety of names, depending on the platform, including thumb, box, knob, scroller, wiper, and grip.

**FIGURE 3.2**

A scrollbar slider is both a control and a display.

**FIGURE 3.3**

A small physical space on a graphical display. There are more than 20 soft controls and displays in a 30-square-centimeters area. A single user action can initiate a complete redefinition of the space.

30-square-centimeters interface. The users are, by all accounts, experts. Yet thousands of times users, even expert users, commit errors working in this same space. Sometimes they act in haste. At other times they're just not sure what to do. It's a complex space, this 30 square centimeters. And all it takes is one user action—a key press or button click—and the entire 30 square centimeters is morphed into a completely different interface. See student exercise 3-1 at the end of this chapter.

The hard-control/soft-control distinction is similar to the idea of *space multiplexing* versus *time multiplexing* (Buxton, 1983). With space multiplexing, there is a dedicated physical control for each parameter that is controlled. Each control has a separate physical location. With time multiplexing, there are fewer controls than parameters, and so a single device (e.g., a small region of a display) is reconfigured to control different parameters at different stages of an operation.

In the next section, we examine controls and displays in terms of their spatial relationships. We begin with hard controls such as a mouse, and typical displays such as are found on desktop computer systems.

### 3.2 Control-display relationships

When a user grasps a computer mouse and moves it to the right, the cursor on the system's display moves to the right. This *control-display relationship* is not something users think much about. This is human-computer interaction as it should

be—where the relationship between what a user does and what is experienced is natural, seamless, intuitive, and efficient. Control-display relationships are sometimes called *mappings*, since the relationships attribute how a controller property maps to a display property.

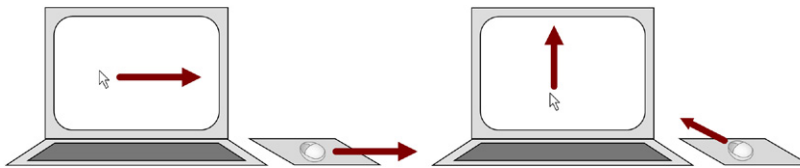
The mouse/cursor example describes a *spatial relationship*. There are also *dynamic relationships*, describing how a controller affects the speed of the response, and *physical relationships*, describing whether the response is to a movement or a force in the controller. We will examine each of these in this section, beginning with spatial relationships.

### 3.2.1 Spatial relationships

The mouse-cursor spatial relationship described above is illustrated in Figure 3.4a. The mapping is *congruent* because there is an exact spatial correspondence between the controller input and the display output. Move the mouse right, and the cursor moves right. The situation in Figure 3.4b is different. Move the mouse “forward” and the cursor moves “up.” As with side-to-side movement, users likely don’t think much about this. Yet the first time someone encounters a mouse, there is likely some uncertainty on how it works. What does it do? What is it controlling? Where is the effect seen? No doubt, the spatial relationship between mouse and cursor is learned very quickly.<sup>4</sup> I will say more about learned versus natural relationships later.

It is useful at this juncture to introduce labels to further examine the spatial mappings between input controls and output displays. Figure 3.5a and Figure 3.5b show the display and mouse pad surfaces in a Cartesian coordinate space with  $x$ -,  $y$ -, and  $z$ -axes.<sup>5</sup> Left-right motion is along the  $x$ -axis. Up-down motion is along the  $y$ -axis. Forward-backward motion is along the  $z$ -axis. Of course, a cursor cannot move along the  $z$ -axis, nor is mouse motion sensed on the  $y$ -axis. The arrows show positive motion along each axis.

The mouse-to-cursor mapping is shown in Figure 3.5c. Here we see the slight disjoint noted above. Mouse  $x$ -axis motion maps to cursor  $x$ -axis motion, but mouse

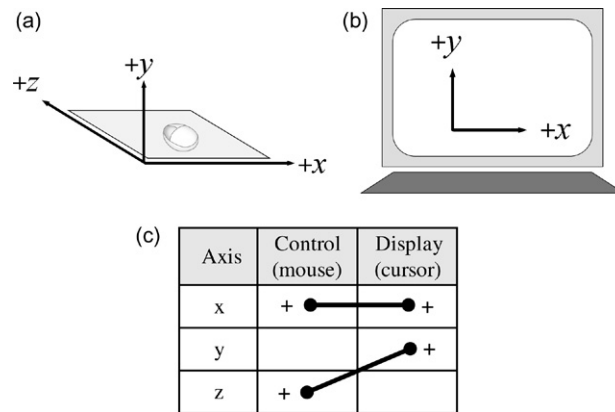


**FIGURE 3.4**

Control-display relationships for a mouse and cursor.

<sup>4</sup>A student once told me of his experience teaching his mother how to use a computer. At one point he told his mother to move the cursor “up” to the menu bar. She raised the mouse “up” off the mousepad.

<sup>5</sup>The choice of labels is, of course, arbitrary.

**FIGURE 3.5**

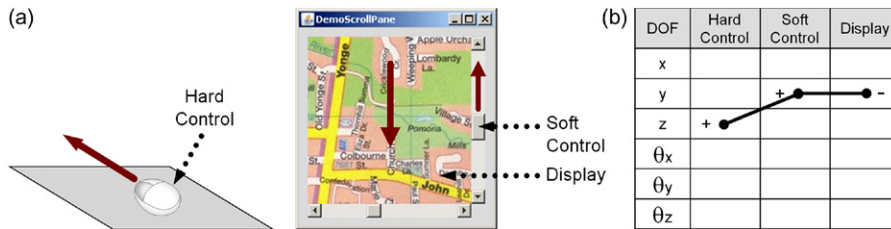
Axis labels. (a) Control space. (b) Display space. (c) Control-display mapping for a mouse and cursor.

$z$ -axis motion maps to cursor  $y$ -axis motion. The plus (+) symbols indicate that positive motion of the control yields positive motion in the display along each axis.

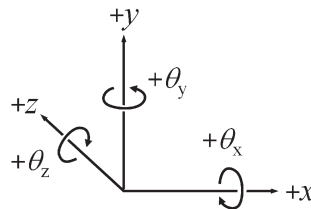
The  $y$ -axis cursor motion is an example of a *transformed spatial mapping*. In this case the transformation is  $90^\circ$ , if viewed along the  $y$ - $z$  plane. We might expect an effect on user performance in the presence of a spatial transformation, and that is the case. Aiming error is known to be higher for  $90^\circ$  to  $135^\circ$  transformations and lower for  $0^\circ$  or  $180^\circ$  transformations (Cunningham, 1989). It is also known that adaptation occurs for all transformations, in a few as 50 trials (Wigdor, Shen, Forlines, and Balakrishnan, 2006). So while initial exposure to a mouse might pose a problem (and yield inaccurate pointing), with practice the relationship is learned and accuracy improves.

Describing mouse-cursor mappings in a three-dimensional (3D) space, as above, invites a deeper analysis of other control-display relationships. This is justified at the very least because HCI is more than desktop interaction. Whether using a thumb-controlled wheel to scroll through a calendar on a personal digital assistant (PDA) or using a handheld controller to annihilate the enemy in a first-person shooter game, the relationship between what we do and what we get is of the utmost importance in designing interfaces and interaction techniques that are efficient and transparent.

Now consider scrolling the view of an image or document. A scroll pane includes scrollbars with sliders to move the view. Interaction involves manipulating a physical controller, such as a mouse (a hard control), to move a pointer to the slider (a soft control) and acquiring it with a button-down action. The slider is then dragged up or down to produce a change in the view. This scenario adds a third tier to the control-display mappings. Figure 3.6a shows a mouse and an image of a map displayed in a GUI window. Scrollbars with sliders appear to the right and

**FIGURE 3.6**

A three-tier control-display relationship: (a) Moving the hard control forward moves the soft control up, which in turn moves the display view down. (b) Control-display mappings.

**FIGURE 3.7**

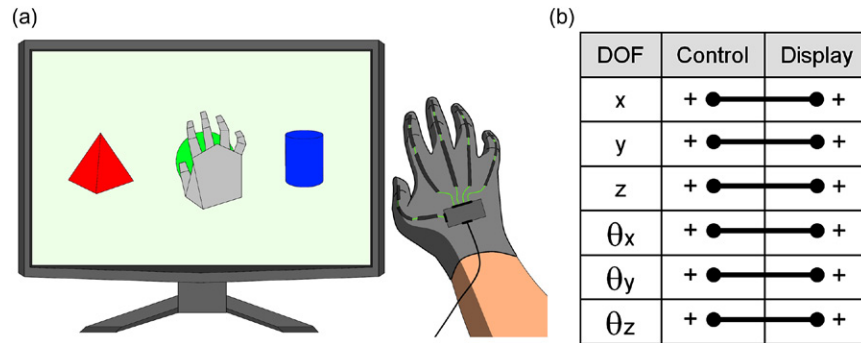
Axis labels for a three-dimensional space.

on the bottom of the window. There is a spatial transformation between the hard control and soft control and between the soft control and the view on the display. (See Figure 3.6b.)

Most computer users are adept at scrolling through documents and have no trouble with the spatial transformations illustrated in Figure 3.6. The  $+/-$  reversal in the control-display relationship along the  $z$ -axis is interesting, however. What if the relationship between the soft control (scrollbar slider) and the display (map image) were reversed? Would this be okay? Yes, but the relationship would have to be learned.

The next step in examining spatial relationships is to expand the 3D space to include rotation about each axis. This is shown in Figure 3.7 using *theta* ( $\theta$ ) to designate angle or rotation. The arrows indicate positive rotation, corresponding to clockwise movement as viewed looking out from the origin along each axis. In all, there are six parameters, or *degrees of freedom* (DOF). Degrees of freedom means that each parameter may be manipulated independently of the others. So to fully place an object in 3D space, six parameters are required: three for the object's *position* in space ( $x$ ,  $y$ ,  $z$ ), and three for the object's *orientation* in space ( $\theta_x$ ,  $\theta_y$ ,  $\theta_z$ ). In aeronautics, the rotation terms are known as pitch ( $\theta_x$ ), roll ( $\theta_z$ ), and yaw ( $\theta_y$ ).

Figure 3.8a shows an example of 3D interaction where congruence is fully attained (see also Zimmerman, Lanier, Blanchard, Bryson, and Harvill, 1987). Of course the depth, or  $z$ -axis, is simulated on the display. Nevertheless, the figure illustrates the goal of facile 6 DOF interaction. The figure shows a 6 DOF input glove (a controller) in exact spatial correspondence with a simulated glove on the

**FIGURE 3.8**

Spatial congruence in 3D: (a) 6 DOF input controller and 6 DOF display (sketch courtesy of Bartosz Bajer). (b) 6 DOF control-display correspondence.

display. The fluidity of the interaction is clearly evident: The mappings are congruent for all 6 DOF, as illustrated in Figure 3.8b.

Most interactive systems support a subset of the 6 DOF available in 3D. And in most cases, there are spatial transformations that encumber the user with small challenges to learn and overcome, like the mouse-cursor interaction noted earlier. Rotation is a challenge since traditional controllers, such as a mouse, only sense translation along the  $x$ -axis and  $z$ -axis. Control-display spatial congruence is not possible, so creative mappings are needed to facilitate interaction. The typical solution is modes, which we examine in more detail later. For the moment, consider Street View in Google Maps, which provides panoramic views of streets in major cities around the world.<sup>6</sup> A street scene showing the city hall in Toronto is seen in Figure 3.9a along with facsimiles of the controls for viewpoint control. The most common manipulations are *panning*, *zooming*, and *positioning*. Panning involves rotating the camera view left-right and up-down. Zooming involves changing the magnification of the view. Positioning is a change in the camera position. In the case of Google Street View, the camera position is constrained to points along the street from which the images were captured. So, while it is possible to zoom in to the city hall, it is not possible to walk toward it.

Scene manipulations are supported through a variety of interactions using the keyboard or mouse. One way to pan with the mouse is to drag the image—position the pointer anywhere on the image, press and hold the primary mouse button, then move the pointer. Left-right, or  $x$ -axis, linear movement of the mouse effects rotation of the scene about the  $y$ -axis. Forward-backward, or  $z$ -axis, linear movement of the mouse rotates the scene about the  $x$ -axis. These mappings are illustrated in Figure 3.9b. Although the mappings look convoluted, the interaction is reasonably simple (i.e., quickly learned).

<sup>6</sup><http://maps.google.com/streetview>.



**FIGURE 3.9**

Rotating a 3D scene: (a) Street scene in Toronto with pan and zoom controls. (b) Mapping of mouse control for panning if the scene is dragged.

Another way to pan with the mouse is through the soft control labeled Pan in the figure. Clicking an arrow effects a discrete pan in the expected direction. Left-right continuous panning is possible by dragging the circle that surrounds the arrows. This interaction is tricky and of questionable merit. It requires a tight coupling of left-right, up-down linear movement of the mouse *and* it requires the user's visual attention—on the soft control! Of course, the user would prefer to look at the scene.

Zooming is performed by clicking on + and – soft controls seen in the figure. The effect is to zoom the view in and out along the  $z$ -axis. It is interesting to consider zooming via  $z$ -axis movement of the mouse, since the result is a spatially congruent control-display mapping.<sup>7</sup> Unfortunately,  $z$ -axis mouse movement is committed to  $x$ -axis scene rotation (i.e., up-down panning; see Figure 3.9b). Perhaps a modifier key, such as SHIFT, could be used to map mouse  $z$ -axis movement to  $z$ -axis zooming.

Another congruent mapping to consider is using  $y$ -axis rotation of the mouse to control  $y$ -axis rotation of the camera to create left-right panning.  $Y$ -axis rotation amounts to rotating the mouse on the mouse pad or desktop surface. Of course, such movement is not sensed. But with some careful reengineering it can be (as discussed later in this chapter).

Using  $y$ -axis rotation is a potentially interesting and new interaction technique. It suggests an interaction worth investigating through empirical research. Can user interaction with Google Street View be improved using other input controllers or other mappings? If so, how is a conjectured improvement evaluated? What tasks and user performance measurements are relevant? The spatial mappings between input controls and output displays are numerous in HCI, and the relationships are often complex. These complexities invite thinking about the problem space, searching out new and potentially facile interactions, and then testing and comparing them through experimental research. See also student exercise 3-2 at the end of this chapter.

<sup>7</sup>Zooming is possible using the mouse wheel, but the detents in the wheel motion make the interaction feel jerky.

### 3.2.2 CD gain and transfer function

There is more to mouse-cursor movement than spatial mappings. Another property is *CD gain*, which represents the amount of movement in a display object, such as a cursor, for a given amount of movement in a control.<sup>8</sup> For example, if a mouse is moved three cm and the cursor also moves three cm, then the CD gain is 1. If the cursor moves six cm, the CD gain is  $6/3 = 2$ . CD gain for a mouse and cursor is typically set in a control panel, as seen in [Figure 3.10](#). Moving the slider to the “slow” end of the scale reduces CD gain, meaning more controller movement is required for each unit of cursor movement.

Often the relationship between controller motion and cursor motion is non-linear and uses a *power function*. In this case, the amount of cursor motion depends on the velocity of mouse motion as well as controller motion. This is enabled in [Figure 3.10](#) by selecting “Enhance pointer precision.” If the mouse moves quickly, CD gain increases. If the mouse moves slowly, CD gain decreases. Lowering the CD gain for slow controller movements is useful to enhance the precision of target selection at the end of a point-select operation. The term *transfer function* is sometimes used since non-linear relationships are more elaborate. To ensure the cursor is responsive to mouse movement, with no perceivable delay, the software implementation of a non-linear relationship typically uses a lookup table to map controller movement to display movement. See also student exercise 3-3 at the end of this chapter.

Research on optimizing CD gain dates back to at least the 1940s, when the first electronic joysticks were developed (Jenkins and Connor, 1949). The problem of optimizing is trickier than it seems, however. Varying CD gain evokes a trade-off between gross positioning time (getting to the vicinity of a target) and fine positioning time (final acquisition). (See [Figure 3.11](#).) In a simplistic sense, the optimal setting is that which minimizes the combined gross and fine positioning times. However, this may be confounded with a potentially non-optimal error rate at the CD gain setting yielding a minimal overall positioning time. Other factors such as display size or scale (independent of the CD gain setting) also bring into question the optimization of this common input device parameter (Arnault and Greenstein, 1990; Blanch, Guiard, and Beaudouin-Lafon, 2004).

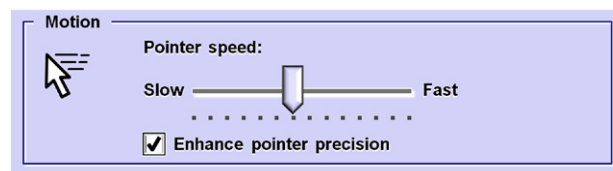
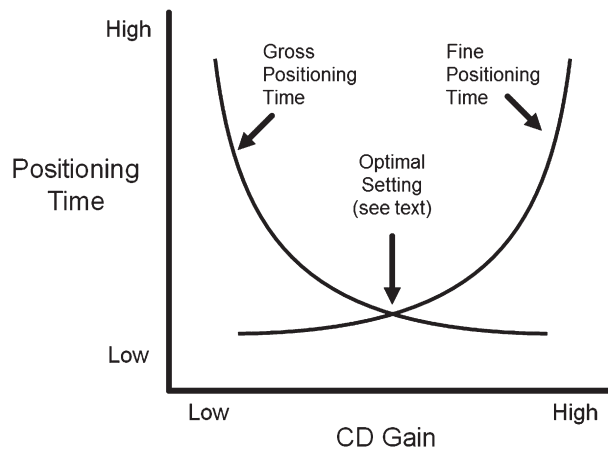


FIGURE 3.10

Controlling the CD gain (pointer speed) from the mouse control panel.

<sup>8</sup>CD gain is sometimes called C:D ratio. A CD gain of 2 is equivalent to a C:D ratio of 1:2; that is, one unit of control (mouse) movement yielding two units of display (cursor) movement.

**FIGURE 3.11**

Trade-off between gross positioning time and fine positioning time as a function of CD gain.

Over the past 20 to 30 years, CD gain has been highly researched as it applies to interaction with GUIs on desktop computing systems (e.g., Barrett, Selker, Rutledge, and Olyha, 1995; Buck, 1980; Jellinek and Card, 1990; B. H. Kantowitz and G. C. Elvers, 1988). The majority of studies found that significant performance benefits are not achieved by adjusting CD gain. One challenge in optimizing CD gain is in defining optimal performance. Is the goal to maximize speed (viz. minimize positioning time) or to maximize accuracy? The goal of optimizing *both* speed and accuracy is problematic, because of the speed-accuracy trade-off. What improves speed tends to worsen accuracy, and vice versa. A useful metric that combines speed and accuracy is Fitts' throughput, but the calculation of throughput is subject to considerable variation in the literature, and this exacerbates assimilating research results into a comprehensive understanding of CD gain and human performance.

Optimizing CD gain is of continuing interest today, although the focus is often in different settings, such very large displays (Kobayashi and Igarashi, 2008; McCallum and Irani, 2009), very small displays (D. Harrison and Hudson, 2009), accessible computing (Wobbrock, Fogarty, Shih-Yen, Kimuro, and Harada, 2009), remote pointing (Konig, Gerken, Dierdorf, and Reiterer, 2009), and 3D interaction (Argelaguet and Andujar, 2008). In each of the examples just cited, the research brought a new idea to the problem of tuning the relationship between an input control and interaction on an output display. Since human performance is the main criterion for optimization, research on CD gain is well suited to empirical enquiry and experimentation. See also student exercise 3-4 at the end of this chapter.

### 3.2.3 Latency

Human interaction with computers is a two-way structure. As participants in a closed-loop system, we issue commands through input controllers and receive

responses on output displays. Subsequent input depends on the latest output. Not surprisingly, human performance and the interaction experience are adversely affected when feedback is delayed. The delay between an input action and the corresponding response on a display is called *latency* or *lag*. Fortunately, latency is negligible in many interactive computing tasks, such as typing or cursor positioning.

The opposite extreme is remote manipulation, where the human operator is physically displaced from the machine under control. The latency may be due to mechanical linkages or to transmission delays in the communications channel. For example, in controlling a space vehicle on a distant moon or planet (as viewed on a display), transmission delays—latency—are on the order of several minutes or longer.

Latency is often present with Internet connections: click on a link and wait a few seconds or longer for a page to appear. The research issues here are more about user experience and frustration than about human performance. Frustration is reduced, for example, if the interface includes feedback informing the user that the network is processing the request (Bouch, Kuchinsky, and Bhatti, 2000).

Virtual reality relies heavily on the tracking of hand, head, or body motion in a simulated 3D environment. The pretense of reality requires a tight link between the user's view of the environment and the actions—usually hand, head, or body motions—that set the view. When changes in the environment lag behind input motions, the loss of fidelity is dramatic.

Latency is attributable to properties of input devices, output devices, and software. Input devices are usually sampled at fixed rates in the range of 10 to 60 samples per second. At 60Hz sampling, for example, an input movement may not be sensed for  $1/60\text{s} = 0.01667\text{s} = 16.67\text{ms}$ . Latency increases further due to software overhead—a loose expression for a variety of system-related factors. Communication modes, network configurations, number crunching, and application software all contribute to it. Latency will increase dramatically when output requires substantial computation for graphics rendering. A display frame rate of 10Hz is considered minimal to achieve real-time animation. Since the construction of the frame begins only once the position is known, the potential for parallel processing is limited. Using standard double buffering (where the frame is presented only once fully drawn), there is 100ms minimum latency to the start of the frame display interval and a latency of 150ms to the middle of the frame display interval.

From the points above, we can infer that it is not possible to accurately measure latency from within the target system. Using software to log timestamps for input and output operations within a system misses the actions and responses at the human interface. To reflect the user's experience as a whole, a proper latency measurement is done externally (Roberts, Duckworth, Moore, Wolff, and O'Hare, 2009; Steed, 2008). Teather et al. (2009) describe a pendulum apparatus for measuring latency. The input controller swings in front of the output display with a high-speed camera recording the temporal relationship between input motion and the output response.

Latency in virtual reality is associated mostly with 6 DOF tracking devices attached to the hand, head, or body as well as the latency caused by the low frame

---

<sup>9</sup>[www.polhemus.com](http://www.polhemus.com).

**FIGURE 3.12**

Polhemus G<sup>4</sup><sup>TM</sup> 6 DOF tracking system. The 6 DOF sensor is identified by the arrow.

*(Photo courtesy of Polhemus)*

rates. Typical tracking devices include the Polhemus G<sup>4</sup><sup>TM</sup> (Figure 3.12)<sup>9</sup> and the Ascension *MotionStar*.<sup>10</sup> These devices and their interfaces transmit six degree-of-freedom position and orientation data to the host computer at a high rate while acting in concert with competing processes in the complete virtual reality (VR) environment. Maintaining negligible latency is usually not possible. The Polhemus G<sup>4</sup><sup>TM</sup>, for example, is sampled at 120Hz, meaning one sample every  $1/120\text{s} = 0.00833\text{s} = 8.33\text{ms}$ . According to the device specification, latency is less than 10ms, meaning the three position and three orientation coordinates are available no later than 10ms after each sample. Of course, this only applies to the input sub-system. From an HCI perspective, this value is irrelevant. The user experiences the system as a whole.

Although latency is known as a compromising parameter in VR systems (Liang, Shaw, and Green, 1991), and its effect on human performance is considerable, the evidence for this is mostly anecdotal. On intuitive grounds, Pausch noted that low-latency is significantly more important than high-quality graphics or stereoscope (Pausch, 1991). His low-end VR system emphasized presence (i.e., low latency) over output graphics by maintaining seven screen updates per second with wire-frame images. In an experiment with force feedback, Brooks et al. (1990) concluded that screen updates are needed 15 times per second minimum to maintain the illusion of continuous change in the force sensed. Even at this rate, subjects noted the “sponginess of the feel”. Others cite motion sickness as a by-product of lag with head-mounted displays (Draper, Viire, Furness, and Gawron, 2001; Foley, van Dam, Feiner, and Hughes, 1987; Laurel, 1991; Roberts et al., 2009).

Empirical measurements of the effect of latency on human performance are rare. There was some research in the 1960s (e.g., Gibbs, 1962, Sheridan, and Ferrell, 1968) on the design of controllers for remote manipulation systems; however, the latency tested was extremely long (up to several seconds). In one instance the apparatus was mechanical (Sheridan and Ferrell, 1968) and in another latency

<sup>10</sup>[www.ascension-tech.com](http://www.ascension-tech.com).

was programmed as an exponential time constant preventing the cursor from fully reaching a target unless a deliberate overshoot was programmed (Gibbs, 1962). In both cases movement time increases due to latency were dramatic—well in excess of the amount of latency. Since latency is on the order of a few hundred milliseconds in VR, its effect on human performance is less apparent.

MacKenzie and Ware (1993) systematically introduced latency in a system and measured the effect on human performance in simple point-select tasks. With 75 ms latency, movement time increased by 16 percent and error rate by 36 percent. At 225 ms the effect was dramatic, with movement time increasing by 64 percent and error rate by 214 percent.

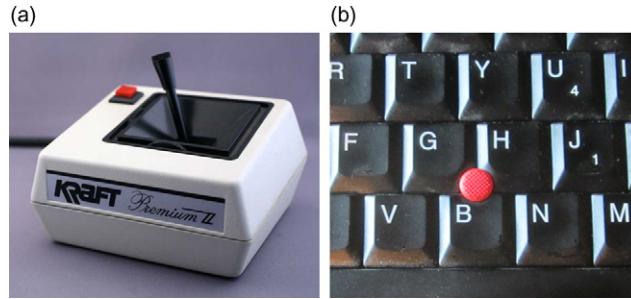
### 3.2.4 Property sensed and order of control

When a human engages an input controller, the dialog involves actions such as touching, tapping, grasping, moving, pushing, flicking, and squeezing. The input controller senses the interaction and converts a *property sensed* into data that are transmitted to the host computer for processing. For pointing devices, the most common properties sensed are *position*, *displacement*, and *force*. In this section, we examine both the property sensed and the way the property is used to control an object or view on an output display. Does the property sensed control the position or the velocity of the object or view? This question speaks to the *order of control*, which I present shortly.

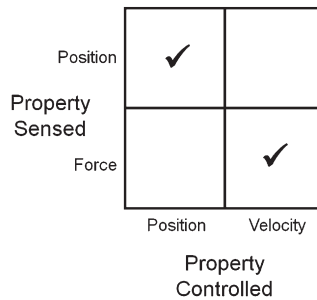
With a graphics tablet, touchpad, or touchscreen, the property sensed is the position of a stylus or a finger on a digitizing surface. Position is sensed as an *absolute* coordinate at the point of contact along the *x*-axis and *z*-axis. (Discussions here assume the coordinate system in Figure 3.7.) A mouse is different. It is not possible to know where a mouse is situated on a mousepad or desktop. With a mouse, the property sensed is the *displacement*—the amount of movement along the *x*-axis and *z*-axis. Each sample is reported *relative* to the last sample. Touchpads in notebook computers typically operate in mouse-emulation mode. Even though the property sensed is the absolute position of the finger on the touchpad surface, the value is reported to the host in relative terms—the amount of finger displacement relative to the last sample. Graphics tablets typically support either absolute or relative modes of reporting.

The most common orders of control are position-control (aka zero-order control) and velocity-control (aka first-order control) (Zhai, 1995, p. 4). With position-control, the sensed property of the input device controls the position of the object or view on a display. With velocity-control, the sense property controls the velocity of the object or view. A mouse is a position-control input device, since mouse displacement controls the position of the cursor.

Order of control is often associated with joysticks. Joysticks are either *isotonic* or *isometric*. With an isotonic joystick, the user manipulates the stick handle, which, in turn, swivels about a pivot-point. The property sensed is the movement of the stick. Isotonic joysticks are also called *displacement joysticks*. The output signal represents the position of the stick as the amount of displacement (i.e., rotation) from the neutral or home position, about the *x*- and *z*-axes. An example is the

**FIGURE 3.13**

Joysticks: (a) Isotonic (displacement sensing).<sup>11</sup> (b) Isometric (force sensing).

**FIGURE 3.14**

Order of control mapping for property sensed versus property controlled. Checks indicate the preferred mappings.

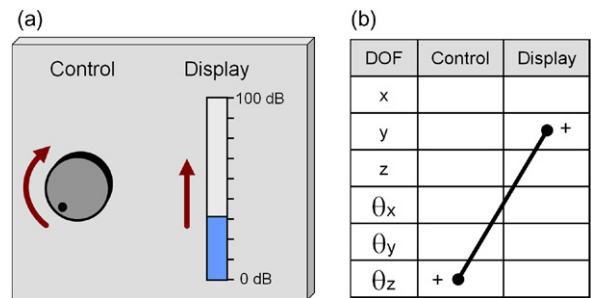
*Premium II* from Kraft Systems, a common joystick for computer games in the 1980s. (See [Figure 3.13a](#).)

With an isometric joystick, the stick does not move. The property sensed is the force applied to the stick. The output signal represents the amount of force along the  $x$ -axis and  $z$ -axis. An example is the joystick between the G, H, and B keys on many notebook computer keyboards. (See [Figure 3.13b](#).)

The order-of-control mappings for isotonic (position sensing) and isometric (force sensing) joysticks are illustrated in [Figure 3.14](#). These possibilities raise the question of performance: Which mappings are best in terms of speed and accuracy for common point-select tasks? Kantowitz and Elvers (1988) evaluated an isometric joystick in both position-control and velocity-control modes. Velocity-control performed best. A full investigation of the possibilities in [Figure 3.14](#) requires evaluation with two independent variables: property sensed and property controlled. In other words, both position-sensing and force-sensing input controllers must be tested in both position-control and velocity-control modes. Zhai conducted such an

<sup>11</sup>From the Buxton Collection (<http://research.microsoft.com>).





**FIGURE 3.15**  
A control-display relationship. Learned or natural?

experiment and observed that position control is best for a position-sensing device and that velocity control is best for a force-sensing device (1995, 35). These mappings are indicated with check marks in the figure.

### 3.3 Natural versus learned relationships

Comments were offered above on the need for users to learn a control-display relationship. (The need arises when there is a spatial transformation between the control motion and the display response.) It is worth examining, then, whether some relationships are more natural than others. Consider the example in [Figure 3.15a](#). Rotating the control produces linear movement in a reading on a display. The arrows show clockwise control motion producing upward display motion. As seen in [Figure 3.15b](#), there is a spatial transformation between control and display. Is the relationship natural? Is it simple, intuitive, and easy? Both arrows in the figure point *up*, so perhaps the relationship is natural. However, if the curved arrow were positioned on the right of the control, rather than on the left, it would point down. So the relationship would then appear backward. The real question, then, is not about the arrows, but whether there is a natural relationship between clockwise motion of a control and linear motion on a display. The answer is *no*. It is a learned relationship. It might seem simple, intuitive, and easy, but it is a learned relationship.

One might argue that turning a rotary dial “clockwise” should produce an “increase” in a parameter. But this is true only if it supports the user’s *expectation*. If the relationship is expected and considered correct by a majority of people within a population, such as a cultural, ethnic, or geographic group, then the relationship holds; it is accepted and considered natural by that population. In this case, it is more accurately a *population stereotype* (B. H. Kantowitz and Sorkin, 1983, p. 325) or *cultural standard* (Norman, 1988, p. 23). The relationship is considered natural for a population, but still, it is a learned relationship.

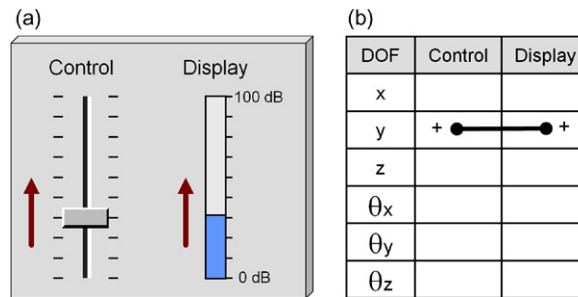
Referring again to [Figure 3.15](#), would placing the rotary control on the opposite side of the display make a difference? According to “Warrick’s principle,” the display



indicator should move in the same direction as the nearest point on the control (B. H. Kantowitz and Sorkin, 1983, p. 330). In this case, the mapping in Figure 3.15 is, in fact, wrong! Warrick's principle is an example of applied ergonomics, with little basis in human sensory-perceptual cognition. Wherever there is a spatial transformation, the ensuing relationship—no matter how easy we accept it or adapt to it—is learned, not natural. In the case of Warrick's principle, we might further inquire whether the principle applies equally to left-handed and right-handed users. Regardless, the best possible scenario is spatial congruence, which for a linear display implies using a sliding or linear control. This is shown in Figure 3.16. Upward movement in the control handle produces upward movement in the display reading.

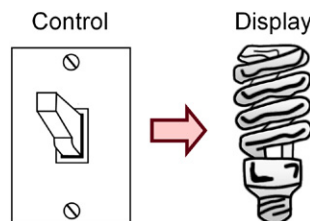
Here's another example. Consider the control (switch) and display (light bulb) in Figure 3.17. There is no spatial meaning to the state of the light bulb (it is either on or off), yet the question remains: is there a natural relationship between the control and the display? As the switch is shown, is the light on or off? Your answer is influenced by where you live. If you live in the United Kingdom, the light is off. If you live in the United States or Canada, the light is on.

A tenable physical analogy is difficult to form in some situations. There is no physical analogy between the position of a switch (up, down) and the state of a light (on, off). Given a consistent recurring implementation within a geographical region, a population stereotype will emerge, even in the absence a physical analogy. And that's what happens with the switch-light interaction. People in different



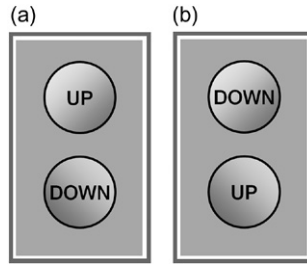
**FIGURE 3.16**

Spatial congruence: (a) Control and display. (b) Mapping.



**FIGURE 3.17**

Culture plays a role. Is the display on or off?

**FIGURE 3.18**

Button arrangements for an elevator control panel. (a) Correct. (b) Incorrect.

geographic regions have experienced and learned it differently. What is accepted in one region may differ from what is accepted in another.

If there is a physical contradiction, then the situation is different. Consider elevators (in buildings, not scrollbars). Early elevators didn't have buttons to specify floors; they only had UP and DOWN buttons. Consider the two arrangements for the button controls shown in Figure 3.18. Clearly the arrangement in (a) is superior. When the UP control is pressed, the display (the elevator) moves UP. The stimulus (control) and response (display) are compatible beyond doubt. In (b) the position of the controls is reversed. Clearly, there is an incompatibility between the stimulus and the response. This situation is different from the scroll pane example given in Figure 3.6 because there is no physical analogy to help the user (can you think of one?). If all elevator control panels had the arrangement in Figure 3.18b, would a population stereotype emerge, as with the light switch example? Well, sort of. People would learn the relationship, because they must. But they would make more errors than if the relationship was based on a correct physical mapping. This particular point has been the subject of considerable experimental testing, dating back to the 1950s (Fitts and Seeger, 1953). See also Newell (1990, 276–278) and Kantowitz and Sorkin (1983, 323–331). The gist of this work is that people take longer and commit more errors if there is a physical misalignment between displays and controls, or between controls and the responses they effect.

This work is important to HCI at the very least to highlight the challenges in designing human-computer systems. The physical analogies that human factors engineers seek out and exploit in designing better systems are few and far between in human-computer interfaces. Sure, there are physical relationships like “mouse right, cursor right,” but considering the diversity of people's interactions with computers, the tasks with physical analogies are the exception. For example, what is the physical analogy for “file save”? Human-computer interfaces require a different way of thinking. Users need help—a lot of help. The use of metaphor is often helpful.

### 3.4 Mental models and metaphor

There is more to learning or adapting than simply experiencing. One of the most common ways to learn and adapt is through *physical analogy* (Norman, 1988, p. 23)

or *metaphor* (Carroll and Thomas, 1982). Once we latch on to a physical understanding of an interaction based on experience, it all makes sense. We've experienced it, we know it, it seems natural. With a scroll pane, moving the slider up moves the *view* up. If the relationship were reversed, moving the slider up would move the *content* up. We could easily develop a physical sense of slider up → view up or slider up → content up. The up-up in each expression demonstrates the importance of finding a spatially congruent physical understanding. These two analogies require opposite control-display relationships, but either is fine and we could work with one just as easily as with the other, provided implementations were consistent across applications and platforms.

Physical analogies and metaphors are examples of the more general concept of *mental models*, also known as *conceptual models*. Mental models are common in HCI. The idea is simple enough: "What is the user's mental model of . . . ?" An association with human experience is required. HCI's first mental model was perhaps that of the office or desktop. The desktop metaphor helped users understand the graphical user interface. Today it is hard to imagine the pre-GUI era, but in the late 1970s and early 1980s, the GUI was strange. It required a new way of thinking. Designers exploited the metaphor of the office or desktop to give users a jump-start on the interface (Johnson et al., 1989). And it worked. Rather than learning something new and unfamiliar, users could act out with concepts already understood: documents, folders, filing cabinets, trashcans, the top of the desk, pointing, selecting, dragging, dropping, and so on. This is the essence of mental models.

*Implementation models* are to be avoided. These are systems that impose on the user a set of interactions that follow the inner workings of an application. Cooper and Reimann give the example of a software-based fax product where the user is paced through a series of agonizing details and dialogs (Cooper and Riemann, 2003, p. 25). Interaction follows an implementation model, rather than the user's mental model of how to send a fax. The user is prompted for information when it is convenient for the program to receive it, not when it makes sense to the user. Users often have pre-existing experiences with artifacts like faxes, calendars, media players, and so on. It is desirable to exploit these at every opportunity in designing a software-based product. Let's examine a few other examples in human-computer interfaces.

Toolbars in GUIs are fertile ground for mental models. To keep the buttons small and of a consistent size, they are adorned with an icon rather than a label. An icon is a pictorial representation. In HCI, icons trigger a mental image in the user's mind, a clue to a real-world experience that is similar to the action associated with the button or tool. Icons in drawing and painting applications provide good examples. Figure 3.19a shows the Tool Palette in Corel's *Paint Shop Pro*, a painting and image manipulation application.<sup>12</sup> The palette contains 21 buttons, each displaying an icon. Each button is associated with a function and its icon is carefully chosen to elicit the association in the user's mind. Some are clear, like the magnifying glass or the paintbrush. Some are less clear. Have a look. Can you tell what action is

---

<sup>12</sup>[www.jasc.com](http://www.jasc.com).



**FIGURE 3.19**

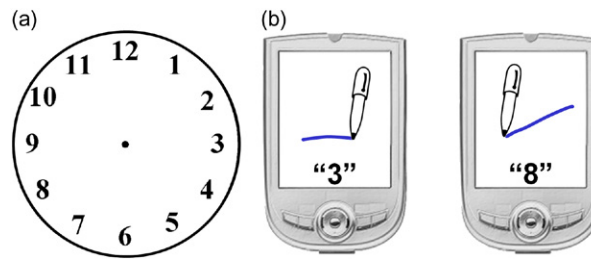
Icons create associations. (a) Array of toolbar buttons from Corel's *Paint Shop Pro*. (b) Tooltip help for "Picture Tube" icon.

associated with each button? Probably not. But users of this application likely know the meaning of most of these buttons.

Preparing this example gave me pause to consider my own experience with this toolbar. I use this application frequently, yet some of the buttons are entirely strange to me. In 1991 Apple introduced a method to help users like me. Hover the mouse pointer over a GUI button and a field pops up providing a terse elaboration on the button's purpose. Apple called the popups *balloons*, although today they are more commonly known as *tooltips* or *screen tips*. Figure 3.19b gives an example for a button in *Paint Shop Pro*. Apparently, the button's purpose is related to a picture tube. I'm still in the dark, but I take solace in knowing that I am just a typical user: "Each user learns the smallest set of features that he needs to get his work done, and he abandons the rest." (Cooper, 1999, p. 33)

Another example of mental models are a compass and a clock face as metaphors for direction. Most users have an ingrained understanding of a compass and a clock. The inherent labels can serve as mental models for direction. Once there is an understanding that a metaphor is present, the user has a mental model and uses it efficiently and accurately for direction: *north*, for straight ahead or up, *west* for left, and so on. As an HCI example, Lindeman et al. (2005) used the mental model of a compass to help virtual reality users navigate a building. Users wore a vibro-tactile belt with eight actuators positioned according to compass directions. They were able to navigate the virtual building using a mental model of the compass. There is also a long history in HCI of using a compass metaphor for stylus gestures with pie menus (Callahan et al., 1988) and marking menus (G. P. Kurtenbach, Sellen, and Buxton, 1993; Li, Hinckley, Guan, and Landay, 2005).

With twelve divisions, a clock provides finer granularity than a compass ("obstacle ahead at 2 o'clock!"). Examples in HCI include numeric entry (Goldstein, Chincholle, and Backström, 2000; Isokoski and Käksi, 2002; McQueen, MacKenzie, and Zhang, 1995) and locating people and objects in an environment (Sáenz and Sánchez, 2009; A. Sellen, Eardley, Iazdl, and Harper, 2006). Using a clock metaphor for numeric entry with a stylus is shown in Figure 3.20. Instead of scripting numbers using Roman characters, the numbers are entered using straight-line strokes. The direction of the stroke is the number's position on a clock face. In a longitudinal study, McQueen et al. (1995) found that numeric entry was about

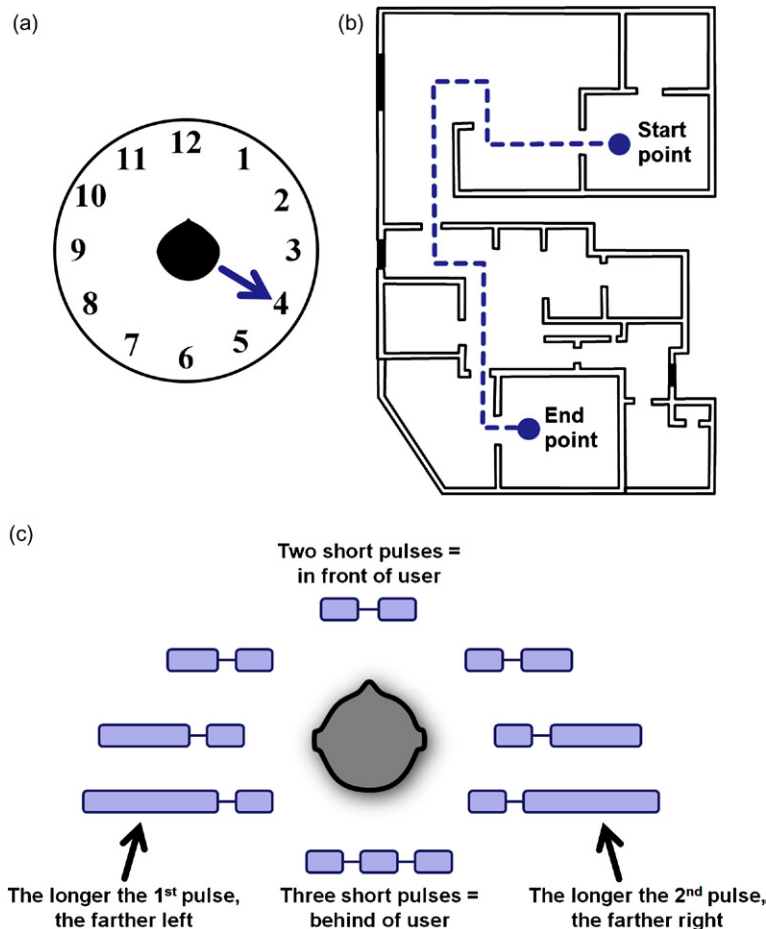
**FIGURE 3.20**

Mental model example: (a) Clock face. (b) Numeric entry with a stylus.

24 percent faster using straight-line strokes compared to handwritten digits. The 12 o'clock position was used for 0. The 10 o'clock and 11 o'clock positions were reserved for system commands.

Sáenz and Sánchez describe a system to assist the blind (Sáenz and Sánchez, 2009) using the clock metaphor. Users carried a mobile locating device that provided spoken audio information about the location of nearby objects (see [Figure 3.21a](#)). For the metaphor to work, the user is assumed to be facing the 12 o'clock position. The system allowed users to navigate a building eyes-free ([Figure 3.21b](#)). Users could request position and orientation information from the locator. Auditory responses were provided using the clock metaphor and a text-to-speech module (e.g., “door at 3 o'clock”). A similar interface is Rümelin et al.'s *NaviRadar* (Rümelin, Rukzio, and Hardy, 2012), which uses tactile feedback rather than auditory feedback. Although not specifically using the clock metaphor, *NaviRadar* leverages users' spatial sense of their surroundings to aid navigation. Users receive combinations of long and short vibratory pulses to indicate direction ([Figure 3.21c](#)). Although the patterns must be learned, the system is simple and avoids auditory feedback, which may be impractical in some situations.

The systems described by Sáenz and Sánchez (2009) and Rümelin et al. (2012) have similar aims yet were presented and evaluated in different ways. Sáenz and Sánchez emphasized and described the system architecture in detail. Although this is of interest to some in the HCI community, from the user's perspective the system architecture is irrelevant. A user test was reported, but the evaluation was not experimental. There were no independent or dependent variables. Users performed tasks with the system and then responded to questionnaire items, expressing their level of agreement to assertions such as “The software was motivating,” or “I like the sounds in the software.” While qualitative assessments are an essential component of any evaluation, the navigation and locating aides described in this work are well suited to experimental testing. Alternative implementations, even minor modifications to the interface, are potential independent variables. Speed (e.g., the time to complete tasks) and accuracy (e.g., the number of wrong turns, retries, direction changes, wall collisions) are potential dependent variables.

**FIGURE 3.21**

Spatial metaphor: (a) Auditory feedback provides information for locating objects, such as “object at 4 o’clock.” (b) Navigation task. (c) NaviRadar.

(Source: b, adapted from Sáenz and Sánchez, 2009; c, adapted from Rumelin et al., 2012)

Rumelin et al. (2012) took an empirical approach to system tests. Their research included both the technical details of *NaviRadar* and an evaluation in a formal experiment with independent variables, dependent variables, and so on. The main independent variable included different intensities, durations, and rhythms in the tactile pulses. Since their approach was empirical, valuable analyses were possible. They reported, for example, the deviation of indicated and reported directions and how this varied according to direction and the type of tactile information given. Their approach enables other researchers to study the strengths and weaknesses in *NaviRadar* in empirical terms and consider methods of improvement.

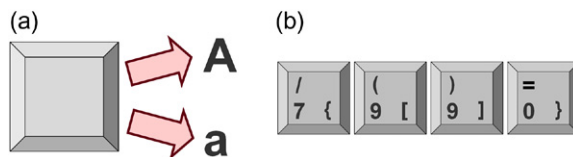
### 3.5 Modes

A common and sometimes frustrating property of user interfaces is modes. A mode is “a functioning arrangement or condition.” The soft controls in [Figure 3.3](#) demonstrate modes, since every new application brings a new functioning arrangement of controls and displays. Usually, though, modes are viewed in a more limited sense, such as lowercase mode versus uppercase mode. Either way, it is a simple fact that systems of all stripes have modes.

We confront modes everywhere, often without realizing it. For example, office phones have modes: light=*on* means *message waiting*. Electric staplers have modes: LED=*on* means *add staples*. Complex physical systems, such as lawn mowers, stoves, traffic intersections, washing machines, and so on, are more challenging. If computers are involved, all bets are off. For example, some lights at traffic intersections—under computer control—include a time-of-day mode to control the advance green signal. There is an advance green signal, but only during rush hours. Or is it non-rush hours? Traffic flow may improve, but other problems may arise, such as pedestrians or motorists incorrectly anticipating a mode.

For interactive systems, challenges with modes occur because there are fewer controls than tasks. A computer keyboard is an example. A standard desktop keyboard has about 100 keys, yet can produce more than 800 key variations, using the modes afforded by modifier keys such as SHIFT, CTRL, and ALT. For example, a key might produce *a* in one mode and *A* in another. (See [Figure 3.22a](#).) Most users are fully aware of a keyboard’s shift mode. Keyboards for European languages, such as German, include three modes for some keys. (See [Figure 3.22b](#).) No doubt, local users are comfortable with the arrangement; however, operating such a keyboard for the first time is a challenge. Still, mode problems with standard keyboards do arise. An example is when CAPS LOCK mode is on, which we’ll visit later.

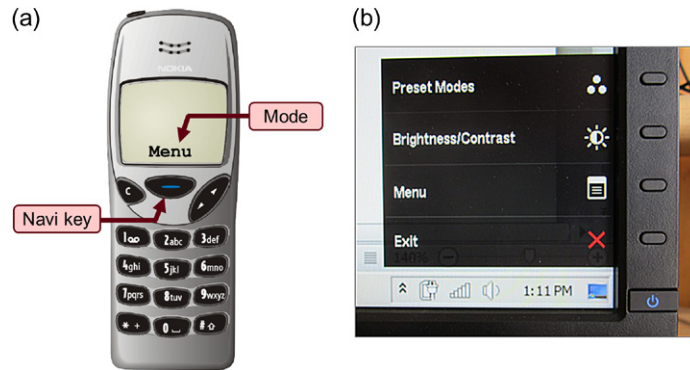
Is it over-reaching to ascribe modes to the A key on a keyboard? Not at all. Taking a research perspective in HCI involves finding simple ways to explain the complex. Having modes assigned to keys on a computer keyboard is worth thinking about. Taking a broad view of modes, or other elements of HCI, is a powerful approach to developing a comprehensive understanding that includes the intricate, the common, and the banal. This idea underpins *descriptive models*, where the goal is to delineate and categorize a problem space. Descriptive models are described in more detail in Chapter 7 (Modeling Interaction).



**FIGURE 3.22**

Keys and modes. (a) Key with two modes. (b) Keys on a German keyboard with three modes.



**FIGURE 3.23**

Physical keys with modes. (a) Navi key on the Nokia 3210 mobile phone. (b) Image adjust buttons on an LCD monitor.

Function keys have many modes. Help for Microsoft Word lists six interpretations of the F9 key. Pressed alone, it means “update selected fields.” With combinations of SHIFT, CTRL, and ALT, there are five additional interpretations.

Some mobile phones have physical keys with modes. The idea first appeared in 1997 as the Navi key, introduced by Nokia (Lindholm, Keinonen, and Kiljander, 2003, p. 24). The Nokia 3210 implementation is shown in Figure 3.23a. The Navi key was embossed with only a blue horizontal line, leaving its interpretation unclear. The trick—novel at the time—was a creative use of the display. The current action (mode) of the Navi key was revealed by the word on the display immediately above the key, so the possibilities exploded. A quick look in the 3210 User Guide reveals at least 14 interpretations of the Navi key: Menu, Select, Answer, Call, End, OK, Options, Assign, Send, Read, Use, View, List, Snooze, and Yes. The word displayed above the key provided *feedback*—a feature that is crucial for systems with modes. The close proximity of the display (the word) with the control (the Navi key) alerted the user of the current mode of the Navi key and the action associated with pressing the key.

A contemporary version of the same concept is shown in Figure 3.23b. The figure shows a portion of an LCD computer monitor. There are four physical buttons above the power button (right side of image). In a general sense the buttons are used to adjust the image or configure the display; however, no purpose is indicated. Press any button and a set of associations appears on the left, superimposed on the display. Since the associations are created in software, the purpose of the buttons can change with the context of use.

If we combine the ideas above with the earlier discussion on soft controls, there is the possibility of a *mode model for keys*. Physical keys on standard keyboards are *hard keys*. They have a few modes. The Navi key and the image adjust buttons just described are *soft-hard* keys, since they involve both a physical key and a definition of the key created through software. They have lots of modes. The small regions on GUIs, shown in Figure 3.3, are the full embodiment of *soft keys*. Their size, shape, location, and function are entirely malleable. Soft keys have an infinite number of modes.



Designing modeless interactions is sometimes viewed as desirable (Ishii and Ullmer, 1997). But the truth is, modes can't be avoided in most cases. A car, for example, has at least three transmission modes: neutral, drive, and reverse. Good luck designing a modeless car.<sup>13</sup> The same is true of computer systems where modes are unavoidable, as demonstrated above. From a design perspective, the main issues with modes are (a) changing modes, and (b) feedback, or making the mode apparent to the user. Let's examine each of these.

Figure 3.24a shows five soft controls for switching view modes in Microsoft PowerPoint. The controls persist on the display, so they are discoverable. They are clearly apparent to the user. That's good. Furthermore, tooltips (as per Figure 3.19b) lie in wait to help the user, if necessary. Also good. Changing modes is a simple matter of clicking the desired soft button. It's all quite simple, and it works reasonably well. But still problems lurk. For example, with five view modes, there are  $5 \times 4 = 20$  mode transitions. Most of these transitions are accomplished using the soft buttons shown in Figure 3.24a. However, transitioning from Slide Show mode to another mode cannot be done with the soft buttons, because they are hidden during Slide Show mode. The user must press the ESC key on the system's keyboard. Pressing ESC transitions from Slide Show mode to the previous mode. While many users know this, not all do. Can you think of an occasion when you were stuck in a mode and had to take a drastic measure to get out of it, like exiting an application or, worse yet, restarting the system?<sup>14</sup> Probably.

Small devices such as mobile phones, media players, watches, radios, and thermostats are more limited. They have a small display, if any, and just a few physical controls, but have lots of modes. Figure 3.24b demonstrates the method of switching modes on a sport watch. The buttons are physical, but there are only five. Changing modes requires cycling through the modes using a single button.

Making the mode apparent to the user is a long-standing principle in user interface design. The issue is the quality of feedback to the user. Shneiderman advises to “offer informative feedback” (2005, 74). Norman argues to “make things visible” (1988, 17–25). Obviously, problems arise if the user doesn't know what mode or state the system is in. A classic example is the *vi* editor in the *unix* operating system. There are two modes: command mode and insert mode. The application is launched in command mode. This is interesting in itself. If a user launches *vi* and immediately types *hello*, look out! The letters typed are interpreted as commands. The first key the user must type is *i* to enter insert mode. Hitting the ESC key switches back to command mode. (See Figure 3.25a.) Remarkably, there is no feedback marking the current mode, nor the transition between modes. Consequently, the *vi* editor is susceptible to mode errors, a phenomenon that is well documented in the HCI literature (e.g., Brewster, Wright, and Edwards, 1994; Norman, 1983; Poller and Garter, 1984; A. J. Sellen, Kurtenbach, and Buxton, 1992).

<sup>13</sup> Actually, it is possible, but the result is a separate car for each mode.

<sup>14</sup> Applications that include a full screen view mode typically use ESC to transition out of full screen mode. But this is not always the case. Users of Mozilla Firefox are invited to investigate this.



Figure 1 consists of two diagrams, (a) and (b), illustrating the modes of a text editor. Diagram (a) shows a cycle between 'i' (Insert) and 'Esc' (Command) with 'no feedback!' labels. Diagram (b) shows a cycle between 'Insert' and 'Overtype' modes, with 'REC', 'TRK', 'EXT', and 'OVR' buttons shown at the top and bottom.

**FIGURE 3.25**

Changing modes and mode visibility: (a) *vi* editor. (b) Microsoft Word.

The text editor *vi* is a legacy application emblematic of pre-GUI text-based command-line interaction. Most *vi* users were computer specialists—expert users. GUI-based systems necessitate a higher standard of usability, since they are targeted at a broad base of users, many of whom are not computer professionals per se. This point was argued in Chapter 1. Yet problems still exist. Similar to the *vi* example is Insert versus Overtyping mode in Microsoft Word. (See [Figure 3.25b](#).) Word’s default mode is Insert. In Insert mode, typed text appears within existing text. In Overtyping mode, typed text overwrites existing text. There are two problems with this system: changing modes and mode visibility. Changing modes uses the `INSERT` key. Simple enough, but a quick look at keyboard images reveals considerable variability in the location of the Insert key.<sup>15</sup> Seems the `INSERT` key is not important enough to have entrenched itself in a consistent location. And for good reason. Many users rarely, if ever, use this key. Furthermore, Insert is sometimes on a key with modes. In

<sup>15</sup>Many keyboard layouts can be found at <http://images.google.com>. The INSERT key may appear (a) along the top row of function keys, (b) in the numeric keypad, (c) in the block above the arrow keys, or (d) along the bottom row in the main keyboard.

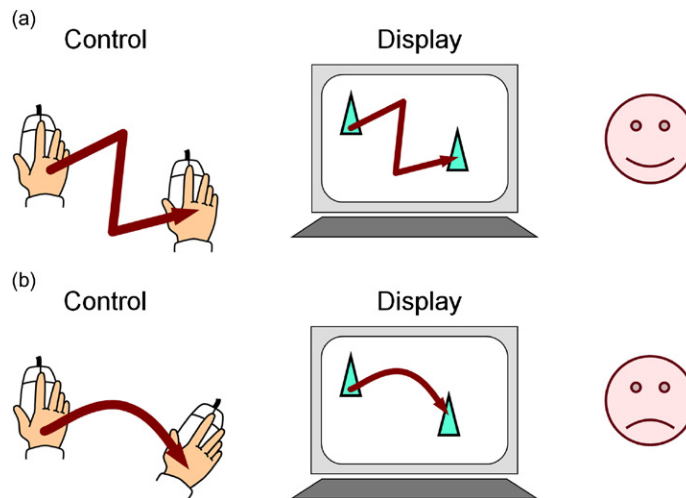
“normal mode” the key means one thing (perhaps Insert); in “function-lock mode,” it means something else. So the INSERT key may be pressed by accident, if there is confusion on the current mode of the key. This amounts to a “mode within a mode.” So changing modes is a potential problem with the INSERT key. Mode visibility is also a problem. There is feedback (see Figure 3.25b), but it is in a small display along the bottom of the screen. It is so subtle that many users are not likely to notice it. Furthermore, a user who inadvertently enters Overtyping mode may not have a linguistic association of overtyping with the problem they are experiencing. What is OVR? This is an example of a *gulf of evaluation*—a disconnect between what the system provides and what the user intends and expects (Norman, 1988, p. 51).<sup>16</sup>

From an empirical research perspective, it is worth thinking about ways to improve interaction that involves modes. Feedback is the critical issue. Can audio feedback improve a user’s awareness of mode transitions? Can a larger visual indication help, and if so, what are the trade-offs? Is mode awareness improved if a large visual indication appears in the center of the screen when a mode is entered, and then shrinks to a small persistent mode display at the bottom of the screen? See also student exercises 3-5 and 3-6 at the end of this chapter.

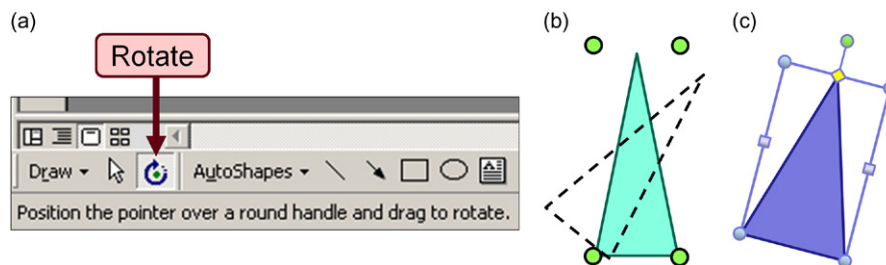
Modes are common in graphics and 3D interaction. Consider the control-display operations in Figure 3.26. In (a) the hand + mouse movement combines side-to-side and forward-backward motion. The object manipulated on the display moves similarly. Of course, the hand + mouse motion is on the desktop, or  $x$ - $z$  plane, while the object motion is on the display, or  $x$ - $y$  plane, using the labeling in Figure 3.5. Despite the spatial incongruence, users are accustomed to this (learned) relationship. In Figure 3.26b the hand + mouse action includes a sweeping or rotating motion. There is an additional deficiency here. Note that the final position of the triangle is the same in (a) and (b). Yet the hand + mouse movements were different. Let’s examine this in more detail.

If the user wanted the triangle to move with the hand + mouse, then there is a breakdown in the interaction. This occurs because of a gap between the capabilities of 2D pointing devices, such as a mouse, and the characteristics of a 2D surface. The typical fix is a *rotate mode*. To rotate the triangle in a typical graphics application, a few steps are involved: (1) select the triangle, (2) select the application’s rotate tool, (3) acquire a handle on the triangle, (4) drag the handle. This is shown in Figure 3.27. Part (a) shows the Microsoft PowerPoint rotate tool, with instructions below on using it. Part (b) shows the triangle being rotated. Side-to-side  $x$ -axis motion or forward-backward  $z$ -axis motion of the mouse rotates the triangle. There is clearly a disconnect. We manage, but the interaction is awkward and difficult. For one, translation of the triangle is prevented while in rotate mode. This control-display relationship is anything but natural.

<sup>16</sup>User frustration with Insert versus Overtyping mode in Microsoft Word is a well-worn discussion in online help forums. With Office 2010, the insert key is disabled by default but is enabled through a setting in the Options | Advanced dialog box. If enabled, the INSERT key toggles between Insert mode and Overtyping mode; however, there is no feedback indicating the current mode.

**FIGURE 3.26**

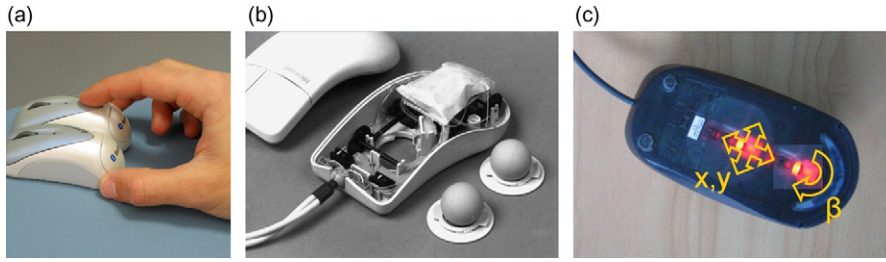
Two-dimensional movement with the hand + mouse: (a)  $x$  and  $y$  displacement only. (b)  $z$ -axis rotation of the hand + mouse is not sensed, so the object on the display undergoes  $x$  and  $y$  displacement only.

**FIGURE 3.27**

Rotate mode: (a) Microsoft PowerPoint's rotate tool is selected, with instructions below on using it. (b) The selected triangle enters rotate mode. The  $x$  and  $y$  displacement of the mouse with the button down (dragging) is used to rotate the triangle. (c) Newer version of PowerPoint. Rotate handle appears when triangle is selected.

In newer versions of PowerPoint, a rotate handle appears whenever an object is selected. (See Figure 3.27c.) This eliminates the need for a rotate mode. However, the control-display relationships are the same—one mapping for rotation, a separate mapping for translation.

Another way to eliminate the rotate mode in 2D is to reengineer a mouse to sense rotation. Several configurations are possible. Almeida and Cubaud (2006) glued two small mice together. (See Figure 3.28a.) MacKenzie et al. (1997) put the mechanical and electrical components of two mice into a single chassis. (See



**FIGURE 3.28**

Mouse configurations sensing rotation: (a) Two small mice glued together (Almeida and Cubaud, 2006). (b) The components from two mice assembled in a single chassis (MacKenzie et al., 1997). (c) Optical mouse with additional camera for rotation.

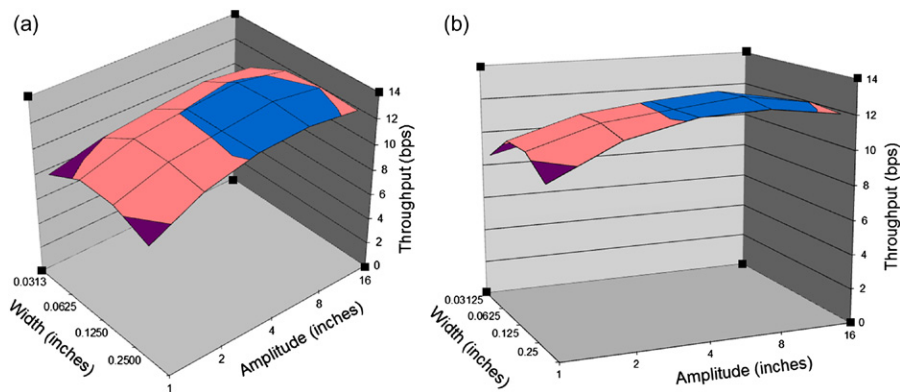
(Source: a, photo courtesy of Rodrigo Almeida; c, Hannagan and Regenbrecht, 2008; Photo courtesy of Holger Regenbrecht)

Figure 3.28b.) Contemporary designs are more likely to use camera technology. Hannagan and Regenbrecht's (2008) *TwistMouse* is an example. (See Figure 3.28c.) The latter two configurations are advantageous from the user's perspective, since the device has the look and feel of a conventional mouse. For all the examples in Figure 3.28, the prototype is equivalent to two mice and requires a custom driver to handle both data streams. If the device is maneuvered with  $x$ - and  $z$ -axis displacement only, the two data streams are the same. If movement includes  $y$ -axis rotation, the two data streams are slightly different. With a little trigonometry, the driver is able to produce  $y$ -axis rotation data coincident with  $x$ - and  $z$ -axis translation. Thus, an acquired object can undergo translation and rotation at the same time.

There are interesting design issues for 3 DOF mice. Usually the extra DOF is not needed; the rotation data can be ignored most of the time. When a task requires both translation and rotation, a modifier key can engage the third DOF. This in itself is interesting. Recall that this analysis began as an effort to eliminate the rotate mode in graphics applications. Yet the solution includes modes: a 2 DOF translate-only mode and a 3 DOF translate + rotate mode.

Another challenge is in accommodating the biomechanics of the lower arm. It is difficult to effect the required wrist movement if an on-screen object is rotated through a wide angle, such as 200 degrees. A solution is to use another modifier key to amplify the angular mapping of mouse rotation. This is somewhat analogous to CD gain, except the gain is applied to rotation. Hinckley et al. (1999) take a different approach. Their system is camera-based, with a mouse pad serving as the ground plane for the camera image. They propose a two-handed technique: one hand manipulates and rotates the mouse, while the other hand counter-rotates the mouse pad. Large rotation angles are thereby possible.

Although 3 DOF mice demo well, a complete bottom-up redesign is necessary for deployment as a product. Today's optical mice are well suited to this. Since the built-in camera captures 2D images (e.g.,  $18 \times 18$ ) of the desktop surface at 1,000+

**FIGURE 3.29**

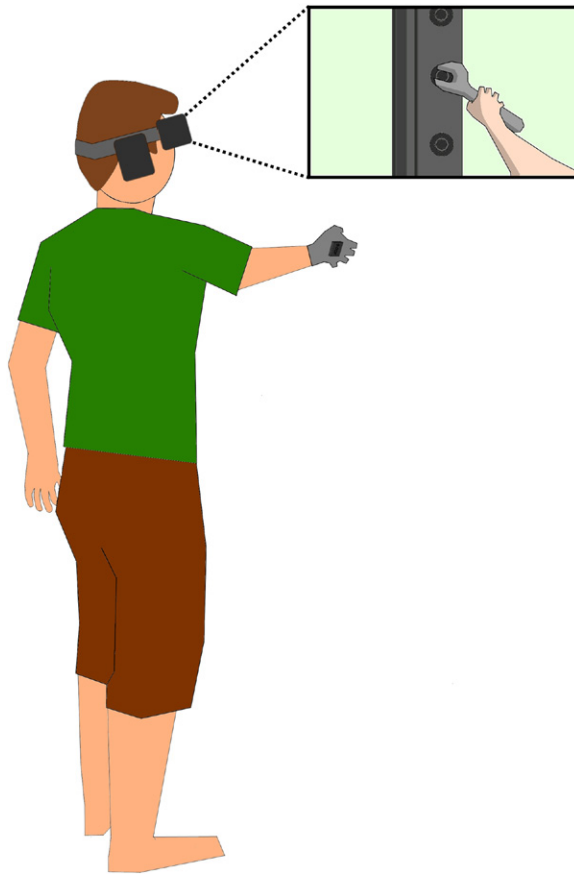
Two views of the same data in the same chart. 3D rotation is performed by dragging a handle (small black square) with the mouse.

frames per second, it should be relatively easy to determine the rotational component of mouse movement without a second camera (as used in the TwistMouse; see [Figure 3.28c](#)). However, turning such initiatives into products is easier said than done. Besides re-engineering the hardware and firmware, a new driver protocol is needed (with y-axis rotation data) along with modifications to applications to effectively use the full potential of 3 DOF data.

The situation is more complicated in 3D environments, since there are six degrees of freedom (see [Figure 3.7](#)). If the input controller has fewer than six degrees of freedom, then modes are needed to fully access the environment. A particularly unnatural interaction is shown in [Figure 3.29](#). The figure shows a 3D data plot.<sup>17</sup> (It is known as a “surface chart” in Microsoft *Excel*.) Part (a) shows one view. To alter the view, the chart is selected. Handles (small black squares) appear and rotate mode is entered. A handle is acquired and dragged with the mouse, much like the 2D rotation presented above. A rotated view of the data is shown in part (b). But the interaction is clumsy. Rotation about two axes is supported, but it is extremely difficult to control. One is never quite sure what effect a particular mouse movement will produce. Interaction degrades to trial and error.

Desktop systems are simply inappropriate for this sort of interaction. If 3D is the goal, then a more sophisticated setup is needed. Typically, different or additional devices are used to provide more natural control-display relationships. These systems may include a head-mounted display and a 6 DOF motion tracker to provide an improved view of 3D objects in 3D space. Not only is the head position tracked with six degrees of freedom, the image provided to each eye is altered slightly to provide depth cues, making the virtual scene appear three-dimensional.

<sup>17</sup>The data are from the pin-transfer task in Fitts', 1954 paper (Fitts, 1954, Table 3). Fitts includes a hand-drawn 3D plot similar to that shown here (Fitts, 1954, [Figure 4](#)).

**FIGURE 3.30**

A head-mounted display along with 6 DOF head and hand trackers provides 3D viewing and control.

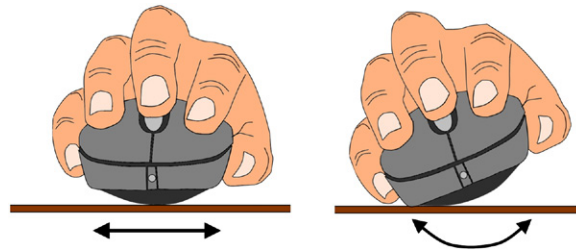
*(Sketch courtesy of Bartosz Bajer)*

With the addition of a 6 DOF hand tracker, more powerful 3D control is possible. An example setup is shown in [Figure 3.30](#) (see also Garau et al., 2003).

### 3.6 More about degrees of freedom

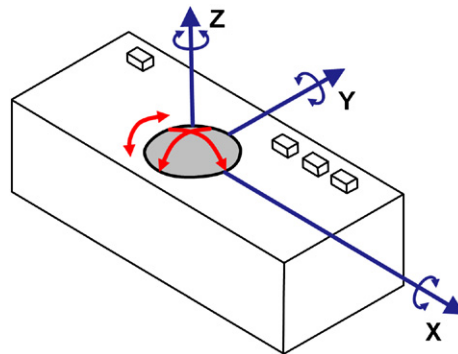
The research literature in HCI has numerous examples of devices engineered to provide additional sensing capabilities. The *Rockin' Mouse* is a 4 DOF mouse with a rounded bottom (Balakrishnan, Baudel, Kurtenbach, and Fitzmaurice, 1997). It senses planar  $x$ - $z$  movement, like a regular mouse, but also senses rocking side-to-side ( $\theta_z$ ) and to-and-fro ( $\theta_x$ ) rotation. (See [Figure 3.31](#).)



**FIGURE 3.31**

The Rockin'Mouse works like a mouse but also senses side-to-side ( $\theta_z$ ) rotation (shown) and forward-backward ( $\theta_x$ ) rotation. (Balakrishnan et al., 1997). Rotation about the  $y$ -axis is not sensed.

*(Sketch courtesy of Bartosz Bajer)*

**FIGURE 3.32**

Three-axis trackball.

*(Adapted from Evans et al., 1981)*

The prototype operated on the surface of a tablet. A cordless sensor was mounted in the center of the Rockin'Mouse. The sensor was extracted from the tablet's stylus. From the tablet's perspective, the Rockin'Mouse is a stylus. From the user's perspective, it looks and feels like a mouse, except for the rounded body. The tablet senses and reports the  $x$  and  $z$  positions of the sensor as well as tilt ( $\theta_x$  and  $\theta_z$ ). Balakrishnan et al. (1997) evaluated the Rockin'Mouse in a 3D positioning task using tilt for the up-down movement of the object. In the mouse comparison condition, participants had to select a handle on the object to switch movement modes. Overall, the Rockin'Mouse was about 30 percent faster than the standard mouse.

Evans et al. (1981) describe a three-axis trackball that senses rotation about each of the  $x$ -,  $y$ -, and  $z$ -axes. (See Figure 3.32.) Even though the usual operation of a trackball is  $x$ - $y$  planar control of a cursor, the ball rotates, so the labeling of the degrees of freedom in Figure 3.32 is appropriate. A device like this would likely perform quite well in controlling the view of the 3D surface plot in Figure 3.29.





**FIGURE 3.33**

Etch-A-Sketch children's toy with separate controls for the  $x$  and  $y$  degrees of freedom.<sup>18</sup>

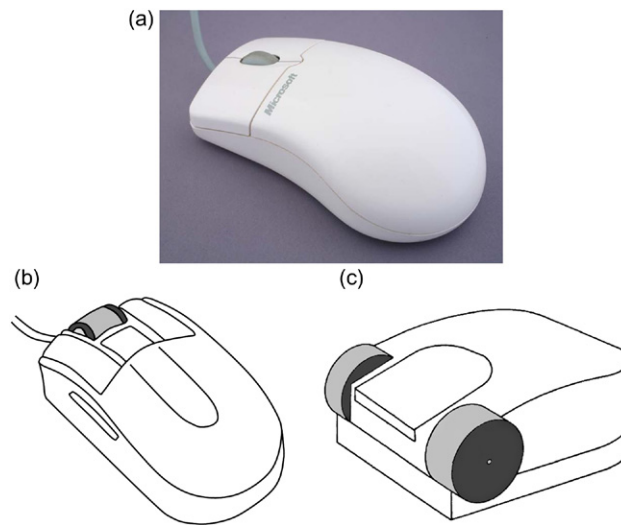
Of course, it is relatively easy to include modes, allowing the trackball to also operate in mouse-emulation mode.

There is more to degrees of freedom than simply counting them. It is important to consider the relationship between device properties, such as degrees of freedom, and task requirements. Jacob et al. (1994) speak of the *integrality* and *separability* of the degrees of freedom. With a regular mouse, controlling one degree of freedom in isolation of the other is difficult. Consider drawing a horizontal line with a mouse using the drawing mode of an application. The task is difficult because the  $x$  and  $y$  degrees of freedom are integrated. Of course, degrees of freedom can be separated. A classic example for  $x$ - $y$  positioning is the children's toy *Etch-A-Sketch* introduced in the 1950s by the Ohio Art Company. (See Figure 3.33.) There are separate 1 DOF controllers to move the drawing instrument along the  $x$ - and  $y$ -axes. Drawing a horizontal line is easy. Drawing a diagonal line is a challenge, however. See Buxton (1986) for additional discussion on input using the Etch-A-Sketch and other toys.

Another example for  $x$ - $y$  positioning is the wheel mouse, released in 1996 as Microsoft's *IntelliMouse*. (See Figure 3.34a.) The rotating wheel gives the mouse an extra—and separate—degree of freedom. A less successful version was Mouse System's *ProAgio*, released about a year earlier. (See Figure 3.34b.) Even earlier was Venolia's *Roller Mouse* (1993), presented at the annual ACM *SIGCHI* conference in 1993. (See Figure 3.34c.) The Roller Mouse had two wheels. The primary intention with the Roller Mouse was to control the many, and often separate, degrees of freedom in 3D graphics applications.

Additional but separate degrees of freedom are evident in Balakrishnan and Patel's *PadMouse* (1998). (See Figure 3.35a.) A touchpad was added to the surface of a regular mouse. The intent was to support pie menu access with gestures on the touchpad while the mouse operated as usual, controlling an on-screen cursor.

<sup>18</sup>From the Buxton Collection (<http://research.microsoft.com>).

**FIGURE 3.34**

Wheel mice: (a) Microsoft's Intellimouse. (b) Mouse Systems' ProAgio. (c) *Roller Mouse* research prototype.

(Source: b, adapted from Gillick and Lam, 1996; c, adapted from Venolia and Ishikawa, 1994)

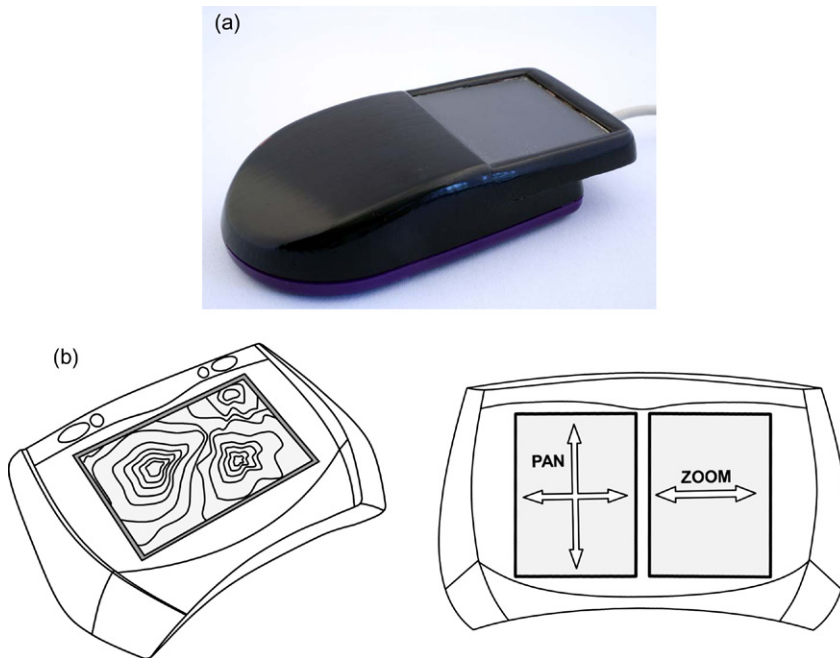
Clearly, combining interaction in this manner requires separating the degrees of freedom for pie menu access and cursor control.

Silfverberg et al. (2003) describe a display screen that allows the displayed image to be zoomed and panned by two touch sensors positioned underneath the device (See Figure 3.34b.). An obvious advantage is that occlusion is avoided since the fingers act below the display. Also, the 1 DOF zooming and 2 DOF panning operations are separated and thus do not interfere.

Villar et al. (2009) describe a variety of mouse configurations instrumented to detect touch, including *multi-touch*. The devices use infrared (IR) transmitters that inject light into the edge of an acrylic surface. An IR camera below the surface receives an image of the surface. When fingers contact the surface, the light is scattered at the points of contact. Image processing software determines the location and pressure of the points of contact. Some of the device configurations are shown in Figure 3.36. The devices support multi-touch gestures as well as regular mouse operations.

One configuration for touch detection is the CapMouse (second from left in Figure 3.36). It uses a capacitive sensor rather than an IR transmitter and camera. The setup is similar to Balakrishnan and Patel's PadMouse (Figure 3.35a), except the touch-sensing surface is contoured and seamlessly integrates with the device chassis.

The message in this section is that degrees of freedom (DOF) is a property that spans a rich complement of tasks and interaction techniques. User interactions are



**FIGURE 3.35**

Adding degrees of freedom with a touchpad: (a) PadMouse (Balakrishnan and Patel, 1998).<sup>19</sup> (b) Zooming and panning display screen (Silfverberg, Korhonen, and MacKenzie, 2003), top view (*left*), bottom view (*right*).



**FIGURE 3.36**

Multi-touch sensing input devices (Villar et al., 2009).

(Photo courtesy of Nicolas Villar)

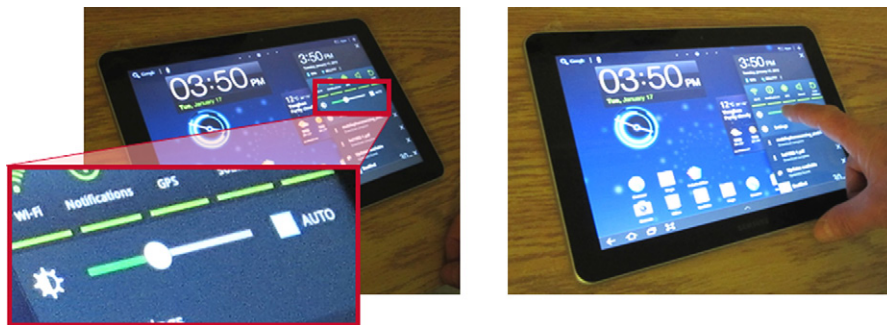
plentiful, and they are juxtaposed in complex ways, such as overlapping, sequential, and in parallel. The degrees of freedom in the interactions merit deeper consideration. Opportunities for empirical research are plentiful. Let's move on to the unique interactions in the world of mobile computing.

<sup>19</sup>From the Buxton Collection (<http://research.microsoft.com>).

### 3.7 Mobile context

Mobile computing dates to the 1980s, with the emergence of pocket organizers and personal digital assistants (PDAs). This was followed in the early 1990s with the first pen-based tablet computers and mobile phones supporting text messaging. The 2007 release of the Apple iPhone was a significant event from a user interface standpoint. With the arrival of the iPhone and follow-on products, such as touch-based tablet computers, the landscape of HCI experienced a dramatic shift. Pressing keys on small keyboards or gesturing with a stylus gave way to finger actions: swiping, flicking, pinching, tapping, and so on. Furthermore, devices in this genre include additional components to enhance the user experience. Cameras, light sensors, vibro-tactile actuators, accelerometers, gyroscopes, GPS receivers, and so on work together in broadening the experience for users. With the prefix “smart,” these devices are full-fledged media players, providing unprecedented connectivity to users through both Third Generation (3G) and Internet access.

Many of the interaction elements noted above for desktop environments don’t even exist in the mobile context. This follows from the distinction between indirect input and direct input. Cursor control using a mouse or touchpad is an example of *indirect input* because the system response is spatially displaced from the input device. Finger input on a touch-sensitive display uses *direct input* because the system response is located at the point of touch. On a touch-sensitive display, a cursor or on-screen tracking symbol is not needed. Properties like control-display mappings and CD gain do not exist for direct input. Mouse-based systems, on the other hand, *require* an on-screen tracking symbol such as a cursor or pointer. Of course, this is a by-product of indirect input. Direct touch input is the full embodiment of WYSIWYG interaction – *what you see is what you get*. A simple example on a tablet computer appears in Figure 3.37. A slider to control screen brightness appears on the tablet’s display (left). The finger interacts directly with the slider (right).



**FIGURE 3.37**

Example of direct input: A slider GUI element to control the display brightness (*expanded, left*) is engaged directly by the user’s finger (*right*).

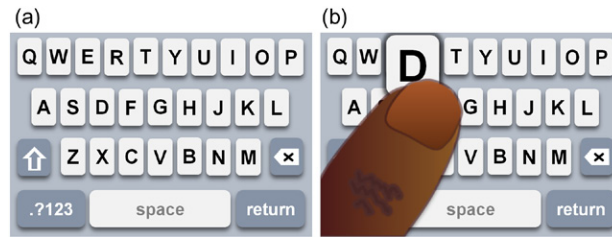
However, direct touch input has an unfortunate side effect: precise pixel-level selection is difficult. On a desktop system, the tip of the on-screen pointer visually identifies the selection point. With touchscreens, users typically do not know, and cannot control, the precise location of the contact point. There are two reasons: (1) the finger occludes the location of contact, and (2) finger contact forms an area, not a point. These issues are sometimes called the “fat finger problem” (Wigdor, Forlines, Baudisch, Barnwell, and Shen, 2007) and follow from the physiology of the human finger. The size and shape of the contact area varies from user to user and also by usage patterns, such as the force applied or the reach of the arm.

New technologies bring possibilities, which in turn bring challenges. With challenges come ideas for solutions. Early research on touchscreens focused on their use in information kiosks. Improving the precision of pointing was a key challenge. How can occlusion and precision be accommodated for touch input? Are there alternate methods to consider and, if so, which ones perform better? These are the sorts of questions that underpin empirical research in HCI. One of the earliest selection enhancements for touchscreens was the *offset cursor* presented by Potter et al. (1988) at the ACM SIGCHI conference in 1988.<sup>20</sup> When the finger contacts the display surface, a crosshair cursor appears just above the fingertip. The user maneuvers his or her finger along the display surface, positions the cursor within a selectable target, then lifts the finger to select the target. Potter et al. called the method *take-off selection*. They described an experimental evaluation to compare take-off selection with two alternate methods, *first-contact*, where selection occurs immediately when the contact point enters a target, and *land-on*, where selection occurs only if the initial contact point is on a target. Of the three methods, they found take-off most accurate, but first-contact fastest. The main point here is that Potter et al. chose a methodology that was both empirical and experimental to present and evaluate a novel interaction method and to compare it with alternatives.

Variations on Potter et al.’s offset cursor combined with take-off selection are seen in contemporary touchscreen devices. The most recognizable is perhaps the soft keyboard on the Apple *iPhone*. Figure 3.38a shows a portion of the keyboard, including the D key. In Figure 3.38b, the user’s finger is in contact with the D key. The key is occluded. However, an offset animation alerts the user that the D key is tentatively selected. The user may slide his or her finger to another key or lift the finger to enter D.

Yet another difference between touch input and mouse input is the ability of touch-sensing technologies to sense multiple contact points. Indeed, multi-touch is one of the most provocative interaction techniques to enter mainstream computing in recent years. An example is shown in Figure 3.39 where a map is maneuvered by a sliding gesture, then resized with a two-finger reverse pinching gesture.

<sup>20</sup>Unfortunately, an image of the offset cursor in use was not included in the cited paper or in related papers. However, a video by Sears and Shneiderman from 1991 shows the offset cursor. (Search YouTube using “1991\_touchscreenkeyboards.mpg.”)

**FIGURE 3.38**

Enhancing touchscreen selection: (a) Apple iPhone soft keyboard. (b) User finger on D key. Selection occurs on finger lift.

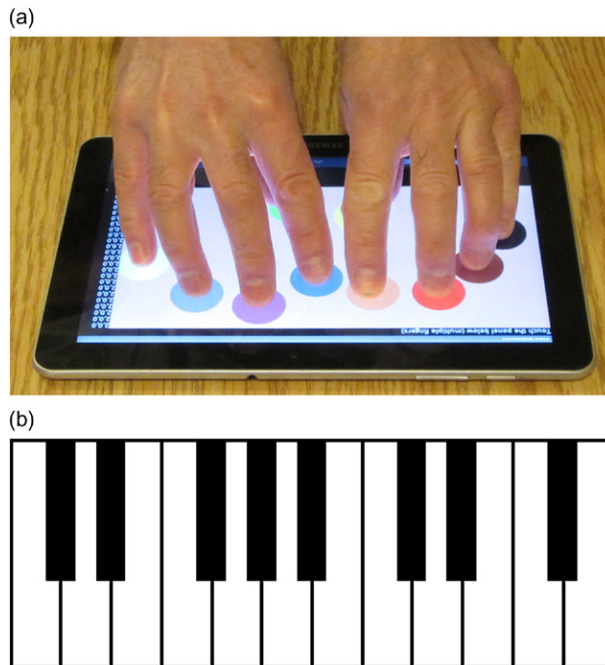
**FIGURE 3.39**

Touch example: A map image (1) is moved by touching the map (2) and dragging the finger (3). The image is zoomed in by touching with two fingers (4) and widening the distance between the fingers (5).

Multi-touch does not end at two points of contact. Recent devices can sense several contact points. Figure 3.40a is a multi-touch demonstration on a Samsung Galaxy Tab 10.1 running the Android 3.1 operating system. Ten contact points are sensed. Although the thumb contacts are partly occluded in the image, ten contact points are apparent. They appear as large colored circles. Beginning with the pinkie contact on the right, the circles follow the color coding for numbers on electronic resistors: black (0), brown (1), red (2), orange (3), yellow (4), green (5), blue (6), violet (7), grey (8), and white (9).

It is easy to imagine applications for input using multiple contacts points ( $>2$ ). A simple example is the piano keyboard in Figure 3.40b. This was suggested as early as 1985 as an application of the multi-touch input system described by Lee et al. (1985). Lee et al.'s system also sensed pressure (as do contemporary touch-sensitive displays). With added pressure data, both the pitch and loudness of notes





**FIGURE 3.40**

Multi-touch demonstration: (a) Ten contact points are sensed. Markers, beginning with the user's pinkie finger on the right, follow the color codes for electronic resistors. (b) Piano keyboard application.

can be controlled upon finger contact. Multi-touch piano keyboard applications are readily downloadable for Android tablets.

Besides the use of multi-touch, today's mobile devices include a host of additional components. A good example is the accelerometer. The technology has matured in terms of pricing, size, data speed, precision, and power consumption to the point where accelerometers are now a standard component in most touchscreen phones and tablet computers. An accelerometer allows the angle or tilt of the device to be sensed. Tilt as an input primitive has entered the mainstream of interaction techniques in mobile computing.

Early research on tilt, dating to the 1990s, examined its potential for tasks such as document scrolling (Bartlett, 2000; B. Harrison et al., 1998; Small and Ishii, 1997), panning and zooming (Small and Ishii, 1997), menu navigation (Rekimoto, 1996), and changing screen orientation (Hinckley, Pierce, Sinclair, and Horvitz, 2000). As with any new technology, research initially focused on what can be done rather than on how well it can be done. The focus was on the technology and its integration in prototype devices. Test results with users tended to be anecdotal, not empirical. In fact, tilt is a user input primitive that is well suited to empirical inquiry. For

each of the tasks just cited, several input methods either currently exist or can be envisioned as alternatives. Implementing the tasks using tilt is novel and exciting. But how well can the tasks be executed? How does tilt input compare to alternative methods in terms of human performance? What aspects of tilt influence human performance? These questions are well suited to empirical inquiry. One such evaluation for tilt has recently been completed. The evaluation tested four settings of tilt gain: 25, 50, 100, 200 (MacKenzie and Teather, 2012).<sup>21</sup> The experiment used a position-select task similar to the multi-direction task in ISO 9241-9 (ISO 2000; Soukoreff and MacKenzie, 2004). Results suggest that performance is best (i.e., lower task base completion time, higher throughput) for a tilt gain setting in the range of 50 to 100.

Another interesting application of accelerometer input is a *spatially aware display*. In this case, a view on the display moves as the device moves, as sensed by the accelerometer. (See Figure 3.41.) A variety of new issues arise. For planar 2D movement, the interaction is straightforward. But what if the device rotates, or moves in and out. How should the view change? Is the interaction fluid and natural? These issues are well-suited to empirical inquiry. Experimental studies of spatially aware displays are reported by Fitzmaurice (1993), Rohs and Oulasvirta (2008), Cao et al. (2008), and others.

If there is one observation that emerges from the annual ACM SIGCHI conference, it is that the pace of innovation in HCI is increasing. New technologies and new ideas for interactions are plentiful and they are emerging quickly. They are often presented, like tilt was initially, as possibilities. Prototypes, demos, videos, posters, work-in-progress contributions, and so on, get the ideas out there. But sooner or later, plausible ideas need testing with users. Testing that is empirical and experimental is recommended. The results will add to the knowledge base that engineers and designers draw upon in bringing to market new and cool products.



**FIGURE 3.41**

Spatially aware display. As the device moves, the view into the virtual image moves.

<sup>21</sup>Tilt gain is a multiplier that maps the magnitude of device tilt (degrees) to the velocity of an object on the display (pixels per second).



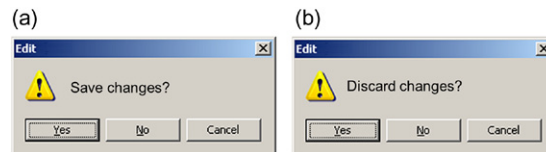
### 3.8 Interaction errors

Most of analyses in this chapter are directed at the physical properties of the human-machine interface, such as degrees of freedom in 2D or 3D or spatial and temporal relationships between input controllers and output displays. Human performance, although elaborated in Chapter 2, has not entered into discussions here, except through secondary observations that certain interactions are better, worse, awkward, or unintuitive. At the end of the day, however, human performance is what counts. Physical properties, although instructive and essential, are secondary. Put another way, human performance is like food, while physical properties are like plates and bowls. It is good and nutritious food that we strive for.

Empirical research in HCI is largely about finding the physical properties and combinations that improve and enhance human performance. We conclude this chapter on interaction elements with comments on that nagging aspect of human performance that frustrates users: interaction errors. Although the time to complete a task can enhance or hinder by degree, errors only hinder. Absence of errors is, for the most part, invisible. As it turns out, errors—interaction errors—are germane to the HCI experience. The big errors are the easy ones—they get fixed. It is the small errors that are interesting.

As the field of HCI matures, a common view that emerges is that the difficult problems (in desktop computing) are solved, and now researchers should focus on new frontiers: mobility, surface computing, ubiquitous computing, online social networking, gaming, and so on. This view is partially correct. Yes, the emerging themes are exciting and fertile ground for HCI research, and many frustrating UI problems from the old days are gone. But desktop computing is still fraught with problems, lots of them. Let's examine a few of these. Although the examples below are from desktop computing, there are counterparts in mobile computing. See also student exercise 3-8 at the end of this chapter.

The four examples developed in the following discussion were chosen for a specific reason. There is a progression between them. In severity, they range from serious problems causing loss of information to innocuous problems that most users rarely think about and may not even notice. In frequency, they range from rarely, if ever, occurring any more, to occurring perhaps multiple times every minute while users engage in computing activities. The big, bad problems are well-traveled in the literature, with many excellent sources providing deep analyses on what went wrong and why (e.g., Casey, 1998, 2006; Cooper, 1999; Johnson, 2007;



**FIGURE 3.42**

HCI has come a long way: (a) Today's UIs consistently use the same, predictable dialog to alert the user to a potential loss of information. (b) Legacy dialog rarely (if ever) seen today.

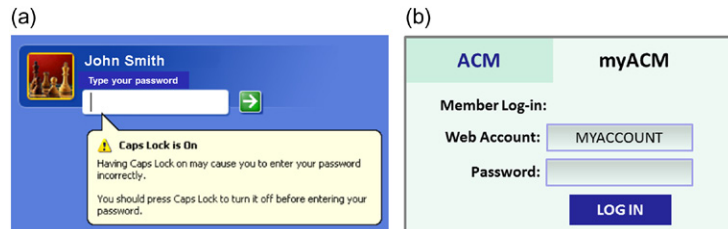
B. H. Kantowitz and Sorkin, 1983; Norman, 1988). While the big problems get lots of attention, and generally get fixed, the little ones tend to linger. We'll see the effect shortly. Let's begin with one of the big problems.

Most users have, at some point, lost information while working on their computers. Instead of saving new work, it was mistakenly discarded, overwritten, or lost in some way. Is there any user who has not experienced this? Of course, nearly everyone has a story of losing data in some silly way. Perhaps there was a distraction. Perhaps they just didn't know what happened. It doesn't matter. It happened. An example is shown in Figure 3.42. A dialog box pops up and the user responds a little too quickly. Press ENTER with the "Save changes?" dialog box (Figure 3.42a) and all is well, but the same response with the "Discard changes?" dialog box spells disaster (Figure 3.42b). The information is lost. This scenario, told by Cooper (1999, 14), is a clear and serious UI design flaw. The alert reader will quickly retort, "Yes, but if the 'Discard changes?' dialog box defaults to 'No,' the information is safe." But that misses the point. The point is that a user expectation is broken. Broken expectations sooner or later cause errors.

Today, systems and applications consistently use the "Save changes?" dialog box in Figure 3.42a. With time and experience, user expectations emerge and congeal. The "Save changes?" dialog box is expected, so we act without hesitating and all is well. But new users have no experiences, no expectations. They will develop them sure enough, but there will be some scars along the way. Fortunately, serious flaws like the "Discard changes?" dialog box are rare in desktop applications today.

The following is another error to consider. If prompted to enter a password, and CAPS\_LOCK mode is in effect, logging on will fail and the password must be reentered. The user may not know that CAPS\_LOCK is on. Perhaps a key-stroking error occurred. The password is reentered, slowly and correctly, with the CAPS\_LOCK mode still in effect. Oops! Commit the same error a third time and further log-on attempts may be blocked. This is not as serious as losing information by pressing ENTER in response to a renegade dialog box, but still, this is an interaction error. Or is it a design flaw? It is completely unnecessary, it is a nuisance, it slows our interaction, and it is easy to correct. Today, many systems have corrected this problem (Figure 3.43a), while others have not (Figure 3.43b).

The CAPS\_LOCK error is not so bad. But it's bad enough that it occasionally receives enough attention to be the beneficiary of the few extra lines of code necessary to pop up a CAPS\_LOCK alert.



**FIGURE 3.43**

Entering a password: (a) Many systems alert the user if CAPS\_LOCK is on. (b) Others do not.

Let's examine another small problem. In editing a document, suppose the user wishes move some text to another location in the document. The task is easy. With the pointer positioned at the beginning of the text, the user presses and holds the primary mouse button and begins dragging. But the text spans several lines and extends past the viewable region. As the dragging extent approaches the edge of the viewable region, the user is venturing into a difficult situation. The interaction is about to change dramatically. (See Figure 3.44.) Within the viewable region, the interaction is position-control—the displacement of the mouse pointer controls the *position* of the dragging extent. As soon as the mouse pointer moves outside the viewable region, scrolling begins and the interaction becomes velocity-control—the displacement of the mouse pointer now controls the *velocity* of the dragging extent. User beware!

Once in velocity-control mode, it is anyone's guess what will happen. This is a design flaw. A quick check of several applications while working on this example revealed dramatically different responses to the transition from position control to velocity control. In one case, scrolling was so fast that the dragging region extended to the end of the document in less time than the user could react ( $\approx 200$  ms). In another case, the velocity of scrolling was controllable but frustratingly slow. Can you think of a way to improve this interaction? A two-handed approach, perhaps. Any technique that gets the job done and allows the user to develop an expectation of the interaction is an improvement. Perhaps there is some empirical research waiting in this area.

Whether the velocity-control is too sensitive or too sluggish really doesn't matter. What matters is that the user experience is broken or awkward. Any pretense to the interaction being facile, seamless, or transparent is gone. The user will recover, and no information will be lost, but the interaction has degraded to error recovery. This is a design error or, at the very least, a design-induced error. Let's move on to a very minor error.

When an application or a dialog box is active, one of the UI components has focus and receives an event from the keyboard if a key is pressed. For buttons,

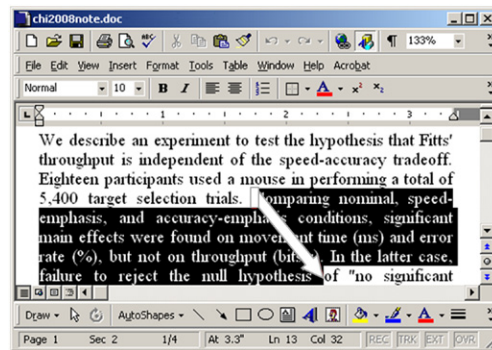


FIGURE 3.44

On the brink of hyper-speed scrolling. As the mouse pointer is dragged toward the edge of the viewable region, the user is precipitously close to losing control over the speed of dragging.

focus is usually indicated with a dashed border (see “Yes” button in Figure 3.42). For input fields, it is usually indicated with a flashing insertion bar (“|”). “Focus advancement” refers to the progression of focus from one UI component to the next. There is wide-spread inconsistency in current applications in the way UI widgets acquire and lose focus and in the way focus advances from one component to the next. The user is in trouble most of the time. Here is a quick example. When a login dialog box pops up, can you immediately begin to enter your username and password? Sometimes yes, sometimes no. In the latter case, the entry field does not have focus. The user must click in the field with the mouse pointer or press TAB to advance the focus point to the input field. Figure 3.43 provides examples. Both are real interfaces. The username field in (a) appears with focus; the same field in (b) appears without focus. The point is simply that users don’t know. This is a small problem (or is it an interaction error?), but it is entirely common. Focus uncertainty is everywhere in today’s user interfaces. Here is another, more specific example:

Many online activities, such as reserving an airline ticket or booking a vacation, require a user to enter data into a form. The input fields often require very specific information, such as a two-digit month, a seven-digit account number, and so on. When the information is entered, does focus advance automatically or is a user action required? Usually, we just don’t know. So we remain “on guard.” Figure 3.45 gives a real example from a typical login dialog box. The user is first requested to enter an account number. Account numbers are nine digits long, in three three-digit segments. After seeing the dialog box, the user looks at the keyboard and begins entering: 9, 8, 0, and then what? Chances are the user is looking at the keyboard while entering the numeric account number. Even though the user can enter the entire nine digits at once, interaction is halted after the first three-digit group because the user doesn’t know if the focus will automatically advance to the next field. There are no expectations here, because this example of GUI interaction has not evolved and stabilized to a consistent pattern. Data entry fields have not reached the evolutionary status of, for example, dialog boxes for saving versus discarding changes (Figure 3.42a). The user either acts, with an approximately 50 percent likelihood of committing an error, or pauses to attend to the display (*Has the focus advanced to the next field?*).

Strictly speaking, there is no gulf of evaluation here. Although not shown in the figure, the insertion point is present. After entering 980, the insertion point is either after the 0 in the first field, if focus did not advance, or at the beginning of the next field, if focus advanced. So the system does indeed “provide a physical

**FIGURE 3.45**

Inconsistent focus advancement keeps the user on guard. “What do I do next?”

representation that can be perceived and that is directly interpretable in terms of the intentions and expectations of the person” (Norman, 1988, p. 51). That’s not good enough. The user’s attention is on the keyboard while the physical presentation is on the system’s display. The disconnect is small, but nevertheless, a shift in the user’s attention is required.

The absence of expectations keeps the user on guard. The user is often never quite sure what to do or what to expect. The result is a slight increase in the attention demanded during interaction, which produces a slight decrease in transparency. Instead of engaging in the task, attention is diverted to the needs of the computer. The user is like a wood carver who sharpens tools rather than creates works of art.

Where the consequences of errors are small, such as an extra button click or a gaze shift, errors tend to linger. For the most part, these errors aren’t on anyone’s radar. The programmers who build the applications have bigger problems to focus on, like working on their checklist of new features to add to version 2.0 of the application before an impending deadline.<sup>22</sup> The little errors persist. Often, programmers’ discretion rules the day (Cooper, 1999, p. 47). An interaction scenario that makes sense to the programmer is likely to percolate through to the final product, particularly if it is just a simple thing like focus advancement. Do programmers ever discuss the nuances of focus advancement in building a GUI? Perhaps. But was the discussion framed in terms of the impact on the attention or gaze shifts imposed on the user? Not likely.

Each time a user shifts his or her attention (e.g., from the keyboard to the display and back), the cost is two gaze shifts. Each gaze shift, or saccade, takes from 70 to 700 ms (Card et al., 1983, p. 28).<sup>23</sup> These little bits of interaction add up. They are the fine-grained details—the microstructures and microstrategies used by, or imposed on, the user. “Microstrategies focus on what designers would regard as the mundane aspects of interface design; the ways in which subtle features of interactive technology influence the ways in which users perform tasks” (W. D. Gray and Boehm-Davis, 2000, p. 322). Designers might view these fine-grained details as a mundane sidebar to the bigger goal, but the reality is different. Details are everything. User experiences exist as collections of microstrategies. Whether booking a vacation online or just hanging out with friends on a social networking site, big actions are collections of little actions. To the extent possible, user actions form the experience, our experience. It is unfortunate that they often exist simply to serve the needs of the computer or application.

---

<sup>22</sup>The reader who detects a modicum of sarcasm here is referred to Cooper (1999, 47–48 and elsewhere) for a full frontal assault on the insidious nature of feature bloat in software applications. The reference to version 2.0 of a nameless application is in deference to Johnson’s second edition of his successful book where the same tone appears in the title: *GUI Bloopers 2.0*. For a more sober and academic look at software bloat, feature creep, and the like, see McGrenere and Moore (2000).

<sup>23</sup>An eye movement involves both a saccade and fixation. A saccade—the actual movement of the eye—is fast, about 30 ms. Fixations takes longer as they involve perceiving the new stimulus and cognitive processing of the stimulus.

Another reason little errors tend to linger is that they are often deemed *user errors*, not design, programming, or system errors. These errors, like most, are more correctly called *design-induced errors* (Casey, 2006, p. 12). They occur “when designers of products, systems, or services fail to account for the characteristics and capabilities of people and the vagaries of human behavior” (Casey, 1998, p. 11). We should all do a little better.

Figure 3.46 illustrates a tradeoff between the cost of errors and the frequency of errors. There is no solid ground here, so it’s just a sketch. The four errors described above are shown. The claim is that high-cost errors occur with low frequency. They receive a lot of attention and they get dealt with. As systems mature and the big errors get fixed, designers shift their efforts to fixing less costly errors, like the CAPS\_LOCK design-induced error, or consistently implementing velocity-controlled scrolling. Over time, more and more systems include reasonable and appropriate implementations of these interactions. Divergence in the implementations diminishes and, taken as a whole, there is an industry-wide coalescing toward the same consistent implementation (e.g., a popup alert for CAPS\_LOCK). The ground is set for user expectation to take hold.

Of the errors noted in Figure 3.46, discard changes is ancient history (in computing terms), CAPS\_LOCK is still a problem but is improving, scrolling frenzy is much more controlled in new applications, and focus uncertainty is, well, a mess. The cost is minor, but the error happens frequently.

In many ways, the little errors are the most interesting, because they slip past designers and programmers. A little self-observation and reflection goes a long way here. Observe little errors that you encounter. What were you trying to do? Did it work the first time, just as expected? Small interactions are revealing. What were your hands and eyes doing? Were your interactions quick and natural, or were there unnecessary or awkward steps? Could a slight reworking of the interaction help? Could an attention shift be averted with the judicious use of auditory or tactile feedback? Is there a “ready for input” auditory signal that could sound when an input field receives focus? Could this reduce the need for an attention shift? Would this improve user performance? Would it improve the user experience? Would users like it, or would it be annoying? The little possibilities add up. Think of them as opportunities for empirical research in HCI.

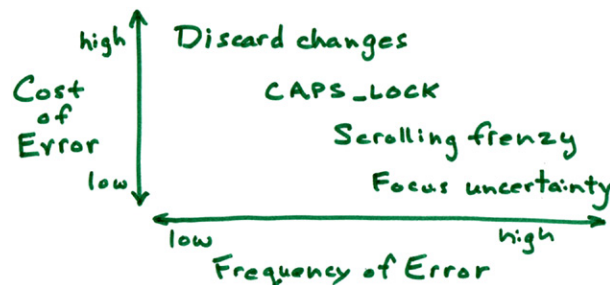


FIGURE 3.46

Trade-off between the cost of errors and the frequency of errors.

## STUDENT EXERCISES

- 3-1.** Conduct a small experiment on icon recognition, as follows. Find 15 computer users (participants) and enquire as to their computer experience. Find users with a range of computing experience (e.g., less than 10 hours per week, more than 25 hours per week. Preferably, find an equal number of participants for each experience level.) Prepare a handout sheet with the six toolbar buttons (soft controls) seen in [Figure 3.3a](#). The general idea is shown below:







**Icon Recognition**

Participant initials: \_\_\_\_\_

Weekly computer usage:

☐ <10 hrs    ☐ 10-25 hrs    ☐ >25 hrs

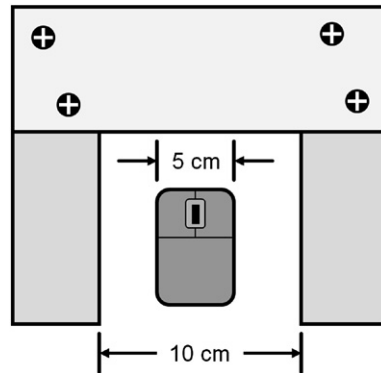
Please indicate the purpose/meaning of the following icons:

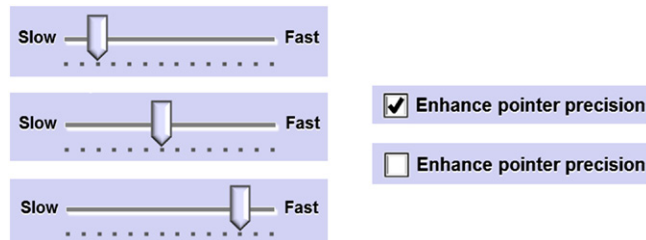
Ask each participant to identify the purpose of the buttons. Record the responses. Write a brief report on the findings, showing and discussing the responses for participants overall and for participants by button and by experience level.

- 3-2.** Conduct a small experiment on panning and zooming using Google Street View. Devise three tasks and measure users' performance using different input methods (e.g., keyboard versus mouse or touchpad versus mouse). As an example task, see [Figure 3.9a](#). For this example, a reasonable task is to pan and zoom to the clock tower (right side of image), determine the time on the clock, then pan and zoom back to the starting position. User performance may be measured in a variety of ways, such as task completion time, accuracy in the final scene position, number of steps or corrective actions, etc. Use five participants. Write a brief report on your observations and measurements.
- 3-3.** Conduct a small experiment to measure the CD gain of the mouse and cursor on a computer system. Build an apparatus, perhaps using wooden blocks, with an opening for a mouse, as below:



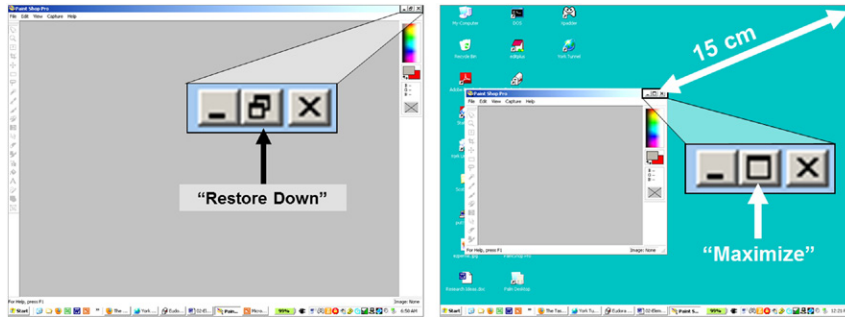


Measure the width of the opening and the mouse, as shown above, and determine the span of mouse movement in the opening. Set the pointer speed in the control panel to the center of the slider position (see below). Position the system pointer (cursor) near the left side of the display and position the mouse on the left side of the opening. Move the mouse at a nominal speed across the opening. Measure and record the amount of pointer movement on the display. Repeat five times. Repeat the above exercise for slow and fast mouse movement. Also repeat for two other pointer speed settings in the control panel (below left) and for two pointer precision settings (below right). Tailor according to the control panel options on the computer system used.



The total number of measurements is 90 (five repetitions  $\times$  three mouse movement speeds  $\times$  three pointer speed settings  $\times$  two enhanced pointer precision settings). Write a brief report on your observations and measurements.

- 3-4. Conduct a small experiment investigating speed and accuracy in cursor positioning as a function of CD gain. Find five computer users (participants). Prepare a computer system as follows. Set the pointer speed in the system control panel to the middle position (see exercise 3-3 above). Open any application in full screen mode (below left). Click the Restore Down button. Position the window by dragging the title bar so that the button (now Maximize) is about 15 cm away from the initial position (below right).



Ask the participants to use the mouse (or other pointing device) to alternately restore down and maximize the window ten times. Measure and record the time to do the task. Observe and record behaviors (e.g., miss-selections) that represent accuracy. Repeat using a lower pointer speed setting and again with a higher setting. Write a brief report on your observations and measurements.

- 3-5. Conduct a small experiment on mode switching, as follows. Find five computer users (participants) who are familiar with Microsoft PowerPoint. Seat them in front of a computer system with a set of slides open in PowerPoint in Slide View mode (aka Normal mode). Have each participant do the following two tasks. First, ask them to use the system's pointing device to switch modes (e.g., to Outline mode or to Slide Sorter mode). Record the time to do the task and whether they used the buttons or menu method (buttons method = soft buttons on the display; menu method = View pull-down menu or ribbon). Return the mode to Slide/Normal. Second, ask them to again change the mode, but this time tell them to use another method, without telling them what the other method is. Write a brief report on your observations.
- 3-6. Conduct a small experiment on mode switching, as follows. Find five computer users (participants) who are familiar with Microsoft Word. Open a text document and switch the mode to Overtyping (see Figure 3.25b and related discussion).<sup>24</sup> Do this without the participants seeing what you did. Ask each participant to insert a word somewhere in the document. Write a brief report on your observations.
- 3-7. Many companies use the World Wide Web to sell their products (e.g., [amazon.com](http://amazon.com)). To assist users in making purchases, these sites often use a metaphor borrowed from the user experience in a grocery store. What is this metaphor? With reference to at least two such websites, discuss the implementation of the metaphor. Describe the metaphor's presentation and the

<sup>24</sup>For Office 2007 or Office 2010, Insert mode must first be enabled. Go to File → Options → Advanced → Editing Options and check "Use the Insert key to control overtype mode."

actions supported. Use screen snapshots or other aides to support the analysis. Present your findings in a brief report or in a slide show presentation.

- 3-8. Consider the trade-off between the cost of errors and the frequency of errors shown in [Figure 3.46](#). The examples are from desktop computing. For each of the four errors, identify a corresponding error from mobile computing using a touchscreen device such as a smartphone or tablet computer. “Corresponding,” in this sense, refers to the level of severity in the error, not to the interaction per se. Position the errors in a similar chart and develop a discussion to justify the choice of error and its placement in the chart. Present your findings in a brief report or in a slide show presentation.