CHAPTER 8 Input Devices, Interaction Techniques, and Interaction Tasks

This is the first of three chapters on designing and implementing graphical user-computer interfaces. As computers become cheaper and more powerful, the major bottleneck for further progress is not so much the need for better hardware or software as the need for better communication between the computer and the human. For that reason, techniques for developing high-quality user interfaces are moving to the forefront in computer science and are becoming the "last frontier" in providing computing to a wide variety of users—as other aspects of technology continue to improve, but the human users remain the same.

Interest in the quality of user-computer interfaces is a recent part of the formal study of computers. The emphasis until the early 1980s was on optimizing two scarce hardware resources, computer time and memory. Program efficiency was the highest goal. With today's plummeting hardware costs and powerful graphics-oriented personal computing environments (as discussed in Chapter **XXX_1_XXX**), the focus turns to optimizing user efficiency rather than computer efficiency. Thus, although many of the ideas presented in this chapter require additional CPU cycles and memory space, the potential rewards in user productivity and satisfaction well outweigh the modest additional cost of these resources.

The quality of the user interface often determines whether users enjoy or despise a system, whether the designers of the system are praised or damned, whether a system succeeds or fails in the market. Indeed, in such critical applications as air-traffic control and nuclearpower-plant monitoring, a poor user interface can contribute to and even cause accidents of catastrophic proportions.

The desktop user-interface metaphor, with its windows, icons, and pull-down menus, all making heavy use of raster graphics, is popular because it is easy to learn and requires little typing skill. Most users of such systems are not computer programmers and have little sympathy for the old-style difficult-to-learn keyboard-oriented command-language interfaces that many programmers take for granted. The designer of an interactive graphics application must be sensitive to users' desire for easy-to-learn yet powerful interfaces.

In this chapter, we discuss the three basic low-level elements of user interfaces: input devices, interaction techniques, and interaction tasks. Input devices were introduced in Chapters **XXX_2 and 4_XXX**: here we elaborate on their use. An interaction technique is a way of using an input device to enter a particular type of value into the computer (such as a pull-down menu, slider, or text area), whereas interaction tasks classify the fundamental types of information entered with the interaction techniques. Interaction techniques are the primitive building blocks from which a user interface is crafted.

In Chapter 9, we discuss the issues involved in putting together the building blocks into a complete user-interface design. The emphasis is on a top-down design approach; first, design objectives are identified, and the design is then developed through a stepwise refinement process. The pros and cons of various dialogue styles—such as direct manipulation, command language, and form fill-in—are discussed, and design guidelines, the dos and don'ts of interface design, are described and illustrated with various positive and negative examples. *Human-computer interaction* is now a field in its own right, and many of the topics in Chapters 8 and 9 are covered in much greater depth elsewhere; see the texts by Baecker and Buxton[BAEC95A], Hutchins, Hollan, and Norman [HUTC86], Mayhew[MAYH99A], Norman [NORM88], Preece et al.[PREE94A], Rubenstein and Hersh [RUBE84], and

Shneiderman[SHNE97A]; the reference books by Boff et al.[BOFF86A], Helander[HELA88A], and Salvendy[SALV97A]; and the survey by Foley, Wallace, and Chan [FOLE84].

Chapter 10 treats user-interface software. It is one thing to design graphic user interfaces that are easy to learn and fast to use; it is quite another to implement them. Having the right software tools and techniques is of critical importance. This chapter reviews the input-handling capabilities of graphics systems and the internal structures and implementation strategies of window managers, a critical element in many high-quality user interfaces, are described. Finally, the key concepts of toolkits, user-interface management systems (UIMSs), interface builders, and other types of user interface software tools are presented. High-level user interface tools provide a means for interface designers and implementors quickly to develop, try out, and modify their interface concepts, and thus decrease the cost of the essential testing and refinement steps in user-interface development.

We focus in this chapter on input devices—those pieces of hardware by which a user enters information into a computer system. Input devices for the earliest computers were switches and knobs, jumper wires placed in patch boards, and punched cards. These were followed by the teletype, the text-only forerunner of today's interactive terminals. The mouse and keyboard now predominate, but a wide variety of input devices can be used. We have already discussed many such devices in Chapter **XXX_4_XXX**. In this chapter, we introduce additional devices, and discuss reasons for preferring one device over another. In Section **XXX_8.1.6_XXX**, we describe input devices oriented specifically toward 3D interaction. We make use of the logical device categories of locator, keyboard, choice, valuator, and pick used in Chapter **XXX_4_XXX** and in various device-independent graphics subroutine packages. More information on input devices can be found in[JACO96A].

An *interaction task* is the entry of a unit of information by the user. Basic interaction tasks are *position*, *text*, *select*, and *quantify*. The unit of information that is input in a position interaction task is of course a position; the text task yields a text string; the select task yields an object identification; and the quantify task yields a numeric value. A designer begins with the interaction tasks necessary for a particular application. For each such task, the designer chooses an appropriate interaction device and interaction technique. Many different *interaction techniques* can be used for a given interaction task, and there may be several different ways of using the same device to perform the same task. For instance, a selection task can be carried out by using a mouse to select items from a menu, using a keyboard to enter the name of the selection, pressing a function key, circling the desired command with the mouse, or even writing the name of the command with the mouse. Similarly, a single device can be used for different tasks: A mouse is often used for both positioning and selecting.

XXX_If drop logical devices, this paragraph should be deleted_XXX Interaction tasks are distinct from the logical input devices discussed in earlier chapters. Interaction tasks are defined by *what* the user accomplishes, whereas logical input devices categorize *how* that task is accomplished by the application program and the graphics system. Interaction tasks are user-centered, whereas logical input devices are a programmer and graphics-system concept.

By analogy with a natural language, single actions with input devices are similar to the individual letters of the alphabet from which words are formed. The sequence of input-device actions that makes up an interaction technique is analogous to the sequence of letters that makes up a word. A word is a unit of meaning; just as several interaction techniques can be used to carry out the same interaction task, so too words that are synonyms convey the same meaning. An interactive dialogue is made up of interaction-task sequences, just as a sentence is constructed from word sequences.

Interaction Hardware

Here, we introduce some interaction devices not covered in Section **XXX_4.6_XXX**, elaborate on how they work, and discuss the advantages and disadvantages of various devices.

The presentation is organized around the logical-device categorization of Section **XXX_4.6_XXX**, and can be thought of as a more detailed continuation of that section. **XXX_Last sentence depends on ch. 4, else delete it_XXX**

The advantages and disadvantages of various interaction devices can be discussed on three levels: device, task, and dialogue (i.e., sequence of several interaction tasks). The *device level* centers on the hardware characteristics *per se*, and does not deal with aspects of the device’s use controlled by software. At the device level, for example, we note that one mouse shape may be more comfortable to hold than another, and that a data tablet takes up more space than a joystick.

At the *task level*, we might compare interaction techniques using different devices for the same task. Thus, we might assert that experienced users can often enter commands more quickly via function keys or a keyboard than via menu selection, or that users can pick displayed objects more quickly using a mouse than they can using a joystick or cursor control keys.

At the *dialogue level*, we consider not just individual interaction tasks, but also sequences of such tasks. Hand movements between devices take time: Although the positioning task is generally faster with a mouse than with cursor-control keys, cursor control keys may be faster than a mouse *if* the user’s hands are already on the keyboard and will need to be on the key-board for the next task in sequence after the cursor is repositioned. Dialogue-level issues are discussed in Chapter 9, where we deal with constructing complete user interfaces from the building blocks introduced in this chapter. Much confusion can be avoided when we think about devices if we keep these three levels in mind.

Important considerations at the device level, discussed in this section, are the device foot-prints (the *footprint* of a piece of equipment is the work area it occupies), operator fatigue, and device resolution. Other important device issues—such as cost, reliability, and maintainability—change too quickly with technological innovation to be discussed here. Also omitted are the details of connecting devices to computers; by far the most common means is the serial asynchronous RS-232 terminal interface, which makes hardware interfacing simple. However, logical protocols for interfacing different devices, even of the same general kind, differ widely; input devices are still far less plug-compatible than, for example, the MIDI devices used for electronic musical instruments.

Today, user-computer dialogues are rather one-sided. The amount of information or bandwidth that is communicated from computer to user, particularly with computer graphics, is typically far greater than the bandwidth from user to computer, This is partly due to human abilities: we can receive visual images with very high bandwidth, but we are not very good at generating them. New input devices are helping to redress this imbalance, but, compared with the great strides made in computer graphics, there has been less progress in this area to date. To be effective, input devices must be tailored to match the characteristics and abilities of humans. It is, after all, much easier to modify a computer device than the human user; the devices must fit the users, not the other way around. The principal means of computer input from users today is through the hands, for example keyboards, mice, gloves, and 3D trackers. Other limb movements are also considered, along with voice, and, finally, eye movements and other physiological measurements that may be used as input in the future.

Locator Devices

It is useful to classify locator devices according to several independent characteristics, including the property and the number of dimensions sensed[BUXT83A], and ergonomic differences between seemingly similar devices[MACK90A, BLES90A].

Absolute devices, such as a data tablet or touch panel, have a frame of reference, or origin, and report positions with respect to that origin. *Relative* devices—such as mice, trackballs, and velocity-control joysticks—have no absolute origin and report only changes from their former position. A relative device can be used to specify an arbitrarily large change in position: A user can move a mouse along the desk top, lift it up and place it back at its initial starting position, and move it again. A data tablet can be programmed to behave as a relative device: The first (x, y) coordinate position read after the pen goes from "far" to "near" state (i.e., close to the tablet) is subtracted from all subsequently read coordinates to yield only the change in x and y, which is added to the previous (x, y) position. This process is continued until the pen again goes to "far" state.

Relative devices cannot be used readily for digitizing drawings, whereas absolute devices can be. The advantage of a relative device is that the application program can reposition the cursor anywhere on the screen.

With a *direct* device—such as a light pen or touch screen—the user points directly at the screen with a finger or stylus; with an *indirect* device—such as a tablet, mouse, or joystick—the user moves a cursor on the screen using a device located somewhere else. New forms of eye-hand coordination must be learned for the latter; the proliferation of computer games, however, has created an environment in which most casual computer users have already learned these skills. However, direct pointing can cause arm fatigue, especially among casual users.

A *continuous* device is one in which a smooth hand motion can create a smooth cursor motion. Tablets, joysticks, and mice are all continuous devices, whereas cursor-control keys are *discrete* devices. Continuous devices typically allow more natural, easier, and faster cursor movement than do discrete devices. Most continuous devices also permit easier movement in arbitrary directions than do cursor control keys.

The type of motion is another characteristic of locator devices. For example, a mouse measures *linear* motion (in two dimensions); a knob, *rotary*. Devices can also differ in the physical property sensed. A mouse measures *position*; it moves when it is pushed. An isometric joystick remains nearly stationary and simply reports the pressure being applied to it;

it measures *force*. For a rotary device, the corresponding properties are angle and torque. Devices might operate in *one, two, or three dimensions*. A mouse measures two linear dimensions; a knob measure one (angular) dimension; and a Polhemus tracker measures three linear dimensions and three angular. The device can use *position* or *rate control*. Moving a mouse changes the position of the cursor. A joystick, however, can be used to control cursor position directly or it can control the rate of speed at which the cursor moves. Since its total range of motion is typically fairly small compared to a display screen, position control is imprecise. However, rate control requires a more complex relationship between the user's action and the result on the display and is therefore more difficult to operate. Finally, motion in several dimensions may be *integral* or *separable*. A mouse allows easy, coordinated movement across both dimensions simultaneously (integral); while a pair of knobs (as in an Etch-a-Sketch **XXX_(TM)?_XXX**toy) does not (separable)[JACO94A].

Speed of cursor positioning with a continuous device is affected by the *control-to-display ratio*, commonly called the C/D ratio [CHAP72]; it is the ratio between the movement of the input device and the corresponding movement of the object it controls. For example, if a mouse (the control) must be moved one inch on the desk in order to move a cursor two inches on the screen (the display), the device has a 1:2 control-display ratio. A large ratio is good for accurate positioning, but makes rapid movements tedious; a small ratio is good for speed but not accuracy and requires less desk space. For a relative positioning device, the ratio need not be constant, but can be changed by an accelerator as a function of control-movement speed. Rapid movements indicate the user is making a gross hand movement, so a small ratio is used; as the speed decreases, the C/D ratio is increased. This allows more efficient use of desk space, but it can disturb the otherwise straightforward physical relationship between mouse movement and cursor movement[JELL90A]. Of course, with a direct input device, such as a touch screen, the C/D ratio is always 1:1.

Fitts' Law provides a way to predict the speed with which the user can move his or her hand to a target, and is a key foundation in input design. [FITT54][CARD83A] It shows that the time required to move is based on the distance to be moved *and* the size of the destination target. The time is proportional to the logarithm of of the distance divided by the target width. This means there is a tradeoff between distance and target width: it takes as much additional time to reach a target that is twice as far away as it does to reach one that is half as large. Different manual input devices give rise to different proportionality constants in the equation. Some thus give better overall performance, and others, better performance either for long moves or short moves, but the tradeoff between distance and target size remains.

Precise positioning is difficult with direct devices, if the arm is unsupported and extended toward the screen. Try writing your name on a blackboard in this pose, and compare the result to your normal signature. This problem can be mitigated if the screen is angled close to horizontal. Indirect devices, on the other hand, allow the heel of the hand to rest on a support, so that the fine motor control of the fingers can be used more effectively. Not all continuous indirect devices are equally satisfactory for drawing, however. Try writing your name with a joystick, a mouse, and a tablet pen stylus. Using the stylus is fastest, and the result is most pleasing.

Today, the mouse is the most widely used device for inputting 2D positions, but it was not the first such device developed. It supplanted devices such as the joystick, trackball, lightpen, and arrow keys and, in an early example of the application of HCI research to practice, was demonstrated to give fastest performance and closest approximation to Fitts' Law compared to alternative devices at the time[CARD78A]. Despite its popularity, some specific, constrained situations call for alternative devices. For example, the Navy uses trackballs instead of mice on shipboard, because the rolling of the ship makes it difficult to keep a mouse in place. Laptop computers use small trackballs, touch-sensitive pads, or tiny joysticks because they are more compact, and pocket computers typically use a stylus and touchscreen. A variant of the mouse contains a sensor that also measures when the user's hand is touching the mouse, to provide additional, passive input from the user[HINC99A].

Other interesting positioning devices use other parts of the body. The *mole* is an experimental foot-operated locator device that uses a footrest suspended on two sets of pivots[PEAR86A]. While control is less precise than a manually-operated mouse, it leaves the hands free for additional operations. The Personics *headmouse* used a head-mounted set of three microphones to measure the distance to a sound source, translating small rotational movements of the head into cursor movements, though this often requires the neck to be held in an awkward fixed position. Another use of head movement is to perform a function more akin to the use of head movement in the natural world—panning and zooming over a display[HIX95A]

While the main role of the eye in computer graphics is to receive output from the computer. *Eye trackers* can determine where the eye is pointing and hence can cause a cursor to move or the object pointed at to be selected [BOLT80; BOLT84; WARE87][BOLT81A]. Because people move their eyes rapidly and almost unconsciously, this can be very effective, but it requires careful design of appropriate interaction techniques to avoid annoying the user with unwanted responses to his actions, the ‘‘Midas Touch’’ problem[JACO91A]. Figure 8.1 shows an eye tracker in use, observing the user's eye through the mirror located under the display screen. Today these devices are less stable and more expensive than more traditional devices, and thus would normally be considered for only hands-free applications, though the technology is steadily improving. Finally, the 3D positioning devices discussed in Section **XXX_8.1.6_XXX** can also be used for 2D positioning by ignoring one of their outputs.

Keyboard Devices

The well-known QWERTY keyboard has been with us for many years. It is ironic that this keyboard was originally designed to *slow down* typists, so that the typewriter hammers would not be so likely to jam. Studies have shown that the newer Dvorak keyboard [DVOR43], which places vowels and other high-frequency characters under the home positions of the fingers, is somewhat faster than is the QWERTY design [GREE87]. It has not been widely accepted. Alphabetically organized keyboards are sometimes used when many of the users are nontypists. But more and more people are being exposed to QWERTY keyboards, and experiments have shown no advantage of alphabetic over QWERTY keyboards [HIRS70]. In recent years, the chief force serving to displace the keyboard has been the shrinking size of computers, with laptops, notebooks, palmtops, and personal digital assistants. The typewriter keyboard is becoming the largest component of such pocket-sized devices, and often the main component standing in the way of reducing its overall size.

The *chord keyboard* has five keys similar to piano keys, and is operated with one hand, by pressing one or more keys simultaneously to "play a chord." With five keys, 31 different chords can be played. Learning to use a chord keyboard (and other similar stenographer style keyboards) takes longer than learning the QWERTY keyboard, but skilled users can type quite rapidly, leaving the second hand free for other tasks. This increased training time means, however, that such keyboards are not suitable substitutes for general use of the standard alphanumeric keyboard. Again, as computers become smaller, the benefit of a keyboard that allows touch typing with only five keys may come to outweigh the additional difficulty of learning the chords.

Other keyboard-oriented considerations, involving not hardware but software design, are arranging for a user to enter frequently used punctuation or correction characters without needing simultaneously to press the control or shift keys, and assigning dangerous actions (such as delete) to keys that are distant from other frequently used keys.

Valuator Devices

A hardware *potentiometer,* like the volume control on a radio, can be used to input a scalar value. Some valuators are *bounded,* like the radio volume control—the dial can be turned only so far before a stop is reached that prevents further turning. A bounded valuator inputs an absolute quantity. A continuous-turn potentiometer, on the other hand, can be turned an *unbounded* number of times in either direction. Given an initial value, the unbounded potentiometer can be used to return absolute values; otherwise, the returned values are treated as relative values. The provision of some sort of echo enables the user to determine what relative or absolute value is currently being specified. The linear or slide potentiometer is inherently bounded. Some systems use a single hardware potentiometer that can be assigned to different input values at different times under software control. The issue of C/D ratio, discussed in the context of positioning devices, also arises in the use of slide and rotary potentiometers to input values.

Choice Devices

Function keys are a common choice device. These may be permanently labeled, special-purpose pushbuttons, or they may have labels that can be changed under computer control. An extreme, but effective use of permanently labeled function keys is found in the cash registers of fast-food restaurants, where a large array of special-purpose function keys is provided, one for every possible item that can be purchased. Variable labels for function keys can be provided by placing the keys near the edge of the display and using the adjacent portion of the display to indicate the key labels, or by providing small alphanumeric LED or LCD displays above each key. Their placement affects their usability: keys mounted on the CRT bezel are harder to use than are keys mounted in the keyboard or in a nearby separate unit.

Other Devices

Here we discuss some of the less common, and in some cases experimental, 2D interaction devices. Voice recognizers, which are useful because they free the user's hands for other uses, apply a pattern-recognition approach to the waveforms created when we speak a word. The waveform is typically separated into a number of different frequency bands, and the variation over time of the magnitude of the waveform. in each band forms the basis for the pattern matching. However, mistakes can occur in the pattern matching, so it is especially important that an application using a recognizer provide convenient correction capabilities.

Voice recognizers differ in whether or not they must be trained to recognize the waveforms of a particular speaker, and whether they can recognize connected speech as opposed to single words or phrases. Speaker-independent recognizers have very limited vocabularies—typically, they include only the ten digits and 50 to 100 words. Some discrete-word recognizers can recognize vocabularies of thousands of different words after appropriate training. But if the user has a cold, the recognizer must be retrained. The user of a discrete-word recognizer must pause for a fraction of a second after each word to cue the system that a word end has occurred. The more difficult task of recognizing connected speech from a limited vocabulary can now be performed by off-the-shelf hardware and software, but with somewhat less accuracy. As the vocabulary becomes larger, however, artificial-intelligence techniques are needed to exploit the context and meaning of a sequence of sentences to remove ambiguity. A few systems with vocabularies of 20,000 or more words can recognize sentences such as "Write Mrs. Wright a letter right now!"

Voice synthesizers create waveforms that approximate, with varying degrees of realism, spoken words[KLAT87A, VAN95A]. The simplest synthesizers use *phonemes*, the basic sound units that form words. This approach creates an artificial-sounding, inflection-free voice. More sophisticated phoneme-based systems add inflections. Other systems actually play back digitized spoken words or phrases. They sound realistic, but require more memory to store the digitized speech.

Speech is best used to augment rather than to replace visual feedback, and is most effective when used sparingly. For instance, a training application could show a student a graphic animation of some process, along with a voice narration describing what is being seen. See [SIMP87] for additional guidelines for the effective application of speech recognition and generation in user-computer interfaces, and[SCHM94A] for an introduction to speech interfaces, and[RABI93A] for speech recognition technology.

The data tablet has been extended in several ways. Many years ago, Herot and Negroponte used an experimental pressure-sensitive stylus [HERO76]: High pressure and a slow drawing speed implied that the user was drawing a line with deliberation, in which case the line was recorded exactly as drawn; low pressure and fast speed implied that the line was being drawn quickly, in which case a straight line connecting the endpoints was recorded. Some commercially available tablets sense not only stylus pressure but orientation as well. The resulting 5 degrees of freedom reported by the tablet can be used in various creative ways. For example, Bleser, Sibert, and McGee implemented the GWPaint system to simulate various artist’s tools, such as an italic pen, that are sensitive to pressure and orientation [BLES88a]. Figure 8.2 shows the artistic creativity thus afforded.

An experimental touch tablet, developed by Buxton and colleagues, can sense multiple finger positions simultaneously, and can also sense the area covered at each point of contact [LEE85a]. The device is essentially a type of touch panel, but is used as a tablet on the work surface, not as a touch panel mounted over the screen. The device can be used in a rich variety of ways [BUXT85]. Different finger pressures correlate with the area covered at a point of contact, and are used to signal user commands: a light pressure causes a cursor to appear and to track finger movement; increased pressure is used, like a button-push on a mouse or puck, to begin feedback such as dragging of an object; decreased pressure causes the dragging to stop.

Emerging Devices

A video camera and frame grabber can be used as an input device in interesting ways. For example the computer can determine relatively easily from camera input whether the user is still sitting in the chair, facing toward the computer or not, using the telephone, or talking to another person in the room. With more sophisticated pattern recognition for interpreting the images, the computer might be able to read the user's facial expression or body posture or determine what objects the user is holding and manipulating.

A camera can also be used to identify objects in the real world, so that when the user manipulates a real object, the computer can make a corresponding update to the object in the computer. Other techniques can also be used for identifying the objects, such as bar codes, color codes, and radio frequency identification (RFID) tags. *Tangible user interfaces* use these approaches to allow the user to interact with the computer by manipulating real objects[ISHI97A, WANT99A, FITZ95A]. A camera can also view the objects on a desk and allow them to be used as computer inputs[NEWM92A]. It can also be located behind a rear projection screen, aimed toward the screen, to watch the user's hands or a stylus touching the front of the screen[MATS97A] and other ways. Figure 8.3 shows the metaDESK[ISHI97A], on which the user can interact with the computer by moving the instruments, objects, and lenses around on the physical desk.

Looking toward future input devices, passive measurement of various physiological characteristics of the user can be input and used to modify the computer's dialogue with its user. Blood pressure, heart rate[ROWE98A], respiration rate, eye pupil diameter, and galvanic skin response (the electrical resistance of the skin) can all be measured easily, though accurate instantaneous interpretation within a user-computer dialogue is an open question. Some input can also be obtained from electro-encephalogram (EEG) signals, though these are even more difficult to interpret automatically. In fact, in the more distant future, the final frontier in user input and output devices may be to measure and stimulate neurons directly, rather than relying on the body’s transducers. The computer would be more like a mental prosthesis, where the explicit input and output tasks disappear, and the communication is direct, from brain to computer.

The decreasing costs of input devices and the increasing availability of computer power for pattern recognition are likely to lead to the continuing introduction of novel interaction devices. However, the gestation times for these have been long. Douglas Engelbart invented the mouse in the 1960s[ENGE68A]. It took approximately 10 years before it was found in many other research labs and nearly 20 before it was widely used in applications outside the research world. The input mechanisms in use 20 years from now may spring from devices and approaches that today appear to be impractical laboratory curiosities.

3D Interaction Devices

Some of the 2D interaction devices are readily extended to 3D. Joysticks can have a shaft that twists for a third dimension (see Fig. **XXX_4.38_XXX**). Trackballs can be made to sense rotation about the vertical axis in addition to that about the two horizontal axes. In both cases, however, there is no direct relationship between hand movements with the device and the corresponding movement in 3-space.

The Spaceball (see Color Plate I.14) is a rigid sphere containing strain gauges. The user pushes or pulls the sphere in any direction, providing 3D translation and orientation, like a 3D version of a rate-controlled isometric joystick. In this case, at least the directions of movement correspond to the user’s attempts at moving the rigid sphere, although the hand does not actually move.

A number of devices, on the other hand, can measure actual 3D hand movements. Magnetic trackers, such as the Polhemus 3Space (shown in Fig. 8.4) and Ascension Bird devices sense three-dimensional position and orientation using electromagnetic coupling between three transmitter antennas and three receiver antennas. The transmitter antenna coils, which are at right angles to one another to form a Cartesian coordinate system, are pulsed in turn. The receiver has three similarly arranged receiver antennas; each time a transmitter coil is pulsed, a current is induced in each of the receiver coils. The strength of the current depends both on the distance between the receiver and transmitter and on the relative orientation of the transmitter and receiver coils. The combination of the nine current values induced by the three successive pulses is used to calculate the 3D position and orientation of the receiver. The device functions like the three-dimensional equivalent of a data tablet in that it provides the absolute position of the receiver along three axes in space, along with its orientation, in the form of angles of rotation about the three axes—heading, pitch, and roll. The receiver of the 3D tracker is typically a one-inch plastic cube, which can be held in the hand, or attached to a glove, foot, the user's head (for virtual reality), or to passive props[HINC94A] that the user will manipulate. The Logitech 3D Mouse uses ultrasonic ranging for this same purpose. It typically provides less precision and fewer axes, but is less expensive and more robust in the face of magnetic interference, such as that from a CRT. Zhai and Milgram[ZHAI98A] evaluate and compare different categories of 3D input devices.

Data glove input devices report the configuration of the fingers of the user's hand. As shown in Fig. 8.5, the CyberGlove device is a glove covered with small, lightweight sensors. Each sensor is a short length of fiberoptic cable, with a light-emitting diode (LED) at one end and a phototransistor at the other end. The surface of the cable is roughened in the area where it is to be sensitive to bending. When the cable is flexed, some of the LED's light is lost, so less light is received by the phototransistor. Other glove technologies use mechanical sensors. All of them normally incorporate a 3D position and orientation sensor records hand movements. Wearing such a glove, a user can grasp objects, move and rotate them, and then release them, thus providing very natural interaction in 3D [ZIMM87]. Color Plate I.15 illustrates this concept.

These devices are typically combined for use in a *virtual reality* interface, a completely computer-generated environment with realistic appearance, behavior, and interaction techniques [FOLE87]. A 3D magnetic tracker is used to sense head position and orientation, which then determines the position of the virtual camera, which generates the scene to be displayed in the user's head-mounted display, typically in stereo. The result is the illusion of a realistic, three-dimensional world that surrounds the user wherever he or she looks. Figure 8.6 shows a user wearing a head-mounted display with an attached 3D tracker. The user can also reach out into this world and touch the objects in it, using a second 3D tracker attached to the hand and grasp them using a glove. However, the user will not feel the object when his or her hand touches it. Mechanisms for providing computer-controlled force and tactile feedback are a topic of current research. Figure 8.7 shows one such device, the Phantom[SALI99A].

All these systems work in relatively small volumes—8 to 27 cubic feet. Optical sensors can give even greater freedom of movement. In one system[AZUM94A], an array of tiny flashing infrared LEDs is placed in the ceiling throughout the tracked area, and the user wears a set of small, orthogonally arranged optical sensors that receive the light pulses and determine the user's location. Another approach is to use sensors that observe the user, without requiring him or her to hold or wear anything. Camera-based locator devices offer the promise of doing this, but today are still limited. A single-camera system is limited to its line of sight; more cameras can be added but full coverage of an area may require many cameras and a way to switch among them smoothly. This approach depends upon using image processing to interpret the picture of the user and extract the desired hand or body position. One of the first such systems was Krueger's [KRUE83] sensor for recording hand and finger movements in 2D. A television camera records hand movements; image-processing techniques of contrast-enhancement and edge detection are used to find the outline of the hand and fingers. Different finger positions can be interpreted as commands, and the user can grasp and manipulate objects, as in Color Plate I.17. This technique could be extended to 3D through use of multiple cameras. Maes[MAES95A] shows another example of this approach. Schmandt[SCHM83A] built a system to allow users to manually manipulate and interact with objects in a 3D computer space using a 3D wand. Again a half-silvered mirror was used to project the computer space over the user's hand and the input device.

Today, all of these 3D devices are still limited compared to a mouse or data tablet—in latency, precision, stability, susceptibility to interference, or number of available samples per second.

Device-Level Human Factors

Not all interaction devices of the same type are equivalent from a human-factors point of view (see [BUXT86] for an elaboration of this theme). For instance, mice differ in important ways. First, the physical shapes are different, ranging from a hemisphere to an elongated, low-profile rectangle. Buttons are positioned differently. Buttons on the side or front of a mouse may cause the mouse to move a bit when the buttons are pressed; buttons on the top of a mouse do not have this effect. The mouse is moved through small distances by wrist and finger movements, with the fingers grasping the mouse toward its front. yet the part of the mouse whose position is sensed is often toward the rear, where fine control is least possible. In fact, a small leftward movement of the mouse under the fingertips can include a bit of rotation, so that the rear of the mouse, where the position sensors are, actually moves a bit to the right!

There is great variation among keyboards in design parameters, such as keycap shape, distance between keys, pressure needed to press a key, travel distance for key depression, key bounce, auditory feedback, the feeling of contact when the key is fully depressed, and the placement and size of important keys such as "return" or "enter." Improper choice of parameters can decrease productivity and increase error rates. For example, making the "return" key too small invites errors. Fortunately, keyboards for desktop computers have largely become standardized, but laptop and palmtop computer keyboards still vary considerably.

The tip of a short joystick shaft moves through a short distance, forcing use of a small C/D ratio; if we try to compensate by using a longer joystick shaft, the user cannot rest the heel of her hand on the work surface and thus does not have a steady platform from which to make fine adjustments. Accuracy and speed therefore suffer.

The implication of these device differences is that it is not enough for a user interface designer to specify a particular device class; specific device characteristics must be defined. Unfortunately, not every user interface designer has the luxury of selecting devices; often, the choice has already been made. In some situations, such as a new airplane cockpit or a surgical teleoperator, the user interface designer can choose whatever input devices best facilitate operator performance. In many other situations, such as a software application to be used on standard workstations, the designer can not choose the devices. In this case, the designer does decide which tasks should be assigned to the mouse and which to the keyboard and which interaction techniques should be used for each task. If we begin our design with each interaction task and then select the best interaction device and interaction technique for each one individually, the result may be a poor overall design, with too many different or inconsistent types of devices or dialogues. It is better not to surround the user with many infrequently-used devices, to reduce the time spent switching between devices.

Graphical Interaction Techniques

We consider next the principal interaction techniques in use in today's graphical user interfaces for performing basic interaction tasks using the interaction devices discussed in the previous section. An interaction technique is a way of using a physical input/output device to perform a generic interaction task in a human-computer dialogue. It represents an abstraction of some common class of interactive task, for example, choosing one of several objects shown on a display screen, so it is not bound to a single application.

The basic interaction tasks for interactive graphics are positioning, selecting, entering text, and entering numeric quantities. Some interaction techniques for each are discussed, but there are many more, and new ones continue to be developed. Where possible, the pros and cons of each technique are discussed; remember that a specific interaction technique may be good in some situations and poor in others.

Basic interaction tasks are indivisible; that is, if they were decomposed into smaller units of information, the smaller units would not in themselves be meaningful to the application. With a basic interaction task, the user of an interactive system enters a unit of information that is meaningful in the context of the application. How large or small is such a unit? For instance, does moving a positioning device a small distance enter a unit of information? yes, if the new position is put to some application purpose, such as repositioning an object or specifying the endpoint of a line. No, if the repositioning is just one of a sequence of repositionings as the user moves the cursor to place it on top of a menu item: here, it is the menu choice that is the unit of information.

Position

The position interaction task involves specifying an (x, y) or (x, y, z) position to the application program. This might be done by moving a screen cursor to the desired location and then pushing a button, or typing the desired position's coordinates. The positioning device can be direct or indirect, continuous or discrete, absolute or relative. In addition, cursor-movement commands can also be typed explicitly on a keyboard, as Up, Left, and so on, or the same commands can be spoken to a voice-recognition unit. Furthermore, techniques can be used together—a mouse controlling a cursor can be used for approximate positioning, and arrow keys can be used to move the cursor a single screen unit at a time for precise positioning.

A number of general issues transcend any one interaction technique for positioning. We first discuss the general issues; we introduce specific positioning techniques as illustrations.

Coordinate systems

An important issue in positioning is the coordinate system in which feedback is provided. If a locator device is moved to the right to drag an object, in which direction should the object move? There are at least three possibilities: the object could move along the increasing *x* direction in the screen-coordinate system, along the increasing *x* direction in world coordinates, or along the increasing *x* direction in the object's own coordinate system.

The first alternative, increasing screen-coordinate *x* direction, is the correct choice. For the latter two options, consider that the increasing *x* direction need not in general be along the screen coordinates' *x* axis. For instance, if the viewing transformation includes a 180 **XXX_DEGREE_XXX** rotation, then the world coordinates' *x* axis goes in the opposite direction to the screen coordinates' *x* axis, so that the right-going movement of the locator would cause a left-going movement of the object. Try positioning with this type of feedback by turning your mouse 180 **XXX_DEGREE_XXX**! Such a system would be a gross violation of the human-factors principle of stimulus-response compatibility (S-R compatibility), which states that system responses to user actions must be in the same direction or same orientation, and that the magnitude of the responses should be proportional to the actions. Similar problems can occur if a data tablet is rotated with respect to the screen.

Resolution

The resolution required in a positioning task may vary from one part in a few hundred to one part in millions. Clearly, keyboard typing of an (*x*, *y*) pair can provide unlimited resolution: The typed digit strings can be as long as necessary. What resolution can cursor-movement techniques achieve? The resolution of tablets, mice, and so on is typically as least as great as the 500 to 2000 resolvable units of the display device. By using the window-to-viewport transformation to zoom in on part of the world, it is possible to arrange for one unit of screen resolution to correspond to an arbitrarily small unit of world-coordinate resolution.

Some touch panels are accurate to 1000 units—but the user's finger is about 1/2 inch wide, so how can this accuracy be achieved? Using the first position the finger touches as the final position does not work. The user must be able to drag a cursor around on the screen by moving or rolling his finger while it is in contact with the touch panel. Because the finger obscures the exact position being indicated, the cursor arms can be made longer than normal, or the cursor can be offset from the actual point of contact. These strategies improve the precision attainable with a finger-operated touch screen[SEAR91A]. Nevertheless, the touch panel is not generally recommended for frequent high-resolution positioning tasks.

Grids

An important visual aid in many positioning tasks is a grid superimposed (perhaps at low intensity) on the work area, to help in aligning positions or objects. It can also be useful to force endpoints of primitives to fall on the grid, as though each grid point were surrounded by a gravity field. Gridding helps users to generate drawings with a neat appearance. To enforce gridding, the application program simply rounds locator coordinates to the nearest grid point (in some cases, only if the point is already close to a grid point). Gridding is usually applied in world coordinates. Although grids often are regular and span the entire display, irregular grids, different grids in different areas, as well as rotated grids, are all useful in creating figures and illustrations [BIER86a; FEIN82a].

Feedback

Positioning tasks can be spatial or linguistic. In a *spatial* positioning task, the user knows where the intended position is, in spatial relation to nearby elements, as in drawing a line between two rectangles or centering an object between two others. In a *linguistic* positioning task, the user knows the numeric values of the (*x*, *y*) coordinates of the position. In the former case, the user wants feedback showing the actual position on the screen; in the latter case, the coordinates of the position are needed. If the wrong form of feedback is provided, the user must mentally convert from one form to the other. Both forms of feedback can be provided by displaying both the cursor and its numeric coordinates, as in Fig. 8.8.

Direction preference

Some positioning devices allow unconstrained movement across all dimensions diagonally (integral dimensions). Others allow motion only in one dimension at a time (separable dimensions). For example, a trackball allows integral motion, but certain joysticks and joyswitches give more resistance to movements off the principal axes than they do to those on the axes, forcing separable motion. The latter is useful only if the positioning task itself is generally constrained to horizontal and vertical movements[JACO94A].

Learning time

Learning the eye-hand coordination for indirect methods is essentially the same process as learning to steer a car. Learning time is a common concern but turns out to be a minor issue. Card and colleagues [CARD78] studied the mouse and joystick. They found that, although practice improved both error rates and speed, even the novices’ performance was quite good. For instance, selection time with a mouse (move cursor to target, press button) decreased with extensive practice from 2.2 to 1.7 seconds. It is true, however, that some users find the indirect coordination difficult at first.

One specific type of positioning task is *continuous positioning*, in which a sequence of positions is used to define a curve. The path taken by the locator is approximated by a connected series of very short lines, as shown in Fig. 8.9. So that the appearance of smoothness is maintained, more lines may be used where the radius of curvature is small, or individual dots may be displayed on the cursor's path, or a higher-order curve can be fitted through the points (see Chapter **XXX_11_XXX**).

Precise continuous positioning is easier with a stylus than with a mouse, because the stylus can be controlled precisely with finger muscles, whereas the mouse is controlled primarily with wrist muscles. Digitizing of drawings is difficult with a mouse for the same reason; in addition, the mouse lacks both an absolute frame of reference and a cross-hair. On the other hand, a mouse requires only a small table area and is less expensive than a tablet.

Select—Variable-Sized Set of Choices

The selection task is that of choosing an element from a *choice set*. Typical choice sets are commands, attribute values, object classes, and object instances. For example, the line-style menu in a typical paint program is a set of attribute values, and the object-type (line, circle, rectangle, text, etc.) menu in such programs is a set of object classes. Some interaction techniques can be used to select from any of these four types of choice sets: others are less general. For example, pointing at a visual representation of a set element can serve to select it, no matter what the set type. On the other hand, although function keys often work quite well for selecting from a command, object class, or attribute set, it is difficult to assign a separate function key to each object instance in a drawing, since the size of the choice set is variable, often is large (larger than the number of available function keys), and changes quite rapidly as the user creates and deletes objects.

We use the terms *(relatively) fixed-sized choice set* and *varying-sized choice set*. The first term characterizes command, attribute, and object-class choice sets; the second, object-instance choice sets. The "relatively" modifier recognizes that any of these sets can change as new commands, attributes, or object classes (such as symbols in a drafting system) are defined. But the set size does not change frequently, and usually does not change much. Varying-sized choice sets, on the other hand, can become quite large, and can change frequently.

In this section, we discuss techniques that are particularly well suited to potentially large varying-sized choice sets; these include naming and pointing. In the following section, we discuss selection techniques particularly well suited to (relatively) fixed-sized choice sets. These sets tend to be small, except for the large (but relatively fixed-sized) command sets found in complex applications.

Selecting objects by naming

The user can type the choice’s name. The idea is simple, but what if the user does not know the object’s name, as could easily happen if hundreds of objects are being displayed, or if the user has no reason to know names? Nevertheless, this technique is useful in several situations. First, if the user is likely to know the names of various objects, as a fleet commander would know the names of the fleet’s ships, then referring to them by name is reasonable, and can be faster than pointing, especially if the user might need to scroll through the display to bring the desired object into view. Second, if the display is so cluttered that picking by pointing is difficult *and* if zooming would be distracting, then naming may be a choice of last resort. If clutter is a problem, then a command to turn object names on and off would be useful.

Typing allows us to make multiple selections through wild-card or don’t-care characters, if the choice set elements are named in a meaningful way. Selection by naming is most appropriate for experienced, regular users, rather than for casual, infrequent users.

If naming by typing is necessary, a useful form of feedback is to display, immediately after each keystroke, the list (or partial list, if the full list is too long) of names in the selection set matching the sequence of characters typed so far. This can help the user to remember just how the name is spelled, if he has recalled the first few characters. As soon as an unambiguous match has been typed, the correct name can be automatically highlighted on the list. Alternatively, the name can be automatically completed as soon as an unambiguous match has been typed. This technique, called *autocompletion*, is sometimes disconcerting to new users, so caution is advisable. A separate strategy for name typing is spelling correction (sometimes called *Do What I Mean*, or DWIM)[TEIT79A]. If the typed name does not match one known to the system, other names that are close to the typed name can be presented to the user as alternatives. Determining closeness can be as simple as searching for single-character errors, or can include multiple-character and missing-character errors.

With a voice recognizer, the user can speak, rather than type, a name, abbreviation, or code. Voice input is a simple way to distinguish commands from data: Commands are entered by voice, the data are entered by keyboard or other means. In a keyboard environment, this eliminates the need for special characters or modes to distinguish data and commands.

Selecting objects by pointing

Any of the pointing techniques mentioned in the introduction to Section **XXX_8.2_XXX** can be used to select an object, by first pointing and then indicating (typically via a button-push) that the desired object is being pointed at. But what if the object has multiple levels of hierarchy, as did the robot of Chapter **XXX_7_XXX**? If the cursor is over the robot's hand, it is not clear whether the user is pointing at the hand, the arm, or the entire robot. Commands like Select-robot and Select-arm can be used to specify the level of hierarchy. On the other hand, if the level at which the user works changes infrequently, the user will be able to work faster with a separate command, such as Set_selection_level, used to change the level of hierarchy.

A different approach is needed if the number of hierarchical levels is unknown to the system designer and is potentially large (as in a drafting system, where symbols are made up of graphics primitives and other symbols). At least two user commands are required: Up_hierarchy and Down_hierarchy. When the user selects something, the system highlights the lowest-level object seen. If this is what he desired, the user can proceed. If not, the user issues the first command: Up_hierarchy. The entire first-level object of which the detected object is a part is highlighted. If this is not what the user wants, he travels up again and still more of the picture is highlighted. If he travels too far up the hierarchy, he reverses direction with the Down_hierarchy command. In addition, a Return_to_lowest_level command can be useful in deep hierarchies, as can a hierarchy diagram in another window, showing where in the hierarchy the current selection is located. The state diagram of Fig. 8.10 shows one approach to hierarchical selection. Alternatively, a single command, say Move_up_hierarchy, can skip back to the originally selected leaf node after the root node is reached.

Some text editors use a character-word-sentence-paragraph hierarchy. In the Xerox Star and many succeeding text editors, for instance, the user selects a character by positioning the screen cursor on the character and clicking the mouse button once. To choose the word rather than the character, the user double clicks. Further moves up the hierarchy are accomplished by additional rapid clicks.

Select—Relatively Fixed-Sized Choice Set

Menu selection is one of the richest techniques for selecting from a relatively fixed-sized choice set. Here we discuss several key factors in menu design.

Menu order

Menu elements can be organized in many different orders, including alphabetical, logically grouped by functional purpose, most frequently used first, most important first, largest first, or most recently created/modified first. These orders can be combined in various ways. A functionally grouped menu may be ordered alphabetically within group, and the functional groups themselves ordered by frequency of use. Figure 8.11 illustrates several such possible organizations. Consistency of organization from one menu to another is useful, so a common strategy across all menus of an application is important. Several researchers have found functional order to be the most helpful, and many menu structures reflect this result. If the items lack a clear logical organization, alphabetical is a good choice, because it is predictable[PERL84A].

Single-level versus hierarchical design

One of the most fundamental menu design decisions arises if the choice set is too large to display all at once. Such a menu can be subdivided into a logically structured hierarchy, or it can presented as a linear sequence of choices to be scrolled through using a scroll bar. In the limit, the size of the window can be reduced to a single menu item, yielding a "slot-machine" menu of the type shown in Fig. 8.12.

With a hierarchical menu, the user first selects from the choice set at the top of the hierarchy, which causes a second choice set to be available. The process is repeated until a leaf node (i.e., an element of the choice set itself) of the hierarchy tree is selected. As with hierarchical object selection, navigation mechanisms need to be provided so that the user can go back up the hierarchy if an incorrect subtree was selected. Visual feedback to the user some sense of place within the hierarchy is also needed, typically via *cascading menus*, as depicted in Fig. 8.13. Enough of each menu must be revealed that the complete highlighted selection path is visible, and some means must be used to indicate whether a menu item is a leaf node or is the name of a lower-level menu (in the figure, the right-pointing arrow fills this role).

When we design a hierarchical menu, the issue of depth versus breadth is always present. Snowberry et al. [SNOW83] found experimentally that selection time and accuracy improve when broader menus with fewer levels of selection are used. Similar results are reported by Landauer and Nachbar [LAND85] and by other researchers. However, these results do not necessarily generalize to menu hierarchies that lack a natural, understandable structure. Norman[NORM91A] provides thorough coverage of the issues that arise in designing menus.

Hierarchical menu selection almost demands an accompanying keyboard or function key accelerator technique to speed up selection for more experienced (so-called "power") users. This is easy if each node of the tree has a unique name, so that the user can enter the name directly, and the menu system provides a backup if the user's memory fails. If the names are unique only within each level of the hierarchy, the power user must type the complete path name to the desired leaf node.

Menu placement

Menus can be shown on the display screen or on an auxiliary screen or be printed on a tablet or on function-key labels. Onscreen menus can be static and permanently visible, or can appear dynamically on request (tear-off, appearing, pop-up, pull-down, and pull-out menus).

A static menu printed on a tablet, as shown in Color Plate I.18, can easily be used in fixed-application systems. Use of a tablet or an auxiliary screen, however, requires that the user look away from the application display, and hence destroys visual continuity. The advantages are the saving of display space, which is often at a premium, and the accommodation of a large set of commands in one menu.

A pop-up menu is normally invisible but appears on the screen in response to a mouse button press. The menu appears at the cursor location, which is usually the user's center of visual attention, thereby maintaining visual continuity. The menu can be positioned so the most recently made selection from the choice set is nearest the cursor, ready to be selected *if* the most recently selected item is more likely to be selected a second time than is another item.

Pop-up and other appearing menus conserve precious screen space—one of the user-interface designer's most valuable commodities. Their use is facilitated by a fast RasterOp instruction, as discussed in Chapters **XXX_2 and 19_XXX**.

Pop-up menus are often also context-sensitive. If the cursor is over a graphic object in a drawing program, the menu might include scale and rotate commands; if it is over a piece of text, the menu would contain commands to change the font. This context-sensitivity may initially be confusing to the novice, but is powerful once understood.

Unlike pop-up menus, pull-down and pull-out menus are anchored in a menu bar along an edge of the screen. While they do permanently occupy valuable screen space, they serve to announce the availability of the menu to a novice user. The Apple Macintosh and Microsoft Windows interfaces use pull-down menus extensively. Macintosh menus, shown in Fig. 8.14, also illustrate accelerator keys and context sensitivity. These menus have a two-level hierarchy: The menu bar is the first level, and the pull-down menu is the second. Pull-down menus can be activated explicitly or implicitly. In explicit activation, a button depression, once the cursor is in the menu bar, makes the second-level menu appear; the cursor is moved on top of the desired selection and the button is then released. In implicit activation, moving the cursor into the heading causes the menu to appear; no button press is needed. Either selecting an entry or moving the cursor out of the menu area dismisses the menu. These menus, sometimes called "lazy" or "drop-down" menus, may also confuse new users by their seemingly mysterious appearance.

Visual representation

The basic decision on representation is whether menus use textual names or iconic or other graphical representations of elements of the choice set. **XXX_If delete the icon section, then fix this to match_XXX** Icons are discussed further in the next chapter; however, note that iconic menus can be spatially organized in more flexible ways than can textual menus, because icons need not be long and thin like text strings; see Fig. 8.15. Also, inherently graphical concepts (particularly graphical attributes and geometrical primitives) are easily depicted.

Size and shape of menu items

Pointing accuracy and speed are affected by the size of each individual menu item. Larger items are faster to select, as predicted by Fitts' law [FITT54; CARD83]; on the other hand, smaller items take less space and permit more menu items to be displayed in a fixed area, but induce more errors during selection. Thus, there is a conflict between using small menu items to preserve screen space versus using larger ones to decrease selection time and to reduce errors.

Pop-up *pie menus*[CALL88A], shown in Fig. 8.16, appear at the cursor. As the user moves the mouse from the center of the pie toward the desired selection, the target width becomes larger, decreasing the likelihood of error. Thus, the user has explicit control over the speed-versus-error tradeoff. In addition, the distance to each menu item is the same, only the angle is different. This means that an expert user could operate the menu without ever seeing it displayed, simply by making a stroke in the right direction. This is the idea behind the *marking menu*, a form of pie menu where the visual feedback is optional; if the user moves rapidly, the menu is never shown on the screen. It simply becomes a gesture command. This can be extended to hierarchical pie menus, where the gesture for a command might be "move upward (for the first menu pick), then right (first submenu), then diagonally downward and to the left (second submenu)." The example in Fig. 8.16 contains a two-level marking menu[TAPI95A].

Pattern recognition

In selection techniques involving pattern recognition, the user makes gestures or sequences of movements with a continuous-positioning device, such as a tablet stylus or mouse. The pattern recognizer automatically compares the sequence with a set of defined patterns, each of which corresponds to an element of the selection set[RUBI91A]. Figure 8.17 shows one set of sketch patterns and their related commands, taken from Wallace's SELMA queueing analyzer [IRAN71]. Proofreader's marks indicating delete, capitalize, move, and so on are attractive candidates for this approach [WOLF87]. Another approach for pen-based systems is to support interaction that more closely resembles the way a person would use a regular pen rather than a mouse, such as making circle and arrow gestures to move blocks of text or writing text insertions directly where they should go[MORA95A]. Given a data tablet and stylus, pattern recognition can be used with at least several dozen patterns, but it is difficult for the user to learn a large number of different patterns.

These techniques require no typing skill and preserve tactile continuity. Furthermore, if the command involves an object, the cursor position at the beginning of the gesture can be used for selection. The drag command used in many graphical interfaces is a simple example: the cursor is positioned on top of the object to be dragged and the mouse button is pressed, selecting the object under the cursor (it is displayed in reverse video for feedback). As the user moves the mouse (still holding down the button), the object moves also. Releasing the mouse button detaches the object from the mouse. Skilled operators can work very rapidly with this technique, because hand movements between the work area and a command-entry device are eliminated.

The "toolglass" technique extends this approach by simulating a semitransparent stencil or template, which a user can manipulate with one hand (using a trackball) while using a mouse or stylus with the other. By placing the selected part of the stencil or toolglass over the selected object in a drawing and clicking, a single click can specify both a command and its target objects[STON94A, BIER93A].

Function keys

Elements of the choice set can be associated with function keys. (We can think of single-keystroke inputs from a regular keyboard as function keys.) Unfortunately, there never seem to be enough keys to go around! The keys can be used in a hierarchical-selection fashion, and their meanings can be altered using chords, say by depressing the keyboard shift and control keys along with the function key itself. Learning exotic key combinations, such as "shift-option-control-L," for some commands is not easy, however, and is left as an exercise for the regular user seeking the productivity gains that typically result. Putting a "cheat-sheet" template on the keyboard to remind users of these obscure combinations can speed up the learning process. Dedicated hardware function keys are beginning to appear on keyboards for web browsing functions and for media playback functions.

In addition to using the different buttons of a mouse, rapid double or triple clicks can provide additional functions. Chording of keyboard keys with mouse buttons (Shift-click, Control-click) can also be used to provide the logical (but not necessarily human-factors) equivalent of more mouse buttons.

Text

The text-string input task entails entering a character string to which the application does not ascribe any special meaning. Thus, typing a command name is *not* a text-entry task. In contrast, typing legends for a graph and typing text into a word processor *are* text input tasks. Clearly, the most common text-input technique is use of the QWERTY keyboard, though small pocket computers require more compact approaches.

Character recognition

The user writes characters with a continuous-positioning device, usually a tablet stylus, and the computer recognizes them. This is considerably easier than recognizing scanned-in characters, because the tablet measures the sequence, direction, and sometimes speed and pressure of strokes, and a pattern-recognition algorithm can match these to stored templates for each character. For instance, the capital letter "A" consists of three strokes—typically, two downward strokes and one horizontal stroke. A recognizer can be trained to identify different styles of block printing: the parameters of each character are calculated from samples drawn by the user. Character recognizers have been used with interactive graphics since the early 1960s [BROW64; TEIT64].

It is difficult to block print more than one or two characters per second (try it!), so character recognition is not appropriate for massive input of text. We write cursive letters faster than we print the same characters, but recognition is less robust. Another technique is to use an alphabet of characters specially designed to be easily distinguishable from one another to facilitate computer recognition and designed so that each can be drawn with a single stroke without lifting the pen, which makes it easier for the computer to find the boundaries between the letters[GOLD93A], such as Graffiti **XXX_(TM)_XXX**. It also makes it possible to use a very small input area, in which the input letters are written in succession, on top of one another, for some applications. This technique is widely used in palmtop computers, where there is no room for a keyboard.

Menu selection

XXX_Could delete this paragraph_XXX A series of letters, syllables, or other basic units is displayed as a menu. The user then inputs text by choosing letters from the menu with a selection device. This technique is attractive if only a short character string is to be entered and the user's hands are already on a pointing device. It is also useful if the character set is large, as in Chinese and Japanese. Another strategy for such languages is to enter the word in phonetic spelling, which string is then matched in a dictionary. For example, the Japanese use two alphabets, the katakana and hiragana, to type phonetically the thousands of kanji characters that their orthography borrows from the Chinese.

Evaluation of text-entry techniques

For massive input of text, the only reasonable substitute for a skilled typist working with a keyboard is an automatic scanner. Figure 8.18 shows experimentally determined keying rates for a variety of techniques. The hunt-and-peck typist is slowed by the perceptual task of finding a key and the ensuing motor task of moving to and striking it, but the trained typist has only the motor task of striking the key, preceded sometimes by a slight hand or finger movement to reach it. Speech input, not shown on the chart, is slower but attractive for applications where the hands must be free for other purposes, such as handling paperwork.

Quantify

The quantify interaction task involves specifying a numeric value between some minimum and maximum value. Typical interaction techniques are typing the value, setting a slider or dial to the value, and using an up-down counter to select the value. Like the positioning task, this task may be either linguistic or spatial. When it is linguistic, the user knows the specific value to be entered; when it is spatial, the user seeks to increase or decrease a value by a certain amount, with perhaps an approximate idea of the desired end value. In the former case, the interaction technique clearly must involve numeric feedback of the value being selected (one way to do this is to have the user type the actual value); in the latter case, it is more important to give a general impression of the approximate setting of the value. This is typically accomplished with a spatially oriented feedback technique, such as display of a slider or dial on which the current (and perhaps previous) value is shown.

With continuous-scale manipulation, the user drags an indicator along a displayed slider or gauge; a numeric echo may also be given. Figure 8.19 shows several such interaction techniques and their associated feedback.

Another hardware technique is the potentiometer, discussed in Section **XXX_8.1.3_XXX**. The decision of whether to use a rotary or linear potentiometer should take into account whether the visual feedback of changing a value is rotary (e.g., a turning clock hand) or linear (e.g., a rising temperature gauge). The current position of one or a group of slide potentiometers is much more easily comprehended at a glance than are those of rotary potentiometers, even if the knobs have pointers. It is important to use directions consistently: clockwise or upward movements normally increase a value.

3D Interaction Tasks

Two of the four interaction tasks described previously for 2D applications become more complicated in 3D: position and select. In this section, we also introduce an additional 3D interaction task: rotate (in the sense of orienting an object in 3-space). The most straightforward way to perform these tasks in 3D is to use an actual 3D input device (like a Polhemus tracker or Spaceball, described in Section **XXX_8.1.6_XXX**).

A complication arises from the difficulty of perceiving 3D depth relationships of a cursor or object relative to other displayed objects on a 2D display screen. This contrasts starkly with 2D interaction, where the user can readily perceive that the cursor is above, next to, or on an object. Display of stereo pairs, corresponding to left- and right-eye views, is helpful for understanding general depth relationships, but is of limited accuracy as a precise locating method. Methods for presenting stereo pairs to the eye are discussed in Chapters **XXX_14 and 18_XXX**, and in [HODG85]. Other ways to show depth relationships are discussed in Chapters **XXX_14-16_XXX**. Researchers have studied the different ways of showing depth on a 2D screen to find out which are most useful for performing interactive 3D graphical tasks. For example, Ware and Zhai[ZHAI96A] have conducted experiments in which subjects perform a 3D manipulation task, such as docking one 3D object with another, using conventional perspective 3D images, stereo images, head-motion parallax, transparency, and other effects to enhance their 3D perception. In general, they have found that each of these cues helps a little, but some are more significant than others, and some combinations are particularly effective. Head-motion parallax is one of the strongest cues, stronger than stereo vision.

Researchers at Brown University[HERN92A] have developed another interesting technique to make 3D manipulation easier to perceive on a 2D screen. They cast imaginary shadows from the 3D object onto "walls" of the surrounding "room," which are aligned with the three coordinate axes, as seen in Fig. 8.20. These shadow images give the two-axis projections of the 3D object, which facilitate aligning and adjusting it. The user can also manipulate the shadows themselves. This causes the real object to move to where it should be to cast the desired shadow.

If the application requires the user to input 3D selection, position, or rotation commands with a 2D device like a mouse or tablet, special interaction techniques are required map movements of these 2D devices into 3D. We describe some of these 2D-to-3D interaction techniques, first, for positioning and selecting, which are closely related, and, second, for interactive rotation.

XXX_Could delete this, except keep virtual trackball_XXX Figure 8.21 shows one way to position in 3D. The 2D cursor, under control of, say, a mouse, moves freely among the three views. The user can select any one of the 3D cursor's dashed lines and drag it. If the button-down event is close to the intersection of two dashed cursor lines, then both are selected and are moved with the mouse (gravity, discussed in Section **XXX_8.3.2_XXX**, can make picking the intersection especially easy). Although this method may appear restrictive in forcing the user to work in one or two dimensions at a time, it is sometimes advantageous to decompose the 3D manipulation task into simpler lower-dimensional tasks. Selecting as well as locating is facilitated with multiple views: Objects that overlap and hence are difficult to distinguish in one view may not overlap in another view.

Another possibility, developed by Nielson and Olsen [NIEL86] and depicted in Fig. 8.22, requires that all three principal axes project with nonzero length. A 3D cross-hair cursor, with cross-hairs parallel to the principal axes, is controlled by moving the mouse in the general direction of the projections of the three principal axes. Figure 8.23 shows how 2D locator movements are mapped into 3D: there are 2D zones in which mouse movements affect a specific axis. Of course, 3D movement is restricted to one axis at a time.

Both of these techniques illustrate ways to map 2D locator movements into 3D movements. We can instead use buttons to control which of the 3D coordinates are affected by the locator's 2 degrees of freedom. For example, the locator might normally control *x* and *y*; but, with a button depressed, it could control *x* and *z* instead.

Constrained 3D movement is effective in 3D locating. Gridding and gravity can sometimes compensate for uncertainties in depth relationships and can aid exact placement. Another form of constraint is provided by those physical devices that make it easier to move along principal axes than in other directions (Section **XXX_8.1.6_XXX**).

Context-specific constraints are often more useful, however, than are these general constraints. It is possible to let the user specify that movements should be parallel to or on lines or planes other than the principal axes and planes. For example, with a method developed by Nielson and Olsen [NIEL86], the local coordinate system of the selected object defines the directions of movement as shown in Fig. 8.24. In a more general technique developed by Bier [BIER86b], the user places a coordinate system, called a *skitter*, on the surface of an object, again defining the possible directions of movement (Fig. 8.25).

One method of 3D picking—finding the output primitive that, for an (*x*, *y*) position determined by a 2D locator, has the maximum *z* value—was discussed in Chapter **XXX_7_XXX**. Another method, which can be used with a 3D locator when wireframe views are shown—is to find the output primitive closest to the locator's (*x*, *y*, *z*) position.

As with locating and selection, the issues in 3D rotation are understanding depth relationships, mapping 2D interaction devices into 3D, and ensuring stimulus-response compatibility. An easily implemented 3D rotation technique provides slider dials or gauges that control rotation about three axes. S-R compatibility suggests that the three axes should normally be in the screen-coordinate system—*x* to the right, *y* increasing upward, *z* out of (or into) the screen [BRIT78]. Of course, the center of rotation either must be explicitly specified as a separate step, or must be implicit (typically the screen-coordinate origin, the origin of the object, or the center of the object). Providing rotation about the screen's *x* and *y* axes is especially simple, as suggested in Fig. 8.26. The (*x*, *y*, *z*) coordinate system associated with the sliders is rotated as the sliders are moved to show the effect of the rotation. The two-axis technique can be easily generalized to three axes by adding a dial for *z*-axis rotation, as in Fig. 8.27 (a dial is preferable to a slider for S-R compatibility).

Mouse movements can also be directly mapped onto object movements, without slider or dial intermediaries. The user can be presented a metaphor in which the two sliders of Fig. 8.26 are invisibly superimposed on top of the object being rotated, so that horizontal mouse movements are mapped into rotations about the screen-coordinate *y* axis, and vertical mouse movements are mapped into rotations about the screen-coordinate *x* axis (Fig. 8.28a). Diagonal motions have no effect. The sliders are not really displayed; the user imagines that they are present. Alternatively, an imaginary 2D trackball can be superimposed on top of the object being rotated, so that the vertical, horizontal, or diagonal motions one would make with the trackball can be made instead with the mouse (Fig. 8.28b). Either of these methods provides two-axis rotation in 3D.

For three-axis rotations, three methods that closely resemble real-world concepts are particularly interesting. In the overlapping-sliders method [CHEN88], the user is shown two linear sliders overlapping a rotary slider, as in Fig. 8.28(c). Motions in the linear sliders control rotation about the *x* and *y* axes, while a rotary motion around the intersection of the two linear sliders controls rotation about the *z* axis. In a technique developed by Evans, Tanner, and Wein [EVAN81], three successive mouse positions are compared to determine whether the mouse motion is linear or rotary. Linear horizontal or vertical movements control rotation about the *x* and *y* axes, a linear diagonal movement rotates about both the *x* and *y* axes, and rotary movements control rotation about the *z* axis. While this is a relative technique and does not require that the movements be made directly over the object being rotated or in a particular area, the user can be instructed to use these motions to manipulate a 3D trackball superimposed on the object (Fig. 8.28d). In the virtual-sphere method, also developed by Chen [CHEN88], the user actually manipulates this superimposed 3D trackball in an absolute fashion as though it were real. With a mouse button down, mouse movements rotate the trackball exactly as your finger would move a real trackball. An experiment [CHEN88] comparing these latter two approaches showed no performance differences, but did yield a user preference for Chen's method.

Figure 8.29 shows an interface for viewing 3D VRML objects in a web browser that combines several of these approaches[MOHA96A]. It provides three separate spatial controls: the arrow pad for *x*-*y* translation, the thumbwheel for *z* translation, and the trackball for rotation. The controls are accessible via the icons at the bottom of the screen or by using the mouse in the main window with different shift-key combinations. The two translation controls are straightforward. The rotation control emulates a trackball centered about the middle of the main window. It also interprets circular motion near the edge of the trackball or window as rotation about the *z* axis. It also allows the user to spin the object by releasing the mouse button while the mouse is moving; the object then continues spinning in the same direction.

It is often necessary to combine 3D interaction tasks. Thus, rotation requires a select task for the object to be rotated, a position task for the center of rotation, and an orient task for the actual rotation. Specifying a 3D view can be thought of as a combined positioning (where the eye is), orientation (how the eye is oriented), and scaling (field of view, or how much of the projection plane is mapped into the viewport) task. We can create such a task by combining some of the techniques we have discussed, or by designing a *fly-around* capability in which the viewer flies an imaginary airplane around a 3D world. The controls are typically pitch, roll, and yaw, plus velocity to speed up or slow down. With the fly-around concept, the user can also be given an overview, such as a 2D plan view, indicating the imaginary airplane's ground position and heading. Navigating in a 3D virtual environment is particularly difficult, since many of the subtle cues present in the real world are lacking. Overview maps can be provided here too, right in the head-mounted display image, as seen in Fig. 8.30[DARK96A]. Another technique uses handheld miniature copy of the virtual environment, displayed in the head-mounted display in addition to the conventional view; the user can manipulate the "world in miniature" to change the viewpoint for the main display[STOA95A].

Virtual environments provide a compelling 3D effect for both viewing and interaction. They also open up a much larger design space for new interaction techniques than conventional interface styles. Research into inventing and testing useful interaction techniques for use in virtual environments is expanding[HINC94B, MINE97A, WARE94A]; Fig. 8.31 shows examples of work on new interaction techniques for VR[MINE97A].

Interaction Techniques for Composite Interaction Tasks

Composite interaction tasks are combinations of basic interaction tasks described above integrated into a unit. If one thinks of basic interaction tasks as atoms, then composite interaction tasks are molecules. We describe interaction techniques for some of them in this section.

Dialogue Boxes

Dialogue boxes are used to make several selections in one operation. We often need to select multiple elements of a selection set. For instance, text attributes, such as italic, bold, underline, hollow, and all caps, are not mutually exclusive, and the user may want to select two or more at once. In addition, there may be several sets of relevant attributes, such as typeface and font. Some of the menu approaches useful in selecting a single element of a selection set are not satisfactory for multiple selections. For example, pull-down and pop-up menus normally disappear when a selection is made, necessitating a second activation to make a second selection. This problem is overcome with dialogue boxes, which remains visible until explicitly dismissed by the user. In addition, dialogue boxes permit selection from more than one selection set and can include areas for interaction techniques for entering text and other values. Selections made in a dialogue box can be corrected immediately. When all the information has been entered into the dialogue box, the box is typically dismissed explicitly with a command. *Modal* dialogue boxes prevent the user from doing anything else until the dialogue box is dismissed; these should be used sparingly, for critical tasks. Figure 8.32 shows a dialogue box with several selected items highlighted.

Construction Techniques

One way to construct a line is to have the user indicate one endpoint and then the other; once the second endpoint is specified, a line is drawn between the two points. With this technique, however, the user has no easy way to try out different line positions before settling on a final one, because the line is not actually drawn until the second endpoint is given. With this style of interaction, the user must invoke a command each time an endpoint is to be repositioned.

A far superior approach is *rubberbanding*, discussed in Chapter **XXX_2_XXX**. When the user pushes a button (often a mouse or stylus button), the starting position of the line is established by the cursor. As the cursor moves, so does the endpoint of the line; when the button is released, the endpoint is frozen. Figure 8.33 show a rubberband line-drawing sequence. The user-action sequence is shown in the state diagram in Fig. 8.34. Notice that the state "rubberband" is active *only* while a button is held down. It is in this state that cursor movements cause the current line to change. See [BUXT85] for an informative discussion of the importance of matching the state transitions in an interaction technique with the transitions afforded by the device used with the technique.

Similarly, the *rubber-rectangle* technique starts by anchoring one corner of a rectangle with a button-down action, after which the opposite corner is dynamically linked to the cursor until a button-up action occurs. The state diagram for this technique differs from that for rubberband line drawing only in the dynamic feedback of a rectangle rather than a line. The *rubber-circle* technique creates a circle that is centered at the initial cursor position and that passes through the current cursor position, or that is within the square defined by opposite corners. The *rubber-ellipse* technique creates an axis-aligned ellipse inside the rectangle defined by the initial and current cursor positions.

One interaction technique for creating a polyline (a sequence of connected lines) is an extension of rubberbanding. After entering the polyline-creation command, the user clicks on a button to anchor each rubberbanded vertex. After all the vertices have been indicated, the user indicates completion, typically by a double click, a click on a different mouse button, or entry of a new command. Figure 8.35 depicts a typical sequence of events in creating a polyline; Fig. 8.36 is the accompanying state diagram.

A polygon can be drawn similarly. In some cases, the user signals to the system that the polygon is complete by returning to the starting vertex of the polygon. In other cases, the user explicitly signals completion using a double click or other command, and the system automatically inserts the final line to close the polygon. Figure 8.37 shows one way to create polygons.

Constraints of various types can be applied to the cursor positions in any of these techniques. For example, Fig. 8.38 shows a sequence of lines drawn using the same cursor positions as in Fig. 8.33, but with a horizontal constraint in effect. A vertical line, or a line at some other orientation, can also be drawn in this manner. Polylines made entirely of horizontal and vertical lines, as in printed circuit boards, VLSI chips, and some city maps, are readily created; right angles are introduced either in response to a user command, or automatically as the cursor changes direction. The idea can be generalized to any shape, such as a circle, ellipse, or any other curve; the curve is initialized at some position, then cursor movements control how much of the curve is displayed. In general, the cursor position is used as input to a constraint function whose output is then used to display the appropriate portion of the object.

Gravity is yet another form of constraint. When constructing drawings, we frequently want a new line to begin at the endpoint of, or on, an existing line. Matching an endpoint is easy if it was created using gridding, but otherwise is difficult without a potentially time-consuming zoom. The difficulty is avoided by programming an imaginary gravity field around each existing line, so that the cursor is attracted to the line as soon as it enters the gravity field. Figure 8.39 shows a line with a gravity field that is larger at the endpoints, so that matching endpoints is especially easy. With the snap-dragging technique, continuous visual feedback is also provided by "snapping" the object being dragged to each successive gravity field as the user moves the cursor [BIER86a]. An analogous idea can be used for object selection. "Smart selection" techniques allow the user to point near, but not directly on, the desired object and still select it, provided the user action was closer to the desired object than any other object on the screen by at least some minimum amount, and, perhaps, closer to the desired object than some minimum threshhold. **XXX_Osga reference?_XXX** The effect is similar to surrounding each of the selectable objects with a gravity field.

Dynamic Manipulation

It is not sufficient to create lines, rectangles, and so on. In many situations, the user must be able to modify previously created geometric entities.

Dragging moves a selected symbol from one position to another under control of a cursor. A button-down action typically starts the dragging and selects the symbol under the cursor to be dragged; then, a button-up freezes the symbol in place, so that further movements of the cursor have no effect on it. Rekimoto[REKI97A] extends this idea to allow drag and drop operations across different computers and to and from wall displays, PDAs, and other devices.

Dynamic rotation of an object can be done in a similar way, except that we must be able to identify the point or axis about which the rotation is to occur. A convenient strategy is to have the system show the current center of rotation and to allow the user to modify it as desired. Figure 8.40 shows one such scenario. Note that the same approach can be used for scaling, with the center of scaling, rather than that of rotation, being specified by the user.

The concept of *handles* is useful to provide scaling of an object, without making the user think explicitly about where the center of scaling is. Figure 8.41 shows an object with eight handles, displayed as small squares at the corners and on the sides of the imaginary box surrounding the object. The user selects one of the handles and drags it to scale the object. If the handle is on a corner, then the corner diagonally opposite is locked in place. If the handle is in the middle of a side, then the opposite side is locked in place. The handles normally appear only when the object is selected to be operated on; they thus provide a unique visual code to indicate that an object is selected, since other visual codings (e.g., line thickness, dashed lines, or changed intensity) might also be used as part of the drawing itself. (Blinking is another unique visual code, but tends to be distracting and annoying.)

Dragging, rotating, and scaling affect an entire object. What if we wish to be able to move individual points, such as the vertices of a polygon? Vertices could be named, and the user could enter the name of a vertex and its new (x, y) coordinates. But the same point-and-drag strategy used to move an entire object is more attractive. In this case, the user points to a vertex, selects it, and drags it to a new position. The vertices adjacent to the one selected remain connected via rubberband lines. To facilitate selecting a vertex, we can establish a gravity field to snap the cursor onto a nearby vertex or we can superimpose handles over each vertex. For smooth curves and surfaces, handles can also be provided to allow the user to manipulate points that control the shape, as discussed further in Chapter **XXX_11_XXX**.

Zooming

Many graphics programs allow the user to zoom in and out to see an expanded or contracted view of the graphic workspace. Zooming by fixed steps with discrete commands is easiest to implement. Continuous zooming of a complex picture requires more powerful graphics hardware. The PAD[PERL93A] and PAD++[BEDE94A] systems extend this notion further. They provide the user an infinitely-zoomable 2D workspace, on which she can draw or place objects at any magnification. Some objects may be invisible or collapsed to a single pixel in the high-level view, but the user can zoom in as far as desired to see the object. She can continue zooming in and even place an entire other object inside a punctuation mark or within the width of a narrow line in its parent object. Figure 8.42 shows the PAD++ system in operation.

These systems also extend this technique to use *semantic zooming*: as an object grows to different sizes in the zoomed image, it not only expands in size but changes the way in which it is displayed. For example, at low magnification, a document might be represented as a solid rectangle; as the user zooms in, the main headings would appear, but not the full text; still further, the full text of the document would appear. Furnas provides a visual formalism for describing and programming this kind of non-geometric zooming operation[FURN95A].

In the next chapter, we discuss design issues involved in combining basic and composite interaction techniques into an overall user-computer dialogue.

EXERCISES

EXERCISE 00. Examine a 2D graphical user-computer interface with which you are familiar. List each interaction task used. Categorize each task into one of the basic interaction techniques of Section **XXX_8.2_XXX**. If an interaction does not fit this classification scheme, try decomposing it further.

EXERCISE 00. Implement adaptive C/D ratio cursor tracking for use with a mouse or other relative-positioning device. Experiment with different relationships between mouse velocity v and the C/D ratio r: r = kv and r = k v **XXX_(SQUARED)_XXX**. You must also find a suitable value for the constant k.

EXERCISE 00. Conduct an experiment to compare the selection speed and accuracy of any of the following pairs of techniques:

- a. Mouse and another pointing device selecting from a static, onscreen menu
- b. Mouse and accelerator keys selecting from a static, onscreen menu
- c. Wide, shallow menu and narrow, deep menu
- d. Pull-down menus that appear as soon as the cursor is in the menu bar, and pull-down menus that require a mouse-button depression.

EXERCISE 00. Extend the state diagram of Fig. 8.10 to include a "return to lowest level" command that takes the selection back to the lowest level of the hierarchy, such that whatever was selected first is selected again.

EXERCISE 00. Implement an autocompletion text-entry technique to use with an arbitrary list of words. Experiment with different word sets, such as the UNIX commands and proper names. Decide how to handle nonexistent matches, corrections typed by the user after a match has been made, and prompting for the user.

EXERCISE 00. Implement cascading hierarchical menus for a series of command or for file-system subdirectories. What issues arise as you do this? Implement an alternative selection technique, and informally compare the speed of the two.

EXERCISE 00. Implement pop-up menus that allow multiple selections prior to dismissal, which the user accomplishes by moving the cursor outside the menu. Alternatively, use a button click for dismissal. Which dismissal method do you prefer? Explain your answer. Ask five people who use the two techniques which dismissal method they prefer.

EXERCISE 00. Implement a menu package on a color display such that the menu is displayed in a strong, bright but partially transparent color, and all the colors underneath the menu are changed to a subdued gray.

EXERCISE 00. Implement any of the 3D interaction techniques discussed in this chapter.

EXERCISE 00. For each of the locating techniques discussed in Section **XXX_8.2.6_XXX**, identify the line or plane into which 2D locator movements are mapped.

EXERCISE 00. Draw the state diagram that controls pop-up hierarchical menus.

Figure 1. A subject using an eye tracker. The eye tracker camera (on the left) observes the user's eye through the servo-controlled mirror located under the display screen, and reports where on the screen the user is looking every 1/60 second. **XXX_ GRAPHIC: eye.tiff (placeholder) SOURCE: NRL _XXX**

Figure 2. Numeral 2, a drawing in the spirit of Jasper Johns, by Teresa Bleser. Drawn with the GWPaint program using a GTCO pressure- and tilt-sensitive tablet. (Courtesy of T. Bleser, George Washington University.) **XXX_ GRAPHIC: old8.1 _XXX**

Figure 3. The metaDESK tangible user interface. The user can manipulate the physical objects on the desk to interact with the computer and see the results on the display projected onto the desktop. **XXX_ GRAPHIC: metadesk.bmp (placeholder) SOURCE: figure 10 (metaDESK, with instrument, phicons, and lenses) (all labeled) from[ISHI97A]: _XXX**

Figure 4. The Polhemus 3SPACE **XXX_(TM)_XXX** FASTRAK **XXX_(TM)_XXX** 3D magnetic tracker. It sends 6 numbers, containing the position and orientation in 3D of each of the four sensors (the small white cubes in the foreground), using a magnetic signal sent from the transmitter (the larger black cube on the right). **XXX_ GRAPHIC: polhemus.tiff (placeholder) SOURCE: From my CRC article, which was from Polhemus, Inc., Colchester, Vt. _XXX**

Figure 5. The Virtual Technologies CyberGlove **XXX_(TM)_XXX**, showing the 18 sensors on each hand that are used to sense finger movements, and the 3D magnetic sensor on the wristband that senses the position and angle of the hand itself. **XXX_ GRAPHIC: glove.tiff (placeholder) SOURCE: From my CRC article, which was from Virtual Technologies,**

Inc., Palo Alto, Calif. _XXX

Figure 6. A head-mounted display in use for virtual reality. The 3D tracker attached above the user's left eye reports the position and orientation of the head. The computer uses this information to update the viewpoint of the display constantly. This unit also measures the position of the user's eye, by monitoring it through the mirror located in front of the left eye. **XXX_ GRAPHIC: hmd.tiff SOURCE: Tufts photographer _XXX**

Figure 7. The Phantom haptic feedback device. As the user moves the end of the arm, he can feel forces pushing back against his hand, giving the sensation of touching an object. **XXX_ GRAPHIC: haptic.jpg (placeholder) SOURCE: Phantom model 1.5, web site of Sensable Technology, Cambridge, Ma. _XXX**

Figure 8. Numeric feedback regarding size of an object being constructed, The height and width are changed as the cursor (+) is moved, so the user can adjust the object to the desired size. **XXX_ GRAPHIC: old8.4 _XXX**

Figure 9. Continuous sketching. **XXX_ GRAPHIC: old8.5 _XXX**

Figure 10. State diagram for an object-selection technique for an arbitrary number of hierarchy levels. Up and Down are commands for moving up and down the hierarchy. In the state "Leaf object selected," the Down_hierarchy command is not available. The user selects an object by pointing at it with a cursor, and pressing and then releasing a button. **XXX_ GRAPHIC: old8.6 _XXX**

Figure 11. Three menu organizations. (a) Menu using an alphabetical sequence. (b) Menu using functional grouping, with alphabetical within-group order as well as alphabetical-between-group order. (c) Menu with commands common to several different application programs placed at the top for consistency with the other application's menus; these commands have heavier borders. Menu items are some of those used in Card's menu-order experiment [CARD82]. **XXX_ GRAPHIC: old8.7 _XXX**

Figure 12. A small menu-selection window. Only one menu item appears at a time. The scroll arrows are used to change the current menu item, which is selected when the Accept button is chosen. **XXX_ GRAPHIC: new8.9.bmp SOURCE: Visual Basic run-time _XXX**

Figure 13. A pop-up hierarchical menu. (a) The first menu appears where the cursor is, in response to a button-down action. The cursor can be moved up and down to select the desired typeface. (b) The cursor is then moved to the right to bring up the second menu. (c) The process is repeated for the third menu. **XXX_ GRAPHIC: new8.10[abc].bmp SOURCE: Visual Basic run-time _XXX**

Figure 14. A Macintosh pull-down menu. The last menu item is gray rather than black, indicating that it is currently not available for selection (the currently selected object, an arc, does not have corners to be rounded). The Undo command is also gray, because the previously executed command cannot be undone. Abbreviations are accelerator keys for power users. (Copyright 1988 Claris Corporation. All rights reserved.) **XXX_ GRAPHIC: old8.13 _XXX**

Figure 15. Iconic and textual menus for the same geometric primitives. The iconic menu takes less space than does the textual menu. (Icons **XXX_(C)_XXX** 1988 Claris Corporation. All rights reserved.) **XXX_ GRAPHIC: old8.15 _XXX**

Figure 16. A pie menu, with a second submenu appearing above it. The optional marking menu gesture command is also shown. **XXX_ GRAPHIC: marking.bmp (placeholder) SOURCE: fig. 2 of[TAPI95A]: _XXX**

Figure 17. Motions, indicated as dotted lines, that are recognized as commands. From Wallace's SELMA queuing analyzer [IRAN71]. **XXX_ GRAPHIC: old8.18 _XXX**

Figure 18. Data-input speeds, in keystrokes per minute, of various techniques for entering text and numeric information. (Adapted from [VANC72, p. 335] and [CARD83, p. 61].) **XXX_ GRAPHIC: old8.20 _XXX**

Figure 19. Several dials that the user can use to input values by dragging the control pointer. Feedback is given by the pointer and, in two cases, by numeric displays. **XXX_ GRAPHIC: new8.21.bmp SOURCE: VERTICAL SLIDERS: SwingSet demo applet and Visual Basic run-time _XXX**

Figure 20. Using interactive shadows to make 3D manipulation easier. The user can manipulate the 3D object itself or any of its 2D shadows. **XXX_ GRAPHIC: shadows1.bmp OR shadows2.bmp (placeholder) SOURCE: Figure 2 or 3 from UIST'92 3D shadows paper _XXX**

Figure 21. 3D positioning technique using three views of the same scene (a house). The 2D cursor (+) is used to select one of the dashed 3D cursor lines. **XXX_ GRAPHIC: old8.23 _XXX**

Figure 22. Movement of the 3D cursor is controlled by the direction in which the 2D cursor is moved. **XXX_ GRAPHIC: old8.24 _XXX**

Figure 23. The six regions of mouse movement, which cause the 3D cursor to move along the principal axes. **XXX_ GRAPHIC: old8.25 _XXX**

Figure 24. The displayed local coordinate system of the house, which shows the three directions in which any translated object will move. To preserve stimulus-response compatibility, we can use the direction of mouse movements to determine the axes chosen, as in Fig. 8.23. **XXX_ GRAPHIC: old8.26 _XXX**

Figure 25. The displayed coordinate system, placed interactively so that its (x, y) plane coincides with the plane of the roof, shows the three directions in which any translated object will move. **XXX_ GRAPHIC: old8.27 _XXX**

Figure 26. Two slider dials for effecting rotation about the screen x and y axes. **XXX_ GRAPHIC: new8.28.bmp SOURCE: Visual Basic run-time _XXX**

Figure 27. Two slider dials for effecting rotation about the screen x and y axes, and a dial for rotation about the screen z axis. The coordinate system represents world coordinates and shows how world coordinates relate to screen coordinates. **XXX_ GRAPHIC: new8.29.bmp SOURCE: Visual Basic run-time _XXX**

Figure 28. Four methods of 3D rotation. In each case, the user makes movements with a 2D device corresponding to those that would be made if the actual devices were superimposed on the object. A 3D trackball can be twisted to give z-axis rotation, whereas a 2D trackball provides only two-axis rotation. **XXX_ GRAPHIC: old8.31 _XXX**

Figure 29. An interface for navigation through 3D VRML worlds with in a web browser. Several navigation controls are provided, along the bottom of the screen, including the arrow pad for *x*-*y* translation, the thumbwheel for *z* translation, and the trackball for rotation. **XXX_ GRAPHIC: cosmo.gif SOURCE: SGI WebSpace online documentation _XXX**

Figure 30. View through a virtual reality head-mounted display, showing a ship on the ocean in the distance. A map of the same area is shown in the foreground, to aid in navigation. **XXX_ GRAPHIC: darken.bmp (placeholder) SOURCE: fig 1 B, p. 145, of[DARK96A] _XXX**

Figure 31. Sketches of two interaction techniques for virtual reality: A pop-up "look-at" menu and a two-handed flying technque, where the relative positions of the user's two hands determine the speed and direction of flight. **XXX_ GRAPHIC: brooksa.bmp AND brooksb.bmp (placeholder) SOURCE: fig. 4 AND 5 from[MINE97A]: _XXX**

Figure 32. A text-attribute dialogue box with several different attributes selected. As the user chooses items in the lists, a sample of the resulting text font is shown in the "Sample" field. **XXX_ GRAPHIC: new8.32.bmp SOURCE: MS Works _XXX**

Figure 33. Rubberband line drawing. **XXX_ GRAPHIC: old8.33 _XXX**

Figure 34. State diagram for rubberband line drawing. **XXX_ GRAPHIC: old8.34 _XXX**

Figure 35. Rubberband polyline sketching. **XXX_ GRAPHIC: old8.35 _XXX**

Figure 36. State diagram for rubberband creation of a polyline. **XXX_ GRAPHIC: old8.36 _XXX**

Figure 37. Rubberband drawing of a polygon. **XXX_ GRAPHIC: old8.37 _XXX**

Figure 38. Horizontally constrained rubberband line drawing. **XXX_ GRAPHIC: old8.38 _XXX**

Figure 39. Line surrounded by a gravity field, to aid picking points on the line: If the cursor falls within the field, it is snapped to the line. **XXX_ GRAPHIC: old8.39 _XXX**

Figure 40. Dynamic rotation. **XXX_ GRAPHIC: old8.41 _XXX**

Figure 41. Handles used to reshape objects. **XXX_ GRAPHIC: old8.42 _XXX**

Figure 42. The PAD++ system. From left to right, top to bottom, the user is zooming into the image, revealing additional information. **XXX_ GRAPHIC: pad.bmp (placeholder) SOURCE: Figure 1 of Bederson UIST 94 _XXX**

NEW REFERENCES

azum94a.R. Azuma and G. Bishop, ‘‘Improving Static and Dynamic Registration in an Optical See-through HMD,’’ *Proc. ACM SIGGRAPH’94 Conference,* pp. 197-204, Addison-Wesley/ACM Press, 1994.

baec95a.R.M. Baecker, J. Grudin, W.A.S. Buxton, and S. Greenberg, *Readings in Human-Computer Interaction: Toward the Year 2,000,* Morgan Kaufmann, San Francisco, 1995.

bede94a.B.B. Bederson and J.D. Hollan, ‘‘Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics,’’ *Proc. ACM UIST’94 Symposium on User Interface Software and Technology*, pp. 17-26, Addison-Wesley/ACM Press, Marina del Rey, Calif., 1994.

bier93a.E.A. Bier, M.C. Stone, K. Pier, W. Buxton, and T. DeRose, ‘‘Toolglass and Magic Lenses: The See-Through Interface,’’ *Proc. ACM SIGGRAPH’93 Conference*, pp. 73-80, Addison-Wesley/ACM Press, 1993.

bles90a.T.W. Bleser and J.L. Sibert, ‘‘Toto: A Tool for Selecting Interaction Techniques,’’ *Proc. ACM UIST’90 Symposium on User Interface Software and Technology*, pp. 135-142, Addison-Wesley/ACM Press, Snowbird, Utah, 1990.

boff86a.K.R. Boff, L. Kaufman, and J. Thomas, *Handbook of Perception and Human Performance*, John Wiley, New York, 1986.

bolt81a.R.A. Bolt, ‘‘Gaze-Orchestrated Dynamic Windows,’’ *Computer Graphics*, vol. 15, no. 3, pp. 109-119, August 1981.

buxt83a.W. Buxton, ‘‘Lexical and Pragmatic Considerations of Input Structures,’’ *Computer Graphics*, vol. 17, no. 1, pp. 31-37, 1983.

call88a.J. Callahan, D. Hopkins, M. Weiser, and B. Shneiderman, ‘‘An Empirical Comparison of Pie vs. Linear Menus,’’ *Proc. ACM CHI’88 Human Factors in Computing Systems Conference*, pp. 95-100, Addison-Wesley/ACM Press, 1988.

card78a.S.K. Card, W.K. English, and B.J. Burr, ‘‘Evaluation of Mouse, Rate-controlled Isometric Joystick, Step Keys, and Text Keys for Text Selection on a CRT,’’ *Ergonomics*, vol. 21, no. 8, pp. 601-613, 1978.

card83a.S.K. Card, T.P. Moran, and A. Newell, *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum, Hillsdale, N.J., 1983.

dark96a.R.P. Darken and J.L. Sibert, ‘‘Wayfinding Strategies and Behaviors in Large Virtual Worlds,’’ *Proc. ACM CHI’96 Human Factors in Computing Systems Conference*, pp. 142-149, Addison-Wesley/ACM Press, 1996.

enge68a.D.C. Engelbart and W.K. English, ‘‘A Research Center for Augmenting Human Intellect,’’ *Proc. 1968 Fall Joint Computer Conference*, pp. 395-410, AFIPS, 1968.

fitz95a.G.W. Fitzmaurice, H. Ishii, and W. Buxton, ‘‘Bricks: Laying the Foundations for Graspable User Interfaces,’’ *Proc. ACM CHI’95 Human Factors in Computing Systems*

Conference, pp. 442-449, Addison-Wesley/ACM Press, 1995.

furn95a.G.W. Furnas and B.B. Bederson, ‘‘Space-Scale Diagrams: Understanding Multiscale Interfaces,’’ *Proc. ACM CHI’95 Human Factors in Computing Systems Conference*, pp. 234-241, Addison-Wesley/ACM Press, 1995.

gold93a.D. Goldberg and C. Richardson, ‘‘Touch-Typing with a Stylus,’’ *Proc. ACM INTER-CHI’93 Human Factors in Computing Systems Conference*, pp. 80-87, Addison-Wesley/ACM Press, 1993.

hela88a.M. Helander, *Handbook of Human-Computer Interaction,* Amsterdam, Elsevier North-Holland, 1988.

hern92a.K.P. Herndon, R.C. Zeleznik, D.C. Robbins, D.B. Conner, S.S. Snibbe, and A. van Dam, ‘‘Interactive Shadows,’’ *Proc. ACM UIST’92 Symposium on User Interface Software and Technology*, pp. 1-6, Addison-Wesley/ACM Press, Monterey, Calif., 1992.

hinc94a.K. Hinckley, R. Pausch, J.C. Goble, and N.F. Kassell, ‘‘Passive Real-World Interface Props for Neurosurgical Visualization,’’ *Proc. ACM CHI’94 Human Factors in Computing Systems Conference*, pp. 452-458, Addison-Wesley/ACM Press, 1994.

hinc94b.K. Hinckley, R. Pausch, J.C. Goble, and N.F. Kassell, ‘‘A Survey of Design Issues in Spatial Input,’’ *Proc. ACM UIST’94 Symposium on User Interface Software and Technology*, pp. 213-222, Marina del Rey, Calif., 1994.

hinc99a.K. Hinckley and M. Sinclair, ‘‘Touch-Sensing Input Devices,’’ *Proc. ACM CHI’99 Human Factors in Computing Systems Conference*, pp. 223-230, Addison-Wesley/ACM Press, 1999.

hix95a.D. Hix, J.N. Templeman, and R.J.K. Jacob, ‘‘Pre-Screen Projection: From Concept to Testing of a New Interaction Technique,’’ *Proc. ACM CHI’95 Human Factors in Computing Systems Conference*, pp. 226-233, Addison-Wesley/ACM Press, 1995.

http://www.cs.tufts.edu/~jacob/papers/chi95.pdf [PDF].

ishi97a.H. Ishii and B. Ullmer, ‘‘Tangible Bits: Towards Seamless Interfaces between People, Bits, and Atoms,’’ *Proc. ACM CHI’97 Human Factors in Computing Systems Conference*, pp. 234-241, Addison-Wesley/ACM Press, 1997.

jaco91a.R.J.K. Jacob, ‘‘The Use of Eye Movements in Human-Computer Interaction Techniques: What You Look At is What You Get,’’ *ACM Transactions on Information Systems*, vol. 9, no. 3, pp. 152-169, April 1991.

jaco94a.R.J.K. Jacob, L.E. Sibert, D.C. McFarlane, and M.P. Mullen, Jr., ‘‘Integrality and Separability of Input Devices,’’ *ACM Transactions on Computer-Human Interaction*, vol. 1, no. 1, pp. 3-26, March 1994. http://www.cs.tufts.edu/~jacob/papers/tochi.txt [ASCII]; http://www.cs.tufts.edu/~jacob/papers/tochi.pdf [PDF].

jaco96a.R.J.K. Jacob, ‘‘Input Devices and Techniques,’’ in *The Computer Science and Engineering Handbook*, ed. by A.B. Tucker, pp. 1494-1511, CRC Press, 1996. http://www.cs.tufts.edu/~jacob/papers/crc.html [HTML]; http://www.cs.tufts.edu/~jacob/papers/crc.pdf [PDF].

jell90a.H.D. Jellinek and S.K. Card, ‘‘Powermice and User Performance,’’ *Proc. ACM CHI’90 Human Factors in Computing Systems Conference*, pp. 213-220, Addison-Wesley/ACM Press, 1990.

klat87a.D.H. Klatt, ‘‘Review of text-to-speech conversion for English,’’ *Journal of the Acoustical Society of America*, vol. 82, no. 3, pp. 737-793, 1987.

mack90a.J.D. Mackinlay, S.K. Card, and G.G. Robertson, ‘‘A Semantic Analysis of the Design Space of Input Devices,’’ *Human-Computer Interaction*, vol. 5, pp. 145-190, 1990.

maes95a.P. Maes, ‘‘Artificial Life Meets Entertainment: Lifelike Autonomous Agents,’’ *Communications of the ACM*, vol. 38, no. 11, pp. 108-114, November 1995.

mats97a.N. Matsushita and J. Rekimoto, ‘‘HoloWall: Designing a Finger, Hand, Body, and Object Sensitive Wall,’’ *Proc. ACM UIST’97 Symposium on User Interface Software and Technology*, pp. 209-210, Addison-Wesley/ACM Press, Banff, Canada, 1997.

mayh99a.D.J. Mayhew, *The Usability Engineering Lifecycle*, Morgan Kaufmann, San Francisco, 1999.

mine97a.M.R. Mine, F.P. Brooks, and C.H. Sequin, ‘‘Moving Objects in Space: Exploiting Proprioception in Virtual-Environment Interaction,’’ *Proc. ACM SIGGRAPH’97 Conference*, pp. 19-26, Addison-Wesley/ACM Press, 1997.

moha96a.M. Mohageg, R. Myers, C. Marrin, J. Kent, D. Mott, and P. Isaacs, ‘‘A User Interface for Accessing 3D Content on the World Wide Web,’’ *Proc. ACM CHI’96 Human Factors in Computing Systems Conference*, pp. 466-472, Addison-Wesley/ACM Press, 1996.

mora95a.T.P. Moran, P. Chiu, W. van Melle, and G. Kurtenbach, ‘‘Implicit Structure for Pen-based Systems Within a Freeform Interaction Paradigm,’’ *Proc. ACM CHI’95 Human Factors in Computing Systems Conference*, pp. 487-494, Addison-Wesley/ACM Press, 1995.

newm92a.W. Newman and P. Wellner, ‘‘A Desk Supporting Computer-based Interaction with Paper Documents,’’ *Proc. ACM CHI’92 Human Factors in Computing Systems Conference*, pp. 587-592, Addison-Wesley/ACM Press, 1992.

norm91a.K. Norman, *The Psychology of Menu Selection*, Ablex Publishing Co., Norwood, N.J., 1991.

pear86a.G. Pearson and M. Weiser, ‘‘Of Moles and Men: The Design of Foot Control for Workstations,’’ *Proc. ACM CHI’86 Human Factors in Computing Systems Conference*, pp. 333-339, 1986.

perl93a.K. Perlin and D. Fox, ‘‘Pad: An Alternative Approach to the Computer Interface,’’ *Proc. ACM SIGGRAPH’93 Conference*, pp. 57-64, Addison-Wesley/ACM Press, 1993.

perl84a.G. Perlman, ‘‘Making the Right Choices with Menus,’’ *Proc. IFIP INTERACT’84 Conference on Human-Computer Interaction*, pp. 317-321, 1984.

pree94a.J. Preece, Y. Rogers, H. Sharp, and D. Benyon, *Human-Computer Interaction,* Addison-Wesley, Reading, Mass., 1994.

rabi93a.L. Rabiner and B. Juang, *Fundamentals of Speech Recognition,* Prentice-Hall, Englewood Cliffs, N.J., 1993.

reki97a.J. Rekimoto, ‘‘Pick-and-Drop: A Direct Manipulation Technique for Multiple Computer Environments,’’ *Proc. ACM UIST’97 Symposium on User Interface Software and Technology*, pp. 31-39, Addison-Wesley/ACM Press, Banff, Canada, 1997.

rowe98a.D.W. Rowe, J. Sibert, and D. Irwin, ‘‘Heart Rate Variability: Indicator of User State as an Aid to Human-Computer Interaction,’’ *Proc. ACM CHI’98 Human Factors in Computing Systems Conference*, pp. 480-487, Addison-Wesley/ACM Press, 1998.

rubi91a.D. Rubine, ‘‘Specifying Gestures by Example,’’ *Proc. ACM SIGGRAPH’91 Conference*, pp. 329-337, Addison-Wesley/ACM Press, 1991.

sali99a.J.K. Salisbury, ‘‘Making Graphics Physically Tangible,’’ *Communications of the ACM*, vol. 42, no. 8, pp. 75-81, August 1999.

salv97a.G. Salvendy, *Handbook of Human Factors and Ergonomics,* John Wiley, New York, 1997.

schm83a.C. Schmandt, ‘‘Spatial Input/display Correspondence in a Stereoscopic Computer Graphic Workstation,’’ *Computer Graphics,* vol. 17, no. 3, pp. 253-259, 1983.

schm94a.C. Schmandt, *Voice Communication with Computers,* Van Nostrand Reinhold, New York, 1994.

sear91a.A. Sears and B. Shneiderman, ‘‘High Precision Touchscreens: Design Strategies and Comparison with a Mouse,’’ *International Journal of Man-Machine Studies*, vol. 43, no. 4, pp. 593-613, April 1991.

shne97a.B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction, Third Edition,* Addison-Wesley, Reading, Mass., 1997.

stoa95a.R. Stoakley, M.J. Conway, and R. Pausch, ‘‘Virtual Reality on a WIM: Interactive Worlds in Miniature,’’ *Proc. ACM CHI’95 Human Factors in Computing Systems Conference*, pp. 265-272, Addison-Wesley/ACM Press, 1995.

ston94a.M.C. Stone, K. Fishkin, and E.A. Bier, ‘‘The Movable Filter as a User Interface Tool,’’ *Proc. ACM CHI’94 Human Factors in Computing Systems Conference*, pp. 306-312, Addison-Wesley/ACM Press, 1994.

tapi95a.M.A. Tapia and G. Kurtenbach, ‘‘Some Design Refinements and Principles on the Appearance and Behavior of Marking Menus,’’ *Proc. ACM UIST’95 Symposium on User Interface Software and Technology*, pp. 189-195, Addison-Wesley/ACM Press, Pittsburgh, Pa., 1995.

teit79a.W. Teitelman, ‘‘A Display Oriented Programmer’s Assistant,’’ *International Journal of Man-Machine Studies*, vol. 11, pp. 157-187, 1979.

van95a.J. Van Santen, R. Sproat, J. Olive, and J. Hirshberg, *Progress in Speech Synthesis,* Springer Verlag, New York, 1995.

want99a.R. Want, K.P. Fishkin, A. Gujar, and B.L. Harrison, ‘‘Bridging Physical and Virtual Worlds with Electronic Tags,’’ *Proc. ACM CHI’99 Human Factors in Computing Systems Conference*, pp. 370-377, Addison-Wesley/ACM Press, 1999.

ware94a.C. Ware and R. Balakrishnan, ‘‘Reaching for Objects in VR Displays: Lag and Frame Rate,’’ *ACM Transactions on Computer-Human Interaction*, vol. 1, no. 4, pp. 331-356, December 1994.

ware97a.C. Ware and K. Lowther, ‘‘Selection Using a One-eyed Cursor in a Fish Tank VR Environment,’’ *ACM Transactions on Computer-Human Interaction*, vol. 4, no. 4, pp. 309-322, December 1997.

zhai98a.A. Zhai and P. Milgram, ‘‘Quantifying Coordination in Multiple DOF Movement and its Application to Evaluating 6 DOF Input Devices,’’ *Proc. ACM CHI’98 Human Factors in Computing Systems Conference*, pp. 320-327, Addison-Wesley/ACM Press, 1998.

zhai96a.S. Zhai, W. Buxton, and P. Milgram, ‘‘The Partial-occlusion Effect: Utilizing Semi-transparency in 3D Human-computer interaction,’’ *ACM Transactions on Computer-Human Interaction*, vol. 3, no. 3, pp. 254-284, September 1996.