
USER CENTERED SYSTEM DESIGN

New Perspectives on Human-Computer Interaction

Edited by

DONALD A. NORMAN
STEPHEN W. DRAPER

University of California, San Diego



1986

LAWRENCE ERLBAUM ASSOCIATES, PUBLISHERS
Hillsdale, New Jersey London

There's More to Interaction Than Meets the Eye: Some Issues in Manual Input

WILLIAM BUXTON

Imagine a time far into the future, when all knowledge about our civilization has been lost. Imagine further, that in the course of planting a garden, a fully stocked computer store from the 1980s was unearthed, and that all of the equipment and software was in working order. Now, based on this find, consider what a physical anthropologist might conclude about the physiology of the humans of our era? My best guess is that we would be pictured as having a well-developed eye, a long right arm, a small left arm, uniform-length fingers and a "low-fi" ear. But the dominating characteristics would be the prevalence of our visual system over our poorly developed manual dexterity.

Obviously, such conclusions do not accurately describe humans of the twentieth century. But they would be perfectly warranted based on the available information. Today's systems have severe shortcomings when it comes to matching the physical characteristics of their operators. Admittedly, in recent years there has been a great improvement in matching computer output to the human visual system. We see this in the improved use of visual communication through typography, color, animation, and iconic interfacing. Hence, our speculative future anthropologist would be correct in assuming that we had fairly well-developed (albeit monocular) vision.

Valve Exhibit 1031 Valve v. Immersion
--

In our example, it is with the human's effectors (arms, legs, hands, etc.) that the greatest distortion occurs. Quite simply, when compared to other human-operated machinery (such as the automobile), today's computer systems make extremely poor use of the potential of the human's sensory and motor systems. The controls on the average user's shower are probably better human-engineered than those of the computer on which far more time is spent. There are a number of reasons for this situation. Most of them are understandable, but none of them should be acceptable.

My thesis is that we can achieve user interfaces that are more natural, easier to learn, easier to use, and less prone to error if we pay more attention to the "body language" of human-computer dialogues. I believe that the quality of human input can be greatly improved through the use of *appropriate gestures*. In order to achieve such benefits, however, we must learn to match human physiology, skills, and expectations with our systems' physical ergonomics, control structures, and functional organization.

In this chapter I look at manual input with the hope of developing a better understanding of how we can better tailor input structures to fit the human operator.

A FEW WORDS ON APPROACH

Due to constraints on space, I restrict myself to the discussion of manual input. I do so fully realizing that most of what I say can be applied to other parts of the body, and I hope that the discussion will encourage the reader to explore other types of transducers.

Just consider the use of the feet in sewing, driving an automobile, or in playing the pipe organ. Now compare this to your average computer system. The feet are totally ignored despite the fact that most users have them, and furthermore, have well-developed motor skills in their use.

I resist the temptation to discuss new and exotic technologies. I want to stick with devices that are real and available, since we haven't come close to using the full potential of those that we already have.

Finally, my approach is somewhat cavalier. I will leap from example to example, and just touch on a few of the relevant points. In the process, it is almost certain that readers will be able to come up with examples counter to my own, and situations where what I say does not apply. *But these contradictions strengthen my argument!* Input is complex, and deserves great attention to detail: more than it generally gets.

That the grain of my analysis is still not fine enough just emphasize how much more we need to understand.

Managing input is so complex that it is unlikely that we will ever totally understand it. No matter how good our theories are, we will probably always have to test designs through actual implementations and prototyping. The consequence of this for the designer is that prototyping tools (software and hardware) must be developed and considered as part of the basic environment.

THE IMPORTANCE OF THE TRANSDUCER

When we discuss user interfaces, consideration of the physical transducers too often comes last, or near last. And yet, the physical properties of the system are those with which the user has the first and most direct contact. This is not just an issue of comfort. Different devices have different properties, and lend themselves to different things. And if gestures are as important as I believe, then we must pay careful attention to the transducers to which we assign them.

An important concept in modern interactive systems is the notion of *device independence*. The idea is that input devices fall into general classes of what are known as *virtual devices*, such as "locators" and "valuators." Dialogues are described in terms of these virtual devices. The objective is to permit the easy substitution of one physical device for another of the same class. One benefit in this is that it facilitates experimentation (with the hopeful consequence of finding the best among the alternatives). The danger, however, is that one can easily be lulled into believing that the technical interchangeability of the devices extends to usability. Wrong! It is always important to keep in mind that even devices within a class have various idiosyncrasies. It is often these very idiosyncratic differences that determine the appropriateness of a device for a given context. So, device independence is a useful concept, but only when additional considerations are made when making choices.

Example 1: The Isometric Joystick

An "isometric joystick" is a joystick whose handle does not move when it is pushed. Rather, its shaft senses how hard you are pushing it, and in what direction. It is, therefore, a pressure-sensitive device. Two isometric joysticks are shown in Figure 15.1. They are both made

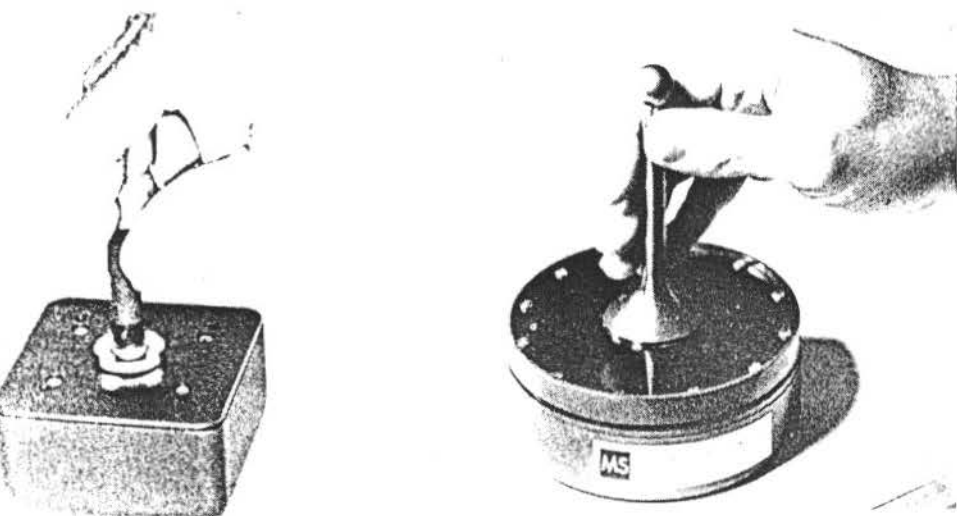


FIGURE 15.1. Two isometric joysticks. (Measurement Systems, Inc.)

the same manufacturer. They cost about the same, and are electronically identical. In fact, they are plug compatible. How they differ is in their size, the muscle groups that they consequently employ, and the amount of force required to get a given output.

Remember, people generally discuss joysticks vs. mice or trackballs. Here we are not only comparing joysticks against joysticks, we are comparing one isometric joystick to another.

When should one be used rather than the other? The answer obviously depends on the context. What can be said is that their differences may often be more significant than their similarities. In the absence of one of the pair, it may be better to utilize a completely different type of transducer (such as a mouse) than to use the other isometric joystick.

Example 2: Joystick vs. Trackball

Let's take an example in which subtle idiosyncratic differences have a strong effect on the appropriateness of the device for a particular transaction. In this example we look at two different devices. One is the

springloaded joystick shown in Figure 15.2A. In many ways, it is very similar to the isometric joysticks seen in the previous example. It is made by the same manufacturer, and it is plug-compatible with respect to the X/Y values that it transmits. However, this new joystick moves when it is pushed, and (as a result of spring action) returns to the center position when released. In addition, it has a third dimension of control accessible by manipulating the self-returning, spring-loaded rotary pot mounted on the top of the shaft.

Rather than contrasting this to the joysticks of the previous example (which would, in fact, be a useful exercise), let us compare it to the 3-D trackball shown in Figure 15.2B. (A 3-D trackball is a trackball constructed so as to enable us to sense clockwise and counter-clockwise "twisting" of the ball as well as the amount that it has been "rolled" in the horizontal and vertical directions.)

This trackball is plug-compatible with the 3-D joystick, costs about the same, has the same "footprint" (consumes the same amount of desk space), and utilizes the same major muscle groups. It has a great deal in common with the 3-D joystick of Figure 15.2A. In many ways the joystick in Figure 15.2A has more in common with the trackball than with the joysticks shown in Figure 15.1!

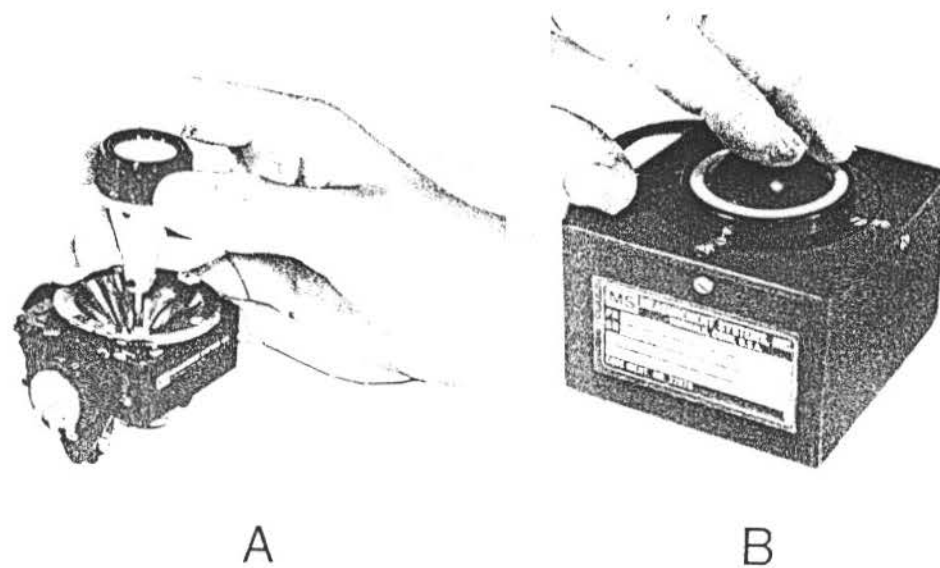


FIGURE 15.2. Comparison of joystick (A) and trackball (B). (Measurement Systems, Inc.)

If you are starting to wonder about the appropriateness of always characterizing input devices by names such as "joystick" or "mouse," then the point of this section is getting across. It is starting to seem that we should lump devices together according to some "dimension of maximum significance," rather than by some (perhaps irrelevant) similarity in their mechanical construction (such as being a mouse or joystick). The prime issue arising from this recognition is the problem of determining which dimension is of maximum significance in a given context. Another is the weakness of our current vocabulary to express such dimensions.

Despite their similarities, these two devices differ in a very subtle, but significant, way. Namely, it is much easier to simultaneously control all three dimensions when using the joystick than when using the trackball. In some applications this will make no difference. But for the moment, we care about instances where it does. We look at two scenarios.

Scenario 1: We are working on a graphics program for doing VLSI layout. The chip on which we are working is quite complex. The only way that the entire mask can be viewed at one time is at a very small scale. To examine a specific area in detail, therefore, we must "pan" over it, and "zoom in." With the joystick, we can pan over the surface of the circuit by adjusting the stick position. Panning direction is determined by the direction in which the spring-loaded stick is off-center, and speed is determined by its distance off-center. With the trackball, we exercise control by rolling the ball in the direction and at the speed that we want to pan.

Panning is easier with trackball than the spring-loaded joystick. This is because of the strong correlation (or compatibility) between stimulus (direction, speed, and amount of roll) and response (direction, speed, and amount of panning) in this example. With the spring-loaded joystick, there was a position-to-motion mapping rather than the motion-to-motion mapping seen with the trackball. Such cross-modality mappings require learning and impede achieving optimal human performance. These issues address the properties of an interface that Hutchins, Hollan, and Norman (Chapter 5) call "formal directions."

If our application demands that we be able to zoom and pan simultaneously, then we have to reconsider our evaluation. With the joystick, it

is easy to zoom in and out of regions of interest while panning. One need only twist the shaft-mounted pot while moving the stick. However, with the trackball, it is nearly impossible to twist the ball at the same time that it is being rolled. The 3-D trackball is, in fact, better described as a 2+1D device.

Scenario 2: I am using the computer to control an oil refinery. The pipes and valves of a complex part of the system are shown graphically on the displays, along with critical status information. My job is to monitor the status information and, when conditions dictate, modify the system by adjusting the settings of specific valves. I do this by means of *direct manipulation*. That is, valves are adjusted by manipulating their graphical representation on the screen. Using the joystick, this is accomplished by pointing at the desired valve, then twisting the pot mounted on the stick. However, it is difficult to twist the joystick-pot without also causing some change in the X and Y values. This causes problems, since graphics pots may be in close proximity on the display. Using the trackball, however, the problem does not occur. In order to twist the trackball, it can be (and is best) gripped so that the finger tips rest against the bezel of the housing. The finger tips thus prevent any rolling of the ball. Hence, twisting is orthogonal to motion in X and Y. The trackball is the better transducer in this example *precisely because of its idiosyncratic 2+1D property*.

Thus, we have seen how the very properties that gave the joystick the advantage in the first scenario were a liability in the second. Conversely, with the trackball, we have seen how the liability became an advantage. What is to be learned here is that if such cases exist between these two devices, then it is most likely that comparable (but different) cases exist among all devices. What we are most lacking is some reasonable methodology for exploiting such characteristics via an appropriate matching of device idiosyncrasies with structures of the dialogue.

APPROPRIATE DEVICES CAN SIMPLIFY SYNTAX

In the previous example we saw how the idiosyncratic properties of an input device could have a strong affect on its appropriateness for a specific task. It would be nice if the world was simple, and we could consequently figure out what a system was for, find the optimal device for the task to be performed on it, and be done. But such is seldom

the case. Computer systems are more often used by a number of people for a number of tasks, each with their own demands and characteristics. One approach to dealing with the resulting diversity of demands is to supply a number of input devices, one optimized for each type of transaction. However, the benefits of the approach would generally break down as the number of devices increased. Usually, a more realistic solution is to attempt to get as much generality as possible from a smaller number of devices. Devices, then, are chosen for their range of applicability. This is, for example, a major attraction of graphics tablets. They can emulate the behavior of a mouse. But unlike the mouse, they can also be used for tracing artwork to digitize it into the machine.

Having raised the issue, I continue to discuss devices in such a way as to focus on their idiosyncratic properties. Why? Because by doing so, I hope to identify the type of properties that one might try to emulate, should emulation be required.

It is often useful to consider the user interface of a system as being made up of a number of horizontal layers. Most commonly, syntax is considered separately from semantics, and lexical issues independent from syntax. Much of this way of analysis is an outgrowth of the theories practiced in the design and parsing of artificial languages, such as in the design of compilers for computer languages. Thinking of the world in this way has many benefits, not the least of which is helping to avoid "apples-and-bananas" type comparisons. There is a problem, however, in that it makes it too easy to fall into the belief that each of these layers is independent. A major objective of this section is to point out how wrong an assumption this is. In particular, I illustrate how decisions at the lowest level, the choice of input devices, can have a pronounced effect on the complexity of the system and on the user's model.

Example 2: Two children's toys. The *Etch-a-Sketch* (shown in Figure 15.3A) is a children's drawing toy that has had a remarkably long life in the marketplace. One draws by manipulating the controls so as to cause a stylus on the back of the drawing surface to trace out the desired image. There are only two controls: Both are rotary pots. One controls left-right motion of the stylus and the other controls its up-down motion.

The *Skedoodle* (shown in Figure 15.3B) is another toy based on very similar principles. In *computerese*, we could even say that the two toys are semantically identical. They draw using a similar stylus mechanism and even have the same "erase" operator (turn the toy upside down and shake it). However, there is one big difference. Whereas the *Etch-a-*

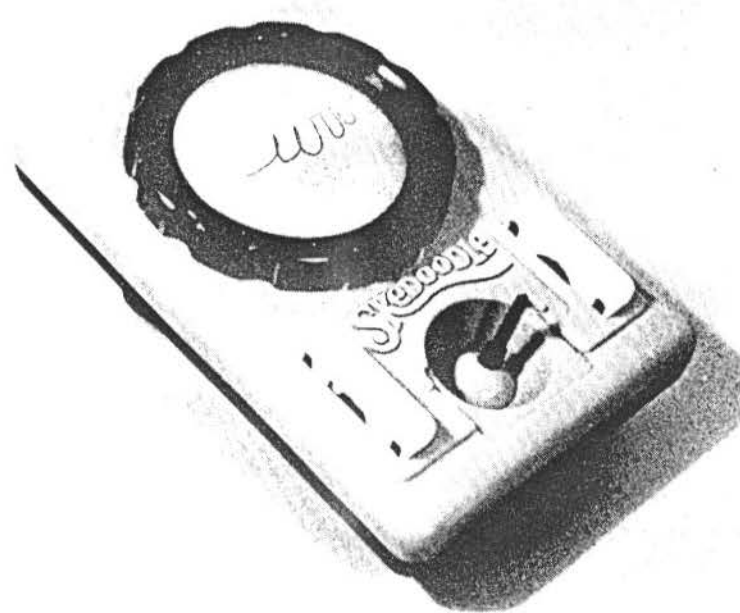
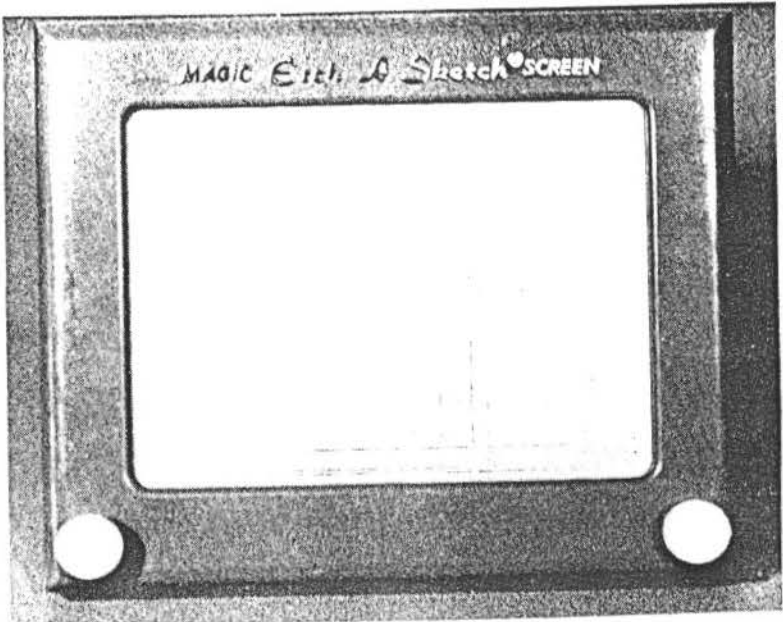


FIGURE 15.3. Two "semantically identical" drawing toys.

Sketch has a separate control for each of the two dimensions of control, the Skedoodle has integrated both dimensions into a single transducer: a joystick.

Since both toys are inexpensive and widely available, they offer an excellent opportunity to conduct some field research. Find a friend and demonstrate each of the two toys. Then ask the friend to select the toy felt to be the best for drawing. What all this is leading to is a drawing competition between you and your friend. However, this is a competition that you will always win. The catch is that since your friend got to choose toys, you get to choose what is drawn. If your friend chose the Skedoodle (as do the majority of people), then make the required drawing be of a horizontally-aligned rectangle. If they chose the Etch-a-Sketch, then have the task be to write your first name. This test has two benefits. First, if you make the competition a bet, you can win back the money that you spent on the toys (an unusual opportunity in research). Second, you can do so while raising the world's enlightenment about the sensitivity of the quality of input devices to the task to which they are applied.

If you understand the importance of the points being made here, you are hereby requested to go out and apply this test on every person that you know who is prone to making unilateral and dogmatic statements of the variety "mice (tablets, joysticks, trackballs, etc.) are best." What is true with these two toys (as illustrated by the example) is equally true for any and all computer input devices: They all shine for some task.

We can build upon what we have seen thus far. What if we asked how we can make the Skedoodle do well at the same class of drawings as the Etch-a-Sketch? An approximation to a solution actually comes with the toy in the form of a set of templates that fit over the joystick (Figure 15.4). The point to make here is that if we have a general-purpose input device (analogous to the joystick of the Skedoodle), then we can provide tools to fit on top of it to customize it for a specific application. (An example would be the use of "sticky" grids in graphics layout programs.) However, this additional level *generally comes at the expense of increased cost in the complexity of the control structure*. If we don't need the generality, then we can often avoid this complexity by choosing a transducer whose operational characteristics implicitly channel user behavior in the desired way (in a way analogous to how the Etch-a-Sketch controls place constraints on what can be easily drawn).

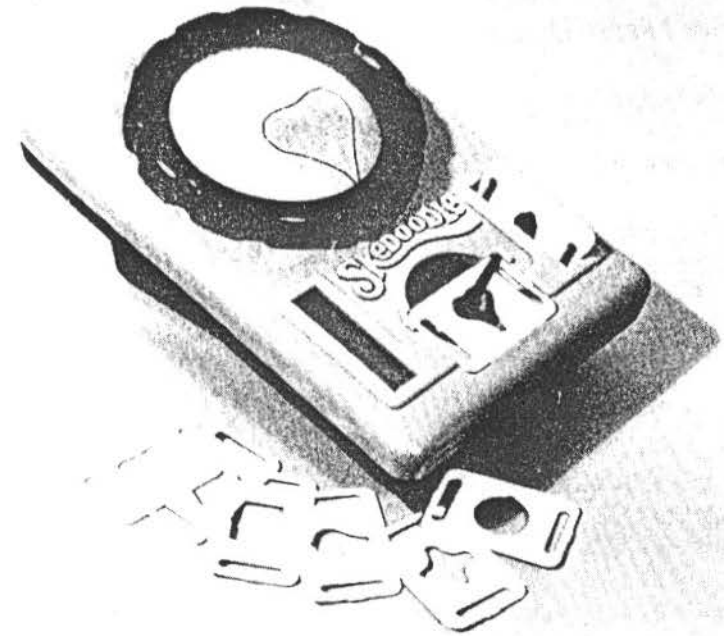


FIGURE 15.4. Adding constraints to an input device. Templates on a Skedoodle joystick.

Example 4: The nulling problem. One of the most important characteristics of input devices is whether they supply *absolute* or *relative* values to the program with which they are interacting. Mice and trackballs, for example, provide relative values. Other devices, such as tablets, touch screens, and potentiometers return absolute values (determined by their measured position). Earlier, I mentioned the importance of the concept of the "dimension of maximum importance." In this example, the choice between absolute *versus* relative mode defines that dimension.

The example comes from process control. There are (at least) two philosophies of design that can be followed in such applications. In the first, space multiplexing, there is a dedicated physical transducer for every parameter that is to be controlled. In the second, time multiplexing, there are fewer transducers than parameters. Such systems are designed so that a single device can be used to control different

parameters at different stages of an operation.

Let us assume that we are implementing a system based on time multiplexing. There are two parameters, A and B, and a single sliding potentiometer to control them, P. The potentiometer P outputs an absolute value proportional to the position of its handle. To begin with, the control potentiometer is set to control parameter A. The initial settings of A, B, and P are all illustrated in Figure 15.5A. First we want to raise the value of A to its maximum. This we do simply by sliding up the controller, P. This leaves us in the state illustrated in Figure 15.5B. We now want to raise parameter B to its maximum value. But how can we raise the value of B if the controller is already in its highest position? Before we can do anything we must adjust the handle of the controller relative to the current value of B. This is illustrated in Figure 15.5C. Once this is done, parameter B can be reset by

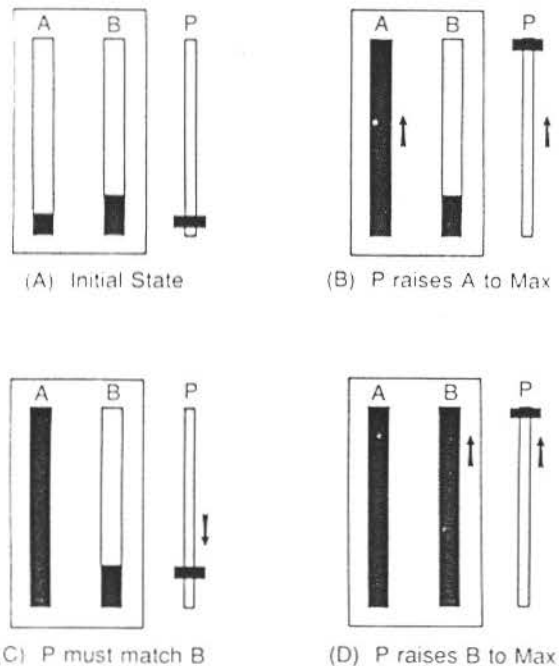


FIGURE 15.5. The Nulling Problem. Potentiometer P controls two parameters, A and B. The initial settings are shown in Panel A. The position of P, after raising parameter A to its maximum value, is shown in Panel B. In order for P to be used to adjust parameter B, it must first be moved to match the value of B (i.e., "null" their difference), as shown in Panels C and D.

adjusting P. The job is done and we are in the state shown in Figure 15.5D.

From an operator's perspective, the most annoying part of the above transaction is having to reset the controller before the second parameter can be adjusted. This is called the *nulling problem*. It is common, takes time to carry out, time to learn, and is a common source of error. Most importantly, it can be totally eliminated if we simply choose a different transducer.

The problems in the last example resulted from the fact that we chose a transducer that returned an absolute value based on a physical handle's position. As an alternative, we could replace it with a touch-sensitive strip of the same size. We will use this strip like a one-dimensional mouse. Instead of moving a handle, the strip is "stroked" up or down using a motion similar to that which adjusted the sliding potentiometer. The output in this case, however, is a value whose magnitude is proportional to the amount and direction of the stroke. In short, we get a relative value which determines the amount of change in the parameter. We simply push values up, or pull them down. The action is totally independent of the current value of the parameter being controlled. There is no handle to get stuck at the top or bottom. The device is like a treadmill, having infinite travel in either direction. In this example, we could have "rolled" the value up and down using one dimension of a trackball and gotten much the same benefit (since it too is a relative device).

An important point in this example is *where* the reduction in complexity occurred: in the syntax of the control language. Here we have a compelling and relevant example of where a simple change in input device has resulted in a significant change in the syntactic complexity of a user interface. The lesson to be learned is that in designing systems in a layered manner—first the semantics, then the syntax, then the lexical component, and the devices—we must take into account interactions among the various strata. *All* components of the system interlink and have a potential effect on the user interface. Systems *must* begin to be designed in an integrated and holistic way.

PHRASING GESTURAL INPUT

Phrasing is a crucial component of speech and music. It determines the ebb and flow of tension in a dialogue. It lets us know when a concept is beginning, and when it ends. It tells us when to be attentive, and when to relax. Why might this be of importance in our discussion of "body language" in human-computer dialogue? Well, for all the same reasons that it is important in all other forms of communication.

Phrases "chunk" related things together. They reinforce their connection. In this section I attempt to demonstrate how we can exploit the benefits of phrasing by building dialogues that enable connected concepts to be expressed by connected physical gestures.

If you look at the literature, you will find that there has been a great deal of study on how quickly humans can push buttons, point at text, and type commands. What the bulk of these studies focus on is the smallest grain of the human-computer dialogue, the *atomic task*. These are the "words" of the dialogue. The problem is, we don't speak in words. We speak in sentences. Much of the problem in applying the results of such studies is that they don't provide much help in understanding how to handle *compound tasks*. My thesis is, if you can say it in words in a single phrase, you should be able to express it to the computer in a single gesture. This binding of concepts and gestures thereby becomes the means of articulating the *unit tasks* of an application.

Most of the tasks which we perform in interacting with computers are compound. In indicating a point on the display with a mouse we think of what we are doing as a single task: picking a point. But what would you have to specify if you had to indicate the same point by typing? Your single-pick operation actually consists of two sub-tasks: specifying an X coordinate and specifying a Y coordinate. You were able to think of the aggregate as a single task because of the appropriate match among transducer, gesture, and context. The desired one-to-one mapping between concept and action has been maintained. My claim is that what we have seen in this simple example can be applied to even higher-level transactions.

Two useful concepts from music that aid in thinking about phrasing are *tension* and *closure*. During a phrase there is a state of tension associated with heightened attention. This is delimited by periods of relaxation that close the thought and state implicitly that another phrase can be introduced by either party in the dialogue. It is my belief that we can reap significant benefits when we carefully design our computer dialogues around such sequences of tension and closure. In manual input, I will want tension to imply muscular tension.

Think about how you interact with pop-up menus with a mouse. Normally you push down the select button, indicate your choice by moving the mouse, and then release the select button to confirm the choice. You are in a state of muscular

tension throughout the dialogue; a state that corresponds exactly with the temporary state of the system. Because of the gesture used, it is impossible to make an error in syntax, and you have a continual active reminder that you are in an uninterruptable temporary state. Because of the gesture used, there is none of the trauma normally associated with being in a mode. That you are in a mode is ironic, since it is precisely the designers of "modeless" systems that make the heaviest use of this technique. The lesson here is that it is not modes per se that cause problems.

In well-structured manual input there is a *kinesthetic* connectivity to reinforce the *conceptual* connectivity of the task. We can start to use such gestures to help develop the role of *muscle memory* as a means through which to provide *mnemonic* aids for performing different tasks. And we can start to develop the notion of *gestural self-consistency* across an interface.

What do graphical potentiometers, pop-up menus, scroll-bars, rubber-band lines, and dragging all have in common? Answer: the potential to be implemented with a uniform form of interaction. Work it out using the pop-up menu protocol given above.

WE HAVE TWO HANDS!

It is interesting that the manufacturers of arcade video games seem to recognize something that the majority of main-stream computer systems ignore: that users are capable of manipulating more than one device at a time in the course of achieving a particular goal. Now this should come as no surprise to anyone who is familiar with driving an automobile. But it would be news to the hypothetical anthropologists that we introduced at the start of the chapter. There are two questions here: "Is anything gained by using two hands?" and "If there is, why aren't we doing it?"

The second question is the easier of the two. With a few exceptions, (the Xerox Star, for example), most systems don't encourage two-handed multiple-device input. First, most of our theories about parsing languages (such as the language of our human-computer dialogue) are only capable of dealing with single-threaded dialogues. Second, there are hardware problems due partially to wanting to do parallel things on a serial machine. Neither of these is unsolvable. But we do need some convincing examples that demonstrate that the time

effort, and expense is worthwhile. So that is what I will attempt to do in the rest of this section.

Example 5: Graphics Design Layout

I am designing a screen to be used in a graphics menu-based system. To be effective, care must be taken in the screen layout. I have to determine the size and placement of a figure and its caption among some other graphical items. I want to use the tablet to preview the figure in different locations and at different sizes in order to determine where it should finally appear. The way that this would be accomplished with most current systems is to go through a cycle of position-scale-position-. . . actions. That is, in order to scale, I have to stop positioning, and vice versa.

This is akin to having to turn off your shower in order to adjust the water temperature.

An alternative design offering more fluid interaction is to position it with one hand and scale it with the other. By using two separate devices I am able to perform both tasks simultaneously and thereby achieve a far more fluid dialogue.

Example 6: Scrolling

A common activity in working with many classes of program is scrolling through data, looking for specific items. Consider scrolling through the text of a document that is being edited. I want to scroll till I find what I'm looking for, then mark it up in some way. With most window systems, this is accomplished by using a mouse to interact with some (usually arcane) scroll bar tool. Scrolling speed is often difficult to control and the mouse spends a large proportion of its time moving between the scroll bar and the text. Furthermore, since the mouse is involved in the scrolling task, any ability to *mouse ahead* (i.e., start moving the mouse towards something before it appears on the display) is eliminated. If a mechanism were provided to enable us to control scrolling with the nonmouse hand, the whole transaction would be simplified.

There is some symmetry here. It is obvious that the same device used to scale the figure in the previous example could be used to scroll the window in this one. Thus, we ourselves would be time-multiplexing the device between the scaling of

examples and the scrolling of this example. An example of space-multiplexing would be the simultaneous use of the scrolling device and the mouse. Thus, we actually have a hybrid type of interface.

Example 7: Financial Modeling

I am using a spread-sheet for financial planning. The method used to change the value in a cell is to point at it with a mouse and type the new entry. For numeric values, this can be done using the numeric keypad or the typewriter keyboard. In most such systems, doing so requires that the hand originally on the mouse moves to the keyboard for typing. Generally, this requires that the eyes be diverted from the screen to the keyboard. Thus, in order to check the result, the user must then visually relocate the cell on a potentially complicated display.

An alternative approach is to use the pointing device in one hand and the numeric keypad in the other. The keypad hand can then remain in home position, and if the user can touch-type on the keypad, the eyes need never leave the screen during the transaction.

Note that in this example the tasks assigned to the two hands are not even being done in parallel. Furthermore, a large population of users—those who have to take notes while making calculations—have developed keypad touch-typing facility in their nonmouse hand (assuming that the same hand is used for writing as for the mouse). So if this technique is viable and presents no serious technical problems, then why is it not in common use? One arguable explanation is that on most systems the numeric keypad is mounted on the same side as the mouse. Thus, physical ergonomics prejudice against the approach.

WHAT ABOUT TRAINING?

Some things are hard to do; they take time and effort before they can be performed at a skilled level. Whenever the issue of two-handed input come up, so does some facsimile of the challenge, "But two-handed actions are hard to coordinate." Well, the point is true. But it is also false! Learning to change gears is hard. So is playing the piano. But on the other hand, we have no trouble turning pages with one hand while writing with the other.

Just because two-handed input is not always suitable is no reason to reject it. The scrolling example described above requires trivial skills,

and it can actually reduce errors and learning time. Multiple-handed input should be one of the techniques considered in design. Only its appropriateness for a given situation can determine if it should be used. In that, it is no different than any other technique in our repertoire.

Example 8: Financial Modeling Revisited

Assume that we have implemented the two-handed version of the spreadsheet program described in Example 7. In order to get the benefits that I suggested, the user would have to be a touch-typist on the numeric keypad. This is a skilled task that is difficult to develop. There is a temptation, then, to say "don't use it." If the program was for school children, then perhaps that would be right. But consider who uses such programs: accountants, for example. Thus, it is reasonable to assume that a significant proportion of the user population *comes to the system with the skill already developed*. By our implementation, we have provided a convenience for those with the skill, without imposing any penalty on those without it—they are no worse off than they would be in the one-handed implementation. *Know your user* is just another (and important) consideration that can be exploited in order to tailor a better user interface.

CONCLUSIONS

I began this chapter by pointing out that there are major shortcomings in our ability to manually enter information into a computer. To this point, input has lagged far behind graphical output. And yet, as some of our examples illustrate, input is of critical importance. If we are to improve the quality of human-computer interfaces we must begin to approach input from two different views. First, we must look inward to the devices and technologies at the finest grain of their detail. One of the main points that I have made is that some of the most potent and useful characteristics of input devices only surface when they are analyzed at a far lower level of detail than has commonly been the case.

Second, we must look outward from the devices themselves to how they fit into a more global, or holistic, view of the user interface. All aspects of the system affect the user interface. Often problems at one level of the system can be easily solved by making a change at some other level. This was shown for example, in the discussion of the nulloing problem.

That the work needs to be done is clear. Now that we've made up our minds about that, all that we have to do is assemble good tools and get down to it. What could be simpler?

SUGGESTED READINGS

The literature on most of the issues that are dealt with in this chapter is pretty sparse. One good source that complements many of the ideas discussed is Foley, Wallace, and Chan (1984). A presentation on the notion of virtual devices can be found in Foley and Wallace (1974). A critique of their use can be found in Baecker (1980). This paper by Baecker is actually part of an important and informative collection of papers on interaction (Guedj, ten Hagen, Hopgood, Tucker, & Duce, 1980).

Some of the notions of "chunking" and phrasing discussed are expanded upon in Buxton (1982) and Buxton, Fiume, Hill, Lee, and Woo (1983). The chapter by Miyata and Norman in this book gives a lot of background on performing multiple tasks, such as in two-handed input. Buxton (1983) presents an attempt to begin to formulate a taxonomy of input devices. This is done with respect to the properties of devices that are relevant to the styles of interaction that they will support. Evans, Tanner, and Wein (1981) do a good job of demonstrating the extent to which one device can emulate properties of another. Their study uses the tablet to emulate a large number of other devices.

A classic study that can be used as a model for experiments to compare input devices can be found in Card, English, and Burr (1978). Another classic study which can serve as the basis for modeling some aspects of performance of a given user performing a given task using given transducers is Card, Moran, and Newell (1980). My discussion in this chapter illustrates now, in some cases, the only way that we can determine answers is by testing. This means prototyping that is often expensive. Buxton, Lamb, Sherman, and Smith (1983) present one example of a tool that can help this process. Olsen, Buxton, Ehrich, Kasik, Rhyne, and Sibert (1984) discusses the environment in which such tools are used. Tanner and Buxton (1985) present a general model of User Interface Management Systems. Finally, Thomas and Hamlin (1983) present an overview of "User Interface Management Tools." Theirs is a good summary of many user interface issues, and has a fairly comprehensive bibliography.

ACKNOWLEDGMENTS

This chapter was written during a work period at Xerox Palo Alto Research Center. During its preparation I had some very helpful input from a number of people, especially Stu Card, Jerry Farrell, Lissa Monty, and Peter Tanner. The other authors of this book also made some insightful and useful comments. To them all, I offer my thanks.

- Baecker, R. (1980). Towards an Effective Characterization of Graphical Interaction, pp 127 - 148 in Guedj et al (Eds), (1980), *Methodology of Interaction*, Amsterdam, North Holland Publishing.
- Buxton, W. (1982). An Informal Study of Selection-Positioning Tasks. *Proceedings of Graphics Interface '82*, Toronto, May 1982, pp. 323 - 328.
- Buxton, W. (1983). Lexical and Pragmatic Issues of Input Structures. *Computer Graphics* 17 (1), 31 - 37.
- Buxton, W., Fiume, E., Hill, R., Lee, A. & Woo, C. (1983). Continuous Hand-Gesture Driven Input. *Proceedings of Graphics Interface '83*, Edmonton, May 1983, pp. 191 - 195.
- Buxton, W., Lamb, M., Sherman, D. & Smith, K.C. (1983). Towards a Comprehensive User Interface Management System. *Computer Graphics* 17(3), 35 - 42.
- Card, S., English, W. & Burr, B. (1978). Evaluation of Mouse, Rate-Controlled Isometric Joystick, Step Keys, and Text Keys. *Ergonomics* 8, 601 - 613.
- Card, S., Moran, T. & Newell, A. (1980). The Keystroke-Level Model for User Performance Time with Interactive Systems. *Communications of the ACM* 23 (7), 396 - 410.
- Evans, K, Tanner, P. & Wein, M. (1981). Tablet-Based Valuator that Produce One, Two or Three Degrees of Freedom. *Computer Graphics* 15 (3), 91 - 97.
- Foley, J.D. & Wallace, V.L. (1974). The Art of Graphic Man-Machine Conversation. *Proc. IEEE*, 62 (4), 462 - 470.
- Foley, J.D., Wallace, V.L. & Chan, P. (1984). The Human Factors of Computer Graphics Interaction Techniques. *IEEE Computer Graphics and Applications*, 4 (11), 13 - 48.
- Guedj, R.A., ten Hagen, P., Hopgood, F.R., Tucker, H. and Duce, D.A. (Eds.), (1980), *Methodology of Interaction*, Amsterdam, North Holland Publishing.
- Olsen, D. R., Buxton, W., Ehrich, R., Kasik, D., Rhyne, J. & Sibert, J. (1984). A Context for User Interface Management. *IEEE Computer Graphics and Applications* 4(12), 33 - 42.
- Thomas, J. & Hamlin, G. (Eds.) (1983). Graphical Input Interaction Technique (GIIT) Workshop Summary, *Computer Graphics*, 17 (1), 5 - 30.