Direct manipulation systems offer the satisfying experience of operating on visible objects. The computer becomes transparent, and users can concentrate on their tasks.

Direct Manipulation: A Step Beyond Programming Languages

Ben Shneiderman, University of Maryland

> Leibniz sought to make the form of a symbol reflect its content. "In signs," he wrote, "one sees an advantage for discovery that is greatest when they express the exact nature of a thing briefly and, as it were, picture it; then, indeed, the labor of thought is wonderfully diminished."
>
> Frederick Kreiling, "Leibniz," *Scientific American*, May 1968

Certain interactive systems generate glowing enthusiasm among users—in marked contrast with the more common reaction of grudging acceptance or outright hostility. The enthusiastic users' reports are filled with positive feelings regarding

- mastery of the system,
- competence in the performance of their task,
- ease in learning the system originally and in assimilating advanced features,
- confidence in their capacity to retain mastery over time,
- enjoyment in using the system,
- eagerness to show it off to novices, and
- desire to explore more powerful aspects of the system.

These feelings are not, of course, universal, but the amalgam does convey an image of the truly pleased user. As I talked with these enthusiasts and examined the systems they used, I began to develop a model of the features that produced such delight. The central ideas seemed to be visibility of the object of interest; rapid, reversible, incremental actions; and replacement of complex command language syntax by direct manipulation of the object of interest—hence the term "direct manipulation."

Examples of direct manipulation systems

No single system has all the attributes or design features that I admire—that may be impossible—but those described below have enough to win the enthusiastic support of many users.

Display editors. "Once you've used a display editor, you'll never want to go back to a line editor. You'll be spoiled." This reaction is typical of those who use full-page display editors, who are great advocates of their systems over line-oriented text editors. I heard similar comments from users of stand-alone word processors such as the Wang system and from users of display editors such as EMACS on the MIT/Honeywell Multics system or "vi" (for visual editor) on the Unix system. A beaming advocate called EMACS "the one true editor."

Roberts[1] found that the overall performance time of display editors is only half that of line-oriented editors, and since display editors also reduce training time, the evidence supports the enthusiasm of display editor devotees. Furthermore, office automation evaluations consistently favor full-page display editors for secretarial and executive use.

The advantages of display editors include

Display of a full 24 to 66 lines of text. This full display enables viewing each sentence in context and simplifies reading and scanning the document. By contrast, the

---

A portion of this article was derived from the author's keynote address at the NYU Symposium on User Interfaces, "The Future of Interactive Systems and the Emergence of Direct Manipulation," published in *Human Factors in Interactive Computer Systems*, Y. Vassiliou, ed., Ablex Publishing Co., Norwood, N.J., 1983.

0018-9162/83/0800-0057$01.00 © 1983 IEEE

one-line-at-a-time view offered by line editors is like seeing the world through a narrow cardboard tube.

Display of the document in its final form. Eliminating the clutter of formatting commands also simplifies reading and scanning the document. Tables, lists, page breaks, skipped lines, section headings, centered text, and figures can be viewed in the form that will be printed. The annoyance and delay of debugging the format commands is eliminated because the errors are immediately apparent.

Cursor action that is visible to the user. Seeing an arrow, underscore, or blinking box on the screen gives the operator a clear sense of where to focus attention and apply action.

Cursor motion through physically obvious and intuitively natural means. Arrow keys or devices such as a mouse, joystick, or graphics tablet provide natural physical mechanisms for moving the cursor. This is in marked contrast with commands such as UP 6, which require an operator to convert the physical action into correct syntactic form and which may be difficult to learn, hard to recall, and a source of frustrating errors.

Labeled buttons for action. Many display editors have buttons etched with commands such as INSERT, DELETE, CENTER, UNDERLINE, SUPERSCRIPT, BOLD, or LOCATE. They act as a permanent menu selection display, reminding the operator of the features and obviating memorization of a complex command-lan-

```
EDIT --- SPFDEMO.MYLIB.PLI(COINS) - 01.04 ------------------ COLUMNS 001 072
COMMAND INPUT ===>                                            SCROLL ===> HALF
****** ************************** TOP OF DATA ******************************
000100  COINS:
000200    PROCEDURE OPTIONS (MAIN);
000300      DECLARE
000400        COUNT    FIXED BINARY (31) AUTOMATIC INIT (1),
000500        HALVES   FIXED BINARY (31),
000600        QUARTERS FIXED BINARY (31),
000700        DIMES    FIXED BINARY (31),
I3            NICKELS  FIXED BINARY (31),
000900        SYSPRINT FILE STREAM OUTPUT PRINT;
001000      DO HALVES = 100 TO 0 BY -50;
001100        DO QUARTERS = (100 - HALVES) TO 0 BY -25;
001200          DO DIMES = ((100 - HALVES - QUARTERS)/10)*10 TO 0 BY -10;
001300            NICKELS = 100 - HALVES - QUARTERS - DIMES;
D _               PUT FILE(SYSPRINT) DATA(COUNT,HALVES,QUARTERS,DIMES,NICKELS);
001500            COUNT = COUNT + 1;
001600          END;
001700        END;
001800      END;
001900    END COINS;
****** ************************** BOTTOM OF DATA ****************************
```

```
EDIT --- SPFDEMO.MYLIB.PLI(COINS) - 01.04 ------------------ COLUMNS 001 072
COMMAND INPUT ===>                                            SCROLL ===> HALF
****** ************************** TOP OF DATA ******************************
000100  COINS:
000200    PROCEDURE OPTIONS (MAIN);
000300      DECLARE
000400        COUNT    FIXED BINARY (31) AUTOMATIC INIT (1),
000500        HALVES   FIXED BINARY (31),
000600        QUARTERS FIXED BINARY (31),
000700        DIMES    FIXED BINARY (31),
000800        NICKELS  FIXED BINARY (31),
''''''        _
''''''
''''''
000900        SYSPRINT FILE STREAM OUTPUT PRINT;
001000      DO HALVES = 100 TO 0 BY -50;
001100        DO QUARTERS = (100 - HALVES) TO 0 BY -25;
001200          DO DIMES = ((100 - HALVES - QUARTERS)/10)*10 TO 0 BY -10;
001300            NICKELS = 100 - HALVES - QUARTERS - DIMES;
001500            COUNT = COUNT + 1;
001600          END;
001700        END;
001800      END;
001900    END COINS;
****** ************************** BOTTOM OF DATA ****************************
```

Figure 1. This example from the IBM SPF display editor shows 19 lines of a PL/I program. The commands to insert three lines (I3) and to delete one line (D or D1) are typed on the appropriate lines in the first screen display. Pressing ENTER causes commands to be executed and the cursor to be placed at the beginning of the inserted line. New program statements can be typed directly in their required positions. Control keys move the cursor around the text to positions where changes are made by overstriking. A delete key causes the character under the cursor to be deleted and the text to the left to be shifted over. After pressing an insert key, the user can type text in place. Programmed function keys allow movement of the window forwards, backwards, left, and right over the text. (Examples courtesy of IBM.)

guage syntax. Some editors provide basic functionality with only 10 or 15 labeled buttons, and a specially marked button may be the gateway to advanced or infrequently used features offered on the screen in menu form.

Immediate display of the results of an action. When a button is pressed to move the cursor or center the text, the results appear on the screen immediately. Deletions are apparent at once, since the character, word, or line is erased and the remaining text rearranged. Similarly, insertions or text movements are shown after each keystroke or function button press. Line editors, on the other hand, require a print or display command before the results of a change can be seen.

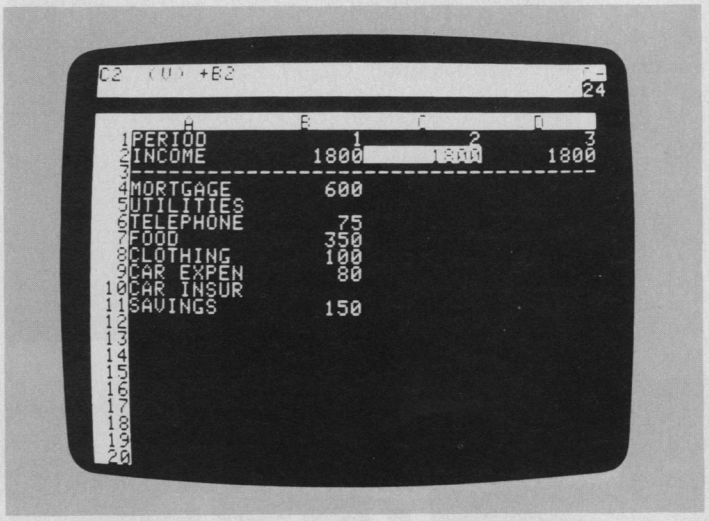
Rapid action and display. Most display editors are designed to operate at high speeds: 120 characters per second (1200 baud), a full page in a second (9600 baud), or even faster. This high display rate coupled with short response time produces a thrilling sense of power and speed. Cursors can be moved quickly, large amounts of text can be scanned rapidly, and the results of commands can be shown almost instantaneously. Rapid action also reduces the need for additional commands, thereby simplifying product design and decreasing learning time. Line editors operating at 30 characters per second with three- to eight-second response times seem sluggish in comparison. Speeding up line editors adds to their attractiveness, but they still lack features such as direct overtyping, deletion, and insertion.

Easily reversible commands. Mistakes in entering text can be easily corrected by backspacing and overstriking. Simple changes can be made by moving the cursor to the problem area and overstriking, inserting, or deleting characters, words, or lines. A useful design strategy is to include natural inverse operations for each operation. Carroll[2] has shown that congruent pairs of operations are easy to learn. As an alternative, many display editors offer a simple UNDO command that cancels the previous command or command sequence and returns the text to its previous state. This easy reversibility reduces user anxiety about making mistakes or destroying a file.

The large market for display editors generates active competition, which accelerates evolutionary design refinements. Figure 1 illustrates the current capabilities of an IBM display editor.

Visicalc. Visicorp's innovative financial forecasting program, called Visicalc, was the product of a Harvard MBA student, who was frustrated by the time needed to carry out multiple calculations in a graduate business course. Described as an "instantly calculating electronic worksheet" in the user's manual, it permits computation and display of results across 254 rows and 63 columns and is programmed without a traditional procedural control structure. For example, positional declarations can prescribe that column 4 displays the sum of columns 1 through 3; then every time a value in the first three columns changes, the fourth column changes as well. Complex dependencies among manufacturing costs, distribution costs, sales revenue, commissions, and profits can be stored for several sales districts and months so that the impact of changes on profits is immediately apparent.

Since Visicalc simulates an accountant's worksheet, it is easy for novices to comprehend. The display of 20 rows and up to nine columns, with the provision for multiple windows, gives the user sufficient visibility to easily scan information and explore relationships among entries (see Figure 2). The command language for setting up the worksheet can be tricky for novices to learn and for infrequent users to remember, but most users need learn only the basic commands. According to Visicalc's distributor, "It jumps," and the user's delight in watching this propagation of changes cross the screen helps explain its appeal.



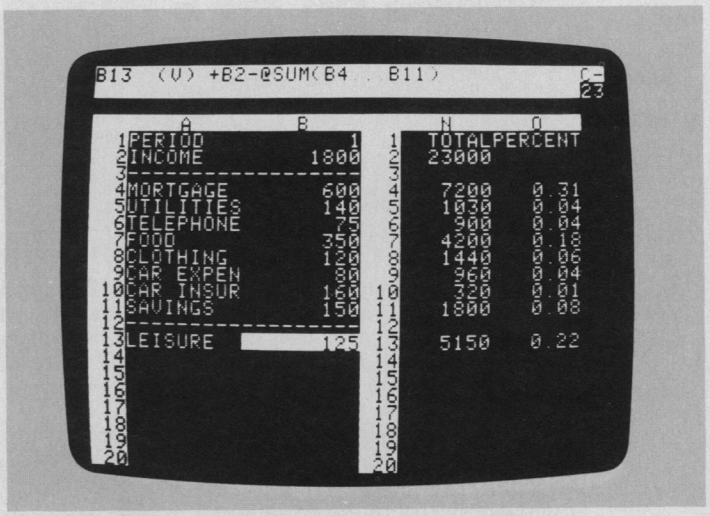
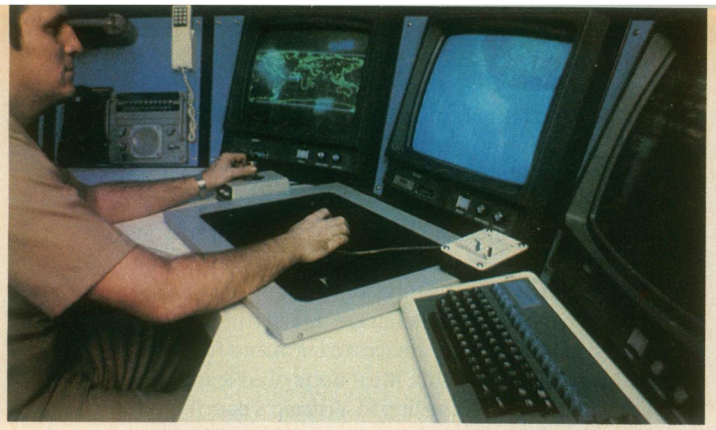
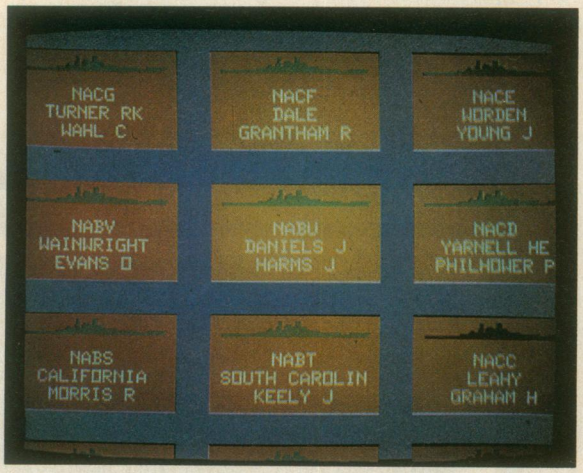


Figure 2. This simple Visicalc program display (top) shows four columns and 20 rows of home budget information. The cursor, an inverse video light bar controlled by key presses, is in position C2. The top command line shows that C2 is a value (as opposed to a text string) that has been set up to have the same value as position B2.

The second display (above) shows two windows over the home budget data with row sums to the right. The last row shows leisure dollar amounts, which are established by the top command line formula as the income minus the sum of expenses. A change to the income or expense values would immediately propagate to all affected values. (Displays reproduced by permission of Visicorp.)



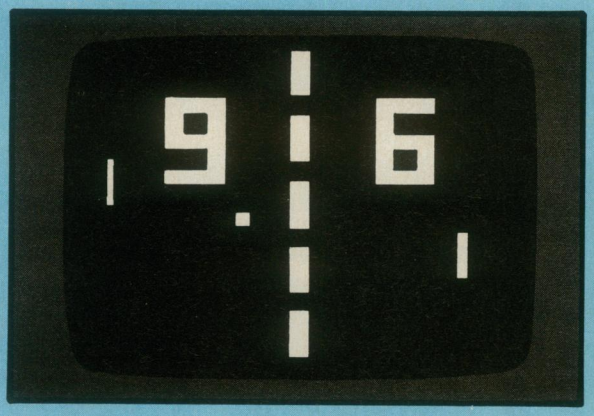


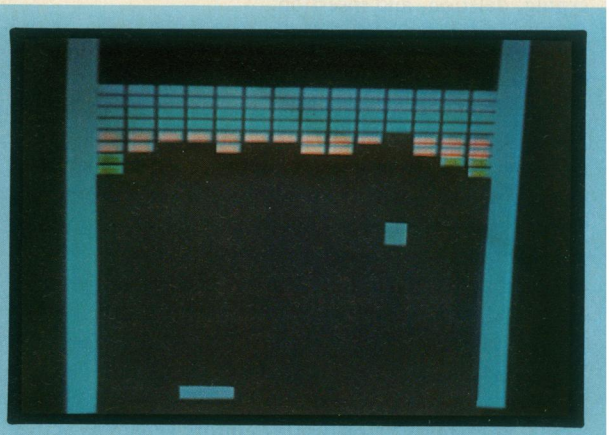
Spatial data management. The developers of the prototype spatial data management system[3] attribute the basic idea to Nicholas Negroponte of MIT.

In one scenario, a user seated before a color graphics display of the world zooms in on the Pacific to see markers for military ship convoys. Moving a joystick fills the screen with silhouettes of individual ships, which can be zoomed in on to display structural details or, ultimately, a full-color picture of the captain. (See Figure 3.)

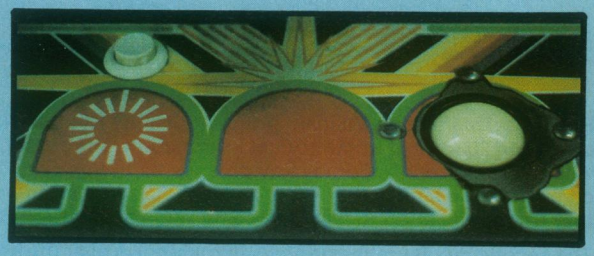
In another scenario, icons representing different aspects of a corporation, such as personnel, organization, travel, production, or schedules, are shown on a screen. Moving the joystick and zooming in on objects takes users through complex "information spaces" or "I-spaces" to locate the item of interest. For example, when they select a department from a building floor

Figure 3. A spatial data management system has been installed on the aircraft carrier USS *Carl Vinson*. In the photo at top left, the operator has a world map on the left screen and a videodisc map of selected areas on the center screen. After some command selections with the data tablet and puck, the operator can zoom in on specific data such as the set of ships shown in the second photo. With further selections the operator can get detailed information about each ship, such as the length, speed, and fuel. (Photos courtesy of Computer Corporation of America.)





In 1971, about the only people playing video games were students in computer science laboratories. By 1973, however, millions of people were familiar with at least one video game—Pong (above left). A few years later came Breakout (above right), which, according to many designers, was the first true video game and the best one ever invented. Pong and other early games imitated real life, but Breakout could not have existed in any medium other than video. In the game, a single paddle directed a ball toward a wall of color bricks; contact made a brick vanish and changed the ball's speed.



When the first arcade video game, Computer Space, went on location in a Sears store, its joystick was torn off before the end of the first day. As a result, game designers have sought controls that were both easy to use and hard to destroy. Centipede (above left) uses simple controls—a trackball and one button. On the other hand, Defender (above right) has five buttons and a joystick; novice players are confused by these relatively complex controls and usually give up after a few seconds.

plan, individual offices become visible. Moving the cursor into a room brings the room's details onto the screen. If they choose the wrong room, they merely back out and try another. The lost effort is minimal, and no stigma is attached to the error.

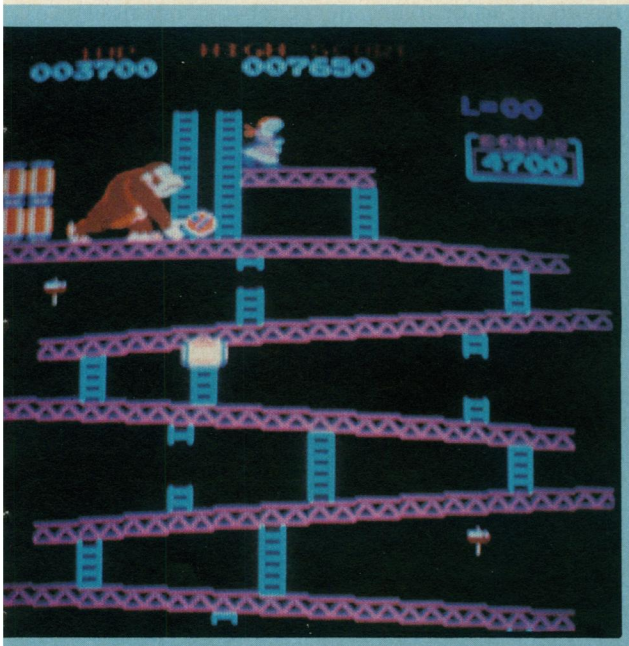
The success of a spatial data management system depends on the designer's skill in choosing icons, graphical representations, and data layouts that are natural and easily understood. Even anxious users enjoy zooming in and out or gliding over data with a joystick, and they quickly demand additional power and data.

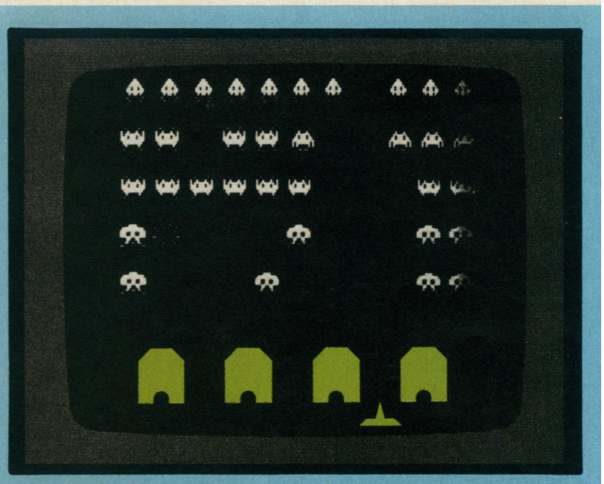
Video games. Perhaps the most exciting, well-engineered—certainly, the most successful—application of direct manipulation is in the world of video games. An early, but simple and popular, game called Pong required the user to rotate a knob, which moved a white rectangle on the screen. A white spot acted as a Ping-Pong ball, which ricocheted off the wall and had to be hit back by the movable white rectangle. The user developed skill involving speed and accuracy in placement of the "paddle" to keep the increasingly speedy ball from getting by, while the speaker emitted a ponging sound when the ball bounced. Watching someone else play for 30 seconds was all the training needed to become a competent novice, but many hours of practice were required to become a skilled expert.

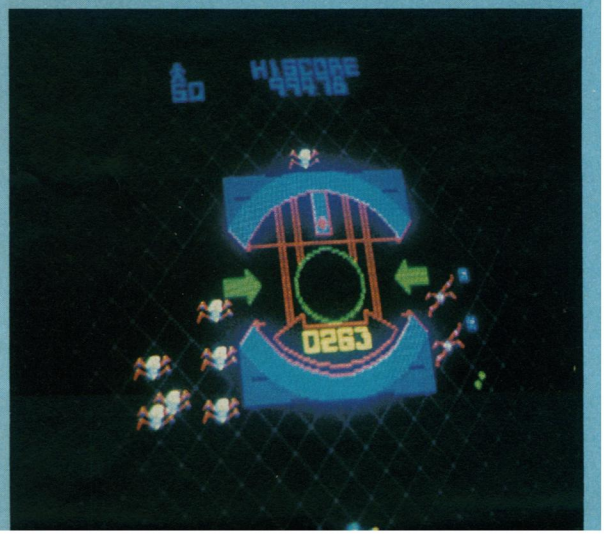
Contemporary games such as Missile Command, Donkey Kong, Pac Man, Tempest, Tron, Centipede, or Space Invaders are far more sophisticated in their rules, color graphics, and sound effects (see sidebar below and on facing page). The designers of these games have provided stimulating entertainment, a challenge for novices and experts, and many intriguing lessons in the human factors of interface design—somehow they have found a way to get people to put coins into the sides of computers. The strong attraction of these games contrasts markedly with the anxiety and resistance many users experience toward office automation equipment.

Because their fields of action are abstractions of reality, these games are easily understood—learning is by analogy. A general idea of the game can be gained by watching the on-line automatic demonstration that runs continuously on the screen, and the basic principles can be learned in a few minutes by watching a knowledgeable player. But there are ample complexities to entice many hours and quarters from experts. The range of skill accommodated is admirable.

The commands are physical actions, such as button presses, joystick motions, or knob rotations, whose results appear immediately on the screen. Since there is no syntax, there are no syntax error messages. If users move their spaceships too far left, then they merely use the natural inverse operation of moving back to the right. Error messages are unnecessary because the results of ac-







nkey Kong, Space Invaders, and Tron (clockwise from above) emplify the lively variety of video games now inviting the er's loose change. As of mid-1981, according to Steve Bloom, thor of *Video Invaders*, more than four billion quarters had en dropped into Space Invaders games around the rld—that's roughly "one game per earthling."

leo game photos reprinted courtesy of *IEEE Spectrum*. For a more complete port on the topic, see "Video Games: The Electronic Big Bang" by Tekla rry, Carol Truxal, and Paul Wallich in *IEEE Spectrum*, Vol. 19, No. 12, Dec. 82, pp. 20-33.

tions are so obvious and easily reversed. These principles can be applied to office automation, personal computing, and other interactive environments.

Every game that I have seen keeps a continuous score so that users can measure their progress and compete with their previous performance, with friends, or with the highest scorers. Typically, the 10 highest scorers get to store their initials in the game for regular display, a form of positive reinforcement that encourages mastery. Malone's[4] and our own studies with elementary school children have shown that continuous display of scores is extremely valuable. Machine-generated value judgments —"Very good" or "You're doing great!"—are not as effective, since the same score means different things to different people. Users prefer to make their own subjective judgments and may perceive machine-generated messages as an annoyance and a deception.

Carroll and Thomas[5] draw productive analogies between game-playing environments and application systems. However, game players seek entertainment and the challenge of mastery, while application-system users focus on the task and may resent forced learning of system constraints. The random events that occur in most games are meant to challenge the user, but predictable system behavior is preferable in nongame designs. Game players compete with the system, but application-system users apparently prefer a strong internal locus of control, which gives them the sense of being in charge.

The pleasure in using these systems stems from the capacity to manipulate the object of interest directly and to generate multiple alternatives rapidly.

Computer-aided design/manufacturing. Many computer-aided design systems for automobiles, electronic circuitry, architecture, aircraft, or newspaper layout use direct manipulation principles. The operator may see a schematic on the screen and with the touch of a lightpen can move resistors or capacitors into or out of the proposed circuit. When the design is complete, the computer can provide information about current, voltage drops, fabrication costs, and warnings about inconsistencies or manufacturing problems. Similarly, newspaper layout artists or automobile body designers can try multiple designs in minutes and record promising approaches until a better one is found.

The pleasure in using these systems stems from the capacity to manipulate the object of interest directly and to generate multiple alternatives rapidly. Some systems have complex command languages, but others have moved to cursor action and graphics-oriented commands.

Another, related application is in computer-aided manufacturing and process control. Honeywell's process control system provides an oil refinery, paper mill, or power utility plant manager with a colored schematic view of the plant. The schematic may be on eight displays, with red lines indicating a sensor value that is out of normal range. By pressing a single numbered button (there are no commands to learn or remember), the operator can get a more detailed view of the troublesome component and, with a second press, move the tree structure down to examine individual sensors or to reset valves and circuits.

The design's basic strategy precludes the necessity of recalling complex commands in once-a-year emergency conditions. The plant schematic facilitates problem solving by analogy, since the link between real-world high temperatures or low pressures and screen representations is so close.

Further examples. Driving an automobile is my favorite example of direct manipulation. The scene is directly visible through the windshield, and actions such as braking or steering have become common skills in our culture. To turn to the left, simply rotate the steering wheel to the left. The response is immediate, and the changing scene provides feedback to refine the turn. Imagine trying to turn by issuing a LEFT 30 DEGREES command and then issuing another command to check your position, but this is the operational level of many office automation tools today.

The term direct manipulation accurately describes the programming of some industrial robots. Here, the operator holds the robot's "hand" and guides it through a spray painting or welding task while the controlling computer records every action. The control computer then repeats the action to operate the robot automatically.

A large part of the success and appeal of the Query-by-Example[6] approach to data manipulation is due to its direct representation of relations on the screen. The user moves a cursor through the columns of the relational table and enters examples of what the result should look like. Just a few single-letter keywords supplement this direct manipulation style. Of course, complex Booleans or mathematical operations require knowledge of syntactic forms. Still, the basic ideas and language facilities can be learned within a half hour by many nonprogrammers. Query-by-Example succeeds because novices can begin work with just a little training, yet there is ample power for the expert. Directly manipulating the cursor across the relation skeleton is a simple task, and how to provide an example that shows the linking variable is intuitively clear to someone who understands tabular data. Zloof[7] recently expanded his ideas into Office-by-Example, which elegantly integrates database search with word processing, electronic mail, business graphics, and menu creation.

Designers of advanced office automation systems have used direct manipulation principles. The Xerox Star[8] offers sophisticated text formatting options, graphics, multiple fonts, and a rapid, high-resolution, cursor-based user interface. Users can drag a document icon and drop it into a printer icon to generate a hardcopy printout. Apple's recently announced Lisa system elegantly applies many of the principles of direct manipulation.

Researchers at IBM's Yorktown Heights facility have proposed a future office system, called Pictureworld, in which graphic icons represent file cabinets, mailboxes, notebooks, phone messages, etc. The user could com-

pose a memo on a display editor and then indicate distribution and filing operations by selecting from the menu of icons. In another project, Yedwab et al.[9] have described a generalized office system, which they call the "automated desk."

Direct manipulation can be applied to replace traditional question-and-answer computer-assisted instruction with more attractive alternatives. Several CDC Plato lessons employ direct manipulation concepts, enabling students to trace inherited characteristics by breeding drosophilla, perform medical procedures to save an emergency room patient, draw and move shapes by finger touches, do chemistry lab projects (see Figure 4), or play games.

Explanations of direct manipulation

Several people have attempted to describe the component principles of direct manipulation. "What you see is what you get," is a phrase used by Don Hatfield of IBM and others to describe the general approach. Hatfield is applying many direct manipulation principles in his work on an advanced office automation system. Expanding Hatfield's premise, Harold Thimbleby of the University of York, England, suggests, "What you see is what you have got." The display should indicate a complete image of what the current status is, what errors have occurred, and what actions are appropriate, according to Thimbleby.

Another imaginative observer of interactive system designs, Ted Nelson,[10] has noticed user excitement over interfaces constructed by what he calls the principle of "virtuality"—a representation of reality that can be manipulated. Rutkowski[11] conveys a similar concept in his principle of transparency: "The user is able to apply intellect directly to the task; the tool itself seems to disappear." MacDonald[12] proposes "visual programming" as a solution to the shortage of application progammers. He feels that visual programming speeds system construction and allows end users to generate or modify applications systems to suit their needs.

Each of these writers has helped increase awareness of the new form that is emerging for interactive systems. Much credit also goes to individual designers who have created systems exemplifying aspects of direct manipulation.

Problem-solving and learning research. Another perspective on direct manipulation comes from psychology literature on problem solving. It shows that suitable representations of problems are crucial to solution finding and to learning.

Polya[13] suggests drawing a picture to represent mathematical problems. This approach is in harmony with Maria Montessori's teaching methods for children.[14] She proposed use of physical objects such as beads or wooden sticks to convey mathematical principles such as addition, multiplication, or size comparison. Bruner[15] extends the physical representation idea to cover polynomial factoring and other mathematical principles. In a recent experiment, Carroll, Thomas, and Malhotra[16] found that subjects given a spatial representation solved problems more rapidly and successfully than subjects given an isomorphic problem with temporal representa-

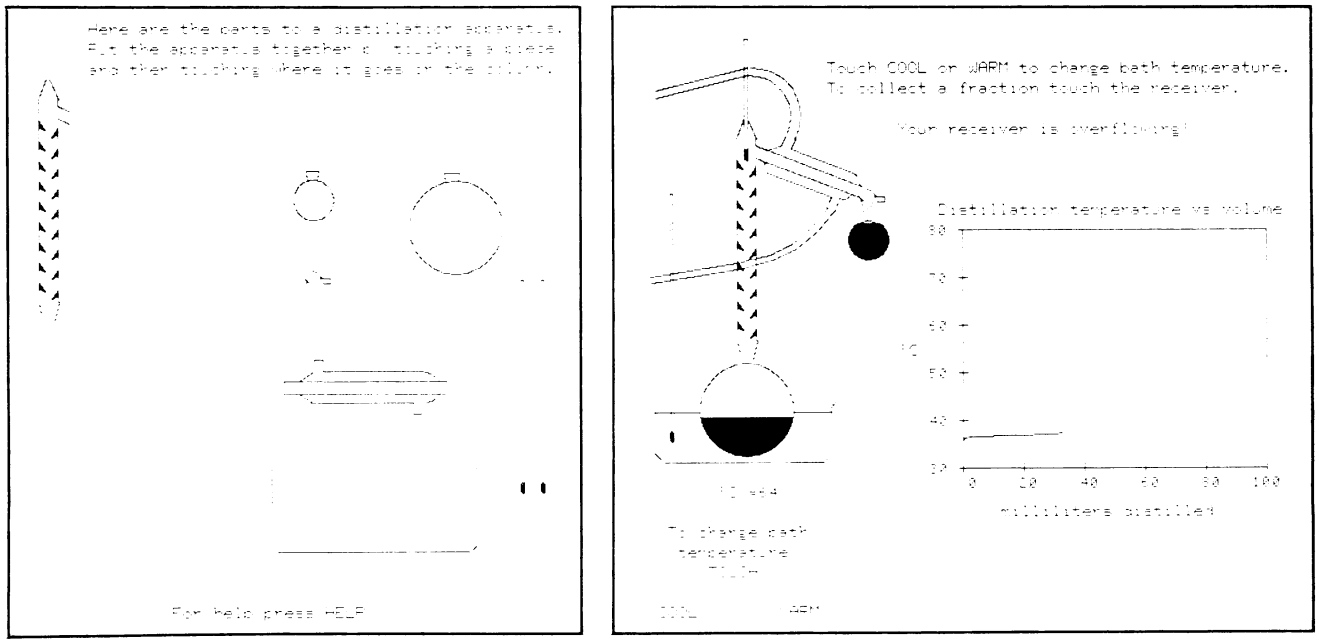


Figure 4. Computer-assisted instruction can become more appealing with direct manipulation, rather than simple question and answer scenarios. This CDC Plato lesson written by Stanley Smith of the Department of Chemistry at the University of Illinois allows students to construct a distillation apparatus by proper finger actions on a touch-sensitive screen (figure at left). Once the student has assembled the apparatus and begun the experiment, the real-time display gives a realistic view of the process with the graph of distillation temperature vs. volume. The student controls the experiment by touching light buttons. The figure at right shows that the student experimenter has gotten into trouble.

tion. (Deeper understanding of visual perception can be obtained from Arnheim[17] and McKim.[18])

Physical, spatial, or visual representations are also easier to retain and manipulate. Wertheimer[19] found that subjects who memorized the formula for the area of a parallelogram, $A = h \times b$, mastered such calculations rapidly. On the other hand, subjects who were given a structural explanation (cut a triangle from one end and place it on the other) retained the knowledge and applied it in similar circumstances more effectively. In plane geometry theorem proving, a spatial representation facilitates discovery of proof procedures more than an axiomatic representation. The diagram provides heuristics that are difficult to extract from the axioms. Similarly, students of algebra are often encouraged to draw a picture to represent a word problem.

Papert's Logo language[20] creates a mathematical microworld in which the principles of geometry are visible. Influenced by the Swiss psychologist Jean Piaget's theory of child development, Logo offers students the opportunity to create line drawings with an electronic turtle displayed on a screen. In this environment, users can receive rapid feedback about their programs, can easily determine what has happened, can quickly spot and repair errors, and can experience creative satisfaction.

Problems with direct manipulation. Some professional programming tasks can be aided by the use of graphic representations such as high-level flowcharts, record structures, or database schema diagrams, but additional effort may be required to absorb the rules of the representation. Graphic representations can be especially helpful when there are multiple relationships among objects and when the representation is more compact than the detailed object. In these cases, selectively screening out detail and presenting a suitable abstraction can facilitate performance.

However, using spatial or graphic representations of the problem does not necessarily improve performance. In a series of studies, subjects given a detailed flowchart did no better in comprehension, debugging, or modification than those given the code only.[21] In a program comprehension task, subjects given a graphic representation of control flow or data structure did no better than those given a textual description.[22] On the other hand, subjects given the data structure documentation consistently did better than subjects given the control flow documentation. This study suggests that the content of graphic representations is a critical determinant of their utility. The wrong information, or a cluttered presentation, can lead to greater confusion.

A second problem is that users must learn the meaning of the components of the graphic representation. A graphic icon, although meaningful to the designer, may require as much—or more—learning time as a word. Some airports serving multilingual communities use graphic icons extensively, but their meaning may not be obvious. Similarly, some computer terminals designed for international use have icons in place of names, but the meaning is not always clear.

A third problem is that the graphic representation may be misleading. The user may rapidly grasp the analogical representation, but then make incorrect conclusions about permissible operations. Designers must be cautious in selecting the displayed representation and the operations. Ample testing must be carried out to refine the representation and minimize negative side effects.

A fourth problem is that graphic representations may take excessive screen display space. For experienced users, a tabular textual display of 50 document names is far more appealing than only 10 document graphic icons with the names abbreviated to fit the icon size. Icons should be evaluated first for their power in displaying static information about objects and their relationship, and second for their utility in the dynamic processes of selection, movement, and deletion.

Choosing the right representations and operations is not easy. Simple metaphors, analogies, or models with a minimal set of concepts seem most appropriate.

Choosing the right representations and operations is not easy. Simple metaphors, analogies, or models with a minimal set of concepts seem most appropriate. Mixing metaphors from two sources adds complexity, which contributes to confusion. The emotional tone of the metaphor should be inviting rather than distasteful or inappropriate[16]—sewage disposal systems are an inappropriate metaphor for electronic message systems. Since users may not share the designer's metaphor, analogy, or conceptual model, ample testing is required.

The syntactic/semantic model. The attraction of systems that use principles of direct manipulation is confirmed by the enthusiasm of their users. The designers of the examples given had an innovative inspiration and an intuitive grasp of what users wanted. Each example has features that could be criticized, but it seems more productive to construct an integrated portrait of direct manipulation:

- Continuous representation of the object of interest.
- Physical actions (movement and selection by mouse, joystick, touch screen, etc.) or labeled button presses instead of complex syntax.
- Rapid, incremental, reversible operations whose impact on the object of interest is immediately visible.
- Layered or spiral approach to learning that permits usage with minimal knowledge. Novices can learn a modest and useful set of commands, which they can exercise till they become an "expert" at level 1 of the system. After obtaining reinforcing feedback from successful operation, users can gracefully expand their knowledge of features and gain fluency.[23]

By using these four principles, it is possible to design systems that have these beneficial attributes:

- Novices can learn basic functionality quickly, usually through a demonstration by a more experienced user.

- Experts can work extremely rapidly to carry out a wide range of tasks, even defining new functions and features.
- Knowledgeable intermittent users can retain operational concepts.
- Error messages are rarely needed.
- Users can immediately see if their actions are furthering their goals, and if not, they can simply change the direction of their activity.
- Users experience less anxiety because the system is comprehensible and because actions are so easily reversible.
- Users gain confidence and mastery because they initiate an action, feel in control, and can predict system responses.

My own understanding of direct manipulation was facilitated by considering the syntactic/semantic model of user behavior. The cognitive model was first developed in the context of programming language experimentation[24,25] and has been applied to database query language questions.[26]

The basic idea is that there are two kinds of knowledge in long-term memory: syntactic and semantic (see Figure 5).

Syntactic knowledge. In a text editor, syntactic knowledge—the details of command syntax—include permissible item delimiters (space, comma, slash, or colon), insertion of a new line after the third line (I3, I 3, or 3I), or the keystroke necessary for erasing a character (delete key, CONTROL-H, or ESCAPE). This knowledge is arbitrary and therefore acquired by rote memorization. Syntactic knowledge is volatile in memory and easily forgotten unless frequently used.[27] This knowledge is system dependent with some possible overlap among systems.

Semantic knowledge. The concepts or functionality—semantic knowledge—are hierarchically structured from low-level functions to higher level concepts. In text editors, lower level functions might be cursor movement, insertion, deletion, changes, text copying, centering, and indentation. These lower level concepts are close to the syntax of the command language. A middle-level semantic concept for text editing might be the process for correcting a misspelling: produce a display of the misspelled word, move the cursor to the appropriate spot, and issue the change command or key in the correct characters. A higher level concept might be the process for moving a sentence from one paragraph to another: move the cursor to the beginning of the sentence, mark this position, move the cursor to the end of the sentence, mark this second position, copy the sentence to a buffer area, clean up the source paragraph, move the cursor to the target location, copy from the buffer, check that the target paragraph is satisfactory, and clear the buffer area.

The higher level concepts in the problem domain (moving a sentence) are decomposed, by the expert user, top-down into multiple, lower level concepts (move cursor, copy from buffer, etc.) closer to the program or syntax domain. Semantic knowledge is largely system independent; text editing functions (inserting/deleting lines, moving sentences, centering, indenting, etc.) are generally available in text editors, although the syntax varies. Semantic knowledge, which is acquired through general explanation, analogy, and example, is easily anchored to familiar concepts and is therefore stable in memory.

The command formulation process in the syntactic/semantic model proceeds from the user's perception of the task in the high-level problem domain to the decomposition into multiple, lower level semantic operations and the conversion into a set of commands. The syntax of text editors may vary, but the decomposition from problem domain into low-level semantics is largely the same. At the syntax level the user must recall whether spaces are permitted, whether program function keys are available, or whether command abbreviations are permitted.

As a user of a half-dozen text editors during a week, I am very aware of the commonality of my thought processes in problem solving and the diversity of syntactic forms with which I must cope. Especially annoying are syntactic clashes such as the different placement of special characters on keyboards, the multiple approaches to backspacing (backspace key, cursor control key, or a mouse), and the fact that one text editor uses "K" for keeping a file while another uses "K" for killing a file.

Implications of the syntactic/semantic model. Novices begin with a close link between syntax and semantics; their attention focuses on the command syntax as they seek to remember the command functions and syntax. In fact, for novice users, the syntax of a precise, concise

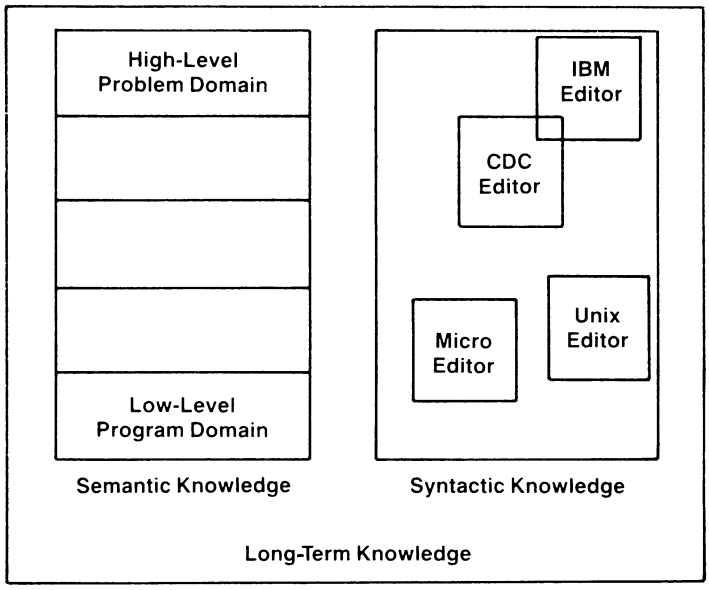


Figure 5. The semantic knowledge in long-term memory goes from high-level problem domain concepts down to numerous low-level program domain details. Semantic knowledge is well-structured, relatively stable, and meaningfully acquired. Syntactic knowledge is arbitrary, relatively volatile unless frequently rehearsed, and acquired by rote memorization. There is usually little overlap between the syntax of different text editors, but they often share semantic concepts about inserting, deleting, and changing lines of text.

command language provides the cues for recalling the semantics. Novices review the command names, in their memory or in a manual, which act as the stimuli for recalling the related semantics. Each command is then evaluated for its applicability to the problem. Novices may have a hard time figuring out how to move a sentence of text, even if they understand each of the commands. Novices using editors that have a "CHANGE /old string/new string/" command must still be taught how to use this command to delete a word or insert a word into a line.

Manuals that have alphabetically arranged sections make it difficult for the novice to anchor material to familiar concepts.

As users gain experience, they increasingly think in higher level semantic terms, which are freer from the syntactic detail and more system independent. In addition to facilitating learning, direct manipulation of a visual representation may aid retention.

The syntactic/semantic model suggests that training manuals should be written from the more familiar, high-level, problem domain viewpoint. The titles of sections should describe problem domain operations that the user deals with regularly. Then the details of the commands used to accomplish the task can be presented, and finally, the actual syntax can be shown. Manuals that have alphabetically arranged sections devoted to each command are very difficult for the novice to learn from, because it is difficult to anchor the material to familiar concepts.

The success of direct manipulation is understandable in the context of the syntactic/semantic model. The object of interest is displayed so that actions are directly in the high-level problem domain. There is little need for decomposition into multiple commands with a complex syntactic form. On the contrary, each command produces a comprehensible action in the problem domain that is immediately visible. The closeness of the problem domain to the command action reduces operator problem-solving load and stress.

Dealing with representations of objects may be more "natural" and closer to innate human capabilities: action and visual skills emerged well before language in human evolution. Psychologists have long known that spatial relationships and actions are more quickly grasped with visual rather than linguistic representations. Furthermore, intuition and discovery are often promoted by suitable visual representations of formal mathematical systems.

Piaget described four stages of growth: sensorimotor (from birth to approximately 2 years), preoperational (2 to 7 years), concrete operational (7 to 11 years), and formal operations (beginning at approximately 11 years).[28] Physical actions on an object are comprehensible during the concrete operational stage, and children acquire the concept of conservation or invariance. At around age 11, children enter the formal operations stage of symbol manipulation to represent actions on objects. Since mathematics and programming require abstract thinking, they are difficult for children, and a greater effort must be made to link the symbolic representation to the actual object. Direct manipulation is an attempt to bring activity to the concrete operational stage or even to the preoperational stage, thus making some tasks easier for children and adults.

It is easy to envision direct manipulation in cases where the physical action is confined to a small number of objects and simple commands, but the approach may be unsuitable for some complex applications. On the other hand, display editors provide impressive functionality in a natural way. The limits of direct manipulation will be determined by the imagination and skill of the designer. With more examples and experience, researchers should be able to test competing theories about the most effective metaphors or analogies. Familiar visual analogies may be more appealing in the early stages of learning the system, while more specific abstract models may be more useful during regular use.

The syntactic/semantic model provides a simple model of human cognitive activity. It must be refined and extended to enhance its explanatory and predictive power. Empirical tests and careful measurements of human performance with a variety of systems are needed to validate the improved model. Cognitive models of user behavior and mental models or system images of computer-supplied functions are rapidly expanding areas of research in computer science and psychology.

Potential applications of direct manipulation

The trick in creating a direct manipulation system is to come up with an appropriate representation or model of reality. I found it difficult to think about information problems in a visual form, but with practice it became more natural. With many applications, the jump to a visual language was initially a struggle, but later I could hardly imagine why anyone would want to use a complex syntactic notation to describe an essentially visual process.

One application that we explored was a personal address list program that displays a Rolodex-like device (see Figure 6). The most recently retrieved address card appears on the screen, and the top line of the next two appear behind, followed by the image of a pack of remaining cards. As the joystick is pushed forward, the Rolodex appears to rotate and successive cards appear in front. As the joystick is pushed further, the cards pass by more quickly; as the joystick is reversed, the direction of movement reverses. To change an entry, users merely move the cursor over the field to be updated and and type the correction. To delete an entry, users merely blank out the fields. Blank cards might be left at the top of the file, but when the fields are filled in, proper alphabetic placement is provided. To find all entries with a specific zip code, users merely type the zip code in the proper field and enter a question mark.

Checkbook maintenance and searching might be done in a similar fashion, by displaying a checkbook register

with labeled columns for check number, date, payee, and amount. The joystick might be used to scan earlier entries. Changes could be made in place, new entries could be made at the first blank line, and a check mark could be made to indicate verification against a monthly report. Searches for a particular payee could be made by filling in a blank payee field and then typing a question mark.

Bibliographic searching has more elaborate requirements, but a basic system could be built by first showing the user a wall of labeled catalog index drawers. A cursor in the shape of a human hand might be moved over to the section labeled "Author Index" and to the drawer labeled "F-L." Depressing the button on the joystick or mouse would cause the drawer to open up and reveal an array of index cards with tabs offering a finer index. Moving the cursor-finger and depressing the selection button would cause the actual index cards to appear. Depressing the button while holding a card would cause copying of the card into the user's notebook, also represented on the screen. Entries in the notebook might be edited to create a printed bibliography or combined with other entries to perform set intersections or unions. Copies of entries could be stored on user files or transmitted to colleagues by electronic mail. It is easy to visualize many alternate approaches, so careful design and experimental testing will be necessary to sort out the successful, comprehensible approaches from the idiosyncratic ones.

It is possible to apply direct manipulation to environments for which there is no obvious physical parallel. Imagine a job control language that shows the file directory continuously, along with representations of computer components. A new file is created by typing its name into the first free spot in the directory listing. A file name is deleted by blanking it out. Copies are made by locking a cursor onto a file name and dragging it to a picture of a tape drive or a printer. For a hierarchical directory, the roots are displayed until a zoom command causes the next level of the tree to appear. With several presses of the button labeled ZOOM a user should be able to find the right item in the directory, but if he goes down the wrong path, the UNZOOM button will return the previous level. (See Figure 7 for a different approach to hierarchical directories.)

Why not make airline reservations by showing the user a map and prompting for cursor motion to the departing and arriving cities? Then use a calendar to select the date, a clock to indicate the time, and the plane's seating plan (with diagonal lines across already reserved seats) to select a seat.

Why not take inventory by showing the aisles of the warehouse with the appropriate number of boxes on each shelf? McDonald[29] has combined videodisc and computer graphics technology in a medical supply inventory with a visual warehouse display.

Why not teach students about polynomial equations by letting them bend the curves and watch how the coefficients change, where the x-axis intersects, and how the derivative equation reacts?[30]

These ideas are sketches for real systems. Competent designers and implementers must complete the sketches and fill in the details. Direct manipulation has the power

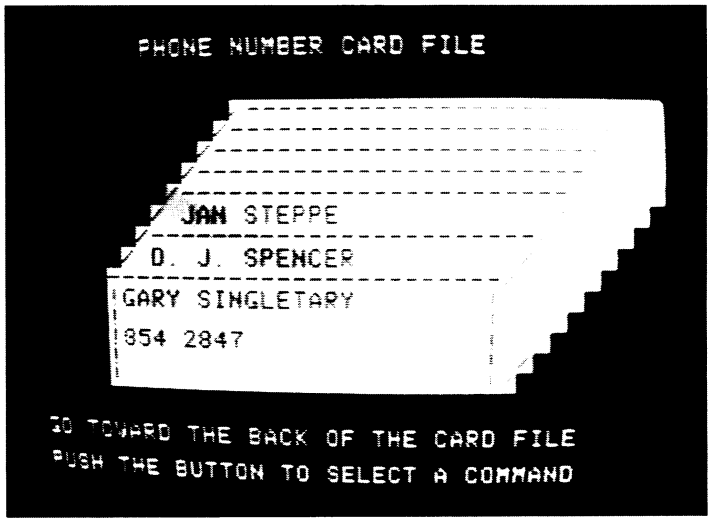


Figure 6. This electronic Rolodex or phone-number card file gives users rapid control over the card motion by a forward or backward joystick press. Different commands can be displayed by moving the joystick left or right. The lively motion of the cards and the natural commands appeal to many users. Implemented by Gary Patterson in Basic on an Apple II, this system was part of a course project at the University of Maryland.

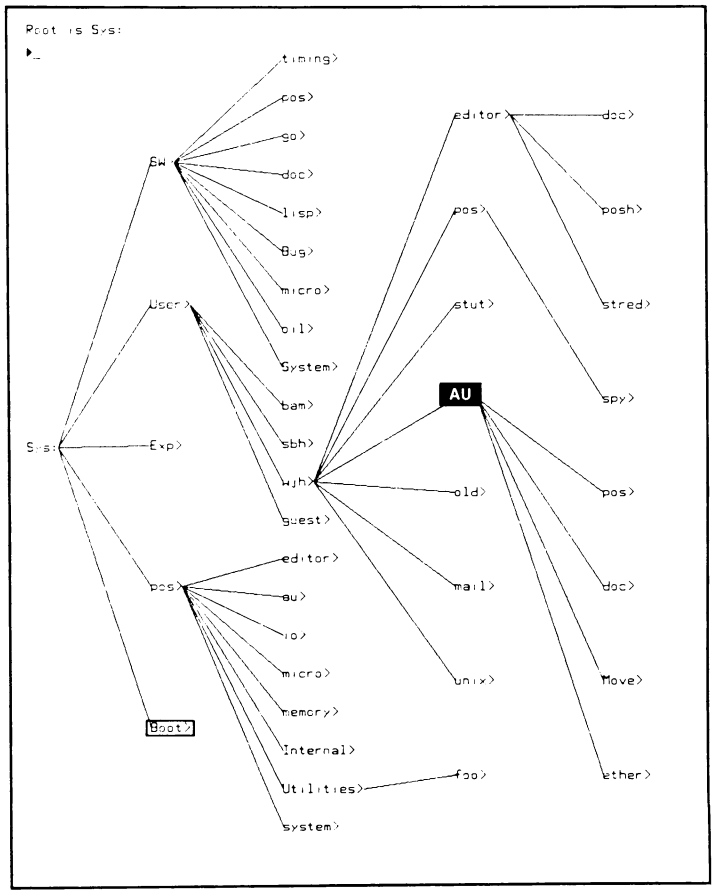


Figure 7. The Dirtree (for directory tree) program on the Perq computer of Three Rivers Computer Corporation is built from left to right by puck selections. The details of lower level directories appear, and the items can then be selected by moving a cursor onto the item. In this figure, the current item is AU, shown in inverse video, but the user has moved the cursor to Boot, which is shown with a box around it. If the button on the puck is pressed, Boot would become the current item. (Figure courtesy of Three Rivers Computer Corporation).

to attract users because it is comprehensible, natural, rapid, and even enjoyable. If actions are simple, reversibility ensured, and retention easy, then anxiety recedes and satisfaction flows in.

The tremendous growth of interest in interactive system design issues in the research community is encouraging. Similarly, the increased concern for improved human engineering in commercial products is a promising sign. Academic and industrial researchers are applying controlled, psychologically oriented experimentation[25] to develop a finer understanding of human performance and to generate a set of practical guidelines. Commercial designers and implementers are eagerly awaiting improved guidelines and increasingly using pilot studies and acceptance tests to refine their designs.

Interactive systems that display a representation of the object of interest and permit rapid, incremental, reversible operations through physical actions rather than command syntax are attracting enthusiastic users. Immediate visibility of the results of operations and a layered or spiral approach to learning contribute to the attraction. Each of these features needs research to refine our understanding of its contributions and limitations. But even while such research is in progress, astute designers can explore this approach.

The future of direct manipulation is promising. Tasks that could have been performed only with tedious command or programming languages may soon be accessible through lively, enjoyable interactive systems that reduce learning time, speed performance, and increase satisfaction. ■

THE NAVAL COASTAL SYSTEMS CENTER

NCSC

Panama City, Florida Offers

SOFTWARE and SONAR ENGINEERING POSITIONS

$20,256 - $53,661 Depending on Qualifications

CHALLENGING ASSIGNMENTS
CAREER GROWTH
LOW COST OF LIVING
BEAUTIFUL ENVIRONMENT

Join a rapidly expanding team of scientists and engineers engaged in Research, Development, Test and Evaluation of major Navy Systems, Supporting Mine Countermeasures, Swimmer Operations, Diving and Salvage, Coastal and Inshore Defense, and Amphibious Warfare.

Qualifications

Bachelors Degree, Engineering, Computer Science, or Physics.

U. S. Citizenship

Significant Experience in:
Development or Life Cycle Support of Embedded Computer Software ---- or ----
Sonar Design and System Development

APPLY IMMEDIATELY

NAVAL COASTAL SYSTEMS CENTER

Dr. D. P. Skinner
Code 03 (83-18)
Panama City, Florida 32407

An Equal Opportunity Employer.

Acknowledgments

I am grateful to the Control Data Corporation for partial support (grant 80M15) of my work and to the University of Maryland Computer Science Center for computer resources to prepare this report. I thank Gordon Braudaway, Jim Foley, John Gannon, Roger Knights, John Lovgren, Harlan Mills, Phyllis Reisner, Sherry Weinberg, and Mark Weiser for their constructive and supportive comments on draft versions. Gio Wiederhold, Stephen Yau, and the reviewers provided useful guidance in shaping the final article.

References

1. Teresa L. Roberts, "Evaluation of Computer Text Editors," PhD dissertation, Stanford University, 1980. Available from University Microfilms, Ann Arbor, Michigan, order number AAD 80-11699.
2. John M. Carroll, "Learning, Using and Designing Command Paradigms," *Human Learning*, Vol. 1, No. 1, 1982, pp. 31-62.
3. Christopher F. Herot, "Spatial Management of Data," *ACM Trans. Database Systems*, Vol. 5, No. 4, Dec. 1980, pp. 493-513.
4. Thomas W. Malone, "What Makes Computer Games Fun?" *Byte*, Vol. 6, No. 12, Dec. 1981, pp. 258-277.
5. John M. Carroll and John C. Thomas, "Metaphor and the Cognitive Representation of Computing Systems, *IEEE Trans. Systems, Man, and Cybernetics*, Vol. SMC-12, No. 2, Mar./Apr. 1982, pp. 107-116.
6. Moshe M. Zloof, "Query-by-Example," *AFIPS Conf. Proc.*, Vol. 44, 1975 NCC, AFIPS Press, Montvale, N.J. 1975.
7. Moshe M. Zloof, "Office-by-Example: A Business Language that Unifies Data and Word Processing and Electronic Mail, *IBM Sys. J.*, Vol. 21, No. 3, 1982, pp. 272-304.
8. Cranfield Smith et al., "Designing the Star User Interface," *Byte*, Vol. 7, No. 4, Apr. 1982, pp. 242-282.
9. Laura Yedwab, Christopher F. Herot, and Ronni L. Rosenberg, "The Automated Desk," *Sigsmall Newsletter*, Vol. 7, No. 2, Oct. 1981, pp. 102-108.
10. Ted Nelson, "Interactive Systems and the Design of Virtuality," *Creative Computing*, Vol. 6, No. 11, Nov. 1980, pp. 56 ff., and Vol. 6, No. 12, Dec. 1980, pp. 94 ff.
11. Chris Rutkowski, "An Introduction to the Human Applications Standard Computer Interface, Part 1: Theory and Principles," *Byte*, Vol. 7, No. 11, Oct. 1982, pp. 291-310.
12. Alan MacDonald, "Visual Programming," *Datamation*, Vol. 28, No. 11, Oct. 1982, pp. 132-140.
13. George Polya, *How to Solve It*, Doubleday, New York, 1957.
14. Maria Montessori, *The Montessori Method*, Schocken, New York, 1964.

15. James Bruner, *Toward a Theory of Instruction,* Harvard University Press, Cambridge, Mass., 1966.

16. John M. Carroll, J. C. Thomas, and A. Malhotra, "Presentation and Representation in Design Problem-Solving," *British J. Psych.,* Vol. 71, 1980, pp. 143-153.

17. Rudolf Arnheim, *Visual Thinking,* University of California Press, Berkeley, Calif., 1972.

18. Robert H. McKim, *Experiences in Visual Thinking,* Brooks/Cole Publishing Co., Monterey, Calif., 1972.

19. Max Wertheimer, *Productive Thinking,* Harper and Row, New York, 1959.

20. Seymour Papert, *Mindstorms: Children, Computers, and Powerful Ideas,* Basic Books, Inc., New York, 1980.

21. Ben Shneiderman, R. Mayer, D. McKay, and P. Heller, "Experimental Investigations of the Utility of Detailed Flowcharts in Programming," *Comm. ACM,* Vol. 20, No. 6, June 1977, pp. 373-381.

22. Ben Shneiderman, "Control Flow and Data Structure Documentation: Two Experiments," *Comm. ACM,* Vol. 25, No. 1, Jan. 1982, pp. 55-63.

23. Michael L. Schneider, "Models for the Design of Static Software User Assistance," *Directions in Human-Computer Interaction,* Albert Badre and Ben Shneiderman, eds., Ablex Publishing Co., Norwood, N.J., 1982.

24. Ben Shneiderman and Richard Mayer, "Syntactic/Semantic Interactions in Programmer Behavior: A Model and Experimental Results," *Int'l J. Computer and Information Sciences,* Vol. 8, No. 3, 1979, pp. 219-239.

25. Ben Shneiderman, *Software Psychology: Human Factors in Computer and Information Systems,* Little, Brown and Co., Boston, Mass., 1980.

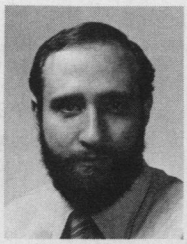
26. Ben Shneiderman, "A Note on Human Factors Issues of Natural Language Interaction with Database Systems," *Information Systems,* Vol. 6, No. 2, Feb. 1981, pp. 125-129.

27. D. P. Ausubel, *Educational Psychology: A Cognitive Approach,* Holt, Rinehart and Winston, New York, 1968.

28. Richard W. Copeland, *How Children Learn Mathematics,* third ed., MacMillan, New York, 1979.

29. Nancy McDonald, "Multi-media Approach to User Interface," *Human Factors in Interactive Computer Systems,* Yannis Vassiliou, ed., Ablex Publishing Co., Norwood, N.J., to appear in 1983.

30. Ben Shneiderman, "A Computer Graphics System for Polynomials," *The Mathematics Teacher,* Vol. 67, No. 2, Feb. 1974, pp. 111-113.



Ben Shneiderman is an associate professor of computer science at the University of Maryland, where he is pursuing research in the design of interactive computer systems. He is the head of the recently formed Laboratory for Human-Computer Interaction within the Center for Automation Research.

Shneiderman is the author of *Software Psychology: Human Factors in Computer and Information Systems,* the coauthor of several textbooks, and the editor of three collections of papers. He has published more than 80 research journal and conference articles.

More Than 31,000 Products Described!

FORBES SAYS ...

"Before even talking to the salesman spend some time with The Software Catalog.*" *(July 4, 1983)*

Indexed 8 different ways, by:

- Application • Subject • Computer System • Operating System
- Programming Language • Microprocessor
- Package Name • Software Vendor

Published every 6 months with supplemental updates three months later.

	Single Copy	Standing Order
MICROCOMPUTERS	$69.00	$58.50
Supplemental Update (Micro)	$15.00	$12.75
MINICOMPUTERS	$95.00	$80.75
Supplemental Update (Mini)	$15.00	$12.75

* **"...it gives concise descriptions of packages that will interest professionals in the accounting, legal, architectural, medical, and dental fields."**

Call 1-800-223-2115 (in New York State) 1-212-867-9040 Ext. 307

ELSEVIER/INTERNATIONAL SOFTWARE DATABASE™

Elsevier Science Publishing, Inc., Box CR, 52 Vanderbilt Ave., New York, New York 10017

Reader Service Number 6