Microsoft Windows

A mouse with modest requirements

by Phil Lemmons

The desktop metaphor and the mouse present attractive concepts, but Apple's Lisa or IBM's PC XT running Visi On exceeds the budget of the average personal computer user. Both of these systems require a hard disk and great quantities of RAM (random-access read/write memory). Although the mouse itself is a small part of the expense, it is a symbol of this approach to software, and some computer users have been heard to mutter, "What price mice?"

Another factor keeping down the mouse population has been the shortage of things for them to point at (or the shortage of applications software). Until there is a large installed base of Lisa and Visi On systems, many software authors will forgo the expense of developing applications programs for these systems. Prospective buyers of personal computers, on the other hand, are unlikely to buy a Lisa or Visi On until more software is available. Apple's own software for Lisa is magnificent, but other applications programs are only now emerging. Visicorp is making a major effort to induce programmers to write more for Visi On, but the requirement of a Unix development system is an obstacle to the smaller software houses and independent designers. The expense underlying the Unix development system is the hardware required to run it—once again, lots of memory and a hard disk.

This keeps most of us staring at the MS-DOS or CP/M command line and hoping that a sudden fall in the prices of RAM and hard disks will open the way to metaphors and mice. With the introduction of Microsoft Windows, however, the company that brought us MS-DOS promises a mouse-and-window show running off two 320K-byte floppy disks and 192K bytes of RAM. (More RAM is required, of course, with each additional application.) To make Microsoft Windows even more attractive to personal computer users, Microsoft promises to price Windows "as an operating-system component"—that is, inexpensively.

The economics of Microsoft Windows will also appeal to programmers. Programmers don't need to buy special hardware or to learn Unix in order to develop software that runs under Microsoft Windows —they can use their own IBM Personal Computers. Moreover, programmers can take advantage of the ability to customize windows so that each software house retains its own distinct look within the Microsoft environ-

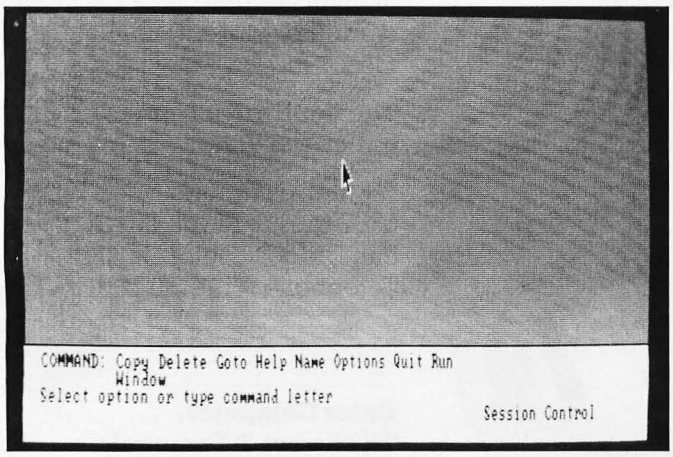


Photo 1

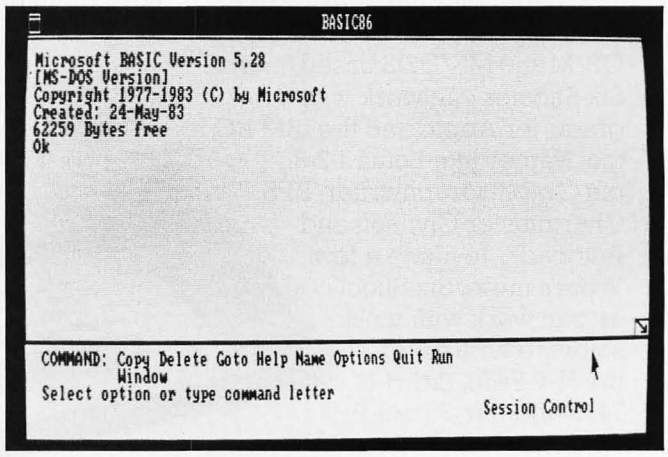


Photo 2

Device-Independent Graphics Output for Microsoft Windows

by John Butler

What makes it possible for Microsoft Windows to output graphics to different devices—printer/plotter devices as well as bit-mapped screens—without changing the graphics code?

Microsoft Windows works with a device-independent graphics system called Graphics Device Interface, or GDI. GDI consists of graphics routines that provide the interface between programs that want to draw images and different output devices. The graphics calls from these programs are not specific to any device. GDI mediates between the graphics calls and the actual devices. The calling program may be an operating-system extension like Microsoft Windows or an application program written in a high-level language.

The design of a device-independent graphics system like GDI begins with the definition of an abstract device. The abstract device is the collection of all the functions that ultimately will be performed by the actual graphics devices. (For example, "draw a circle" or "change hatch style" would be functions for devices to perform.)

When a function is called, GDI takes the function parameters, in abstract-device terms, and passes them to a logical-device driver. A logical-device driver is the software that translates abstract-device functions into a sequence of device-specific actions. These actions (communicated through a physical-device driver) result in the appearance of graphics on the device.

The GDI Abstract Device

The design of the abstract device ultimately determines the types of devices the system can talk to and to what degree the system will be device independent. To define the abstract device for GDI, Microsoft included graphics commands from the current ANSI-VDI (American National Standards Institute-Video Display Interface) standard for drawing on plotting devices. The raster frame-buffer class of device was included by adding the graphics functionality from IBM Personal Computer BASIC. A screen-dump facility and additional raster support provide hard copy and animation capability. GDI's abstract device can support any of the usual graphics subroutine libraries (for example, SIGGRAPH/ACM CORE, ISO GKS, Plot-10) as applications.

The Graphics Primitives

The language of the abstract device is made up of "primitives." The primitives are the calls to the graphics functions available at the lowest level of GDI—the level of the logical-device driver. They are described functionally as follows:

•Control Primitives. These primitives initialize, terminate, and clear the device.

•Output Primitives. These primitives result in the appearance of an actual image on a graphics device. Included are move, mark, polymark, line, polyline, polygon, rectangle, circle, arc, text, and put/get/move bit maps.

•Attribute Primitives. These primitives describe something about the appearance of the output primitives. Each output primitive has a set of appearance commands, including size, color, and style. The filled-output primitives (those defining closed areas, such as polygon and circle) take on additional attributes for the color and style of the interior. Attribute primitives are also provided for using color translation tables and doing high-quality text.

•Viewing Primitives. These primitives control clipping, relative or absolute coordinates, and absolute sizing of images (to inches or meters). They define the border to which output primitives will be clipped. The viewing primitives also map coordinates from the logical device driver to the physical device driver and from one coordinate space to another, and they set up the resolution of the logical coordinate space.

•Inquiry Primitives. These primitives return information to the application program about the current attributes, viewing pipeline, and control flags from the logical-device driver.

GDI provides a language that application programs can use to create images. An application program can create images without knowing about the characteristics of the output device.

ment. The same enlightened attitude enabled Microsoft to resist the temptation to reserve Windows as an environment for its own applications programs. Microsoft is making Windows available to a number of applications software houses, including some major competitors.

Microsoft Windows is an installable device driver under MS-DOS 2.0 using ordinary MS-DOS files. Complete compatibility with MS-DOS means that Windows will at least let you run any application that runs under MS-DOS. In the worst case, Windows will turn the full display over to an MS-DOS application and return you to your place in Windows. "Language bindings" will enable programmers to write software for Microsoft Windows in any Microsoft programming language.

Running Microsoft Windows

Photos 1–13 show a sequence of operations in Microsoft Windows. The photos on pages 52–53 show a variety of machines whose manufacturers have adopted Microsoft Windows as an applications environment.

During normal use, Microsoft Windows displays one or more windows, each with a different application. You can move the cursor from one window to another. You can move windows, change their size, scroll, get help appropriate to the context in which you are working, and transfer data among windows. Windows determines the highest level of data transfer mutually acceptable to the two applications, with plain ASCII (American National Standard Code for Information Interchange) as the last resort.

The "session-control layer" becomes the equivalent of the empty desktop where you can manipulate files. The available commands appear near the bottom of the screen. Normally, Microsoft Windows will restore the desktop to the state at the time of its last use. In photo 1, we start from scratch.

To see the available applications programs, you either use the mouse to position the cursor on the command "Run" or type the letter "R." Windows lists all the applications programs as commands, and you

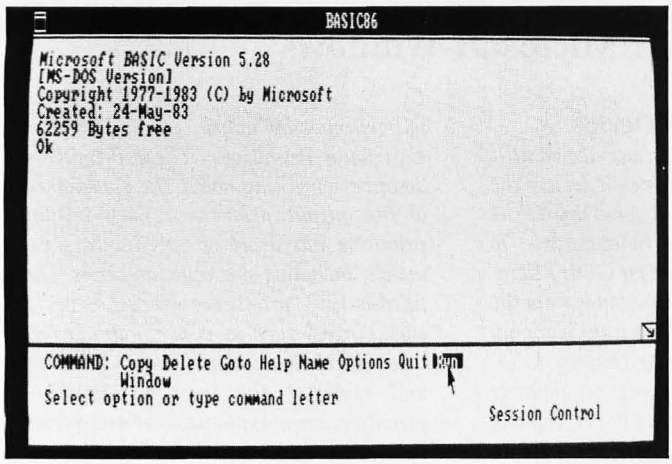


Photo 3

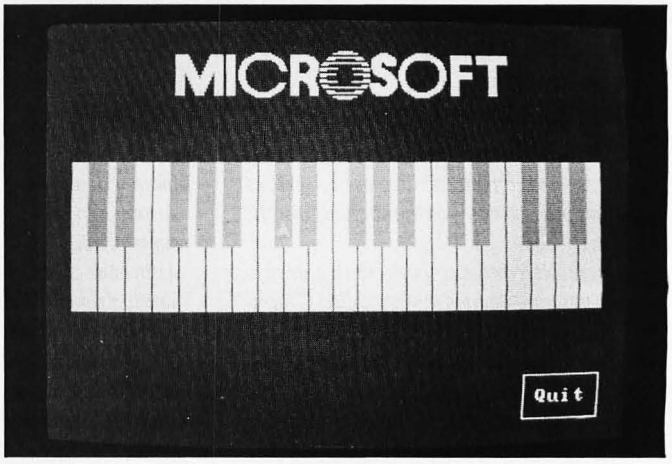


Photo 4

point at the desired program and click the mouse to run it. You could also type the appropriate letter instead.

In photo 2, BASIC 86 is running in a large window extending the full width of the desktop. Because BASIC 86 does all its input/output through MS-DOS, it can run in a Window. Microsoft calls such software "cooperative." The bottom of the screen shows the commands available in the session-control layer. You can use the session-control layer to run another program in parallel with BASIC 86.

The first step toward running a program is shown in photo 3, where the cursor points at "Run." Microsoft Windows will now display a list of the programs available.

Photo 4 shows the next application selected. In this case, the program that's run is "uncooperative"—that is, it doesn't do everything through MS-DOS system calls, sometimes going beyond the operating system to write directly to hardware addresses such as those of screen memory. Microsoft Windows can't run such a program in a window and must give it the entire screen. That is why photo 4 does not show the session-control layer beneath the display of "Piano."

Certain programs can't use the multiple-window feature.

Photo 5 shows the transition from the uncooperative program to a "smart" one that can live happily in a smaller window and share the screen with other programs that take full advantage of Microsoft Windows. The smart program is Microsoft Word. Photo 6 shows two applications—Word in the upper window and Multiplan in the lower; both these programs were written to take advantage of Microsoft Windows. Because the cursor is pointing at one of the cells in the Multiplan spreadsheet, the command bar at the bottom of the screen shows Multiplan's commands. You can move either window by grabbing its title bar with the mouse. You could "grow" either window by grabbing the "grow box." Although these photos show the title bar at the top of the window and the grow box at the lower right, software developers can put them elsewhere if desired.

(In fact, Microsoft's own standard window has changed since these photos were taken. The latest version provides a question mark on the right

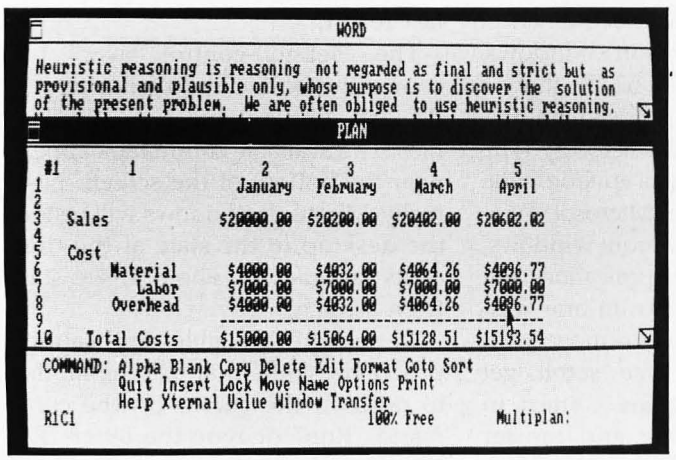


Photo 7

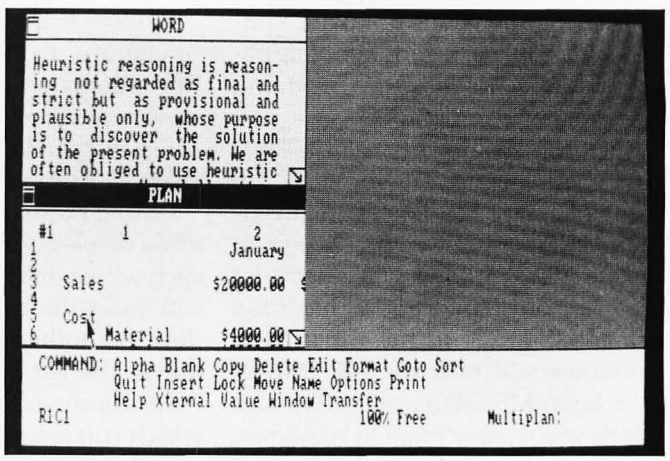


Photo 8

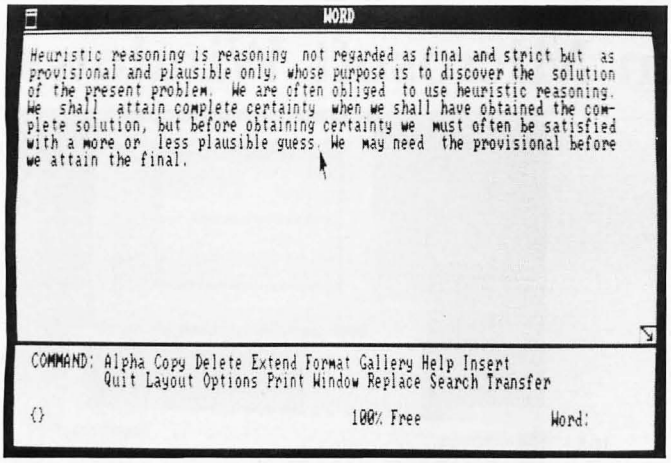


Photo 5

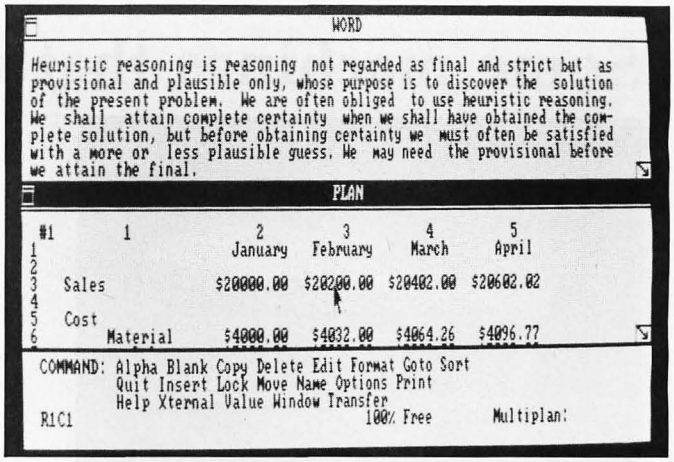


Photo 6

part of the title bar. Selecting the question mark brings help information. If you put the cursor on the title itself, it is replaced by little pictures that represent what you can do with the window. The new version also includes a status line at the top of the screen and an area for icons at the bottom.)

In photo 7, Multiplan's window has been enlarged to show more cells and more data, and Microsoft Word's window has been reduced as necessary.

Photo 8 shows both the Multiplan window and the Microsoft Word window reduced. (Since photo 8 was taken, Microsoft Windows has been adapted to use an automatic resizing process called "tiling." Rather than letting windows overlap or leaving part of the desktop empty, Microsoft Windows always gives all the space on the screen to the applications that are running.)

Photo 9 shows a charting program occupying a large window at the right-hand side of the screen. With the cursor in that large window, the command bar at the bottom of the screen lists charting commands. Note that when the window containing the charting program is expanded by moving the title bar and grabbing the grow box, the line graph has been automatically rescaled (see photo 10). Microsoft Windows can rescale graphics if desired.

Microsoft Windows can rescale graphics if desired.

Photo 11 shows a sample "pop up" menu for the charting program. Pointing at the PEN command on the command bar at the bottom of the screen has brought the display of the menu of pen sizes and patterns. You select sizes and patterns by using the mouse to point at one of the boxes shown in each list, then pointing at the "OK" box (see photo 12). As with other aspects of the Microsoft Windows displays, programmers can redesign menus to their own taste.

Photo 13 shows the graph displayed in accordance with the instructions entered—with a 4 by 4 pixel-pen size and a gray shading. The graphics capabilities of Microsoft Windows owe much to the device-independent graphics system described by John Butler in the text box "Device-Independent Graphics Output for Microsoft Windows" on page 49.

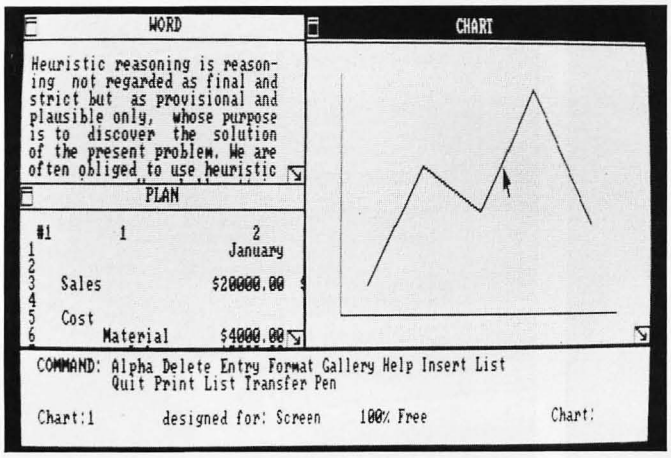


Photo 9

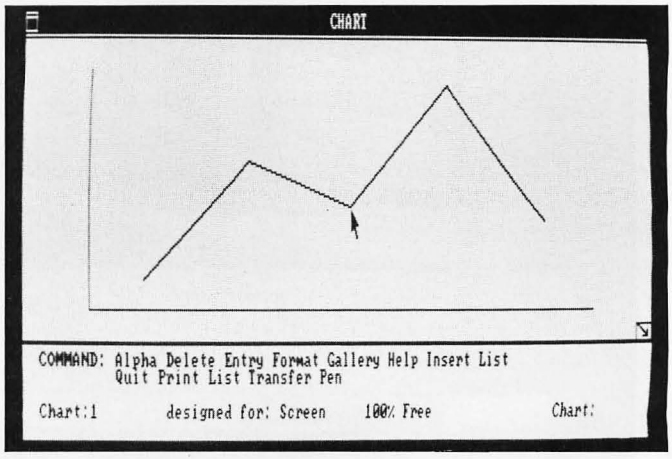
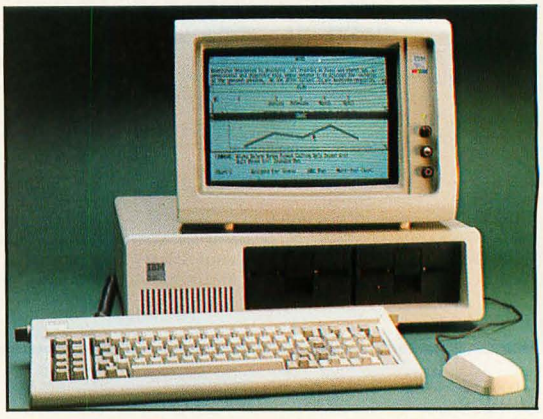


Photo 10

Some machines that run Microsoft Windows

Hewlett-Packard 150



IBM
Personal
Computer

DEC
Rainbow 100

Bytec
Hyperion

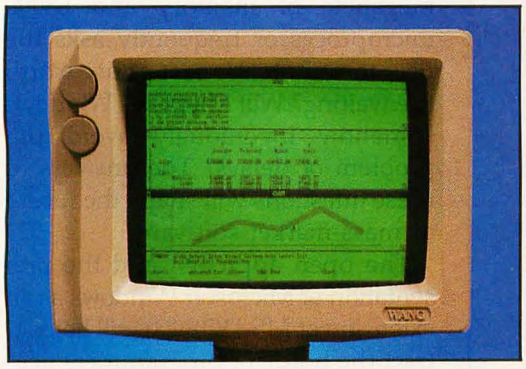
Apple IIe/
Rana Drive
System
with 8086

Eagle PC

Burroughs B20

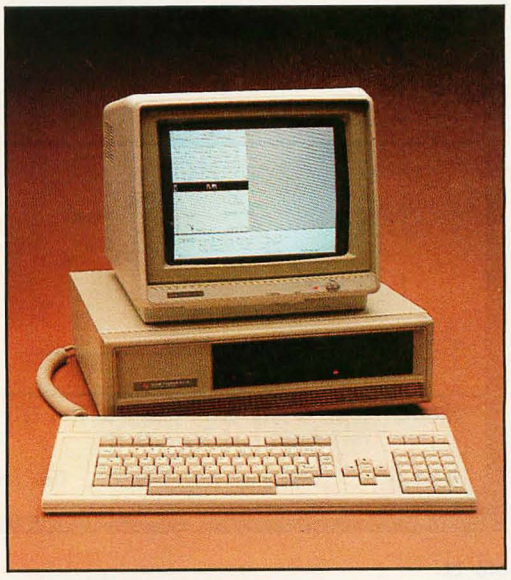
Compaq Portable

Zenith Z-100



Wang Professional Computer

Texas Instruments Professional Computer



Honeywell Microsystem 6/10

A package from Radio Shack, not to be opened before December 1

Columbia Data Products MPC Portable

Computer Devices Dot

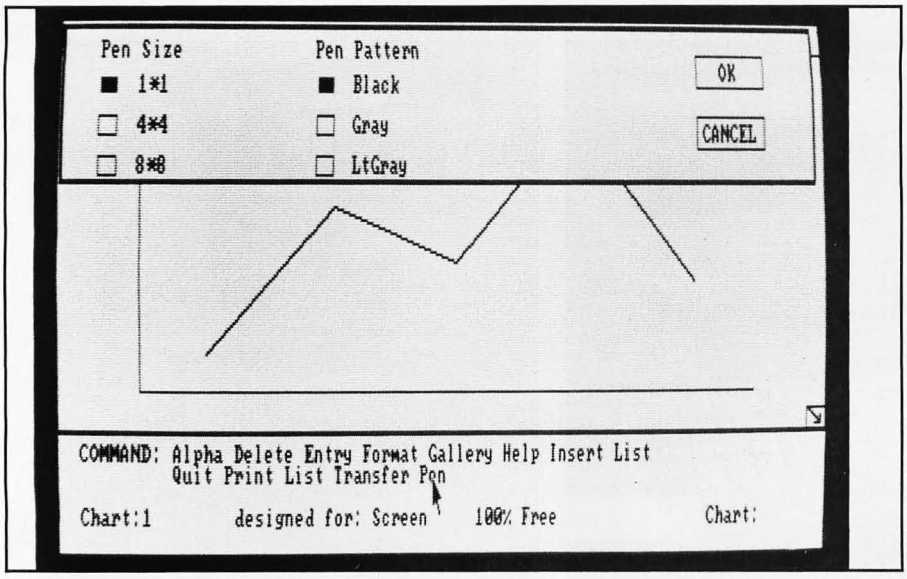


Photo 11

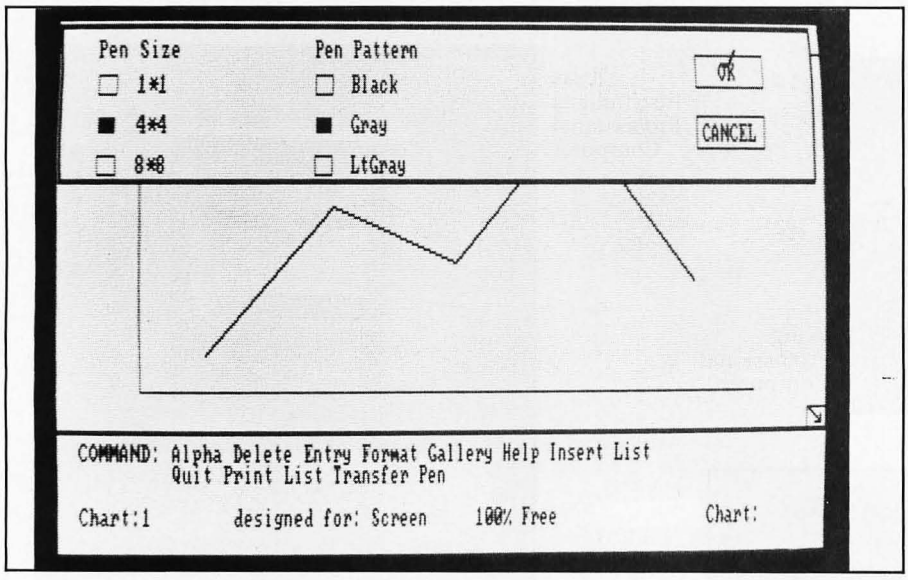


Photo 12

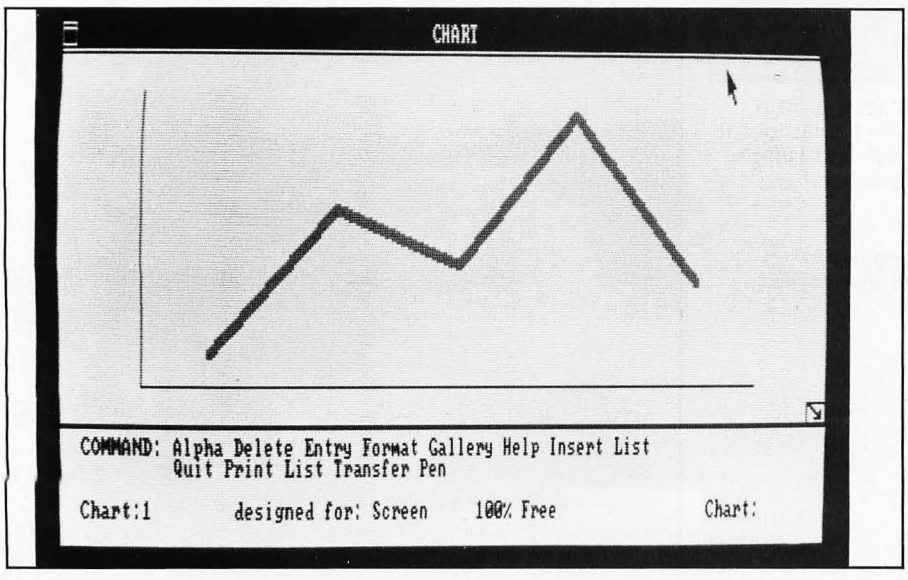


Photo 13

Conclusions

Microsoft Windows seems to offer remarkable openness, reconfigurability, and transportability as well as modest hardware requirements and pricing. As a result, the desktop metaphor and mouse, intended to bring computing power to nontechnical people, are finally going to reach the hands of many such people. Barring a surprise product introduction from another company, Microsoft Windows will be the first large-scale test of the desktop metaphor in the hands of its intended users.

It is natural to wonder whether Microsoft Windows' ability to run in limited memory and off floppy disks will result in noticeable delays during execution. Even Lisa with its megabyte of memory and 68000 microprocessor frequently asks the user to wait. Is the ease of use worth the waiting? Will Microsoft Windows somehow ingeniously avoid the problem of delays? The answers to these questions will shape the future of mass-market software.

The open approach and the presentation of Microsoft Windows as an extension of MS-DOS 2.0 will help attract the horde of programmers necessary to assure acceptable execution speeds on the IBM PC. Just as enough programmers working long enough on enough different approaches have made the Apple II perform feats that once seemed incredible, enough programmers working long enough on different approaches will make applications run fast under Microsoft Windows on ordinary hardware. Even if this judgment proves mistaken, Microsoft's policy of openness and low pricing will have made possible a major experiment in mass-market software. For many software authors as well as users, this will be the first chance to test an approach to the user interface that has hovered just beyond reach for several years. ■

Phil Lemmons, BYTE's West Coast Bureau Chief, can be reached at McGraw-Hill, 425 Battery St., San Francisco, CA 94111.