

Challenges in Graphical User Interfaces

Robert F. Sproull

Sutherland, Sproull, and Associates, Inc. and

Carnegie-Mellon University

Abstract

The principal advances in graphics over the last ten years have been economic—inexpensive raster displays have made good-quality interactive graphics affordable in computer terminals, and together with high-performance microprocessors have led to the "workstation" with an integral display. There have also been gains in graphics hardware, such as improved color displays, raster printers, and inexpensive input devices such as the mouse.

The advance of graphical user interfaces has occurred on a more narrow front. The principal thread is work at the Xerox Palo Alto Research Center that has emerged in the Star and Smalltalk-80 products, and has been ably employed in Apple's LISA. Much of this work is restricted to "office applications."

There remain numerous challenging problems in extending graphical user interfaces. How can pictures and text be routinely intermixed in documents? How can pictures be used as input to the computer as well as output? How can graphical interfaces to programming systems be extended to deal with more complex environments? How can lessons from communication and graphic design be incorporated into user-interface design?

Challenges

This section describes several challenges in the design of graphical user interfaces. Although the first two items on the list concern graphics hardware, the list reflects the view that many of the current limitations in graphical user interfaces stem not from hardware constraints but from interactive software and tools for building software that are commonly available. Many of these limitations have been surmounted by one or more research projects, but remain obstacles in current practice.

Exploiting raster imaging. Graphical user interfaces depend on the frame-buffer raster display to present arbitrary images on the screen. Although such displays first became practical ten years ago, their widespread use depends on the declining cost of semiconductor memory. Now, specialized memory parts (Texas Instruments' dual-ported video memory) and VLSI processors that cater for graphics applications promise even greater display performance than is now achieved with conventional microprocessor systems. Accompanying the rise of raster displays are raster printers using pin-matrix, ink-jet, or laser-scanned electrography. All of these raster devices share the important property that the imaging technique is insensitive to image content: any image can be approximated by a suitable pattern of raster dots. Thus raster technologies provide the designer of interactive programs with tremendous flexibility in the kind of imagery he can use.

At present, some display controllers and most printer controllers limit the flexibility intrinsic in raster imaging. These limitations arise principally from economic considerations, such as the cost of a full frame buffer for a laser-scanned page image (at 300 pixels/inch, a 90-inch² page requires about 1 MByte of memory), or the opportunity to design limited products that meet certain limited markets. With time, these limitations can be expected to disappear.

High-resolution color displays. The hardware development that is likely to have the greatest impact on user interface design is the advent of a color display on which a user can comfortably view one or two pages of text in a document such as this one. While high-resolution color displays exist now, the spatial structures necessary to achieve color (e.g., shadow mask) interfere with the display of characters. Although pages such as this one are legible on these displays, editing or long periods of work are not comfortable. Along with new display and monitor technology must come increased memory bandwidths to refresh the displays, and increased processor performance to update the frame-buffer memory.

Mixing pictures and text in documents routinely. Although some interactive systems allow graphics and text to be combined in a single document, these systems are not widely applied. Some of these systems are restricted to publishing applications, where graphics are required. Others are intended for the office worker, such as Star [9] and LISA, and are closed off from other applications. Some of these systems treat graphics as a second-class form and may require the user to work extra hard to include a diagram or figure in a document.

A prerequisite to a wider exploration of graphical interfaces is the routine use of text and graphics in computer systems. Computer systems today use the "text file" as the most important information structure the user sees. If that structure were extended to become the "text and graphics file," a great

number of possibilities would open up, some of which are explored further below.

Raising the status of graphics to become a natural mode in working with computer systems will have many benefits. Consider a computer program that looks like Figure 1. The drawing is clearly a "comment" which the compiler will ignore, since it is enclosed by the comment conventions of the programming language. Drawings such as this are part of a program's documentation, and ought to be part of the programs and specifications themselves, if only as comments.

Graphical literals could appear in programs, as shown in Figure 2. The principal use for such literals would presumably be in "print" statements or other forms that generate output on a display or printer. Graphical literals would be a natural part of any serious attempt to provide graphical data types in a programming language.

The problem of integrating text and graphics routinely is not so much one of technical capability as it is of social pressure and diffusion. Much of the spread of computing technology occurs through the languages and systems that are spread: Pascal and Unix are prime examples. If we are to use graphics routinely in computer systems, we must find ways to spread systems that offer useful graphical facilities, that provide a sound basis on which to build further applications, and that set good examples for others to follow and extend.

```

procedure AddEntry (l: list, e: entry);
  (*
    
  *)
  var n,f: list;
  begin ...
  
```

Figure 1. A drawing used as a comment.

```

print(
  
)
  
```

Figure 2. A drawing used as a literal.

Interpreting drawings. We use drawings routinely in human communication, but in human-computer communications, we tend to use drawings only for computer output, not for input. Yet there are many applications where a drawing, prepared using some suitable "drawing conventions," could be interpreted or parsed to serve as a principal means of input to a computer program.

A very few examples of this technique have appeared. Anderson devised a technique for parsing two-dimensional mathematical expressions to recover their computational structure, which Lewis [7] embedded in a program for experimenting with functions in complex analysis. Thacker [11] uses

Challenges in Graphical User Interfaces

drawings such as the one in Figure 3 as the sole input to an electrical circuit computer-aided design system. The system interprets the drawings to determine the identities and interconnections of circuit parts.

There are lower-level techniques that may be applied to derive clean drawings from sloppier hand-drawn input. On-line character recognition is one such technique. Herot [5] is able to turn certain kinds of rough sketches into finished drawings.

Drawings used as the primary input to application programs have several advantages. The principal gain is that only a single interactive drawing program need be written, which can be used to prepare input for a wide variety of applications. The drawing program is insensitive to the drawing's interpretation, just as a text editor is insensitive to a text's interpretation. The user is probably well served by this arrangement, since he will need to learn to use only one interactive system rather than one for each application he uses. (It is presumably a simpler matter to learn the drawing conventions associated with different interpretations of drawings, since these conventions can often match a visual symbology that has already been developed, such as ways to draw circuit diagrams.) And finally, the drawings prepared in this way can be merged into text documents for documentation, can be printed on various output devices, and so on without separate provisions in each application program.

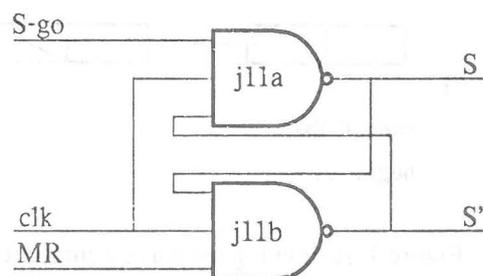


Figure 3. A drawing that can be interpreted to yield a circuit model.

Extending graphical interfaces to more complex applications. The graphical user interfaces of Star and LISA address fairly limited applications. Consider extending interfaces of this sort to address more complex situations. How, for example, would the objects and operations of a complete computer system such as Unix be presented through such an interface? The user-interface designer faces questions such as:

How is a graphical interface to be extended to deal with a large number of data types? The LISA interface, in which the user identifies objects by pointing at them, associates a unique application program with each file type (e.g., a document is identified with the editor, a drawing with the illustrator); this approach is probably too restrictive in general.

Many times, application programs are invoked with other files as arguments. How are these objects given to the application? By copying them and moving the copies into some argument

Challenges in Graphical User Interfaces

containers associated with each application?

Most user interfaces to computer systems have some sort of written command language that allows common command sequences to be written in "command files." In many cases, these command files allow new commands to be defined. What is the analog of the command file in a graphical user interface? How are "arguments" to a command file or a macro processor to be treated? How can one write "graphical programs" without substantial complexity? Is there any virtue in graphical command files? If not, how can conventional command files be made to work consistently with a graphical interface?

While it is not hard to imagine a user interface that deals with these questions, would such an interface be valuable? It might be that the user's model of such an interface would be much more complex than for a conventional text-oriented interface. Can such interfaces be designed to be practical, or at best "cute"?

Constraints. Although many interactive drawing programs expect the user to build drawings literally, by commanding the system to construct each of the elements of the drawing, correct as to size and position, an alternative approach allows the user to specify the rules that will generate the proper drawing. Rules take the form of constraints, such as that two lines be parallel, that two lines intersect at a given angle, that a line have a certain length, and so on. Sketchpad [10] introduced constraints for this purpose, and solved the resulting constraint network using relaxation techniques.

Sporadic progress in constraint-based systems has been made since then, although a growing number of applications seem to have a need for constraint-based solutions (e.g., VLSI artwork layout). However, it now appears that many linear constraint networks can be solved very fast, perhaps even at interactive speeds [4], and non-linear constraints of the sort that crop up frequently in graphical applications are tractable [8].

Examples abound of systems that ought to be based on constraints. The popular "spread sheet" programs such as VisiCalc are in fact simple constraint systems, in which numeric entries in a table are related to other entries by means of simple algebraic constraints. (Alas, some of these systems require the user to order entries in the table so that the constraints can be solved in a top-to-bottom pass over the table. None of the systems allows cycles in the constraints.) Business-graphics programs that build pie charts or bar graphs based on numeric input could profitably be expressed as constraint systems, in which certain geometric properties of the chart depend on numeric data, but other properties are unconstrained, so that the user may alter them.

Now that techniques for solving constraint networks are better understood, the problem of designing a user interface to constraints remains the biggest challenge. The general notion of constraint is not an easy one for non-technical people to understand. Moreover, interacting with constraint-based systems sometimes leads to surprising results, for example when a drawing is improperly constrained it may change into something very far from the user's intentions. It seems clear that the user interface will need to allow mixing literal graphics and constraints.

Challenges in Graphical User Interfaces

Making better pictorial presentations. Many of the images that are presented to users could be made much more effective. While earlier technologies of glass teletypes and low-resolution displays prevented more pleasing images, current displays are less restrictive. We must now approach graphical presentation as a serious design problem.

This task can be greatly aided by graphic designers, who have considerable experience with presentation, typography, document design, and so on. Much of their experience comes from dealing with the general public in non-technical settings, a viewpoint that is likely to help in the design of user interfaces.

There are opportunities for building tools that make good graphical presentations easier to achieve. For example, the notion of separating the *content* of a document, which the author must control, from its *form*, which a document designer might better control, is now becoming popular in the publishing industry, where it goes by the name of *generic coding*. A similar approach can be applied to illustration, where details of graphical execution are defined by a *graphical style*, although the illustration as a whole is defined by the user [1].

Computer programs could offer advice or criticism of an illustration prepared by a user. Since many of the people who create illustrations with interactive illustration programs are not skilled illustrators, they can often use help in making effective illustrations. The drawing program could, for example, provide counsel on the selection of colors and the effect the selection will have on the way the illustration will be understood. Here is an area for experimentation with "expert systems."

Spreading the art. The design of user interfaces would doubtless improve if designers could study other cases of problems and solutions in user interface design. Unfortunately, designers usually do not have sufficient access to other systems to carry out these studies. While movies or video tapes of interactive systems are a help, because they are demonstrations rather than analyses, they are not sufficient for a designer to learn much about an interactive program. The designer will have questions such as:

Exactly what is the design for the user interface? It would be nice if we had succinct ways to describe a user interface. Such descriptions could be used as the basis for evaluating a performance model such as the keystroke model [2].

What are some of the *strategies* that the designer intends users to know about? For example, an editor with "cut" and "paste" commands may intend text to be moved by first cutting it and then pasting it.

What are some scenarios of users operating the system? Here movies or video tapes may be useful.

What do users rave or complain about? What do they find easy and hard? What *post facto* analysis or reflection have the designers done?

Perhaps it is too soon to expect a library of carefully documented user interface designs, in part because there is still a great deal of invention in user interfaces rather than methodical, incremental

improvements. However, it is probably not too soon to begin work on the notations and analysis techniques necessary to build such a library.

Infrastructure

Several of the challenges listed above deal with creating an *infrastructure* in which graphics are accommodated, used routinely, and even encouraged. At present, computing infrastructures in widespread use cater only for linear text. While many kinds of isolated graphics applications packages can be purchased, there are no programming environments available that can be said to provide a comfortable base for programming applications that make heavy use of graphics. Even Smalltalk [3] falls short in this area (chiefly because its "documents" do not contain graphics routinely), and graphics subroutine packages such as GKS [6] address only a tiny part of the problem, the need to generate output on a display.

The user interface is only the surface

Although graphical user interfaces have a great deal of appeal and may lead to ways to make interactive systems easy to learn and to use, the user interface is probably not the most important ingredient of a successful interactive program. Rather, the underlying application and the concepts it provides are the principal determinants of how well the program meets the user's needs. A poor user interface can spoil a fine application, but a wonderful user interface is unlikely to broaden the scope of a narrowly-defined application.

Perhaps the best example of this phenomenon comes from computer-aided design, and is deeply rooted in the history of computer graphics. Sutherland's Sketchpad program was hailed as a major breakthrough in CAD; predictions were made that an interactive drawing program of this sort would revolutionize engineering. Indeed, interactive drafting systems were among the first graphical CAD applications. But interactive graphics has been more slowly applied to *design*, because the engineer is less concerned with a drawing than with a *model* of a mechanical structure that can be subjected to various analyses, of which picture formation is only one. The engineer is concerned with mass, surface area, strength, vibration modes, manufacturability, and so on, all of which require calculations that Sketchpad's internal model could not support. Thus, to engineers, the technology of solid modeling is more important than that of graphical user interfaces.

On the other hand, the hope that created the euphoria around Sketchpad twenty years ago is still strong today: graphical expression can and should be integrated into our habitual ways of interacting with computers.

References

- [1] R.J. Beach and M. Stone, "Graphical Style—Towards High Quality Illustrations," *Computer Graphics*, 17(3):127–135, July 1983.
- [2] S. Card, T. Moran, and A. Newell, "The Keystroke-Level Model for User Performance Time with Interactive Systems," *CACM*, 23(7):396–410, July 1980.
- [3] A. Goldberg and D. Robson, *Smalltalk-80, The Language and its Implementation*, Addison-Wesley, Reading, Mass. 1983.

Challenges in Graphical User Interfaces

- [4] J.A. Gosling, "Algebraic Constraints," Technical Report, Computer Science Department, Carnegie-Mellon University, 1983.
- [5] C.F. Herot, "Graphical input through machine recognition of sketches," *Computer Graphics*, 10(2):97–102, Summer 1976.
- [6] International Standards Organization, *Graphical Kernel System (GKS)—Functional Description*, ISO DP 7942, November 1982.
- [7] H.R. Lewis, "ShapeShifter: An Interactive Program for Experimenting with Complex-Plane Transformations," *Proc. ACM National Conference*, 1968, p. 717.
- [8] R. Light and D. Gossard, "Modification of geometric models through variational geometry," *Computer Aided Design*, January 1982, p. 209.
- [9] D.C. Smith, E. Harslem, C. Irby, and R. Kimball, "The Star User Interface: An Overview," *Proc. National Computer Conference*, June 1982, pp. 7–10.
- [10] I.E. Sutherland, "Sketchpad: A Man-Machine Graphical Communication System," MIT Lincoln Laboratories Technical Report 296, May 1965. Abridged version in *Spring Joint Computer Conference* 1963, Spartan Books, Baltimore, Md.
- [11] C.P. Thacker, "SIL—A Simple Illustrator for CAD," in S.S.L. Chang, ed., *Fundamentals Handbook of Electrical and Computer Engineering*, Vol. 3, Wiley, New York, 1983, p. 477.

DISCUSSION

Professor G.A. Rose asked about the interaction between image processing and interactive graphics and modelling. In reply, Dr. Sproull discussed the different approaches of the two techniques. He expressed his belief that a better understanding of image processing would result in the avoidance of making the kind of errors that had been made in the design of flight simulators.

Professor G.F. Coulouris wanted to know what kind of software environment was required, with particular reference to the PERQ. Dr. Sproull briefly described the CEDAR system produced at Xerox - a graphics system for the fast updating of images and text, with some window management. He thought that the design of such a system would not be a major undertaking. What was essential to the software environment was, he thought, a large personal computer with virtual memory.

Dr. E.S. Page drew the audience's attention to a proposal at Reading to combine graphic design and typography. An attempt had been made, unsuccessfully, to raise money for a joint Computing Science and Cybernetics one year project. Dr. Page intimated that the proposal would be re-submitted; this received Dr. Sproull's enthusiastic support.

