Producing data in pictorial form is a type of computer graphics application known as presentation graphics. One approach that has been used for this type of graphics is a graphic support package using APL as the command language. Here discussed is the evolution of this approach up to its currently available forms.

An APL approach to presentation graphics

by W. H. Niehoff and A. L. Jones

The use of visual display technology to present the results of programmed digital computations is as old as the stored program digital computer.[1] But it is the reduced costs of this technology and its widespread application in the interactive computer environment that have rejuvenated the idea during the past decade and made it appear new. This paper describes one adaptation of *presentation graphics* to a particular interactive computing environment, that of APL.[2] Presentation graphics is a term used to describe the subset of computer graphics applications principally characterized by the use of graphics technology to present data in pictorial form, enhancing discovery and comprehension of relationships and characteristics. Generally, the emphasis is on graphic output, as opposed to those applications like computer-aided design that depend on dynamic human interaction using graphic input techniques. This does not imply that presentation graphics is not interactive; interactivity may be achieved through alternative techniques, including the use of command languages. It is in this context that APL is particularly appropriate.

The use of APL in what can be broadly interpreted as presentation graphics began in the late 1960s. A graphics support package that resulted from one of these early efforts, APL GRAPHPAK, has continued to evolve in the form of two IBM Field-Developed Programs,[3,4] an IBM Programming RPQ (a customized program),[5] and, as recently announced, a component to be included in future releases of VS APL. This most recent embodiment provides support for IBM's newest graphics products, including the IBM 3279 Color Display and IBM 3287 Color Printer.

Experience derived from the use of the package has played a major role in its evolution. In particular, its embodiment as the IBM 5100 APL GRAPHPAK served to broaden its applicability across a range of graphics devices and its utility among a wide spectrum of users. Throughout this decade of experience the natural applicability of APL to presentation graphics has been recurringly demonstrated. This is a consequence of characteristics of the APL language and its implementations. Finally, the experience has identified challenges for the future, specifically in the area of graphic data structures and in developing APL formulations of algorithms for specific problem areas. Overall, however, the experience has solidified an architecture and program structure for addressing presentation graphics.

Concepts and facilities

A key requirement associated with the development of recent versions of GRAPHPAK was the ability to utilize a wide variety of plotting and display devices with a minimum amount of customization programming. These devices differ in the content and structures of their order sets, in their communication protocols, and in their display space addressing schemes. All of these considerations differ widely among devices, largely stemming from original requirements for device design and the experience and ingenuity of the device designers. While the last consideration (display space addressing) can be numerically characterized, it is difficult to do so for the first two; rather, a procedural characterization is required.

The approach used to address this problem was to separate device-dependent functional requirements in simple device support functions within the scope of the design capabilities. The division represented three common areas of required function among most devices:

1. Encoding of orders for graphic vectors.
2. Encoding of orders for hardware-generated character strings.
3. Handling communication with the graphics device.

This functional division has proved useful recurringly in customizing device-dependent support for more than a score of devices and system environments.

The typical user need not be aware of these device-dependent functions since the lowest-level, user-accessed functions are at a higher level:

1. *DRAW M* displays a set of graphic vectors whose attributes and coordinates are defined by the APL matrix *M*.

2. *P WRITE C* displays sets of character strings, *C*, whose positions, sizes, orientations, and attributes are defined by *P*.

Of course, *DRAW* and *WRITE* ultimately call isolated, lower-level, device-dependent functions in a disciplined manner.

display spaces

The display space addressing consideration is handled more straightforwardly because it can be quantified numerically; however, an associated architectural decision affecting usability was required. The fundamental addressing problem is caused by wide variations in the way in which devices expect to receive coordinate specifications. For example, widely used devices use one or another of the following coordinate ranges: 0–511, 0–1023, or 0–4095 (all related to binary addressing schemes). Others use decimal schemes, addressing the range 0–9999, for example. Moreover, while most devices have uniform spatial resolution (equal distances per addressing unit in the x and y directions), some do not. For example, the IBM 5103 Printer, which can be used as a plotter,[6] has resolutions of approximately 3.9 and 2.7 addresses per millimeter in the x and y directions, respectively.

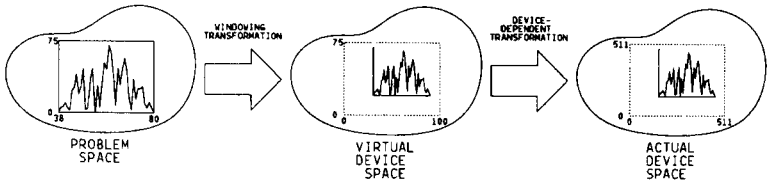
The general problem of variation is handled by introducing the concept of a *virtual device* with a *virtual display space* characterized by a common address range among all actual devices. The actual display space range and limits are defined by four numbers carried globally in the APL workspace. These quantities are sufficient to quantify devices that are rectangular and whose minimum coordinates are referred to by (0, 0).

Selecting an appropriate common denominator—the range of coordinates characterizing the virtual display space—is not obvious. At the time of selection, the graphics community generally argued that either the ranges 0 to 1 or -1 to 1 should be used.[7] Reviewing the pros and cons of those suggestions motivated an assessment of how coordinates were used at this relatively low level of GRAPHPAK. The decision made was influenced largely by the novice user who frequently defines coordinates manually via a keyboard. The range 0 to 100 was selected because of its identification with percentages and its ability to represent coordinates without negative signs and frequently without decimal points. (The upper limit of 100 does not limit resolution since the user has the freedom to use nonintegers if desired.)

The next step was to define facilities for mapping from problem spaces to the virtual device space as depicted in Figure 1.

A problem space (characterized by real numbers frequently called *world coordinates*) is generally preferred by high-level users. A user will typically define, explicitly or implicitly, a rectangular region in the problem space called the *window* (defined

Figure 1 Fundamental spaces and transformations



by the coordinates of the lower left and upper right corners) which contains the coordinates of interest. The window illustrated in Figure 1 might be used, for example, if a user was interested in plotting the number of people joining a company in each of the years from 1938 through 1980. A *windowing transformation* serves to map problem space coordinates into a user-selected rectangular region called a *viewport* in the virtual device space. Finally, the virtual device coordinates are mapped into actual device coordinates by the device-dependent APL functions.

viewports

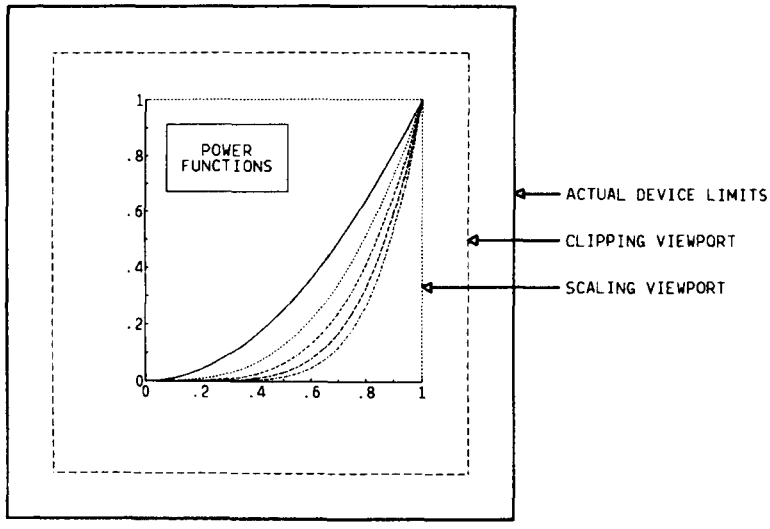
GRAPHPAK deviates slightly from common graphics practice[7] in its viewport conventions. Commonly, a viewport plays two roles:

1. With the window it completes definition of the windowing transformation.
2. It defines the limits of a rectangular boundary on which any crossing vectors are *clipped*.

The original and continuing major role of GRAPHPAK is to plot presentations of data made up of the presentation (e.g., a curve), a framework of axes, and "boilerplate" (labels and axis annotations, titles, etc.). A viewport used for scaling is generally perceived to coincide with the limits of the system of axes that will be drawn. Ultimately, however, a viewport used for clipping will have to extend beyond the region bounded by the axes to permit rendering axis annotations and labels. Explicitly extending a common viewport after drawing a curve and before rendering boilerplate seemed an unnecessary penalty. Hence, GRAPHPAK defines two distinct viewports (Figure 2): a scaling *viewport* used for scaling and positioning and a *clipping viewport* used for specification of drawing limits. The scaling viewport is specified by coordinates of its corners (usually, but not necessarily, the lower left and upper right). The clipping boundaries, however, are defined more generally by the homogeneous line coordinates[8] of the boundary lines. This permits generalizing the clipping region to convex polygons rather than limiting it to rectangles (the usual region).

The choice of the 0 to 100 virtual display space range and the distinction between scaling and clipping viewports appear to be

Figure 2 Viewports

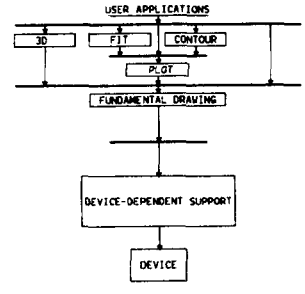


well accepted among GRAPHPAK users. A user who desires or needs to conform to the conventional single viewport approach can define a nearly trivial viewport setting APL function that will establish identical scaling and clipping viewports.

application structure

An additional facility in GRAPHPAK, one that evolved in the early years of development, is a structure on which presentation graphics applications can be built. The first application area that was addressed was general plotting, a pervasive requirement in engineering and scientific user communities. The package that resulted, which is described later in the paper, has proved exceptionally useful as a foundation for additional, higher-level applications. Figure 3 illustrates the present application hierarchy in GRAPHPAK and demonstrates the central role played by the plotting component.

Figure 3 GRAPHPAK component hierarchy



In particular, two high-level components—for contour plotting and curve fitting—are built on the facilities of the plotting component. Figure 4 shows a curve-fitting example; the curve-fitting component calculates the characteristics of the curve and calls on the plotting component to plot the curve, symbols, and axes. Then, the user has all of the facilities of the plotting component available for the production of boilerplate. Figure 5 shows a simple contour plot of a terrain. Here, the contour component derives the constant elevation contours and calls on the plotting component for the curves and axes. Figure 6 shows an application of the three-dimensional display component to view the terrain data in a different manner. Figures 5 and 6 illustrate the contrast in visualization techniques sometimes required to

Figure 4 A typical curve fit

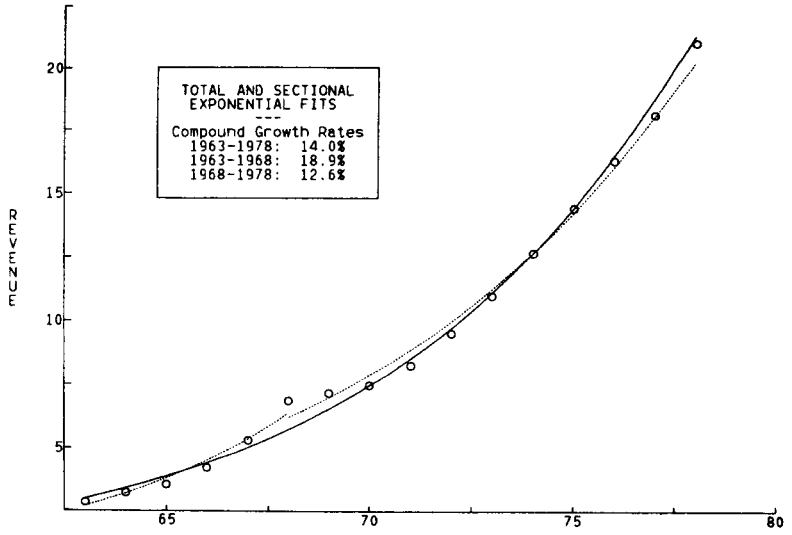


Figure 5 Contours for a terrain

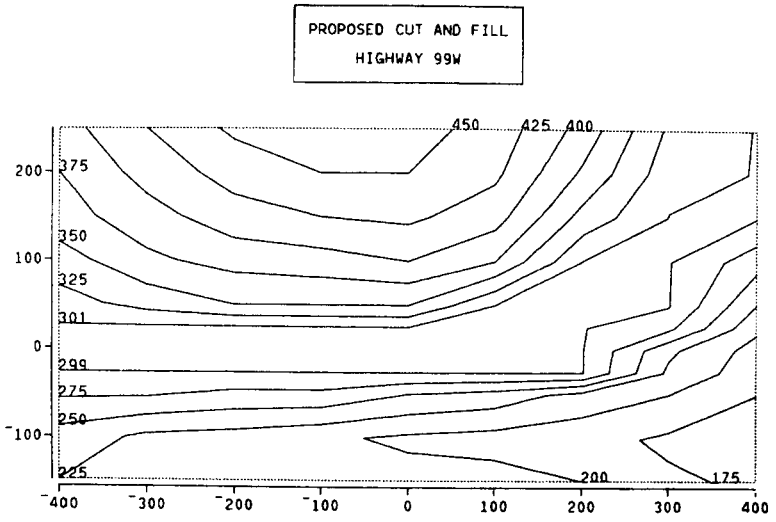


Figure 6 Alternative visualization of the terrain

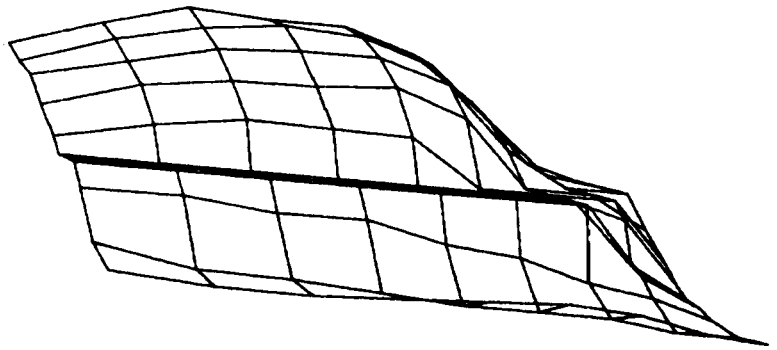
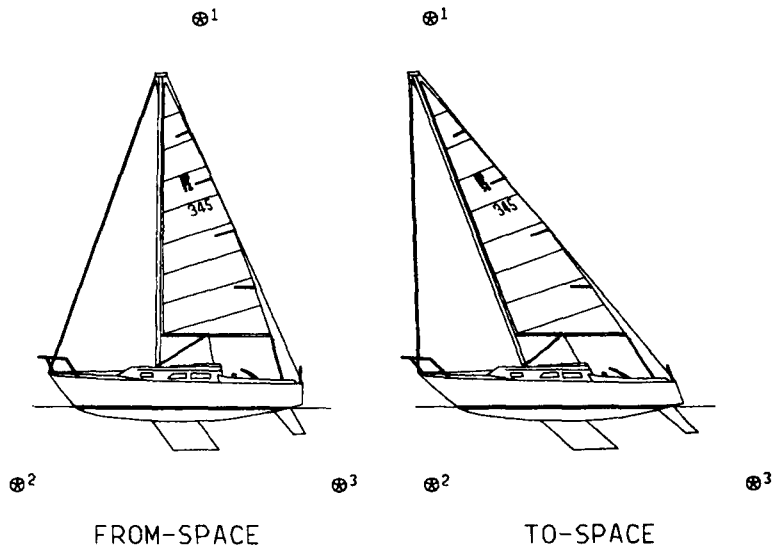


Figure 7 A skew transformation



present the same data to different audiences. Other kinds of applications can be added to the structure in a similar manner—generally with ease.

tool functions

Penetrating all levels of usage of GRAPHPAK are a variety of easy-to-use APL *tool functions*, which can be used solely for their defined purposes or in combination to serve as building blocks for new applications.

For example, the tool function *XFM* serves as a general-purpose coordinate transforming function. It derives from its left-hand argument, which is a two-plane array of corresponding coordinates in two spaces, a transformation which will map coordinates in one space to the other. It then applies the transformation to the right-hand argument, which is a matrix of coordinates, yielding a matrix of coordinates mapped accordingly into the other space. This function, whose simple implementation is based on the primitive *matrix division* and *inner product* functions of APL, can be quite powerful.

For example, if the left-hand argument represents correspondingly the coordinates of the two sets of numbered points illustrated in Figure 7, the transformation will skew the picture as illustrated. Moreover, if the transformation is "overdetermined" by specifying more than three points in each of a pair of two-dimensional spaces, a mapping is determined using the "least-squares" properties of APL matrix division.

In many cases, functions are provided simply to improve useability. For example,

```
DRAW (W INTO SVP) XFM OBJECT
```

is a single statement implementation of vector drawing with a windowing transformation. Here *W* and *SVP* are four-element vectors representing the window and scaling viewports. The simple function *INTO* structures the two-plane array argument required by *XFM* for the user who is not acquainted with APL arrays. Somewhat natural language structures like this, relatively difficult to implement with flexibility in other language environments, are quite easy for APL.

Addressing device attribute variability

In the previous section, three areas were identified in which available plotting and display devices differ widely, and the approaches for handling these differences were discussed. An additional area—variability in graphic attributes—deserves separate discussion.

The late 1970s saw an explosion in the application of technology to the graphic presentation of data. This explosion resulted in a corresponding explosion in the number of graphic attributes to be recognized by graphic support programs—attributes of vectors like style, width, and color, to name just a few.

In GRAPHPAK, attributes are selected by an encoded element of an attribute vector for both individual graphic vectors and character strings. For the most part, the variations have been handled pragmatically for each device supported. However, the variety of attributes selectable and experience using them has identified a need for a classification that will provide a unified treatment. The taxonomy presented here is used as a design guide; implementation proposals are evaluated against it for tests of orthogonality and completeness. At the highest level of the taxonomy, attributes are classified into three general families: appearance attributes, temporal attributes, and reference attributes.

appearance attributes

In the family of appearance attributes are those that directly affect the appearance of graphics vectors. Within this category are *spatial modulation* and *color*. Spatial modulation is a term selected to describe patterns or styles of spatial change associated with a graphics vector. Modulation may be *axial* or *transverse*, that is, vary along the length of the graphics vector or perpendicular to it. Combinations of axial and transverse modulation, of course, encompass the familiar attributes of style and thickness. The concept of modulation extends naturally to two-dimensional graphic entities (filled polygons). Color is a general-

ization of intensity and can be represented by a vector of descriptors in a selected color space or by a color number. The specification and treatment of color is far from a trivial consideration. For example, for many of the color display devices available, a color is selected by specifying relative proportions of the red, green, and blue primary colors. This method of specification, however, is not particularly suitable for user specification. (As a mental exercise, the reader might consider what proportions of these primaries would come close to matching the color of the reader's office walls.) A more intuitively appealing method may be to use the perceptual descriptors: hue, saturation, and lightness.

temporal attributes

Two common approaches to graphic display devices have involved the use of refreshed cathode-ray tube technology (e.g., IBM 3250) and direct view storage tube technology (e.g., Tektronix 4015). The IBM 3277 Graphics Attachment RPQ[9,10] is an embodiment that permits use of both the refreshed and storage approaches. Since the time behavior of graphics vectors is influenced by these approaches, the attributes that characterize this behavior are termed temporal. Two temporal attributes are identified: *persistence* and *periodicity*.

Persistence quantifies the longevity or permanence of a graphic vector, while periodicity reflects its frequency of recurrence. In the context of the IBM 3250, for example, the employed technology is characterized by relatively low persistence, hence vectors must have relatively high periodicity to be seen at all. However, where direct view storage tubes are used in their normal storage mode, vectors are highly persistent, hence need not be refreshed or be periodic at all.

reference attributes

The third attribute family also has been motivated by characteristics of the IBM 3277 Graphics Attachment RPQ. That hardware provides facilities for referencing an unbound, operator-controlled coordinate held in a *tracking register* and for defining vectors with respect to it. These capabilities can be encompassed by a class of attributes called reference attributes that characterize the independence of coordinates and the base to which they are referred.

The reference attribute of *mobility* characterizes the degree to which a coordinate is bound. Usually, coordinates are completely immobile; that is, they are bound as constants prior to the time of display. The tracking register referred to above is an example of a mobile coordinate.

The attribute of *relativity* has been associated with graphics since its beginning, and it becomes an essential attribute to take practical advantage of mobile coordinates. Specifically, relativity characterizes the origin of reference for a coordinate. In most casual

graphics applications, coordinates are *absolute*; that is, their base of reference or relativity is to the origin of the base or *world* coordinate system.

The benefit to be derived from an attribute classification like this one is that a wide variety of potential and device-independent attributes can be associated with the graphic vectors themselves. Moreover, the attributes, geometry, and topology associated with a picture can then be implemented as a rectangular array—a distinct advantage for an APL implementation.

Applicability of APL

APL is an interesting language and seems naturally applicable to implementing algorithms related to presentation graphics. This aspect should not be unexpected, since among APL design principles are goals of generality and practicality.[11] This applicability can be seen in several forms—its adaptability to implementing algorithms and its orientation toward interactive use.

In particular, the native capabilities of APL adapt themselves nicely to many of the techniques found in the graphics literature that are matrix formulations of techniques for transforming homogeneous coordinates (e.g., References 8 and 12). For example, if XY is a two-column matrix of coordinates of ordinary points in two-dimensional space, the expression

```
0 1⍉(XY,1)+.×C
```

relates a transformation of those points controlled by elements of matrix C. C itself may be a compound transformation derived, for example, from the inner products $C \leftarrow T +.\times R +.\times M$, where T, R, and M are 3-by-3 matrices for the fundamental transformations of translation, rotation, and magnification, respectively. The transformations can be achieved as well in three dimensions where the transformation matrices are 4-by-4 matrices. In fact, in this case, perspective projection can be achieved by a matrix inner product

```
XYZ←(XYZ,1)+.×P
```

followed by a division

```
XY←XYZ[;1 2]÷XYZ[;4 4]
```

where matrix P contains a definition of an observer's position (0, 0, h) on the z axis as

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -(1 \div h) \\ 0 & 0 & 0 & 1 \end{matrix}$$

Some techniques important to graphics are not so straightforwardly implemented in APL. In some cases, algorithms outlined in the graphics literature are not satisfactory when transliterated into APL implementations. This occurs because frequently the algorithms were originally formulated with a bias toward more traditional programming languages and disciplines. In these cases, it is often rewarding (for both performance and understanding) to rethink the requirements of the problem with an eye toward the fundamental nature and capabilities of APL.

The interpretive implementations of APL permit more flexibility in adapting a program for human use than would be possible in the more traditional, compiled program environment. These characteristics will be illustrated by a discussion of the GRAPHPAK plotting component.

The plotting component was originally designed to serve as a general tool for plotting APL variables, and, to a large degree, it was modeled after the printer-plotting workspace originally provided with APL program products.[13] Its characteristics have been popular among users. For example, it permits simple syntactic constructions like

```
PLOT XY
```

where *XY* is a matrix whose first column defines the abscissa values and whose second and possible subsequent columns define sets of ordinate values; or

```
PLOT Y
```

where *Y* is a vector of ordinate values, and an index vector is used by default for abscissa values; or natural language structures like

```
PLOT GROSS AND NET VS TIME
```

Notice that, in these examples, the user is expected to know only enough APL to put his data in the form of APL variables. But, as well, the proficient APL user may generate data in his own APL functions along with interspersed calls of GRAPHPAK functions like *PLOT*. This procedure is to be contrasted with the so-called conversational approach for the novice user where a "question and answer" procedure is utilized. By taking the nonconversational, APL "tool function" approach, an APL application programmer can easily build conversational programs on top of the GRAPHPAK base to be used by someone who knows no APL at all. To do the reverse is much more difficult.

The high degree of interactivity that characterizes current implementations of APL plays an important role in the development of useful plots. It enables users to make spontaneous changes based on judgment and desire. For example, after plotting a curve of

data (*PLOT Y VS X*) , if a user is curious about the behavior of *Y* with respect to the *reciprocal* of *X* , the curiosity can be satisfied merely by typing *PLOT Y VS ÷X* . This ability to make spontaneous changes is undoubtedly the most important single factor that makes APL GRAPHPAK an attractive tool.

In all of these cases illustrated above, *PLOT* examines the data and derives *x*- and *y*-scaling factors that will ensure that the plotted data will fit within the plot area. The factors are chosen in general so that the data will fill the frame as fully as possible while maintaining reasonable value labels to be associated with axis tick marks. This automatic scaling approach avoids the necessity for the user to specify scale factors or limit values. Early experience with *PLOT* showed that users did not necessarily want *PLOT* to produce axis labels, annotations, title blocks, etc. (to be referred to as "boilerplate" for ease of reference). For example, the user may determine on inspection of the picture that the argument provided to *PLOT* was incorrect and may not want to wait for rendering of the boilerplate. For this reason, GRAPHPAK always renders the data representation first, then requested boilerplate. This approach can be especially important if the plotting device is as slow as a mechanical plotter.

After the desired data representation is rendered, the user has the ability to add customized boilerplate at will through the use of appropriate APL functions that are provided.

This characteristic of avoiding binding program behavior into a fixed, single-minded sequence gives the user the flexibility needed to respond appropriately to the appearance and form of the visual presentation.

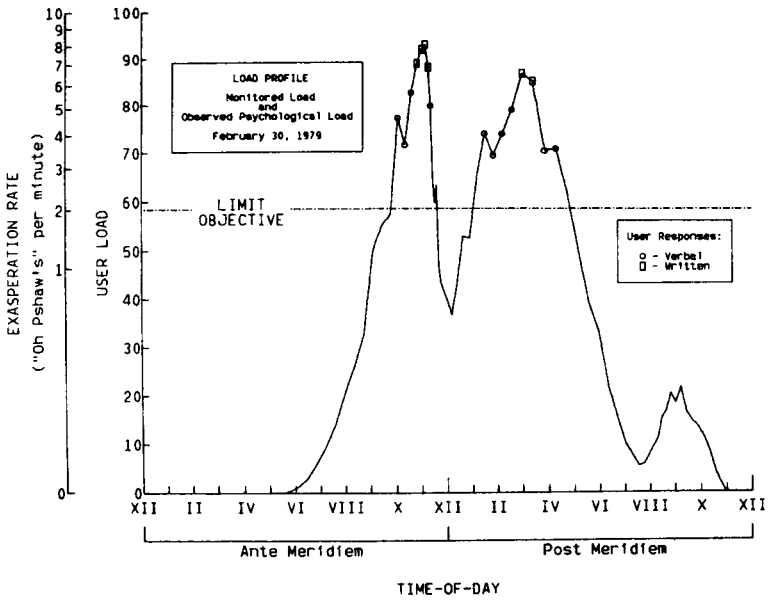
For the cases where the user desires to specifically override default characteristics, a dyadic form of *PLOT* called *SPLOT* (for "Specified Plot") is provided. Its left-hand argument permits:

1. Specifying the plot frame limits to be used (overriding automatic derivation)
2. Inhibiting drawing of axes
3. Producing plots using symbols and/or vectors
4. Controlling the attributes of the vectors produced
5. Selecting linear or logarithmic axes
6. Automatically producing axis labels

In addition, a variety of "tool functions" are provided for the production of boilerplate. They include:

1. An axis generator that controls tick mark placement, tick mark length and attributes, and axis offset

Figure 8 A plot rich in "boilerplate"



2. Corresponding axis label generators for producing arbitrary labels
3. Corresponding axis annotation generators
4. A title block generator

The plot functions, with the tool functions, permit the production of rich plots like the *atypical* one shown in Figure 8.

The general and flexible nature of the plotting component permits it to be used as a tool for implementing higher-level applications. The curve-fitting and contouring components in GRAPHPAK are good examples.

Challenges

For the most part, the combination of APL and presentation graphics represents a happy marriage, and there is great potential for extension and enrichment. However, there do exist design and implementation challenges. Three general areas of challenge can be identified: (1) device independence, (2) graphic data representation, and (3) unification.

challenge of device independence

General approaches must be sought to achieve device independence. The variations in device characteristics must be subsumed in a general way by the representation of graphic elements of APL.

A promising approach is to use the notion of element attributes described previously, where attributes are defined by a collection of orthogonal and universally appropriate characteristics.

challenge of graphic data representation

The variety and nature of the characteristics of families of graphic elements motivate a natural requirement for data structures more general than present APL arrays. The concepts of proposed APL *general arrays*[14] are promising, permitting the representation of attribute, geometric, and topological information by non-homogeneous, list-like structures. General arrays may be ideal for representing fundamental graphic elements, segmented display entities, and higher-level graphic structures. The challenge is to discover the best means of representing the desired structures and manipulating them with the proposed facilities.

challenge of unification

As the thought concerning the marriage of APL and graphics grows in maturity, it becomes more appropriate to think about treating graphics entities as more primitive elements of the language. Graphics entities are defined by numerical, attribute, and topological information. The challenge is to enclose this information in a manner that represents a natural, compatible, general, and useful extension of the present language. Approaches should seek facilities for structuring and transforming graphic entities and for communicating them to and from visualization devices in a consistent framework that is in the spirit of APL.

An additional broad area of challenge is the continued search for formulations of graphics-related algorithms that promote natural (hence usually efficient) implementations in APL.

Finally, there is the challenge of providing facilities that are genuinely easy to use. The fundamental approach of GRAPHPAK has been largely successful: It uses a simple set of command-like APL statements to maintain the flexibility and generality characteristic of APL, while retaining the ability to use functions as tools or building blocks to develop new, unforeseen applications.

Satisfying these challenges should involve an ideal: unifying a two-dimensional means for communication—graphics—with a technique for communication—APL. The language, influenced by somewhat traditional applications and by one-dimensional communication means, has proved admirably adaptable to presentation graphics. But it is the spirit of APL that will motivate discovery of ways of satisfying the challenges that are characterized by generality, richness, and utility. And the nature of APL and its current implementations will facilitate exploring possible approaches, evaluating alternatives, and ultimately meeting the challenges.

CITED REFERENCES AND NOTES

1. A. W. Burks *et al.*, "Preliminary discussion of the logical design of an electronic computing instrument," *Collected Works of John von Neumann* **V**, Section 6.8.3, 34-79 (1963).
2. *APL Language*, GC26-3847, IBM Corporation; available through IBM branch offices.
3. *GRAPHPAK—Interactive Graphics Package for APL\360: Program Description/Operations Manual*, SB21-0412, IBM Corporation; available through IBM branch offices.
4. *IBM 5100 APL GRAPHPAK: Program Description/Operations Manual*, SB30-0805, IBM Corporation; available through IBM branch offices.
5. *IBM 3277 APL Graphics Attachment Support PRPQ: General Information Manual*, GH20-2147, IBM Corporation; available through IBM branch offices.
6. *IBM 5100 APL Print Plot Problem Solver Library: User Manual*, SA21-9264, IBM Corporation; available through IBM branch offices.
7. "Status report of the Graphics Standards Planning Committee," *Computer Graphics* **13**, No. 3 (1979).
8. D. V. Ahuja and S. A. Coons, "Geometry for construction and display," *IBM Systems Journal* **7**, Nos. 3 & 4, 188-205 (1968).
9. *IBM 3277 Graphics Attachment RPQ: General Information Manual*, GA33-3039, IBM Corporation; available through IBM branch offices.
10. D. F. McManigal and D. A. Stevenson, "Architecture of the IBM 3277 Graphics Attachment," *IBM Systems Journal* **19**, No. 3, 331-344 (1980, this issue).
11. A. D. Falkoff and K. E. Iverson, "The design of APL," *IBM Journal of Research and Development* **17**, No. 4, 324-334 (1973).
12. W. M. Newman and R. F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill Book Company, Inc., New York (1973).
13. The printer-plotting workspace is currently distributed as PLOT with VS APL (IBM Program No. 5748-AP1) or PLOTFORMAT with APL.SV (IBM Program No. 5799-AQC).
14. See, for example, Z. Ghandour and J. Mezei, "General arrays, operators and functions," *IBM Journal of Research and Development* **17**, No. 4, 335-352 (1973).

The authors are with the IBM System Products Division, P.O. Box 6, Endicott, NY 13760.

Reprint Order No. G321-5131.