The development of interactive graphic application programs, designed for fast response, high productivity, and moderate system load, is difficult and time-consuming. Therefore, a structured approach has to be employed using function distribution in system design and application support program development.

This paper describes a comprehensive interactive graphic system that provides an environment for development and execution of graphic applications. It features an interactive graphic command language, a hierarchical structure of system, semantics, and storage, a set-oriented data concept, and library facilities.

A graphic interactive application monitor

by J. H. Bleher, P. G. Caspers, H. H. Henn, and K. Maerker

Interactive computer graphics is a powerful productivity tool of tremendous potential.[1] A number of excellent workstation products now available on the market and a continuously improving price/performance ratio of computer mainframes support the introduction of computer graphics into more and more industries.

Figure 1 Comparative aspects of human intelligence

Definition of Human Intelligence

- Prudence
- Comprehension
  Perceptiveness
  Judgment
- Adaptability
  Learning ability

Significant efforts still have to be undertaken in system and application software development to provide simplicity of installation and ease of use for problem-solving graphic application systems. Especially in smaller corporations and in branch offices where trained personnel with programming skills are at a premium, the decision to introduce graphic applications, among other considerations, very much depends on the *intelligence* of the graphic system offered.

Intelligence in this context describes how graphic workstations, local or central mainframe processors, and system and application programs support the dialogue between the user and his graphic and associated alphanumeric data to do the job.

Figure 2 Intelligent graphic system

Man-machine interface
Adaptive interactive user language

Elements of communication	*Characters, Words*	*Pointing*
Semantic relationship	Linear	Multidimensional
	Sequential time related	Parallel environment related
User interface Input devices	Alphanumeric keyboard (Mnemonic) Function keyboard (Menu)	Light pen Cursor/crosshair controlled by joystick, tracking ball, or mouse Data tablet (Menu)
User interface Output device	Alphanumeric display	Graphic display

Intelligent graphic systems

The intelligence of a graphic system can be defined by

1. The available *means of man-machine communication and interaction*, which is comparable to the power of comprehension of a human being (Figure 1). Particularly useful is an interactive user language for communication with both alphanumeric information and geometric representations (Figure 2). The physical user interface in the form of keyboards and pointing devices allows input and manipulation of alphanumeric and graphic data, which are presented on displays.
2. The support provided to the user in *selecting the solution out of ambiguous possibilities* to solve his or her particular application problem, which is comparable to elements of human judgment. Examples are the selection of ambiguous intersection points of graphic elements, such as lines and circles, and selection of curves and shapes out of a set.
3. The *dynamic adaptability of the workstation functions* to individual user needs through interactive programming, which is comparable to the adaptability of human individuals. This adaptability asks for predesign and implementation of a com-

prehensive set of generic system and application functions into the graphic system. It also requires graphic application software to be written in an easy-to-use, application-oriented, high-level language to allow extensions in a quick and simple way, if necessary.

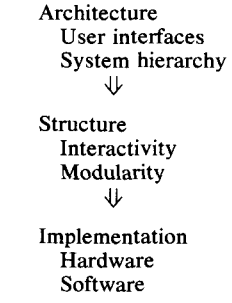
The intelligence of a graphic system is reflected by the way alphanumeric and graphic data are gathered, processed under user control, and presented to support different applications in changing environments.

ease of use

Intelligent graphic systems must be able to adapt easily to user requirements to be immediately accepted for the jobs at hand. Several system parameters are critical for ease of use:

1. The *system response time* tolerated by the user is related to the complexity of the task performed, as perceived by the user. The designer at a graphic workstation is not willing to tolerate a long response time for seemingly trivial manipulations *while disregarding the computing power required for execution* but gladly waits for the results of an obviously powerful transaction.
2. *Application-tailored data input procedures* must be able to avoid trivial repetition of commands. These commands must be replaceable by user-definable command sequences and macros (macroinstructions) with parameters automatically referenced from tables. *User-definable function keys* and prompting features must be available. Identification, selection, and manipulation of graphic entities must be possible for single objects and user-definable aggregates. *Dynamic graphic object-dragging and rubber-banding features* are highly instrumental in producing good interactivity. *Menu techniques* are very helpful for tasks with complex procedures and for casual users.
3. *Failure tolerance or fail-soft features* are required to help the user to recover from operating errors. Adequate failure messages, HELP facilities and user-initiated checkpoint/restart features (ACCEPT/REJECT), must be available. System failures must be covered by automatic facilities allowing different levels of checkpoint/restart as required with data protection and system recovery.

Figure 3 Hierarchical system concept



From these requirements it is quite obvious that only a well-structured system and a balanced combination of hardware and software are able to achieve such a high degree of intelligence in a graphic system.

A *hierarchical system concept* allows the necessary implementation flexibility (Figure 3). A *modular system structure* will provide well-defined interfaces and a high level of freedom for *hard-*

ware or software implementation of specific system or application functions. Care has to be taken not to increase program path lengths and system response time through inefficient partitioning.

By these means the design of an intelligent graphic system can be made generic and almost independent of technology (hardware).

structural requirements

A high degree of interactivity and short response time require at least some local intelligence in the graphic workstation as follows:

1. Transmission buffer for transfer of alphanumeric characters and vectors from the computer mainframe to the workstation with appropriate line protocols at different line speeds suitable for a particular application.
2. Line and character generation logic supporting different line types and line widths. Very advantageous is a loadable character generator for user-defined fonts and symbols.
3. Image (refresh) buffer, required for vector or raster displays but not needed for storage tubes. For storage tubes with write-through capability, the refreshing of a limited number of vectors can be accomplished directly from the transmission buffer.
4. Interactive cursor/light-pen support.
5. Address calculation logic to support relative vectors for dynamic image manipulation (shape dragging, rubber-banding) is highly desirable.

There are other functions, which can be located either at the workstation or at the computer mainframe depending on the interactivity required and the workstation price justified:

6. Support of high-level graphic objects like circles and conic sections.
7. Coordinate transformation logic or programs for shift, zoom, rotate, pan of graphic images.
8. Logic for identification of graphic objects associated with pointing information.

The central intelligence of the host computer will handle less time-critical functions:

9. Complex alphanumeric and graphic input processing.
10. Functions to reformat, transform, and subset graphic data entities for manipulation and display.
11. Application programs.
12. Fail-soft features (checkpoint/restart).
13. File handling and data base support with access authorization and multiuser access control.

The boundaries between these groups are flexible and closely related to performance and price of available graphic workstations and computer mainframes. Therefore, the system structure must be modular, such that the distribution of functions between computer mainframe and workstation can be optimized.

Graphic interactive application monitor

With the above requirements of an intelligent graphic system in mind, a graphic program and programming system, the Graphic Interactive Application Monitor (GIAM),[2] has been developed, running on System/370 computer systems that support IBM 3277 GA workstations (Figure 4). GIAM is based on the operating system VM/CMS (Virtual Machine/Conversational Monitor System), using VS APL as the implementation language.

GIAM contains two program packages providing two different levels of support:

1. BGSS (Basic Graphic Support System) handles alphanumeric and graphic input and display and provides some basic graphic interactive support functions.
2. BIGAM (Basic Interactive Graphic Application Monitor) is based on BGSS and supports complex interactive application functions and provides a high-level interactive graphic command language interface to the user.

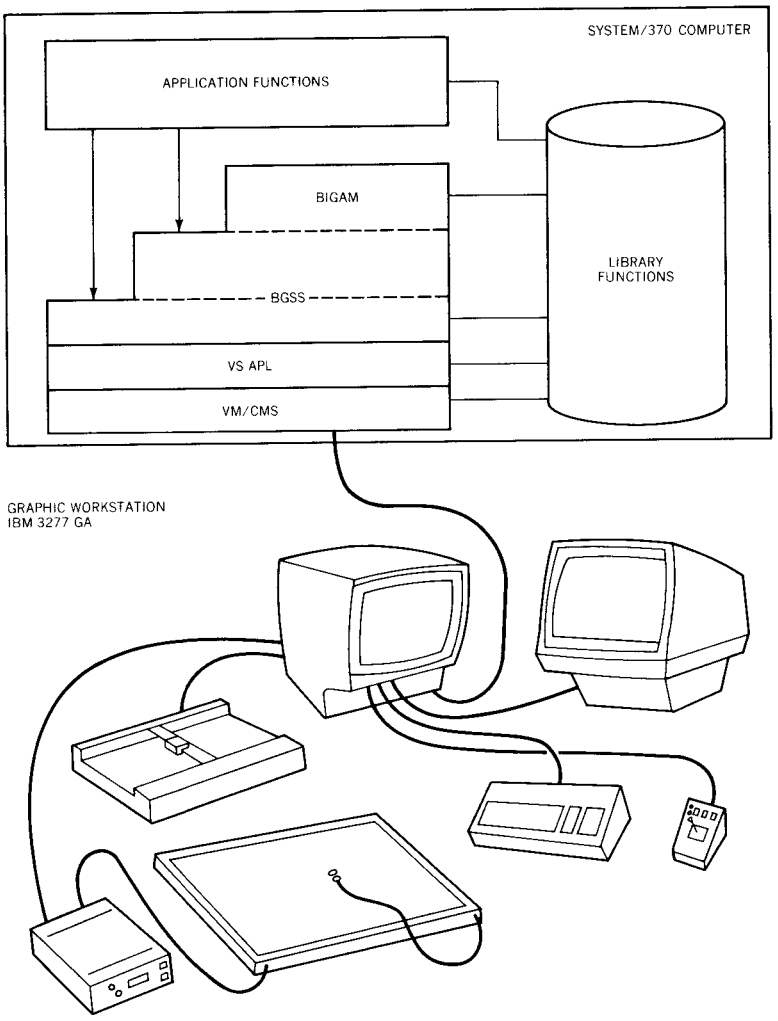
Both program packages are designed for easy personalization of specific application requirements by the user and for fast (interactive) execution of application functions. The hierarchical system structure contains library services on various levels.

The user interfaces are simple, allowing extensions to new sets of applications to be made rather easily. The APL implementation language is a key ingredient for these capabilities.

The 3277 GA dual screen graphic workstation (Figure 4) has a number of advantages for the man-machine interface in improving communication and increasing productivity:

- Alphanumeric and graphic representations can be separated. The graphic display will not be cluttered with commands and with text inputs prior to verification and editing or with system messages.
- The alphanumeric display can also be used for prompting, for menu selection with the keyboard or with a light pen, and for interactive programming, if desired, leaving an undisturbed geometric image on the graphic screen.

Figure 4 GIAM system structure



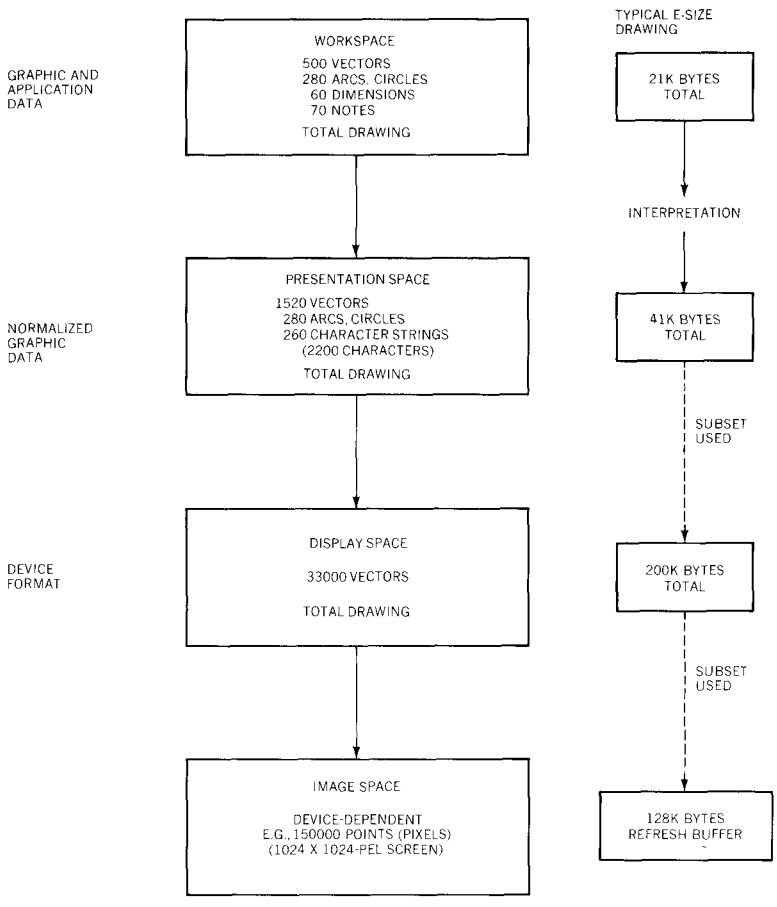
- Representation of graphic images and identification and manipulation of graphic object are accomplished on the graphic display using a cursor controlled by a joystick. Graphic input is also possible through a data tablet. Hard-copy output can be provided by a plotter attachable to the workstation.

However, GIAM is not limited to specific workstation hardware, but provides device- and application-independent interfaces through its general structure.

GIAM system and data structures

The complexity of graphic representations, e.g., engineering drawings, requires a well-defined organization and structure of graphic information. The prime data area of GIAM is the *work-*

Figure 5 Graphic data structuring example



space containing both graphic and other application-dependent data. The graphic data may not only be basic graphic elements (such as vectors, circles, arcs, etc.) but also higher graphic objects (Figure 5). The type and number of such objects varies from application to application. An engineering drawing, for example, may include (1) basic graphic elements showing the geometrical contour of the mechanical part, (2) dimensions consisting of leader lines, arrows, and numbers, and (3) notes containing alphanumeric information on the drawing.

When such a workspace is to be displayed, the higher logical objects have to be broken down into basic graphic elements through interpretation and semantic expansion. A subset of the workspace to be used for current operations is mapped into a *presentation space* which now contains only normalized application-independent graphic data of the following types:

- Geometrical objects (i.e., vectors, circles, arcs, ellipses, ellipse segments)
- Patterns
- Text objects using both hardware-generated fonts and individually spaced stroke-coded software characters

These graphic objects are still device-independent. The presentation space coordinate system represents a virtual screen. In turn, a subset of the presentation space data actually to be displayed has to be converted to device code, i.e., to hardware commands that can be directly executed by the display device. These commands are stored in a *display space*.

For storage tube and vector-refresh display devices, the commands of the display space can be directly executed to generate the image on the screen. For a television monitor with pixel buffer, however, the picture has to be first generated point-by-point and stored in an *image space*. From this buffer, the screen can be refreshed about 50 times per second.

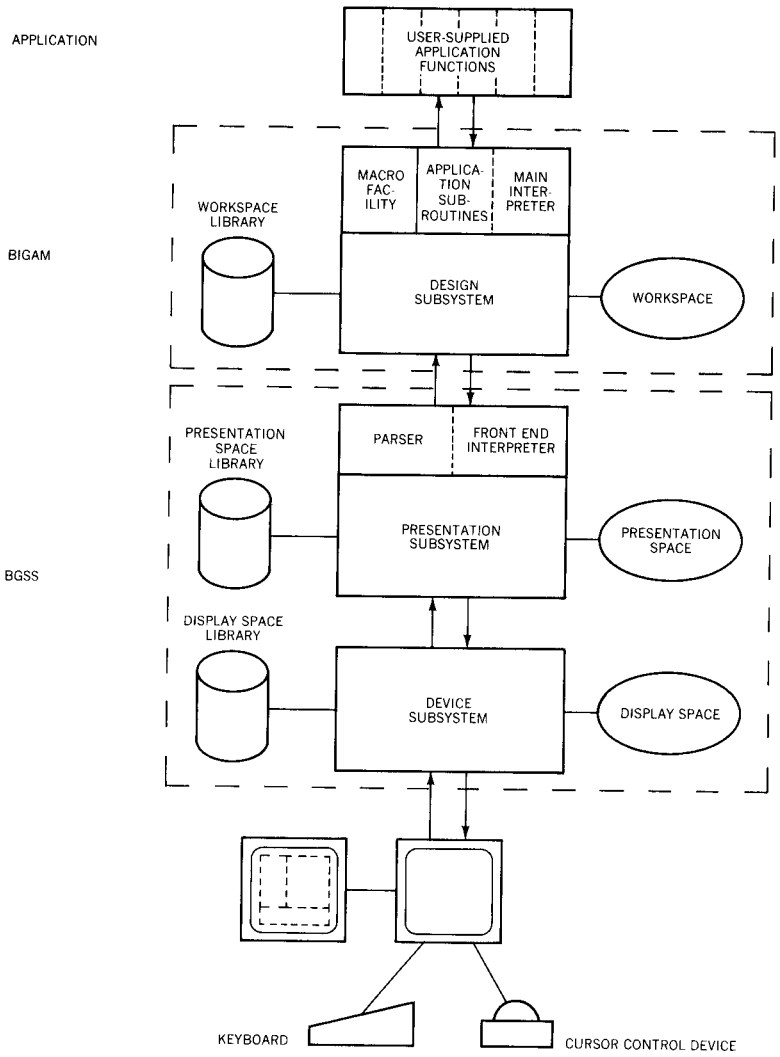
The distribution of graphic data is reflected in the hierarchical structure of GIAM (Figure 6). The program component BIGAM with its *design subsystem* manipulates the data in the workspace. These data are the master data of the application. Since the objects can have a large range of dimensions, all data are represented in double-precision floating-point representation, guaranteeing the accuracy needed for complex application calculations (e.g., calculation of intersection points between two circles).

This high precision is no longer required in the presentation space because the only calculations performed are directly related to the display. Since a screen has only a limited addressing capability (e.g., 4096 × 3072 points), fixed-point representation of data is sufficient, allowing faster and more efficient program routines. Thus, data are converted from floating-point to fixed-point representation when mapped from the workspace into the presentation space.

The presentation space is manipulated by the *presentation subsystem* which is one part of the program component BGSS. The other part is the *device subsystem* that converts data from the presentation space into device orders kept in a display space that is device-dependent. It performs the actual communication with the workstation devices. Other devices can be supported by modifying the device subsystem.

The hierarchical structure of GIAM provides three well-defined data interfaces for efficient and simple communication with application programs. The lowest level is the display space from which data can be sent directly to the device. Display spaces may be

Figure 6 GIAM hierarchical structure



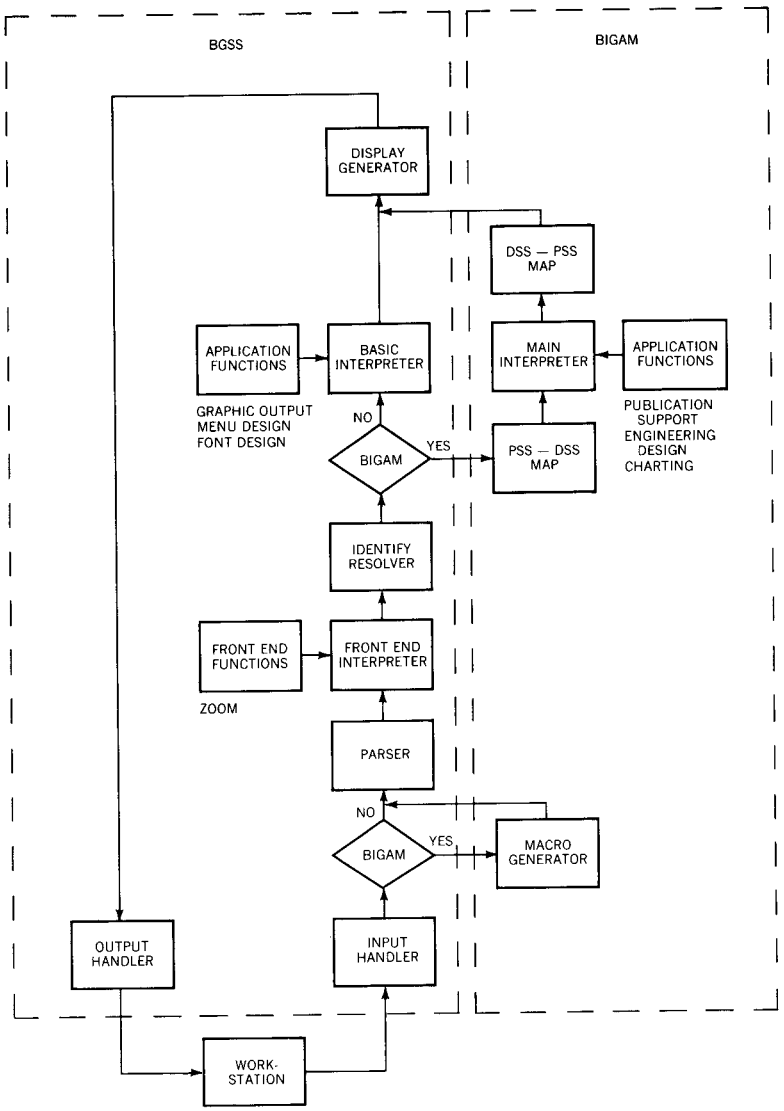
used for the presentation of forms, schematics, etc., which are basic background information for applications.

The next interface level is the presentation space. It may be used by an application if only low precision is required, e.g., in business and scientific data presentations.

The highest interface level is the workspace. It supports high-precision graphic data and higher-level graphic objects.

Figure 7 shows the flow of control in GIAM and the program modules involved during a typical transaction. Input from the work-

Figure 7 GIAM functional structure



station is processed and presented to the front-end interpreter. It handles those commands mainly concerned with workstation operations such as graphic picking and display formatting. Then the input is passed either to the basic interpreter or the main interpreter, depending on whether BGSS or BIGAM functions are used.

Command language

GIAM has been designed to provide a unified interface to a large variety of graphic applications and to achieve operator productiv-

ity through a language called the Interactive Graphic Command Language (IGCL). This language is intended to be used either by a trained operator or by an application programmer customizing the graphic system for a specific application and for untrained operators. It was not designed as a language for the graphic system programmer as was APLG,[3] for example.

The purpose of IGCL and its corresponding processors is to translate an input stream generated by the workstation operator or by a high-level application program into normalized input to application programs, which perform requested functions and initiate feedback displayed at the workstation. Thus low-level programming for new application functions and for adaptation of application functions to new workstation hardware is not required.

Although command language interfaces for graphic systems were proposed and implemented as early as 1966,[4,5] they are not widely used in graphic systems of today, mainly due to the fact that the proposed command languages could not handle the application data management aspects of graphic applications. IGCL provides an embedded APL system for the manipulation of application data as well as application programs.

The use of a command language implies that graphic input and output must be standardized in some way. In fact, application programmers have accepted standards for *alphanumeric* input and output like READLINE and WRITELINE in order to create application programs, which are independent from the particular terminal hardware. However, though many graphic standards have been proposed,[6] there seem to be no accepted standards with respect to *graphic* input and output, preventing the portability of graphic application programs.

An analysis of a variety of graphic application programs indicates that two basic graphic input operations cover almost all graphic input requirements:

1. Entry of a point not related to any displayed graphic object, which may or may not be located on an application-defined grid. This function is usually implemented via a crosshair cursor, a tracking symbol, or a pointer on a tablet.
2. Identification of displayed graphic objects for subsequent processing. This function is usually implemented via a light-pen pick or selection with an identification trap symbol.

In addition to these basic operations, GIAM provides an additional EXCLUDE input operation, which allows the operator to identify graphic objects that are to be excluded from subsequent processing. The EXCLUDE and the IDENTIFY (include) operators provide a powerful mechanism for interactive definition of sets of graphic

objects. The GIAM implementation uses rectangular trap areas that can be defined and placed under operator control.

GIAM provides two basic graphic output functions for display update and for highlighting selected sets of graphic objects. The display update function is implemented such that any transaction through which graphic data have been changed can be rejected. This feature is extremely important for the user interface, giving the operator confidence to explore the complex functions of the graphic system on a trial and error basis.

Using the interactive graphic command language, the operator can enter arbitrarily long command strings without entering a separator between subsequent commands. This gives the operator the ability to initiate either very simple or very complex transactions according to his skill. Command strings may be entered either directly via a keyboard or via tablet or menu picks.

The language elements of the interactive graphic command language are:

- Command identifier (mnemonic)
- Number
- Character string
- Expression
- Macro designator
- Explicit delimiter

The command language uses the APL character set. However, any character set is applicable within character strings or text variables in order to allow applications with special or multiple fonts. Identifiers serve to denote function invocations within the system: *Reserved identifiers* are defined with GIAM, and *application-defined identifiers* can be defined by the application programmer and may be used to invoke application functions. Reserved identifiers are used to invoke the standard input functions, e.g., *MG* (move grid = get point on grid) or *I* (identify graphic objects), or to control command execution, e.g., *BEGIN*, *END*, *REPEAT*, *ACCEPT*, *IF*, *ELSE*.

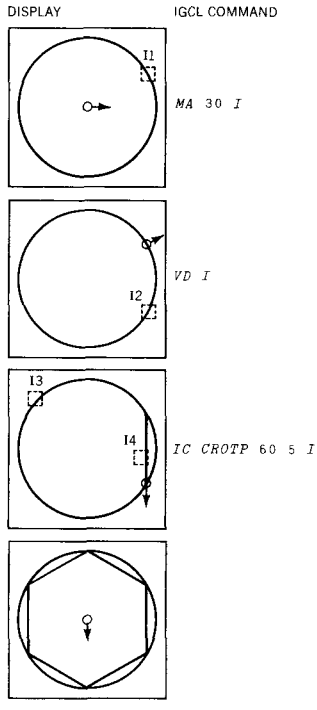
A *macro designator* specifies the activation of a macro. It consists of the identifier designating the macro and a list of actual parameters enclosed in parentheses, e.g., *BOX* (40 20).

A *number* is a valid number according to APL conventions. A *character string* is a sequence of characters enclosed by quote marks. An *expression* is a valid APL statement enclosed in parentheses, e.g., (2 × *X* - 3) or (*X* ← *SIN* 30). Blanks and ends of lines are considered as separators between language elements. The semicolon serves as *explicit separator*. It separates

command strings that have to be evaluated consecutively and is mostly used within macros.

A valid command may consist of an expression, a macro designator preceded by the EXEC or MACRO identifier, or a command identifier followed by any number of command identifiers, numbers, character strings, expressions, or explicit delimiters. Any number of commands can be entered in a command stream. Application functions can be invoked within expressions as standard APL functions or as BIGAM application functions which are invoked via command identifiers. Command identifiers, numbers, character strings or APL variables can be passed to GIAM application programs as parameters. The command language processor does not check the syntax of application-defined commands. Every GIAM application function has to check the syntactic and semantic validity of the input parameters. Because most GIAM application functions allow a large number of input parameter combinations, a system-provided syntax check is of limited value and would complicate the linkage of application functions.

Figure 8 Geometrical construction with IGCL



Most application functions that generate graphic objects require one or a number of points as user input. These points may or may not be related to objects already defined. The interactive graphic command language provides a set of move commands that allow the operator to define points with complex relationships to existing graphic objects on the fly. System functions are provided to resolve these point-defining command sequences into coordinates that are then passed to the application functions. Using this facility, an application function which, for example, was designed to draw a circle with the center point and a point on the circle as parameters, can handle a variety of geometrical constraints that usually must be explicitly defined and programmed in the application program. The graphic system also maintains a graphic parameter represented by a current point and an angle that are displayed with a dot and an arrow on the screen. The current point and angle are usually used by application functions that require at least one point as parameter.

Figure 8 shows a typical geometrical construction using IGCL. With the first command, *MA 30 I*, the current point is moved from the circle center to the intersection point with the circle and a line (invisible) that has an angle of 30 degrees passing through the center of the circle. The intersection point is computed with the full precision available in the system (double precision, floating point). The user indicates with the location of the identifying pick I1 which one of the two possible intersection points should be selected. The second command, *VD I*, draws a vector down to the intersection point with the circle. In this case, there is only one possible solution, and the location of the second identifying pick I2 is uncritical. The third command, *IC CROTP 60 5 I*,

moves the current point to the circle center (IC − I3) and copies and rotates this vector five times by 60 degrees to complete the hexagon. The current point in the circle center defines the center of rotation, and the identify operation I4 selects the set of graphic objects to be rotated.

A more skilled operator may perform the same geometrical construction with two transactions with the command strings:

```
MA 30 I VD I
IC CROTP 60 5 I
```

or even in a single transaction using the command string:

```
MA 30 I VD I ACC IC CROTP 60 5 I
```

Although IGCL is very concise, complex operations may require quite a bit of operator action. In order to free the operator from repetitive entry of often-used command strings, a MACRO facility is provided within IGCL. A circle with the inscribed hexagon can be drawn, e.g., with a single macro call *HEX* (20), provided the macro *HEX* is available in the system. A macro can be entered in BIGAM either via text entry using an editor or in an interactive-by-example mode. In order to define a macro *BOX*, which draws a rectangular box with width *W* and height *H*, the operator may enter the following commands:

MACRO BOX (*W*←40 *H*←20)	Start macro definition; define parameters *W* and *H*.
VR (*W*)	Draw a vector to the right of length *W*.
VU (*H*)	Draw a vector up the length of *H*.
VL (*W*)	Draw a vector to the left of length *W*.
VD (*H*)	Draw a vector down the length of *H*.
EOM	End of macro definition.

In this interactive macro definition mode, the operator can visually check every step and reject any command line in case of error. The above macro can be used immediately after definition with arbitrary parameters. *BOX* (20 20), for example, will generate a box with $W = 20$ and $H = 20$. Macros can be nested up to any level and are stored together with graphic data, application data, and application functions in BIGAM workspaces.

It should be pointed out that the end user (e.g., the untrained operator) does not have to learn the IGCL commands. The application programmer will personalize GIAM for the application by such means as function keys, menus, and mnemonics, which then become the application-specific repertoire of the end user.

Library functions and data base

Related to the hierarchy of subsystems in GIAM is a corresponding hierarchy of external storages with three library levels:

- A *workspace library* associated with the design subsystem
- A *presentation space library* associated with the presentation subsystem
- A *display space library* associated with the device subsystem

In these libraries, GIAM workspaces, presentation spaces, and display spaces, respectively, can be stored permanently during a work session for later reference. According to their different information content, they provide an additional tool to tailor GIAM to the special needs of an application.

A workspace primarily contains application data. Besides the geometric elements that make up the picture, it may contain structural information. This allows geometric elements to be grouped together to form higher-level objects organized in tree structures.

In general, BIGAM supports interrelated sets of geometrical and structural data, combined with user-definable BIGAM variables (geometric or nongeometric, e.g., standards, notes, dimensional information), as real data. However, it also supports virtual data consisting of BIGAM functions and macros (geometric or nongeometric, e.g., DRAW BAR GRAPH, SELECT MESSAGE, CALCULATE AREA) that are invoked at execution time.

Groups, variables, functions, and macros are named entities. They are stored in workspaces and may be directly accessed, copied, and evaluated.

Presentation spaces may contain reference data supporting the interaction of the user with the screen content. Furthermore, they may contain application reference data, such as fonts and patterns, and system reference data, such as menus. The graphic data are represented in a normalized device-independent form consisting only of basic geometric elements. Thus the presentation spaces can also be used as data interfaces to plotter programs for hard-copy output generation.

Display spaces contain device-dependent data that can be used to display application-specific patterns or overlays on the screen. This is graphic information which serves as reference only and is not for interaction (e.g., form layouts, basic design patterns, etc.). They are stored directly as device commands in the display space so that no display generation is necessary each time they are referred to.

For each of the three library types a set of the usual service functions such as LOAD, STORE, COPY, DROP, QUERY, etc. is provided. In addition, a three-level access control hierarchy is implemented for each of the three library types:

- Private libraries may only be accessed by the owner.
- Project libraries may be commonly used by a restricted group, e.g., people working at the same project.
- Public libraries are available for widespread use within an application or an installation.

Both public and project libraries may be protected by passwords.

Application function environment

As a basic application monitor, GIAM is open to any application requiring interactive graphic support. Although a lot of features are provided for tailoring the user interface (e.g., function keys, menus, macros) it may be necessary to introduce new commands and high-level application programs for the solution of application-specific tasks. According to its hierarchical structure, GIAM provides two well-defined environments for extensions by user-written application functions: the BIGAM environment and the BGSS environment.

Each environment is described by a set of data areas used for manipulation of graphic objects by application functions and a set of basic service routines to simplify such manipulations. Figure 9 shows the logical layout of the BIGAM application function environment. The arrows represent the flow of data; the names beside the arrows describe some of the service routines.

Graphic objects are stored as matrices in the workspace (Figure 10). There is one matrix for each type of graphic object, with one entry for each individual object. In addition, there is a further set of matrices, with a one-to-one correspondence to the above ones, which contain attribute information.

The workspace contains all the graphic objects of a picture. Most commands, however, are only applied to a subset of these objects. Such a subset may be selected by a sequence of picking operations on the screen. In this case, a set of indices to the picked objects is built up by GIAM in the Work Index (WI). This index may be used to select graphic objects either to be erased from the screen or to be transferred from the workspace to a work area called X-space for further manipulation.

The application function interacts with the graphic data in the X-space. If needed, additional reference information is available in

Figure 9 GIAM application function environment

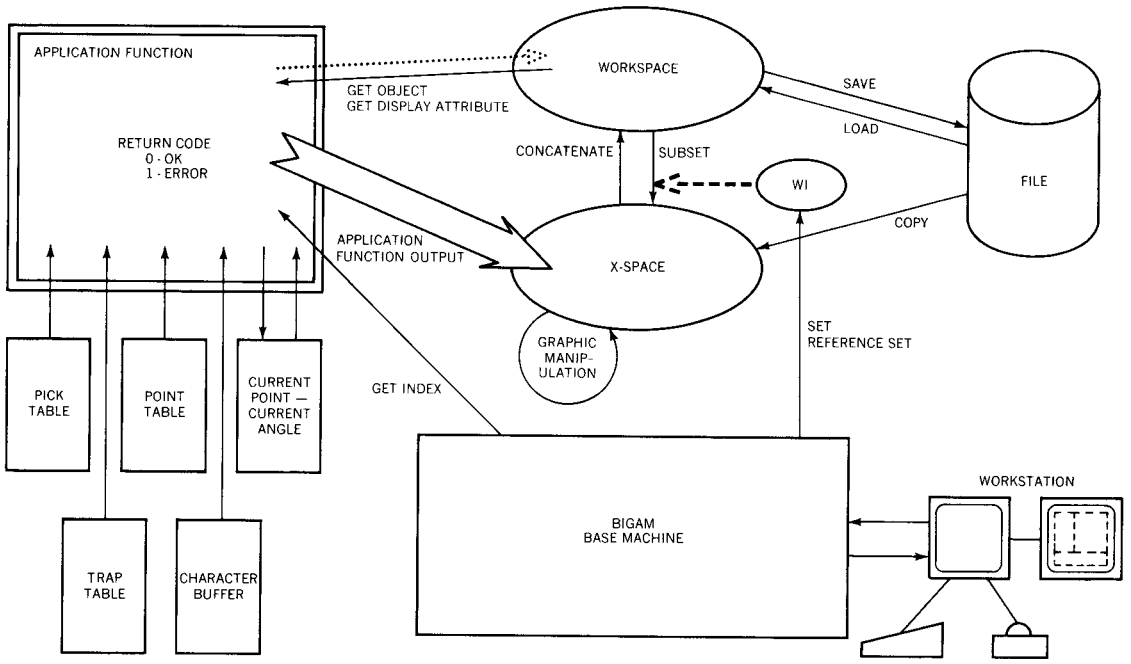


Figure 10 Generic BIGAM data formats

TYPE-ORIENTED GRAPHIC ELEMENT MATRIX
— DOUBLE PRECISION FLOATING POINT

GENERIC REFERENCE POINT	RELATIVE DIMENSIONAL ATTRIBUTES

RELATED DISPLAY ATTRIBUTE MATRIX
— 16 BITS PER ENTRY

ON/OFF	GRAY LEVEL OR COLOR	FONT TYPE	LINE TYPE	LINE WIDTH

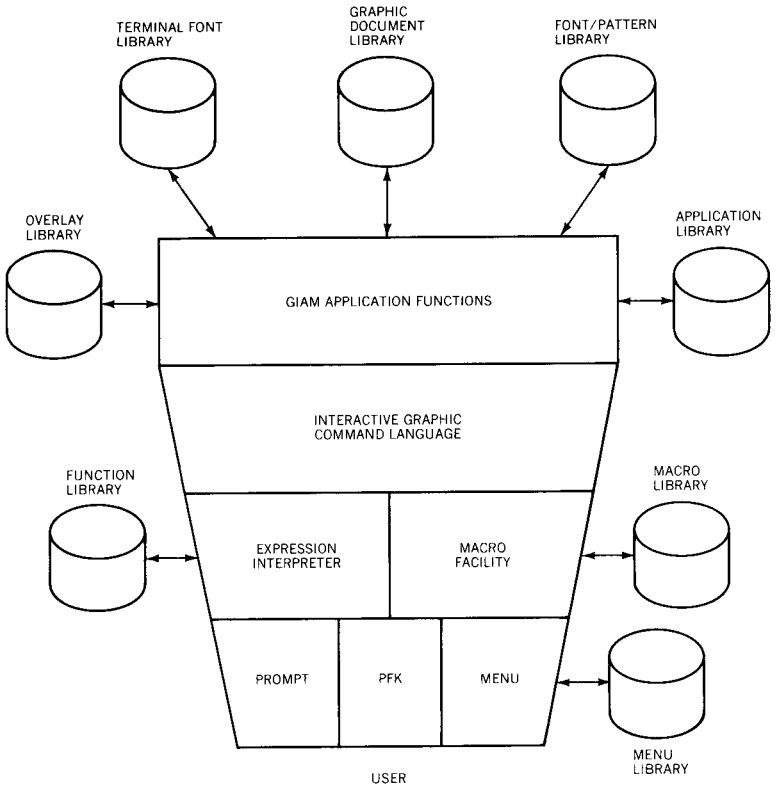
internal tables. Examples are the pick table containing the parameters of the function call in a standardized format and the point table containing the coordinates of points selected with the aid of graphic pointing commands.

Since GIAM is implemented in APL, the full power of this language is available to the user to program his specific functions. He has to comply only with a few simple GIAM conventions having to do with naming functions and variables, managing the pick table, assigning return codes, and displaying messages. The above-mentioned service routines assist the application programmer, providing a variety of basic facilities for managing and manipulating data in the GIAM application function environment.

Embedding APL into GIAM allows very productive programming of completely new application functions, e.g., isometric representation of parts ($2^1/_2$ D) and generation of input data for finite element analysis and for numerical control (NC) of machines.

After an application function has been written, it is linked to GIAM by means of a command utility. Then it can be used like any of the standard GIAM commands.

Figure 11 GIAM user facilities



Application-related interface tailoring

Graphic systems must be tailored to specific applications. An engineering drafting system for building-contractor use, for instance, has to support totally different procedures and drafting standards than an engineering drafting system used for mechanical design. Even within mechanical design and drafting applications, the standards may vary from country to country or even among corporations.

GIAM addresses this problem with a number of facilities that allow the customization of the general-purpose graphic system with a minimum of (APL) programming or with no programming at all. The general strategy is to use the graphic system itself for the customization and the documentation of the resulting special application system. A set of hierarchically organized tools provided by GIAM may be used for application tailoring as shown in Figure 11.

The customization usually requires macros as well as overlay forms and menus that are drawn with the general-purpose system and stored in menu and overlay libraries for subsequent use. The

macros can be defined either to optimize often-used procedures to boost operator productivity or to provide a simplified user interface with graphic and alphanumeric prompting and user guidance.

Standard libraries, which hold groups of graphical data (e.g., standard parts) as well as application data and functions, can also be created. A PROFILE macro may be used to customize the workstation, e.g., to define the function keys, the display parameters, and defaults of the graphic display. The PROFILE macro may also load application-specific fonts, menus, and reference data.

In addition to the BIGAM facilities for application tailoring, a set of BGSS facilities are provided. These BGSS facilities allow the application programmer to define symbols or special fonts that are used to program the graphic workstation hardware to improve the efficiency of the graphic system.

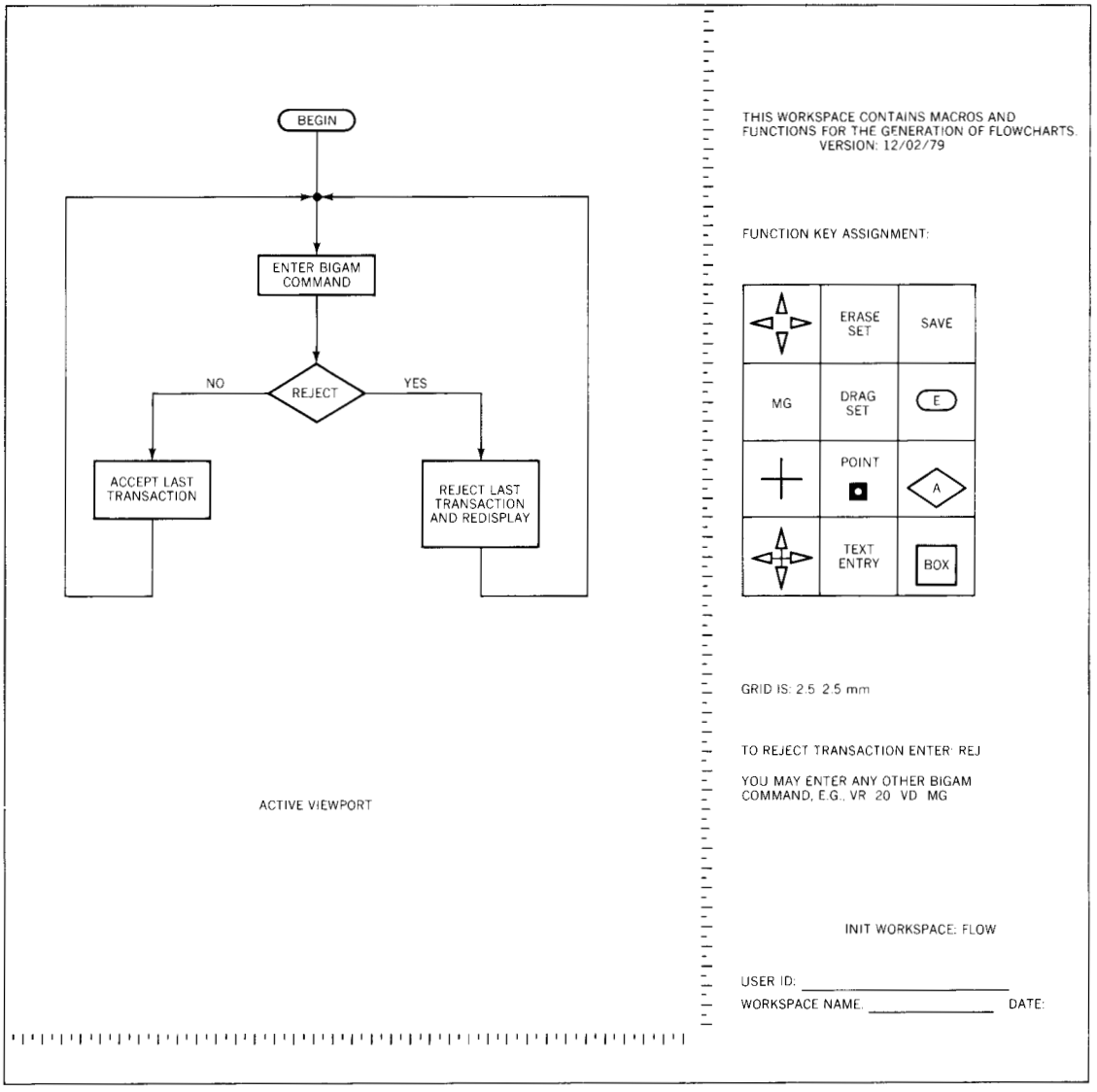
A simple example for application customization is shown in Figure 12. A number of elementary functions for the drafting of flowcharts are assembled in a BIGAM workspace. When the user loads this workspace, the PROFILE macro is executed, an overlay display space explaining the available functions is displayed, and the workstation is properly set up. Special application macros are invoked via function keys; e.g., function key 12 is used for the generation of a function box. When the operator hits this key, he is prompted to enter the text for the function box. An adequate size for the surrounding box is computed using BIGAM functions, and finally, the text and the surrounding box are generated. All the application-specific functions and macros are contained in a BIGAM workspace FLOW. The code of the basic BIGAM system is not changed, which simplifies code maintenance and update. In this case, the Interactive Graphic Command Language is not used by the operator at all, which is adequate for this simple application.

Summary

With the graphic workstation and computer hardware available at an attractive price today, graphic system development no longer has to concentrate on low-level optimization of application code. The main emphasis can now be put on ease of use, ease of programming, and ease of application system management and maintenance. The Graphic Interactive Application Monitor, GIAM, has been developed to provide an intelligent graphic base system that can easily be used and customized by application programmers as well as by end users.

It has been shown that by proper structuring of a system and with the use of a high-level, pragmatic, and user-extendable language,

Figure 12 Example for application tailoring



a powerful and highly flexible graphic base system, which achieves excellent user productivity with a minimum of application program development, can be implemented.

ACKNOWLEDGMENTS

The initial GIAM prototype was a product of many endeavors, beginning with a customer joint study in 1976 with contributions from F. Gracer and R. N. Wolfe from IBM Research at Yorktown

Heights, NY. A special effort by IBM Fellow W. F. Beausoleil to provide the group with early 3277 GA hardware was highly appreciated through a critical phase of the project.

The authors would like to acknowledge also a special debt to P. Grunau from IBM Germany Graphic Products Marketing and to H. Fleming and U. Weissflog from the IBM Germany Program Product Development Center, who were instrumental in the release of GIAM as an International Field Program (IFP) for IBM Europe.

CITED REFERENCES AND NOTE

1. T. P. Kurlak, *Computer Aided Design and Manufacturing Industry CAD/CAM, Review and Outlook*, Institutional Report, Merrill Lynch, Pierce, Fenner and Smith, Inc., Securities Research Division, New York (September 1979).
2. GIAM is available as an International Field Program only from IBM Europe.
3. W. K. Giloi and J. Encarnacao, "APLG—An APL-based system for interactive computer graphics," *AFIPS National Computer Conference* **43,** 521–528 (1974).
4. L. G. Roberts, "A graphical service system with variable syntax," *Communications of the ACM* **9,** No. 3, 173–176 (March 1966).
5. R. A. Morrison, "Graphic language translation with a language independent processor," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **31,** 723–731 (1967).
6. Graphic Standards Planning Committee, "Computer Graphics," *Quarterly Report on SIGGRAPH-ACM* **11,** No. 3 (Fall 1977).

GENERAL REFERENCES

K. P. Beier and H. Nowacki, "Konzept und Realisierung eines Rechnersystems fuer rechnergestuetzten Entwurf," *HANSA-Schiffahrt-Schiffbau-Hafen* **116,** No. 8, 631–637 (1979).

W. M. Newman and R. F. Sproull, *Principles of Interactive Computer Graphics*, Second Edition, McGraw-Hill Book Co., Inc., New York (1979).

H. Nowacki, "Grundsoftware fuer technische Rechneranwendungen," *Handbuch der Werften*, Verlag Schoetter & Co., Hamburg (1978), pp. 19–67.

F. Gracer, R. N. Wolfe, and W. J. Fitzgerald, *A Graphic Application Development Facility*, Research Report RC 7896, IBM Corporation (October 9, 1979); available through IBM branch offices.

The authors are located at the IBM System Products Division Development Laboratory, 220 Schoenaicher Strasse, 7030 Boeblingen, Germany.

Reprint Order No. G321-5132.