A basic objective in the design of a package of general-purpose graphics subroutines was to make them accessible to FORTRAN *programmers while circumventing some of the limitations of that language for graphics applications.*

This paper discusses how observation of graphics applications led to the establishment of design criteria for a subroutine package to facilitate the generation of interactive displays. It outlines how application programmers would use the package, including provisions for communication between the console and the program. Many of the fundamental concepts that characterize the package are described, including provisions for display modification and animation.

INTERACTIVE GRAPHICS IN DATA PROCESSING

A subroutine package for FORTRAN

by A. D. Rully

Data processing systems equipped with graphic display consoles have provided an opportunity for new and unusual applications involving man-machine interaction. Several of these applications are discussed in other papers of this issue. The potential of such systems has not always been realized, partly because the use of graphics data processing systems often involves rather complex programming.

General-purpose programs whose execution can be requested in higher-level languages can greatly facilitate the use of graphic display devices. Such programs are needed, for example, to create the graphics orders and data necessary to produce the display. Some general-purpose programs of this type have been provided,[1] but these programs require use of assembler language. Graphic display systems are likely to be used for engineering and scientific applications, and FORTRAN is the most widely accepted programming language in these fields. Thus, it ought to be possible for a programmer preparing a specific application program to request general-purpose graphics functions in this programming language. It should be noted that graphics application programs may be prepared either by the console operator, who actually solves problems at the graphic display console, or by application programmers, who prepare the programs for the console operator.

A major problem in using the FORTRAN language for graphics is that FORTRAN does not have all of the facilities needed for interactive graphics. Therefore, a graphics subroutine package (GSP)

was designed,[2] the subroutines of which can be called from FORTRAN programs. This paper deals with that portion of a total graphic system design that was extracted for implementation.

The subroutines facilitate the display of characters and geometric forms on a graphic display screen, specifically the IBM 2250, and they control the communication between a graphics application program in the main storage of a computer and the operator at the display console. The subroutines enable an application programmer using FORTRAN or assembler language to create for the console user a variety of displays, which are constructed from points, lines, and alphanumeric characters.

We first discuss the design criteria established for the graphics subroutine package. We then consider application programs and the provisions for communication between them and the graphic console operator. The remainder of the paper deals with the fundamental concepts underlying the design of GSP.

Design criteria

Many of the design decisions that resulted in the graphics subroutine package are based on observations of the applications for which display devices are being used or their use is planned.

Because of the dynamically changing requirements typical of graphics application programs, we decided to make the facilities of an operating system indirectly available to the application programmer through the use of the subroutines. Moreover, these functions are performed automatically, so that the FORTRAN programmer is not confronted with problems to which he is not accustomed. For example, the subroutine package should automatically decide which routines should be brought into main storage, how they should be loaded, when additional main storage is needed, and when it can be relinquished. To provide these capabilities, GSP is designed to run under the IBM SYSTEM/360 Operating System.[3] The subroutines are accessible to all FORTRAN and assembler-language programmers whose programs are executed under control of any of the optional configurations of the operating system.

The decision to use subroutines, rather than to extend the FORTRAN language, makes the subroutines available to assembler-language as well as FORTRAN programmers. The subroutine approach also offers considerably greater flexibility. If future experience indicates the need for additional functions, the appropriate subroutines can be added much more easily than FORTRAN compilers can be modified. A more basic reason for not extending the FORTRAN language, however, is that operating system facilities, such as dynamic program loading or storage allocation, would not otherwise be available.

Among other objectives for GSP is ease of use for the programmer, the lack of which is believed to be an important factor in deterring use of interactive display devices. We attempt to make use

of graphic display consoles as easy as use of the more familiar magnetic tape devices, card readers, and printers. However, all of the facilities of the graphic display device remain accessible to the application programmer.

The graphics support package is also designed to provide some degree of device independence. Although this package is intended primarily for the IBM 2250 display console, it is designed so that other graphic devices can be incorporated with limited impact on the system. Our observations of graphics applications also led to the conclusion that the package should support multiple consoles for a single graphics application program.

Early in its design, it became apparent that GSP should provide only general-purpose facilities for using graphic display consoles. For example, the subroutines should neither generate geometric figures nor impose constraints on the structuring of models, since the ways in which geometric figures are generated appear to be too dependent on applications. For these purposes, users can develop libraries of routines that are more in accord with their needs.

Finally, it was decided that GSP should be compatible with the general-purpose programs mentioned earlier to simplify problems for users of that support. In fact, the assembler-language programmer can mix use of that support with GSP.

Application programs

GSP is designed so that by calling the subroutines in an appropriate order, the application programmer creates the environment in which the console operator can solve his problem. The functions of the subroutines in the package are summarized in Table 1.

A major design objective of GSP was that it simplify the job of the graphics application programmer. One of the more obvious approaches was to perform functions automatically where feasible. Some subroutines need not be called unless the function they perform is to be done in a nonstandard fashion. Similarly, default conditions prevail when many of the subroutines are called unless the programmer explicitly chooses optional alternatives. Also, the syntax of the FORTRAN language was modified so that calling GSP subroutines requires less coding. (These last two topics are dealt with in greater detail later in the paper.)

using GSP

To prepare a graphics application program, the programmer first calls subroutines that inform the operating system of his intent to use GSP and that identify the device on which his display is to appear.

Next, the programmer defines one or more graphics data sets (files that will later contain all the graphics orders and graphics data,[4] either in main storage or in the display buffer, needed to display an image). The package is designed so that the application programmer can create as many graphics data sets as needed and destroy those no longer needed to conserve main storage space.

Table 1 Graphics subroutines

Subroutine	*Purpose*
Initialize GSP	Establish communication between application program and GSP
Initialize graphics device	Identify 2250 on which displays are to appear
Initialize graphics data set	Create a graphics data set for a particular 2250
Set data mode (optional)	Define type and form of input data for a graphics data set
Set graphic mode (optional)	Define form of output to be produced by image generation subroutines for a graphics data set
Set character mode (optional)	Define character size for a data set and specify whether characters can be replaced from alphanumeric keyboard
Set graphics data set limits	Define boundaries of a graphics data set relative to the screen
Set data set limits (optional)	Define scaling factor for input data associated with a graphics data set
Set scissoring option (optional)	Define action to be taken if image exceeds boundaries of graphics data set or screen
Move beam to position	Control position of beam on screen
Set beam at absolute position	Control position of beam based on absolute input data
Plot line(s) Plot point(s) Plot line segment(s) Plot text	Create orders and data necessary to display lines, points, line segments, and text
Execute	Cause display to be produced from graphic orders and data
Create attention level	Establish level of an attention signal within a hierarchy
Enable attention sources	Designate types of attention signals to be accepted or ignored
Request attention information	Indicate whether an attention signal has occurred and, if so, its source
Terminate use of graphics data set (optional)	Terminate use of particular graphics data set
Terminate use of graphics device (optional)	Terminate use of particular 2250
Terminate use of GSP	Terminate use of all graphics data sets and 2250's, and free all storage used by GSP

The programmer next defines the characteristics of the data, such as how it is to be scaled, the size of characters, and its location relative to the boundaries of the screen. He also specifies the type (real or integer) and form (absolute or incremental) of the coordinate values that he will supply later. Many of these parameters are predefined as default conditions and used automatically unless the programmer specifies alternatives.

The graphics orders and data itself are now created by image generation subroutines. So as not to unduly restrict the applica-

tion programmer, these subroutines enable him to display points, lines, and characters, from which he can form his composite image. Actual display of the image requires calling another subroutine. Subroutines are also provided to modify images, another topic that is expanded later.

program-to-console communication

One of the most important capabilities of GSP is that it enables the application programmer to establish communication between his program and the display console. The basic mechanism that permits interaction between an application program and a 2250 display console is the I/O interruption, which is usually called a *graphics attention signal.*

In designing GSP, we made an initial observation that influenced much of the approach to handling attention signals. The more directly the console operator can exert control over the application program, as opposed to the application program asking the console operator for information, the more successful the graphics application. Thus, GSP provides facilities that enable the application programmer to structure his program so that it can be directed by the console operator. GSP not only allows the programmer to designate the functional significance of attention signal sources but to change their significance at will.

The application programmer using GSP can enable or disable any attention signal source. He may choose to light those keys on the program function keyboard that are enabled. Attention signals are appropriately decoded and stacked in a queue, which is interrogated at appropriate times during execution of the program. Thus, attention handling is much like a standard FORTRAN I/O operation.

When an attention signal is generated, information is made available to the application program. The program function keyboard provides a numeric value for identification purposes. The alphanumeric keyboard supplies the identification of the end key and the cancel key. The light pen can be controlled by the program both to identify the data used to generate the part of the image under the pen and to return the corresponding x-y coordinates in either the programmer-defined or the device coordinate system.

The ability to draw freehand and to track with the light pen is also a feature of attention handling in GSP. This function is device dependent and requires use of the 2250 Model 3. The console operator using the light pen traces a pattern on the screen, such as a design to be used in a fabric or a mechanical drawing. The trace appears on the screen, and the data needed to regenerate the image is automatically created and can be stored for future use.

GSP concepts

A number of underlying concepts permeate the entire graphics subroutine package. Some of these reflect generally accepted program design practices, and others are efforts to provide an easily used but flexible package of subroutines.

null variable

In designing the interface between a FORTRAN program and GSP, adhering to the strict syntax of a FORTRAN calling sequence became a problem. Since the FORTRAN calling sequence makes no allowance for missing arguments or keyword parameters, all arguments are positional and must be present. The list of arguments used to call graphics subroutines is usually long to allow for generality. Frequently, however, the programmer may use only a few arguments. To get around this problem in GSP, the FORTRAN programmer is allowed to define, by name, a particular variable to the system as a *null variable.* Whenever the null variable appears in a calling sequence, it merely takes up a position in the sequence. The value of the variable that it replaces remains the same; if this parameter has never been entered by the programmer, the system default conditions prevail. Through use of variable-length calling sequences and the null variable, the number of parameters required in GSP calling sequences is reduced to manageable proportions.

default conditions

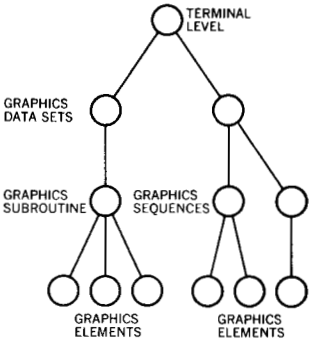
In designing general-purpose graphics programs, one may assume a set of conditions that an application programmer would usually require. These conditions, which are present in GSP, are called default conditions. If the programmer does not take positive action to change them, they remain in effect throughout execution of his program. Whenever default conditions are overridden, the newly entered condition prevails from that point on, rather than the default condition.

GSP director

The facilities for dynamic program management provided by the SYSTEM/360 Operating System are made indirectly accessible to the FORTRAN programmer by the GSP director. The GSP initialization, termination, and director subroutines are the only subroutines in the package that reside in the FORTRAN library. The entry points of all other subroutines are included in the director, but the subroutines themselves reside in the link library of the SYSTEM/360 Operating System.

A table containing the names of all of the GSP reentrant subroutines is brought into main storage during system initialization, along with usage codes. When a particular entry point in the GSP director is referred to, the director determines from the table whether the subroutine is already in main storage, and, if not, whether it should be linked or loaded into main storage. This decision is based on whether projected usage indicates that the subroutine is likely to be required in the future; hence the need for the usage code in the table. These usage codes may be set at the particular installation by reassembling this table or through system operator communication at the time of system initial program loading (IPL).

Figure 1 GSP graphics data structure



data structure

Graphics data can be generated and manipulated by GSP on the following five levels, as shown in Figure 1.

At the terminal (or highest) level, manipulation of graphics data is analogous to the handling of tape reels or disk packs. The functions performed by GSP are analogous to rewinding and unloading a tape reel or removing a disk pack. Instructions may be included

in the graphics application program to either temporarily disable a particular graphics terminal or permanently remove the data that has been assigned to it. The analogy between the graphic display terminal and serial devices ends at the terminal level; at lower levels, the handling of graphics data can only be related to the comparable operations on direct access devices.

The next lower graphics data entity is the *graphics data set.* The graphics data generation routines communicate with a graphics device as if it were a random access storage device. The programmer can define one or more graphics data sets that will exist on a particular graphics device. At this time, the graphics data set is represented by rectangular areas on, or at least partially on, the display screen. These areas may overlap in any fashion required by the application programmer. Since GSP is somewhat device-independent, provided the data sets are rectangular, the application programmer can specify any type of Cartesian coordinate system to define his graphics data set on the display screen. The programmer may then define a totally different set of Cartesian coordinates within a particular graphics data set. Thus, the programmer need never concern himself with the actual coordinate system of the device.

The lowest level of data within the framework of GSP is the *graphics element,* which we define as that unit of graphics data generated as a result of one call to a graphics data-generation subroutine. (The graphics element is described here since the remaining levels can more easily be understood in relation to the graphics element.) The data generated by one call must all be of the same type (i.e., line, point, or character). However, the amount of data generated depends upon the input parameters. For example, a single line, a set of contiguous lines, or a set of disjointed line segments may be generated from the same call.

A *graphics sequence* is defined here as a variable-length, not necessarily contiguous string of graphics elements, which need not be of the same type. Thus, the graphics sequence is data at a level above that of the graphics element but below that of a graphics data set. The facility for manipulating data at the graphics sequence level was provided because it was recognized that most graphic displays are made up of groups of elements that are logically connected through the application program. It is therefore possible to construct an object or a set of objects and to logically connect them through use of a programmer-defined sequence. For example, let us assume that a graphics data set has been defined such that its boundaries coincide with the display screen boundaries. The graphics application programmer wishes to display a string of disconnected squares and enable the console operator to select a particular square. At this point in the graphics program, assume that a decision must be made whether a particular square selected by the console operator is in an odd or an even position in the string. To make possible this decision, the programmer may generate all odd-position squares in a single sequence and all even-position

squares in another sequence. When the console operator selects the square in which he is interested with the light pen, the application program can determine in which sequence the selected square belongs.

The *graphics subroutine* is very similar to a graphics sequence. It is device-dependent and operational only on an IBM 2250 Model 3 graphic display console. A graphics subroutine is analogous to a closed FORTRAN subroutine; i.e., a graphics subroutine is placed in the graphic device buffer and entered each time the particular graphics entity it generates is to be displayed on any part of the screen. If a graphics subroutine were generated to draw a circle of a particular diameter and if the subroutine were stored in the 2250 buffer, the same subroutine could be used to display many circles of the same diameter on various parts of the screen by entering a new starting position for each circle.

correlation value and key variable

One need not look at many graphics programming applications before recognizing the need for a means of naming graphics entities and relating them to application data. The design of GSP provides two naming capabilities. A *correlation value* is a FORTRAN variable or constant, the value of which—on entry to a graphics data generation subroutine—is associated with the graphics element, sequence, or subroutine generated. This value, specified by the graphics application programmer, may be used, for example, as a pointer into a data structure, a value for making a logical decision, or the value of a variable that is to be used in a calculation. The same correlation value may be assigned to more than one element if desired. A *key variable*, the second naming facility, is a FORTRAN variable of integer type. The value of the key variable is set by a graphics data generation subroutine to the identifying key of the graphics element, sequence, or subroutine generated.

Both of these variables are optional parameters, their values being returned to the application program as a result of detecting with the light pen the graphics entities with which they are associated. The correlation value and key variable may also be used as input to the graphics data manipulation subroutines to identify graphics entities that are to be modified. Note that, since the value of the key variable generated by a particular subroutine is unique on the graphics data set level, it may be used for building application data structures.

animated displays

GSP provides the programmer with the ability to "equivalence" groups of graphics data sets. This feature was included to facilitate the production of animated displays, in which multiple frames must be displayed in rapid sequence to achieve the illusion of motion. Since the graphics data sets must use by design the same contiguous block of buffer locations, only one can reside in the buffer at a time. To replace the graphics data set currently in the buffer and just executed, the application programmer simply calls the execution routine, specifying the name of the replacement data set.

Replacement graphics data sets are generated in exactly the same way as standard graphics data sets except that a complete

image of the graphics data in the graphics data set is kept in main storage until a complete replacement frame has been generated. The timing of frame replacement is critical. Replacement timing is facilitated by entering a like amount of data in each frame (even if no-operation type data must be used to completely fill the allocated graphics data set). Timing can also be controlled using either the internal timer or buffer-generated interruptions.

updating displays

Again, by observing a variety of graphics applications, it was noted that minor modifications to displays are often required. Normally, modification requires complete regeneration of the graphics data for the display, which seemed rather inefficient. Therefore, GSP was designed so that relatively small amounts of data can be replaced without complete regeneration. For example, it might be necessary to modify one side of a polygon being displayed on the screen. The console operator, using the light pen and the alphanumeric keyboard, can identify to the application program the side to be modified and how it is to be modified. The application program, using the key variable or correlation value returned because of the light pen detection, replaces with new graphics data the data used to display the side to be modified. The polygon is then redisplayed with the new side.

The limitation on the updating facility is the amount of data that can be inserted. Update data must be less than or equal to the graphics data being replaced.

Summary

The complex programs needed to use graphic display consoles interactively has hampered fuller exploitation of a potentially useful data processing facility. General-purpose programs have been provided in the past, but they required use of a programming language unfamiliar to many users of graphics data processing equipment. The graphics subroutine package described here can be used by both FORTRAN and assembler language programmers, and it enables the programmer to cope with the dynamically changing conditions typical of interactive display systems.

CITED REFERENCES AND FOOTNOTES

1. IBM SYSTEM/360 *Operating System: Basic Graphic Programming Services for* 2250 *Display Unit,* C27-6909, International Business Machines Corporation, Data Processing Division, White Plains, New York.
2. IBM SYSTEM/360 *Operating System: Graphic Programming Services for* FORTRAN IV, C27-6932-1, International Business Machines Corporation, Data Processing Division, White Plains, New York. This package, program number 360S-LM-537, can be obtained through IBM Branch Offices.
3. IBM SYSTEM/360 Operating System options perform a single data processing task at a time or perform a fixed or a variable number of tasks concurrently.
4. See the paper by A. Appel, T. P. Dankowski, R. L. Dougherty in this series.