Graphics and interactive systems—Design considerations of a software system

by DR. ROBERT C. GAMMILL

University of Colorado
Boulder, Colorado

and

DAVID ROBERTSON

National Center for Atmospheric Research*
Boulder, Colorado

INTRODUCTION

The National Center for Atmospheric Research (NCAR) maintains a CDC 6600 and a CDC 7600 computer for doing numerical studies related to the atmosphere. These computers are operated in a batched multi-programming mode by a unique operating system implemented by the system’s development staff (seven programmers). Hard copy output from the computers is handled either by the line printers or by the two Computer Output Microfilm Recorders (COM). A large library of graphic routines is frequently used by the applications programmers to produce outputs ranging from contour maps to movie titles. Around 600,000 frames of graphic output are produced each month, and nearly 80 percent of all jobs generate some graphic output.

The obvious success and importance of microfilm graphics at NCAR led to the rental of a CDC GRID (Graphical Interactive Display) to investigate the potential of interactive graphics in atmospheric research. The primary question was whether interaction would prove as important to the atmospheric scientist as does graphical output. The answer to that question has not yet become clear, with user reactions ranging from enthusiasm to disinterest. The purpose of this paper is to describe a software system aimed at making the GRID as useful as possible to the atmospheric scientists and programmers at NCAR within a context of carefully limited allocation of system and manpower resources.

Three topics will be examined. First, the context and limitations imposed by choice, by the existing facilities and by the orientation toward atmospheric research will be described. Second, a set of goals which grew out of the context will be itemized. Finally, the form of the implementation will be briefly described, related to the goals, and evaluated as to success.

* The National Center for Atmospheric Research is sponsored by the National Science Foundation.

SYSTEM CONTEXT

Perhaps the most important features of a context are the set of constraints under which any new system must exist. At the time the GRID arrived, the NCAR operating system was undergoing major revision and change brought on by the arrival of a CDC 7600. Little help could be provided by the systems staff. Any implementation had to mesh with existing system structures. The NCAR operating system uses quite different I/O and supervisory linkages than CDC systems. Therefore the GRID system implementation had to be complete, not just an extension of a CDC package. At least as important were a set of philosophical constraints imposed by NCAR’s computing environment.

The CDC 6600 to which the GRID was to be connected is used primarily for fast turnaround batched Fortran jobs. Fast turnaround, for all but very long running jobs, has been one of NCAR’s most cherished traditions for which the operating system as well as human procedures have been specifically tuned. It was therefore decided that under no circumstances was the GRID to cause significant degradation of that fast turnaround. This led to the decision to treat GRID jobs the same as any other batched job for system resource allocation. It also implied that host computer-GRID communication should be held to as low a frequency as possible.

The last constraint was that the GRID was on trial to see if it was in fact useful. This made it mandatory to develop software that could be fruitfully applied to the class of problems that currently existed at NCAR, rather than develop a complete tool and hope that given the new sophisticated tool, a new class of problems would be posed to use the new tool.

Atmospheric scientists have depended heavily upon all kinds of graphics since long before the computer existed. They also have made extensive use of computers since their inception. What a scientist does on the next computer run is a function of what the graphics, say contour
maps, show him about the last one. In this sense, NCAR scientists have been doing interactive graphics for many years with a response time on the order of minutes rather than seconds. What we needed were interactive programs with graphic output (display), not interactive graphics. Therefore it was possible to ignore a large number of the subjects currently in fashion among interactive graphics systems developers.

The last context element was the very large library of available graphics routines at NCAR, their extensive documentation and the great familiarity of users with them. Harnessing the power of these without requiring extensive program changes by the user was a must. Since the COM units had the same 1024 X 1024 address space as the GRID, we decided upon the straightforward expedient of direct translation of COM instructions to GRID instructions.

DESIGN GOALS

The context, just described, led to the adoption of a number of carefully chosen goals. The most important considerations involved deciding what not to do, in order that a useful system could be produced under our particular limitations. Some of these goals are quite different than those chosen in other graphics systems. Despite their limited and simple character, the design decisions they caused have provided a surprisingly versatile and powerful system.

Simplicity, compatibility and usability

It was important that the whole effort be kept small, as only four people, each working part time, were available to work on it. Compatibility of the software with existing operating system conventions was necessary because there was not systems programmer manpower available for any extensive modification. The graphics generation portion needed to be compatible with existing graphics generation procedures both to utilize that power and to avoid large manpower expenditures on a parallel effort. Since the GRID had only the status of visitor on trial, it was absolutely essential to make it easy for the user to understand how it worked and how to use it. Lack of manpower precluded voluminous manuals and extensive training programs. Therefore the more the GRID could be made to look like a collection of familiar I/O devices the more the user could capitalize on his previous knowledge.

Device independence

Making the GRID look like a combination of reader, printer, and COM unit was important for other reasons. We were to have only one GRID and experience told us that it was apt to be down for extended periods. It might also be removed permanently at any time. Research activities at NCAR had to continue no matter what the current state of the GRID, so making alternative non-interactive devices available as a surrogate interactive terminal made program running always possible, if not interactive. This problem is not unique to NCAR, and greater emphasis upon device and interaction independence in the user interface would make interactive graphics a much more usable and dependable tool for the average computer user.

Minimum degradation of operating system.

Besides the decision to treat an interactive job as a batch job for resource allocation, already mentioned, this goal implied that communication between the host computer and the GRID be kept to a minimum. Keeping the number of communications to a minimum simply means that each transmission must contain the maximum amount of information. This in turn forces the interactive graphics terminal resident program to be as self-sufficient as possible, and capable of collecting a fairly large fund of information before communication becomes necessary.

DESIGN DECISIONS

The preceding goals coupled with the initial constraints led to three crucial design decisions:

(a) A high degree of modularity was to be used.
(b) All graphics were to be generated by translation of COM graphic commands.
(c) All interaction was to be through the use of card image transmission as line by line text in an extended form of the NAMELIST I/O input language.

Modularity and staged implementation

The system was to be designed as a set of modules embodying some of the principles of software engineering as expounded by Liskov. In particular, levels of abstraction would be identified, and modules designed to implement each level. However, in this case the primary motivation was not correctness, but the ability to stage the implementation. The most essential (bottom most) levels of abstraction would be implemented first, leaving the less essential top levels till later. An advantage of a very structured approach was that part-time personnel could work on different levels of abstraction without necessity for constant interaction. Note that the bottom-up approach appears to be contrary to the top-down method recommended by Liskov and Dijkstra. A schematic diagram of this scheme is shown in Figure 1.

To further ease the development task, it was decided to use a high level language (Fortran), wherever possible, with subprogram calls and named common blocks as the primary mechanism of communication between and within levels of abstraction.
Graphics generation

The advantage of having the entire existing graphical software library available for generating graphical output was self-evident. User familiarity with and good documentation of that library provided a valuable bonus. The need to provide archival graphical output was also solved by translation of the COM instructions since they would be available not only as input to a translation routine but also for direct use.

A decision on how to use the light-pen was deferred until the main body of the package was functional.

Interaction through NAMELIST I/O

Card images are a rather limited form of communication, but the advantage of allowing the GRID to be used as if it were a combined card reader, line printer and COM let the inexperienced user begin writing programs for it without going very deeply into the methods being used. Furthermore, this approach took account of the fact that the user might desire his card image input from one of several possible locations (e.g., card reader, permanent file, etc.) besides the GRID. This not only provided flexibility for interactive runs, when the source of the next input card could be specified interactively, but allowed non-interactive use of the programs when the GRID was out of order.

An extremely important decision was to use NAMELIST I/O as the primary method of communication. A feature of NAMELIST input which made it extremely attractive was what might be termed “mode independence.” By this is meant the fact that data-directed I/O is easy to use in either interactive or batched mode. Each statement is self-describing and self-contained, and the ordering makes little difference (unlike list directed mode).

Since the NAMELIST I/O service routines at NCAR are implemented in Fortran, it was quite easy to modify them to add new features which made NAMELIST input an especially effective tool for interaction. One of these additions was to allow specification of output as well as input values, in a NAMELIST input statement. An example of this is $A = ?$ (a statement meaning, “print the value of $A$”).

These extensions to NAMELIST input resulted from the realization that NAMELIST serves to allow interpretive execution of simple assignment statements which are for all practical purposes being inserted in the code at the point of the NAMELIST read. Examples of extensions which proved particularly useful were:

(a) implied DO loop
(b) print statements
(c) assignment statements with variables and expressions on the right hand side
(d) definition of a special array called CORE beginning at location one (allowing patching and dumps).

IMPLEMENTATION

The discussion of the implementation must necessarily be brief. Further descriptive material can be found in the GRID User Manual.

The GRID resident software

The hardware of the GRID is composed of:

(a) a refreshed CRT tube
(b) a light-pen
(c) a typewriter keyboard with supplementary function keys
(d) a mini-computer for operating the device
(e) an operator’s console for the mini

Even though the user is not concerned with the mini-computer software, we will briefly discuss how it was designed to match our overall context and constraints. Here again, restriction of the capabilities has allowed simplicity. The primary mechanism for transmission of information from the user of the GRID to the host computer is through the line of text (card image) typed in at the keyboard, and transmitted after possible backspacing, line clearing and retyping, by a SEND key. Text may also be generated by positioning the light-pen tracking cross at a desired position and pushing a special function key, causing text of the form “$MX = 132, MY = 427,$” to be generated. This means, that for the majority of informa-
tion transmitted from the GRID to the host, there is no essential difference between the GRID and a card reader or a file of card images.

The only other kind of GRID input allowed is item selection by the use of the light-pen. The result of such a selection is the transmission of the number associated with the graphical object at the time it was generated by the host computer.

Graphical displays and card images are handled by separate parts of the GRID resident software. This means that text and graphics can overlap, distinctly reducing readability. As a practical matter, overlap is normally not a great problem, since the screen is completely cleared preceding each new graphical display transmission.

**Host computer software**

**Abstraction level 1**

The lowest level of abstraction of the host computer software, and the first to be implemented, is an assembly language routine named GRIDIO. This routine allows the host computer system to execute op-codes which control or interrogate the GRID. Arguments of the routine include: the op-code, a word for return of status bits, a buffer area and a count of words of information in the buffer area. The op-codes, status word and buffer areas are kept in named common blocks of the Fortran programs which call GRIDIO. GRIDIO has no knowledge of or access to the routines which call it. It simply serves as the sole access mechanism for the Fortran routines at the next level of abstraction, which must communicate with the GRID.

**Abstraction level 2**

This level includes a single Fortran routine named WAIT. The host computer routines are able to request a number of different kinds of services from the GRID, such as clearing its screen or putting the next line of text on the screen. When the GRID is in the process of providing such service, it must not be interrupted. WAIT provides the synchronization of such service requests.

**Abstraction level 3**

This level is concerned with card image communication both ways, and graphical communication (complete pictures) to the GRID.

Two Fortran routines, named RDCARD and WRCARD, read and write card images to and from the GRID (and other devices as well). Each of these routines has a single argument, the core location of the input or output card image. Besides calling GRIDIO to execute the appropriate op-codes for communication with the GRID, these subprograms also take responsibility for conversion of the character codes.

The RDCARD and WRCARD routines each use a specified location in a named common block which tells the subroutine the source or destination of the card image. This location contains a logical device name. Changing that logical device name during execution allows the sources and destinations of card images to vary. That change can be effected from within the stream of card images itself, by using the NAMELIST input mechanism to assign a new name to the specified location. An especially important point is that any new interactive device which becomes available will only require software implementation for text from this level down.

Graphical entity transmission takes place at this level through a routine called FRAMESG. The products of COM instruction translation are kept in a work space classified by an entity number. A user indicates which entities he wishes to be displayed by this entity number. FRAMESG moves and combines these various entity display instructions into a single display file and calls the lower level routine to accomplish the transmission. Hard copy can be produced on the COM unit at the same time.

It is important to have the provision for many separate entities which may be combined, revised or recreated. For instance, a common occurrence is a weather map displayed on the background of a world map. By creating the world map once as an entity, any number of weather maps may be displayed upon it without the computer time involved in creating the instructions for the background map each time it is wanted. In the GRID, as in most refresh type display units, flicker and display file space are always a problem. It often happens that a complicated weather map uses so many instructions that the background map won't fit along with it. At this point the user may simply request that the background be omitted for this particular display.

**Abstraction level 4**

A user may communicate directly with the third level routines by creating and breaking down card images using ENCODE and DECODE statements. These are formatted core-to-core write and read respectively, and are provided on most Fortran compilers used on CDC machines. It would have been more desirable, in some ways, to hook the card image reads and writes directly into Fortran list directed I/O, but this would have required substantial changes to the Fortran I/O service package, which was undesirable due to the context of the project. The result is that a user desiring to put a simple message, with some formatted data, on the screen of the GRID must execute an ENCODE and then call WRCARD to transmit the card image.

This level contains the extended NAMELIST I/O, COM to GRID translation with associated work space management, and provision for error recovery.

NAMELIST input is accomplished through a routine called READLX. It is totally isolated from the GRID, and has no dependence upon specific GRID characteris-
tics. READLX acquires its card images through RDCARD, and prints through WRCARD, the third level routines.

The basic tool of READLX is associating a memory location with a name. This is accomplished by the user calling a routine LEXCON (NAME, IWHERE, ITYPE). NAME is the hollerith character string defining what he is going to call this particular variable, IWHERE is the variable name itself, (an address) which READLX will use in locating it, and ITYPE is the type of the variable (integer, real, double, complex, or character). If ITYPE is zero or the argument not provided, LEXCON assigns a type of integer or real depending on the first character of NAME according to Fortran conventions. LEXCON must be called for each variable name one wishes to reference during a run. Each variable is assumed to be the first address of a singly dimensioned array, so that any variable may have a subscript during READLX input.

Extensions have been added to the input form to allow printing and multiple element definition, as well as full arithmetic assignment statement capability. In each case the extension has been borrowed from a familiar Fortran usage in order to seem as natural as possible.

Several variables are automatically defined by the initial call to LEXCON. These are variables the user may use to help define the behavior of the interactive package, such as ECHO, a variable that controls auxiliary printer and COM output of what goes on during an interactive run.

In the early stages of system development the light-pen was completely ignored. The light-pen is rather difficult to handle in a device independent fashion, as it does not act in a way that is similar to other I/O devices. Upon completion of the early development stages, the 80 character card image and NAMELIST I/O had clearly established themselves as simple yet powerful abstractions. As a result, it was decided to experiment with techniques for hiding the light-pen under the card-image, as this would allow the card reader to substitute for the light-pen. It proved quite simple, using NAMELIST input language, to represent the present position of the light-pen tracking cross by a pair of statements "MX=1014, MY=523."

This statement pair is generated by the GRID when a certain function key is struck. The NAMELIST I/O facility allowed long sequences of such data pairs to be transmitted and saved in a special array where they could be retrieved by the user program after completion of the NAMELIST read. A particular advantage of this approach was that pen positions selected by the function key but later recognized as undesirable, could be edited out of the card image before transmission to the 6600. Furthermore, this method tended to collect a sequence of pen positions before action was requested by the user from the host computer. It must be recognized that this method of handling positions of the light-pen tracking cross would be unsuitable in any application where drawing of general curves or objects was done on a regular basis, as a function key strike must be made for each X-Y position to be transmitted. However, in applications where the user wishes to pick accurately selected positions on the screen (e.g., when making corrections to graphically presented data) the technique is very successful.

In the final stages of system development a capability was added for light-pen selection of a menu item or graphical object. The result of such a selection is that a number assigned to that object is immediately returned to the host computer. Those numbers are assigned to the graphical objects at the time they are generated by the 6600. The number returned to the 6600 is not coded into the form of a card image. This is the one exception to the axiom of this system, that all information transmitted from the GRID to a user program is in the form of a card image. The only justification is that it seemed unesthetic. Note that device independence has not been lost, because the routine which returns the number of the object selected to the user program can read a number from a formatted card image instead.

Error recovery is a feature not mentioned thus far but important nonetheless. While GRID jobs are treated the same as batch jobs for resource allocation, it is necessary to avoid run termination in the event of a normally fatal error. Errors frequently occur in typing NAMELIST statements such as A=B/C where C is inadvertently zero. IERPROC is a machine language routine that causes control to be returned to the user when a fatal error occurs. The facility for this was already in the operating system for use with system simulation.

When such an error occurs, a message is sent to the screen containing the error message and a request to the user to specify what to do next. His options are to go back to the NAMELIST routine, return to a specific part of his program, terminate the run, or get more CPU time. CPU time is doled out in one minute units, with a maximum of 10 minutes. To avoid infinite loops, the program is automatically terminated when IERPROC is entered more than 30 times.

EVALUATION

The software system for the GRID has achieved considerable success within the limited goals set for it. The design decisions, to translate display commands, use of the card image as the message unit for interaction, and the use of data directed input as the primary method of interaction, have all proved fortunate. The system is relatively easy to understand and modify. This is attested by the number of different programmers who at one time or another, participated in the development of a module. Users of the GRID found it easy to use, because only a few new calls on system subroutines were needed, beyond those graphics calls they were already making to generate microfilm. The data directed input facility provided users with great flexibility and power for examining the contents of different locations, either by name or by address, and modifying those contents. On many occasions the GRID proved to be especially useful as a debugging tool.
On the negative side, the reader should notice that the only facility available in the final design which requires a refreshed graphic display with light-pen is the menuing and object selection. This capability has not been terribly important in any success the GRID has enjoyed. Furthermore, the refreshed display is often incapable of displaying all the graphic information that a scientist desires, both because of excessive flicker and insufficient storage space. This lack, which is inherent in refreshed displays, has been the most detrimental to the usefulness of the GRID for the atmospheric scientist. Currently the use of storage scope technology (with cursor) is being investigated. Costs of such devices are much lower, and there is no flicker or storage space problem. Somewhat higher demands may be placed on the host computer if it must deal with characters instead of card images. The interactive portion of the software system, and the user programs which depend upon it, would continue to be valid down to the level of RDCARD and WRCARD on the text side, and down to the translator on the graphics side, no matter what device may replace the GRID.

SUMMARY

A software system for a graphical interactive display terminal, and the motivation for the decisions involved, has been described. Several abstractions proved useful in achieving the goals set. The most important of those were, the use of card images for interactive communication, the use of NAMELIST input as the primary user tool, and the direct translation of display commands. Most important system features were a high degree of modularity and device independence.

REFERENCES