LEVEL (12) B.S.

AD A089596

Richard Bisbey II
Dennis Hollingworth

# A Distributable, Display-Device-Independent Vector Graphics System for Command and Control

DTIC
ELECTE
SEP 2 6 1980
S
C

## INFORMATION SCIENCES INSTITUTE

UNIVERSITY OF SOUTHERN CALIFORNIA

ISI

80 9 26 027

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|

| 1. REPORT NUMBER | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
|---|---|---|
| ISI/RR-80-87 | AD- A089 | 596 |

| 4. TITLE (and Subtitle) | 5. TYPE OF REPORT & PERIOD COVERED |
|---|---|
| A Distributable, Display-Device-Independent Vector Graphics System for Command and Control | Research Report |
| | 6. PERFORMING ORG. REPORT NUMBER |

| 7. AUTHOR(s) | 8. CONTRACT OR GRANT NUMBER(s) |
|---|---|
| Richard Bisbey, II and Dennis Hollingworth | DAHC 15-72-C-0308, ARPA Order-2223 |

| 9. PERFORMING ORGANIZATION NAME AND ADDRESS | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
|---|---|
| USC/Information Sciences Institute 4676 Admiralty Way Marina del Rey, CA 90291 | 42 |

| 11. CONTROLLING OFFICE NAME AND ADDRESS | 12. REPORT DATE |
|---|---|
| Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, VA 22209 | July 80 |
| | 13. NUMBER OF PAGES |
| | 40 |

| 14. MONITORING AGENCY NAME & ADDRESS(If different from Controlling Office) | 15. SECURITY CLASS. (of this report) |
|---|---|
| | Unclassified |
| | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

This document is approved for public release and sale; distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

--------

18. SUPPLEMENTARY NOTES

--------

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

application program, command and control graphics, communications protocol, computer graphics, high-level graphics language, on-line map display, system architecture

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

(Over)

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE
1 JAN 73
S/N 0102-014-6601

## 20. ABSTRACT

This report documents a distributable, device-independent vector graphics system developed by ISI for the Defense Advanced Research Projects Agency. It describes the system architecture, communications elements, and a phased implementation strategy. The system supports graphics-based command and control applications in distributed computational environments such as the ARPANET. The system has been in use at ISI and at the Naval Ocean Systems Center (NOSC) in the Advanced Command and Control Architectural Testbed (ACCAT) since January 1977. The principal aim of the development effort is the device-independence of the vector graphics. "Device-independence" means that graphic application programs can be written without regard to the particular display-device on which the output will ultimately be displayed. This system achieves display-device independence by providing the application program with a set of generic, two-dimensional vector graphic primitives by which pictures can be described and interacted with at the application-level. The particular graphics model used structures pictures as sets of subpictures that are absolute-transformed-segments, as defined by Newman and Sproull.

Richard Bisbey II

Dennis Hollingworth

# A Distributable, Display-Device-Independent Vector Graphics System for Command and Control

# CONTENTS

# OVERVIEW

This report documents a distributable, device-independent vector graphics system developed by ISI for the Defense Advanced Research Projects Agency. It describes the system architecture, communications elements, and a phased implementation strategy. The system supports graphics-based command and control applications in distributed computational environments such as the ARPANET. The system has been in use at ISI and at the Naval Ocean Systems Center (NOSC) in the Advanced Command and Control Architectural Testbed (ACCAT) since January 1977.

The principal aim of the development effort is the device-independence of the vector graphics. "Device-independence" means that graphic application programs can be written without regard to the particular display-device on which the output will ultimately be displayed. This system achieves display-device independence by providing the application program with a set of generic, two-dimensional vector graphic primitives (Graphics Language [2]) by which pictures can be described and interacted with at the application-level.

The particular graphics model used structures pictures as sets of subpictures that are absolute-transformed-segments, as defined by Newman and Sproull [3]. At the most general level, the application program deals with graphics in terms of named picture elements, called segments. Segments can be created, destroyed, merged, made visible or invisible, made touch-sensitive, and highlighted. Segments themselves are specified in terms of generic graphic primitives including vectors, arcs, dots, text, and filled sectors and polygons, which may vary according to color, intensity, and font.

Graphics Language (GL) defines the set of application-independent functions for performing the above operations. Since several programming languages are expected to be used in the Command and Control environment (mainly Ada and FORTRAN, but also LISP, BLISS, MACRO and other languages), GL is defined (and implemented) in such a way that it can be easily used by application programs written in any of them.

The binding of the application program to a particular display device is deferred until program execution time, when the generic graphics primitives are mapped by the system into specific operations and display modes appropriate to the device selected. The quality of the resulting picture is limited only by the capability of the display device.

The second significant aim is that the graphics system be distributable across multiple host computers interconnected by a communications network (such as the ARPANET). This allows the graphics display device to be located away from the graphics application program and the computational load introduced by the graphics system to be distributed across multiple processors, balanced to the computational resources and communications bandwidth available. The graphics system achieves distributability by virtue of its modular design and implementation, i.e., the system consists of a series of isolable functions that communicate via a common communications mechanism.

The design separates the issue of system functionality from communications; the functionality of the system does not depend on the physical processor on which a given function resides during a graphics session. *The physical location of graphic functions (as well as the display device) only affects performance issues; they have no effect whatsoever on the application program.* Moreover, any intraprocess/interprocessor communications mechanism might be used for communications between functions of the graphics system (including telephone, radio, or digital network/internetwork links). As suggested by R. F. Sproull in "Network Graphics Isn't Networking" [4], a graphics system used on a network does not have to address networking issues, but only device independence.

Modular organization of the graphics system has several benefits in addition to distributability. It allows the system to be easily extended to support other application languages and display device types. The former is accomplished by replacing the language interface component with one that interfaces to the new language, the latter by replacing the component that generates device orders. *Such replacement has no effect whatsoever on the application programs or the fashion in which application programs use the graphics system and graphics language.*

Modularity also allows new graphic functions to be easily added to the capabilities of the basic system. The architecture supports a single-terminal-to-single-application graphics environment. Additional graphic functions may be added to support more complicated configurations, including multiple programs concurrently connected to a single terminal and a single application program simultaneously generating graphics output for and accepting input from multiple (possibly dissimilar) display devices.

ISI has implemented the graphics system on the TENEX and TOPS-20 operating systems. Application languages supported by this implementation include Fortran-10, Macro-10, Bliss-10, and Interlisp-10; display devices include the Tektronix 4010, 4012, 4014 series, Tektronix 4027, Advanced Electronic Design 512, Genisco GCT-3000, and Hewlett-Packard 2648A and 9872A.

This report describes some of the perceived goals and requirements that motivated the graphics system architecture, then describes the graphics system architecture itself: its functional components, their relationship to one another, and the intrafunction communication mechanism. Extensions to the graphics system that support saving and reincorporating previously generated graphics output, as well as multidevice and multiapplication configurations, are presented. This is followed by a short description of the communications protocol between the application program and the graphics system (Graphics Language), and the external communications protocol (Graphics Protocol), used for communicating over external communications links and for storing graphics in external files. Finally, the document discusses ISI's current and planned implementation of the graphics system, presents possible configurations of the graphics system, and suggests various usage scenarios that can be supported.

# I. GRAPHICS REQUIREMENTS FOR COMMAND AND CONTROL

Effective command and control depends upon accurate and timely exchange of information between command levels and effective presentation of relevant data. Communication both up and down the command chain is required, as well as communication with databases, either static or dynamically updated. Advanced information processing technology can and should be exploited to manage and augment information exchange and enhance the quality of C2 communications, thereby making it easier for military personnel to interpret and respond to events.

Effective communication is crucial for real-time operation in crises, as the 55 hours of voice conferencing during the Mayaguez incident testify. Simultaneous graphic communication, typically two-dimensional information like maps and charts, can greatly enhance both the quality and the efficiency of communications, and provide depth to the information exchange. Furthermore, graphics is probably the most effective and universal way of communicating spatial information.

Computer graphics is not new to the military. Immediate or anticipated military requirements have quite often been the basis for technology development and deployment; this has been true in various areas of computer graphics. The military has employed computer graphics in a variety of special-purpose applications. However, the utility of graphics as a communications vehicle suggests that graphics should be made available to the military in a more generalized way, much as voice communication is made available via the AUTOVON system. This has become both attractive and economically feasible with the advent of both low-cost graphics terminals and digital computer communications networks such as the ARPANET and its planned military counterpart, AUTODIN-II, now under construction.

Crises like the Mayaguez incident are generally unpredicted and may require quick response to a rapidly changing situation. Because the nature and amount of the resources available to meet the crisis may vary, the C2 graphics system must be adaptable to the resources at hand, including processing, communications bandwidth, and display equipment. Furthermore, it should be extensible so that it can evolve to accommodate (for example) new display devices or new communications technologies. The graphics system must evolve in a way that is not traumatic to its users.

These general observations suggest a more specific set of requirements for the graphics system designed to accommodate a command and control environment.

1. *The development and use of graphics application programs should be independent of the display device type, with the graphics system adapting to and supporting terminals of varying capability.* The developer of an application should be able to produce the application unmindful of the specific display device(s) that will ultimately be used. Furthermore, the developer should be able to test and debug the application program on whatever display device is at hand or is otherwise accessible and reasonably convenient to the test environment. It is quite likely that the equipment available for application

development and testing will differ from that planned for use with the application. This situation is often solved by the developer's making repeated trips between the development site (where tools for program development and testing are readily available) and the application site (where the application hardware exists). Such need not be the case. Finally, the application program should be able to exploit the features available with the particular device to which it is connected without being limited to the greatest common subset of features of the terminals used; the graphics system should perform the requisite mapping of the requested capability to the features available.

2. *The graphics system should be adaptable to the processing and communications resources available when the application program display device connection is established.* The unpredictability of the communications bandwidth available and the location of and accessibility to computational resources during a crisis situation, together with the need to optimize use of available resources to meet the situation, suggest that the graphics system be tailorable to a wide variety of processor/communication combinations. The choice of a particular configuration should be deferred until program execution. For example, suppose a display device is connected to a small local processor that is in turn connected via a very low bandwidth channel to a larger remote processor containing a graphics application. The graphics system should be tailorable so as to minimize the communications bandwidth required between the application and the display device. This might be done by distributing the graphics system across the two processors, i.e., performing some graphics functions on the small processor local to the display device, and by optimizing data transmission formats used. Alternatively, if the communications topology or bandwidth was different or processors with different capabilities were involved, the graphics system should be tailorable to optimally use those resources.

3. *The graphics application should be oblivious to the placement of graphics system components and the requisite communications established to support their placement.* Whether the graphics system is distributed across several computers to take advantage of processing or communications resources, or whether the entire system resides within a single processor, should be invisible to the application program. Any special machinations necessary to establish and support a particular configuration should be handled by and within the graphics system itself with any side effects constrained to the graphics system, with the possible exception of system performance. Thus, the fact that the display terminal is on a ship and the application is running at a land-based WWMCCS site should not be any more apparent to the application program than if the terminal is directly connected to the application computer.

4. *The graphics system should support the incorporation of possibly independently generated graphics pictures into an application program as well as the creation of*

*picture descriptions for use outside the immediate application environment.* In conventional data processing environments, files are a communications mechanism that allow results calculated by one program to be shared with other programs. Thus, files permit a large computational task to be divided into separate subtasks that individually prepare their respective results without regard to the existence and definition of the other subtasks. In addition, files permit temporal discontinuities to exist between subtasks. Results may be calculated in advance of their use, and need not be calculated each and every time they are needed. An equivalent mechanism should be provided for sharing graphic information. Since the display device available to the application that creates a "graphic file" will likely be different from those available to programs using the resulting pictures, the files must be created and stored in a display-device-independent format.

5. *The graphics functions available to an application should be sufficiently general to support a variety of application domains.* This generality should include system configurability, communications adaptability, and device-independent multiterminal usage. The user interface itself should provide a sufficiently rich set of graphics primitives such that highly tailored, application-specific graphics environments can be constructed on top of the interface without requiring alteration of the functionality of the graphics system itself.

The remaining sections describe a graphics system designed and implemented to meet the above requirements. Section 2 describes the overall system architecture: the functional units, their placement, and the intrafunction communications mechanism. Section 3 presents a graphics file extension to the architecture that permits the incorporation of independently generated graphic results in an application program. Section 4 describes extensions to the architecture that allow both multiterminal and multiapplication usages. Section 5 describes the communications forms used between various functional units within the system. Section 6 describes implementations of the system architecture, both planned and completed. Section 7 describes example configurations and provides examples of possible scenarios of use to illustrate the viability of the architecture.

## 2. THE GRAPHICS SYSTEM ARCHITECTURE

This section describes the overall graphics system architecture: the functional components, their placement, and the mechanism by which they communicate. This section is not an implementation description. It does not specify algorithms that might or should be employed by specific functional components; it does not specify how storage requirements should be realized in actual physical storage devices. Such specifics are implementation issues, and their resolution is left to the individual implementer. ISI's particular implementation strategy is discussed in Section 6.

## ARCHITECTURE

The architecture of the C2 graphics system is conceptually simple. It is a series of major graphic functions serially connected on a single bidirectional data channel. The architecture is shown in Figure 2.1. The six functions are:

1. The Application Interface performs parameter checking and data channel interfacing. It contains as a replaceable subcomponent a Language Interface, which interfaces the application programming language to the graphics system.

2. The Model performs segment validation.

3. The Clipper removes nonviewable graphics.

4. The Normalizer converts number modes and ranges.

5. The Storage Manager queues display output and graphic input.

6. The Device Order Generator interfaces the graphics system to the display device.

These functions transform high-level, device-independent application program requests for graphic services into display device order codes for a particular device. They perform the reverse transformation for graphic input originating at a display device. The following describes the architecture in more detail: the intrafunction communications mechanism, then the individual graphics functions and their placement relative to one another.

### Intrafunction Communications/The Data Channel

The data channel is the logical mechanism by which graphics functions communicate. It spans the system, from the application interface to the display device. The data channel isolates functions from each other; functions can communicate with each other only by using the data channel. Architecturally, the data channel provides a uniform
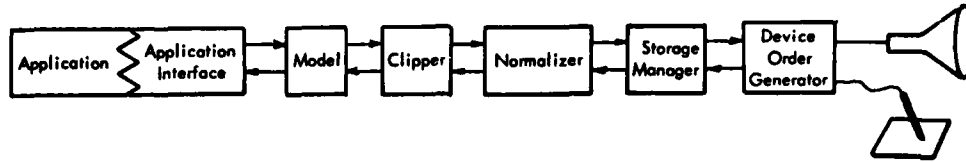
Figure 2.1   Basic system architecture

communication mechanism to which graphics functions Interface.   Because of this inherent isolation of functions and uniformity of communication between functions, the architecture permits new functions to be inserted between any pair of existing functions, and the system readily extended.   The data channel also allows flexible distribution of the functions over various hardware configurations without disturbing the application program or the overall functionality of the system.

Data passes through (is read by and possibly acted on by) each function on the data channel until it is removed by its ultimate recipient.   Functions perform four basic operations on data on the data channel.   They can add a data element to the data channel, remove a data element from the data channel, modify a data element, or pass a data element unchanged to the next sequential function on the data channel.

## The Graphic Functions

### Application Interface

The Application Interface is responsible for interfacing the graphics system and the data channel to the application program and the runtime environment of the particular programming language in which the application is written.   It contains a replaceable, functionally separate language interface that performs all programming specific tasks involved with interfacing to the application program.   The Application Interface represents one end of the data channel.

When invoked (via subroutine call, messages, interprocess communication, etc.) by an application program for graphic services, the Application Interface performs three tasks. First, the language interface retrieves the graphics command and the associated arguments from the programming language environment.   Next, the mainline application interface code validates the command and arguments:   it checks that the command is legitimate and that it is supported by the system; it also checks that the arguments are within the appropriate argument domains of the specified command (although it does not check that a given data value is otherwise valid).   Finally, it translates the command and

arguments into internal data channel values, and forwards the values to the next function on the data channel.

The Application Interface also returns data from the graphics system to the application and signals the application program that data values are available.

## Model

The Model is responsible for validating the legitimacy of graphic operations on segments. It ensures that user-specified segments exist and are in the proper state before permitting the requested operations to take place. Additionally, it eliminates segment-idiosyncratic aspects of the user interface, such as deleting an old segment when a new segment is created with the same name, from the communications protocol.

## Clipper

The Clipper function is responsible for eliminating those portions of vectors, arcs, filled areas, and text that are outside an application-specified window. The Clipper removes, modifies, or passes unchanged these graphics requests depending on whether the graphics are outside, partially inside, or totally inside the display window. The Clipper relieves the application of the responsibility for determining that portion of the application's graphics that will be visible on the display device. In doing so, it shields the application from display device idiosyncrasies regarding treatment of information outside the device coordinate range (e.g., address wrap-around).

## Normalizer

The Normalizer function is responsible for transforming application data values into an internal data mode and normalized value range. Normalizing graphic data values insulates the application from the device and simplifies writing of the application program by providing the application with the ability to specify values in number ranges appropriate to the application. It also simplifies subsequent functions by standardizing modes and value ranges and reduces subsequent communication and storage requirements by removing excessive precision. Arbitrary number ranges established by the application for specifying graphic coordinates are transformed and mapped by the Normalizer into a standard internal mode and range. This permits an application to be written with coordinate values that are totally independent of the actual physical coordinates of the particular display device being used. The Normalizer performs a similar operation for other data values such as those for specifying color. The Normalizer performs the reverse transformation for data originating at the display device destined for the application, i.e., it transforms internal device-independent values into application coordinate and value ranges.

*Storage Manager*

The Storage Manager is responsible for all processing involving the storage and management of both graphical input and output data. The Storage Manager allows applications to specify pictures as a series of named sub-picture elements (called segments) and to change sub-picture elements without having to respecify the entire picture. It is also responsible for supporting the logical input functions in the system including the segment-touching facility. The Storage Manager contains storage (a segment pool) into which information associated with each active segment is retained. It maintains the status of named segments and performs any associated action, such as destroying the segment and freeing the associated storage (redrawing the picture if necessary), adding a new segment to the segment pool, or changing the state of an existing segment.

*Device Order Generator*

The Device Order Generator is responsible for interfacing the data channel to the physical display device. It represents the other end of the data channel. For display device output, it translates data channel command and data representations into device-specific order codes and associated data values. For device input, it performs the reverse translation. In the event that a generic graphics function is being simulated by a physical device capability, the device order generator performs all associated physical device mappings.

The Device Order Generator maintains all device-specific information in the system, making such information available to other graphic functions via data channel requests. It also supports special-purpose facilities that are unique to the display device and that are not otherwise supported by the graphic system.

Function Placement on the Data Channel

The sequential ordering of functions on the data channel is in some cases dictated by the requisite order in which operations must be performed. In others, it is motivated by performance issues associated with the distributability of the graphics system.

Two functions whose placement is dictated by their functionality are the Application Interface and the Device Order Generator. The Application Interface is directly associated with the application program, while the Order Generator is associated with the display device; these two functions represent the two ends of the data channel. Their placement on the data channel is shown in Figure 2.2.

Location of the Model on the data channel is motivated partially by efficiency issues and partially by design objective considerations. Since the model eliminates erroneous segment operations, it is desirable for it to detect and discard these calls as early as possible in the execution sequence. Additionally, placement of the model immediately after the application interface means that segment-idiosyncratic aspects of the user

language are handled early in the execution sequence, thereby shielding the bulk of the graphics system from many of the more idiosyncratic aspects of a particular form of the user language. The next functions for consideration are the Clipper and the Normalizer. The Clipper functionally precedes the Normalizer, since the Clipper bounds coordinate values to a fixed range (the application window). Adding the Clipper and Normalizer to the Model yields Figure 2.3.
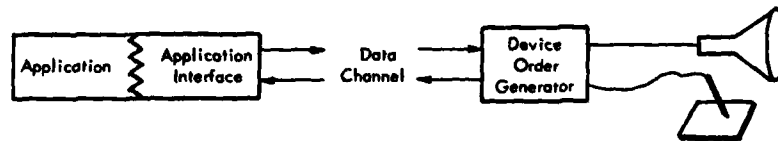


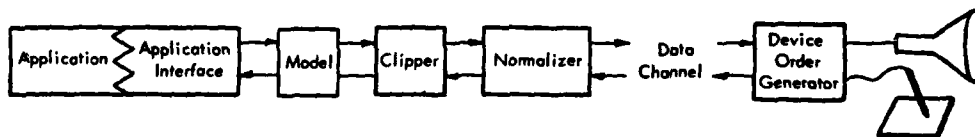Figure 2.2   Application interface and device order generator placement



Figure 2.3 Clipper and normalizer placement

The final function for consideration is the Storage Manager. This function could be placed in one of three positions: between the Model and Clipper, between the Clipper and the Normalizer, or between the Normalizer and Order Generator. Placing it between the Normalizer and Order Generator offers the following advantages over the other possible placements:

1. It minimizes the space requirements for the segment pool, since only graphics that might actually appear on the display need be stored; graphics outside the application-specified window would have previously been discarded.

2. It minimizes data channel usage required for redrawing the picture, since only one link of the data channel must be used as opposed to two or more

in the other cases.  Furthermore, it minimizes data channel usage required
for   input   processing,   since   it   eliminates   the   transmission   of
non-requested/non-sampled input data, particularly important for
high-bandwidth, high-data-rate input devices.

3. It minimizes function application, since lines and text do not have to be
   clipped and/or normalized each time the picture is redrawn. Instead, lines
   and text are clipped and/or normalized only once, when first specified by
   the application.

Placement of the Storage Manager on the data channel results in Figure 2.1, the basic
system architecture.

## IMPLICATIONS OF THE ARCHITECTURE

The architecture defined above has several significant implications with regard to
functional extendibility, separability, and external communications placement.

1. The architecture allows new functions to be easily added to extend the
   system functionality (see Section 3 and 4).  This is the result of functional
   isolation and modularity provided by the data channel.  Functions interface
   to a uniform, well-defined communications mechanism, i.e., the data channel.
   Moreover, since the data channel separates the issues of communication
   and functionality, new functions can be developed for a given data channel
   implementation without regard to the eventual interface point.

2. Because of 1, *external communications links (e.g., a network link, internetting)
   can be added anywhere in the system.* Such a link can be regarded as merely a
   new functional element, each side of which is connected to a data channel.
   As an example, consider an ARPANET Communications Function inserted
   between the *Normalizer and Storage Manager,* Figure 2.4.  This function
   would connect each half of the data channel using the ARPANET, making the
   two halves appear as one logical data channel.  Such a link can be placed
   anywhere along the data channel.  (Note that the data channel is assumed
   by the graphics system to be a 100 percent reliable communication
   mechanism: data is never lost and is always correctly communicated from
   one functional component to the next. Any error recovery or retransmission
   required as a consequence of some failure of the specific external
   communication mechanism(s) must be handled by that communication
   mechanism and be invisible to the graphics system itself.)

3. The data bandwidth will generally be lowest after the Clipper and
   Normalizer (because superfluous lines, text, and precision have been
   removed) and before the Storage Manager (responsible for redrawing
   pictures).   This point represents the optimum point (in terms of
   communication  bandwidth  required)  for  introducing  an  external
   communications function (see #2 above).

4. All of the above functions can be implemented in a device-independent manner with the exception of the Device Order Generator. Whether or not they are depends on the objectives of the implementation effort. In particular, the Storage Manager (and its associated storage pools) might be implemented in a device-dependent fashion tailoring stored values to the display device, minimizing the amount of storage required for individual data types, and reducing the amount of work involved in redrawing the display.

5. Display devices may include one or more of the graphic functions within the physical device itself, e.g., clipping, segment storage. For these devices the corresponding graphic functional element can be disabled.

6. The architecture supports a variety of upward-compatible implementation levels exploiting more and more functionality. Three such levels are discussed in Section 6, which describes ISI's implementation of the graphics system architecture.

7. New applications languages can be supported by replacing only the Language Interface component of the Application Interface. *All other functional elements remain unchanged.* Furthermore, the device-independent graphics functions provided by the system remain the same, regardless of the application language.

8. New display devices can be supported by replacing only the Device Order Generator. *All other functional elements remain unchanged. Existing application programs need not be recoded to use new display devices unless they contain device-dependent escape sequences that bypass the standard graphics system.*
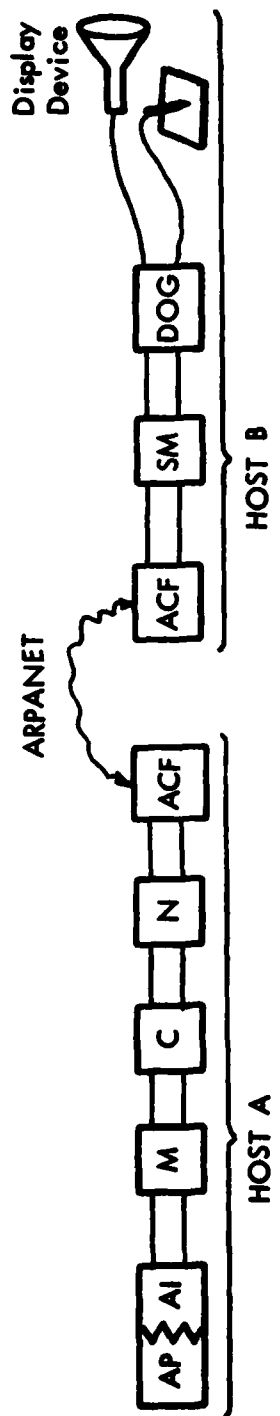
KEY:
ACF – ARPANET Communication Function
AI – Application Interface
AP – Application Program
C – Clipper
DOG – Device Order Generator
M – Model
N – Normalizer
SM – Storage Manager

Figure 2.4 Graphics system with external communications function

## 3. FILE I/O EXTENSION TO THE SYSTEM ARCHITECTURE

The overview identified a requirement for the support of a graphics file mechanism, permitting independently generated graphic pictures to be incorporated into an application program as well as such results to be created for use outside the immediate application environment. This requirement can be supported by introducing into the graphics system two additional components: a File Input Manager and a File Output Manager.

### FILE INPUT MANAGER

The File Input Manager is responsible for incorporating externally stored graphics into the system. It processes application requests for file input, establishes a connection to the desired file, and injects the requested information into the data channel stream. In performing this function, the File Input Manager translates external file commands and arguments into modes and ranges that are compatible with the data channel values. The external input is introduced into the standard graphics environment and can be subsequently manipulated using standard graphic functions (i.e., the normal graphics constructs used to control visibility/invisibility, highlighting, sensitivity to touching, destruction, etc.).

### FILE OUTPUT MANAGER

The File Output Manager is responsible for generating external files of graphics data. It processes application requests for file output, establishes a connection to the desired file, and copies information from the data channel stream into the file. The File Output Manager also translates internal commands and arguments into modes and ranges compatible with external data storage.

### FILE FUNCTION PLACEMENT

The file I/O functional components can be located at several places along the data channel depending upon whether the system architect wants to minimize external storage requirements or retain the file data in unclipped form. The File Input Manager must precede the File Output Manager to allow file input to be automatically collected in the file output stream. The File Output Manager should logically precede the Storage Manager and segment pools since files represent collections of picture definitions, not a series of picture redraws. Placement of the pair after the Normalizer avoids redundant function application and communication and minimizes file storage requirements. The addition of the file managers yields Figure 3.1, the architecture with file I/O extension.
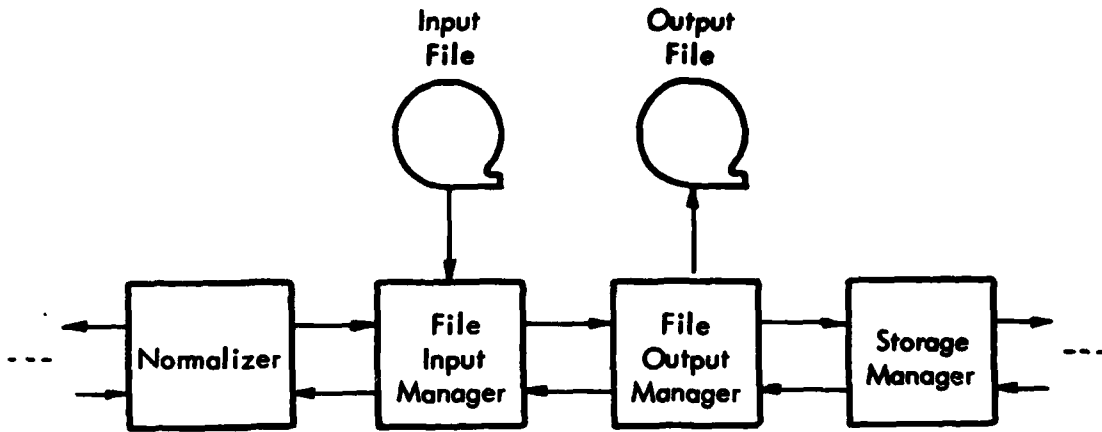
Figure 3.1   File input/output extension

# 4.  ADDITIONAL FUNCTIONAL EXTENSIONS TO THE
# SYSTEM ARCHITECTURE

The system architecture described in Section 2 consists of a single data channel connecting a single application program to a single display device.  This section describes four new functional units that can be added to the system allowing a single data channel to be split into two or more data channels, and two or more data channels to be spliced into a single data channel.  These extensions support simultaneous connection of multiple dissimilar display devices to a single application program and simultaneous connection of multiple application programs to a single display device.

## DATA CHANNEL SPLITTER

The ability to provide duplicate graphic output on two or more display devices (i.e., to slave multiple, possibly dissimilar, graphics display devices to a single application) can be achieved by adding a Data Channel Splitter function to the data channel.  The Data Channel Splitter creates one or more new data channels, copying graphic output from the original data channel onto the new data channels.  Graphic output appearing on the display device connected to the original data channel is duplicated on the display device connected to each of the secondary data channels, Figure 4.1.
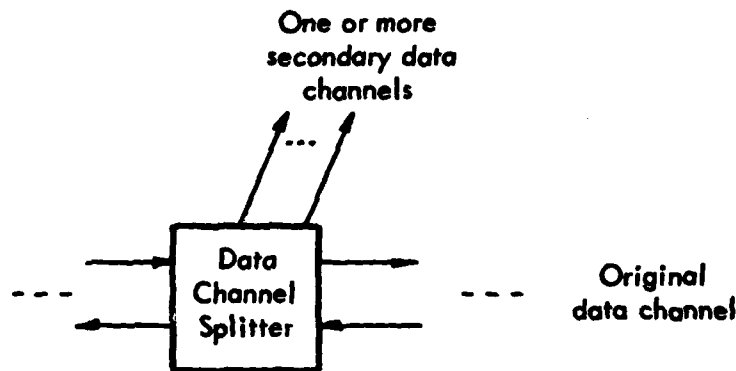


Figure 4.1   Data channel splitter

## MULTIPLE DISPLAY DEVICE CONTROLLER

It is sometimes necessary for a single application to support multiple terminals simultaneously.  Multiple terminal configurations can be realized using the architecture described in Section 2.  This would be achieved by physically partitioning the application into separate components (one per display device).  Each component would manage its own device as well as communications with other components of the application.  An

example of a multiterminal application structured in this manner is the Warfare Effectiveness Simulator program at the Naval Oceans Systems Center.

The addition of a Multidevice Controller function provides the multiterminal capability within the graphics system, eliminating the need to restructure the application. It allows a single application program to be simultaneously connected to and interact with multiple (possibly dissimilar) graphics display devices. The application program can display the same or different pictures on each terminal. It also allows the application to retrieve graphic input from any of the display devices.

The Multidevice Controller demultiplexes a single data channel containing information for two or more devices into separate data channels, one per device. The reverse operation is performed for graphic input data, i.e., it multiplexes graphic input data from two or more data channels onto a single data channel.

The Multidevice Controller can be introduced at one of several possible places on the data channel, the placement having significant implications with regard to the functionality of the graphics system. To maximize functionality, the Multiple Display Device Controller is placed between the Application Interface and the Clipper, Figure 4.2.

This allows the application to use each device independently of the others in terms of the information displayed. Since clipping has not yet been performed, each device can potentially display data under the control of different windows and coordinate ranges. Placement of the Multiple Display Device Controller after the Clipper would mean that each device would have to be treated in terms of the same coordinate range and viewing parameters.
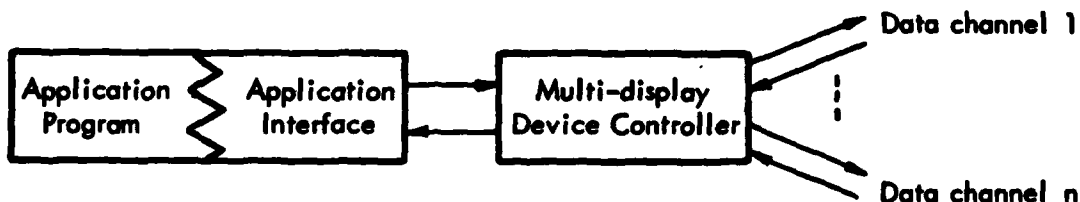


Figure 4.2   Multiple display device controller

## DISPLAY DEVICE MULTIPLEXOR

The graphics system architecture allows a display device to be connected to and display the output from only one application program at a time. In some situations, a user may desire to interact concurrently with two or more programs. The Display Device Multiplexor is a functional extension to the graphics system that provides this capability. Two different modes of concurrent use are possible.

The first mode allows the display surface of a terminal to be subdivided by the user at execution time into two or more nonoverlapping areas, each of which appears to be an independent terminal that can be allocated to an application program. Thus, a user at a single terminal might simultaneously view and interact with the graphics of two or more programs. Currently, considerable research is being conducted using this technique (often called multiple windows) for textual presentation. The work, to date, has been limited to nonscaled, nonpositionable text. Two-dimensional textual displays such as tables and charts that require more than the allocated screen area cannot be viewed in their entirety without their representation and formatting being destroyed (since they cannot be scaled or adjusted to small display areas). The Display Device Multiplexor extends this initial concept by allowing both scalable text and graphics to be displayed as originally prepared by the application programs in individually allocated areas of the display surface.

The second mode allows the display device (including its associated input devices) to be simultaneously connected to and switched, under user control, between two or more application programs (much like a television is switched between channels). Examples of this second technique are found in many contemporary systems where, for example, a user types some particular control key to regain control of the terminal from an application program, permitting him to subsequently redirect control to some other application.

Functionally, the Display Multiplexor either spatially or temporally integrates two or more data channel streams into a single stream. For the former, it suballocates the display surface into two or more subsurfaces, associates a particular channel with a subsurface, and scales and relocates pictures for their particular subsurface. For the latter, it maintains and switches the graphic context for each application. Upon user input, it routes the input data to the correct data channel and application. The Display Multiplexor is positioned between two or more Storage Managers and the Device Order Generator, Figure 4.3.
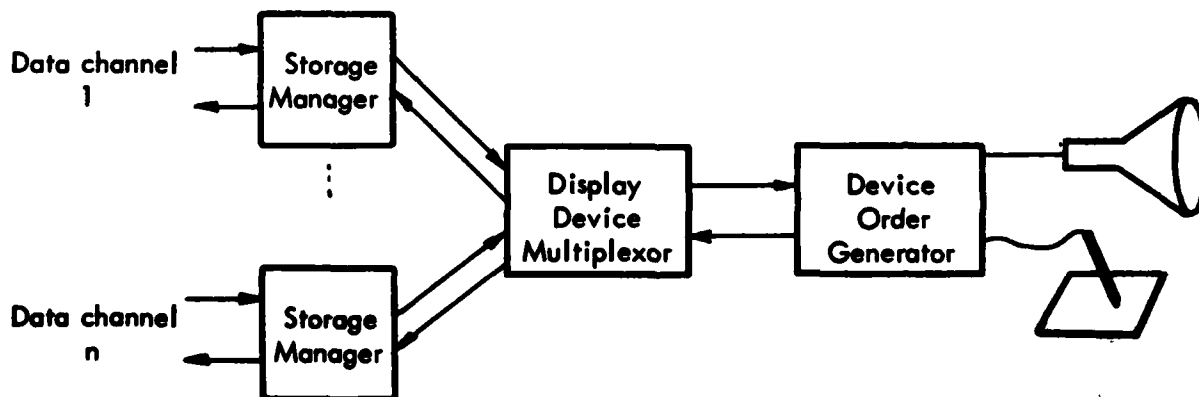


Figure 4.3  Display device multiplexor

## MULTI-WAY DATA CHANNEL SWITCH

The Multi-Way Switch provides a multi-way data channel routing capability to the graphics system. The switch accepts one or more data channels as input and provides one or more data channels as output, Figure 4.4. It allows any of the output data channels to be connected to any of the input channels. The switch is under the control of a switch operator who independently determines the input-data-channel-to-output-data-channel connection (and hence the picture displayed). The switch operator can change input to output connections at will and has total control over the switching mechanism.

This function provides the capability necessary for a controllable graphics distribution system and might be used to control output selection for a large screen display or a hierarchical picture distribution system.
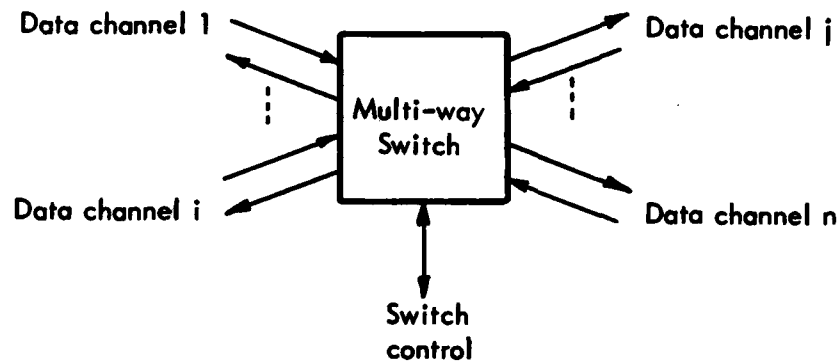


Figure 4.4 Multi-way switch

# 5. COMMUNICATIONS

Sections 2, 3, and 4 described the architecture of the Graphics System and functional extensions. For any two functions to communicate, a mutually agreed upon set of communications conventions must exist. This section briefly summarizes communications at two points within the system, between the Application Program and the Application Interface and between the Normalizer and the Storage Manager. The section also summarizes the external data format used for graphic files.

## APPLICATION PROGRAM/APPLICATION INTERFACE COMMUNICATIONS

This is the point at which an application program makes requests of the graphics system for graphic services. The program communicates with an Application Interface using Graphics Language [2]. Graphics Language provides a set of device-independent two-dimensional vector graphic primitives upon which more complex systems (e.g., three-dimensional) can be built. Device independence is achieved by virtue of the fact that the graphic primitives are generic capabilities and their arguments are specified in a universal value range.

Graphics Language includes device-independent, generic constructs for

1. Specifying the aspect ratio of the virtual display surface used, a viewport within that display surface, and a coordinate range to be mapped onto that viewport.

2. Grouping graphics primitives in named segments that can be created, destroyed, merged, made visible or invisible, highlighted, or made sensitive to touching.

3. Specifying graphics entities such as lines, dots, text, and arcs (both in relative and absolute form).

4. Specifying the display characteristics of graphic entities (e.g., font, intensity, color).

5. Accepting display input from function keys, analog devices, or pointing devices (including segment touching).

6. Retrieving graphics system status information.

Graphics Language is rich enough to make use of the majority of functions provided in contemporary display devices. However, some display devices have special-purpose features that Graphics Language cannot use. For example, Plato plasma terminals may have built-in 35mm slide projectors allowing simultaneous rear projection of a transparency on the graphics display surface. Raster scan displays may have video disks, video drums, video tapes, or live video images mixable at the display device. To control these special features or other device-idiosyncratic functions, constructs are

provided in Graphics Language for sending and receiving display-device-specific information (e.g., "does this Plato terminal have a built-in 35mm slide projector" or "turn on the rear projection slide projector and cycle the carrousel to the 61st slide"). However, use of device-specific features by an application program may render the program device-dependent.
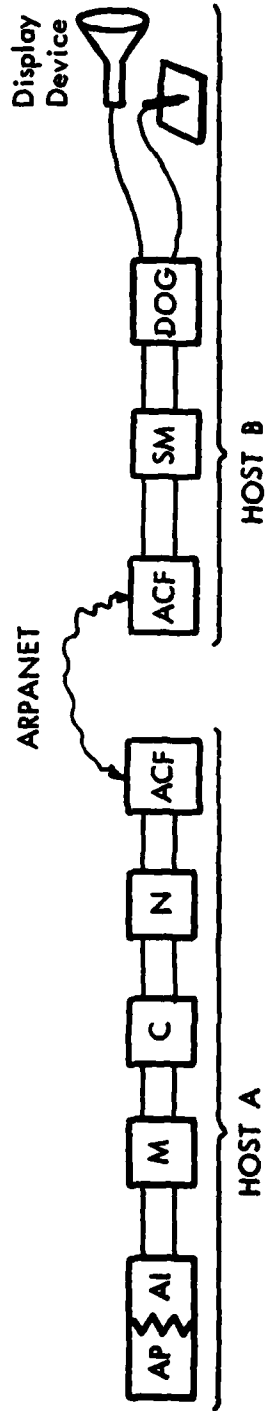
## NORMALIZER/STORAGE MANAGER

The graphics system must be distributable across two or more possibly dissimilar hosts. Distributability makes it possible for the system to adapt to a variety of computational and communications resource configurations. To achieve distributability, however, requires that the data channel be divided into two or more subpieces, with each subpiece potentially residing in a different host. The best point for dividing the data channel is between the Normalizer and the Storage Manager, the location on the data channel at which bandwidth requirements are typically lowest, and, hence, an optimal point for a minimum bandwidth communications link.

The two physical subportions of the data channel are connected by a Communications Function (Fig. 5.1). A Communications Function embodies all the communications and data transformation routines necessary to make two physically separate subportions of a data channel function as one logical data channel. In particular, the Communications Function is responsible for mapping data between the two hosts, a mapping that may involve changes in data formats (e.g., converting one's complement integers to two's complement integers or EBCDIC text to ASCII text or BCD text). It is also responsible for insuring reliable communications over the external communications link; hence, it must include error detection and correction.

Graphics Protocol [1] is the data communications form used by the Communication Function. It is intended specifically for communications between the two halves of a communications function (hence, two portions of a data channel). It and the communications function are transparent to both the application program and the terminal user; neither the application programmer nor the user need have any knowledge of its characteristics or even its existence, since it is inaccessible to both. The following briefly summarizes its two most important properties.

*Graphics protocol optimizes vector specification.* It contains multiple formats for vectors. Both absolute and relative vector formats are provided, with multiple precisions available for the latter. Precisions are settable by other protocol constructs. This allows the system to achieve better communication performance for pictures containing large numbers of short vectors. Graphics Protocol also contains vector streaming. Two or more vectors, regardless of format, can be sent using a single protocol element. This allows better communications performance for batches of two or more vectors, since each vector does not have to be placed in a separate protocol element.

*Graphics protocol allows the size of the logical containers in which protocol elements are placed to be determined at execution time.* Each protocol element has a corresponding logical container. The size of the container, its alignment within the data stream, and the

KEY:
ACF – ARPANET Communication Function
AI – Application Interface
AP – Application Program
C – Clipper
DOG – Device Order Generator
M – Model
N – Normalizer
SM – Storage Manager

Figure 5.1 Graphics system with external communications function

position of data within a container are specified by other protocol elements. This permits a single device-independent protocol to be dynamically configured to meet a variety of communication bandwidth and computer processing/display device requirements. Examples of container selection for three different communications/processing requirements are provided.

1. Minimum Bandwidth. To minimize bandwidth, the system would choose for each protocol element the smallest container that holds the maximum precision of the corresponding protocol element. The containers would be unaligned, producing a "bumper-to-bumper" stream of bits. For example, the protocol element <segment.name> can be an integer between 0 and 65535. A 16-bit container would be the minimum size required for sending <segment.name>s. On the other hand, <highlight.state>, an integer, either 0 or 1, would require a 1-bit container.

2. Maximum Throughput. To maximize throughput, the system would choose for each protocol element a container whose size and alignment were integral machine boundaries. For example, for maximum throughput between two PDP-10s, <segment.name> could be placed right-justified in an 18- or 36-bit container corresponding to either a PDP-10 halfword or fullword. The container could be aligned on an 18- or 36-bit boundary within the protocol stream.

3. Minimum Computation. One of the host computers may have a limited computational capability. In this case, container sizes, alignments, and data justification can be chosen to minimize computational requirements in that host, i.e., shifting and masking operations needed to transform data between Graphics Protocol and the data format within a host. (This may lead to containers whose sizes, alignments, and justifications are similar to those derived for maximum throughput.) An example would be the transmission of <highlight.state> to a small 16-bit machine such as a LSI-11. To minimize processing, the single bit could be sent left-justified in a 16-bit container aligned on a 16-bit boundary. This would allow the bit to be tested by simply loading the word into a register and branching on the sign.

## GRAPHIC FILES

Graphic Files provide the mechanism for saving and reincorporating previously generated graphics output into the same or a different application. Since files must be usable by potentially different applications connected to different devices on different hosts, the content of the file must be application, host, and device-independent. The Graphics Protocol defined above meets these requirements. It is both application-independent and display-device-independent. In addition, since it is a sequential stream of bits, it can be easily represented on any sequential file storage mechanism.

## 6. IMPLEMENTATION STRATEGY

The preceding sections discussed the design requirements for a Command and Control oriented graphics system and the architecture developed to meet those requirements. The following section describes three upward compatible implementations of that architecture. The implementations are denoted by level numbers corresponding to the graphic functions included in the system. This multilevel implementation strategy provides an orderly stepwise development, testing, and refinement of graphic system functions. The section concludes with a discussion of how the system can be supported on other processor types.

### LEVEL I SYSTEM

A Level 1 system provides a single-device-to-single-application interactive graphics environment. It consists of a data channel and the five basic graphic functions described in Section 2, i.e., Application Interface, Clipper, Normalizer, Storage Manager, and Device Order Generator.

A Level 1 system was implemented by ISI in January of 1977. It ran on the PDP-10 under the TENEX/TOPS-20 operating systems and supported the following application languages and display devices:

|  |  |
|---|---|
| Languages: | FORTRAN-10 |
|  | Bliss-10 |
|  | Macro-10 |
|  | Interlisp-10 |
| | |
| Display Devices: | Tektronix 4000 series |
|  | Genisco GCT-3000 |
|  | Hewlett-Packard 2648A |

The system was structured with all functions under a single process. Data channel communications as well as communications between the application program and the Application Interface was via procedure call. Device type selection took place at linkedit time. A particular application program was linkedited with an appropriate Application Interface for the required language, an Order Generator for the desired display device type, and the remainder of the graphics system forming a single load module. Selection of the specific display device to be used for a particular session was performed at execution time.

A display device can be connected to the graphics system by any communications interface supported by the host operating system. In the case of TENEX/TOPS-20 these include:

1. Hardwired connection of the display device to the host.

2. ARPANET TIP connection. (This looks just like a hardwired connection since the TIP merely provides communications between the graphics system and the display device.)

3. ARPANET socket-to-socket connection to another processor.

## LEVEL 2 SYSTEM

The Level 2 system, like the Level 1 system, provides a single-device-to-single-application interactive graphics environment. The Level 2 system also provides an External Communications Function between the Normalizer and Storage Manager functions, the point of lowest bandwidth on the data channel (see Section 2). For purposes of discussion, the Application Interface, Model, Clipper, Normalizer, and half of an External Communications Function are collectively referred to as a "Frontend"; the other half of the External Communication Function, Storage Manager and Order Generator are collectively referred to as a "Backend". A Frontend is device-independent, since it contains only device-independent functions. A Backend is device-dependent since it contains a device-specific Device Order Generator.

The Level 2 system was implemented by ISI in June 1979. It runs on the PDP-10 under the TENEX/TOPS-20 operating system and supports the same languages and devices supported by the Level 1 System. Data channel communications within the Frontend and Backend is via procedure call. Communications between the two halves of the External Communications Function (i.e., between the Frontend and Backend) is by ARPANET socket-to-socket connections. Use of the ARPANET is motivated by the fact that it is a solid, well-proven, widely used digital computer communications mechanism with guaranteed delivery of messages (required by the graphics system) and is representative of the type of communications mechanism that might ultimately be involved, AUTODIN-II. Data transmission is according to Graphics Protocol, highlighted briefly in Section 5. Container size, container alignment, and data justification are negotiated between the Frontend and Backend during the connection sequence, allowing communications/processing optimization.

Selection of the type and location of the display device and the location of the Backend to be used for a particular session are done at execution time. A particular application program is linkedited with the appropriate Application Interface and the remainder of the Frontend forming a single load module. At runtime, the Frontend starts up and establishes a socket-to-socket communications channel with an appropriate Backend on the same or another processor. Separate Backends are required for each host-type/display-device-type supported. ISI has produced PDP-10 TENEX/TOPS-20 Backends for the Tektronix 4010, 4012, 4014 series, Tektronix 4027, Hewlett-Packard 2648A, Hewlett-Packard 9872A, Genisco GCT-3000, and Advanced Electronic 512.

As with the Level 1 System, the display device is connected to the graphics system by any communications interface supported by the host operating system. For TENEX/TOPS-20, this includes hardwired, ARPANET TIP, or ARPANET socket-to-socket connections.

In April 1980, the Level 2 system was upgraded to support device-independent graphic files. The file input and file output functions are contained in the Backend. Graphic files store Graphics Protocol.

## LEVEL 3 SYSTEM

The Level 3 system provides a multiple-device-to-multiple-application interactive graphics environment. It will allow the simultaneous connection of multiple dissimilar display devices to the same application program as well as concurrent connection of the same display device to different application programs. It includes all the features described in the Level 2 implementation as well as the architectural extensions described in Section 4. Implementation of a Level 3 system is being planned for FY81 and beyond.

## IMPLEMENTING THE GRAPHICS SYSTEM ON ADDITIONAL PROCESSOR TYPES

ISI's Level 2 implementation of the graphics system supports a graphics application environment distributed across two PDP-10s running the TENEX/TOPS-20 operating systems. Others may produce Level 2 compatible Frontends and/or Backends to run on different machines. The following provides a guide for such implementations.

Supporting a Frontend or Backend on another type of computer requires that the Frontend or Backend be rewritten for that processor type. For a Frontend, the input specifications are defined by the Graphics Language and the output specifications by the Graphics Protocol. For a Backend, the input specifications are defined by the Graphics Protocol, and the output specifications by the order codes for the device type being supported.

Different Backends might be written for the same processor type to support processors with substantially different resources (e.g., shipboard processors with limited storage). For example, one might be written to fully optimize the host processor's capabilities and another one written to preserve device independence up to the Order Generator.

A new Frontend is made available to the graphics users by incorporating it in one or more graphics directories on participating machines, thereby permitting users to link it with their application program. New Backends are made available by incorporating information about the Backend's location, a generic name by which it can be invoked (corresponding to the type of Backend and the processor type on which it runs), and other appropriate information in the data file that will be accessed by the connection mechanism. A direct ARPANET socket-to-socket connection is used for communications between the two components.

## 7. EXAMPLE CONFIGURATIONS AND SCENARIOS

This section provides the reader with examples of the various ways the graphics system can be configured and with scenarios of use. It begins by describing important configurations provided for by the graphics system, then provides specific examples of possible system usage. The section concludes with a short discussion of issues relevant to runtime selection and placement of a backend and communications optimization. All of the configurations and scenarios apply to ISI's Level 2 or Level 3 implementation and their use on the ARPANET.

### SINGLE DEVICE, SINGLE APPLICATION

Probably the simplest configuration is that shown in Figure 7.1, a single user application interacting with a single display device. The user's application program is linkedited with a Level 2 Frontend. An ARPANET socket connects the Frontend with a Level 2 Backend. Communication between the Frontend and Backend is by Stream Graphics Protocol. The Backend is in turn connected to the display device. The display may be either local to the host or remote, e.g., connected to an ARPANET TIP or another host. Communications between the Backend and the display is by the order codes interpreted by the device. In Figure 7.1, the Frontend and Backend are on different host processors. Both the Frontend and Backend can reside in the same host processor, giving rise to Figure 7.2. The selection of the Backend host, as well as the type and location of the device, is performed at execution time.
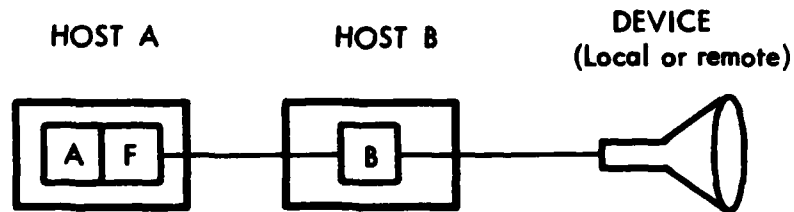


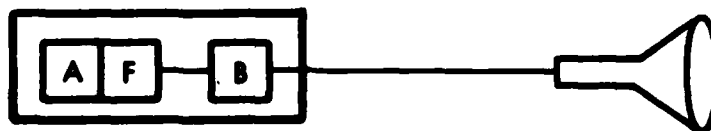Figure 7.1   Frontend and backend running on different hosts



Figure 7.2   Frontend and backend running on same host

## MULTIPLE DEVICES, SINGLE APPLICATION

Figure 7.3 shows a configuration available if a Multiple Display Device Controller is added to the system. This allows a single application to simultaneously generate output for and accept input from multiple display devices. An application can display either the same or different graphics on each of the display devices. Thus, the configuration allows for multiple people interacting individually or collectively with the same picture, or multiple people interacting individually with individual pictures. Any combination of display devices is allowed, e.g., some could be Tektronix, others Hewlett-Packards. As with the single-device, single-application configuration, the display devices, their location, and the location of the respective Backends are determined at execution time.
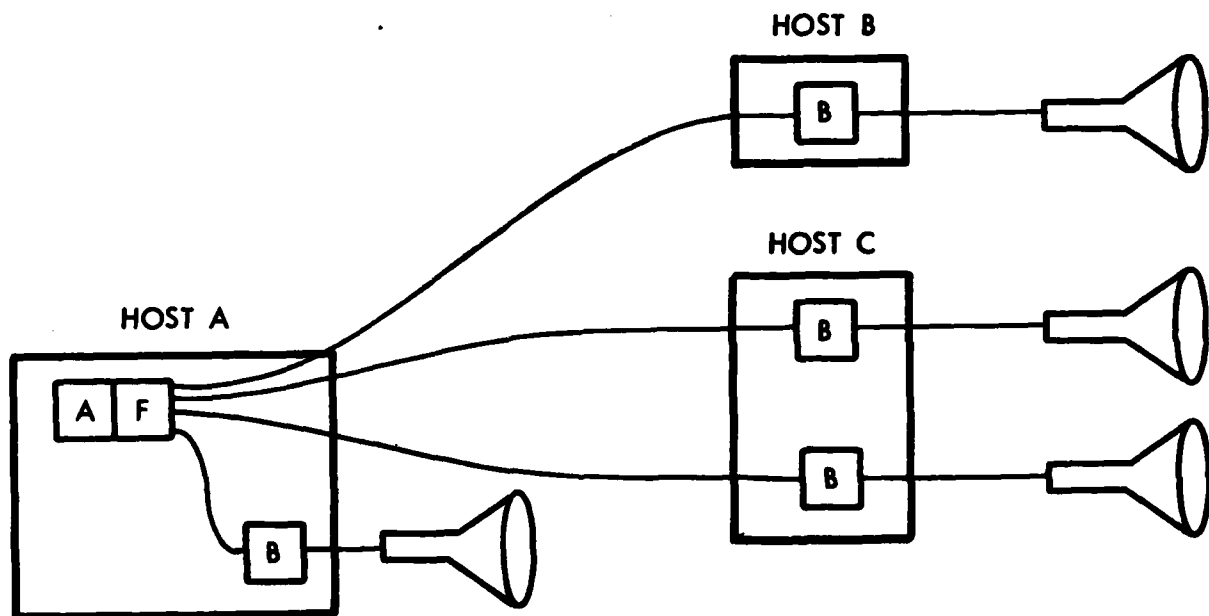


Figure 7.3   Single application with multiple device

## SINGLE DEVICE, MULTIPLE APPLICATIONS

Figure 7.4 shows a configuration available if a Display Device Multiplexor is added to the system. This allows a single display device to simultaneously display graphics from and supply input to multiple independent applications. The various applications and their

associated Frontends can reside either on the same host as the Backend, or on different hosts.  As with the single-device, single-application configuration, the location of the display device and the Backend is determined at execution time.
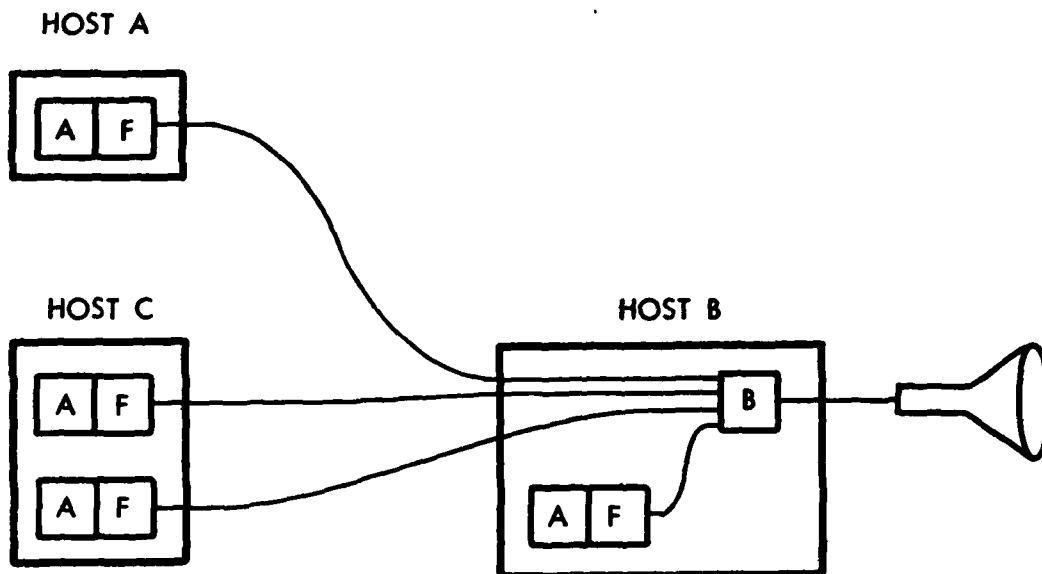


Figure 7.4   Single device with multiple applications

## MULTIPLE DEVICES, MULTIPLE APPLICATIONS

The capabilities described in the previous two configurations can be combined to produce configurations such as Figure 7.5.  The figure depicts both applications generating graphics for one or more devices and devices displaying graphics from one or more applications.

## EXAMPLE SCENARIOS

The above configurations form a family around which a variety of scenarios of usage can be constructed.  The following are three possible scenarios.

*Scenario 1*

A user in the field has a hand-held graphics terminal.  The terminal supports secure radio communications to a base station that is in turn connected to a digital communications network similar to the ARPANET.  The user needs to

interact with and display the graphical output of a program available on a computer on the network. The configuration in either Figure 7.1 or 7.2 would meet the user's display requirements. In this configuration, the Backend to display device link would be a combination of the communications network and the radio link.
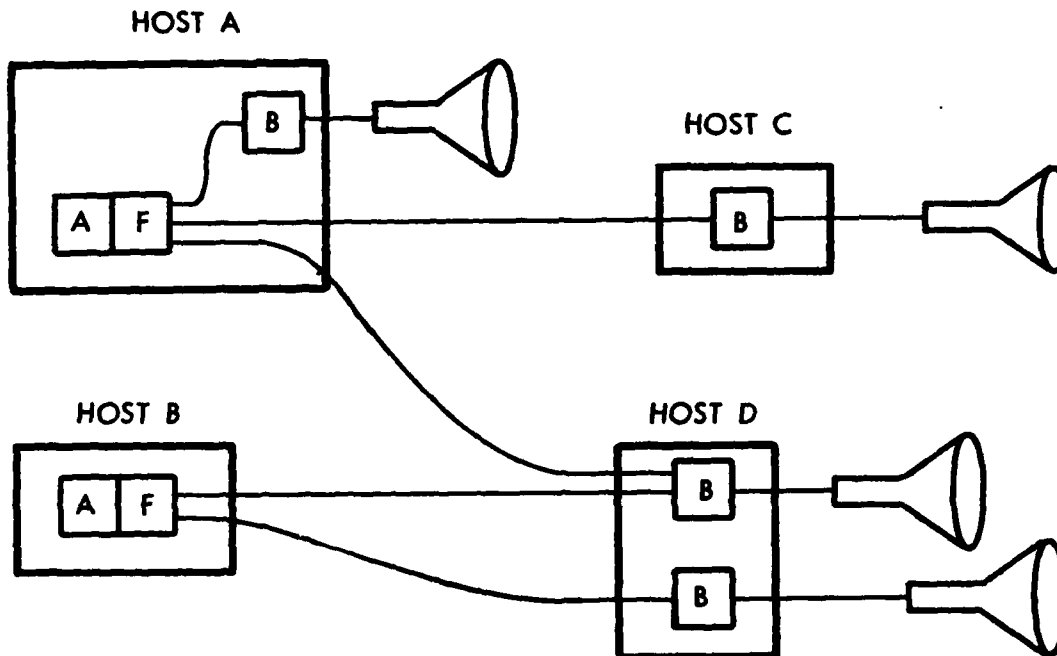


Figure 7.5    Multiple applications and multiple devices

*Scenario 2*

A user is on board ship. The ship has, in addition to a display device, a small host processor capable of supporting limited computation. The ship is connected to a shore-based digital communications network similar to the ARPANET via a low-bandwidth ship-to-shore link. The user needs to interact with and display the graphic output of a program that runs on a computer on the network, but cannot run the entire application on the local ship-board processor due to resource constraints. The configuration in Figure 7.1 would meet both the user's display and the bandwidth requirements. The Backend for the display device would run on the ship's local processor, with Frontend/Backend communication appearing at the ship-to-shore link.

*Scenario 3*

Two or more users, each with his own graphic display device, desire to communicate with each other. The users could be located in the field, on board ship, and/or in command centers. The communications include the exchange of and interaction with shared graphics information. The configuration in Figure 7.3 would meet the requirements for the above scenario as well as others involving graphic conferencing. As noted in the description of this configuration, the application can supply an individual display with the same image as displayed on all other displays, or one or more individual images.

Up to this point, the discussion of configurations and scenarios has been limited to direct communication between one or more applications and one or more display devices. A second form of communication is available through the graphic file capability, which allows a graphic picture to be stored and reincorporated into the same or another application at some later time. The following is a scenario illustrating one use of graphic files.

*Scenario 4*

A particular graphics system user is requested to make available to another user a copy of a particular chart generated by the application program the first user is running. He does this by making a graphics file of the chart, and sending it via ordinary file communication means to the second user. The second user subsequently displays the contents of the file on his display device, possibly incorporating it into a more complex display. An example of this is shown in Figure 7.6.

While the file capability for sending and receiving static pictures can be used with any of the previously described configurations, its usage takes on new interpretation when used with configurations similar to Figure 7.3: The combination of files and multiple users forms the basis for a graphics-based briefing system. One or more briefers control the order and presentation of graphic information to the other participants. The "briefing slides" are merely contents of graphic files.

The ability of the graphics system to allow display devices to be located anywhere is an important aspect of such a briefing system. It makes possible a briefing in which neither preparers, presenters, nor participants need be collocated. They can all be geographically separated, connected only by suitable communications.

The final scenario addresses the real-time selection and distribution of graphic pictures. It is included to demonstrate both the flexibility of the system and possible extensions.
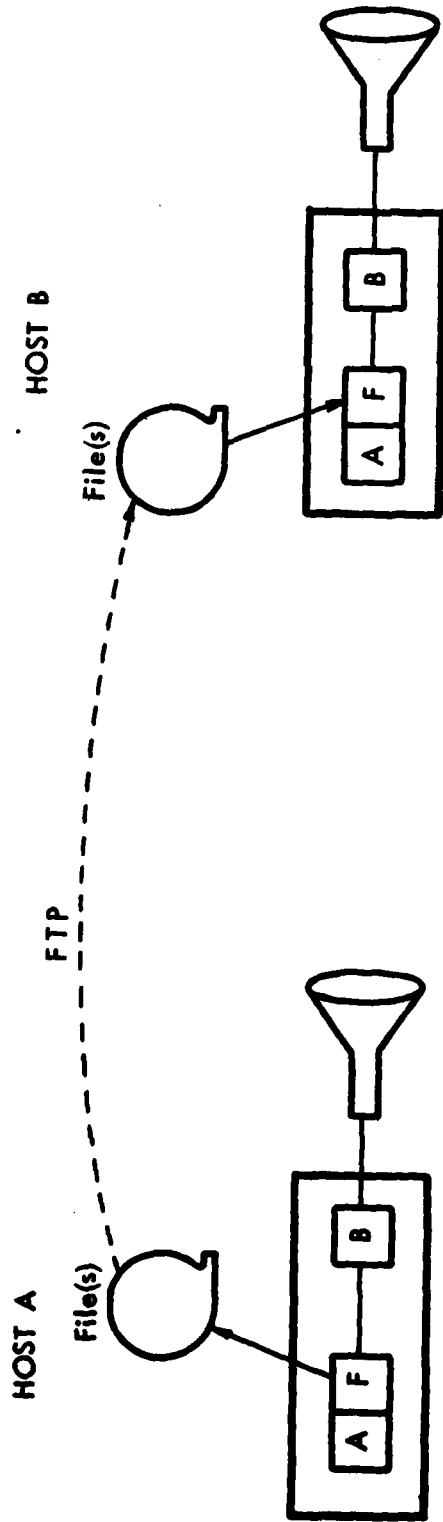
Figure 7.6   File communication

*Scenario 5*

Consider a hierarchically structured organization such as the command structure within the military. Each level within the hierarchy is responsible for receiving information from its immediate subordinates, possibly integrating it with additional information, and supplying this information to its immediate superior. The graphics system, with the addition of the Data Channel Splitter and Multi-Way Video Switch, can be configured to support this structure. A typical configuration is shown in Figure 7.7. In order to simplify the figure, host computer information has been removed.

As with other configurations, the location of the display device, application, etc., can vary and is determined at execution time. At the extreme left, two or more graphics applications are being run. Their output, in addition to going to the display device, is simultaneously feeding a Multi-Way Video Switch. The current video output from each of the applications can be viewed by a Video Switch operator and selected for forwarding to the next higher level in the hierarchy, which in the figure is another Video Switch. The next Switch receives input from both subordinate Video Switches and local applications, and in turn forwards a picture to its superior. This hierarchical structure continues, with final graphic output appearing on a display at the highest level within the hierarchy.

Note that the problem is not unlike that found in commercial television networks. At the lowest level of the hierarchy, camera men supply pictures to a director, who chooses one and forwards it to the next higher level in the hierarchy, perhaps a network director. The network director chooses a picture from remotely and locally generated pictures and forwards one to local stations. An analogous function is performed by the local station director.

Ultimately each viewer sitting in front of a television performs the same type of function, selecting from the stations in his community or locally generated pictures (e.g., a good book).
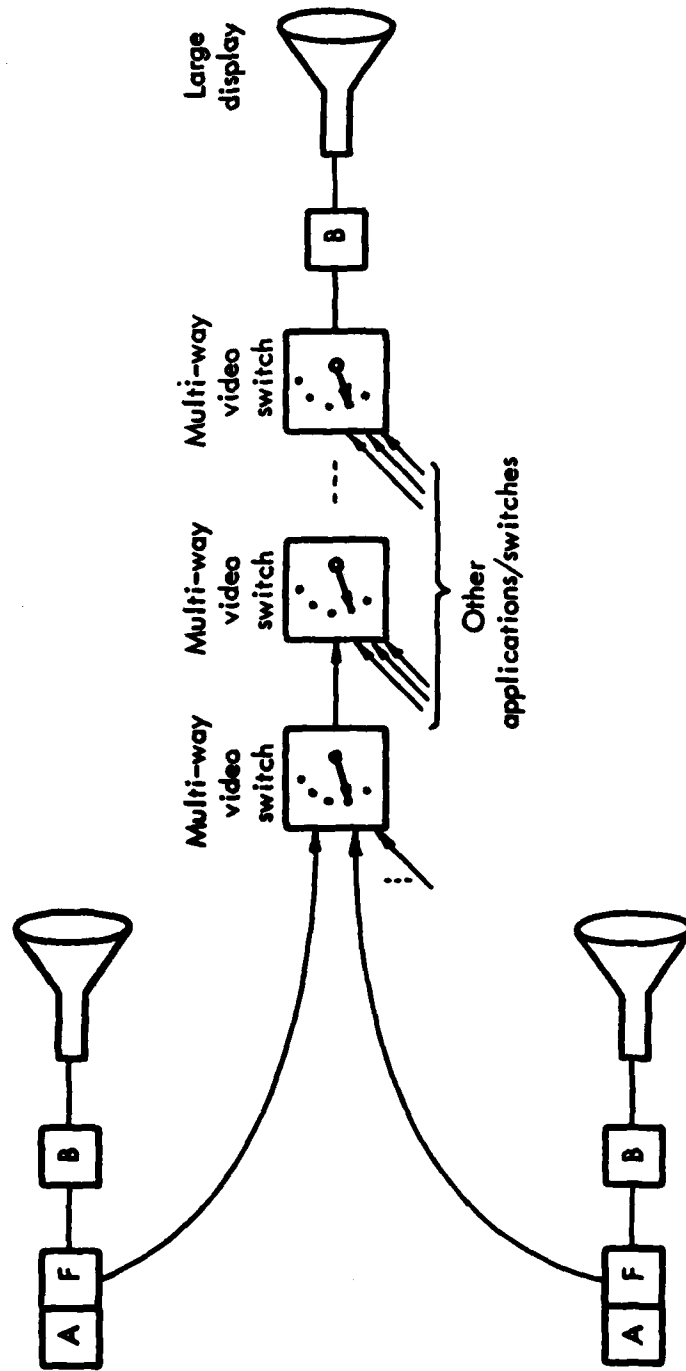
Figure 7.7  Real-time video distribution

# 8. CONCLUSION

The graphics system architecture described in this document has been designed to meet the display requirements for vector graphics command and control applications. Its three outstanding attributes are:

1. It is device-independent. Graphics application programs can be written without regard to the particular display device on which the output will ultimately be displayed.

2. It is distributable. The graphics system can be distributed across multiple host computers connected by a communications network.

3. It is extensible. The architecture allows new functions to be easily added to the basic system by extending its capabilities.

The system will satisfy command and control graphics requirements through 1981. Its extensibility will allow it to grow to meet future requirements beyond 1981.

34

## REFERENCES

1.  Bisbey, Richard, II, and Dennis Hollingworth, Graphics Protocol  (forthcoming).

2.  Bisbey, Richard, II, Dennis Hollingworth, and Benjamin Britt, *Graphics Language*, USC/Information Sciences Institute, TM-80-18, 1980.

3.  Newman, W. M., and R. F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, New York, 1979,  second edition.

4.  Sproull, R., "Network Graphics Isn't Networking," In *Proceedings of the Berkeley Workshop on Distributed Data Management and Computer Networks*, pp. 38-40, University of California, Lawrence Berkeley Laboratory, May 1976.