MULTIPLE PARALLEL GKS WORKSTATIONS

By

MUKUND JAGANNATHAN

Bachelor of Science
Bangalore University
Bangalore, India
1978

Master of Science
Kansas State University
Manhattan, Kansas
1981

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1986

MULTIPLE PARALLEL GKS WORKSTATIONS

Thesis Approved:

D. E. Hedrick
Thesis Adviser

D W Grace

M J Folk

Norman N Durham
Dean of the Graduate College

PREFACE

This work deals with the design and implementation of a graphical workstation , applicable to the Graphical Kernel System (GKS). A full implementation of the design methodology includes a level m subset of GKS as well. The design is implemented under UNIX.

I wish to thank my advisor Dr. G.E. Hedrick, for all the timely help, suggestions and encouragement for this work as well as during the course of my academic pursuits. I thank Dr. M.J Folk and Dr. D.W. Grace for having served on my Graduate Committee and expressing interest and enthusiasm on this project. I thank Mr. Mark Vasoll and Mr. Gregg Wonderly for their help in the skillful use of UNIX. A number of graduate students and faculty members of the Department of Computing and Information Science, deserve a note of thanks, for maintaining an atmosphere of friendliness and good humor.

I express my gratitude and thanks to my brother Dr. J. Murali and Mrs. Murali for providing a good part of financial support. I thank J. Mitra and my parents for their encouragement and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS

ACM	- Association of Computing Machinery
ANSI	- American National Standards Institute
ANSI/X3H3	- ANSI subcommittee for Graphics Standards Development
COMSC	- Computer Science. Used to specify Computing & Information Sciences Department at Oklahoma State University.
CRT	- Cathode Ray Tube
DIGGRAF	- Device Independent Graphics from FORTRAN
DIN	- Deutsches Institute fur Normung, the West German Standards Organization
gdi	- Global Data Interface
GINO-F	- Graphics Input/Output
GKS	- Graphical Kernel System
GRAF	- Graphical Extensions to Fortran
GSPC	- Graphics Standard Palnning Committee of ACM.
IDIGS	- Interactive Device Independent Graphic System A graphics standard of Norway (also adopted by The Netherlands)
IFIPS	- International Federation of Information Processing Societies

IFIPS/WG 5.2	- IFIPS Working Group 5.2 (Graphics Standards)
ISO	- International Standards Organization
ISO/TC97/SC5/ ° WG2	- International Standards Organization ; Information processing ; Subcommittee 5 ; Working Group 2.
MIT	- Massachusetts Institute of Technology
NGS	- Network Graphics System
Omnigraph	- A Simple Terminal -independent Graphics Software Released by Xerox Palo Alto Research Center
OSU	- Oklahoma State University, Stillwater, Oklahoma.
PHIGS	- Programmer's Hierarchical Interface to Graphics Systems
SIGGRAPH	- Special Interest Group on Graphics of the ACM
UIMS	- User Interface Management System
UNIX	- An Operating System. UNIX is a trademark of AT&T Bell Laboratories.
VDI	- Virtual Device Interface
VWS	- Virtual WorkStation

CHAPTER I

INTRODUCTION

Introduction

Computer Graphics is the science of synthesis of pictures, either real or imaginary. Graphical displays using computer based systems have been in use for over three decades. The MIT's Whirlwind Computer and the Defense Department's SAGE Air Defense System of the 1950's used CRT's for graphical displays. The number of leading publications and books pertaining to Computer Graphics is too large for enumeration. It suffices to mention that quality research has been pursued in this area and the trend will continue in the future.

The development of a standard of practice follows an initial period of growth of an industry. Computer Graphics is no exception to this phenomenon. Initially the developers of graphics systems such as GINO-F, Omnigraph, GRAF etc., adopted a uniform standard of practice and consistent product guidelines as far as their product releases were concerned. Each package that existed would claim that its package was an industry standard. The diverse standards so established called for national bodies such as Association of Computing Machinery - USA (ACM) and their

counterparts in the United Kingdom, Germany, The Netherlands etc., to consider the issue of graphics standards development seriously. This led to the formation of GSPC (Graphics Standard Planning Committee) for developing the GSPC Core Graphics System (hereafter referred to as the Core System). The GINO-F package of Britain was the British work item for graphics standardization, GKS (Graphical Kernel System) under DIN of Germany, IDIGS (of The Netherlands) etc., were other standards that were being developed around that time. Due to active interaction in the field of graphics among many nations, a special group under the International Standards Organization (ISO) was formed to develop an International standard for computer graphics. The development of manufacturers standards and subsequently the various national standards were the precursors to the ISO GKS. The current draft proposed international graphics standard known as GKS is the result of an intense effort put forth by the special group under ISO/TC97/SC5/WG2 (The working group under subcommittee 5 of the Information Systems group of the International Standards Organization).

In this thesis, a unique GKS workstation design is being considered. It is necessary to provide a brief look at graphics standards, before outlining the workstation design. In later sections of this chapter, a Core System overview is provided. Chapter II describes GKS along with some notes of comparison with respect to the Core System. This is followed by an implementation of a GKS subset under

UNIX, residing at level m of ANSI GKS [28]. In chapter IV, the unique workstation design is being outlined. Results and discussions of the design and implementation follows the above.

Graphical Standards Development

The history of computer graphics spans approximately three decades. A recognized contribution in terms of research in computer graphics started in the early 1960's. In 1963 a Ph.D thesis submitted at MIT [32] by Ivan Sutherland was a turning point for the computer graphics industry. This thesis is referred to as the SKETCHPAD. SKETCHPAD described computer graphics as a complete system by itself. The concepts of a graphical model, hierarchy of graphic entities of a model, the data structures necessary for graphic primitives and topology of the model were among the many introduced. Geometric models, transformations, clipping and windowing were discussed at length. The manipulation of a graphics model in an interactive manner was also described. No such comprehensive picture of a graphics system existed prior to SKETCHPAD. Following this pioneering work, research in the field of computer graphics centered around the improvement of basic algorithms for primitives, models and graphic systems. Graphics hardware was becoming increasingly sophisticated with a high degree of local control.

Manufacturers such as Tektronix, digital, California Computer Products etc., produced sophisticated hardware for graphics. Direct view storage tube, silicon target tube, color plotters etc., with easy to use display processors were being designed. A principal shortcoming that existed during this fast pace of technological strides was a lack of unity among manufacturers and a total neglect of a unified standard of practice. The user community found it extremely difficult to port graphics applications across systems. Even upward compatibility was remote. Naturally a reluctance in investing in graphical systems was observed.

In 1974 ACM's special interest group in graphics, hereafter referred to as SIGGRAPH, formed a committee to look into the development of a graphics standard. This committee was known as GSPC (Graphics Standard Planning Committee). In 1975 the IFIP (International Federation for Information Processing) formed a working group known as IFIP/WG 5.2 to develop a methodology for computer graphics standardization. The growth of graphics in Europe was rapid as well. As mentioned in the previous section, European nations were developing their individual computer graphics standards at the time. However, the European nations had realized the need for a computer graphics standard to promote better cooperation among community nations. There definitely existed a need for a graphics standard.

GSPC Core System

The GSPC was formed in 1974 to look into the matter of computer graphics standardization in the USA. In 1975, the IFIP formed a working group to consider the matter of a computer graphics methodology. A workshop on graphics known as "Workshop on Graphics Methodology" was conducted in Seillac, France in 1976. Members of GSPC were present at the workshop. The GSPC was highly influenced by the efforts of IFIP and in particular, the workshop proceedings of Seillac. The workshop brought to attention, the need to study the structure of application programs in order to arrive at a software design method. The need to separate the picture generating functions (core) from the modeling functions etc., were discussed. The members of GSPC were very much influenced by this workshop to direct a focussed effort in developing the Core System.

In 1977, the recommendations of the GSPC were published as a status report[14]. In 1978, a whole issue of ACM Computing Surveys (Dec. 1978) was devoted to describing the activities of GSPC and the Core System. The development of functional capabilities and programming considerations were discussed in detail. In the next section a brief review of the Core System will be provided.

Core System Overview

Core System Description

The Core System is based on the criterion of "what is good for most programmers on generally available displays most of the time". It is a rich package with 3D capabilities. A powerful 2D subset is part of the standard. It also has raster graphics extensions. Newman and van Dam[34] discussed the history, design goals, application program structure and modularization of the Core System. A detailed review of the Core System's functional capabilities by Michner and van Dam provides a complete picture of the Core System.

The Core System falls into five functional groups; viz.,

1) Output primitives
2) Viewing
3) Segments
4) Input
5) Control

The Core System has four levels of implementation. The levels are arranged to have increasing capabilities and degree of sophistication. These levels were chosen to provide a reasonable amount of functionality for varied hardware environments and software requirements.

The users object is described by invocations of the output primitive functions. An output function has an output

primitive such as MOVE or DRAW with it's associated attributes. The attributes have a current value that needs to be set by the application.

The viewing transformation selects a portion of the user's world known as the window. The synthetic camera describes the projection of the object on the window, such that it can be displayed on a view surface area known as the viewport. The viewport is specified in normalized device co-ordinates. Two dimensional and three dimensional window descriptions as well as clipping specifications are included.

A segment is said to define an image. The Core System allows for retained and nonretained segments. The minimum Core System application runs as a nonretained segment. The Core System segments can be modified dynamically for visibility, highlighting, detectability and total image transformations of scaling, rotation and translation.

The input is specified through logical input devices. Sampled and event causing classification of input devices have been specified. The input devices are classified into the following: pick, locator, valuator, keyboard and button devices. Each device belongs to either an event causing or sampling class but not both.

The Core System control takes place at various functional areas of the system. Multiple view surface control, inquiry functions and batch mode of segment handling have also been provided.

Status of GSPC Core System

A second status report of GSPC was published in 1979 [33]. Issues such as viewing versus modeling, 2D versus 3D relationships etc., were addressed. Between 1977 and 1979 many implementations of the Core System were underway. Foley et al.,[10] published a Core System implementation. Warner et al., [16] implemented a FORTRAN version known as DIGGRAF. This provided an attempt at language binding for the Core System. A Pascal implementation by Nicol and Kilgour [9] and a University of Pennsylvania package [8] were also among the published implementations.

Many industries have invested considerable man-years in developing the Core System. Towards 1984, the ISO graphics standard, namely the GKS was released for public review. Many industries are opposed to having ISO GKS as well as the current ANSI GKS, primarily due to the investment in Core System already made. A relatively strong letter to the ACM [5] by Joseph to oppose acceptance of GKS as an American standard is an indication of the time and effort invested in the Core System. The SIGGRAPH executive committee meeting of 1985 [2] has adopted a "wait and see" policy before finalizing the rejection or acceptance of the Core System as an ACM standard.

CHAPTER II

GKS

GKS Development

Members from North America, Europe etc., representing graphics standardization institutions in their respective nations convened at Seillac, France in 1976. A proposal to draft an internationally acceptable standard in computer graphics was debated at this workshop. Following this workshop the GSPC took an active interest in the Core System. Efforts to standardize computer graphics practice were underway in Britain (GINO-F), Germany (GKS), Netherlands (IDIGS) etc., around the same time. A strong need for an acceptable international graphics standard was realized at Seillac. Representatives of appropriate committees within GSPC, DIN as well as national standards organizations of other nations jointly formed a committee under ISO in 1977, towards developing an international graphics standard.

In 1979 a meeting in Amsterdam resolved the selection of a work item on which to build the proposed international standard. A 2D candidate was considered as a more readily acceptable item for an international standard at the time. GKS was 2D and was also small. The DIN was ready to sponsor

GKS for an international standards development. The ISO charged the working group WG2 in 1979, to recognize GKS as a work item. GSPC moved it's operation to the American National Standards Institute (ANSI) under the X3H3 (Committee for Graphical Standards) committee to redirect efforts to build the GKS.

GKS - Functional Description

The GKS originally started as a 2D graphics standard. 3D extensions to GKS have been included recently. GKS developers maintained a set of design guidelines as follows:

i) To include capabilities essential for a wide variety of graphics applications ranging from passive output to highly interactive system.

ii) Graphic devices including vector, raster, microfilm recorders, storage tube displays, refresh displays and color displays are to be controlled uniformly.

iii) To keep GKS small and yet cater to the majority of applications.

iv) To be consistent and compatible with existing norms of computer graphics practice. The modules of GKS are to be orthogonal. Orthogonality (of Graphics Systems) is a principle that states that functions or modules of the system should be independent of each other, or the dependency shall be well structured.

The GKS design of user interface is clear, allows for user friendliness and has excellent error handling

specification. GKS allows for total device independence. It can be implemented using existing ANSI standard languages. However, languages which are not as yet accepted as ANSI standard can still implement GKS.

The functional components of GKS can be classified into the following groups based on the GKS function description:

Control Functions

Output Functions

Output Attributes

Workstation Attributes

Transformation Functions

Segment Functions

Input Functions

Metafile Functions

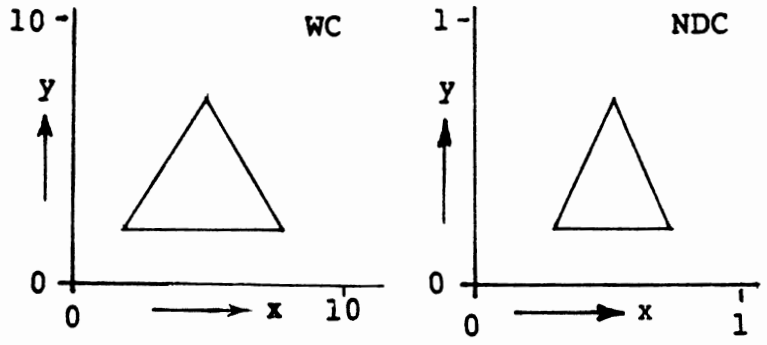
Utility Functions

Inquiry Functions

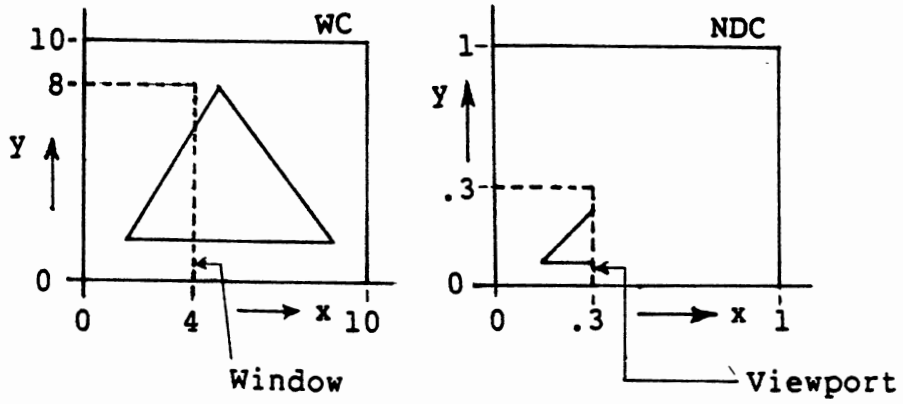
Error Handling

In what follows, each module is briefly described outlining it's principal function. Figure 1 describes graphics terminologies such as workstation, viewport, window and the associated co-ordinate system.

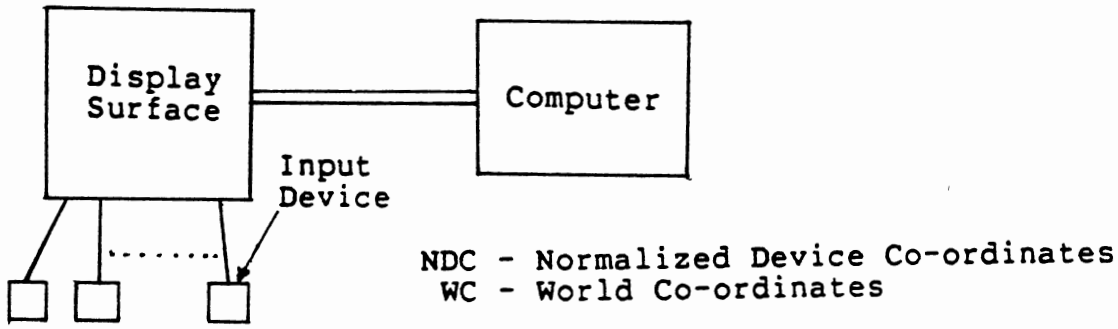
Control Functions - These are a set of functions that initialize GKS environment. These functions provide for proper handling of workstations, segments, escaping to and restoring from device dependent functions and error handling.



(A) World and Normalized Device Co-ordinates



(B) Window and Viewport with Clipping of Image



(C) Workstation

Figure 1. Description of Graphics Terminologies

Output Functions - An image or a picture is described to GKS through invocations of graphic primitive output functions. Each function invoked needs a primitive such as draw or move together with it's set of attributes.

Output Attributes - The output attributes are specifications as to how an output primitive is to be displayed. GKS allows both individual attributes setting as well as bundled attributes. The GKS state list and workstation state lists maintain explicit predefined attribute bundles.

Workstation Attributes - A GKS workstation is one which has a single real or abstract display surface for graphical output and any number of real input devices. Workstation attribute functions specify output attributes in a device-independent manner. The mapping from device-independent specification to device-dependent specification is done by GKS using the workstation state list data. The workstation state list contains information among others screen size and screen units which aid in the mapping. GKS maintains a workstation state list for every open workstation.

Transformation Functions - These functions define the mapping from workstation window to workstation viewport. The workstation window defines a rectangle in world co-ordinates. The transformation functions provide an image on the workstation viewport, which corresponds to the normal projection of the user's world on the workstation window. The workstation viewport is a specified section of the

actual display screen.

Segment Functions - When an image or a picture needs to be modified and redisplayed, or, several instances of the same image needs to be generated, it is convenient to store the image. A set of output functions and their attributes that define an image can be stored as a segment in GKS. A segment can be considered to be an abstract data type where the model is a GKS specification of an image, together with the operations of scaling,translation and rotation. The operations are further qualified using segment visibility, highlighting and priority setting. GKS segments are named for identification.

Input Functions - GKS classifies input devices into six categories.

1) Locator - specifies a position by it's (x,y) value.

2) Pick - identifies a displayed object.

3) Choice - selects from a set of alternatives.

4) Valuator- inputs a value.

5) String - inputs a string of characters.

6) Stroke - inputs a sequence of (x,y) positions.

Sampling and event causing distinctions are made. A set of functions define the input device initialization. Sample and event functions are specified. A set of functions for request of input are also provided.

Metafile Functions - A mechanism for long time storage even after GKS is closed. The metafile stores a GKS transaction in an address format. A GKS metafile is treated

as an output workstation. It has the following characteristics:

1) Output functions are stored if the workstation is active.
2) Attribute functions are stored.
3) Segments are stored if the workstation is active.
4) Geometric data is stored in NDC.
5) Non GKS data maybe written by a special function.

Each metafile is a sequence of items, each of which has the following components:

1) Item type
2) Item data record length
3) Item data record

Utility Functions - These functions aid other GKS functions. Evaluation of transformation matrix etc., are examples of utility functions.

Inquiry Functions - This set of functions retains values corresponding to GKS state, workstation state, segment state etc., which can be used for further processing.

GKS and GSPC Core System

The Core System and GKS were two graphics standards that were developed almost at the same time. The functional similarities between the two standards is largely due to similar design goals. However, differences between the two standards are evident.

The Core System published in 1979 [21] was a 3D system. GKS [28], originally 2D, has 3D extensions included after an initial draft proposed ISO GKS was released. The Core System is exhaustive and detailed. GKS is small, yet functionally complete. Attribute handling in GKS was originally inadequate to specify individual attributes. GKS originally used bundled attributes. The Core System used individual attribute specification. Considerable effort put forth by members of GSPC led to the inclusion of individual attributes in GKS. The current version of GKS provides for both bundled and individual attributes. The workstation concept was a unique contribution of GKS. A GKS workstation is based on the concept of abstract graphical workstations. An abstract graphical workstation provides a logical interface through which the application program controls physical devices. A clear relation between output and input devices was lacking in the Core System. The Core System of 1979 [21] lacked a workstation definition. The Core System did not originally provide for the storage of application programs. The GKS metafile on the other hand readily provides for a mechanism to store application programs using a metacode format.

Current Status of GKS

The ANSI used the Core System as its groundwork for contribution towards GKS. GKS uses metafile to store graphics instructions in an address format. This format is

a GKS standard and as such is both system and device independent. Hence, an application program is easily ported across systems. The ISO group had meetings in Europe and USA to resolve issues pertaining to input handling, multiple window to viewport transformations and bundled attributes [24]. Text and segment handling were fine tuned. A draft proposed GKS standard was published in 1981 [26]. In 1983 a draft ISO GKS was released. ANSI released ANSI GKS [28] on July 1, 1984.

The ANSI GKS is currently under public review. It's use is widespread in Europe. It is evident from the minutes of the SIGGRAPH meetings that GKS will be passed as an acceptable ANSI standard.

CHAPTER III

LITERATURE SURVEY

Simultaneously Active GKS Workstations

The ANSI GKS[28], while specifying the functional details of each GKS function vividly, does not address implementation aspects of the concepts. This is, however, deliberate. A standards specification need to be system independent. Specifically GKS does not specify a design methodology or guideline for workstation design. The subject matter of this thesis is to consider an efficient design methodology for GKS workstations. As a first step it would be necessary to examine GKS workstations in greater detail.

At this point it would be appropriate to recall and restate the GKS workstation concept. A workstation in GKS consists of a single display surface with its associated set of input peripherals attached to a single line to the computer. Before a display can be made on the screen or input received for interaction, GKS should first open a given workstation. Several workstations can be open at the same time. However, prior to actual input from or output to a workstation, it (workstation) needs to be activated. This will allow the user to control the portions of the picture

that need to be displayed at a given workstation. GKS directs output to all active workstations.

In the next section, a review of workstation designs adopted in the past will be considered. Following this, a unique design method for GKS workstations under UNIX will be outlined.

Literature Review

In this section attention is devoted to previous work carried out in the area of workstation design. When a multiple GKS workstation environment is considered, it is possible to have a network of GKS workstations. This situation will call for user protocols. Common network protocol methods will be considered for completeness of survey.

As the graphics standards were developed there have been useful extensions to them. A raster graphics extension to the Core System [27] were presented by Foley. This contribution was adopted by GKS as well. A three dimensional extension to GKS is currently being finalized. In order for GKS to be an acceptable ANSI standard, GKS may have to undergo some modifications and accept extensions in the future. Since in the future graphics will involve highly interactive methods, GKS may have to include interactive components, as well as network extensions. The need for interactive components and extensions to a network approach are seen as essential in the near future.

Simmons has implemented a Minimal GKS [4]. The Minimal GKS is one that comprises the lowest output level and level 'a' input of GKS. This was developed at Sandia Laboratories. The implementation is under UNIX and coded using C. The Minimal GKS [4] uses a virtual device interface (VDI) that lies between the device drivers and the application. Each device driver that lies under the VDI is responsible for maintaining the devices under it. The workstation state list associated with each workstation is maintained by the appropriate VDI. A subroutine or data interface is suggested as a method of implementation. Simmons describes the implementation of simultaneous workstations using the C function pointers. The devices are obligated to be under the control of the VDI. The GKS functions need not pass the workstation identifier across the VDI. A multiple simultaneous workstation has been implemented with the VDI and C function pointer approach.

Guttman and Weiss [29] implemented a device independent decentralized graphics system for GKS. The GKS output is in the form of a pseudo-code. A device supervisor performs the decoding, subsequent device output and input handling. A salient feature of this system is the decentralization of certain GKS support functions. Clipping, zooming, rotation, scrolling, viewing etc., are handled at the device. This causes reduction of code in the main memory and processing time of the host computer. A principal disadvantage with this system, not mentioned by the authors, is that the

design assumes sophisticated devices. GKS is device independent and thus cannot leave some processing at the device end.

DIGGRAF [16] which is a Core System implementation uses subroutine interface between the device and the VDI. Kellner at al., [34] implemented a Core System in a multiuser environment. The virtual device concept is used, with device drivers being dynamically linked as need arises.

Interactive methods in workstation design has been the source of attention in designing the emerging Programmer's Hierarchical Interface to Graphics Systems (PHIGS). UIMS [6] (User Interface Management system) describes the user interaction methods on a workstation. User-to-application program interaction is described. Mark Green [11] has addressed the methodology for user-to-application interface, with no attention being given to, user-to-user communication. The protocols necessary for user-to-user communication can parallel that of communication protocols among processes. Datagrams and virtual circuits are standard methodologies for process communication. A packet is a prescribed number of bytes defined as a unit for network data transfer purposes. Typical values are 64-byte 128-byte etc. Packets are transmitted via datagrams or virtual circuits. In a datagram packets are transported as isolated units addressed to a specific destination. The receiver may or may not receive the data in the same order. In a virtual circuit the network provides a channel in which

the packets are sent error free and arrive in the precise order in which they were sent. A user-to-user protocol in a workstation environment needs to use the process communication methods and adapt them to graphic workstation environment. The need to establish a standard practice for user communication in a GKS environment may be necessary in an industrial use of GKS workstations.

CHAPTER IV

GKS WORKSTATION DESIGN AND SUBSET IMPLEMENTATION

Multiple GKS Workstations

A survey of past work in the area of workstation design for GKS and other graphics systems reviewed in the previous chapter indicates that a layer of software needs to be maintained between the device independent portions and the device dependent code. The VDI and the device drivers under it are seen as essential. Some designs use a decentralization of certain GKS functions. This may have its advantage in terms of host computer's memory and processor time saving. However GKS has to be device independent. Hence central processing is seen as more desirable. Minimal GKS [14] maintains a workstation state list at the VDI level for that workstation. This complicates the use of GKS inquiry functions.

GKS stipulates that the output functions and attributes are to be executed in all active workstations. The concept of simultaneousness is used in the sense that at a given time many workstations are to be serviced with the same display.

A close study of GKS indicates that the output on active workstations can proceed in parallel. In a simple GKS workstation design, the parallel paths exist between the VDI and the device drivers.

Based on the workstation design strategy, parallelism can exist at more than one phase of a workstation layout. In the next section a unique GKS workstation design will be outlined, that utilizes the parallelism inherent in the workstation specification.

Multiple Parallel GKS Workstations

Consider an abstract display surface which is partitioned into many viewing sections. If a GKS application has many active workstations with an output component, then each output display surface maybe conceived as one viewport of the abstract display surface. Each such viewport would belong to at most one type of workstation. A given type of workstation may have one or more viewports on the abstract display surface subscribing to its type. Thus all viewports belonging to a given type need at most one device driver. Let this type of a workstation be called a Virtual Workstation (VWS). Figure 2 describes a VWS.

It is clear from the description of VWS that parallel processing is possible at two points in its layout.
i) The VDI provides as output, the pseudo-code corresponding to GKS output functions. The VDI has the set of active device drivers under it, which it can execute in parallel.

ii) The VWS has a single device driver to serve all viewports on the display surface subscribing to a given type of GKS workstation. Hence the device drivers can service the viewports under them in parallel.

The aspect of parallel execution of GKS workstations has not been addressed in the past. A design method that exploits the parallel execution of the phases of a GKS workstation will be superior and faster. A conceptual outline of the design is as follows:

- Each GKS output function needs to display the graphical information to a single Virtual Workstation.

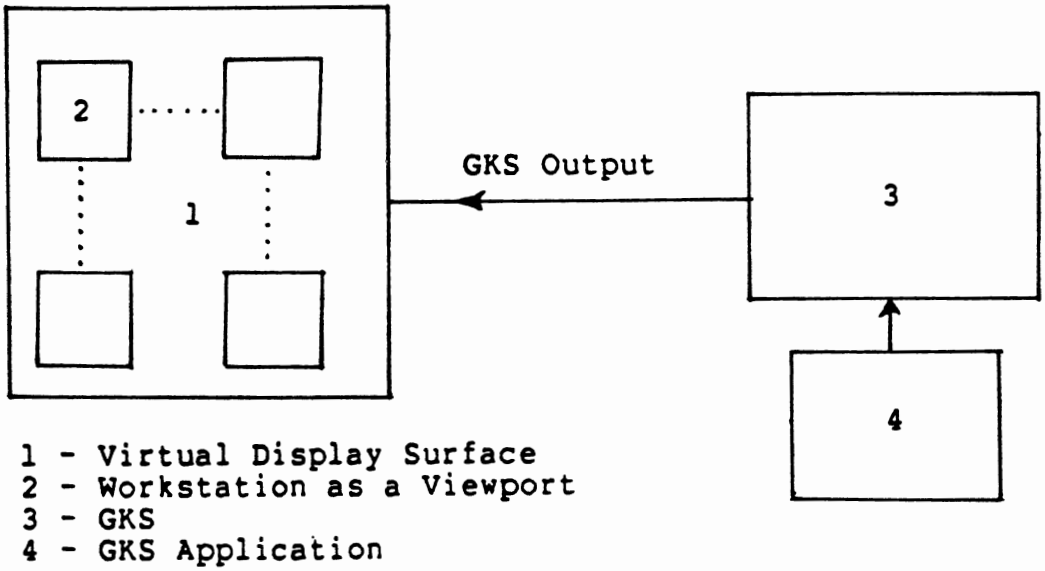
- Each real GKS workstation, having a display component, will occupy a viewport on the Virtual workstation display surface.

- The VWS will group the virtual viewports according to their GKS types.

- Each device driver will receive a copy of the pseudocode output from the VDI.

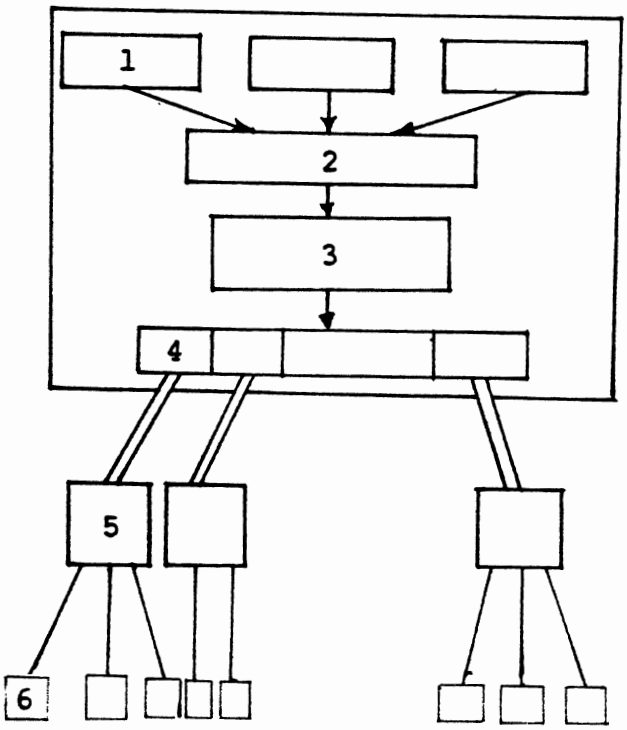
- The device driver will display the graphical output on all virtual viewports under it, through device calls.

Figure 3 provides a block diagram representation of the above design. Figure 4 provides a block diagram representation of a workstation design, if the design included a subroutine interface to the device drivers. Using figure 3, a brief description of the implementation of the above design is provided. Each GKS output function,



1 - Virtual Display Surface
2 - Workstation as a Viewport
3 - GKS
4 - GKS Application

Figure 2. Virtual Workstation



1 - GKS Output
2 - Global Data Interface
3 - Virtual Device Interface
4 - Device Driver Process List
5 - Device Drivers
6 - Output Device

Figure 3. Virtual Workstation Design

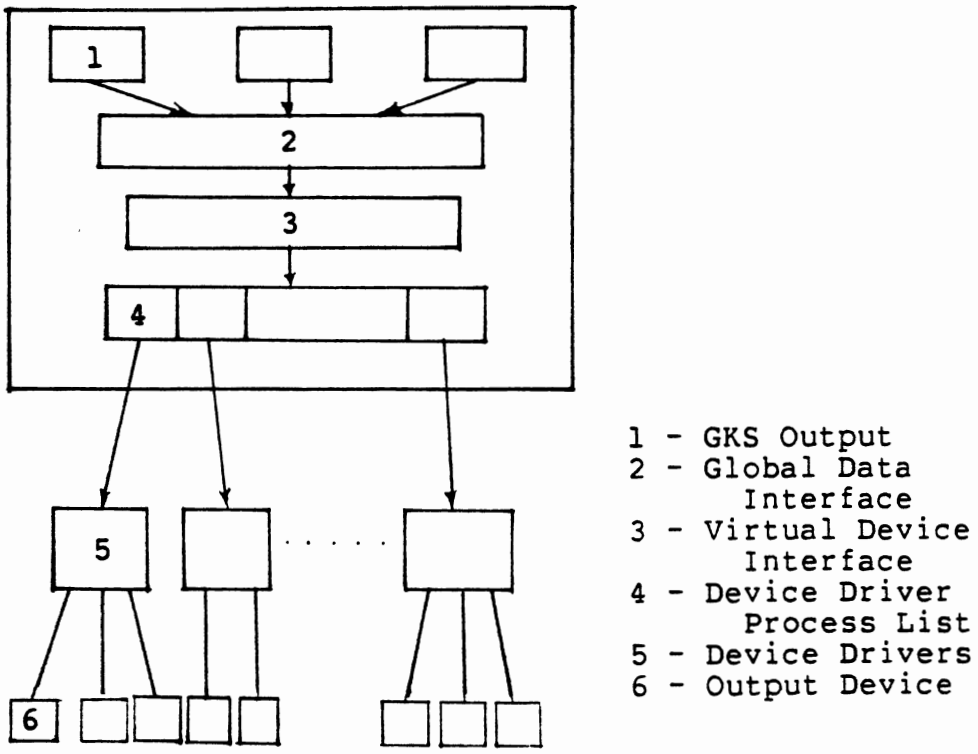


Figure 4. Virtual Workstation Design - Subroutine Interface

OPCODE	MOVE
Parameter 1	200
Parameter 2	200
Parameter 3	---
Parameter 4	---

Figure 5. Global Data Interface: An Example

control function and attribute function uses a global work area known as the global data interface (gdi). In the gdi the operation (an op-code) and the necessary parameters are written. The Virtual Device Interface (VDI) reads the gdi and buffers the data in an appropriate form (buffers all data in data-type character). The VDI has access to the set of device driver process list. Each device driver that has at least one active workstation under it exists as a child process directly under the VDI. The use of UNIX mechanisms of fork, pipe and the call "execlp" are used to achieve this goal. The call "execlp" is used to execute another program (the device driver) without returning. This call needs the full path name to the executable file. The UNIX system call fork, splits the program into two copies both of which continue to run. The UNIX system call pipe, creates an I/O mechanism. A call to pipe returns 2 file descriptors. The file descriptors at the device driver process end, are further duplicated to act as standard input and standard output. The VDI which belongs to the parent process, writes the GKS data buffer to the file descriptors resident at it's end. Hence with the help of the above UNIX mechanisms it is possible to allow the device drivers to reside as independent executable code. The fork mechanism allows multiple workstations to execute simultaneously, while the GKS application program continues to execute uninterrupted.

Minimum GKS Subset

In order to demonstrate the workstation design mentioned in section 4.1 adequately, a minimum subset of GKS is implemented under UNIX. The language used is C. The GKS data types and abstract functions are bound to C. The subset developed resides at level m (Minimal Output) of ANSI GKS [28]. The following capabilities reside in the minimum subset developed:

i) subset of control
ii) polyline output
iii) subset of attributes
iv) no bundled attributes
v) no color representation modification
vi) multiple workstations with output capabilities.
vii) normalization transformation 0 by default. Only one current normalization transformation is allowed.

The minimum subset deviates from level m of GKS . As required by GKS, the implementation allows the system to be easily upgraded from a given level to a higher level and vice versa. The principal deviation is the inclusion of multiple active workstations at the minimum level. This is justified since the workstation design allows an easy implementation of multiple active workstations. The GKS restriction of a single workstation at a time, under level m, needs to be relaxed for systems such as UNIX.

A set of GKS functions that reside at level m of ANSI GKS were implemented under UNIX on the Perkin Elmer mini

computer. Appendix B, which is a user's guide for the current implementation of level m GKS lists the above set of functions. Explanations as to their use is also provided. Refer Appendix A and Appendix C for further details regarding the set of GKS functions under discussion. The language used for coding was C.

Merits of the Design

This design was the result of experiments and simulations conducted at the Computing and Information Sciences Department at OSU, Stillwater, OK during the course of implementing GKS.

- The concept of treating each device driver as a process and the output device as a file under that process is unique to this design of workstation.

- Each type of device driver needs only one device driver process that can serve any number of devices of that type under it.

- The design also allows any number of device drivers for the same type of device. However the system administrator should recognize them as separate GKS workstation types.

- Since the device driver process is accessed through its full path name in the UNIX tree of directories, a device driver may reside even on a remote UNIX network node with its associated devices. However, the workstation identifier that specifies the tele-type terminal (tty) number of the

display device needs to be mapped to the local device number in a network situation.

- The GKS output functions and attribute functions need to consider only a single VWS. This allows for easy growth of GKS levels.

Applications

The design suggested is applicable in a heterogeneous or homogeneous graphic device environment. It suggests itself for multiple simultaneous display. Where the addition of new devices are frequent, the device drivers can be written without interrupting the GKS system. The design is suitable for workstations located in remote network nodes. This design allows a single user to subscribe to multiple GKS applications, since the device drivers enjoy total independence (the device drivers are spawned processes for GKS).

CHAPTER V

RESULTS, CONCLUSIONS AND FUTURE WORK

Results

A minimum GKS was implemented comprising a subset of control functions, a subset of output attributes and a polyline output facility. The implementation is capable of serving multiple parallel output workstations. Workstation transformations have been included. The implementation is at level m of ANSI GKS with deviations in workstation handling. GKS level m allows for a single open workstation at a time. The implementation under discussion is capable of multiple open workstations and multiple active workstations. The implementation supports regis graphics and HP 7470a flatbed plotter (hardcopy). The implementation under UNIX on a Perkin- Elmer mini computer was carried out at the Computing and Information Sciences Laboratories of OSU, Stillwater, OK. The entire implementation was coded in C.

Discussion

UNIX is a highly suitable environment for the implementation of GKS. The UNIX mechanisms of pipe and fork were used to effect a simulation of parallel workstations.

The UNIX command called execlp was used to spawn a device driver process. This allows the device drivers to be totally independent of GKS kernel or the core.

The workstation design method adopted is applicable even at level m of GKS. GKS [28] stipulates that at most one workstation may be open at a given time when it is run at level m. In view of the workstation design adopted in this implementation, this restriction is unnecessary. It is suggested that future specification of GKS remove this restriction.

The parallel display of all active devices that are effected under this design is not strictly parallel. In reality, the spawned processes and the GKS application execute on a time sharing basis(UNIX). The spawned processes (child processes) have a lower priority than the parent process. The GKS application is the parent process and the device drivers are the child processes. Refer to fig (6) for priorities of the processes during the execution of a demonstration application. It was observed that when the parent had a priority of 30, the child had a priority of 26. However, all child processes had the same priority of 26. Lower the value higher is the priority of the process. Hence, the snapshot of fig (6) indicates that the child enjoys a higher priority than the parent. This is possible if at the instance a snapshot of the systems process state is taken, the parent is in a ready or wait state, while the child is running. In general if both the parent and the

```
F S UID   PID  PPID CPU PRI  NICE  ADDR  SZ  WCHAN TTY  TIME CMD
3 S   0     0     0 112   0    20    F2   1  16C98 ?   1316:55 swapper
1 S   0     1     0   0  30    20   18C   1  291F0 ?     1:30 /etc/init
1 S 553 16166     1   0  30    20   2F8   3  29214 20    0:07 -csh
1 S 553 16468     1   0  28    20   352   3  17988 24    0:06 -csh
1 S   0    37     1   0  40    20   101   1 FF0000 ?     8:50 /etc/update
1 S   0    39     1   0  40    20    F4   2 FF0000 ?     3:12 /etc/cron
0 S 183    43     1   0  40    20   41E   9 FF0000 con   2:53 deliver -b
                                                        -clist,local,uucp
1 S 183    44     1   0  40    20   399   9 FF0000 con   3:30 deliver -b
                                                        -cxokscc1,xokseel,x
1 S   0 16490     1   0  28    20   185   1  17250 2     0:00 - z
0 S   0 12410     1   0  28    20   596   1  1773C ?     0:00 /etc/init
0 S   0    52     1   0  28    20   402   1  172F8 ?     0:00 /etc/init
1 S   0 16488     1   0  28    20   13D   1  172A4 ?     0:00 /etc/init
1 S   0 16180     1   0  28    20   11A   1  173A0 ?     0:00 /etc/init
1 S 595 13344     1   0  30    20   2C4   3  293A0 7     0:07 -csh
0 S 579 16499     1   0  30    20   64A   7  293C4 18    0:21 -vish
0 S   0    57     1   0  28    20   726   1  174F0 ?     0:00 /etc/init
0 S   0    58     1   0  28    20   742   1  17544 ?     0:00 /etc/init
0 S   0 13413     1   0  28    20   60E   1  17598 ?     0:00 /etc/init
0 S   0  9025     1   0  28    20   5F2   1  175EC ?     0:00 /etc/init
1 S 553 16465     1   0  28    20   220   3  17694 15    0:06 -csh
1 S 538 16088     1   0  30    20   23D   3  2949C 16    0:09 -csh
0 S   0 12497     1   0  28    20   796   1  177E4 ?     0:00 /etc/init
1 S   0 16226     1   0  28    20   152   1  1734C ?     0:00 /etc/init
1 R   0 16251 16187  33  52    20   1A4   7         9     2:47 /usr/lib/
                                                        uucp/g.uucico -rl -su
1 S   1 16307 13344  88  29    20   174   3  1740C 7     3:42 robots
0 S   0    67     1   0  28    20   36A   1  1788C ?     0:00 /etc/init
0 S   0    68     1   0  28    20   386   1  178E0 ?     0:00 /etc/init
0 S   0    69     1   0  28    20   62E   1  17934 ?     0:00 /etc/init
0 S   0 10091     1   0  28    20   DB2   1  17448 ?     0:00 /etc/init
1 S   0 14793     1   0  28    20   137   1  171A8 con   0:00 - 6
1 S   0 16187 16186   0  30    20   168   2  29604 ?     0:28 sh /usr/
                                                        lib/cron/cron-hourly
0 S 579 16532 16499   0  30    20   912   9  29628 18    0:02 send  vasoll
1 S   0 16184     1   0  30    20   12A   2  2964C ?     0:00 sh -c sh /
                                                         usr/lib/cron/cron-ho
1 S   0 16186 16184   0  30    20   15D   2  29670 ?     0:00 sh /usr/lib
                                                        /cron/cron-hourly
1 S 553 16589 16562   0  26    20   291   3  1D842 20    0:01 regpl
0 S 579 16534 16532   0  26    20   7B2  11  1E38A 18    0:02 submit
1 S 553 16562 16166   2  30    20   3DB   5  296DC 20    0:04 prog
1 S 553 16590 16562   3  26    20   1DC   3  1EB42 20    0:00 hppl
1 S 579 16543 16532  12  28    20   31B   5  17790 18    0:07 vi /u/gregg
                                                            /drft.016532
1 S 538 16384 16088   6  28    20   1FC   5  176E8 16    0:31 vi insert.c
1 S 553 16591 16562   4  30    20   310   2  2976C 20    0:00 sh -c ps
                                                          -alx > prifile
1 R 553 16592 16591 124  57    20   108   3        20    0:03 ps -alx

    PRI - Process Priority
    CMD - Command or Process
   hppl - HP Device Driver Process
  regpl - Regis Device Driver Process
   prog - GKS Application Process
```

Figure 6. Snapshot of Process Priorities

child are running, the parent will run with a higher priority. In the current environment the application programmer has no privilege to increase the priority of the child process. It is suggested that a stand alone system with root as the user be used while utilizing this design. A privileged user may also use this design by equalizing the priorities of the parent and child processes.

The UNIX tool of pipe imposes a restriction under this design. When sending the GKS output and attribute code through the pipe to the device drivers, a buffer in excess of 4K tends to introduce the "broken pipe" error. Should more than 4096 bytes be necessary in any pipe among communicating processes, deadlock will occur indicated by "broken pipe". Handshaking protocols maybe used to avoid this. The current implementation transfers GKS output code when the buffer is in excess of 2K bytes.

Scope for Further Work

The current implementation needs additional functions to provide diagnostics and error messages. Error recovery needs attention. The implementation has the necessary data structures and data types to build additional levels.

Each device driver maintains the set of active devices corresponding to the set of active output workstations for a given type. The current implementation executes the display on these devices sequentially. Note that once a device is activated, the device driver corresponding to that device

maintains the device list. The fork mechanism can be used to execute the display on all devices under a given driver, in parallel.

Since the number of workstations anticipated at the installation is less than five, the implementation used simple linear arrays and linked lists for the majority of the operations. Sequential searching was employed where searching was necessary. However, when the number of workstations increase hashing, binary search and stack operations may be necessary. The abstract data types to support these models need to be constructed.

It is possible to have a network of GKS output workstations subscribing to a single GKS application. For example, a design office may want to display the design on various sites needing the graphical information. This will not allow the receiver to interact with the application programmer directly. Hence a layer consisting of user level interaction during the course of a GKS transaction maybe necessary. The necessary protocols and network layer development are seen as useful extensions to this project.

SELECTED BIBLIOGRAPHY

1. Thomas Wright, "An Update on GKS: The Final Changes," ACM Computer Graphics, vol. 19, no. 1, January 1985.

2. "Minutes from the 6/16/84 SIGGRAPH Executive Committee Meeting," ACM Computer Graphics, vol. 19, no. 1, January 1985.

3. Deborah U. Cahn, Albert C. Yen,"A Device Independent Graphics System," ACM SIGGRAPH, vol. 17, no. 3, July 1983, pp 167-173.

4. Randall W. Simmons, "Minimal GKS," ACM SIGGRAPH, vol. 17, no. 3, July 1983, pp 183-189.

5. John J. Joseph, "Letter on Core System Standardization," ACM Computer Graphics, vol. 18, no. 2, May 1984.

6. "Graphical Input Interaction Technique Workshop Summary," ACM Computer Graphics, vol. 17, no. 1, January 1983, pp 5-30.

7. Lansing Hatfield, "GKS and the Alphabet Soup of Graphics Standards," ACM SIGGRAPH, vol. 16, no. 2, June 1982, pp 161-162.

8. Fredrick P. Stulk, Brian F. Saunders, Paul M. Slayton and Norman I. Badler, "Overview of the Univ. of Penn. Core System Standard Graphics Package Implementation," ACM Computer Graphics, vol. 16, no. 2, June 1982, pp 177- 186.

9. Nicol, C.J., and Kilgour, A.C., "A pascal Implementation of the GSPC Core Graphics Package," vol. 15, no. 4, December 1981, pp 327-335.

10. James D. Foley, and Patricia A. Wenner, "The George Washington University Core System Implementation," ACM SIGGRAPH, vol. 15, no.3, August 1981.

11. Mark Green, "A methodology for the Specification of Graphical User Interface," ACM SIGGRAPH, vol. 15, no. 3, August 1981.

12. Alan Freiden, "A two Dimensional Level 2 Core System for the Apple II," ACM Computer Graphics, vol. 14, no. 4, March 1981.

13. Deborah U. Cahn, Nancy E. Johnston, and William E. Johnston, "A Response to the 1977 Core Graphics System," ACM SIGGRAPH, vol. 13, no. 2, August 1979.

14. "Status Report of the Graphics Standard Planning Committee," Computer Graphics, vol. 11, no. 3, Fall 1977.

15. Theodore N. Reed, "A Metafile for Efficient Sequential and Random Display of Graphics", Computer Graphics, vol. 16, no. 4, July 1982.

16. James R. Warner, Margaret A. Polisher, and Robert N. Kopolow, "DIGGRAF - A FORTRAN Implementation of the Proposed GSPC Standard," ACM SIGGRAPH, vol. 12, no. 3, August 1978.

17. William M. Newman, and Andries van Dam, "Recent Efforts Towards Graphics Standardization," Computing Surveys, vol. 10, no. 4, December 1978.

18. James C. Michner, and Andries van Dam, "Functional Overview of the Core System with Glossary," Computing Surveys, vol. 10, no. 4, December 1978.

19. Daniel R. Bergeron, Peter R. Bono, and James D. Foley, "Graphics Programming Using the Core System," Computing Surveys, vol. 10, no. 4, December 1978.

20. James C. Michener, and James D. Foley, "Some Major Issues in the Design of the Core Graphics System," Computing Surveys, vol. 10, no. 4, December 1978.

21. "Status Report of the Graphics Standard Planning Committee," ACM Computer Graphics, vol. 13, no. 3, 1979.

22. "Graphical Kernel System (GKS) Version 6.6," International Standards Organization, ISO/TC97/SC5/WG2, May 1981.

23. Rosenthal, D.S.H., et. al., "The Detailed Semantics of Graphics Input Devices," Computer Graphics, vol. 16, no. 3, 1982, pp 33-38.

24. Introduction to The Graphical Kernel System GKS. Hopgood, F.R.A., Duce, D.A, Gallop, J.R, and Sutcliffe, D.C., Academic Press 1982.

25. "ISO/DIS 7942 Information Processing - Graphical Kernel System (GKS). Functional Description : GKS Version 7.2," ISO/TC97/SC5/WG2/N163, 1982.

26. Bono P.R., et. al., "GKS - The First Graphics Standard," IEEE Computer Graphics and Applications, vol. 2, no. 5, 1982, pp 9-23.

27. Foley, et. al., "Some Raster Graphics Extensions to the Core System," ACM SIGGRAPH, vol. 13, no. 2, August 1979.

28. "Draft Proposed ANSI Graphical Kernel System," ANSI X3H3, July 1, 1984.

29. Guttmann, H and Weiss, J., "Device Independent and decentralized Graphics System," ACM SIGGRAPH, vol. 13, no. 4, February 1980, pp 288-302.

30. Keith A. Lantz, and William I. Nowicki, "Structured Graphics for Distributed Systems," ACM Transactions on Graphics, vol. 3, no. 1, January 1984, pp 23-51.

31. James H. Clark, and Tom Davis, "Work Station unites Real-time Graphics with UNIX, ETHERNET," Electronics, October 20, 1983.

32. Sutherland, I.E., "SKETCHPAD : A Man-Machine Graphical Communication System," SJCC 1963, Spartan Books, Baltimore, MD, USA.

33. "Status Report of the Graphics Standards Planning Committee," Computer Graphics, vol. 13, no. 3, August 1979.

APPENDIX A

GKS - LEVEL m IMPLEMENTATION

The minimum subset of GKS function at level m of ANSI draft proposed GKS [28] is being presented. The list contains the set of function names and their purpose. Where appropriate, the module to which the function belongs is identified.

A function may fall into one of the following GKS functional classifications:

i) Control

ii) Attributes

iii) Output

iv) Utility

v) Workstation Transformation

vi) Virtual Device Interface (not a GKS classification)

vii) Transformation

ix) Data (data structures, data types etc.,)

x) Other

FUNCTION NAME : defs

MODULE TYPE : data

USAGE : N/A

INPUT PARAMETER : N/A

RETURNED VALUE : N/A

BRIEF DESCRIPTION : Values for named constants. Some of these specify the upper limit for allowable aspects of certain GKS arrays, lists etc.,.

FUNCTION NAME : atdecl.h

MODULE TYPE : data

USAGE : N/A

INPUT PARAMETER : N/A

RETURNED VALUE : N/A

BRIEF DESCRIPTION : The attributes specify attribute names and values associated with them. The attributes concern GKS state lists, workstation state lists and segment state lists.

FUNCTION NAME : atexdecl.h

MODULE TYPE : data

USAGE : N/A

INPUT PARAMETER : N/A

RETURNED VALUE : N/A

BRIEF DESCRIPTION : These are external definitions of the attributes declared in atdecl.h. Attribute are declared global.

FUNCTION NAME : decl.h

MODULE TYPE : data

USAGE : N/A

INPUT PARAMETER : N/A

RETURNED VALUE : N/A

BRIEF DESCRIPTION : The data types necessary for operating states, GKS statelist, workstation statelist and global utility arrays are declared as C type definitions. In addition, global flags necessary for debugging utility, virtual workstation buffer, display file and virtual workstation's data types are declared.

FUNCTION NAME : extdecl.h

MODULE TYPE : data

USAGE : N/A

INPUT PARAMETER : N/A

RETURNED VALUE : N/A

BRIEF DESCRIPTION : Contains the external declarations for variables and arrays needed for GKS operating state, GKS description table and GKS state list.

FUNCTION NAME : incfiles.h

MODULE TYPE : other

USAGE : as C include statement

INPUT PARAMETER : N/A

RETURNED VALUE : N/A

BRIEF DESCRIPTION : The set of GKS declarations and external definitions needed are grouped into a set of C include statements in order to ease compilation using the UNIX make facility.

FUNCTION NAME : types.h

MODULE TYPE : other

USAGE : as C include statement

INPUT PARAMETER : N/A

RETURNED VALUE : N/A

BRIEF DESCRIPTION : These are GKS data types that need to be declared with each GKS function that need type definitions. All data types are hence clustered. All GKS functions that use any of these data types need to include this file. Also, any new type definition included in the declaration file needs an entry in the types file.

FUNCTION NAME : GOPKS

MODULE TYPE : control

USAGE : GOPKS(ERRFIL)

INPUT PARAMETER : ERRFIL - a pointer to an error file.

RETURNED VALUE : none

BRIEF DESCRIPTION : This function is the first call to GKS. It checks for any errors and if none, initializes GKS and opens it for GKS application. The initialization includes setting the proper level. IT initializes the GKS description table and sets the default current attributes. Performs initialization of utility arrays, lists etc.,.

FUNCTION NAME : OPEN_WKSTN

MODULE TYPE : control

USAGE : OPEN_WKSTN
(wkst_id,id_connect,typ_wkstn)

INPUT PARAMETER : wkst_id - int : workstation identifier. For output workstation it is the tty number of the device.
id_connect - int : channel for this workstation. This implementation uses the value 1 uniformly.
typ_wkstn - int : workstation type. Refer GKS system manual of the installation for appropriate mapping.

RETURNED VALUE : return as exit where necessary. Value none.

BRIEF DESCRIPTION : To initialize a workstation identified by the workstation identifier. To make an entry in the list of open workstations. If the specified type does not have an entry in the workstation statelist, then, an entry is made and also initialized.

FUNCTION NAME : INIT_WKSTN

MODULE TYPE : control

USAGE : INIT_WKSTN (i)

INPUT PARAMETER : i - int : workstation identifier.

RETURNED VALUE : return at points where necessary. value none.

BRIEF DESCRIPTION : This function is called by OPEN_WKSTN if the workstation type has no entry in the workstation state list. The data required to initialize the given type of workstation will be maintained in an external file.

FUNCTION NAME : FILL_PTR

MODULE TYPE : control

USAGE : FILL_PTR(prev,fd)

INPUT PARAMETER : prev - SET : an allocated record of type SET
fd - FILE * : a pointer to an external file.

RETURNED VALUE : none

BRIEF DESCRIPTION : This function reads data from a file whose pointer is fd and records the data into the record prev. It inserts the record into the list to which prev belongs. The list mentioned is a member of the workstation state list.

FUNCTION NAME : WKSTLIST_INIT

MODULE TYPE : control

USAGE : WKSTLIST_INIT(i)

INPUT PARAMETER : i - int : workstation identifier.

RETURNED VALUE : index of workstation pointer in an array of pointers.

BRIEF DESCRIPTION : To insert a new workstation state list if one does not already exist for the workstation specified.

FUNCTION NAME : WK_ST_ALL_WKSTN

MODULE TYPE : control

USAGE : WK_ST_ALL_WKSTN()

INPUT PARAMETER : none

RETURNED VALUE : newptr - PT_MAS_WKSTLIST : a pointer to record of workstation state list.

BRIEF DESCRIPTION : Allocates the data structures necessary for a workstation statelist and links them appropriately.

FUNCTION NAME : itoa

MODULE TYPE : other

USAGE : itoa(n)

INPUT PARAMETER : n - int :

RETURNED VALUE : char pointer

BRIEF DESCRIPTION : Converts an integer to an equivalent character string.

FUNCTION NAME : ermsg

MODULE TYPE : utility

USAGE : ermsg (i)

INPUT PARAMETER : i - int : error number

RETURNED VALUE : none

BRIEF DESCRIPTION : Prints the error message corresponding to the number.

FUNCTION NAME : act_wkstn

MODULE TYPE : control

USAGE : act_wkstn(i)

INPUT PARAMETER : i - int : workstation identifier.

RETURNED VALUE : none

BRIEF DESCRIPTION : If the specified workstation is already open, this function activates the workstation for graphical I/O. If the device driver process corresponding to the type of this workstation is not already activated, necessary steps to activate the device driver is also done. The activation is written to the global data interface (gdi) and executed on the VWS.

FUNCTION NAME : IN_WKST

MODULE TYPE : control

USAGE : IN_WKST(j)

INPUT PARAMETER : j - int : workstation identifier.

RETURNED VALUE : none

BRIEF DESCRIPTION : To make an entry of the workstation into the set of active workstations. If the device driver does not exist for the type to which this workstation belongs, then an appropriate routine is called.

FUNCTION NAME : ins_pr

MODULE TYPE : control

USAGE : ins_pr(k)

INPUT PARAMETER : k - int : workstation type

RETURNED VALUE : none

BRIEF DESCRIPTION : For the given workstation type a pipe is opened for I/O to and from a device driver process. The routine uses fork and execlp to spawn the device driver process.

k = 1 regis

k = 2 hp 7470a

FUNCTION NAME : FN_WKTYP

MODULE TYPE : control

USAGE : FN_WKTYP (k)

INPUT PARAMETER : k - int : workstation identifier

RETURNED VALUE : int - workstation type for workstation k. -1 if error.

BRIEF DESCRIPTION : Given the workstation identifier, return the workstation type.

FUNCTION NAME : FR_WKST

MODULE TYPE : control

USAGE : FR_WKST ()

INPUT PARAMETER : none

RETURNED VALUE : FALSE if list is not empty. NEGATIVE if empty.

BRIEF DESCRIPTION : Returns 0 if set of active workstations is non empty. returns -1 if empty.

FUNCTION NAME : NE_WKST

MODULE TYPE : control

USAGE : NE_WKST(i)

INPUT PARAMETER : i - int : index of the location of an active workstation in the array of active workstations.

RETURNED VALUE : int - index of next active workstation in the list.

BRIEF DESCRIPTION : Next operation on the list of active workstations.

FUNCTION NAME : sel_ntran_num

MODULE TYPE : transformation

USAGE : sel_ntran_num (ntrans)

INPUT PARAMETER : ntrans - int : normalization transformation number

RETURNED VALUE : none

BRIEF DESCRIPTION : Sets the global current normalization transformation number to the value ntrans.

FUNCTION NAME : set_viewport

MODULE TYPE : workstation transformation

USAGE : set_viewport (num,xvp_min,xvp_max,yvp_min,yvp_max)

INPUT PARAMETER : num - normalization transformation number

xvp_min - float : x minimum of viewport

xvp_max - float : x maximum of viewport

yvp_min - float : y minimum of viewport

yvp_max - float : y maximum of viewport

RETURNED VALUE : none

BRIEF DESCRIPTION : Enters the viewport coordinates in the appropriate location of the list of normalization transformations.

FUNCTION NAME : set_window

MODULE TYPE : workstation transformation

USAGE : set_window
(num,xw_min,xw_max,yw_min,yw_max)

INPUT PARAMETER : num - int : normalization
transformation number.

xw_min - float : x minimum of the window

xw_max - float : x maximum of the window

yw_min - float : y minimum of the window

yw_max - float : y maximum of the window

RETURNED VALUE : none

BRIEF DESCRIPTION : Allocates space for the normalization transformation in the list of normalization transformations and enters the values for the window dimensions. Also sets the default viewport of unity.

FUNCTION NAME : polyline

MODULE TYPE : output

USAGE : polyline (n,x_array,y_array)

INPUT PARAMETER : n - int : number of points
x_array - float : array of x coordinates
y_array - float : array of y coordinates

RETURNED VALUE : none

BRIEF DESCRIPTION : To display the line comprising n points whose coordinates are in x_array and y_array. The outline is as follows :
1. perform clipping if clipping is on.
2. perform window to viewport transformation.
3. store the normalized coordinate values in the display file.
4. write the gdi corresponding to the action, move cursor, to the first point in the array.
5. specify color setting action to the gdi
6. specify polyline action to the gdi

FUNCTION NAME : CLIP

MODULE TYPE : output

USAGE : CLIP (xfirst,yfirst,xsecond,ysecond)

INPUT PARAMETER : xfirst - float : x of first point
yfirst - float : y of first point
xsecond - float : x of second point
ysecond - float : y of second point

RETURNED VALUE : int - 1 2 or 3

BRIEF DESCRIPTION : Clips the line such that the visible portion lies within the window of the current normalization transformation.
returns 1 if line is entirely within window
returns 2 if line is entirely outside window
returns 4 if clipped

FUNCTION NAME : clip_mat

MODULE TYPE : output

USAGE : clip_mat (x,y,mat)

INPUT PARAMETER : x , y - float : x and y coordinates of point.

mat - int array of size 3x3

RETURNED VALUE : none

BRIEF DESCRIPTION : Based on the location of the point with respect the window, a value of true is set for an appropriate element of the array mat. The rest of the values are zero or false.

FUNCTION NAME : clip_AND

MODULE TYPE : output

USAGE : clip_AND (mat_first,mat_second)

INPUT PARAMETER : mat_first, mat_second - int : 3x3 matrices

RETURNED VALUE : int 1 2 or 3

BRIEF DESCRIPTION : The two matrices are anded to determine the position of the line with respect to the window.

returns 1 if line is entirely in the window

returns 2 if line is entirely outside the window

returns 3 if the line is partially within the window

FUNCTION NAME : run_plot

MODULE TYPE : VDI

USAGE : run_plot()

INPUT PARAMETER : none

RETURNED VALUE : none

BRIEF DESCRIPTION: This is the device driver for the virtual workstation. Each output primitive, it's associated parameters and attributes are pseudo coded and buffered. Each active workstation receives a copy of this buffer

---

FUNCTION NAME : pr_to_proc

MODULE TYPE : VDI

USAGE : pr_to_proc ()

INPUT PARAMETER : none

RETURNED VALUE : none

BRIEF DESCRIPTION: Each active workstation resides under a device driver process. This routine selects the proper pipe to direct the copy of the output buffer from the VWS such that all active workstations are serviced.

Each device driver for a real device needs to receive a copy of the output buffer from the VWS. The device driver interprets this buffer to the devices under it. An execution routine placed right above the device routines is responsible for calling the appropriate device calls. All devices under the device driver display the image or picture. A brief description of the device driver and the routines to support them are described below.

FUNCTION NAME : main -

MODULE TYPE : device driver

USAGE : through execlp call in GKS ins_pr call. Full path name of the executable file is necessary.

INPUT PARAMETER : display output buffer through standard input. The input pipe is duplicated as standard input.

RETURNED VALUE : none

BRIEF DESCRIPTION: Interprets each pseudo code received from the input pipe. The opening of the device and closing of the device is handled. Output primitives and attributes are transformed from character to appropriate types and buffered. All active devices under the device driver receive a copy of the buffer.

FUNCTION NAME : run_dev

MODULE TYPE : device driver

USAGE : run_dev()

INPUT PARAMETER : none

RETURNED VALUE : none

BRIEF DESCRIPTION : Sends a copy of the buffer to each active device under it.

FUNCTION NAME : exec_plot

MODULE TYPE : device driver

USAGE : exec_plot(i)

INPUT PARAMETER : i - int : index of the location of the device pointer in the array of active devices.

RETURNED VALUE : none

BRIEF DESCRIPTION : Interprets the output buffer from the device driver interface. Displays the image on the device whose file pointer is in the i th location in the array of active devices.

FUNCTION NAME : get_doptr

MODULE TYPE : device driver

USAGE : get_doptr(i)

INPUT PARAMETER : i - int : workstation identifier.

RETURNED VALUE : fa - FILE pointer : a pointer to the device whose tty number is i.

BRIEF DESCRIPTION : Returns the file pointer to the device, given the tty number of the device.

FUNCTION NAME : space

MODULE TYPE : device

USAGE : space (devfl,x0,y0,x2,y2)

INPUT PARAMETER : devfl - FILE Pointer : device pointer
x0,y0,x1,y1 - float : viewport dimensions.

RETURNED VALUE : none

BRIEF DESCRIPTION : NDC to device coordinates transformation. Scaling established.

FUNCTION NAME : move

MODULE TYPE : device

USAGE : move (devfl,x,y)

INPUT PARAMETER : x , y - float : x, y coordinates of a point.

RETURNED VALUE : none

BRIEF DESCRIPTION : Scales and moves from current location to the specified point.

FUNCTION NAME : cont

MODULE TYPE : device

USAGE : cont(devfl,x,y)

INPUT PARAMETER : x,y - float : x, y coordinates of a point.

RETURNED VALUE : none

BRIEF DESCRIPTION : Scales x,y and connects current position to the specified point by drawing a line.

FUNCTION NAME : color

MODULE TYPE : device

USAGE : color(devfl,i)

INPUT PARAMETER : i - int : color index

RETURNED VALUE : none

BRIEF DESCRIPTION : Maps the index to a color type and sets it as the current color.

FUNCTION NAME : label

MODULE TYPE : device

USAGE : label(devfl,s)

INPUT PARAMETER : s - char pointer

RETURNED VALUE : none

BRIEF DESCRIPTION : prints a graphic character string at the current cursor location. s points to the required character string.

APPENDIX B

USERS GUIDE

The current set of GKS functions that have been implemented is being outlined.

GOPKS (ERRFIL)

The application should have a file named erfil open prior to a call to GOPKS. This call should precede any other GKS call.

OPEN_WKSTN (wkst_id,id_connect,typ_wkstn)

to open a workstation of type output. Currently supports output type workstations only. No input or metafile. Initialization is done. Supports upto 20 simultaneous open workstations (can be easily changed to include more).

act_wkstn(i)

To activate a given output workstation. This function is necessary if the output needs to be displayed on a device identified by the tty number i.

deact-wkstn(i) To deactivate an already active workstation. This will block the image from being displayed on

the device identified by the tty number i.

set_viewport (num,xvp_min,xvp_max,yvp_min,yvp_max)

Defines the rectangular viewport on the normalized device, for the normalization transformation specified, using the dimensions passed as parameters.

set_window (num,xmin,ymin,xmax,ymax)

Defines the window in world coordinates. The GKS output will display all portions of the world that is projected on the window. A default viewport of unity is set for the normalization transformation number specified. A subsequent set_viewport call may modify the viewport dimension.

sel_ntran_num (ntrans)

Sets the current normalization transformation number as the one specified. GKS expects that the said normalization transformation number is predefined using set_window and set_viewport. A minimum of set_window for the said normalization transformation number is necessary.

set_colo_type(i)

Sets the current color index to the value specified.

number	regis	hp 7470a
1	dark	dark

2	blue	blue
3	red	red
4	green	magenta
5		green
6		cyan
7		yellow
8		white

** regis under vt125 takes the mod 4 to compute index

set_ltype(i)

Sets the current line type to be the one specified by i. The line types based on the value of i are as follows:

number	linetype
1	solid
2	dash
3	dot
4	dash dot

Currently solid alone available (by default)

polyline(n,x_array,y_array) To draw lines through n consequetive points whose coordinates are in the x_array and the y-array. The default attributes are as follows:

linetype - solid

clipping - clip

fill area - hollow

Clipping is done in the world coordinates using

the current window based on the current normalization transformation.

APPENDIX C

GKS SYSTEM PROGRAMMER'S GUIDE

Introduction

This section is intended to serve the GKS system programmer to

1) Follow the current implementation

2) Aid in future extensions

3) debugging

The current implementation lies at level m of the ANSI GKS [28]. It has the following capabilities.

Polyline Output

Selection of color attribute

Workstation transformation

Multiple simultaneous workstations

Multiple active workstations (capable of parallel execution)

Two types of output workstations

Independently residing device drivers

The implementation conforms to the specifications of ANSI GKS Draft Proposed Standard [28].

The routines and data structures comprising this implementation is divided into modules that fit their GKS functional classification. The salient functions and

procedures under each module is further outlined using flow charts. The functions and procedures supporting the salient functions are described where necessary. Further detailed documentation is provided in Supplement A.

Modules

Each module has a listing of filenames, procedures under the file, description of the procedure and a descriptive flow chart where necessary.

GKS Module Listing

Data Module

File names	Procedures
defs	
atdecl.h	
atexdecl.h	
decl.h	
extdecl.h	
types.h	

defs : contains symbolic constants and the values associated with them.
atdecl.h : declaration of attribute variables and their default values.
atexdecl.h : external declaration of attributes.
decl.h : declaration of GKS global variables & data types.
extdecl.h : external declaration of GKS variables.
types.h : declaration of GKS data types. This is necessary

when using the UNIX make facility.

Control Module

File name	Procedure	Support Routine
gopks.c	GOPKS	
open_wkstn.c	OPEN_WKSTN	INIT_WKSTN
		FILL_PTR
		WKSTLIST_INIT
		WK_ST_ALL_WKSTN
		FIND_TBLWK
		SET_CREATE
gks_fun.c	act_wkstn	FR_WKST
		NE_WKST
		IN_WKST
		ispr_open
		ins_pr
		run_plot
	deact_wkstn	FN_WKTYP
		run_plot

GOPKS - To open GKS for the application and to initialize GKS.

OPEN_WKSTN - Opens an output type workstation. Allocates a workstation state list and inserts it into the set of workstation state lists.

INIT_WKSTN - Allocates space for and initializes a given workstation

FIND_TBLWK - returns a pointer to the entry in the

workstation description table corresponding to the type of workstation. NULL if entry does not exist.

SET_CREATE - allocates a structure of type SETNODE and returns a pointer to it.

FILL_PTR - Reads data to fill and initialize a record of type SETNODE. Inserts the record into the appropriate list.

WK_ST_ALL_WKSTN - Allocates all sub-structures for the workstation state list

act_wkstn - To insert the workstation in the set of active devices. Open the pipes for I/O to the device driver process (if necessary).

FR_WKST - Obtain the first of the active workstation index.

NE_WKST - Obtain the next active workstation index.

IN_WKST - To insert the workstation into the active list.

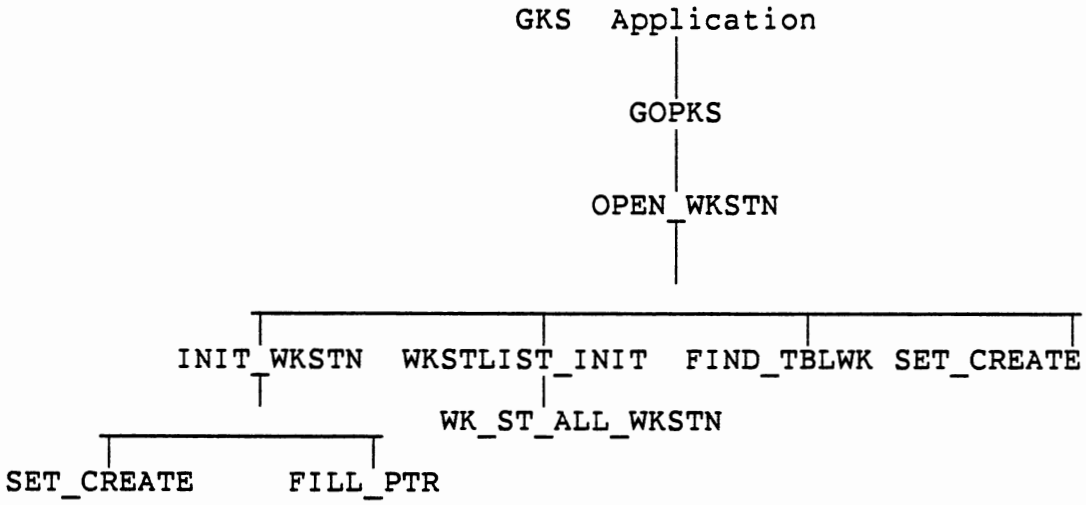
ispr_open - Returns true (1) if device driver for the workstation has already been spawned.

ins_pr - Open the pipe for I/O. Spawns a new process and duplicates the appropriate ends of the pipes to act as standard input and standard output to the device driver process.

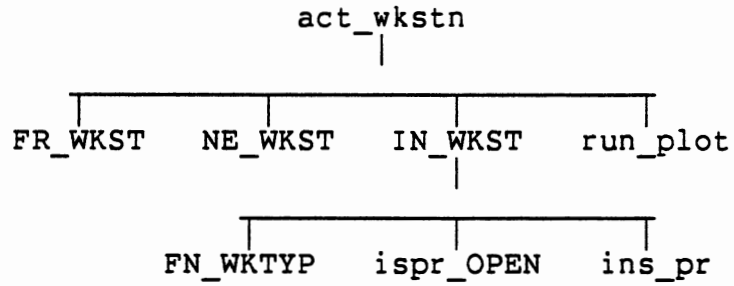
deact_wkstn - Deletes the workstation from the active list. If it is the last of it's type then closes the appropriate pipes to the device driver process corresponding

to that type.

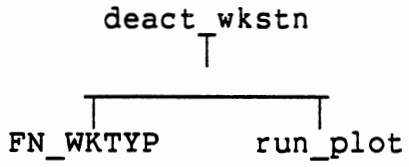
FN_WKTYP - returns the workstation type number.



(A) GKS open and Workstation open



(B) Activate Workstation



(C) Deactivate Workstation

Figure 7. Structured Representation of Control Module

Attribute Module

File name	Procedure
select.c	sel_ntran_num
	set_ltype
	set_col_type

sel_ntran_num - Sets the current normalization transformation number.

set_ltype - Sets the current linetype index (individual attribute)

set_col_type - Sets the current color index (individual attribute)

Workstation Transformation

As described in APPENDIX A. Please refer Supplement A for further details. The GKS functions of set_window and set_viewport belong in this module.

Output Module

File name	Procedure	Support Functions
Polyline.c	Polyline	CLIP
		clip_mat
		clip_AND

Polyline - To draw a line connecting n points whose coordinates are in x_array and y_array.

linetype = solid

clipping = clip

fill area = hollow

color = red by default. (not applicable for monochrome devices)

CLIP - Clips the line using the current normalization transformation. Sutherland's algorithm is used.

clip_mat - The position of the given end of a line with respect to the clipping rectangle is specified by a true value in a 3 X 3 matrix.

clip_AND - The relative location of the two ends of a line with respect to the clipping rectangle is specified by ANDing the clip matrices for the two ends.

Virtual Workstation Module

File Name	Procedure
devdriv.c	run_plot

run_plot - This routine reads the global data interface. It executes the GKS output on the VWS by buffering the command and it's parameters in a formatted fashion. When the buffer is in excess of 2K, each device driver process receives the data in the buffer through the appropriate pipes.

Device Driver Module

File name	Procedure	Support Functions

reggrfl.c	main	exec_plot
		del_dev

main - Scans the standard input for GKS output data. The opening and closing of devices is handled by this routine. GKS output data is converted from char to appropriate types and buffered. Devices receive the buffer to execute the image on the screen.

exec_plot - Interprets the display buffer and displays it on the screen.

del_dev - Deletes the output device from the set of open graphics devices. If it is the last device, it closes the device driver process.

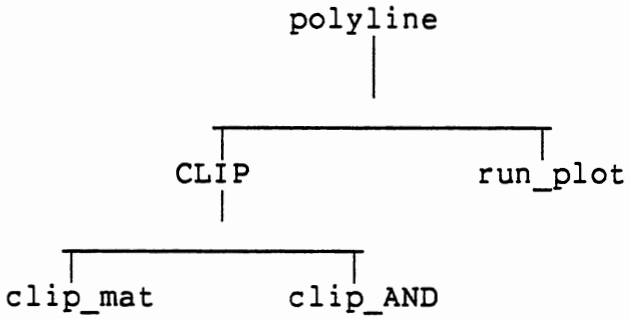


Figure 8. Structure of Polyline Output Function

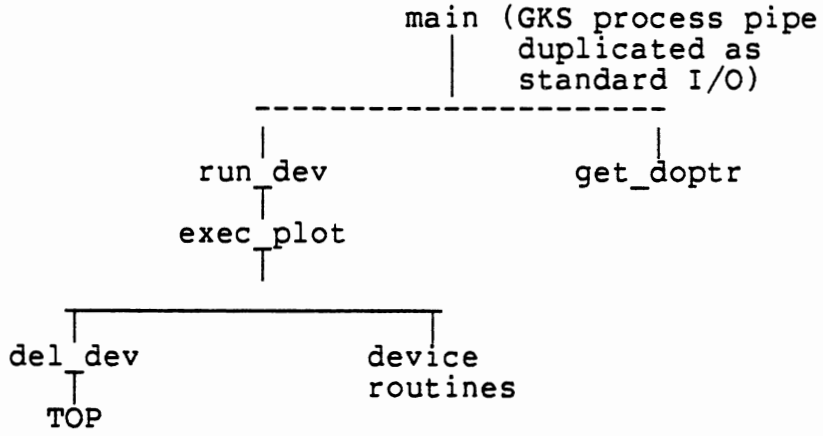


Figure 9. Structure of Device Driver Interface

APPENDIX D

LIST OF SOME COMPUTER GRAPHICS TERMINOLOGIES

Attribute:

A particular property that applies to a display element (output primitive) or a segment. Examples: line color, line type.

Bundled Attributes:

A method of specifying all the attributes that qualify a display element in one area. This bundle is represented by a bundle index.

Clipping:

Removing parts of display elements that lie outside a given boundary, usually a window or a viewport (This implementation clips against a window).

Device Co-ordinates(DC):

A coordinate system that is device dependent.

Device Driver:

The device dependent part of a GKS implementation intended to support a graphics device. The device driver generates device dependent output and handles device dependent interaction.

Display Element:

A basic graphic element that can be used to construct a display image.

Display Image; Picture:

A collection of display elements or segments that are represented together at any one time on a display surface.

Display Surface; View Surface:

In a display device, that medium on which display images may appear.

Event Mode:

In this mode both the GKS application program and the input process are active. The dominant process is the input process. The input process provides data and the application program acts on the data immediately (interrupt).

GKS level:

Two values from the set (m,1,2,2) and (a,b,c) which together define the minimal functional capabilities provided by a specific GKS implementation.

GKS metafile:

A sequential file that can be written or read by GKS; used for long term storage, transmittal and transferral of graphical information.

Highlighting:

A device independent way of emphasizing a segment by modifying its visual attributes.

Locator Devices:

A GKS logical input device providing a position in world co-ordinates and a normalization transformation number.

Normalization Transformation; Viewing Transformation; Window-to-Viewport Transformation:

A transformation that maps the boundary and interior of a window to the boundary and interior of a viewport. The viewport definition in GKS assumes normalized device co-ordinates.

Normalized Device Co-ordinates:

A co-ordinate specified in a device independent intermediate co-ordinate system, normalized to a range which is typically 0 to 1.

Orthogonal Functions:

In computer graphics an orthogonal function or an orthogonal module means that a module is independent of other modules or that the dependency is defined and well structured.

Output Primitive; Graphic Primitive:

A display element. GKS output primitives are polyline, polymarker,text,fill area,cell array and generalized drawing primitive.

Pick Device:

A GKS logical input device providing the pick identifier attached to an output primitive and the associated segment name.

Pixel; Picture Element:

The smallest element of a display surface that can be independently assigned a color or intensity.

Polyline:

A GKS output primitive consisting of a set of connected lines.

Raster Graphics:

Computer graphics in which a display image is composed of an array of pixels arranged in rows and columns.

Rotation:

Turning all or part of a display image about an axis.

Sampling Mode:

In this mode a GKS application program and input process are both active. The application program is dominant. The input device buffers the required input data. The application program uses the data in the buffer as and when they are needed.

Scaling; Zooming:

Enlarging or reducing all or part of a display image by multiplying the co-ordinates of display elements by a constant value.

Segment:

A collection of display elements that can be manipulated as a unit.

State list:

GKS data structure or data-types that provides a convenient way to maintain information regarding

workstations, segments and the state of GKS.

String Device:

A GKS logical input device providing a character string as its result.

Translation; Shift:

The application of a constant displacement to the position of all or part of a display image.

Valuator Device:

A GKS logical input device providing a real number.

Viewport:

An application program specified part of normalized device co-ordinate space.

Window:

A predefined part of a virtual space.

Workstation:

GKS is based on the concept of abstract graphical workstations, which provide the logical interface through which the application program controls physical devices.

Workstation Transformation:

A transformation that maps the boundary and interior of a workstation window into the boundary and interior of a workstation viewport(part of display space) , preserving aspect ratio.

Workstation Viewport:

A portion of display space currently selected for output of graphics.

Workstation Window:

A rectangular region within the normalized device co-ordinate system which is represented on a display space.

World Co-ordinate(WC):

A device independent Cartesian co-ordinate system used by the GKS application program.

VITA

MUKUND JAGANNATHAN

Candidate for the Degree of
Master of Science

Thesis: MULTIPLE PARALLEL GKS WORKSTATIONS UNDER UNIX

Major Field: Computing and Information Sciences

Biographical:

Personal Data: Born in Coimbatore, India, April 2, 1956 to D. Jagannathan and Lakshmi Jagannathan

Academic: Completed Requirements towards a Master's in Computing and Information Sciences at OSU, OK; Masters in Mechanical Engineering from Kansas State University, Manhattan, Kansas, Spring, 1981; Bachelor of Engineering from University College of Engineering, Bangalore University, Bangalore, India, Spring 1978.

Professional Experience: Graduate Research Assistant, Department of Interior Design, Oklahoma State University, Stillwater, OK, March 1984-May 1985.

Graduate Research Assistant, Department of Mechanical Engineering, Kansas State University, Manhattan, Kansas, August 1979-July 1981
Junior Engineer, Bharat Heavy Electricals, Ltd., New Delhi, India, September 1978-August 1979.