Science and Engineering Research Council
Rutherford Appleton Laboratory
CHILTON, DIDCOT, OXON, OX11 0QX

RAL-86-081

Graphics Standards – The Current State

F R A Hopgood and D A Duce

August 1986

Science and Engineering
Research Council

GRAPHICS STANDARDS - THE CURRENT STATE

F R A Hopgood and D A Duce

This volume contains two papers presented by the authors at the Ausgraph 86 conference held in Sydney, Australia from 6-11 July 1986.

The first paper, 'Graphics Standards - The Current State', formed the basis for an introductory tutorial on graphics standards. The tutorial covers the GKS standard in some detail, together with descriptions of the other standards in computer graphics currently under development, including GKS-3D, PHIGS, CGM and CGI. This paper also contains a survey of GKS implementations.

The second paper, 'Computer Graphics Programming', formed the basis for a professional seminar which examined the philosophy behind graphics standardisation, the major concepts embodied in graphics standards, and some of the problems that will be encountered when moving from existing graphics packages to GKS. The paper discusses GKS-3D and PHIGS in more detail than the introductory tutorial and discusses implementation strategies for GKS. The final section in the paper discusses window managers and the current state of standardisation activities in that field.

INTRODUCTORY TUTORIAL

AUSGRAPH - 6 July 1986

GRAPHICS STANDARDS - THE CURRENT STATE

F R A HOPGOOD
D A DUCE

Informatics Division
Rutherford Appleton Laboratory, UK

1. INTRODUCTION

After more than 10 years of effort by many people from a number of countries, graphics has its first international standard in GKS, the Graphical Kernel System. Rather than being the end of the road it is just the beginning. GKS is the main building block of a set of inter-related standards that are due to appear over the next few years with the aim of producing a comprehensive set to cover the graphics area.

Standardisation in the area of computer graphics had a slow start partly due to the rapid changes in hardware but also due to a number of different methodologies that had grown up in separate communities. The origins of the current standards activities can be traced back to 1974 when IFIP Working Group 5.2 invited Richard Guedj of France to initiate a programme of work which would establish a methodology for computer graphics. A workshop of leading experts in the field was held in Seillac, France in 1976 and this proposed a strategy which directed future work [1].

The main thrust of that strategy was that existing graphics systems confused the modelling side of computer graphics (where pictures were composed) with the viewing side (where composed pictures were displayed in a particular orientation on a specified device). The workshop recommended a clearer separation between these two functions and proposed that the viewing system should be the first area to be standardised. GKS is the result of this initial activity. A description of the history and basic concepts of GKS are given in [2].

Standards provide a mechanism for formally specifying the exchange of information across an interface. GKS concentrates on standardising the interface between the application and the graphics system together with the interface between the graphics system and a workstation (which in GKS terms can be thought of as an intelligent device capable of controlling a variety of input devices, a display surface and providing local picture manipulation and storage of a simple nature). Figure 1 gives a representation of GKS and its interfaces.

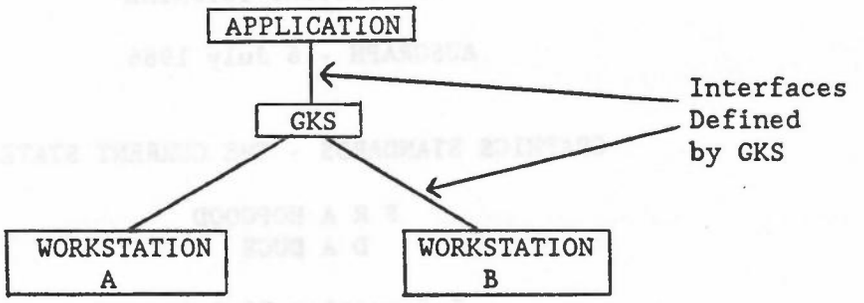


Figure 1: Interfaces defined by GKS

Both GKS interfaces are defined functionally with no specification of how they should be realised in terms of a language interface to the application or a protocol between GKS and an intelligent device. Consequently, it is feasible for a variety of systems to arise all conforming to the GKS definition but having totally different characteristics.

The future standards activities are partially aimed at making the existing interfaces more tightly specified but also introducing other interfaces and providing greater functionality. Figure 2 shows how the current standards activities fit together.

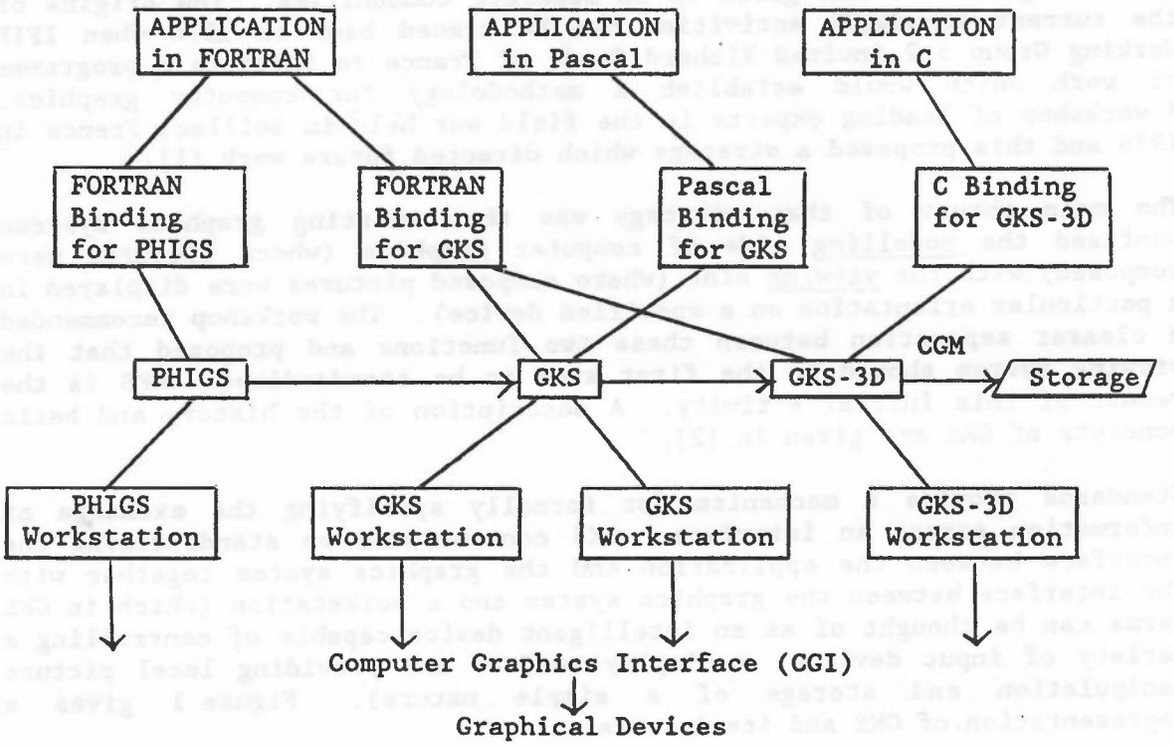


Figure 2: Standards

Not all the interfaces are shown in Figure 2. For example, there will be a range of language bindings to all the graphics standards rather than the limited number shown. The main thrusts of the standardisation activities following GKS have been:

1. To provide language bindings to be used with GKS.
2. To provide protocols for long-term storage of graphical information (CGM - Computer Graphics Metafile).
3. To provide a standard interface to graphics devices (CGI - Computer Graphics Interface).
4. To increase the functionality of the standards by extending GKS to 3D and also providing a modelling facility (PHIGS).

The rapid increase in the number of activities has caused individuals involved to concentrate on one or other of the standards activities. As a result, there is a danger that the various activities will drift apart. To avoid this, the aim is to standardise a Reference Model of how the various standards interact.

2. ISO TERMINOLOGY AND MEETING SCHEDULE

More than 200 people are currently involved in the ISO graphics standards activities with international and national meetings taking place each year. There tends to be one full ISO Meeting each year but, depending on a standard, there may be additional technical and editorial meetings. The full ISO meetings and major technical meetings held have been:

1979	Budapest, Hungary
1980	Tiefenbach, W.Germany
1981	Melbourne, Florida, USA
1981	Abingdon, Oxon, UK
1982	Steensel, Netherlands
1983	Gananoque, Canada
1984	Benodet, France
1985	Timberline, Oregon, USA
1986	Frankfurt, W.Germany
1986	Egham, Surrey, UK (to be held in September)

Since the last full meeting at Timberline, there have been a number of Technical Meetings held in Frankfurt and a GKS-3D editorial meeting in the USA.

In any activity that involves communication between people, a few terms are used so regularly that they become an extension to the vocabulary. The standards area is no exception and it is important that a reader is aware of the various stages that a standards document goes through within ISO before becoming an international standard. These are:

1. WORKITEM: an official project with agreed scope and goals and timescales. When an area has been identified for standardisation, a proposal for a project is prepared. There is a ballot on the proposal within the appropriate Technical Committee (TC) and, if successful, the workitem is assigned to a particular Subcommittee

(SC), who manage the project and in turn assign it to a particular Working Group (WG) who carry out the technical work. (The Working Group for computer graphics is WG2 within SC21 (Open Systems) within TC97 (Information Processing Systems) designated TC97/SC21/WG2.)

2. DRAFT PROPOSAL (DP): the Working Group produces successive working drafts of the standard until the document is sufficiently mature that it can be submitted to the Subcommittee for registration as a DP. The DP is then circulated within the SC for technical review and ballot. Comments submitted with the votes are addressed and resolution of them is sought. If sufficient agreement is reached the document proceeds to the next stage. If not, or if the document has undergone substantial change, then it has to be circulated for a further DP ballot.

3. DRAFT INTERNATIONAL STANDARD (DIS): when sufficient agreement is reached on the DP document, the revised document is registered as a DIS. The publication of a DIS should indicate that technical agreement has been reached. The document is then circulated within the TC for editorial review and DIS ballot. Comments submitted with the votes are addressed and resolution sought. Any remaining problems at this stage can cause another DIS ballot, but normally the document proceeds to the next stage.

4. INTERNATIONAL STANDARD (IS): the DIS revised in the light of comments received with the DIS ballot becomes the Final Text. A final ballot within ISO Council ensures that all ISO members are satisfied that ISO procedures have been followed in the production of the standard and that the document is suitable for publication. The IS is then published. The document will be reviewed 5 years after publication at which time it may be endorsed, revised or abandoned.

The voting process in ISO is by letter ballot and takes many months each time. Consequently progress is slow and getting to a full international standard is a long and time-consuming activity requiring considerable stamina.

A question immediately arises as to when a new standard proposal is sufficiently well developed to warrant either implementing or using. In theory, all technical change should be complete by the DIS stage and this is, therefore, a reasonable time to start using a standard. However, the move from DIS to IS inevitably includes some technical revisions if only to remove ambiguities. Consequently, time should be allowed for updating any programs written at the DIS stage when the IS finally arrives. In the case of GKS, minor changes did occur between the DIS and IS stages. As a result, most of the books currently on the market have minor errors. The only one known to be up-to-date is [3].

Pilot implementations of standards often occur at the DP stage. This is right and proper as such implementations often point out problems in implementation. A danger with such products is that there is a tendency to massage the DP version to produce the DIS one with a consequent loss of efficiency and elegance. Choosing a particular implementation of the standard should at least give some thought to the history of the product. Probably the best product would be a new DIS one where the company had tested out algorithms and techniques on an earlier DP prototype.

The dates for GKS were as follows:

Workitem	Spring 1981
Draft Proposal	February 1982
Draft International Standard	June 1983
International Standard	August 1985

Note the long elapsed time between Workitem and International Standard. Predicted dates for future standards should be judged against this schedule to decide how likely the progress will be made to keep the predicted dates.

3. GKS

The Graphical Kernel System - GKS is formally defined in the standard document itself [4]. A more detailed introduction and primer is given in Hopgood et al [3] while a full and comprehensive treatment with many examples is given in Enderle et al [5]. The latter book is in the process of being updated to agree with the International Standard and current FORTRAN binding. It should appear later in 1986. A computer graphics text book based on GKS that can be recommended is Hearn and Baker [6]. Readers should be warned that it has a few errors as it uses an earlier version of the FORTRAN binding.

Rather than give a full description of GKS here, we will concentrate on those areas which are significantly different from previously accepted packages or are major features of GKS and the related standards.

3.1 Dimensionality

GKS is a two-dimensional graphical system and provides no support for three dimensions. The major reason for this was that it was realised that the standardisation of a 3D system would take significantly longer than a 2D-only system. There was an urgent need for a standard and large parts of industry had no interest in 3D. Consequently, the right approach was to move quickly to the definition of a 2D standard with the intention of defining a 3D standard above the 2D system at a later date. The extension of GKS to 3D will be described later.

3.2 Primitives

The six basic output primitives are polyline, polymarker, fill area, text, cell array and generalised drawing primitive (GDP).

Previous packages including the GSPC CORE system were based on the concept of Current Position (CP). The usual line drawing primitive in such packages is to generate a line from CP to a specified point followed by updating CP to be the specified point.

GKS, on the other hand, defines output of this type by specifying a sequence of points and the polyline output primitive draws a set of lines between the sequence of points. One advantage of this approach is that aspects such as dotted or broken apply to the complete polyline rather than individual line segments, a more natural view when drawing curves depicted as polylines.

The polymarker primitive is similar to the polyline primitive but marks the sequence of points with a specified symbol rather than connecting the points with lines.

The text primitive in GKS provides considerable flexibility in defining the quality of the text, its size and orientation, the origin etc. Figure 3, where the asterisk defines the text origin, indicates the types of text available in GKS. It also supports the text path being in any of the major directions providing support for those languages not writing from left to right.

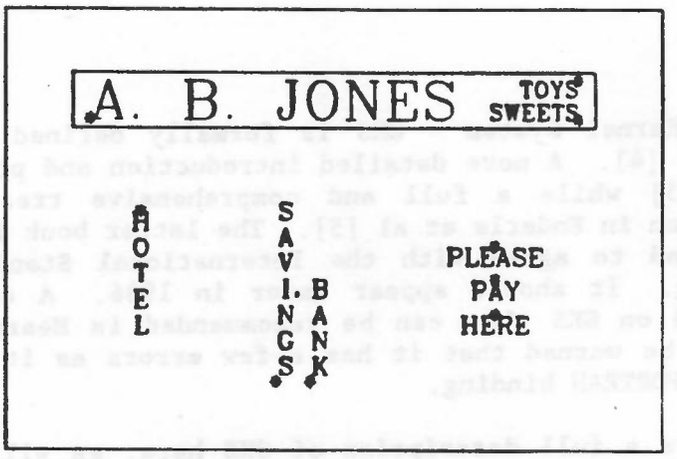


Figure 3

The fill area primitive is defined in terms of a set of points which specify a closed curve. The primitive fills the enclosed area with a solid colour or allows it to be filled by a specified pattern or hatch style.

The cell array primitive is specifically aimed at the image processing community where the cell array defines the colour or grey level to be associated with individual elements of a rectangular array.

Finally, GDP defines a controlled method of adding more exotic primitives. Particular implementations are free to add to the basic primitive set by specifying particular GDP types as producing higher level shapes such as circles, ellipses etc.

The appearance of primitives on a display is determined by the aspects of the primitives. For example, the aspects of a polyline are: linetype, linewidth scale factor and polyline colour index. Linetype may be solid, dashed, dotted, dashed-dotted or other implementation dependent possibilities. Linewidth scale factor defines linewidth as a function of a standard linewidth for the output device. The polyline colour index points to a colour description. The method of setting the values of aspects will be described shortly.

3.3 Coordinate Systems and Workstations

GKS has introduced the concept of an abstract workstation to hide the peculiarities of device hardware. A workstation consists of zero or one display surfaces and zero or more input devices. GKS assumes that

applications will frequently want to use more than one workstation simultaneously. For example, an operator may be interacting with a design through a refresh display, whilst taking copies of completed parts of the design on a plotter.

Coordinate data in the parameters of an output primitive are specified in world coordinates (WC), a Cartesian coordinate system. Transformation to the coordinate system of the display device is accomplished in two stages; firstly, world coordinates are transformed to an intermediate coordinate system called normalised device coordinates (NDC) by a window to viewport mapping termed a normalization transformation, then a second window viewport mapping, called the workstation transformation transforms these coordinates to device coordinates (DC). The aspect ratios of window and viewport may differ in the normalization transformation, but the workstation transformation maps the workstation window to the largest possible region of the workstation viewport with the same aspect ratio.

A major difference between GKS and other earlier systems is that it provides multiple normalization transformations all defined at the same time. Coordinates of primitives are transformed by the currently selected normalization transformation. Figure 4 shows three different objects (duck, tree and house) defined in different world coordinate systems. Three different normalization transformations can be defined which map these world coordinates onto specific areas of the NDC space.

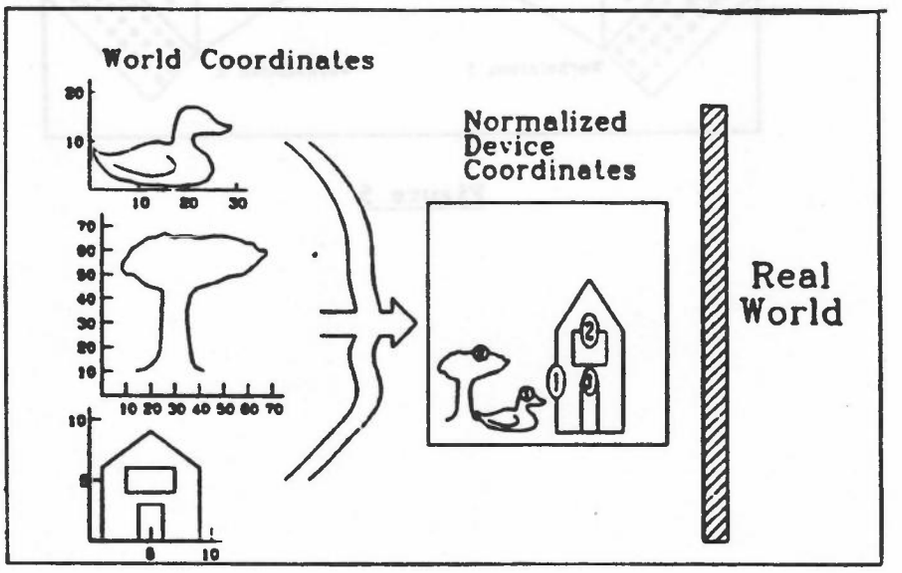


Figure 4

This leads to a different style of programming with the normalization transformations, rather like declarations, being defined at the head of the program and not changed. Earlier systems tended to mix calls of primitives with changes in coordinate system.

The workstation transformation may be set differently for different workstations, thus allowing different parts of the virtual picture to be displayed on different workstations.

The workstation can be regarded as a camera pointing at some part of the NDC space. In Figure 5, the first workstation is pointing at the tree while the second points at the house.

The boundary of the window of the currently selected normalization transformation serves as a clipping rectangle against which output primitives may be clipped. There is also a compulsory clip to the boundary of the window of the workstation transformation.

The Viewing Pipeline in GKS is given in Figure 6.

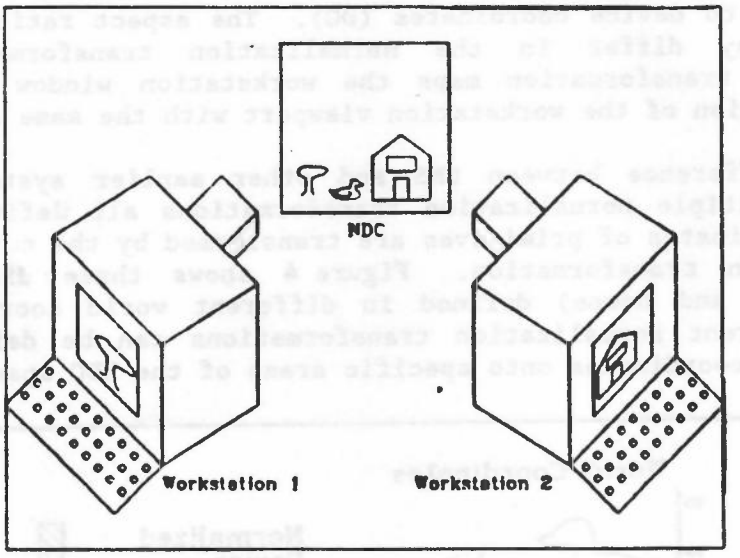


Figure 5

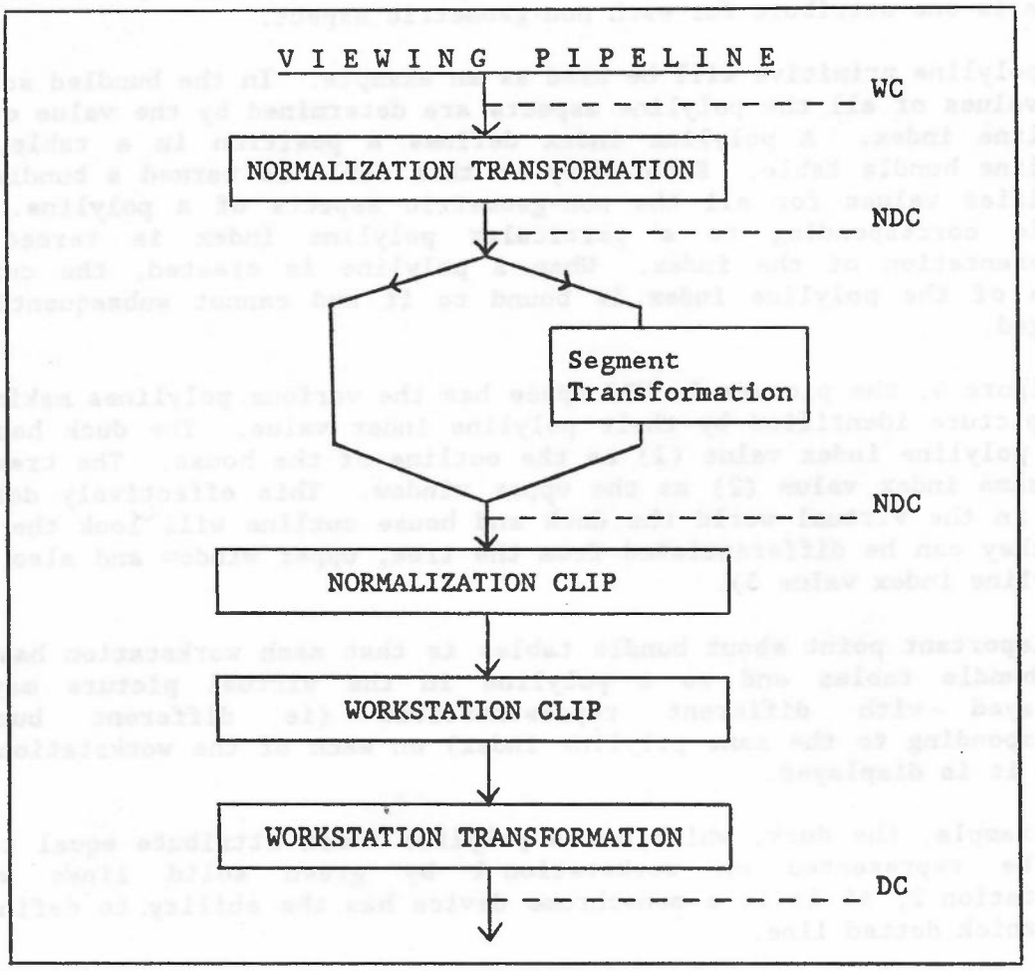


Figure 6

3.4 Attributes

As noted above, the appearance of a primitive displayed on a workstation is determined by its parameters and by additional data termed aspects. The values of aspects are determined by attributes. Aspects fall into two categories: geometric and non-geometric. Geometric aspects control the shape or size of a primitive, for example the height of a text primitive. Each geometric aspect is controlled by a single geometric attribute. Geometric attributes are specified in world coordinates, are set modally and are subject to the normalization and workstation transformations. A primitive has the same geometric aspect values on all the workstations on which it is displayed. Only the text and fill area primitives in fact have geometric aspects.

Non-geometric aspects control facets of the appearance of a primitive which are not related to its shape or size, for example the linetype (solid, dotted etc) with which a polyline is displayed. The values of non-geometric aspects may be controlled in one of two ways: bundled specification - there is one attribute per output primitive which controls

the values of all the non-geometric aspects; individual specification - there is one attribute for each non-geometric aspect.

The polyline primitive will be used as an example. In the bundled scheme, the values of all the polyline aspects are determined by the value of the polyline index. A polyline index defines a position in a table, the polyline bundle table. Each entry in this table is termed a bundle and specifies values for all the non-geometric aspects of a polyline. The bundle corresponding to a particular polyline index is termed the representation of the index. When a polyline is created, the current value of the polyline index is bound to it and cannot subsequently be changed.

In Figure 4, the picture in NDC space has the various polylines making up the picture identified by their polyline index value. The duck has the same polyline index value (1) as the outline of the house. The tree has the same index value (2) as the upper window. This effectively defines that in the virtual world the duck and house outline will look the same but they can be differentiated from the tree, upper window and also door (polyline index value 3).

The important point about bundle tables is that each workstation has its own bundle tables and so a polyline in the virtual picture may be displayed with different representations (ie different bundles corresponding to the same polyline index) on each of the workstations on which it is displayed.

For example, the duck, which has a polyline index attribute equal to 1, can be represented on workstation 1 by green solid lines while workstation 2, if it is a monochrome device has the ability to define it as a thick dotted line.

This is a powerful tool for achieving application program portability between different workstation environments. If carefully constructed, moving a program to a different environment will merely mean defining new representations for the different indices used in the picture, to employ in the best way possible, the characteristics of the workstations in the new environment.

In the individual scheme the values of the polyline aspects will be the same on all the workstations on which the polyline is displayed and each workstation must do the best it can to display the polyline with the requested aspect values.

The bundled scheme is important then when it is necessary to ensure that primitives with different attributes can be differentiated on different workstations; whilst the individual scheme is important when primitives with specific attributes are to be represented on each workstation as closely as possible to the specification.

3.5 Input

One of the areas which was considerably refined during the GKS review process was the model of graphical input. Just as output was defined in terms of device independent primitives and attributes, the aim was to specify a set of virtual input devices on to which real input devices could be mapped.

All input devices in GKS were formalised as having a measure and a trigger. The measure describes the type of input value returned by the device (position, value, text etc) while the trigger causes the measure value to be returned to the application program in certain styles of input.

Six logical input devices are defined in GKS each delivering measures as follows:

LOCATOR : a position in world coordinates and the associated normalization transformation number used to convert back from device coordinates via NDC to world coordinates.

STROKE : similar to LOCATOR but delivering a complete sequence of world coordinate positions.

VALUATOR : a real number.

CHOICE : an integer representing a selection from a set of choices.

PICK : the name of a selected segment and an identifier indicating which set of primitives in the segment has been picked.

STRING : a string of characters.

A GKS implementation supporting input must provide a simulation of each logical input type to the operator. They need not all be on one workstation if he has several workstations under his control.

The three operating modes in which GKS input devices may be set to provide input are:

REQUEST : rather like a FORTRAN READ. A request is made by the application program for a measure of the specified device to be returned. GKS will wait until the operator has set the measure to the desired value and activated the trigger. The measure value is returned to the application program which then continues executing.

SAMPLE : the measure is continually updated and GKS returns the current value of the measure whenever the application program requests it. For SAMPLE input, the trigger is not required.

EVENT : a number of input devices may be active together. Each time the trigger for a particular device is activated, the current measure value is added to a single queue of input events for all the devices in use in EVENT mode.

The application program can interrogate the queue acting on the events that have taken place. It is possible to couple more than one input device to the same trigger so that multiple events at the same time are possible.

The local installation decides how the real input devices are modelled to provide the necessary logical input devices.

A major difference from many interactive graphics systems is that all logical input devices can be used in all three operating modes.

3.6 Segments

The segment model in GKS is less innovative than some parts of the standard. Associated with each workstation is a segment store in which segments consisting of sets of GKS commands can be stored. Functions exist to create, delete, rename and manipulate segments. Associated with each segment are a set of attributes which control visibility, highlighting, priority ordering for output when segments are overlayed and detectability from a pick device. It is also possible to transform segments such that the picture defined by the segment can be scaled, moved, rotated etc.

As well as the segment store associated with the workstation, there is also a workstation independent segment storage (WISS) which is used as a central library. Segments can be moved from WISS to a workstation. A macro facility is also provided so that segments in WISS can be inserted into other segments.

3.7 Levels

Rather than insist that all facilities in GKS are supported by every implementation, GKS is defined as a set of levels on two orthogonal axes:

0 : simple output
1 : output including segments
2 : full output including all the facilities for inserting segments from WISS into the current segment
a : no input
b : REQUEST input only
c : all forms of input

There are, therefore, 9 levels in total with 0a the simplest and 2c the most comprehensive.

3.8 Summary

GKS is the first international standard. Its major limitation is that it is a 2D standard. In compensation, its attribute and input models are much better than those previously used. Its coordinate systems provide a greater level of flexibility and device independence than most earlier systems.

3.9 GKS Implementations

The two early implementations of GKS which are still widely available are GKSGRAL which derives from the version implemented at the Technical University of Darmstadt and the joint ICL-Rutherford Appleton Laboratory implementation which is widely used in the UK university environment.

Since these early implementations, there have been a number of additional ones falling into three classes:

(1) Device Manufacturers: these implementations are oriented towards the hardware of the particular manufacturer although the host package is usually portable.

(2) Host Manufacturers: implementations by a mainframe manufacturer for his range of systems hopefully with a good coverage of popular devices.

(3) Software Houses: these are products aimed at being sold to other organisations for their products (either device or host).

The following table lists some of the implementations that we have knowledge of with details where known. The information is not complete and absence of particular points does not necessarily imply that the implementation does not have that facility (only that we are unaware of it). Again the list of hosts and devices supported is not complete. The intention is to give a flavour of the extent to which it has been made generally available.

Company	Level	Language Binding (* under devlmt)	Hosts	Devices	Metafile	GDPs	Comments
(1) Rutherford Appleton/ ICL, UK	1b going to 2b	FORTRAN	PRIME VAX IBM GEC PYRAMID UNIX ICL2900 PERQ	Tektronix Sigma Benson Calcomp	GKSM		Well established implementation compatible with the standard.
(2) GTS-GRAL Darmstadt, W.Germany GIXI in USA	2c	FORTRAN C* Pascal* ADA*	APOLLO VAX PRIME IBM SUN HONEYWELL UNIVAC	Benson Calcomp DEC Genisco IBM Ramtek Tektronix Versatec	GKSM and user Defined	Circle Arc Bezier Cubic	Fast implementation with good characteristics. Several packages available on top.
(3) NOVA Graphics Austin, Texas	2b	FORTRAN Pascal C ADA	IBM VAX UNIX CRAY IBM PC	Tektronix Ramtek IBM HP Sigma Seiko			Distributed implementation allowing part in workstation
(4) Centre for Mathematics and Informatics (CWI) Amsterdam	2c	FORTRAN C	Various	Tektronix AED Versatec Ramtek IBM	GKSM		Widely available. Frequently sold by other companies under their name.
(5) Tektronix Ltd	2b	FORTRAN	VAX IBM UNIX GEC	Tektronix Skeleton Driver	Non Standard GKSM at next release	None	Contains non-standard routines to handle Tektronix input.

Company	Level	Language Binding	Hosts	Devices	Metafile	GDPs	Comments
(6) CEEGEN Corp Los Gatos, Calif, USA	2b	FORTRAN	Pyramid VAX Silicon Graphics UNIX Honeywell IBM	Tektronix Epson Matrox Strobe		Circle Arcs Ellipses Bezier	Graphics Modelling System on top of GKS
(7) Whitechapel Ltd, UK	2b	FORTRAN Pascal C	White-chapel MG-1		GKSM		Based on Mel Slater's QMC Implementation
(8) Prior Data Sciences Ltd Ottawa, Ontario	2b	C	UNIX VAX Ridge IBM PC	Tektronix Versatec IBM HP Epson Adage AED	GKSM	Circle	Non-standard functions for deactivating segments have been added
(9) Precision Visuals Boulder, Colorado	2b	FORTRAN	IBM VAX UNIX	Various (80 in all)	GKSM		Rich implementation, many linestyles fonts etc
(10) Dataplotting Services Inc	2b	FORTRAN Pascal C	VAX PRIME CDC IBM INTEL UNIX Siemens	Various	GKSM	Curve Arc Circle	Interfaces to Calcomp and PLOT10 packages.
(11) Ramtek	2b	FORTRAN	VAX Norsk Data Perkin Elmer	Ramtek 2020-4220 firmware support			

Company	Level	Language Binding	Hosts	Devices	Metafile	GDPs	Comments
(12) Visual Engineering San Jose			Pyramid				
(13) Advanced Technology Centre, Culver City, California	2c	FORTRAN C	Various	Various			
(14) TEMPLATE (Megatek) San Diego (Derived from NOVA-GKS)	2b	FORTRAN	Various	Megatek			
(15) UNIRAS Burlington, Mass and Free University Berlin	2b	FORTRAN	CDC Siemens IBM DEC PRIME UNIVAC HARRIS	Benson Calcomp Tektronix Ramtek HP Sigma Grinnell DEC DScan Lexidata Jupiter			Many application packages built on top
(16) DEC	0b	FORTRAN	DEC	DEC			
(17) Data General	2b	FORTRAN	Data General	Data General			
(18) IBM/ Graphic Software Systems		FORTRAN C Basic	IBM PC	IBM PC			
(19) Infolytica, Montreal	mb	FORTRAN	Various	Tektronix			CAD System being built on top.

Company	Level	Language Binding	Hosts	Devices	Metafile	GDPs	Comments
(20) AED-GKS, Bonn	2c	FORTRAN	VAX UNIX GOULD UNIVAC	Tektronix Megatek Ramtek Calcomp			
(21) CMC, Bombay	2b	FORTRAN	VAX	Tektronix Plotters			
(22) SYSGRAPH, Vienna	2b	FORTRAN	VAX PRIME UNIX CDC Siemens	Tektronix Calcomp DEC HP Westward			
(23) System Simulation Ltd, London	1a	FORTRAN	UNIX	Westward Megatek Calcomp HP			
(24) XGKS, Hungary	2b	FORTRAN	UNIX DEC	Tektronix Calcomp DEC			
(25) Harris	?	Ada	Harris				
(26) Queen Mary's College London	2b	FORTRAN	Several				

4. GKS-3D

4.1 Introduction

This seeks to extend GKS to 3D by adding various capabilities, as can be seen from the scope given in the Draft Proposal:

(a) the definition and the display of 3D graphical primitives;

(b) mechanisms to control viewing transformations and associated parameters;

(c) mechanisms to control the appearance of primitives including optional support for hidden line and/or hidden surface elimination but excluding light source, shading and shadow computation;

(d) mechanisms to obtain 3D input.

The aim is to specify the system in such a way that existing (2D) GKS programs would run without any modifications and that the general style of capabilities provided would match those included in GKS.

To achieve compatibility between GKS and GKS-3D, existing GKS 2D functions are still provided in GKS-3D but, conceptually, have a Z=0 coordinate added to every position. As a result, existing GKS output sits on the Z=0 plane. The default viewing transformations provided will produce a parallel projection on to the same part of the workstation display screen as if the output had come from a GKS 2D system.

4.2 Output Primitives

GKS-3D has seven output primitives. Six of these correspond to the GKS primitives, the seventh is fill area set which displays a set of polygonal areas (this is particularly convenient for specifying areas with holes or disjoint areas that are to be treated as a single entity). This primitive was added because it was felt that in a 3D environment rendering may be needed across a set of areas.

Text, fill area, fill area set and cell array are planar primitives in arbitrary planes. Planar primitives have an obverse and a reverse side (and zero thickness), determined uniquely from the parameters and aspects of the primitives. Characters of the text primitive and patterns of the area primitives are generated on the obverse. Viewing the reverse displays a mirror image of the obverse.

GKS-3D provides both 3D and 2D functions to generate instances of the primitives. The 3D functions accept 3D coordinate data whereas the 2D functions only accept 2D data and generate primitives in the Z=0 plane (in world coordinate space).

The appearance of fill area set primitives is determined by the fill area aspects and by a new set of aspects controlling the appearance of the edges of the primitive. An edge bundle table has been introduced with corresponding bundled and individual attributes.

4.3 Viewing

The Viewing Pipeline in GKS-3D is given in Figure 7. Similar to GKS, a transformation exists to change the user defined world coordinates into a consistent Normalised Device Coordinates for internal use within GKS-3D. The next operation in the pipeline is to view the NDC picture. For most workstations viewing consists of projecting the 3D image on to a 2D projection plane. Functions are provided to assist with the definition of this viewing operation. There is a change in coordinates from Normalised Device Coordinates (NDC3) to Viewing Coordinates by defining a View Reference Point and a set of axes associated with it. The intention is that this point has some relationship to the object to be viewed and makes the setting up of the projection transformation that much easier.

Once the Viewing Coordinates are established, Front and Back Planes are defined which specify the limits of the object to be viewed. A Projection Reference Point can be specified and a Projection Plane which allows the object to be viewed by projecting it onto the projection plane. The View Window specifies that part of the projection plane to be output to the workstation. Both parallel and perspective projections are provided.

For some workstations, capable of providing 3D geometric transformations and for genuine 3D devices, the viewing operation specified by the functions provided may not be appropriate. Therefore, it is possible for applications to construct their own viewing pipeline or ignore parts of it.

4.4 Multiple Views and Hidden Surface Calculations

Each primitive in GKS-3D has a View Index associated with it which defines which viewing transformation is to apply to it on a particular workstation.

It was believed that, unlike GKS, there is a need for more than one view to be available at a time on a workstation. This would, for example, allow titles to be output using a parallel projection while a 3D object to which the titles are associated is output using a perspective transformation.

Support for Hidden Line and Hidden Surface calculations is provided at the workstation level. Associated with primitives is an attribute defining which method of rendering is to be used on the workstation. The workstation can be asked to render or not and it has flexibility in how it does the rendering. Consequently, a variety of workstations can choose the most appropriate methods depending on their hardware characteristics.

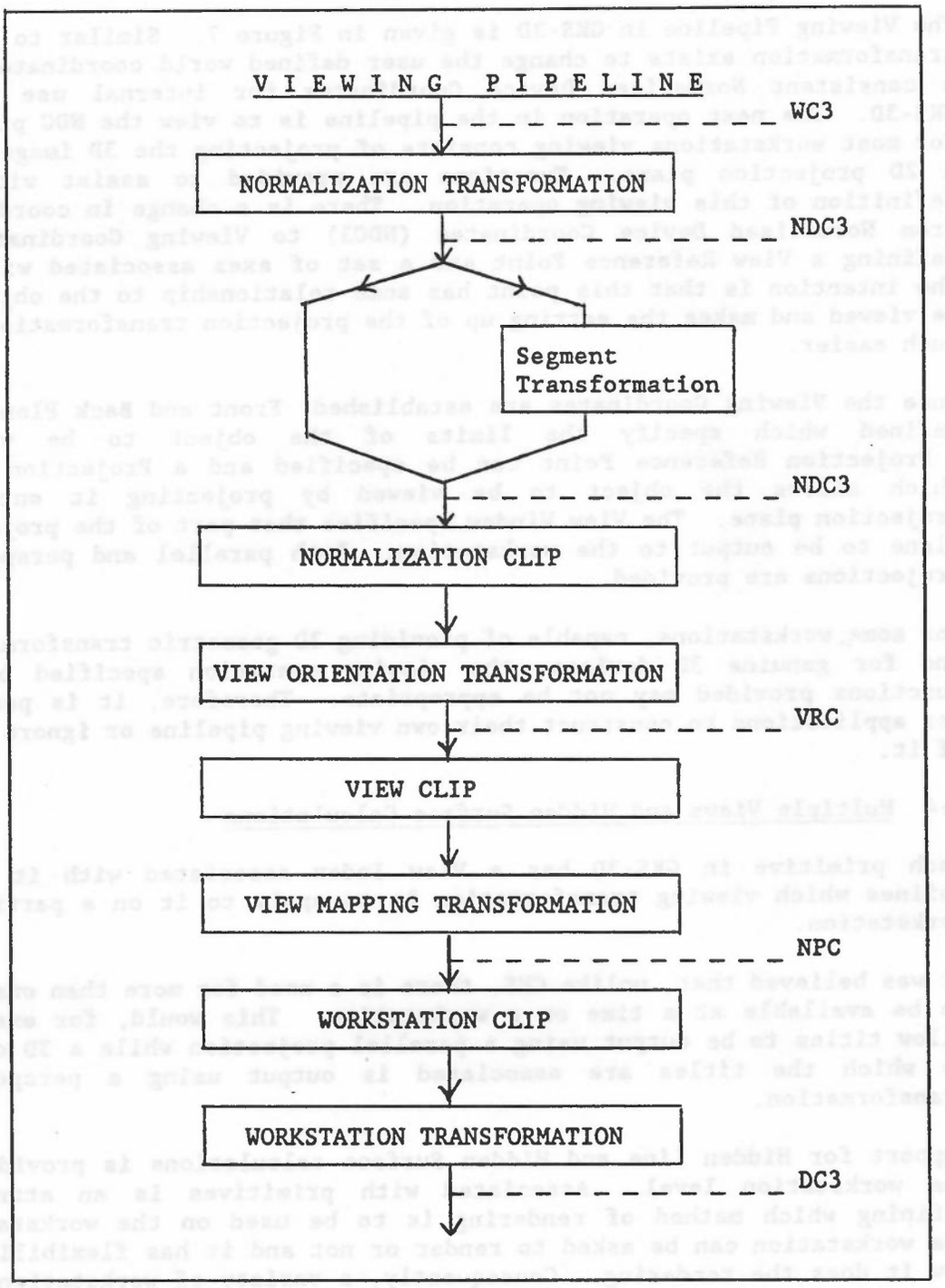

VIEWING PIPELINE
WC3
NORMALIZATION TRANSFORMATION
NDC3
Segment Transformation
NDC3
NORMALIZATION CLIP
VIEW ORIENTATION TRANSFORMATION
VRC
VIEW CLIP
VIEW MAPPING TRANSFORMATION
NPC
WORKSTATION CLIP
WORKSTATION TRANSFORMATION
DC3

Figure 7

5. PHIGS

5.1 Introduction

A major problem with GKS-3D is that it does not cater for the most sophisticated end of the device market nor does it adequately handle the application where there is a need to make rapid changes to the complex hierarchical pictures often found in some areas of CAD.

The problem is not so much with the primitive definitions, attribute models or viewing transformations but with the segmentation facility which does not allow the application database and graphical picture structure to be closely integrated. In fact, the one-level segmentation facility in GKS forces any hierarchical filestore to be built on top within the application program. However, by doing this, it makes it almost impossible to use hierarchical segmentation facilities if they are available in the device.

The solution, in terms of standardisation activities, has been to extend GKS in two different directions. The first is GKS-3D and the second is the Programmers Hierarchical Interactive Graphics System (PHIGS).

5.2 Main Features

The primitives and attributes in PHIGS closely follow those in GKS-2D and GKS-3D. The viewing models in GKS-3D and PHIGS are also very close. The one major difference between PHIGS and GKS is that in PHIGS the creation and display of a picture are very explicitly independent phases. (It is worth noting that PHIGS and GKS are not as far apart as it might at first appear, though this point cannot be developed here.)

At the heart of PHIGS is a single structure store. A structure consists of a number of structure elements which can be both graphical and non-graphical. Thus it is possible to keep application data associated with graphics in the same database. The PHIGS structure store replaces the GKS segmentation facility, though the two do not occur at the same point in the viewing pipeline.

Structures are not displayed on workstations until they are posted to workstations. Posting a structure to a workstation causes the structure to be traversed, generating graphical output for display on the workstation. Structure elements representing application data are ignored by traversal. Once a structure has been posted, changes made to the structure are reflected on the workstation until the structure is unposted. Unposting a structure does not cause it to be removed from the central structure store.

Particular features of the structure facility are:

(1) Hierarchy: structures can call other substructures and the same substructure may be called more than once from a higher level. Thus a car may need only a single wheel substructure which is called four times.

(2) Modelling Coordinates: structure elements contain positional information in modelling coordinates. Each structure has a global and local modelling transformation which are concatenated to produce

the transformation to be applied to points to turn the modelling coordinates into the coordinates to be passed to the viewing pipeline.

(3) Inheritance: substructures inherit attributes from the calling structure. Thus, the global modelling transformation is the one passed in by the calling structure. Similarly, attributes such as colour can be passed to the substructure.

On completion of traversing a structure, control reverts to the higher structure that called it and the attributes are reset to those in force on entry to the substructure. Thus the substructure can have no effect on the calling structure.

(4) Editing: labels can be placed in structures and there is a structure element pointer. Consequently, it is possible to move around a structure and edit it after initial creation. This is unlike GKS segments which cannot be changed once the segment is created.

For high quality displays, it is possible for structure traversal to be done by the hardware in the workstation allowing fast graphical movement of complex pictures in 3 dimensions.

5.3 Implementations of GKS-3D and PHIGS

Only two implementations of GKS-3D are commercially available at the time of writing (as far as we know):

(1) GKSGRAL-3D: this is available from GTS-GRAL in Darmstadt. It is upward compatible with the company's GKSGRAL 2D system. The implementation does include optional modules to support hidden line/hidden surface calculations. Only a FORTRAN binding is available. The system runs on a range of hardware from PCs to host mainframes. The current implementation is to Level 2b. A number of drivers are available for standard terminals.

(2) CMC: CMC Ltd of Bombay, India have a GKS-3D implementation which in the UK is being sold through device manufacturers. The product has a FORTRAN binding which predates the ISO DP and the company has indicated that it will change the FORTRAN binding to agree with the ISO one when it becomes stable.

With the current early stage of PHIGS in the standardisation process, it is too early to talk about commercially available implementations. However, a pilot implementation has been produced by IBM and Rensselaer Polytechnic Institute. Details of this implementation are given in [8].

A number of manufacturers have indicated that their existing computer graphics software is PHIGS compatible (for example, Apollo and Megatek). We have been unable to assess these products to see how closely they coincide with the PHIGS ISO working draft. As the Apollo one is based on their current graphics system which has significant differences from PHIGS, it is unlikely that full compatibility can be achieved either with the current working draft or the future standard.

6. LANGUAGE BINDINGS

6.1 Introduction

In the early days of ISO work on graphics standards, it was realised that the different languages from which people used graphics made it necessary to define the functional specification in a way that was independent of any one language. For this reason, the GKS document does not specify the interface from FORTRAN, Pascal, or any other language. Instead, a separate ISO standardisation activity has been set up to cover all the language bindings for GKS, GKS-3D and PHIGS.

The status of the various language bindings in progress is given in section 9.

6.2 FORTRAN

The language binding gives FORTRAN representations of the data types of GKS and FORTRAN subroutine names for the GKS functions. In most cases, there is a one to one correspondence between the functions of GKS and FORTRAN subroutines. The exceptions to this will be discussed shortly.

The GKS functional description uses very long function names. In the GKS FORTRAN language binding these function names are mapped to subroutine names which all begin with the letter 'G', thus leaving only 5 characters in which to encode the rest of the name. After much discussion a complex algorithm was devised which ensures that the mapping of names is consistent. The algorithm is documented in the language binding.

In general, the order of GKS function parameters is preserved in the order of subroutine parameters, though it is sometimes necessary to insert extra parameters into the normal sequence, for example the lengths of arrays for output parameters.

Some of the data type mappings are worthy of comment. GKS uses a data type 'point' to describe points represented by (x, y) coordinates, and lists of points to describe the parameters of the polyline, polymarker and fill area primitives. Lists of points in the FORTRAN language binding are represented as a pair of arrays containing x and y coordinates respectively. This allows a natural extension to GKS-3D where points in 3-space are represented by three one dimensional arrays.

Character strings are represented by the CHARACTER*(*) data type.

The GKS cell array function requires an array of colour index values as one of its arguments. The FORTRAN language binding allows this array to be specified as a slice of a larger array. The relevant parameters to the subroutine are:

```
INTEGER DIMX, DIMY              the dimensions of COLIA which contains the
                                cell array
INTEGER SX, SY                  indices of start column, start row
INTEGER DX, DY                  number of columns, number of rows
INTEGER COLIA(DIMX, DIMY)       colour index array
```

The way this works is shown in Figure 8.

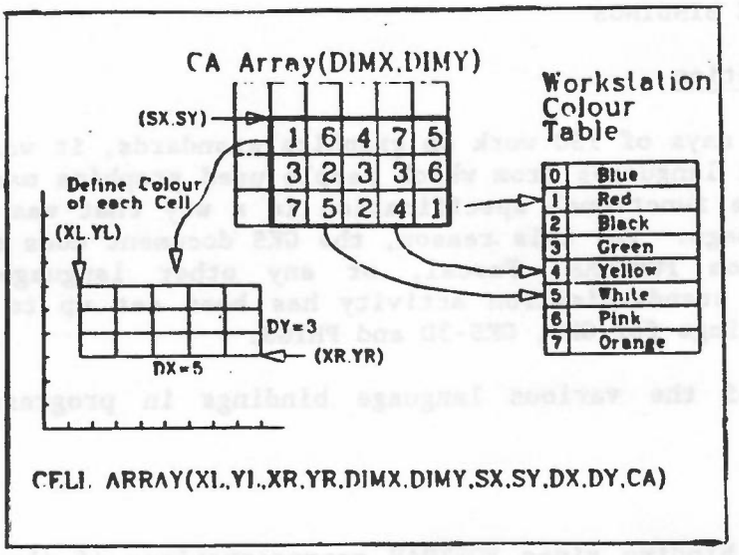


Figure 8

The GKS enumeration types (for example NOCLIP, CLIP) are mapped to FORTRAN integers. A method for mapping the enumeration types to variable names is also given. It is expected that the enumeration types would be defined in PARAMETER or DATA statements.

The convention that GKS functions correspond one for one with FORTRAN subroutines is not followed for certain inquiry functions, in particular functions which return a list of values of variable length and functions which return large amounts of information of heterogeneous types.

An example of the first type of function is INQUIRE SET OF SEGMENT NAMES IN USE. The corresponding function in the language binding is:

```
    SUBROUTINE GQSGUS(N, ERRIND, OL, SGNA)

Input parameters
     INTEGER N        set member required

Output parameters
     INTEGER ERRIND   error indicator
     INTEGER OL       number of segment names
     INTEGER SGNA     Nth member of set of segment names in use
```

An example of a function returning a large amount of heterogeneous data is the function INQUIRE CURRENT INDIVIDUAL ATTRIBUTE VALUES. This is broken down in the FORTRAN binding into one subroutine per individual attribute. Only inquiry functions are bound in this way.

The ISO FORTRAN standard also defines a subset of the language. In order to accommodate the subset in the binding, some alternative bindings have to be given. Subroutines that have arguments of type CHARACTER*(*) have alternative definitions for the subset that include fixed length character strings, CHARACTER*80 and in some cases an additional parameter, the number of characters in the string.

A full FORTRAN implementation must also provide the subset subroutines, and so an application written for the subset language binding will also run on a full implementation.

6.3 Pascal

The Pascal binding follows similar principles to the FORTRAN binding. GKS functions are mapped to Pascal procedures.

The naming restrictions in Pascal are not so severe as those in FORTRAN. Abbreviations of long names is still necessary, however, and a common list of abbreviations was drawn up. This list is also used in other language bindings, for example Ada, to minimise divergence between bindings.

Lists of points in the Pascal binding are represented by the data type:

```
type GRpoint = record
                 x, y: REAL
               end

array [min..max : INTEGER] of GRpoint
```

The GKS enumeration types are represented by Pascal enumerated types.

GKS functions which are actually variants of one underlying function (for example, SET POLYLINE REPRESENTATION, SET POLYMARKER REPRESENTATION ...) are grouped together in a single Pascal procedure, for example:

```
GSetPrimRep(Polyline, WSid, 2, rep)
```

An enumerated type is used to select the required GKS function. In this example a representation is set for polyline index 2 on workstation WSid, using the values in the record 'rep'. This device considerably reduces the number of procedures that a programmer has to know about.

The ISO Pascal standard defines two levels of Pascal, 1 and 0. In level 1 Pascal, it is possible to pass arrays of different lengths on different calls to the same procedure. In Pascal level 0, procedures only accept arrays of a declared fixed size. ANSI Pascal is equivalent to ISO Pascal level 0. In consequence, additional procedures and data types have been defined for level 0 Pascal. These procedures and data types are also available in level 1 Pascal.

6.4 Ada

The proposed Ada binding follows similar principles to the Pascal binding. The binding defines an Ada package called GKS. Procedure names are prefixed by GKS, for example GKS.Polyline, unless a 'with' statement is used.

Ada includes multi-tasking. The Ada binding allows an implementation to refuse access from multiple tasks, but also allows the implementation to permit such access so long as concurrent access to GKS functions does not take place.

6.5 C

There is currently no standardised version of the C language, and under ISO rules it is only possible to standardise a language binding for languages which are defined in 'referenceable documents'. In practice this means languages defined as ISO or national standards. An ANSI standard for C is under development and when this is completed, formal work in ISO to standardise a C binding can commence. This will be based on work within ANSI on a C binding.

6.6 Future Bindings

One very important decision has been made. Bindings for future graphics standards will be based on the GKS bindings. This means that once the major problems have been resolved for a language binding of GKS, the same solutions will be used in subsequent bindings. A common set of guidelines for language bindings has been produced as a result of some 4 years hard work. Language bindings are by no means as trivial as they might at first sight appear.

7. GRAPHICS INTERCHANGE

7.1 Introduction

GKS provides a standard for graphics in two dimensions (both input and output). The philosophy in GKS is that the operations requested by the application are for almost immediate action. The segmentation facility provides an on-line method of storage of transient graphical information but is not designed for long-term storage between sessions. Once the workstation is closed, the segment store ceases to exist.

GKS recognised the need for storage of graphical information between sessions and initially included within it a GKS Metafile facility as part of the standard which allowed an audit trail of GKS commands (used to create and manipulate pictures) to be stored and later retrieved and executed.

Once it became clear that there was likely to be more than one graphics standard at the functional level and all would have a need for long term storage and retrieval, it was decided to separate out the metafile function as a separate standard. That standard activity is the Computer Graphics Metafile (CGM).

The final GKS standard retains a set of functions for reading and writing metafiles. The intention is that these functions could be used to read and write CGM metafiles. However, there is also a need to provide more specific metafile facilities specifically for the GKS environment. GKS provides an Annex to the standard where a protocol is defined for communication in the GKS environment. This protocol will provide an audit trail as described above. The Annex is not an intrinsic part of the standard but, if present, will allow communication between GKS systems or long-term storage and auditing within a GKS system. It has greater functionality than CGM in the area of segmentation. The GKS metafile, for example, can be used to store a set of segments. A later use of GKS could read in these predefined segments. On the other hand, CGM is much more a facility for picture storage.

The GKS metafile looks very much like a workstation. Once the special workstation defined as a metafile is opened, any graphical commands obeyed are stored in the metafile. This continues until the metafile is closed. This similarity between a workstation and metafile implies that there is a close relationship between the protocol used to define the metafile and that required to define the interface between GKS and the virtual device. As shown in Figure 2, the standards activity to provide an interface to the graphical device is CGI, the Computer Graphics Interface. In this section, we will show the inter-relationship between CGM and CGI.

7.2 Computer Graphics Metafile - CGM

The Computer Graphics Metafile (CGM) is a file of more or less device independent graphics orders. It provides a standard for:

(1) Retaining and transporting graphical data defined as pictures.
(2) A data interface for graphics packages.
(3) A picture transfer mechanism between different devices, installations and systems.

CGM is defined as being compatible with GKS but allows a wider range of functionality so that it can be used for interchange of graphical information in a wider context than just GKS. A key element in the philosophy is that the process creating the information in the CGM can be separated in time and space from the process using it. Thus CGM could be used to generate a magnetic tape to be read at a remote installation many weeks later using a different type of graphics system from the one that generated it.

The elements making up the CGM are broadly split into seven classes:

(1) Description Elements: these elements specify the version of the CGM used in defining the file and information concerning the capabilities of the process needed to read the CGM.

(2) Control Elements: these elements define the size and orientation of the space in which the CGM is defined.

(3) Picture Descriptor Elements: the CGM provides more flexibility in some areas than GKS. For example, line width in GKS is specified by giving the width of the line as a factor of the standard line width on the specified device. The CGM allows as an alternative the width to be specified in virtual device coordinates. The descriptor elements declare the modes in use for this particular CGM.

(4) Graphical Elements: these describe the visual components of the picture being transferred. They include the GKS output primitives as a subset but also include elements for the efficient transfer of circles and arcs.

(5) Attribute Elements: these specify the attributes of graphical elements and are equivalent to the GKS attribute model.

(6) Escape Elements: these describe device or system dependent elements where no constraint is placed on the contents.

(7) External Elements: these elements are used to include relevant messages and application data not directly related to the graphical image of the picture.

The CGM is effectively transporting a virtual picture and, consequently, defines all picture elements in Virtual Device Coordinates which are closely coupled to Normalised Device Coordinates.

The CGM description includes the coding of how the information is formatted. The CGM standard document now consists of four parts. The first contains the functional specification of any conforming metafile. The other three parts contain specifications of three methods of encoding, each with its own particular goal:

(1) Character Encoding: this is intended for use where it is important to minimise the size of the metafile; where necessary, this is regarded as more important than processing speed. This encoding makes it suitable for transmission through 'ASCII' networks.

(2) Binary Encoding: this encoding aims to minimise the processor effort required to generate and/or interpret the metafile. It is therefore highly suitable for storage and retrieval of graphical data within a computer system.

(3) Clear Text Encoding: this encoding is aimed at the requirement of having a metafile that can be read and edited by people. It is also very safe to transport, even between systems with different native character sets.

In addition, the document allows private encodings as long as they conform to the Functional Description and general rules of conformance in Part 1. An example of this would be a binary encoding using word lengths or representations other than those specified in Part 3.

7.3 Computer Graphics Interface - CGI

There is a close affinity between the CGM standard and the Computer Graphics Interface (CGI) standard. This is not surprising as both are seeking to describe pictures by a linear sequence of commands.

The CGI defines the interface between the device independent and device dependent parts of a graphics package. Unlike the CGM standard which defines the output from a graphics package for transmission or storage, the CGI standard must handle both output and input and it is assumed that the device is on-line and capable of supporting dynamic interactive graphics.

The CGI has to support a whole range of terminals from simple plotters to high powered interactive terminals. To do this in a sensible way, physical devices are required to support a minimum set of functions such as line drawing and may support, but are not required to, a richer set of functions including functions such as arc and circle generation.

The device driver on the device side of the CGI will generate the necessary device codes for the required functions and will either emulate the non-required functions in terms of the required ones if the device does not support them or uses the device functions if it does.

The CGI supports segmentation and input while the CGM does not. Consequently, the set of elements defined in the CGM is augmented by input elements and the picture description elements including segmentation. The CGI, however, is not a superset of the CGM as the Descriptor Elements are not provided in the CGI. Instead, interrogation of the device by the graphics system establishes the attributes of a device prior to a session starting.

At this point in time, there is also a major difference in that the CGI contains a set of commands specific to the control of raster devices. As well as cell array, CGI also includes a more device dependent primitive called pixel array which performs functions on the actual physical pixels of the device. The BITBLT operation is provided for moving, copying and constructing bit maps.

The CGI includes a very large and complex set of functions (the current document is 536 pages long). Many of the functions are inappropriate for many devices. Consequently, the proposal is structured in terms of option sets which provide access to a particular facility (for example, colour and segmentation are two option sets). There are currently 7 option sets. The intention is that particular device classes will correspond to some subset of the option sets.

7.4 IGES

7.4.1 Background

IGES stands for 'Initial Graphics Exchange Specification'. It is a standard for the transfer of CAD/CAM information. IGES is thus a standard for the exchange of application data in contrast to CGM which is a standard for the exchange of graphical data. The two standards thus address very different requirements.

IGES was developed initially as a format for the transmission of representations of 2D engineering drawings between dissimilar CAD/CAM systms, motivated by a very real practical problem being faced by many large companies. However, since the development of IGES started, there have been important changes in the way engineering parts are represented by CAD/CAM systems. 2D draughting systems have given way to 3D wireframe systems, including the capability for defining complex free-form surfaces. The emerging generation of CAD/CAM systems are based on solid modellers. The concern now is with complete product models, which will contain not only the geometry of the part, but also a great deal of other information, for example manufacturing information. IGES is evolving into a standard for product definition data exchange.

The development of IGES started in the US in the late 1970's under the auspices of the National Bureau of Standards. The first version was based on a data exchange format developed by Boeing in the late 1960's which was known to work. The format was finalised in 1980 and became an ANSI standard in 1981. It is incorporated in ANSI Y14.26M 'Digital Representation for Communication of Product Definition Data'. This is IGES Version 1. Versions 2 (1982) and 3 (1985) incorporate various improvements, for example better facilities for representing surfaces. Version 4, under development, is looking at incorporating solid modelling information.

7.4.2 Overview of IGES

The minimum requirement for a standard for the transmission of product definition data is that it should be able to transmit geometric data, annotation and organisational information. IGES treats a product definition as a file of entities, each of which is represented in an application-independent format, to and from which the representations of specific CAD/CAM systems can be mapped. Three types of entity can be transmitted; geometrical (lines, arcs, spline curves etc used to compose the model), annotation (dimensions, construction lines, arrows and textual notes such as appear on an engineering drawing) and structural.

Because of its origins, IGES (and its successors) is based on an 80-column card image format. A binary format has been introduced more recently which reduces the data storage requirements of an IGES file by at least 50%.

An IGES file consists of 5 sections.

(1) User comments. This section is meant to provide a human readable introduction to the file.

(2) Global section. This section contains global information necessary for interpreting the file, for example the units used in the file, the characters representing certain delimiter functions.

(3) Directory Entry section. This is effectively a dictionary of all the entities transmitted in the file.

(4) Parameter Data section. This section contains the data defining each entity. Pointers link each directory entry with its corresponding parameter data and vice versa. Pointers are in fact card sequence numbers!

(5) Terminator card.

An additional Binary information section precedes a binary IGES file. This contains information concerning precision and the length of each section following.

7.4.3 The Future of IGES

Initially IGES was slow to gain acceptance, largely because CAD/CAM vendors were not committed to the concept. Vendors were more keen to see IGES files translated to their system than the other way round, consequently preprocessor-translators were often in advance of the corresponding postprocessors. However, increasing pressure from customers has lead to major improvements over the last two years. There are some incompatibility problems which remain, notably in the area of free-form curves and surfaces.

IGES is being used by a number of large organisations and certainly goes a good part of the way to solving the problem. There is no better alternative at this point in time.

The developers of IGES are currently developing a proposal called PDES (Product Data Exchange Specification) which builds on the IGES experience

and will be much more forward looking. Two notable influences on the design of PDES are XBF (Experimental Boundary File, developed by the international organisation CAM-I, for the transmission of solid modelling data) and PDDI (Product Definition Data Interface developed under the US Air Force ICAM program).

The French company Aerospatiale have developed a format called SET (Systeme d'Echange et de Transfer) which it is claimed gives 5 or 6-fold improved translation times over IGES.

ISO technical committee 184 now has a working group in this area developing a standard called STEP (Standard for Transfer and Exchange of Product Model Data) which is attempting to merge the above work into one common approach. This work is unlikely to result in an international standard before 1990.

8. REGISTRATION

During the evolution of all these standards it has become obvious that it is not possible to standardise everything at once. In particular, there are a number of graphical elements that can be found in a bewildering number of varieties. An example would be the set of all 'useful' marker types. Any one person could probably think of a hundred or more: a specialist could probably think up a thousand in just his own field.

Rather than delay the standards in progress by trying to get agreement on extensive lists of such elements for each standard in turn, the documents now just refer to a single registration mechanism and mandate only a very small number of such elements. The registration mechanism is being set up to deal with the standardisation of the following elements initially.

- Generalised Drawing Primitives (GDPs)
- Escapes
- Line types
- Marker types
- Hatch styles
- Text font usage
- Prompt/Echo types
- Error messages

This registration mechanism will, by default, provide for the extension of each relevant graphics standard in each of these areas. Thus for an extra marker type, it will define the appearance of the marker, allocate a marker type number for GKS, do the same for the CGM (including each of the bindings) and so on for each standard. In some cases, the element will not be appropriate (for example, Prompt/Echo types are not relevant to the CGM).

By this means, it is expected that the requirement for extensions to the standards in these areas will be met - without having to update each standard - and also that all the standards will stay in step with minimum effort and confusion.

The US National Bureau of Standards has been approved as the Registration Authority for the Register of Graphical Items. However, the precise

procedures to be followed by the Registration Authority are still being defined.

9. STATUS OF GRAPHICS STANDARDS

Only GKS has so far reached full International Standard status. The following table gives our best guess (in April 1986) as to when each project will reach its next stage in the ISO pipeline. By their very nature, such dates are approximate as they depend on the results of ISO ballots. Any decision to have a second DP or DIS Ballot will increase the timescales. As each DP Ballot is out for voting for three months and DIS Ballots for six months, any second iteration will cause a significant delay.

Once a project has reached DIS stage, it can be regarded as reasonably complete and unlikely to change in significant ways. At that stage, products could be based on the standard proposal with only minor retrofitting being necessary. On the other hand, if a proposal has only reached the DP stage, it is quite likely that significant changes will appear before it reaches DIS status.

From the dates below, it can be seen that GKS and the FORTRAN and Pascal Bindings are almost complete. GKS-3D and CGM are getting to a stage where major changes will not occur in the future. It should be stressed that both CGI and PHIGS are at a very early stage of standardisation and will inevitably have changes made to them before they become Draft International Standards.

Project	Ref Doc	Status	Expected Progress		
			DP	DIS	IS
GKS	ISO7942	Published 15 Aug 1985	-	-	-
GKS Language Bindings					
FORTRAN	DP8651/1	DIS Ballot expected	-	May 86	Aug 87
Pascal	DP8651/2	DIS Ballot started Feb 1986	-	Feb 86	May 87
ADA	DP8651/3	Second DP Ballot completed	-	Oct 86	1988
C		Working draft in circulation	Sep 86?		
GKS-3D	DP8805	Second DP Ballot completed		Sep 86	1988
GKS-3D Language Bindings					
FORTRAN	-	Working Draft	Jul 86		
Pascal	-	Planned			
ADA	-	Planned			
C	-	Planned			
PHIGS	SC21 N819	Working Draft	Dec 86	1988	
PHIGS Language Bindings	)				
FORTRAN	)	Approved as Work	1987		
Pascal	)	Item			
ADA	)				
CGM					
Functional Description	DP8632/1	DIS Ballot started		Feb 86	
Character Encoding	DP8632/2	Feb 1986		"	
Binary Encoding	DP8632/3	"		"	
Clear Text Encoding	DP8632/4	"		"	
CGI	SC21/WG2 N356	Working Draft Available	Dec 86	1988	

The dates given for expected progress are the dates at which the DP and DIS ballots start. A minimum of 15 months delay is likely between the start of the DIS ballot and the publication of the IS.

REFERENCES

[1] R.A.Guedj and H.A.Tucker (eds). 'Methodology in Computer Graphics', North Holland 1979.

[2] P.R.Bono, J.L.Encarnacao, F.R.A.Hopgood and P.J.W.ten Hagen. 'GKS - the First Graphics Standard', IEEE Computer Graphics and Applications, Vol 2, No 5, July 1982.

[3] F.R.A.Hopgood, D.A.Duce, J.R.Gallop and D.C.Sutcliffe. 'Introduction to the Graphical Kernel System GKS', Academic Press (2nd edition) 1986.

[4] 'Information processing systems - Computer graphics - Graphical Kernel System (GKS) functional description', ISO 7942, ISO Central Secretariat, Geneva, August 1985.

[5] G.Enderle, K.Kansy and G.Pfaff. 'Computer Graphics Programming, GKS - The Graphics Standard'. Springer-Verlag 1984.

[6] D.Hearn and M.P.Baker. 'Computer Graphics'. Prentice-Hall 1986.

[7] 'Digital Representation for Communication of Product Definition Data'. ANSI Y14.26M - 1981.

[8] S.S.Abi-Ezzi and A.J.Bunshaft. 'An Implementer's View of PHIGS', IEEE Computer Graphics and Applications, Vol 6 No 2, Feb 1986.

cb/M9/GS

PROFESSIONAL SEMINAR

COMPUTER GRAPHICS PROGRAMMING

F R A HOPGOOD
D A DUCE

Informatics Division
Rutherford Appleton Laboratory (RAL), UK

1. PRINCIPLES OF COMPUTER GRAPHICS AND THE GRAPHICAL KERNEL SYSTEM

1.1 Introduction

In many areas of computer science, standardisation is an activity which is aimed at formalising current practice. However, it is often the case that the act of standardisation reveals flaws in what was assumed to be a well understood area. This leads to a rethinking and strengthening of the original methodology.

To a large extent, computer graphics fits this picture. At the start of the graphics standardisation work, most people believed that the basic methodology was well understood yet, during the standardisation process, many changes to the methodology have been accepted.

This section gives an overview of the major concepts which any graphics system has to embody and discusses how they have been incorporated into the first ISO standard for computer graphics programming, the Graphical Kernel System (GKS). Later sections review how these concepts are addressed by other emerging graphics standards.

1.2 Principles of Computer Graphics

There are a number of fundamental topics that must be addressed by any computer graphics system. The most important of these are listed below.

(1) Output Primitives. The primary purpose of generative computer graphics is to generate a picture on the display surface of some graphics device. Graphical output is defined in terms of basic building blocks which we will call output primitives. Points, lines and areas are examples of output primitives which may or may not be included in any particular graphics system. The questions facing a graphics standard is what is the fundamental set of output primitives, and at what level of abstraction are they defined. Clearly such a set must be rich enough to cover all possible graphical pictures in the field of application of the standard.

(2) Primitive Aspects. When an output primitive is displayed on a graphics device it will have a particular appearance, for example lines may be displayed with a particular colour, and solid or

dotted. Those facets of a primitive which determine its appearance when displayed are termed the aspects of the primitive. In the example given, colour and linestyle would be aspects. A standard needs to decide what the aspects should be for each type of output primitive. The number of aspects recognised governs the flexibility given to the programmer to control the appearance of pictures.

(3) Coordinates. It is accepted that the user of a graphics system should be able to define graphics in a coordinate system relevant to the application. It is also accepted that this coordinate system is unlikely to be the one specified for the device by the manufacturer. As a result, there is a need to transform user coordinates to device coordinates. The problem that arises is what kinds of user coordinate system should be supported (for example, cartesian, polar, spherical polar), and how should the mapping to device coordinates be specified.

(4) Input Primitives. Interactive programs need to be able to accept graphical and non-graphical input. These data correspond to a number of different types, for example coordinate positions, character strings for labelling and values for identifying groups of primitives. The different types of input data are termed input primitives. Questions facing a graphics standard include what kinds of input primitive should be provided, should all input be handled by the graphics system or just graphical input? Should graphical and non-graphical input be differentiated?

(5) Input Model. An input model describes how input primitives are related to physical input devices, the degree of control provided to the application (for example selection of the way in which input values are echoed to the device's operator), and the styles of interaction available to an application program (for example, can the operator generate input values asynchronously from the execution of the program?). What kind of input model should a graphics standard support?

(6) Device Independence. Most high level standards are defined so that the user can be isolated from specific characteristics of the graphics output and input devices being used and can concentrate on defining graphical output and input in a device independent way. It might be deemed desirable to be able to move an application from one device environment to another without having to make major changes to the program structure. Issues that arise in connection with device independence include how is the user to be insulated from the details of device hardware whilst having some degree of control over the appearance of pictures on devices with differing capabilites (for example colour and monochrome), how is the transformation from user to device coordinates to be specified so that different device coordinate systems can be accommodated? How can the user retain some degree of control over the mapping from abstract input and output to physical input and output so that the devices can be used in an optimal way?

1.3 Principles of GKS

1.3.1 Introduction

This section examines how the principles of computer graphics are realised in the graphical kernel system. Other graphics standards under development embody these principles in very similar ways. It is appropriate to examine GKS first, since GKS is currently the only system that has been all the way through the standardisation process and has been approved and published as an International Standard. A full description of GKS is given in [1,2]. GKS is a 2D graphics system providing both graphical output and input.

1.3.2 Output Primitives

The output primitives in GKS are abstractions from the output primitives typically provided by graphical output devices. GKS defines six output primitives.

(1) polyline: which draws a sequence of connected line segments.

(2) polymarker: which marks a sequence of points with the same symbol.

(3) fill area: which displays a specified area.

(4) text: which draws a string of characters.

(5) cell array: which displays an image composed of cells with specified colours or grey scales.

(6) generalized drawing primitive (GDP): a controlled method of adding more exotic primitives, for example conic arcs and splines.

This choice of primitives calls for some comment.

Many existing graphics packages use the concept of current position. GKS does not. There are several problems with current position, for example should there be one current position or should each type of primitive have its own associated current position; what happens to current position when transformations are changed - there are several answers to these questions, each of which has its merits for particular applications. GKS therefore did not incorporate this concept, though it is interesting to note that current position has been introduced in a controlled way in at least one application oriented layer (see later).

Another notable feature of the GKS output primitives is the absence of relative coordinates. All coordinate data are absolute. Relative coordinates were omitted on the grounds that they could be implemented on top of GKS.

The line drawing primitive in GKS draws a sequence of lines rather than a single line. This choice stems from the observation that many applications deal in sequences of lines rather than single lines and the decomposition to single lines is rather unnatural, and also the

observation that aspects of polylines such as dotted or broken apply to the complete polyline rather than individual line segments which is a more natural view when drawing curves or contours depicted as polylines.

The text primitive is the most complex of the output primitives and aroused the most discussion during the development of GKS. The difficulty with text is that text has a variety of functions in graphics ranging from graphics art quality text which is a part of the picture (for example writing on the side of a building) to low quality text used for messages to the operator of an interactive display. The GKS text primitive attempts to encompass all these varying requirements and provides a rich set of facilities including support for languages not written from left to right.

Implementors of GKS might like to know of the existence of an algorithmic description of the text primitive contained in a paper by Brodlie and Pfaff [3]. They give an algorithm for calculating the starting position of each character in a text string.

The fill area and cell array primitives are abstractions from the kinds of primitive typically provided by raster devices. However it is important to notice that they are abstractions; the primitives are defined in world coordinates rather than device coordinates, and they have well-defined meanings for devices other than raster devices.

The GKS output primitives have a rich set of aspects, allowing a high degree of control over the way primitives are rendered on displays. The aspects of a polyline primitive for example allow control over the linetype (solid, dotted etc), linewidth and colour. The mechanisms by which the values of apsects are determined are described shortly.

1.3.3 Coordinate Systems and Device Independence

The GKS concept of a workstation is the key to device independence in GKS. A workstation consists of zero or one display surfaces and zero or more input devices plus associated software. The GKS idea of a workstation is an abstraction from physical device hardware.

A major difference from many earlier graphics systems is that GKS allows more than one workstation to be in use simultaneously. For example, an operator may be interacting with a design through an interactive display, whilst taking copies of completed parts of the design on a plotter.

The GKS viewing pipeline is shown in Figure 1.

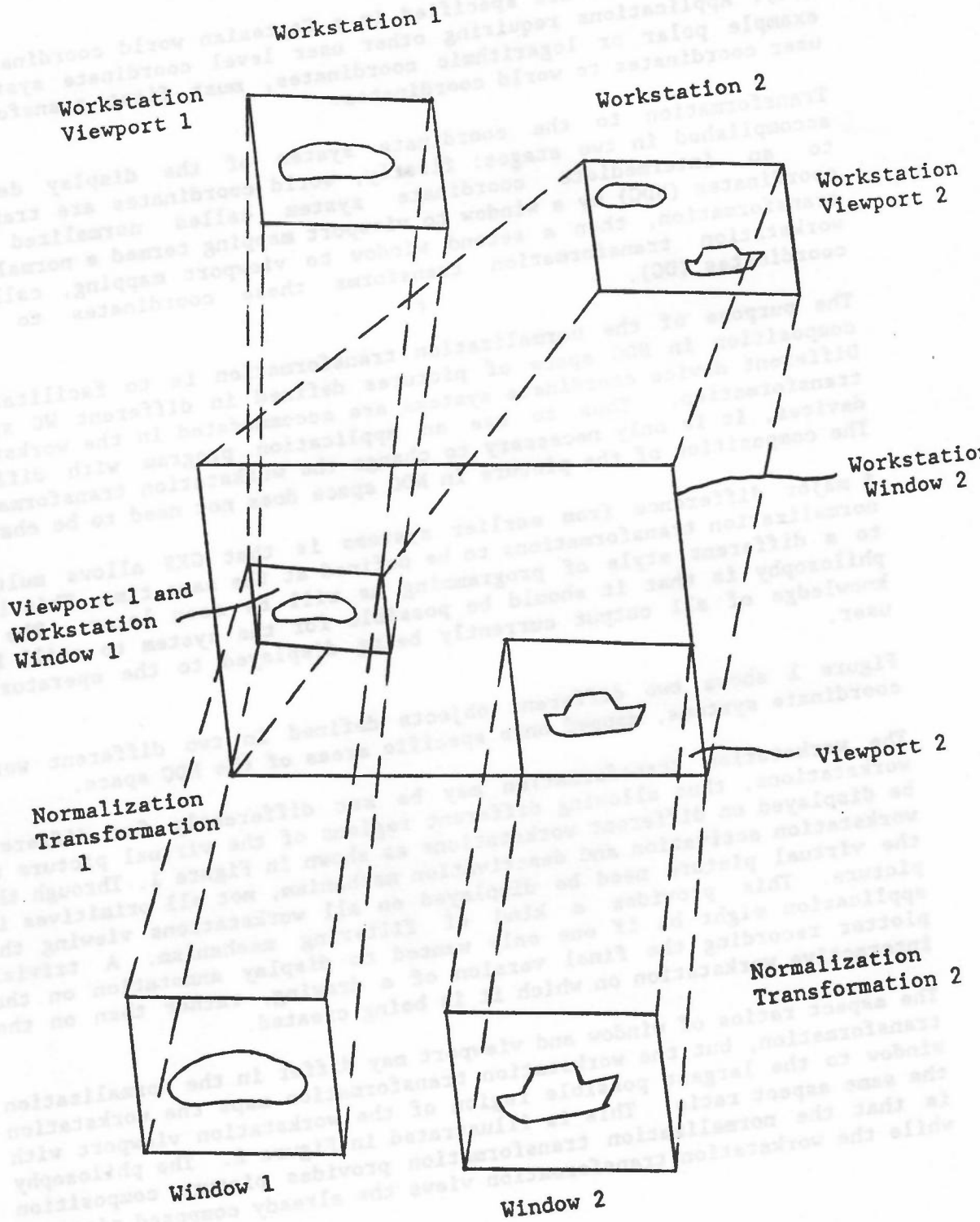

Workstation 1
Workstation Viewport 1
Workstation 2
Workstation Viewport 2
Workstation Window 2
Viewport 1 and Workstation Window 1
Viewport 2
Normalization Transformation 1
Normalization Transformation 2
Window 1
Window 2

Figure 1

Output primitives are specified in a Cartesian world coordinate system (WC). Applications requiring other user level coordinate systems, for example polar or logarithmic coordinates, must first transform these user coordinates to world coordinates.

Transformation to the coordinate system of the display device is accomplished in two stages: firstly, world coordinates are transformed to an intermediate coordinate system called normalized device coordinates (NDC) by a window to viewport mapping termed a normalization transformation, then a second window to viewport mapping, called the workstation transformation transforms these coordinates to device coordinates (DC).

The purpose of the normalization transformation is to facilitate the composition in NDC space of pictures defined in different WC spaces. Different device coordinate systems are accommodated in the workstation transformation. Thus to use an application program with different devices, it is only necessary to change the workstation transformation. The composition of the picture in NDC space does not need to be changed.

A major difference from earlier systems is that GKS allows multiple normalization transformations to be defined at the same time. This leads to a different style of programming as will be seen later. The GKS philosophy is that it should be possible for the system to still have knowledge of all output currently being displayed to the operator or user.

Figure 1 shows two different objects defined in two different world coordinate systems, mapped onto specific areas of the NDC space.

The workstation transformation may be set differently for different workstations, thus allowing different regions of the virtual picture to be displayed on different workstations as shown in Figure 1. Through the workstation activation and deactivation mechanism, not all primitives in the virtual picture need be displayed on all workstations viewing the picture. This provides a kind of filtering mechanism. A trivial application might be if one only wanted to display annotation on the plotter recording the final version of a drawing, rather than on the interactive workstation on which it is being created.

The aspect ratios of window and viewport may differ in the normalization transformation, but the workstation transformation maps the workstation window to the largest possible region of the workstation viewport with the same aspect ratio. This is illustrated in Figure 2. The philosophy is that the normalization transformation provides picture composition while the workstation transformation views the already composed picture.

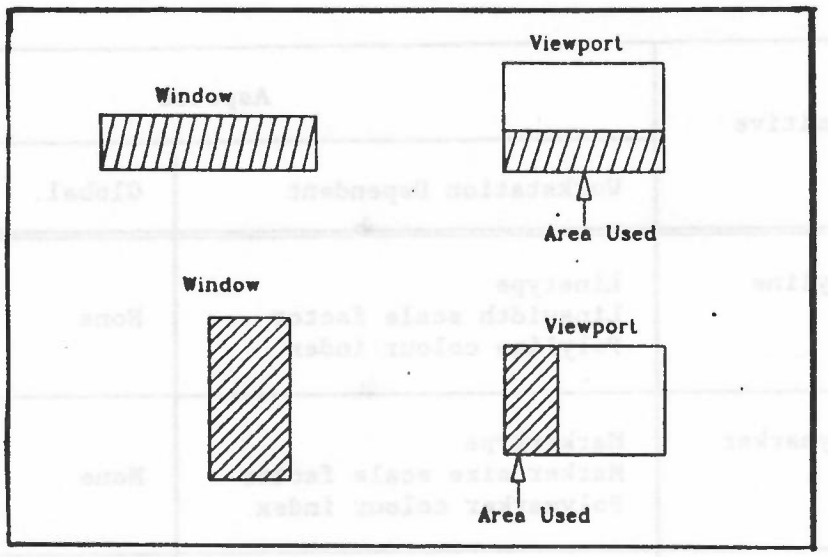


Figure 2

1.3.4 Aspects

The appearance of primitives on the display surface of a workstation is controlled by their aspects. One of the more innovative parts of GKS is the mechanism by which the values of aspects are determined. Earlier graphics packages tended to associate values of aspects directly with primitives. Thus an application would request that a line be drawn in red and it was up to the implementation to determine how red lines would be represented on monochrome devices. The application programmer had no control over this. A consequence of this was that it was difficult to move programs from device to device, because the code controlling the appearance of primitives was typically scattered throughout the program. It was also difficult to write device independent library routines. The solution adopted by GKS is a table-driven approach. Aspects of a primitive in GKS belong to two classes:

(1) Global aspects are applicable on all workstations and take the same value on all workstations on which the primitive is displayed.

(2) Workstation dependent aspects may vary from workstation to workstation so that the same primitive may be displayed quite differently on different workstations.

The aspects of each type of primitive are listed below.

Primitive	Aspects	
	Workstation Dependent	Global
Polyline	Linetype Linewidth scale factor Polyline colour index	None
Polymarker	Markertype Marker size scale factor Polymarker colour index	None
Fill area	Fill area interior style Fill area style index Fill area colour index	Pattern reference point Pattern size
Text	Text font and precision Character expansion factor Character spacing Text colour index	Character height Character up vector Text path Text alignment
Cell array	None	None

The values of aspects are controlled by attributes. For global aspects there is one attribute per aspect.

There are two modes for specifying the workstation dependent aspects of a primitive, bundled specification and individual specification.

Bundled specification uses a lookup table approach. A single attribute per primitive, the primitive index, controls the values of all the workstation dependent aspects of the primitive.

The polyline primitive will be used as an example. The values of all the aspects of a polyline (linetype, linewidth scale factor and polyline colour index) are determined by the value of the polyline index attribute. A polyline index defines a position in a table, the polyline bundle table. Each entry in this table specifies values for all the non-geometric aspects of a polyline. Each workstation has its own bundle tables and so a polyline in the virtual picture may be displayed with different representations on different workstations.

Bundled specification also provides a convenient mechanism for writing library routines. This approach has been used to good effect in the NAG library graphics chapter [4].

GKS does also provide the traditional method of aspect control through the individual specification mechanism. In this case, there is one attribute for each workstation dependent aspect, and thus each aspect has the same value on each workstation on which the primitive is displayed. How each workstation approximates this value is workstation and implementation dependent.

A set of aspect source flags control the mode of specification of each aspect. It is possible to have some aspects specified by a bundle and others individually. The paper by Duce and Fielding [5] looks at these mechanisms from a formal specification viewpoint.

Bundled specification is important when it is necessary to ensure that primitives with different attributes can be differentiated on different workstations; whilst the individual scheme is important when primitives with specific aspects are to be represented on each workstation as closely as possible to the specification.

Each workstation has its own colour table. The aspects of primitives specifying colour (polyline colour index, polymarker colour index etc) all point into this one colour table. The specification of colour is a complex subject. GKS uses the RGB colour model (entries in the colour table define RGB triples). This is an area in which more work is needed in a standards context.

There is another difference between aspects specified by bundles and aspects specified individually, which is worth mentioning briefly. The values of a primitive's attributes are bound to the primitive when it is created, and cannot subsequently be altered. Thus the value of the primitive index associated with a primitive cannot be changed, nor can the value of, say, the linetype attribute. A direct consequence of this is that the values of aspects specified individually cannot subsequently be changed. However bundle table entries can be changed and GKS allows these changes to affect primitives previously as well as subsequently displayed. Thus if a polyline bundle table entry is changed, the workstation is expected to change the aspects of all polylines that use that entry. The mechanisms controlling this are rather complicated and in fact the change is only guaranteed for primitives stored in segments.

1.3.5 Graphical Input

A major innovation in GKS is the model of input. The paper by Rosenthal et al [6] gives a detailed exposition of this model.

The data that can be input to an application program by the operator are divided into six different types and six classes of logical input device are defined corresponding to these six data types. Logical input devices in GKS are characterised by a measure and a trigger. The measure describes the type and value of the input to be returned to the application. The trigger is an event, which for certain styles of input,

determines when the measure value is returned to the application program.

The six logical input data types are:

LOCATOR: a position in world coordinates and the associated number of the normalizaion transformation used to convert back from device coordinates via NDC to world coordinates. The normalization transformation used is that whose viewport contains the data point. Overlapping viewports are dealt with by a priority mechanism; viewport priorities are under the control of the application program.

STROKE: similar to LOCATOR except it represents a sequence of world coordinate positions rather than a single position.

VALUATOR: a real number in some range.

CHOICE: an integer representing a selection from a set of choices.

PICK: the name of a selected segment and an identifier indicating which set of primitives in the segment has been picked.

STRING: a character string.

The idea that different parts of the picture can be defined in different world coordinate systems and input can subsequently be returned to the application program in the right world coordinate system is a very powerful feature of GKS.

Logical input devices may operate in one of three modes:

REQUEST: rather like FORTRAN READ. A request is made by the application program for a measure to be returned from a specified device. GKS waits until the operator has set the measure to the desired value and has activated the trigger.

SAMPLE: the current measure value is returned whenever requested by the application program. The trigger is not used by SAMPLE input.

EVENT: a number of input devices may be active together. Each time the trigger for a particular device is activated, the current measure value and data identifying the device are added to a single queue of input events for all the devices used in event mode. The application program can interrogate the queue to retrieve the input events. It is possible to couple more than one input device to the same trigger so that multiple events can be generated from a single trigger event.

Figure 3 shows the relationship between the measure and trigger for different operating modes.

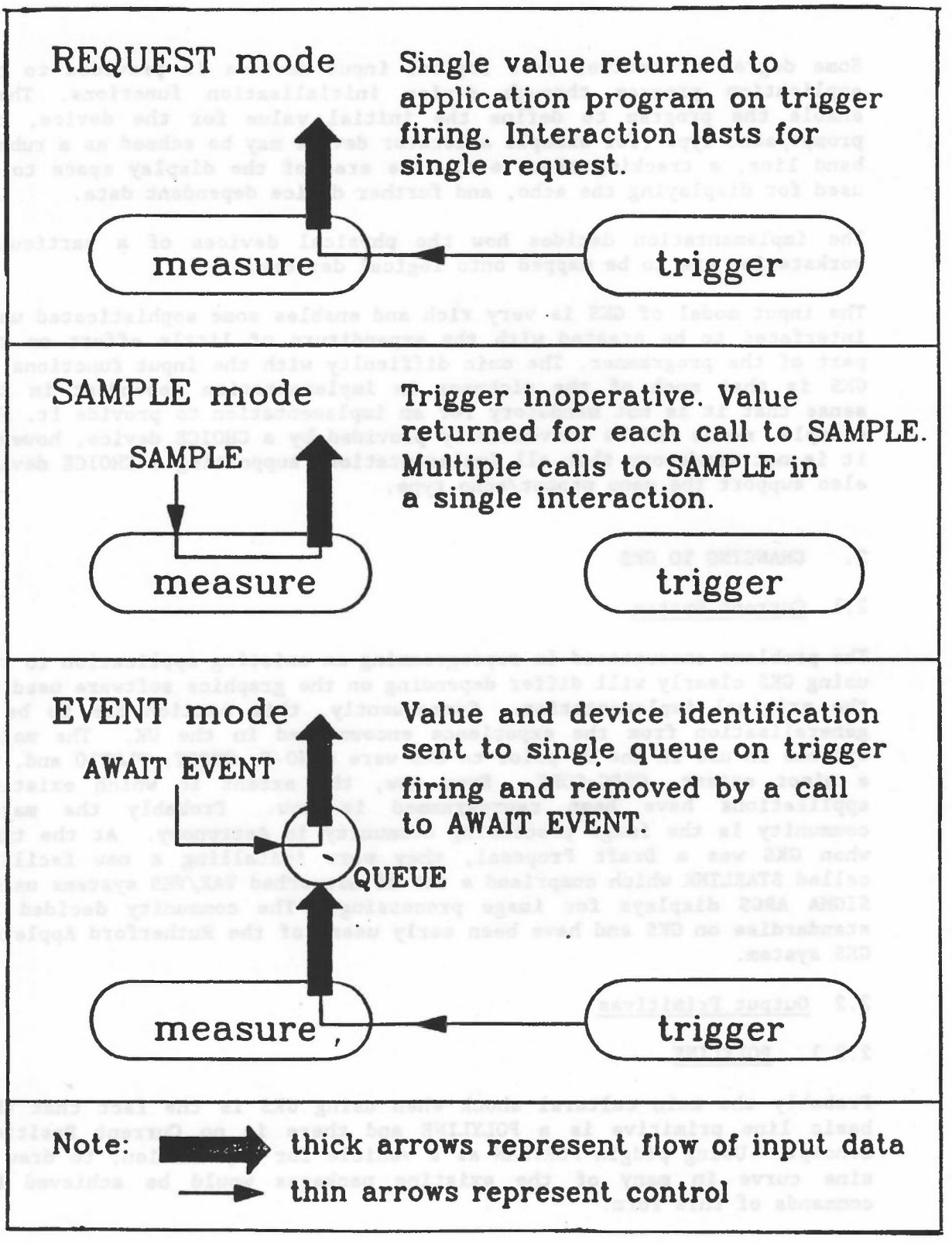

REQUEST mode
Single value returned to application program on trigger firing. Interaction lasts for single request.
measure
trigger
SAMPLE mode
Trigger inoperative. Value returned for each call to SAMPLE. Multiple calls to SAMPLE in a single interaction.
SAMPLE
measure
trigger
EVENT mode
Value and device identification sent to single queue on trigger firing and removed by a call to AWAIT EVENT.
AWAIT EVENT
QUEUE
measure
trigger
Note:
thick arrows represent flow of input data
thin arrows represent control

Figure 3

Some degree of control over logical input devices is provided to the application program through device initialisation functions. These enable the program to define the initial value for the device, the prompt/echo type (for example a locator device may be echoed as a rubber band line, a tracking cross etc), the area of the display space to be used for displaying the echo, and further device dependent data.

The implementation decides how the physical devices of a particular workstation are to be mapped onto logical devices.

The input model of GKS is very rich and enables some sophisticated user interfaces to be created with the expenditure of little effort on the part of the programmer. The main difficulty with the input functions in GKS is that much of the richness is implementation dependent in the sense that it is not mandatory for an implementation to provide it. For example, menus can be conveniently provided by a CHOICE device, however it is not mandatory that all implementations supporting a CHOICE device also support the menu prompt/echo type.

2. CHANGING TO GKS

2.1 Current System

The problems encountered in reprogramming an existing application to run using GKS clearly will differ depending on the graphics software used in the original implementation. Consequently, this section has to be a generalisation from the experience encountered in the UK. The major systems in use in the UK prior to GKS were GINO-F, GHOST, PLOT10 and, to a minor extent, GSPC-CORE. Even now, the extent to which existing applications have been reprogrammed is low. Probably the major community is the image processing community in Astronomy. At the time when GKS was a Draft Proposal, they were installing a new facility called STARLINK which comprised a set of networked VAX/VMS systems using SIGMA ARGS displays for image processing. The community decided to standardise on GKS and have been early users of the Rutherford Appleton GKS system.

2.2 Output Primitives

2.2.1 POLYLINE

Probably the main cultural shock when using GKS is the fact that the basic line primitive is a POLYLINE and there is no Current Position concept. Using pidgin FORTRAN as a vehicle for explanation, to draw a sine curve in many of the existing packages would be achieved by commands of this form:

```
      MOVE ABS (0,0)
      DO 100 J=2,21
      X = (J-1)*0.1*PI
      LINE ABS (X,SIN(X))
100   CONTINUE
```

Similarly, a routine for drawing symmetric axes about a given origin (see Figure 4) might be:

```
      SUBROUTINE AXES (XORIGIN,YORIGIN,DX,DY)
      MOVE ABS (XORIGIN-DX,YORIGIN)
      LINE REL (2*DX,0)
      MOVE REL (-DX,-DY)
      LINE REL (0,2*DY)
      RETURN
      END
```

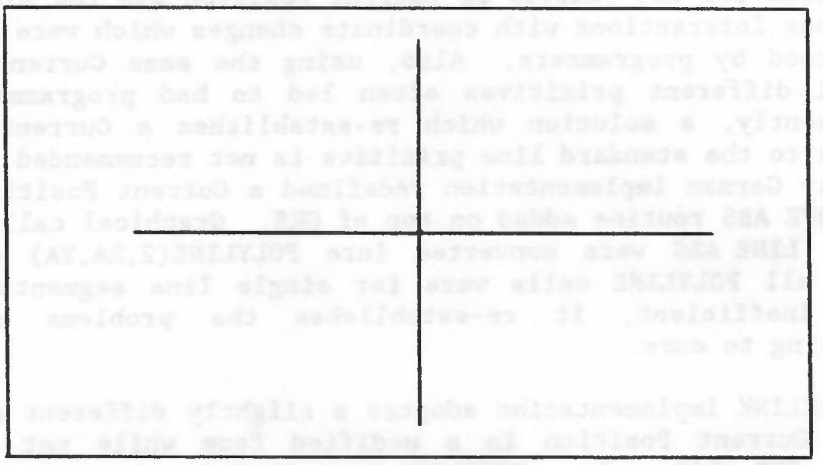


Figure 4

In both examples, the code has to be rewritten in terms of POLYLINEs using only absolute coordinates. The simple change in the first example would be:

```
      REAL   XA(21),YA(21)
      DO 100 J=1,21
      XA(J) = (J-1)*0.1*PI
      YA(J) = SIN(XA(J))
100   CONTINUE

      POLYLINE (21,XA,YA)
```

The major changes are that the user needs to define array storage and the two graphics calls, MOVE ABS and LINE ABS, are replaced by a single POLYLINE call. If the code in the original application is transparent, this tends not to be a problem. However, older programs may well have the MOVE ABS away from the loop and there is then a difficulty in

associating the origin with the rest of the loop. The advantage of the new code is that it is easier to read, the graphics is more clearly differentiated from the data definition and aspect setting for the sine curve as a whole is straightforward. In some existing systems, it would not be possible to produce a dotted line through the sine curve without major changes to the program. Some systems such as GINO-F recognise the need for applying the aspect dotted to the complete set of line segments and provide special facilities which are not particularly attractive either in concept or programming. A disadvantage may be the need to define data storage for the individual points. Normally this is done within the package. The attraction of the GKS approach may be that the user has more control over the size of data arrays used for buffering. A GKS implementation is, for this reason, likely to be smaller than an equivalent GINO-F one.

The major educational change is to remove the concept of Current Position. Unfortunately it has been ingrained into the subconscious of many programmers and it may be difficult to get them to change. The motivation for the removal of Current Position was the view that it had dangerous interactions with coordinate changes which were frequently not understood by programmers. Also, using the same Current Position for several different primitives often led to bad programming practices. Consequently, a solution which re-establishes a Current Position and reverts to the standard line primitive is not recommended. For example, an early German implementation redefined a Current Position established by a MOVE ABS routine added on top of GKS. Graphical calls such as LINE REL or LINE ABS were converted into POLYLINE(2,XA,YA) calls so that almost all POLYLINE calls were for single line segments. Apart from being inefficient, it re-establishes the problems that GKS was attempting to cure.

The STARLINK implementation adopted a slightly different approach which allows Current Position in a modified form while not violating the central GKS philosophy. STARLINK have introduced a set of routines for putting data into arrays:

(1) BEGIN POLY initialises an array and stores the point specified in the first array element.

(2) ADD POLY adds another absolute point to the array incrementing the count of points stored.

(3) OUTPUT POLY outputs a polyline whose length is given by the previous calls of type (1) and (2).

These routines can only be used for creating polylines and have no interaction with other primitives. The values stored in the array are World Coordinate positions and the Current Position is just a useful concept to reduce the number of parameters to routines without having any real graphical significance.

The previous example to generate a sine curve would look like:

```
      BEGIN POLY (0,0)
      DO 100 J=2,21
      X = (J-1)*0.1*PI
      ADD POLY (X,SIN(X))
100   CONTINUE
      OUTPUT POLY
```

Notice the similarity with the original coding although the semantics have changed significantly with the DO loop not generating graphical output any more.

The second example of the AXES subroutine shows GKS in its worst light. The equivalent GKS code would be:

```
      SUBROUTINE AXES (XORIGIN,YORIGIN,DX,DY)
      REAL  XA(2),YA(2)
      XA(1) = XORIGIN-DX
      XA(2) = XORIGIN+DX
      YA(1) = 0
      YA(2) = 0
      POLYLINE (2,XA,YA)
      XA(1) = 0
      XA(2) = 0
      YA(1) = YORIGIN-DY
      YA(2) = YORIGIN+DY
      POLYLINE (2,XA,YA)
      RETURN
      END
```

Variants can be defined which shorten the code somewhat but GKS is not designed for the production of single line segments. An early survey in Germany showed that most real applications tended to be dominated by multi-line output. Consequently, examples such as the AXES routine should be relatively infrequent in real programs. Some implementations have added a single line routine on top of GKS when it has been felt to be appropriate. The danger is that it will be used in the wrong places.

2.2.1 Output Primitives - Fill Area, Text

Most of the existing systems have much less control over area fill and text output than GKS. Consequently, there tends to be an easier 1-1 mapping from earlier systems when the equivalent facilities are provided.

In some systems, area fill is not provided. Consequently, if the facility is used, a considerable amount of existing code may be replaced by a single call.

2.2.3 Cell Array

The Cell Array primitive has been much criticised in the GKS development as not providing the facilities needed and not fitting into the GKS primitive set. However, STARLINK have found [7] that the primitive does

give them what they need and the ability to define it in World Coordinates while still controlling its mapping to the display allows them to overlay contour information easily and efficiently.

One function frequently required by astronomers is to toggle between two images so that the differences can be identified. STARLINK have included ESCAPE functions for this purpose.

2.2.4 GDP

Many of the existing implementations do not provide a rich set of Generalized Drawing Primitives. Those that do have tended to provide arcs, conics, and various curve fitting routines. There is little standardisation and it is probably wise to clearly identify any use of such facilities as they are bound to change once some of the more frequently used examples are registered and assigned specific forms by the GKS Registration Authority.

2.3 Coordinates

Most existing systems provide WINDOW/VIEWPORT mapping as a concept although GINO-F has a more primitive mechanism for converting world coordinates to device coordinates.

A simple method of reprogramming would be to use a single GKS normalization transformation in place of the existing system. None of the systems in common use have multiple normalization transformations. This may not be a sensible approach for several reasons:

(1) Multiple normalization transformations allow you to separate coordinate specification from their use. For static use of multiple coordinate systems, it is possible to declare the coordinate systems in GKS once and for all at the head of the program. The attraction is that redefinition is no longer necessary, only reselection. As GKS has a number of attributes and functions whose meaning changes on redefinition of normalization transformation, using a single normalization transformation and redefining it may lead to run time inefficiency in a GKS environment.

(2) Multiple normalization transformations were included to allow LOCATOR input to be returned to the user in appropriate world coordinates. Depending on the application, it may be that correct use of normalization transformations will simplify the input side of an application.

The mapping from Normalized Device Coordinates (NDC) to Device Coordinates appears to be a lesser problem. Either the earlier system had a similar facility or the system went straight from world to device coordinates. In the latter case, NDC can be treated as the device coordinates with a default workstation transformation.

GKS provides a wide range of inquiry functions which allow the application to access information in various internal GKS state tables.

Precise scaling of output provides a good example of the use of inquiry functions. For example, an application may require to output to a plotter a scale drawing at a precise size. Suppose the application defines the picture in NDC space so that the range 0 to 0.5 in the x and y directions correspond to 400 centimetres on the plotter output. If the application is to be used in a number of different environments, it will be unaware of the sizes of the plotters available or the device coordinates that they use. The inquiry function:

```
INQUIRE DISPLAY SPACE SIZE(WSTYP,IND,UNITS,RX,RY,IX,IY)
```

returns the characteristics of a workstation of type WSTYP. UNITS indicates whether the device is addressed in metres or not. RX and RY give the maximum size of the display either in metres, if precise scaling is possible, or the device coordinates used if not. IX and IY give the display size in terms of addressable positions. To achieve the desired effect would require:

```
INQUIRE DISPLAY SPACE SIZE (WSTYP,IND,UNITS,RX,RY,IX,IY)
IF (UNITS .NE. METRES) GOTO 100
IF (MIN(RX,RY) .LT. 0.4) GOTO 100
SET WORKSTATION WINDOW (WS,0,0.5,0,0.5)
SET WORKSTATION VIEWPORT (WS,0,0.4,0,0.4)
...
100  ERROR
```

2.4 Aspects

The richer aspect model of GKS does present a problem in reprogramming existing applications. Most will have been defined with an attribute model similar to INDIVIDUAL specification rather than BUNDLED specification in GKS.

A decision has to be made whether the BUNDLED mode of working which provides maximum differentiability across a range of workstations is an asset for the particular application. If it is and the number of variants in a particular primitive's aspects is small, a move to BUNDLED working is appropriate. If it is large (greater than the number of bundle table entries allowed in the implementation) either a hybrid scheme or INDIVIDUAL working will still be needed. If only one aspect is being used with many different values, it is possible to define that as an INDIVIDUAL aspect while leaving the rest bundled.

Each workstation has a number of predefined bundles. It is unlikely that these will be appropriate for your application. The tendency has been for implementations to have only one aspect changing in the predefined bundles. For monochrome displays, the polyline linetype tends to vary with the other aspects being set to the default value. For colour displays, the colour changes and the other aspects are set to the default. Clearly, some thought has to go into the reorganisation and choice of settings.

GKS does allow more generic library functions to be defined. If aspect handling is changed significantly, it may be sensible to consider how

library routines can be used for maximum effect. For example, contour routines can be defined with bundle table indices being the only mechanism used for differentiation. The application can then tailor the use by suitable definition of bundle table entries. Use at RAL and in the NAG Graphics supplement, has demonstrated that the structure of library routines can be simplified while retaining or increasing their flexibility.

2.5 Input

Input is the area where existing systems differ most from GKS. For a full GKS Level 2c implementation, the GKS input model is sufficiently rich and flexible that most existing input models can be handled without too much reprogramming. For example, GINO has different input models depending on the device and these can now be unified into one.

A problem that does exist is how to map the Tektronix input from tablet or thumb wheels which delivers both a locator position and a key hit. The device is a hybrid between a LOCATOR device and a CHOICE device. A good Level 2c implementation would model this as two devices with a single trigger, the key hit. Two simultaneous events would be delivered to the event queue where one gives the locator position and the other the CHOICE from a set of possible key hits.

For Level b implementations, there is a problem in that only REQUEST input is available and simultaneous events are not allowed. The tendency has been for implementations to provide a LOCATOR input device and an ESCAPE function which provides the key hit that activated the trigger for the previous event. An alternative approach, which has been proposed for the RAL implementation, is to regard the thumbwheels as a LOCATOR device and CHOICE device as above. REQUEST LOCATOR will return the position and REQUEST CHOICE the key hit, arising from a single operator event, the LOCATOR and CHOICE measures being held as an internal state of the devices in GKS. An ESCAPE function will be used to associate particular LOCATOR and CHOICE devices with the thumbwheels. The theory is that if an application program written in this way is moved to an environment which does not support this combination, the ESCAPE function will fail, but the application will still work, although the operator would have to input the position and key hit on separate events. Different implementations have taken different approaches and care should be taken as to the need to use this facility and how it is identified in the program. It is one where the user may well have to modify the application when he changes to a different GKS or when GKS itself is updated.

A GKS implementation providing good prompt/echo facilities may well do a number of functions currently handled by the application. At RAL, several applications have been able to remove significant amounts of code and replace them by a single input call.

For devices with mouse or tablet input, positioning has often been used as the mechanism for menu input (pull-down and pop-up menus frequently use a mixture of mouse button pressing and positioning to define a CHOICE). In such cases, the GKS implementation should provide CHOICE

implementations that use the echo area to define the extent of the pop-up menu, provide echoing via inversion etc. If such devices are in wide use in your installation, it may well be important to compare the richness of the input devices provided. They may seriously alter the amount of work needed in reprogramming.

2.6 Summary

This section has only really touched on the reprogramming problems likely when moving to GKS. The main points are:

(1) Keep to the GKS philosophy rather than modifying GKS to fit to your own practice. The reprogramming may be higher but the later benefits will be well worth it.

(2) Look seriously at the attribute model you need in your area now and in the future.

(3) A Level 2c implementation may well make it easier to reprogram your application particularly if it is highly interactive. Failing that, be careful to assess the sophistication of the standard input defaults provided by the system. If you use a window manager or User Interface Management System, GKS may need to be compatible with it.

The communities that have moved to GKS appear to be satisfied with the change. However, it should be clear that GKS is a 2D system that is not designed for highly interactive environments where a close coupling is required between the application database and the graphics. In such environments, it may well be worth waiting until an appropriate standard is provided.

3. GKS IMPLEMENTATIONS AND LANGUAGE BINDINGS

3.1 Implementation Differences

Choosing a GKS product will depend significantly on your current and future needs. Some points that need to be considered are:

(1) GKS Level: the majority of implementations on the market are either Level 1b or 2b at the moment. That means that complex interaction is not possible. There is no SAMPLE and EVENT mode input. An implementation which has a rich set of echo types may make the need for these modes less essential. A particularly good test of an implementation is how it handles Tektronix 4010 input from a cursor position and key hit. The Level 2b implementation has to provide some non-standard action to cover this.

The richer the implementation, the larger it will be. Thus if you are only looking for a minimal graphics package, Level 0a may be all you need. The ANSI standard defines a Level ma which is even smaller. For flexbility, a Level 2b implementation should also allow the user to access it as a Level 0a with reduction in space

and, hopefully, some improvement in speed. Again, implementations vary as to whether this is possible.

(2) ASF Settings: the default setting of the ASF Flags to INDIVIDUAL or BUNDLED is a cultural difference between the USA and Europe. Many USA implementations have the default as INDIVIDUAL while the European ones favour BUNDLED. This provides a clue to which mode of working the implementation is optimised for. If the installation only requires one or other mode of working, looking at the default may help in making a decision between two GKS implementations.

(3) Font Definitions: most of the laser printer output devices now have a wide range of fonts available. For good previewing of text, a range of fonts is essential. Again in the presentation graphics area, multi-font use is frequent. Implementations vary between two or three fonts being available to 30 or 40.

(4) Language Binding: there are effectively three FORTRAN language bindings, a FORTRAN 77, a FORTRAN 77 subset and a FORTRAN 66. Some FORTRAN compilers cannot handle the full FORTRAN 77 dialect. Consequently, the FORTRAN 77 subset has been introduced to ensure that these compilers can be used. The major problem is whether or not variable length strings can be defined using CHARACTER*(*).

For other languages, a number of implementations allow Pascal and C programs to call the FORTRAN binding. You should be aware that this is non-standard and Pascal programs should use the correct Pascal binding. The major differences will be in how enumeration types, lists of points and character strings, are handled. An implementation which has genuine FORTRAN and Pascal bindings may be worthwhile if you make use of both languages.

Some C implementations have used features of the C language which go against the GKS philosophy (for example, amalgamating many INQUIRIES into a single function), but these implementations predate the ISO working drafts of the C binding.

(5) Minimal Support: GKS insists that a particular level of GKS must have a minimal level of richness. Implementations should be checked against this criteria. Otherwise, it is feasible that applications ported from other sites may not work. Table 1 gives details of minimal support.

(6) Devices: some devices provide the facilities of a GKS workstation including segment operations. Effective use of these devices may not be possible with some implementations which have limited internal workstation interfaces.

(7) Richness: Table 1 gives a minimum set of requirements. GKS implementations vary considerably in their richness. Particular areas of concern are the number of workstation types supported, variety of linetypes, number of GDPs, colour table size etc.

Table 1 - Minimum support required at each level

CAPABILITY	Level 0a	0b	0c	1a	1b	1c	2a	2b	2c
Foreground colours (intensity)	1	1	1	1	1	1	1	1	1
Linetypes	4	4	4	4	4	4	4	4	4
Linewidths	1	1	1	1	1	1	1	1	1
Predefined polyline bundles	5	5	5	5	5	5	5	5	5
Settable polyline bundles	-	-	-	20	20	20	20	20	20
Marker types	5	5	5	5	5	5	5	5	5
Marker sizes	1	1	1	1	1	1	1	1	1
Predefined polymarker bundles	5	5	5	5	5	5	5	5	5
Settable polymarker bundles	-	-	-	20	20	20	20	20	20
Character heights (see note 1)	1	1	1	1	1	1	1	1	1
Character expansion factors (see note 1)	1	1	1	1	1	1	1	1	1
String precision fonts	1	1	1	1	1	1	1	1	1
Character precision fonts	1	1	1	1	1	1	1	1	1
Stroke precision fonts	0	0	0	2	2	2	2	2	2
Predefined text bundles	2	2	2	6	6	6	6	6	6
Settable text bundles	-	-	-	20	20	20	20	20	20
Predefined patterns (see note 2)	1	1	1	1	1	1	1	1	1
Settable patterns (see notes 2 and 5)	-	-	-	10	10	10	10	10	10
Hatch styles (see note 3)	3	3	3	3	3	3	3	3	3
Predefined fill area bundles	5	5	5	5	5	5	5	5	5
Settable fill area bundles	-	-	-	10	10	10	10	10	10
Settable normalization transformations	1	1	1	10	10	10	10	10	10
Segment priorities (see note 4)	-	-	-	2	2	2	2	2	2
Input classes	-	5	5	-	6	6	-	6	6
Prompt and echo types per device	-	1	1	-	1	1	-	1	1
Length of input queue (see note 5)	-	-	20	-	-	20	-	-	20
Maximum string buffer size (characters)	-	72	72	-	72	72	-	72	72
Maximum stroke buffer size (points)	-	64	64	-	64	64	-	64	64
Workstations of category OUTPUT or OUTIN	1	1	1	1	1	1	1	1	1
Workstations of category INPUT or OUTIN	-	1	1	-	1	1	-	1	1
Workstation Independent Segment Storage	-	-	-	-	-	-	1	1	1
METAFILE OUTPUT workstations	0	0	0	1	1	1	1	1	1
METAFILE INPUT workstations	0	0	0	1	1	1	1	1	1

0 indicates explicitly defined and non-required at that level.
- indicates not defined at that level.

NOTES

1. Relevant only for character and string precision text.
2. Relevant only for workstation supporting pattern interior style.
3. Relevant only for workstation supporting hatch interior style.
4. Relevant only for workstation supporting segment priorities.
5. Since available resources are finite and entries have variable size, it may not always be possible to achieve the minimum values in a particular application.

(8) Input: Levels of GKS which support input require the user to have access to at least one input device of each class at his work space. That does not imply that all workstations have to support all input device classes but an installation is required to ensure that a workstation cluster fulfills this requirement. An application program can expect to have at least one input device of each class available to it.

A number of implementations provide workstations with only a subset of the input devices available. For example, LOCATOR, STRING, VALUATOR but not CHOICE, STROKE and PICK. An organisation needs to decide if the workstations provided by the implementation are valid within its environment.

As mentioned above, GKS insists on a minimum of prompt/echo types of which the first is natural to the device. Again implementations differ as to this first echo method chosen and how many additional ones are provided.

For devices such as STROKE and STRING, a buffer is needed to hold the individual elements of the input. This buffer size must be appropriate for the installation.

(9) Environment: any GKS implementation chosen must fit into your local environment. At the simplest level, it must support the devices available and run on the hosts owned by the site. A useful facility is to have a skeleton driver available which can be used as a basis for constructing new drivers. A portable implementation available on a range of hosts might be desirable if there is a range of hosts on a particular site. Most suppliers are tending to prefer a single implementation of GKS on their system. Large sites may therefore run several different implementations of GKS in which case compatibility between implementations is important. For example, metafiles should be transportable from one implementation to another.

3.2 Language Bindings

3.2.1 Introduction

In the early days of ISO work on graphics standards, it was realised that the different languages from which people used graphics made it necessary to define the functional specification in a way that was independent of any one language. For this reason, the GKS document does not specify the interface from FORTRAN, Pascal, or any other language. Instead, a separate ISO standardisation activity has been set up to cover all the language bindings for GKS, GKS-3D and PHIGS.

The GKS FORTRAN binding for GKS was the first binding to be completed technically. During the development of the FORTRAN binding, a number of issues were identified that are relevant to all programming language bindings. These issues fall into the following categories: goals and scope, naming/abbreviations, data typing, error handling, and multi-language environment considerations.

3.2.2 Goals and Scope

The first problem to be considered is the status of the programming language to which the binding is being made. An ISO standard language binding may only be promulgated for a programming language defined in a referenceable document. In practice this means that the programming language must have been defined in an ISO standard or a national standard. Bindings to languages not in this category (for example Algol 68) have been published in the technical literature, but have not been and cannot be standardised. This constraint causes problems where there is pressure from the user community for a binding to a language that has not been standardised, C is a good case in point. There is a document describing a C language binding for GKS, however this cannot be standardised by ISO until C has been standardised. Processing of the ISO C language binding has started, but cannot progress significantly until the C language is standardised.

A second problem following on from the above is how to cope with the situation in which the ISO standard definition of a programming language differs from a national standard. Pascal is a case in point. ISO Pascal has two levels, with conformant arrays. ANSI has one level, without conformant arrays. The view taken is that national differences must not be allowed to impose artificial constraints.

The GKS Pascal binding may be used under either level 0 or level 1 Pascal, and a mechanism is defined for replacing conformant array parameters with fixed length array parameters for level 0.

Another important issue is the extent to which language facilities should be used in the binding. Should the full richness of the language be exploited or should a minimal subset of features be used to facilitate portability between languages? This problem is a difficult one.

Some languages provide more capabilities than the functional specification requires. The multi-tasking and exception handling capabilities of Ada are a good example of this. Should an Ada binding exploit these capabilities, or should the binding be close to the Pascal binding? Other languages are not able to provide the functional requirements in a one-for-one way. For example, some of the GKS inquiry functions return a list of related, but heterogeneously typed data items. These would result in extremely long parameter lists if bound one for one with FORTRAN subroutines, but can be handled neatly in Pascal using records. Should the single GKS function be split into several subroutines in the FORTRAN binding, but remain as a single procedure in the Pascal binding? These questions do not have simple answers.

Further problems are caused when the language standard describes several subsets or levels. The FORTRAN standard defines a subset to maintain compatibility with FORTRAN 66; however the subset imposes severe limitations on the data-types available. Where bindings have to be defined for subsets, to what extent should the subset bindings be compatible with the full language binding, ie should the full language binding be compromised so that compatibility is maintained with subset bindings?

What account should be taken of compilers that impose more or less restrictions than the standard requires. For example, the Pascal standard allows 32 character identifiers, however many compilers for micros impose a 6 character limit. Do you bind for the 'real world' or the standard? Again there is no easy answer to this problem.

3.2.3 Naming/Abbreviations

The GKS functional description uses very long function names, for example, INQUIRE NUMBER OF SEGMENT PRIORITIES SUPPORTED. This posed an immediate problem for the FORTRAN binding, because of the language's restriction on the length of external names. Should a consistent scheme be used to map these names to a (probably) meaningless acronym or should a more ad hoc scheme be used which would admit more meaningful acronyms? Eventually it was agreed that the mapping needed to be consistent and a complex algorithm was devised and documented in the FORTRAN binding.

Having devised such a naming scheme should it be used across similar languages? This problem was solved by devising a common abbreviation scheme that allows names longer than 6 characters.

It is clear that GKS function names should be distinguishable from other functions in the application, however, how should this be done? If a sentinel character is used, what character or characters should it be? In the case of FORTRAN all GKS function names start with the letter G, however this then leaves only 5 characters for the remainder of the name!

A good attempt has been made to devise a consistent abbreviation scheme, and this is evident from the similarities between the Pascal, Ada and C bindings.

3.2.4 Data Types

Selecting function names is a major task in developing a language binding. An equally major task is defining the mapping between the data types of the GKS functional description and the data types of the programming language. The GKS functional description uses structured data types including enumeration types for the specification of a limited number of choices, record data types for grouping heterogeneous primitive data types and a point data type to represent 2D coordinate data.

One for one mappings of the GKS data types can easily be given for languages such as Pascal and Ada which have equally rich or richer typing structures. However, for languages such as FORTRAN, which offer a more restricted typing structure, significant problems arise.

In the case of languages with rich typing structures, should similar representations of the GKS data structures be used, or should the capabilities of each language be fully exploited?

File access mechanisms are a problem in all programming languages. Interactions between languages and operating systems have meant that in some cases some data types and values have to remain implementation dependent.

Some languages support pointer manipulation. Taking advantage of this facility could give the application access to implementation data structures. Should this be allowed? The decision has been made that parameters defined as 'input' parameters in GKS must be input parameters in language bindings.

If all the parameters in a GKS function definition are bound one-to-one with procedure parameters, unnecessary parameters may result, for example the length of a text string when the text string is represented by the FORTRAN CHARACTER type. The decision has been made to eliminate redundant parameters, but stick closely to the GKS definition of input and output parameters.

3.2.5 Error Handling

The error handling mechanisms defined in the GKS document are very FORTRAN-like. Should modern methods for exception handling (such as that found in Ada) be used in a language binding?

3.2.6 Multi-Language Environment

Some implementors and users feel it is important to be able to access GKS functions from more than one language, within a single implementation. This type of environment poses many problems. One approach would be to define a single 'generic' binding, which would probably look like a FORTRAN subset binding.

So far only single language bindings have been developed and this area remains open.

3.2.7 Future Bindings

One very important decision has been made. Bindings for future graphics standards will be based on the GKS bindings. This means that once the major problems have been resolved for a language binding of GKS, the same solutions will be used in subsequent bindings. A common set of guidelines for language bindings has been produced as a result of some 4 years hard work. Language bindings are by no means as trivial as they might at first sight appear.

3.3 Availability of GKS Implementations

The two early implementations of GKS which are still widely available are GKSGRAL which derives from the version implemented at the Technical University of Darmstadt and the joint ICL-Rutherford Appleton Laboratory implementation which is widely used in the UK university environment.

A list of implementations (to Level 2b unless indicated) with FORTRAN as the main binding (unless indicated) is:

(1) Rutherford Appleton (RAL)/ICL (Level 1b going to 2b)
(2) GTS-GRAL (Level 2c)
(3) NOVA Graphics, Austin
(4) CWI Amsterdam (Level 2c)
(5) Tektronix
(6) CEEGEN, Los Gatos
(7) Whitechapel, UK
(8) Prior Data Sciences, Ottawa (C Binding)
(9) Precision Visuals, Boulder
(10) Dataplotting Services Inc
(11) RAMTEK
(12) Visual Engineering, San Jose
(13) Advanced Technology Centre, Culver City (Level 2c)
(14) TEMPLATE
(15) UNIRAS, Mass
(16) DEC (Level 0b)
(17) Data General
(18) IBM/Graphic Software Systems
(19) Infolytica, Montreal (Level mb)
(20) AED-GKS (Level 2c)
(21) CMC, Bombay
(22) Sysgraph, Vienna
(23) System Simulation Ltd, London (Level 1a)
(24) XGKS, Hungary

Fuller details are given in 'Graphics Standards - The Current State', AUSGRAPH 86 Introductory Tutorial.

3.4 Conformance

3.4.1 Introduction

The existence of standards for computer graphics immediately raises the important question: how can one be sure that an implementation of a standard adheres to the standard? In the absence of an answer to this question, it is arguable that standardisation is a pointless activity.

At the present time, the only practical approach to this question is to consider methods of falsification, in which systematic efforts are made to demonstrate that an implementation is incorrect. This approach has been adopted for programming languages. Typically a compiler is subjected to a large suite of test programs, specially designed to uncover likely errors or possible misinterpretations of the standard. This approach has been adopted for Cobol, Pascal and now Ada. There are some deep philosophical questions raised by this approach, for example how does one know the behaviour prescribed for the test suite adheres to the standard. However, with current programming technology, it is the best that can be done.

The EEC, recognising the importance of the area, sponsored a number of workshops during 1982, aimed at developing a falsification procedure for validating GKS implementations. A comprehensive test suite for GKS was subsequently developed by the University of Leicester (UK) and the Technische Hochschule Darmstadt (Germany) with support from funding agencies in the UK and Germany.

3.4.1 The GKS Test Suite

The candidate GKS implementation is stimulated in different ways and the response to the stimulation is then checked. The stimulus in the case of GKS takes the form of a GKS function call or sequence of GKS function calls, possibly in conjunction with an operator action on an input device. The response can be in the form of graphical output on one or more devices, a change to the GKS data structures, or data returned as an output parameter of a GKS function.

There are two interfaces across which stimuli can be sent and responses observed, the application program interface and the human operator interface.

The GKS conformance testing scheme is based upon a suite of test programs coupled with operator action on input devices to provide the stimulation. The test suite is described in some detail in [8,9,10]. The checking of responses is automated as far as possible.

GKS defines a large set of data structures that maintain the current values of GKS data such as the current polyline index. The standard defines precisely the action of each GKS function on these data structures. Data in the GKS state lists can be accessed through inquiry functions. One part of the test suite is devoted to testing the effect of each GKS function on the state lists. Checking in this case can be done automatically without the need for human interaction.

GKS has a well-defined error reporting mechanism, with an explicit set of error messages associated with each function. Another part of the test suite is devoted to testing the response of each GKS function in error situations.

Response at the human interface is a much more difficult problem, consisting of output on one or more graphical output devices. The problem is compounded by the workstation dependencies allowed by the GKS standard, for example whether linetype is continuous or restarted at the start of a polyline, at the start of a clipped piece of polyline and at each vertex of a polyline.

This part of the GKS test suite consists of a series of test programs which generate visual images. The (human) tester then has to check each image generated against a reference image in the testing manual and has to decide whether or not the image from the candidate implementation conforms to the reference image. A great deal of care has been taken with this part of the test suite to design images which check large amounts of GKS with a small amount of visual effort. The images are carefully annotated to aid the tester, for example a test of fill area interior style can attempt to produce the four styles HOLLOW, PATTERN, SOLID and HATCH, and then label each to indicate which it is, or is not, mandatory for the workstation to display. The visual aspects of the test suite are discussed in [10].

The test suite is structured so that testing proceeds with increasing complexity, fundamentals being tested first. On the output side, for example, the tests are organised in six classes, simple control, output, primitive attributes, normalization transformations and clipping, workstations and workstation attributes, workstation transformations.

3.4.2 Conformance Testing Services

The Commission of the European Communities has taken the first steps towards establishing European Conformance Testing Services for standards in the information technology and telecommunications areas. Three European laboratories are collaborating to establish a testing service for GKS; BNI in Paris, GMD in Bonn and the National Computer Centre in the UK. Contracts have just been signed with these organisations and it is expected that basic testing services will emerge in a 6 to 18 month timescale (from February 1986).

The GKS testing service is based on the GKS test suite described in 3.4.1.

Further details of the testing service can be obtained from:

Jane Pink
Language and Validation Services
The National Computing Centre Ltd
Oxford Road
Manchester M1 7ED
U.K.

4. IMPLEMENTATION PHILOSOPHIES

4.1 Introduction

References [11] to [22] give details of some of the GKS implementations. If we look at these in detail, differences in implementation philosophy can be seen. These differences give some clue as to the audience that the implementation is targetted at. This section aims to give some pointers towards the key implementation features.

4.2 Positioning the DI/DD Interface

GKS does not have a well defined internal interface between the device independent (DI) front-end part of GKS which the application calls and that part of GKS, the device dependent part (DD), which is responsible for the interface to devices. The reason for this is that devices vary a great deal in capability and a rigid specification of this interface would cause problems for specific devices. Intelligent devices might not be able to use all their features and dumb devices might have a large amount of simulation to do. Looking at the interfaces available, they can be categorised as follows:

(1) Thin Back-End: the DI/DD interface is positioned very close to the devices. The device independent part maps all GKS functions to a simple device order code. This approach decreases the amount of code to be written and makes device drivers simple to implement. The penalty is that intelligent devices are unlikely to be able to make use of many of their features. The Amsterdam CWI implementation was originally of this type. The motivation for this was that it was a portable implementation to be used in a UNIX environment on a variety of sites.

(2) Thin Front-End: all GKS functions are passed to the workstation for execution with a minimal amount of workstation independent code. There is some GKS State information that needs to be kept centrally. However, primitives can be passed to the workstation in world coordinates leaving all attribute binding, coordinate transformation and segment storage to the workstation. The advantages of this approach are that intelligent workstations have maximum flexibility to use their facilities. A disadvantage is that simple devices may have very large and complex drivers with the possibility of enormous duplication.

With the appearance on the market of a number of GKS workstations with a high level interface, this approach becomes more applicable.

(3) Fixed Workstation Descriptors: it is feasible to define a TERMCAP-like facility (GRAPHCAP?) which provides a data file which describes workstation characteristics. The front-end code modifies the protocol to be sent to the workstation dependent on the data file. The major problem with this approach is being able to define a data file which encapsulates the characteristics of a wide range of devices. Herman's implementations have attempted to provide a flexible way of doing this description and automatically modifying

the code in the GKS system loaded so that it is tuned to the workstation set available.

Such an implementation tends to have most GKS functions simulated in the device independent part as well as similar code being available in the drivers.

(4) Negotiation: a technique pioneered by GINO-F was to allow negotiation between the front and back-ends to decide who is responsible for particular functions. A well defined protocol between the two parts is maintained but the device dependent part has the ability to refuse to do certain operations. There is a minimal set that it has to handle but that is all. For example, the front-end may request the back-end to draw a dotted polyline. It can refuse, in which case the front-end is forced to simulate the dotted polyline sending individual line segments to the back end for output.

This is a more flexible variant than (3) in that the back-end can pick and choose which functions it can handle and do a partial set of one type if it is appropriate. For example, it may be able to handle short dashed polylines and solid ones but that is all.

(5) Device Dependent Tool Set: the major disadvantage of the thin front-end technique is the duplication of code at the driver level. An approach that solves this is to produce a large tool kit for individual drivers to use. The RAL/ICL implementation has taken this approach.

(6) Layered: it is possible to define more than one internal interface in GKS. Between the device independent and device dependent code, there is a set of operations that apply to all workstations that are currently active. It is possible to define an interface between the device independent part and the workstation manager. A second interface can be defined between the workstation manager and a particular workstation. This has some similarity with the window manager in a single user system. This approach is particularly useful in a distributed environment where data physically crosses the interface with the possibility that the two sides of the interface are in different processors. For example, a set of simple devices could be controlled by a single workstation manager remote from the host system where the application resides. This can be a considerable saving in communications traffic. Segments to be displayed on several workstations attached to the workstation manager need only be sent across the network once to the workstation manager. This approach was pioneered in the NOVA GKS implementation.

The positioning of this interface in a particular implementation is an important attribute of the implementation and will have a considerable effect on the properties of the implementation.

4.3 Immediate/Delayed Execution

Many functions in GKS cause changes to the overall GKS state which needs to be up-to-date before the next input or output occurs. For example, changes to primitive attributes will affect the way the primitive is output. A particular example is changing the TEXT FONT AND PRECISION aspect which may require a different font to be loaded or a switch from direct execution in the workstation to simulation in the front-end. When should this work be done? Depending on the positioning of the DI/DD interface, there are choices that can be made which depend on the assumed usage of GKS.

Stretching the above example further than may be sensible, it is possible that an application may change TEXT FONT AND PRECISION several times before it outputs any text. Imagine an interactive application where the user selects a font and inquires its characteristics before deciding on which one to use. Another application may select the font required initially and make no further changes throughout the program execution. Implementation efficiency will vary depending on which of these two scenarios is most likely.

One approach is for the implementation to delay all operations until it is essential that they be done. In the example above, the resetting of TEXT FONT AND PRECISION will just set a flag indicating that the font available is not correct and that is all. Each output of the text primitive will then need to check whether the selected font is the current one or a new one has to be loaded.

The alternative approach is for the implementation to load the font as soon as the TEXT FONT AND PRECISION aspect is changed. Each output of the text primitive can now assume that the correct font information is available.

In GKS, there are a large number of features of this type. The situation is complicated by the decision on whether BUNDLED or INDIVIDUAL mode of working is in place. Particular GKS operations (for example, setting primitive aspects) can be negated or made temporarily inaccessible by other operations (setting ASFs). Other data are defined by more than one function call. For example the normalization transformation is defined by setting the window and viewport in two function calls that can be done in either order. Does the implementation update the transformation each time one or other is called?

Some implementations have made the decision to do immediate execution of all GKS functions while others (for example, RAL/ICL) delay the execution as long as possible. These strategies will cause the GKS implementation to have different properties that can be tested by benchmarks.

4.4 Segment Storage

Conceptually, each workstation has its own segment store. For intelligent workstations this is sensible as it makes it possible for good interaction to be achieved at the workstation. If the level of GKS in use is Level 2, GKS also has a Workstation Independent Segment Store (WISS) which contains a set of segments which can later be manipulated and moved from WISS to specific workstations under application control. There are a variety of strategies that can be taken to achieve the desired results:

(1) Central Segment Store: a single central segment store could be implemented with tables indicating which segments are available to which workstations. Segment updating at the workstation generates traffic from the central store to the workstation. If all or nearly all the devices in use are relatively unintelligent, this is a valid approach. Implementing WISS is achieved by adding WISS functions to the central store.

(2) Phantom Workstation: all workstations are assumed to be capable of holding their own segments. An unintelligent workstation can ask the phantom workstation to hold its segments for it. This means that the only segments stored centrally in the phantom are the ones required by unintelligent workstations. An interesting aspect of this approach is to label WISS as unintelligent in which case the phantom workstation will also handle the workstation independent segment store.

(3) Distributed Segment Store: assuming at least one intelligent workstation capable of storing segments, it is feasible to distribute all segments including WISS to the workstations. Intelligent workstations can hold both the WISS and the segment stores of unintelligent workstations. As a particular application tends to only use WISS for intermediate storage of segments ultimately destined for a workstation and the unintelligent workstations have a close affinity to the intelligent workstation (a plotter attached to a raster display), this approach is more sensible than it may at first seem.

    To do this, it does require the workstation to have the ability to read back segments to the host for manipulation and display on other workstations. Some intelligent GKS workstations on the market do not have this capability.

Another feature of GKS is that segments can be created with random names. On creating a new segment, the segment store needs to be checked to see if the name is already in use. This can be a significant table handling problem that needs to be done efficiently particularly if a large number of segments are in use. As the population of legal segment names is very large, some kind of hashing technique or caching of recently used segment names is essential for rapid picture updating. It is worth checking the performance of implementations in this area if it is important in the GKS environment where it is to be used.

Finally, as segments cannot be reopened they are effectively linear and can be stored in contiguous locations. However, gaps appear in storage as soon as segments are deleted. This produces a typical garbage collection problem. A simple implementation of segments store may produce fast traversal but slow garbage collection. Alternatively, the garbage collection problem can be largely solved by allowing segments to be stored in fragments with links between them. This slows down traversal but speeds up garbage collection. Again, benchmarks can easily be designed to test an implementation against the environment where it is to be used.

5. RELATED STANDARDS

5.1 Reference Model

Following the development of GKS, standardisation activities have expanded to include 3D graphics and various interfaces.

The work has tended to fragment with individual groups of experts concentrating on one or two of the many activities. As a result, there has been a tendency for the standards to drift apart. In order to bring the standards more into line, an overall Reference Model for the graphics standards activities is being defined. This will help to clarify the interrelationship between the various standards and indicate their area of applicability.

The major standards in progress are:

(1) GKS - a set of basic functions for 2D device-independent computer graphics programming.

(2) CGM - a device independent data exchange format for computer graphics pictures.

(3) CGI - a set of basic elements for the control and data exchange between device-dependent and device-independent levels in graphics.

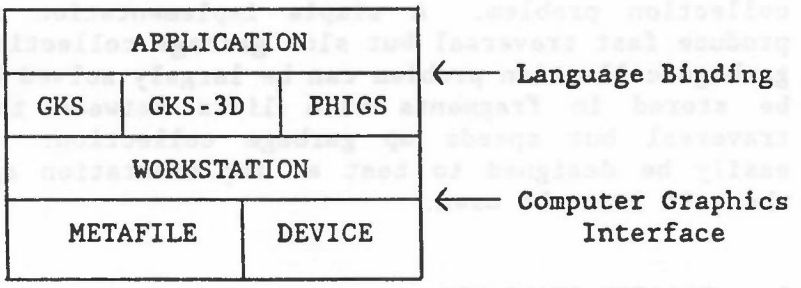
(4) GKS-3D - an extension of GKS to provide the basic functions for computer graphics programming in 3D.

(5) PHIGS - a set of functions for computer graphics programming in environments requiring rapid modification of graphical data that describes geometrically related objects.

(6) Language Bindings - bindings of the functions and data types of the functional standards to standardised programming languages.

(7) Registration - a registration mechanism is being set up to deal with the standardisation of line and marker types, GDPs, Escapes etc.

These various standards fit together as follows:



The main points of the model are:

(1) Applications access the functional standards via well defined language bindings.

(2) There is no explicit standard for the interface between the functional standards and the workstation.

(3) Workstations talk to devices or metafiles via a Computer Graphics Interface.

(4) The Computer Graphics Interface is comprehensive and can be tailored for most devices. It is divided into Option Sets related to the major features. A workstation is likely to use only a fraction of the option sets available.

(5) CGI is the protocol used to generate the picture part of the Computer Graphics Metafile (CGM). Consequently, there is a close relationship between CGI and the computer graphics metafile.

(6) Language Bindings have a consistent structure for each of the functional standards.

5.2 Functional Standards

The three functional standards GKS, GKS-3D and PHIGS, are closely related in that they share the same:

(1) Output Primitives: with an additional primitive FILL AREA SET for GKS-3D and PHIGS. There is an obvious extension of the GKS 2D primitives to 3D.

(2) Attribute Model: the attribute model is the same for all three standards with bundled and individual modes of specification.

(3) Input Model: with obvious extensions to 3D for LOCATOR and STROKE devices.

(4) Workstation Concept: all provide for intelligent workstations having at most one display screen and multiple input devices.

The major differences are:

(1) PHIGS Structures: PHIGS allows graphics and applications data to reside in structures. When structures are traversed, primitives are created with associated attributes which are processed by the same pipeline as GKS-3D. A modelling transformation can be applied to PHIGS structure elements at the start of the pipeline.

(2) Segments: assuming PHIGS is used with high quality terminals capable of doing structure traversal in the terminal, it is believed that no segment storage is required. Effectively, PHIGS creates primitives which are immediately output.

5.3 GKS-3D and PHIGS

The two functional standards that extend GKS to 3 dimensions are GKS-3D and PHIGS. GKS-3D is a minimal extension to GKS to allow 3D working. PHIGS builds on GKS-3D to provide a structure facility for dynamic picture changes. In this section, the aim will be to highlight the similarities and differences. It is worth making the point that the two standards are at different stages in their evolution consequently differences in the standard proposals now may be removed during the review process.

5.3.1 Scope and Goals

The scope of GKS-3D is similar to that of GKS except that GKS-3D provides application programs with the capability to define and display 3D graphical primitives specified using 3D coordinates. In addition, the GKS input model has been extended to provide 3D locator and stroke input. A major goal is that existing GKS programs should run without change where possible.

Neither GKS or GKS-3D satisfy the requirements of application programs where modification of the graphical data is required in an efficient manner, where the objects to be displayed consist of geometrically related parts and where rapid dynamic articulation of graphical entities is required. PHIGS is aimed at addressing that community. As rapid movement will tend to require hardware assistance, PHIGS is oriented towards high class intelligent workstations with the ability to manipulate display files locally.

5.3.2 Segments and Structures

The Segment Store in GKS-3D is identical to that in GKS except for the extension to 3 dimensions. Segments are normally stored on workstations. In addition, there is a central Workstation Independent Segment Store (WISS) to allow movement of segments from this central store to workstations as required.

In PHIGS, to achieve its objectives, the segment store has been replaced by a structure store which has greater functionality than the GKS-3D segment store and appears at a different place in the viewing pipeline. The difference can be seen in Figure 5.

The major points are that, in GKS-3D, primitives created when a segment is open are stored in the segment storage for all active workstations. The normalization transformation is applied to the primitive before it is stored in the segment but not the associated clip. Consequently, each primitive is stored in segment store in normalized device coordinates (NDC3) with the associated clipping volume. When a segment is displayed on the workstation, the primitive is read from segment store, the segment transformation is applied, the normalization volume clipping is done and the primitive is then presented to the viewing pipeline.

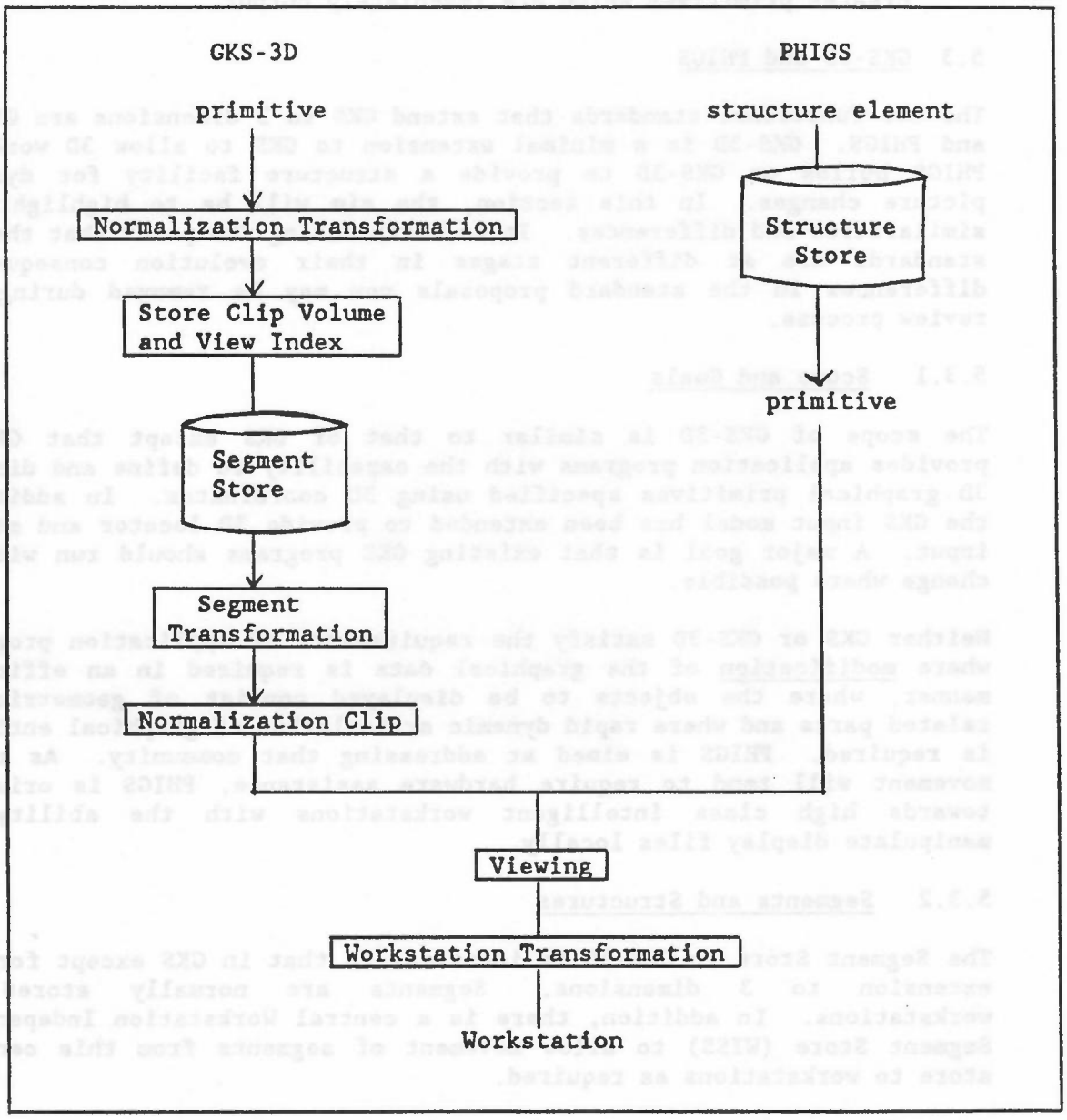


Figure 5

The PHIGS pipeline for viewing is identical but the creation of the primitive is significantly different. PHIGS provides a set of functions which define structure elements. There is a 1-1 correspondence between primitives and structure elements. In addition, PHIGS has structure elements for attribute setting and specification of application data. A major feature of the structure store is that it is hierarchical. Structure elements are provided which call other structures.

Structures are displayed when they are posted to a workstation. Posting causes the structure to be traversed and interpreted. A complex modelling transformation is applied to coordinates in the structure elements as they are interpreted.

The primitives generated by PHIGS enter the viewing pipeline immediately prior to the viewing operation. The complex modelling transformation provides a similar function to the normalization transformation.

Another difference between the two storage structures is that the Structure Store is held centrally. Posting a structure effectively causes a display file to be moved to a workstation. For those workstations specifically designed for PHIGS, it is assumed that structure traversal and viewing are both performed by the workstation.

5.3.3 Viewing

Viewing consists of projecting the 3D image on to a 2D projection plane. The Viewing Pipeline is given in Figure 6. Functions are provided to assist with the definition of this viewing operation. These are shown in Figure 7. The initial coordinates (NDC3 in GKS-3D) are changed to Viewing Coordinates by defining a View Reference Point and a set of axes associated with it. The intention is that this point has some relationship to the object to be viewed and makes the setting up of the projection transformation that much easier.

Once the Viewing Coordinates are established, Front and Back Planes are defined which specify the limits of the object to be viewed. A Projection Reference Point can be specified and a Projection Plane which allows the object to be viewed by projecting it onto the projection plane. The View Window specifies that part of the projection plane to be output to the workstation. Both parallel and perspective projections are provided.

For some workstations, capable of providing 3D geometric transformations and for genuine 3D devices, the viewing operation specified by the functions provided may not be appropriate. Therefore, it is possible for applications to construct their own viewing pipeline or ignore parts of it.

Each primitive has a View Index associated with it which defines the view bundle table entry on the workstation to be used. This contains details of the viewing transformation and clipping to be applied. It was believed that, unlike GKS, there is a need for more than one view to be available at a time on a workstation. This would, for example,

allow titles to be output using a parallel projection while a 3D object to which the titles are associated is output using a perspective transformation. The view bundle is analogous to the polyline bundle in that it allows views to be significantly different on different workstations.

GKS-3D provides support for Hidden Line and Hidden Surface calculations at the workstation level. Associated with primitives is an attribute defining which method of rendering is to be used on the workstation. The workstation can be asked to render or not and it has flexibility in how it does the rendering. Consequently, a variety of workstations can choose the most appropriate methods depending on their hardware characteristics. No similar facility is provided in PHIGS at the moment. Consequently, for rendered images, GKS-3D is more appropriate than PHIGS.

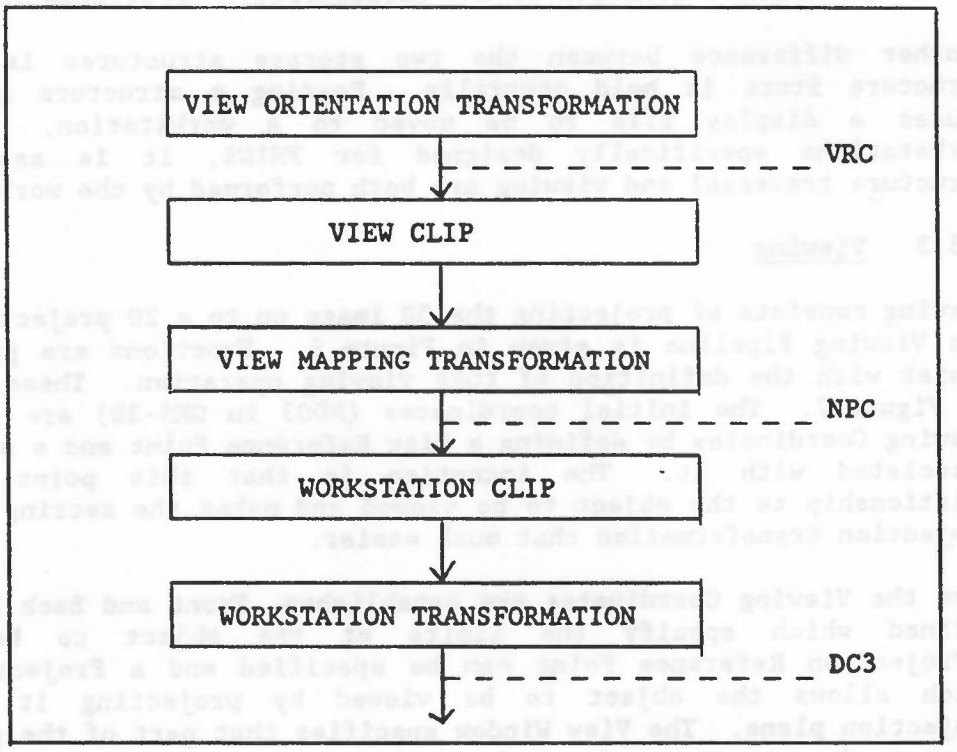


Figure 6

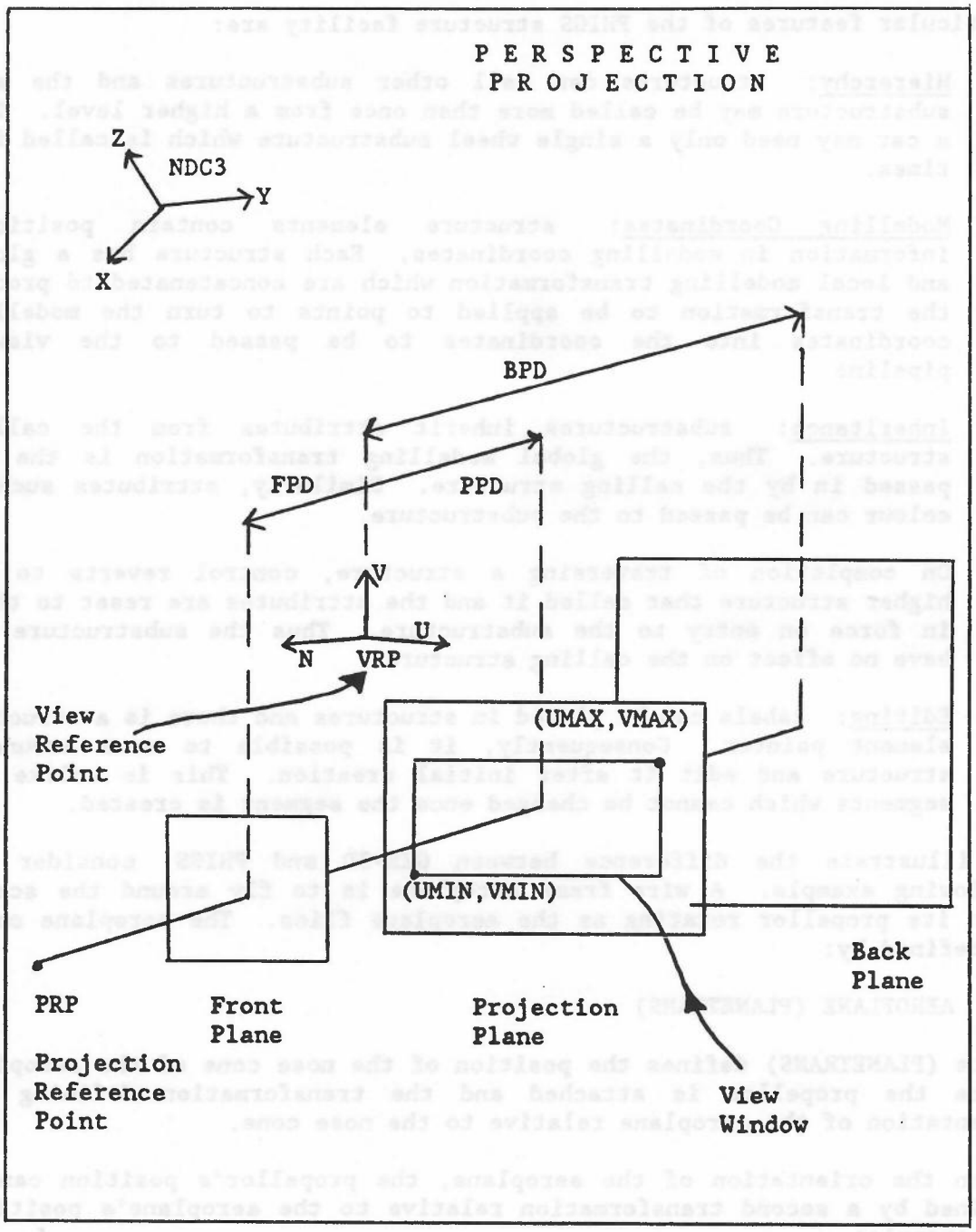

PERSPECTIVE
PROJECTION
Z
NDC3
Y
X
BPD
FPD
PPD
V
U
N
VRP
View
Reference
Point
(UMAX,VMAX)
(UMIN,VMIN)
Back
Plane
PRP
Front
Plane
Projection
Plane
Projection
Reference
Point
View
Window

Figure 7

5.3.4 Structures

Particular features of the PHIGS structure facility are:

(1) Hierarchy: structures can call other substructures and the same substructure may be called more than once from a higher level. Thus a car may need only a single wheel substructure which is called four times.

(2) Modelling Coordinates: structure elements contain positional information in modelling coordinates. Each structure has a global and local modelling transformation which are concatenated to produce the transformation to be applied to points to turn the modelling coordinates into the coordinates to be passed to the viewing pipeline.

(3) Inheritance: substructures inherit attributes from the calling structure. Thus, the global modelling transformation is the one passed in by the calling structure. Similarly, attributes such as colour can be passed to the substructure.

    On completion of traversing a structure, control reverts to the higher structure that called it and the attributes are reset to those in force on entry to the substructure. Thus the substructure can have no effect on the calling structure.

(4) Editing: labels can be placed in structures and there is a structure element pointer. Consequently, it is possible to move around a structure and edit it after initial creation. This is unlike GKS segments which cannot be changed once the segment is created.

To illustrate the difference between GKS-3D and PHIGS, consider the following example. A wire frame aeroplane is to fly around the screen with its propellor rotating as the aeroplane flies. The aeroplane could be defined by:

    AEROPLANE (PLANETRANS)

where (PLANETRANS) defines the position of the nose cone of the aeroplane where the propellor is attached and the transformation defining the orientation of the aeroplane relative to the nose cone.

Given the orientation of the aeroplane, the propellor's position can be defined by a second transformation relative to the aeroplane's position. For example:

    PROPELLOR (PLANETRANS, PROPTRANS)

Clearly the second transformation has constraints on it so that the propellor stays perpendicular to the length of the aeroplane but we shall ignore that in the example!

To generate this scene in PHIGS would be done by defining two structures:

```
OPEN STRUCTURE (PLANE)
SET LOCAL TRANSFORMATION (PLANETRANS)
AEROPLANE (IDEN)
EXECUTE STRUCTURE (PROPELLOR)
CLOSE STRUCTURE

OPEN STRUCTURE (PROPELLOR)
SET LOCAL TRANSFORMATION (PROPTRANS)
PROPELLOR (IDEN,IDEN)
CLOSE STRUCTURE
```

We have taken a few liberties with the definitions above to simplify the example. Each structure has a local transformation that can be set up. In the example, we have set up the aeroplane transformation and the propellor transformation as local transformations to be applied to the structure. The plane and propellor are drawn using the identity matrix, IDEN. The EXECUTE STRUCTURE command creates the structure hierarchy by effectively calling the PROPELLOR structure as part of the PLANE structure. The PROPELLOR structure inherits the PLANE's orientation as a global transformation applied to it and this gets concatenated with the local transformation.

To make the aeroplane fly around, we need to generate a loop which updates both transformations and redisplay. This would be achieved in PHIGS by editing the two structures. When a structure is reopened, the editing position is situated at the last element. Consequently, it would need to be repositioned:

```
OPEN STRUCTURE (PLANE)
SET ELEMENT POINTER (1)
SET LOCAL TRANSFORMATION (NEWPLANETRANS)
CLOSE STRUCTURE

OPEN STRUCTURE (PROPELLOR)
SET ELEMENT POINTER (1)
SET LOCAL TRANSFORMATION (NEWPROPTRANS)
CLOSE STRUCTURE

REPEAT
```

Note that all we have done is edit the two structures by replacing the local transformation matrices.

In GKS-3D, the problem is different in that we can define a transformation matrix to be applied to the aeroplane but there is no way in which the propellor can appear at a different orientation if it is part of the same segment as no editing is possible on the segment. A possible approach would be:

```
CREATE SEGMENT (PROP)
PROPELLOR (IDEN, IDEN)
CLOSE SEGMENT

Start loop
DELETE SEGMENT (PLANE)
CREATE SEGMENT (PLANE)

SET SEGMENT TRANSFORMATION (PLANE, NEWPLANETRANS)
AEROPLANE (IDEN)
INSERT SEGMENT (PROP, NEWPROPTRANS)
CLOSE SEGMENT

REPEAT
```

Here the segment PLANE has the transformation NEW PLANE TRANS applied to it. The loop consists of deleting the PLANE segment each time and defining a new one where the propellor is inserted as part of it.

This is clearly a great deal more work than in the PHIGS case. You can imagine garbage collection of the old segment taking place and a large new segment having to be created. The display is likely to go blank when the old segment is deleted and before the new one can be regenerated. Given the correct hardware, the PHIGS editing could be achieved almost instantaneously.

The example chosen above is one which is reasonably sympathetic to GKS-3D. If primitives in the structure are being changed, GKS-3D has no alternative but to regenerate the segment from scratch or via inserting segments.

5.3.5 Primitives

Both GKS-3D and PHIGS have attempted to keep the same primitives as GKS except they are extended to work in the 3D environment. Existing GKS functions can be called and produce the equivalent GKS-3D primitive on the Z=0 plane (effectively, a Z=0 coordinate is added to each position in a GKS function call).

In GKS-3D and PHIGS, text, fill area, and cell array remain planar primitives but can be positioned in an arbitrary plane. Polyline and polymarker become genuine 3 dimensional primitives with no constraints on the positions used in the function call.

One additional primitive has been added, FILL AREA SET, which specifies a set of fill areas all of which will be patterned together as a single entity. For rendering 3 dimensional objects, it was believed that this extension was necessary. For the same reason, FILL AREA SET has more control on how the boundary edge is rendered than the original fill area primitive.

5.3.6 Input

The input models in all three standards are very similar. Both GKS-3D and PHIGS extend the logical input devices by allowing 3 dimensional LOCATOR and STROKE devices as well as the six logical input devices defined in GKS.

PICK input in PHIGS has been extended to give more information about what has been picked. In GKS and GKS-3D, the PICK device returns the name of the segment and the PICK identifier within the segment. As a structure in PHIGS could be executed as part of one or more parent structures, some applications will need to know more than just the local structure name. Consequently, in PHIGS it is possible to recover the names of the structures in the hierarchy that led to the invocation of the structure that has been picked.

5.3.7 Summary

This section has only given an informal introduction to the new facilities available and the differences between PHIGS and GKS-3D concentrating on the ones that are major.

There are many minor differences which, in some cases, turn out to be applicable in the other environment. Consequently, as both standards are progressing together, there tends to be a retrofitting of extensions from one to the other.

The aim is to only have differences between the two standards where absolutely necessary to allow them to perform effectively in their defined user community. Consequently, many minor differences now will be sorted out during the review process.

6. WINDOW MANAGEMENT

6.1 Introduction

With the advent of single user systems with bit map displays, it is possible for a single operator to work effectively on a number of separate tasks allowing the window manager to provide individual windows which appear to the operator like separate devices. For a single application, it is possible for the window manager to multiplex several workstations on the same bit map display. The graphics system needs the window manager to define areas of the screen as virtual devices. The window manager needs the graphics system to define icons, window borders etc. A key problem is how should the two relate. This section will pose some problems and indicate some possible solutions. For further discussion of these issues see [23].

6.2 History

The earliest mention of a window management system is in Alan Kay's 1969 thesis [24] at the University of Utah where he states that 'the display for FLEX was a large virtual screen on which displays may be tacked rather

like notices on a notice board'. The major input device associated with window management systems is the mouse which Doug Engelbart designed at Stanford Research Institute in 1963. These dates show that the concepts were available early on but progress was restricted to a few research laboratories due to the high cost of hardware.

The first realisation of a window management system was at Xerox PARC where Larry Tesler, Dan Ingalls and Alan Kay produced a system for SMALLTALK which introduced the overlapping window paradigm. Windows on the screen could be put on top of one another like papers on a desk. Operations were provided to rearrange the order of the window. In the early SMALLTALK system, you could only interact with the top exposed window.

Later systems at Xerox PARC (several produced by Warren Teitelman) introduced and experimented with features such as scrolling, window borders, pop-up menus, icons etc. the overlapping window paradigm was questioned and tiled window managers were tried whereby individual windows were carefully positioned in a regular order on the screen with no overlapping.

The appearance of the Three Rivers PERQ meant that there were commercially available single user systems with high resolution bit map displays capable of supporting a window management system at low cost. APOLLO and SUN systems soon followed and the appearance of the MACINTOSH at the low end of the market made window management systems and the associated paradigm widely available.

6.3 Model

A general model of window management systems is given in Figure 8.

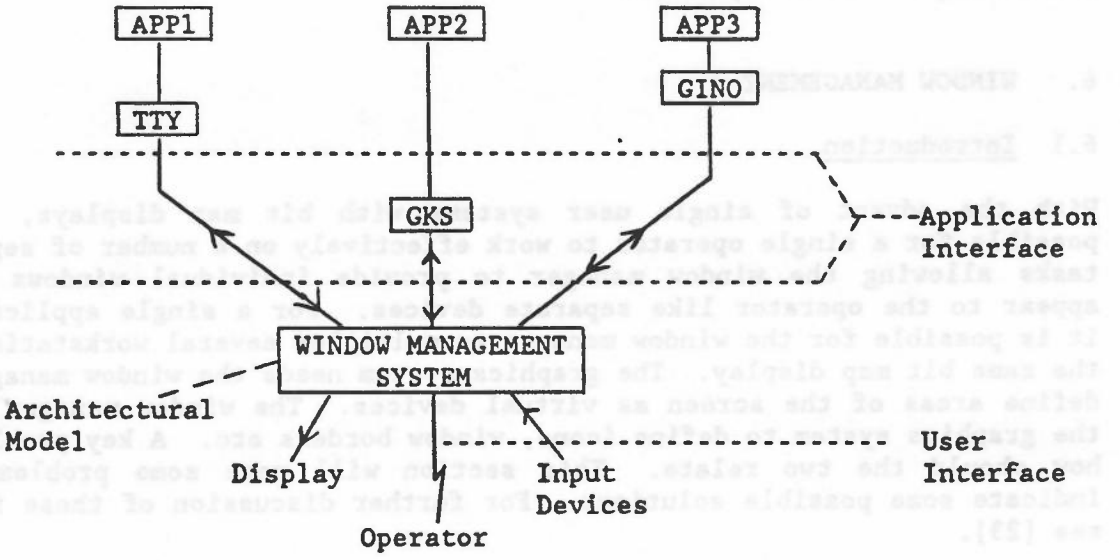


Figure 8

Three major areas for discussion are:

(1) Application Interfaces: at what level should the application interface be placed. At the two extremes you could have the graphics system producing pixel changes in windows or being incorporated as part of the window management system. Almost certainly, the interface will be somewhere between those two extremes.

(2) Architectural Model: both the user and the application need to understand the architectural model of the window management system. As can be seen from the history, various facilities have been tried without them being put together in a sensible reference model.

(3) User Interface: many of the existing systems appear the same on the surface but frequently have different philosophies once the surface is scratched. Some consistency in design principles, while still leaving the ability to do commercial tailoring, is essential.

The following sections will point to some of the issues.

6.4 Application Interface

Standardisation at the Application Interface is important now. Without it, much of the application portability achieved by the introduction of graphics standards will be lost. This is already being seen. At RAL, we have attempted to port applications from one window management system to another with enormous difficulties due to the differences in philosophies adopted.

A major issue is whether the application can control the amount of real estate allocated to it by the window manager. For example, if an application requires a window of a certain size, who is responsible for its allocation? If the operator has to define all windows, it may be difficult for the application to signal its needs and may be tedious for the operator. If the application is allowed to do it, there is the possibility that it will monopolise the resources available making it impossible for other applications to run. For example, one window management system on the market allows an application to seize the whole of the display space and inhibit interrupts causing the whole system to be locked solid. If the window manager itself allocates screen space to windows, it may not allocate sufficient space for the application to run. This is the typical dilemma of the person defining the window management system.

The application/graphics package needs to know what facilities are provided by the window manager. It is generally assumed that the window manager will draw lines, for example. Will it support GDPs and patterning?

Many window systems provide some ability to split windows into sub-parts. Often frames are provided with control information and titling. Again, what control does the application have over these

activities? Can it provide the title information? Can it redefine the control functions?

6.5 Architectural Model

Probably the major question concerns the management of the overlapping window paradigm. When a window is uncovered, moved or deleted, who has responsibility for repainting the pixels? If it is the window manager, it needs to keep bit maps of all the windows and if they are in colour, that could be very large. If it is the application, this may require unacceptable restructuring of the application to allow it to accept random demands to repaint the picture. However, if storage is a major constraint, it may be the only acceptable approach.

Any architectural model needs to define the relationship between windows and icons and the extent to which windows can be grouped together or sub-structured. Some window managers use icons as alternatives to windows while others use them as well as windows. Sometimes they relate to processes rather than windows. If windows can be sub-structured, what is made available to the application? Can it, for example, change the cursor echo as the cursor moves between sub-windows?

Most window systems have some concept of the current window which is listening to the input device. How is that window specified and is there a single window which listens to all input devices or a listener per input device?

A frequent requirement is to move information from one window to another. Can this be done between windows of different applications? What is the structure of the information being moved?

6.6 User Interface

The current window systems have confusing sets of concepts. Facilities available in one are often achieved in a significantly different way in another. As with driving a car, minor variations are acceptable and often desirable but major changes, like putting the brake pedal where the accelerator is, are unacceptable and dangerous. Icons are a good example of a facility which has significantly different meanings in different implementations.

The window management system is a necessary evil and should be as unobtrusive as possible. In the user interface context, there should be simple functions for manipulating windows, undoing actions, etc. If possible, the window manager should not constrain the operator and application. Having dedicated resources for the window management system is probably unacceptable. Thus a window manager might use buttons as accelerators for window manager functions but it should be possible to return the use of that button to the application.

6.7 Summary

This section has indicated the many problems to be addressed before any uniform methodology for window management systems can be achieved. Standardisation of the application interface has just started but it will be many years before substantial progress is made.

REFERENCES

[1] F.R.A. Hopgood, D.A. Duce, J.R. Gallop and D.C. Sutcliffe, 'Introduction to the Graphical Kernel System (GKS)', 2nd Edition, Academic Press, 1986.

[2] 'Information processing systems - Computer graphics - Graphical Kernel System (GKS) functional description', ISO 7942, ISO Central Secretariat, Geneva, August 1985.

[3] K.W. Brodlie and G. Pfaff, 'An Algorithmic Interpretation of the GKS TEXT Primitive', Computer Graphics Forum, 2(4), pp. 233-241, North Holland Publishing Company, 1983.

[4] K.W. Brodlie, D.L. Fisher, G.G. Tolton and T.W. Lambert, 'The Development of the NAG Graphical Supplement', Computer Graphics Forum, 1(3), pp. 133-142, North Holland Publishing Company, 1982.

[5] D.A. Duce and E.V.C. Fielding, 'Better Understanding through Formal Specification', Computer Graphics Forum, 4(4), pp. 333-348, North Holland Publishing Company, 1985.

[6] D.S.H. Rosenthal, J.C. Michener, G. Pfaff, R. Kessener and M. Sabin, 'The Detailed Semantics of Graphics Input Devices', Computer Graphics, 16(3), pp. 33-38, 1982.

[7] C. Stoll, 'GKS for imaging', Computer Graphics, 18(3), July 1984.

[8] K.W. Brodlie, 'GKS Certification - An Overview', Computers and Graphics, 8(1), pp. 5-12, 1984.

[9] K.W. Brodlie, M.C. Maguire and G.E. Pfaff, 'A Practical Strategy for Certifying GKS Implementations', Computers and Graphics, 8(2), pp. 125-134, 1984.

[10] M.C. Maguire, 'Visual Testing of GKS at the Human Interface', Computers and Graphics, 8(1), pp. 19-28, 1984.

[11] A. Ducrot, A. Lemaire, H. Watkins, 'A GKS Implementation for Meteorological Applications', Eurographics 81 Proceedings, North Holland.

[12] D. Rosenthal, P. ten Hagen, 'GKS in C', Eurographics 82 Proceedings, North Holland.

[13] R. Buhtz, 'CGM - Concepts and their Realisation', Eurographics 82 Proceedings, North Holland.

[14] M. Slater, R.J. Baker, 'GRAPH: An Interactive Program based on the Graphical Kernel System', Eurographics 82 Proceedings, North Holland.

[15] I. Herman, T. Tolnay-Knefely, A. Vincze. 'XGKS - A Multitask Implementation of GKS', Eurographics 83 Proceedings, North Holland.

[16] R. Bettarini, G. Faconti, L. Moltedo, 'Extending GKS to a Distributed Architecture', Eurographics 85 Proceedings, North Holland.

[17] I. Herman, J. Reviczky, T. Tolnay-Knefely, 'A Concept for a GKS Machine', Eurographics 85 Proceedings, North Holland.

[18] C.D. Osland, 'Case Study of GKS Development', Eurographics Tutorials 83, Springer Verlag, 1983.

[19] C.N. Waggoner, C. Tucker, C.J. Nelson, 'NOVA*GKS, A Distributed Implementation of the Graphical Kernel System', Computer Graphics, 18(3), July 1984.

[20] R.W. Simons, 'Minimal GKS', Computer Graphics, 17(3), July 1983.

[21] I. Herman, J. Reviczky, 'A General Device Driver for GKS', Computer Graphics Forum, 4(3), Sept 1985.

[22] J.R. Gallop, C.D. Osland, 'Experiences with implementing GKS on a PERQ and other computers', Computers and Graphics, 9(1), 1985.

[23] F.R.A. Hopgood, D.A. Duce, E.V.C. Fielding, K. Robinson and A.S. Williams, 'Methodology of Window Management', Springer Verlag, 1985.

[24] A.C. Kay, 'The Reactive Engine', PhD Thesis, University of Utah, 1970.

cb/M9/CGP2