Data Handling Division
DD/87/4 rev.
David MYERS
July 1987

[REDACTED]

CERN LIBRARIES, GENEVA

CM-P00059865

Interactive Computer Graphics and PHIGS

(Submitted to the proceedings of the 7th Summer School on Computing Techniques in Physics, 9-18 June, 1987, Bechyne, Czechoslovakia, and also to Computer Physics Communications)

Interactive Computer Graphics and PHIGS

David R. Myers

Data Handling Division, CERN, Geneva, Switzerland.

January, 1987.

Submitted to the proceedings of the 7th Summer School on
Computing Techniques in Physics, 9-18 June, 1987,
Bechyne, Czechoslovakia,
and also to
Computer Physics Communications.

Abstract

This paper attempts to provide an overview of the techniques and system requirements of Interactive Computer Graphics. It then discusses in more detail the proposed international standard PHIGS: *The Programmer's Hierarchical Interactive Graphics System*. Finally, some outstanding problems in the area of graphics standards are mentioned.

Chapter 1

What is Interactive Graphics?

1.1 Introduction

When I was asked to write this paper it was suggested that an appropriate length might be 10 to 15 pages. However, as I soon filled more than 12 pages with a list of topics to cover, it should be clear that such a paper can provide only a brief summary of the subject. A certain familiarity with basic graphics terminology has been assumed, and little space devoted to historical details or previous generations of technology. Thus, the environment is assumed to be GKS or PHIGS using raster display systems. Readers not familiar with the jargon will find a short explanation in [1] but, for those wishing to delve deeper, I would recommend the text by Foley and van Dam [2] as being one of the best places to start. Note that a paper by C. Osland on GKS should be published as a companion to this one.

Computer programs invariably manage an application *database* which is modified during execution. In a batch system the way in which the database may be changed is completely determined before the program is started. However, the essence of interactive software is that the user may alter the program flow during execution. We may say that an interactive program is *event driven*, in that it waits for an external event before responding. These events may consist of a typist hitting a key on a word processor, or a pilot moving the joy stick in an aircraft simulator. For both examples it is clear what would be the immediate effect on the database. In one case a new character would be added to a document and, in the other, the variables defining the attitude of the (simulated) aircraft would be changed. It is also clear in these two examples that the program has to do more than just modify the database; it must provide some *feedback* to show the user that the command was understood. This might consist of echoing a character in one case and of dynamically transforming a very complicated image in the other.

The word processing example is interesting in that it may soon become one of the few interactive applications which does not necessarily incorporate graphics, although, even in this case, systems with graphics capability are being used in order to display a wide range of proportionally-spaced fonts. In other words, interactive graphics is becoming synonymous with interactive computing. Whereas programs previously made use of graphics only if they were designed specifically to display images, interactive graphics has now become a tool to be employed as the obvious way of communicating with computer users. Applications include data presentation and business graphics, process control, simulation, animation, Computer Aided Engineering and Instruction (CAE & CAI), molecular modelling, and many more.

In the commercial market the Apple LISA and MACINTOSH personal computers were the first machines to become widely available with high-quality graphics capabilities. Apart from the Xerox STAR, these machines were the first to provide multiple overlapping *windows* and *icons* (see Figure 1 on page 15) as a way to make computers easier for use by non-specialists. The key point is that the display screen corresponds to a desk-top, and the windows to pieces of paper. Just as one can pick up and read any piece of paper off a desk without filing away all the others, so one can work with the contents of any window on the screen at any time. Thus, the user retains flexibility and is not locked into a particular mode of working, nor forced to carry out actions in a pre-determined sequence, which is often the case when using a 'dumb' terminal. If you forget an address whilst typing a letter into one window, you simply look it up using another one. The basis for these ideas rests on work done at the Xerox Palo Alto Research Center during the 1970's in the context of the SMALLTALK system [3,4]. However, it is only recently that the cost of hardware has dropped sufficiently for such features to become the norm, and this low-cost hardware is generating many new applications.

1.2 System Requirements

One of the first discoveries to be made by those new to computer graphics is that it has an insatiable appetite for processor cycles. Even before considering the resources required by the application itself, whether mechanical CAD or simply document preparation, there is almost no limit to the computing power which can be expended to display the results. Thus, as in many human endeavours, the requirements of an interactive graphics system are limited by economics. For example, to produce laser-printer quality text on an A4 sheet of paper requires about 8 million *pixels* (picture elements). By comparison, currently-affordable *bit-mapped* display systems, which now essentially have replaced all other types of display technology, provide a *frame store* containing about an order of magnitude less pixels (corresponding to 1024x1024 pixels on a 38 cm screen). The colour and/or intensity of each pixel written to the screen is controlled by a set of one or more memory bits which map from rectangular bit arrays, or *bit-planes*, to the display screen (see Figure 2 on page 16). Considering that to avoid flicker these pixels must be refreshed at 60 Hz, one immediately arrives at a point which has been a major stumbling block; namely the necessity for a bandwidth of 60 MHz per bit-plane between the frame-store and the screen. Thus, for a display providing 256 colours and hence employing 8 bit-planes, one has to read 8 bits from memory every 16 ns. Another problem is that to produce a picture, even assuming only 5% of the screen contains information, requires calculating the values of at least 50'000 pixels by *rasterizing* a description of the image defined in terms of graphics primitives. To simulate motion this must be repeated at 15 to 20 Hz, which requires the execution every second of many millions of instructions, and hence the investment of considerable sums of money.

For applications which do not require animation, a response time of about a second is an acceptable upper limit. If the time to respond to input becomes longer than this then a good system should have a mechanism to inform the user of how long (s)he will have to wait. Note that some investigators [5] believe that the non-variability or, better, predictability of response is more important than the absolute length of time one has to wait.

It is perhaps interesting to contrast the current refresh raster display technology with the previous use of TEKTRONIX 4014-style Direct View Storage Tube displays. These had very high resolution (up to 4096x4096 pixels) but very poor possibilities for interaction, and were inherently monochrome. The rasterization process was not necessary as the vectors were stored directly on the screen, rather than in a bit-mapped frame-store. This had another advantage in that vectors were always straight lines without the jagged edges produced by rasterization. Although some raster display systems use *anti-aliasing* techniques to make vectors appear less jagged, in some cases this tends effectively to reduce resolution by broadening the lines.

A major result of machines such as the MACINTOSH becoming consumer items has been that users' expectations have increased. A decreasing proportion of applications will remain for simple graphics terminals because the bandwidth of an asynchronous communications line is much too slow to support high-quality interaction, which requires very close coupling between the processor and the screen. Graphics terminals may be improved by introducing local processing capabilities and *display-list* memory (see next section). The use of high-speed Local Area Networks (LANs), such as Ethernet, could also improve the situation regarding bandwidth, although there are available as yet rather few graphics terminals with LAN interfaces. However, the trend is now to personal workstations incorporating bit-mapped display systems, such as those from APOLLO, DEC, and SUN, in which the processor has direct access at full memory speed to the image representation.

A requirement for any interactive system is that it must be both robust and user-friendly. Robustness means that users should never be able to break the system by providing an unforeseen input sequence. All error conditions must be trapped and reported by comprehensible messages. User-friendliness is a somewhat vague term which I believe should encompass the following points: clear prompts must be issued so that the user is aware of the choice of options provided, and help information should be available at any time; user actions should always produce predictable and consistent results without unexpected side-effects and, as previously mentioned, feedback must be provided to

indicate that the system understood each command. Apart from displaying messages or transforming the image, feedback can also be provided by *highlighting* objects by making them blink, or by changing their brightness or colour. Finally, user mistakes should not be irrecoverable. For example, the MACINTOSH allows the user to 'un-delete' the last file that was thrown away.

1.3 Changing What You See is a Question of Architecture

A graphics image is composed of a set of *primitives* (lines, text, filled areas, ...) bound to *attributes* (colour, character height, texture, ...). It has already been stated that an interactive system allows the user to initiate events which modify the contents of an application database. How does this change the image seen by the user? The strategy which is conceptually the most simple, and that which was very often used in the past, is to actuate some module of application code whenever the application database is modified. This would clear the display and then process the database to generate a new image, which can be very time consuming. A more sophisticated approach would be to re-draw *only* those parts of the picture which have been changed. This is potentially much faster, but there are *snags*. For example, if the user wishes to remove a line from the image this might be attempted by re-drawing it in the background colour. However, if the background was blue and the line to be removed happened to cut across the edge of a red polygon, then re-drawing it in the background colour would leave a blue line across the corner of the polygon!

There is no solution to this problem short of regenerating the entire image. However, we are not obliged to regenerate the image starting from the application database. Instead, we can make use of an intermediate *display-list* in which to store a description of the image in graphical form. In this case it is only necessary to change those display-list elements corresponding to the edited part of the database, and then use the updated list to regenerate the image. The scheme has the added advantage that much less data needs to be transmitted if the display-list is stored in a memory local to the device (see Figure 3 on page 16).

Unfortunately, this solution is not so simple as it might appear at first sight. The phrase 'in graphical form' poses a multitude of questions. If the display-list consists of a simple list of primitives and attributes, then should they be stored in the user's world coordinates system, or in the display hardware reference system? Can we afford to use a floating point number representation, or are we restricted to 16 bit (or even 10 bit) integers? If the display-list is unstructured then it becomes very difficult to edit. However, if it is to be structured, should it have a single level of segmentation, like CORE [6] and GKS [7], or should it have a hierarchy like PHIGS [9]? Are we allowed to edit the contents of a segment, or have we to replace the entirety of those which must be changed? How is display-list memory management to be handled?

Over the last 10 years almost all combinations of answers to the above questions have been implemented by one manufacturer or another. In some cases the solutions have incorporated several levels of display-lists, for example, a structured display-list above a linear one was implemented for the Evans & Sutherland PS200. Actions which can be performed at the lowest level update the image quickly, whilst more complicated actions which require traversing the higher levels take longer. Graphics users and programmers may not have access to all these levels directly, but often have to be aware that they exist if efficient use is to be made of the system.

As hardware has become cheaper and faster the tendency has been to simplify the architecture. However, from time to time a new threshold is reached and an attempt is made to add some extra feature which again requires a more complicated design. For example, when using a single level 2D display-list, projections from 3D had to be carried out by the host processor. Later, when the low level display-list became capable of handling 3D *wire-frame* (vector) graphics, both surface shading and hidden line/hidden surface removal had to be handled at a higher level. The situation is now being repeated in relation to support for smooth shading techniques and lighting models. This cycle was noted already in 1968, and has been called 'the wheel of reincarnation' by Myer and Sutherland [10]. Direct

image generation from the application database is clearly the cleanest solution, and this is the direction in which graphics architecture is heading. It may seem ironic, but when hardware becomes fast enough for this to happen the wheel will have come full circle, as this is the architecture with which we started.

Whilst dealing with graphics system architecture, the introduction from GKS of the *Workstation* concept should be mentioned. This provides an interface layer within the graphics package below which are situated a set of workstation drivers. The workstations simulate in software any features which are not provided by a particular display, and so simplify the problem of driving new devices. There are difficulties, however. If the interface level is too high, then most workstation drivers must incorporate a great deal of code. If, on the other hand the level is too low, then the software will not be able to make use of advanced features in high-performance display systems. Workstation drivers may not be moved from one supplier's implementation of a graphics package to another until the interface, called the *Computer Graphics Virtual Device Interface* (CGI), is standardized [11]. This may prove to be a lengthy business, as even a draft standard is not expected before mid-1988.

As a final point in this section, the emerging *X-Windows* de-facto standard can not be passed over. *X-Windows* originated in the Athena Project at M.I.T., and defines a communication protocol supporting a window manager and graphics interface. Thus, using *X-Windows*, it is possible to run a graphics application on a remote mainframe with a large database and powerful cpu whilst using the interactive capabilities of a personal workstation. The protocol provides the ability to create windows and move them around the screen, to change their size, and so on, without the users being concerned about whether their programs are running locally or remotely.

However, one of the main reason why *X-Windows* is so interesting is that more than a dozen major computer manufacturers have announced that they will support it, and this has the advantage that it will become possible to split graphics applications across hardware from multiple vendors. The *X-Windows* protocols sit on top of a network transport layer, such as the Transmission Control Protocol/Internet Protocol (TCP/IP), which itself sits on top of the network hardware, such as Ethernet. In other words, *X-Windows* is situated above level four in the OSI network model, and so is independent of the network transport layer used. Although the implementation of *X-Windows* written at M.I.T. is coded in C to run under UNIX, one may expect to see implementations for many other operating systems.

1.4 The Graphics Pipeline

The previous section discussed some of the practical difficulties in the design of graphics system architectures. We shall now consider another aspect of the problem. Irrespective of whether the primitives are generated directly from the application database, or from a display-list, they must pass through a series of operations before reaching the display surface. This process is performed by the *Graphics Pipeline*. Although variations are possible, the basic steps for a 3D pipeline are as follows (Figure 7 on page 18 shows the PHIGS case):

1. The attributes controlling the appearance of each primitive are *bound* (attached) to the primitives.
2. If primitives have been defined in different coordinate systems then they must be transformed to a common reference system. This transformation is called *Normalization* in GKS, and *Modelling* in PHIGS. However, these transformations are not completely equivalent, as GKS Normalization allows for a shift and a scale but no rotation. There may also be a clipping operation at this stage.
3. Having assembled the components in a unique *World Coordinate System*, the next operation is to 'view' them; in effect, to orient the model in relation to the eye point. This requires definition of the *View Reference Point* (within the object to be viewed), the *View Up Vector*, and the

View Plane Normal (see Figure 4 on page 17).

4. Primitives may next be clipped to a box to remove extraneous information, and are then projected. Apart from the clipping planes and indicators, this involves specifying the *View Window*, *View Plane*, *Back Plane*, *Front Plane*, and the *Projection Reference Point* (see Figure 5 on page 17).
5. At this stage there is another clipping operation to ensure that no primitives lie outside the Workstation Viewport, and there may also be a *Workstation Transformation* to position the projected image on the display surface.
6. Workstation-Dependent attributes are now bound to primitives which are then *rendered* onto the device. For example, on a bit-mapped display the pixel positions and colours corresponding to each primitive must be calculated.

A good implementation of the graphics pipeline will attempt to combine as many as possible of these stages using matrix concatenation in order to reduce the computational load. Unfortunately, the clipping operations are a problem as they take place in intermediate coordinate systems. In theory the clipping can all be performed at one place by transforming the clipping planes. However, if any of the transformations include rotational components, then the potential savings are lost due to the complexity of the resulting clipping volume.

Some display systems with high-level display-lists provide a mechanism to vary the parameters of the pipeline locally without reference to the application. These *Viewing Engines* allow the application to insert graphical objects into the display-list which can then be looked at from different angles, or be clipped at different planes, without loading the host processor. At a more basic level many display terminals provide *image transformations*, which allow the user to *zoom* or *pan* the image in two dimensions, by performing operations directly at the level of the frame-store. There have also been successful attempts to construct VLSI chip sets to carry out the viewing pipeline operations [12].

1.5 The Input Model

Apart from a keyboard, an interactive graphics system requires some mechanism with which to move the cursor or the image, such as a *tablet*, *joystick*, or *mouse*. It may also have *function keys* and a set of dials used as *valuators* to provide analogue input values. The GKS input model, which has now become widely accepted, defines six *Logical Input Devices* as idealized versions of the above. Namely:

Locator	returns the cursor position,
Stroke	returns a set of cursor positions (defines a polyline),
String	returns a character string,
Choice	returns a selection from a set (menu) of possibilities,
Pick	returns the name and pick identifier of the picked segment,
Valuator	returns a value as a real number.

Thus, the programmer no longer has to worry about the sordid details of a particular device as these are taken care of by the graphics package. Note that the pick device is a new concept. Rather than the package returning the cursor position and then letting the application discover if it corresponds with a graphics object, the pick device does the work instead and returns the identity of the picked object directly. Another innovation is that the locator returns a value in the user's coordinate system, rather than that of the display device, by performing the requisite back transformations. Of course, with 2D physical devices it is a problem to input the third coordinate for 3D systems.

A *Prompt* and an *Echo* type is defined for each logical input device. For example, enabling the pick device might prompt with a cursor of a particular shape which would track the tablet or mouse to

constitute an echo. Acceptance of a *trigger* by the application causes feedback via an *acknowledgment* process. For example, the picked object could be made to blink. Logical input devices can be operated in three modes: *Request*, *Sample*, and *Event*. In request mode the application enables a device and then waits for the user to trigger input, perhaps by pushing a key. In sample mode the application can *measure* the current value of the device, for example the locator position, without waiting for a trigger. Finally, in event mode, the application can enable a set of several logical devices simultaneously. Output from devices which have been triggered will be placed in an *Event Queue*, from whence they can be extracted by the application.

Note that the model just described, although very powerful, has several deficiencies. This is especially true for GKS, where the lowest input level, to which most implementations correspond, provides only Request Mode. The most obvious problem is that no provision is made for input caused by multiple logical devices. As an example, if input is requested from the Logical Locator, and this is triggered by a key stroke, no provision is made for informing the application of *which* key was hit. Thus, amazing as it may seem, most implementations of GKS do not support the ubiquitous input model of a TEKTRONIX 4014 display, which returns the trigger character along with the cursor coordinates.

Due to these deficiencies, discussions will almost certainly take place on ways to expand the input capabilities for future versions of the GKS and PHIGS standards. The discussions may be organized either within the PHIGS Working Group, or within a new Working Group set up specifically to review the input model for all graphics standards.

Even without radical changes, there are several extensions which could be made to the current model. The first one would be simply for the standard to include more than the six existing logical device types. As an example, an N-dimensional valuator would allow input from a 6D airplane joystick. Another possibility would be to allow the application dynamically to define Composite Logical Device types. This would solve the problem of how to find out which key triggered a locator input. Going further than this, one could envisage providing local input processing by application-defined tasks. An example of this would be to change a viewing parameter, or modelling transformation, as a function of some input measure without having to close the loop via the application program. Unfortunately, this would require ADA-style multi-tasking facilities if it were not to be very implementation dependent, and this poses serious language standardization problems.

Chapter 2

The Programmer's Hierarchical Interactive Graphics System

2.1 Why PHIGS?

Just as for GKS and GKS-3D, the PHIGS standard consists of a functional description and a separate set of language bindings (for FORTRAN, ADA, C, and PASCAL). At the time of writing, PHIGS itself is at the level of a Draft Proposed International Standard, and there will be a vote on whether or not the document is acceptable as a Draft International Standard towards the end of 1987. Thus, if all goes well, PHIGS could become an International Standard before the end of 1988, and so it is unlikely that there will be more than minor changes to the technical content.

Intentionally, a great deal of PHIGS is identical to GKS-3D, including the primitives, the attributes, the workstation concept, and the viewing and input models. To answer the question as to why one might need another standard so similar to an existing one, perhaps its worthwhile to review the differences.

1. PHIGS has a *Centralised Structure Store* (CSS) which is a conceptually centralized hierarchical graphics database, and may be thought of as a sophisticated display-list. Unlike GKS *Workstation Independent Segment Storage*, data in the CSS is available to all workstations at all times (even though this might be difficult to implement). There is no equivalent to GKS *Workstation Dependent Segment Storage*. The CSS provides all the functionality necessary to build general acyclic directed-graph data structures (networks), and supports dynamic editing.
2. Picture creation is completely separate from picture display. Editing the CSS and viewing its contents are logically asynchronous processes, and so workstation control is performed differently to GKS.
3. Attributes are bound to primitives when they are displayed, not when they are created. This allows the concept of attribute inheritance.
4. The PHIGS viewing pipeline is conceptually cleaner than for GKS-3D; normalization and segment transformations are replaced by multi-level modelling transformations.
5. PHIGS has no Non-Retained Data; primitives to be output **must** be stored in the CSS.

Although there have been suggestions to combine PHIGS with GKS-3D [13], it is now generally accepted that their incompatibilities are sufficiently great to warrant that they should be 'considered to be distinct within a family of graphics standards'. However, by the inclusion of a 'thin shell' of code, it should be possible to run a GKS or GKS-3D program on top of PHIGS. Clearly there will be an overlap region of requirements where applications could use either standard. However, PHIGS is aimed particularly at highly interactive applications with complicated hierarchical data structures. For example, Mechanical CAD, Molecular Modelling, Simulation and Process Control. Another implicit assumption made by the PHIGS standard is that implementations should run on very high-performance display systems, whereas this is not a requirement for GKS.

2.2 The CSS: Hierarchy and Modelling

The heart of PHIGS is the CSS, which is itself composed of *Structures* accessed via application-specified identifiers. Structures contain ordered sets of *Structure Elements* which are the basic data entities and which, unlike GKS segments, do not contain headers. Thus, there are no specific structure attributes, and structures may contain any combination of elements. This lack of mandatory header information within structures allows them to be implemented with low overheads, and so there is no substantial penalty to pay for building complicated structure networks.

An essential concept to understand is that an application program can create and edit structures without producing an image on any display. To produce a picture some structure must be *posted* to a workstation at which time PHIGS conceptually performs a tree-traversal of the CSS starting from this *Root Structure*. The example given in Figure 6 on page 18 shows how graphs may be formed by structures invoking other structures, and so forming a hierarchy. The traversal process visits all structures in a network which are reachable from the root.

Another important feature is that modelling transformations may be inserted in a structure before invoking lower-level structures, and these transformations may be concatenated at multiple levels. Thus, the fingers of a robot can be oriented and invoked with reference to the wrist, and the lower arm can be oriented and invoked with reference to the upper arm, etc. Changing the modelling transformations will result in the relative movement of the lower level structures.

Unlike previous graphics systems, such as CORE or GKS, the application does not call functions which send primitives to a device, but builds up a graphical database containing structure elements, only some of which define primitives. Once posted to a workstation any changes made to a structure network would normally be reflected automatically on the display, although mechanisms are provided to control the details of how and when this happens. Thus, the CSS appears to be a database which, as a 'side effect', produces graphics output.

2.3 CSS Structure Elements

Primitive and Attribute Structure Elements produce on traversal the same primitives and attributes as for GKS-3D, with one exception. The primitives are: polyline, polymarker, text, annotation text, cell array, fill area, fill area set, and the Generalized Drawing Primitive (GDP). The exception is annotation text, which allows points within a model to be annotated with text strings which are always written in a plane parallel to the view plane in order to remain readable. Primitives may be either 2D or 3D and, unlike GKS-3D, 2D primitives are not converted to 3D before being stored in the CSS. The attribute types are too numerous to list here. Note that primitive and attribute structure elements are **NOT** the same as primitives and attributes, which are only produced at structure traversal time before being sent to the workstation. This is a subtle but important distinction. As for GKS, attributes are modal, and affect all the primitives which follow. They may also be defined by *attribute bundles* or individually, under control of *Aspect Source Flags*.

Modelling Transformation Elements consist of 4x4 *affine* homogeneous matrices. An affine transformation is a linear transformation plus a translation, and so allows for scaling, rotation, and translation of the model.

Modelling Clipping Volume Elements allow clipping acceptance regions to be built up by applying boolean operators to half-spaces defined by points and normal vectors. The resulting clip volumes may be very complicated, and are not restricted to having planes parallel to the axes of the modelling coordinate system in which they are defined.

Execute Structure Elements are analogous to subroutine calls in a computer language and allow the designated structure to be executed within the context of the caller. There is no explicit parameter

passing, but the executed structure inherits the attributes and transformations currently in effect.

View Selector Elements are indices used to specify which viewing transformations are to be used to process the primitives which follow when the structure is posted to a workstation. The viewing parameters corresponding to a particular view selector are workstation dependent. If several view selectors are present in a traversed structure network then the appropriate parts of the network will appear in each viewport. For example, stereo images could be produced by having two views of the same structure network with slightly rotated view plane normals.

Label Elements are used to navigate within a structure when editing. They are ignored during structure traversal.

Pick Identifier Elements are associated with the primitives which are traversed after them. They are returned to the application if these primitives are chosen by a pick operation.

Application Data Elements consist of any information of use to the application. They are ignored by PHIGS during traversal.

Name Set Elements contain groups of names which are associated with the name set attribute of the following primitives. These names are used in *Inclusion* and/or *Exclusion* filters to control the visibility, highlighting and pickability of primitives without editing the structure. For example, to be pickable a primitive must have a name in the pickability inclusion filter but no name in the pickability exclusion filter. Filters are workstation dependent.

Generalized Structure Elements (GSEs) provide an escape mechanism to store in a standard way within the CSS attribute and control information which is dependent on a particular PHIGS implementation.

2.4 Picture Generation: Traversal, Inheritance and Attribute Binding

During traversal, structure elements are processed sequentially starting with the first element in the posted 'root' structure until an Execute Structure Element is encountered. Having saved the current structure's state on the stack (attributes, modelling transformations and clip limits, view indices etc.), the invoked structure network is traversed. The state is then restored and traversal resumed at the next element of the original structure. Traversal implies that invoked structures can and do *inherit* attributes from their ancestors, but children can not influence their parents.

Consider the following sequence of elements: {Set modelling transformation; Set colour attribute 'red'; Execute 'FLAG'; Set modelling transformation; Set colour attribute 'blue'; Execute 'FLAG'}. This fragment of a structure *instantiates* the FLAG object twice in different places and with different colours. Of course, there must be no elements within the FLAG structure which would override the inherited transformations or attribute settings. This possibility of instantiation allows for the re-use of structures, leading to reduced database size.

An important property of the CSS is that structure networks can be built 'top down', which is the way applications usually work, and their organization may reflect the geometric structure of an application model. Thus, it is possible to invoke lower-level structures which have not yet been defined without producing a traversal-time error. The effect is equivalent to creation of an empty structure.

2.5 The Viewing Pipeline: Coordinate Systems and Transformations

The sequence of actions performed on primitives generated during the traversal process is shown in Figure 7 on page 18. From the Viewing Transformation onwards, the PHIGS pipeline is similar to GKS-3D. The following Coordinate Systems and Transformations are involved:

The *Composite Modelling Transformation* takes *Modelling Coordinates* (MC3) to *World Coordinates*. The transformation is called 'composite' because it may be composed by the concatenation of multiple matrices. PHIGS also includes the possibility to clip in modelling coordinates.

The *View Orientation Transformation* (see Figure 4 on page 17) takes *World Coordinates* (WC3) to *View Reference Coordinates* (UVN system). The default view table entry (0) maps the Window $[0,1] \times [0,1] \times [-1,0]$ to the Viewport $[0,1] \times [0,1] \times [-1,0]$.

The *Clipping and View Mapping (Projection) Transformation* (see Figure 5 on page 17) takes *View Reference Coordinates* (VRC) to *Normalized Projection Coordinates*, and the projection may be either parallel or perspective. The default view table entry (0) uses the identity matrix for this transformation with clipping limits at $[0,1] \times [0,1] \times [-1,0]$ and all clipping indicators set to 'ON'.

The *Workstation Transformation* takes *Normalized Projection Coordinates* (NPC) to *Display Coordinates* (DC3). This is similar to the image transformations mentioned earlier and can be used for pan and zoom. It preserves the aspect ratio, and includes a clipping operation which cannot be disabled. The default Workstation Window is $[0,1] \times [0,1] \times [-1,0]$. DC3 coordinates may be in meters or raster units.

During traversal, PHIGS maintains three Modelling Transformation (MT) matrices. The Global MT is the composite MT of the parent structure up to the point where the current structure was invoked. The Local MT is the composite MT defined in the current structure up to the last encountered transformation element. Finally, the Composite MT is the composition of the current Local MT pre-concatenated with the Global MT, and is the matrix used to process the primitives. All three are necessary, as new modelling transformations encountered whilst traversing a structure may be pre- or post-concatenated to the current Local MT. A function is also provided to replace the existing Global MT. Note that such modifications effect the rest of the current structure and its descendant networks, but not its ancestors.

2.6 Editing, Name Sets and Dynamics.

At the level of structures, PHIGS provides the following editing commands, which are self-explanatory: *Create*, *Delete*, *Rename*, *Copy*, and *Empty*. Apart from renaming structures themselves, it is also possible to change all the references to a particular structure. There are inquiry functions to obtain a list all existing structure identifiers and structure network tree definitions. PHIGS also allows editing within structures using the concept of an *element pointer*. Having opened a structure, this pointer can be set to a particular element by specifying an offset to the start of the structure, relative to the current position, or relative to a label element. A new element may inserted at this position, or the following element may be deleted or replaced. Inquiry functions are provided to find the position of the element pointer, the type and size of the following element, its contents, and a list of its ancestors or descendants.

These editing capabilities of PHIGS do not exist in GKS, and are capable of supporting highly interactive applications. For example, by replacing a Modelling Transformation Element with a slightly different one at a rate of about 15 Hz it is possible to provide animation effects. However, the following points should not be overlooked. In order to know which structure, and which element within a structure, should be edited, the PHIGS programmer must be very careful to provide efficient mapping between the application database and the corresponding CSS structures. There is also the question of

performance to consider, because once a change is made within the CSS it is difficult to envisage a more efficient strategy to update the displayed image than to re-traverse all structure networks posted to a workstation [14]. For a fast response to changes of non-trivial structure networks this requires purpose-built hardware for CSS traversal, as well as for the viewing pipeline.

Apart from editing the CSS directly, an image can be changed by modifying attribute bundles or viewing parameters stored at the workstation, or by adding or removing names from the Name Set inclusion and exclusion filters. This last mechanism can be a very convenient way to highlight objects of interest. As an example, consider a structure network representing a motor car with different name set attributes for elements of the brake, transmission, electrical, and fuel systems. It would be possible with this arrangement to highlight, make invisible, or make detectable, all components of any one of these systems simply by adding the requisite name to the appropriate inclusion filter.

2.7 Input

The PHIGS input model is almost identical to that of GKS, which has been summarised in an earlier section. The only significant difference is that the Logical Pick Device returns more information than in the GKS case, as it includes the path through the network from the root to the picked structure. Also, as PHIGS has only a single implementation level, Request, Sample, and Event input modes are always available.

2.8 Archiving and Metafiles

PHIGS provides the possibility to save and restore the whole of the CSS, or any structure network, onto one or more *Archive Files*. The archiving system retains all the relationships between the archived structures. Portability of archive files between implementations is assured by the definition within the PHIGS standard of a Clear-Text Encoding. This makes no assumptions about the internal representation of structures within any particular implementation of PHIGS.

PHIGS also incorporates an interface to the *Computer Graphics Metafile* (CGM) [15]. Whilst archive files are restricted to the PHIGS world, CGM metafiles allow for the transfer of graphical data to or from other graphics systems, such as GKS. As the CGM essentially allows a 'snapshot' to be taken of the display surface, this is useful for making hardcopies, or sending finished pictures to a remote site. The input of CGM metafiles to PHIGS would seem to be less useful as each picture within the file is unstructured, although this may change with a later extended version of the CGM standard. In addition to archive files and the CGM, it would be quite straightforward for an implementation of PHIGS to incorporate other types of metafile, such as the GKSM metafile, by providing a workstation to output the information in an appropriate format. Metafile input could be handled by an application-level interpreter.

Chapter 3

Unsolved Problems

Although GKS/GKS-3D and PHIGS have made great strides in improving the situation within graphics standards, lest we become too complacent, it may be worthwhile to consider some of the problems which remain. Taking PHIGS first, the most obvious difficulty is that the standard is more than 350 pages long and includes about 320 functions. There is a good reason to include each one of them. However, this complexity, plus the lack of familiarity with concepts in PHIGS which did not exist in previous graphics systems, implies that PHIGS will not be for occasional users. In fact the

document states clearly that PHIGS programmers should already 'have a working knowledge of Computer Graphics'.

This is not anyone's fault. Graphics at this level *is* complicated. How then do home computers seem to make graphics look easy? The most obvious answer is that graphics on home computers is 2D. Moreover, the software is machine dependent and typically does not easily allow the introduction or transmission of external data. Another answer is that graphics is not really as easy as all that on home computers. The point is that the owners of home computers rarely write their own graphics software. They use programs which someone else took a lot of effort to produce.

Even with all the functionality described, and apart from the problems mentioned with input, PHIGS still has deficiencies shared with GKS. For example, both standards have only very simple graphical primitives. Although PHIGS is aimed at the Mechanical CAD market, there are no primitives to describe solid objects or three dimensional surfaces, such as splines, Bezier curves or meshes. Nor does PHIGS provide for the definition of light sources, or techniques for the smooth shading of surfaces. The CSS has the capability to build acyclic directed graphs, but does not support generalized parameter passing to invoked structures, or a mechanism to provide for conditional structure invocation. Debugging programs which build PHIGS structure networks is also unlikely to be a trivial occupation.

All standards suffer from problems about which the committees could not agree. These become *implementation-dependent* features. A particularly serious current omission is that although there is a standard for character encoding (ISO 2022), there does not exist a standard set of character fonts. Thus, on two different implementation of PHIGS or GKS, not only might font number *n* have a different style, it might also cover a larger area, and so unintentionally over-write another part of the image! Another example is the initialization of input devices to specify prompt and echo types, which requires implementation and device-dependent information to be supplied.

Perhaps one of the most pressing problems today is that none of the current standards effectively address the issue of the so-called *BitBlT*, or *RasterOp*, functionality of bit-mapped display systems. BitBlT (Bit BLock Transfer) functions are logical operations between rectangular arrays of pixels, and are normally supported by very fast hardware. However, the cell-array primitive in GKS and PHIGS is inadequate to support such capabilities, and so provide some support for image processing applications. Window Systems, which depend on BitBlTs, are another area on which a standardization effort is starting, but which are not yet properly integrated with graphics. For example, at the time of writing, X-Windows has yet to be extended to three dimensions.

One must note, however, that BitBlT functions were designed for 2D monochrome displays. These function clearly are not very useful for 3D images, and neither is it obvious how they should operate with colour systems, as performing BitBlT operations bit-plane by bit-plane does not necessarily produce a reasonable result. Thus, in the long-term, high-level support for BitBlT functionality will decline in importance as cheap hardware becomes sufficiently fast to emulate systems, such as those from Silicon Graphics, which are able to regenerate the entire image directly from the display list.

Chapter 4

Conclusion

I believe that we are beginning to see Computer Graphics come of age. Rather than being an end in itself, it is now becoming an integral part of most interactive computing applications. Major strides have been taken in the area of standardization, and the introduction of such concepts as the Workstation and the Logical Device is of great help to the graphics programmer. However, we have also seen that there remain many problems to solve, and so those wishing to work in the area of computer graphics may rest assured that there is still plenty left for them to do in the future.

Chapter 5

References

1. D.R. Myers, *A Background to Computer Graphics* (CERN Yellow Report 83-07, Geneva, July, 1983).
2. J.D. Foley and A. van Dam, *Fundamentals of Interactive Computer Graphics* (Addison-Wesley, Reading, Mass., 1982).
3. BYTE (August 1981), Issue devoted to SMALLTALK-80, pp. 14...
4. A. Goldberg and D. Robson, *SMALLTALK-80: The Language and its Implementation* (Addison-Wesley, Reading, 1983).
5. R.B. Miller, *Response Time in Man-Computer Conversational Transactions*, Proceedings of Fall Joint Conference, AFIPS (1968).
6. *The CORE Graphics System*, SIGGRAPH-ACM Computer Graphics, 13, 3, (August 1979).
7. *The Graphical Kernel System*, International Standard ISO/DIS 7942 (Geneva, July 1985).
8. *The Graphical Kernel System for Three Dimensions*, Draft Proposed International Standard ISO/DP 8805 (Geneva, October, 1985).
9. *The Programmer's Hierarchical Interactive Graphics System*, Draft Proposed International Standard ISO/DP 9592 (Geneva, October 1986).
10. T.H. Myer & I.E. Sutherland, *On the Design of Display Processors*, Communications of the ACM 11, 6 (June 1968) pp. 410-414.
11. *Computer Graphics Interface*, ISO/TC 97/SC21/WG2 N 1406/1411 Working Draft (Geneva, May, 1986).
12. J.H. Clark, *The Geometry Engine: A VLSI Geometry System for Graphics*, Computer Graphics 16, 3 (July 1982) pp. 127-133.
13. S. Daun, L. Froböse, M. Göbel, S. Noll, W. Puchtler, J. Rix, J. Schönhut (Editors) *PHI-GKS Wenschedorf Workshop Report* (Zentrum für Graphische Datenverarbeitung, Bleichstr. 10-12, 6100 Darmstadt, May, 1986).

14. Salim S. Abi-Ezzi and Albert J. Bunshaft, *An Implementor's View of PHIGS*, Computer Graphics and Applications 6, 2 (February 1986) p. 20.
15. *Computer Graphics Metafile for the storage and transfer of picture description information*, Draft International Standard ISO/DIS 8632 (Geneva, February, 1986).

Figure 1: The screen of a bit-mapped workstation showing windows and an icon. Each window corresponds to a separate process, the top one of which is active. A *HELP* window can be brought up at any time. An icon is a symbol which can represent some process or command.

Figure 2: Organization of a Bit-mapped Frame-store. The set of bits from the frame-store corresponding to each pixel is used to look-up the intensities of the three primary colours used to drive the CRT. There may also be an alpha-numeric overlay plane.

Figure 3: An example of a display-list architecture.

Figure 4: The GKS-3D/PHIGS View Reference System. The View Reference Point, defined in World Coordinates, should be situated near the object to be viewed. The View Plane Normal is directed at the eye point. (Reproduced from the PHIGS Working Draft.)

Figure 5: The GKS-3D/PHIGS Projection System. The figure shows the definition of the View Volume with Perspective Projection. (Reproduced from the PHIGS Working Draft.)

Figure 6: Simplified Structure Network for a Motor Car. Note that one copy of the Wheel is instantiated four times.

Figure 7: The PHIGS Graphics Pipeline used to process primitives.