EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN-DD / 89/8

March 21, 1989

11 AVR. 1989

HEP Graphics: Standards and Portability versus Performance and Cost

D.R. Myers

Data Handling Division
CERN
Geneva, Switzerland

Abstract

After mentioning the range of applications for which computer graphics is used in High Energy Physics, some of the characteristics required of graphics hardware are presented, along with a discussion of how well these requirements are likely to be met by the computer industry. Moving on to the topic of software, the status of the more important graphics standards is covered briefly, as well as their evolution in the medium term. The point will be made that HEP, perhaps even more than other areas, has conflicting requirements for graphics systems. The subject is manpower-intensive, and the physics community has to make a trade-off.

CERN LIBRARIES, GENEVA

CM-P00059909

To be presented at 'Computing in High Energy Physics'
Oxford, 10-14 April, 1989

1. HEP applications and their requirements

The discipline of Computer Graphics covers a wide area. Thus, of necessity, what follows cannot do more than skim the surface, with emphasis on a personal selection of topics. Perhaps the first point to note is how the leading edge of computer graphics has moved, over the last ten years, relative to the requirements of graphics within High Energy Physics (HEP). These now look rather modest. For example, the graphics applications listed in the CERN Report on Computing in the 1990s [1] are as follows:

Detector design:	CAD and physics simulation in order to optimize detector performance and save beam test time.
Software development:	Detector representation, interactive display of reconstructed events, debugging pattern recognition programs, etc.
Experiment analysis:	Event and detector viewing, physics parameter representation, multi-dimensional statistical analysis, histogramming.
Publications:	use of graphics displays with a graphics editor to produce figures for output onto high-quality hardcopy devices.
Control applications:	Human interface for control systems.

One might also add the use of graphics for software engineering applications, such as the display of SA/SD diagrams, and so forth.

If one does not include commercial CAE packages, then currently these applications require the ability to draw up to several thousand coloured vectors or markers (dots or crosses) on a display, and maybe some tens or hundreds of filled areas (surfaces). Typically, a response time under a second is adequate. However, in many cases it is the calculation time required by the application itself which limits the performance. In the case of 3D applications used for event analysis, one might wish to display some tens of thousands of vectors and perhaps some tens of filled areas to indicate energy deposition in calorimeters. It becomes self-defeating to put too much detail on the screen. If such a 3D image is to be rotated or scaled smoothly in real-time, the display system needs to be able to re-draw this number of vectors and filled areas about fifteen to twenty times/second.

These numbers are now modest compared with the requirements of what is becoming the major use for high performance display systems; namely mechanical CAE applications. These typically might require the display of half a million vectors and/or several tens of thousands of filled areas per second. Moreover, to provide realistic images, CAE applications require the use of smooth shading techniques using many colour planes and the individual calculation of the colour of each pixel. In addition, CAE image structures may be very complicated and some parts of the image, such as robot arms, may be required to move relative to others. This is not usually the case for HEP detectors, where one goes to some trouble to ensure that the pieces of the detector remain absolutely stationary.

An area where graphics applications are advancing concerns the use of animation techniques to aid the understanding of physical phenomena. A typical example would be to solve the Navier-Stokes equations at a series of time steps for a hydro-dynamic problem, such as the flow of air over an air-foil. However, in this case the problems typically need to be run for many hours on a super-computer, with the images simply being stored as pixel-maps on a laser disk and then played back onto a display which does not have to be very sophisticated. The results are impressive (especially when presented at a seminar), but it is not evident how such techniques could be used within HEP where each event is statistically independent. Nevertheless, there may be now a heightened awareness that improved visualization techniques could aid experiment analysis, and one example is reported in [2]. Thus, the relatively modest HEP requirements for graphics systems may be taken as an advantage, or perhaps as a challenge to the physics community to think up more sophisticated applications. The problems of HEP graphics are of a rather different nature, and will be covered in the last section.

2. Graphics Hardware

In the case of 2D applications, there exists already a large range of graphics terminals with adequate performance; bottlenecks are found rather in the area of communication with the host computer. RS232 terminal lines are not sufficiently fast for good quality interaction and, although the bandwidth of a Local Area Network (LAN) is large enough, it remains to be seen how the LANs and their communications protocols will manage the traffic if it consists of a lot of small packets, for example, when tracking the cursor in sample mode. The use of workstations solves this problem, or at least moves it somewhere else. It is no longer the graphics applications which place the load on the network, but the need to distribute the data samples to each physicist's workstation.

As for terminals, there exists already on the market a range of workstations with sufficient performance for most applications. Of course, the most powerful machines may not be cheap, but time is on the side of the user as the ever-increasing density of VLSI components continues to provide an improved price/performance ratio. For 3D applications the situation is slightly different. Although 3D hardware is also falling in price, the mass-market is increasingly for workstations capable of drawing smooth-shaded surfaces with Hidden Line Hidden Surface Removal (HLHSR) in order to represent mechanical components realistically. The sort of wire-frame (vector) machines useful for drawing HEP events are of less interest. Thus, although it is possible to purchase hardware capable of drawing vectors it tends to include features which one might be able to manage without, and so be more expensive than otherwise would be necessary. Note that hardware transformation support cannot be used if non-linear transformations are applied to aid pattern recognition.

For both 2D and 3D applications there is a strong trend to the use of colour: although graphics applications may be run on monochrome workstations purchased for program development, increasingly, colour stations are purchased if their primary use is for graphics. However, unlike CAE programs which generate realistic images, many HEP applications require only a small number of colours. This is because the eye cannot distinguish more than about six or eight different colours if they are not physically adjacent, and the primary use of colour in HEP applications has been to differentiate physical properties, such as the energy or momentum of particle tracks which may be anywhere on the screen. It remains to be seen if the use of a large number of colours can aid data presentation significantly.

Resolution (roughly, the number of dots/inch on the screen), is a feature which has not improved so fast as memory size or processor speed. The problem is twofold. To double the linear resolution quadruples the amount of memory of the frame store in which the image is held, and hence its price. A more serious difficulty is to produce screens with sufficiently fine colour masks, and which will accept the extremely high bandwidths required. For example, a display with 1280x1024 pixels, 24 colour planes, and a refresh rate of 60Hz requires a total bandwidth approaching 2GHz. For comparison, a 'back-of-an-envelope' calculation suggests that a high-quality 35mm photograph might contain up to about 40 million pixels, so graphics displays still have some way to go. This is an area where substantial improvements will be required if one wishes to pre-view printed pages containing small well-formed characters on a display screen, and which also affects the ability to untangle complicated events.

The trend for hard-copy systems, as for displays, is towards colour, although colour plotters still tend to have relatively poor resolution and speed, whereas cheap monochrome laser printers are available which provide results of excellent quality. Another problem with output printed in colour is the inability to make photo-copies for distribution. It is anticipated that this situation will change substantially over the next few years with the introduction of affordable colour laser printers and photo-copiers.

3. Graphics Software and Standards

Until recently the software interface to graphics display systems has been provided by the manufacturers of the hardware. Thus, many programmers know and love packages from Hewlett-Packard, Calcomp, Tektronix, Versatec, and so on. This used to be inconvenient when one wished to move an application, but still supportable, as the number of routines involved was fairly small. However, as display systems became more powerful, requiring much larger numbers of interface routines in order to access the hardware capabilities, the effort required to move an application started to become prohibitive. This generated an interest in the possibility to create a uniform application interface, and in 1974 ACM SIGGRAPH set up the Graphics Standards Planning Committee (GSPC), which led to the publication of the well-known CORE system [7] in 1979. However, CORE never became a standard.

3.1 GKS, The first Graphics Standard

In parallel with the work of the GSPC, a German study group had been developing a Graphical Kernel System (GKS) which was presented by the Deutsches Institut für Normung (DIN) to the International Standards Organization in 1978. Much of the work at this stage had been at the level of establishing basic concepts which could be used to provide a generalized interface, and the collaboration of many countries within ISO on the best way to combine ideas from CORE and the original GKS proposal finally produced the document defining the GKS ISO Standard [8] in 1984. Perhaps the principal innovations were the following:

- In most cases earlier software interfaces could control only a single device. GKS simplified this situation considerably by defining *Logical Workstations*, which correspond to 'virtual devices'. Each logical workstation is mapped onto a real piece of hardware, but to the application programmer they are all handled in a similar way, and several workstations can be active simultaneously.
- There is a special 'workstation' which writes the output it receives onto a *Metafile* for storage and/or transmission over a network.
- GKS incorporates the concept of *Logical Devices* to control input. These are mapped onto the actual physical devices in use without the application programmer having to write special code if, for example, (s)he is using a mouse instead of a tablet.
- GKS provides a mechanism to group the graphics output commands into *Segments*, where they may be stored for later use. Thus, part of an image stored in a particular segment could be made invisible and then visible again, or transformed relative to the rest of the image, without the application having to re-draw it. This can provide fast interaction, especially for display systems which provide local memory for segment storage.
- GKS provides a whole host of functions allowing the application to *inquire* at run-time the state and/or capabilities of a workstation or the implementation of GKS itself. This is a very important feature for allowing code to be written in a portable way and still to make the best possible use of the environment in which it is running.

It is interesting to note that whilst CORE defined a set of abstract functions it did not define a way to access them via standard procedure calls. Thus, although many CORE implementations were produced they did not provide a unique application program interface. An innovation with GKS was that the standard defined a functional interface independent of a specific programming language, and then as a separate standardization effort *bindings* were defined to individual languages, such as FORTRAN [9], C, PASCAL and ADA. However, whilst CORE was a 3D system GKS provides a 2D interface, and the GKS-3D [11] extension to three dimensions was accepted as an ISO standard only in 1988.

3.2 PHIGS and PHIGS PLUS

In parallel to the production of the three dimensional GKS super-set another ISO group was working on a standard called the Programmer's Hierarchical Interactive Graphics System, PHIGS [12]. A great deal of PHIGS is identical to GKS-3D, including the primitives, attributes, workstation concept, and the viewing and input models. However, PHIGS has a single *Central Structure Store* (CSS), unlike the separate Workstation Dependent and Workstation Independent Segment Storage of GKS. The CSS contains *Structures* which can be configured into a hierarchical directed-graph database, and within the structures themselves are stored the graphics primitives, attributes, and so forth. Also, unlike GKS, the structures within the CSS may be edited. An overview of PHIGS is given in [3].

Although PHIGS was only accepted as a standard at the end of 1988, there is already a proposal for an extension, called PHIGS PLUS. This includes additional primitives and the possibility to define smoothly shaded surfaces. Thus, it is much more applicable to mechanical CAE applications than either PHIGS or GKS-3D, and it is unlikely that there will be many PHIGS implementations produced which do not incorporate the PHIGS PLUS extensions.

Clearly an overlap region of requirements exists where applications could use either GKS-3D or PHIGS [6]. However, PHIGS is aimed particularly at highly interactive applications with complicated hierarchical data structures, and was designed with the assumption that its users would be experienced computer graphics professionals. GKS-3D places lower requirements on both the display system and the expertise of the programmer. It has also the considerable advantage that systems can be built which incorporate modules of existing two dimensional GKS code without change. The overlap between GKS-3D and PHIGS PLUS is smaller, and PHIGS PLUS may represent a substantial overhead on applications for which the functionality of GKS-3D is sufficient.

3.3 Graphics Data Storage

GKS allows the instructions used to build an image on a logical workstation to be captured on a file, called a GKS *graphics metafile*, which can be used for picture storage, transmission and replay. However, the metafile definition is included only as an Appendix, rather than an integral part of the GKS standard, and the information recorded can only be re-interpreted by GKS. A separate standardization effort has produced the Computer Graphics Metafile (CGM) [13], which specifies how to record a series of static 2D pictures on a file in a way which is independent of the graphics system. Thus, CGM allows the transmission of pictures between different graphics standards, and also with applications in other areas. For example, CGM is being used by text processing systems in order to incorporate pictures within documents. Three addenda to the CGM standard currently are being discussed to provide support for picture segmentation, 3D primitives, and facilities used by such applications as Engineering Drawing, Graphics Arts, and Technical Publishing.

3.4 Window Systems

The term *Window System* may be defined as a mechanism to support multiple virtual displays on a single workstation screen. Typical implementations use bit-mapped displays with rectangular arrays of pixels, or *windows*, representing the virtual displays. The size and position of the windows can be changed, the windows may overlap each other, and often they may be used for interactive graphics as well as for the emulation of simple alpha-numeric terminals. A 'mouse' or tablet usually is provided to move a cursor on the screen, and support software is often available to provide 'pop-up' menus and other features to improve the user interface. Unfortunately, every workstation vendor markets a system with different application program and user interfaces, and only now is the standardization of window systems being taken up by ISO. The leading contender for standardization clearly is the system from MIT, called *X-Window* [15], which has also been adopted by the Open Software Foundation.

Apart from the ability to manipulate windows, an important feature of X-Window is that it defines a communications protocol rather than a functional interface. Thus, an application using the X-Window system is not obliged to reside on the same machine as that on which the interaction is performed, but may be run on a *client* processor, such as a super-computer or database machine, whilst the user interacts with a local workstation acting as an X-Window *server*. Moreover, the client applications connected to the different windows on the screen of an X-Window server do not all have to run on the same host. These features of X-Window are very powerful, but may well generate a substantial overhead in both computing cycles as well as LAN traffic. Another problem with X-Window is that its support for graphics is very rudimentary, but this should be addressed by an extension, called PEX [16], which provides the extra hooks necessary to run GKS and PHIGS PLUS via the X-Window protocol.

3.5 Standards in the Future.

Although computer graphics standards are far from being a panacea, the situation now is very different from the Babel of the recent past. Perhaps this is best indicated by the current efforts being made by the industry to support existing standards, and the fact that outside of 'niche' markets manufacturers are becoming reluctant to attempt the introduction of yet another proprietary interface. As an example, several major companies recommend that only GKS, GKS-3D and PHIGS be used to access the graphics functionality of their workstations. However, this is not to overlook the problems. The number of users and the variety of their applications continues to increase with the capabilities of the hardware. In addition there are requirements for graphics standards to be well integrated with window systems, not to mention with text, image processing, and so on. Trying to provide a standard interface in this environment is really like attempting to shoot at a moving target, and since the completion of the 'first generation' of standards, there has been much discussion within the standards community as how best to proceed [4].

In all probability there will now be a pause. Work will proceed to complete the language bindings to existing functional standards, to standardize PHIGS PLUS as a super-set of PHIGS, and to have a maintenance review of GKS in order to remove some implementation dependencies and to tidy up some other loose ends. This maintenance review may even be able to include some features to help GKS operate in a windowing environment. However, the goal will be to produce the review quickly, rather than to make major changes or additions. In general, there is a feeling that both manufacturers and users need a period of stability in which to capitalize on the gains achieved so far.

In parallel, work will start on a second generation of standards with reduced requirements of backwards compatability for introduction in the mid-nineties. This second generation must find a way to take into account an ever widening spectrum of diverse applications, and the current view is that this might be achievable by designing a number of standards, each aimed at a particular area, on the basis of a set of 'components' [4]. In addition, these new standards must find a mechanism to incorporate a large range of graphics primitives, as well as to provide a more powerful input model, as the logical input devices of GKS and PHIGS do not provide the programmer with sufficient control over feedback to the user.

4. Portability, Performance and Cost

When I was asked to give this paper, a suggested title was:

'HEP Graphics: When will we get the performance we need?'

Presumably this was supposed to be provocative, with the implication that somehow performance was being withheld. However, I would suggest that this reveals a misconception. Certainly there exists 2D and 3D equipment on the market with enough performance to provide an acceptable response time for the majority of HEP graphics applications. The question to ask is rather *'How much is the HEP community prepared to pay?'*

The problems of HEP graphics are organizational rather than technical:

- Many experiments now involve hundreds of physicists working in different laboratories world-wide and using diverse equipment.
- Physicists working in a single institute on different experiments often wish to share the same graphics hardware and software interfaces.
- Some applications are 2D and others 3D. Of the 3D applications, some require real-time transformations, and some do not. (Although, in the first case, real-time transformations may not be essential during program development.)
- 3D applications usually wish to incorporate existing modules of 2D code.
- It is economically and politically impossible to ensure that all users purchase the same hardware.
- It is probable that several generations of displays will be used during the lifetime of an experiment.

All these requirements point to the need for program portability, and this might be achieved either by programming to a standard interface or by having a mechanism to maintain multiple program versions. Even with the use of a code-management system it is very unlikely that the second option would be viable. Hence the possibilities are reduced to mandating a common interface, but which one, and what does it cost? Clearly one cannot choose a proprietary interface, as this would not operate on the required range of equipment. Moreover, if one wished to define a new interface there is another set of problems. If the interface were to be set at a high level, then there would be too much work to carry out in order to adapt it to all the different proprietary hardware drivers. However, if it were to be set at a low level it would not provide access to the features found in modern display systems. Such a proposal would have also much in common with an attempt to re-invent the wheel. Thus, one is led inexorably to one of the ISO standards.

For 2D applications GKS is the only possibility. For 3D there is now a choice between GKS-3D and PHIGS. However, the hierarchical data structures and database editing facilities of PHIGS are not essential for HEP applications, and they must be offset by the disadvantage that the use of PHIGS for 3D would also make it much more difficult to incorporate modules of 2D GKS code. In addition, on displays without all the necessary hardware support, PHIGS is likely to run more slowly than GKS-3D.

The use of GKS and GKS-3D as a standard interface works. There are already example of large 3D programs being moved from one manufacturer's equipment to another in the space of a few hours [5]. A job which could well have taken between one and six months if proprietary interfaces had been used. Of course there is a price to pay. First of all, if one sticks to the rules, then a standard interface may sometimes hide special features of a particular display system. Moreover, as GKS and PHIGS are designed to provide simultaneous access to an inhomogeneous set of displays, the code path to the display is longer than that for a specialized package, and hence slower; perhaps by up to a factor of two or

more depending on how well a particular implementation has been written. Certainly, it is possible to make gains by optimizing device driver code, and user education is important as misuse of any system can give poor results. (When making comparisons it is essential to perform comparable operations and to use the packages in the way that was intended!)

Thus, I have been led to formulate Myers' Law for graphics, based on Boyle's Law for gases, and generating a similar amount of heat:

$$\frac{P \cdot V}{S} = f T \quad (1)$$

where:

P	is Performance
V	is Variation of supported hardware
T	is Tax (the cost)
S	is the Standardization level
f	is a constant of proportionality called the <i>frustration factor</i>

To mix another metaphor, the law says that without paying a price HEP can not have its performance *and* port it. Graphics users can get the maximum possible performance from the hardware either by having all members of a collaboration buy the same equipment, or else by expending vast amounts of effort to adapt the code to different systems, *and then maintaining all these different versions in operation over the lifetime of the experiment*. It is unlikely that the first condition could be met even with unlimited funds, and those working in the discipline may make their own judgement as to the availability of the manpower required in the second case. The only way to reduce the cost is to increase the level of standardization. This relationship is true for any HEP program, but is particularly important for graphics code which must run not only under many different operating systems but also must drive many different display and hard-copy devices. (At CERN alone there are more than 20 different graphics devices being driven via five operating systems, and there are regular requests to increase these numbers .)

If it essential to extract the last ounce of performance from a system then it is possible to write a one-off package consisting of a sub-set of the standard calling sequences interfaced as efficiently as possible to the device in question. Such a procedure has drawbacks, for example it may not be able to produce an output metafile, but at least it does not require major changes to the application code. However, as faster processors continue to become available then even this sort of exercise becomes less cost effective. To conclude, this paper does not try to prove that there is a unique solution. Several possibilities exist depending on what one is trying to optimize, and each experiment must decide for itself which trade-offs to make. However, be aware of the cost when making the decisions; ultimately equation (1) will prevail.

5. Acknowledgments

Thanks are due to A. Grant, M. Metcalf, and D.O. Williams for reading draft versions of this paper and making helpful comments. However, no blame should be attached to them for the views expressed herein.

References

1. *Computing at CERN in the 1990s*, To be published
2. *Display of track coordinates and tracks and their extrapolation to the calorimeters by the graphics program DALI*, H. Drevermann, CERN/ALEPH Note 89-53/Software 89-5, March 1989.
3. D.R. Myers, *Interactive Computer Graphics and PHIGS*, Computer Physics Communications, **50**, (1988) 143-157.
4. D.R. Myers, *Report on the status of Graphics Standards*, CERN/DD 88/14, September, 1988.
5. A. Grant, F. Etienne, private communication.
6. J. Bettels, P. Bono, E. McGinnis, and J. Rix, *Guidelines for Determining When to Use GKS and When to Use PHIGS*, Computer Graphics Forum, **7**, 4, (December 1988) 347-354.
7. *The CORE Graphics System*, SIGGRAPH-ACM Computer Graphics, **13**, 3, August 1979.
8. *The Graphical Kernel System (GKS)*, International Standard ISO 7942, Geneva, July 1985.
9. *The Graphical Kernel System (GKS) language bindings – Part 1: FORTRAN*, Draft International Standard ISO 8651-1, Geneva, July 1986.
10. *The Graphical Kernel System for Three Dimensions (GKS-3D)*, International Standard ISO/IEC 8805, Geneva, June 1988.
11. *The Graphical Kernel System for Three Dimensions (GKS-3D) language bindings – Part 1: FORTRAN*, Draft International Standard ISO/IEC DIS 8806-1, Geneva, November 1988.
12. *The Programmer's Hierarchical Interactive Graphics System*, Draft International Standard ISO/IEC DIS 9592, Geneva, May 1988.
13. *Computer Graphics Metafile for the storage and transfer of picture description information*, International Standard ISO 8632, Geneva, February, 1986.
14. *PHIGS PLUS Functional Description Rev. 3.0*, ISO JTC1/SC24/WG2 N3, Geneva, March 1988.
15. J. Getties, R. Newman, R.W. Scheifler, *Xlib – C Language Interface*, Protocol Version 11, MIT Project Athena, September, 1987.
16. *PEX Protocol Specification*, ISO JTC1/SC24/WG2 N2, Geneva, December 1987.