ARCHIVIO

Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

PISA

THE EVOLUTION OF GRAPHICS STANDARD: FROM CORE TO PHIGS THROUGH GKD-RD

B. Biagi, C. Montani, R. Scopigno

Nota interna B85-09

Agosto 1985

B85-09

THE EVOLUTION OF GRAPHICS STANDARD: FROM CORE TO PHIGS THROUGH GKD-3D

Benedetto BIAGI^, Claudio MONTANI^ and Roberto SCOPIGNO^^

^ Istituto di Elaborazione dell'Informazione, Consiglio Nazionale delle Ricerche, Via Santa Maria 46, 56100 PISA (Italy).

^^ Cascade Graphics Development S.p.A., Via Monte Carmelo, 00166 ROMA (Italy).

ABSTRACT

The paper presents a comparative analysis of three graphic languages (CORE, GKS-3D and PHIGS) which can be considered as representing the main stages in the process of standardizing a general purpose graphic language.

The way in which the evolution of graphic languages has favourd a greater flexibilty for the user at the price of a heavier load for the system is underlined.

The different conceptual bankgrounds of the three languages are briefly discussed with respect to their embedding in graphic data bases for sophisticated applications.

1. INTRODUCTION

In recent, there has been much interest in the definition of a general purpose graphic language which could become an international standard. The large amount of research activity in this period [1] bears witness to this fact.

This paper presents the conclusions of a long study and of much discussion on the evolution of proposals for general purpose graphic languages during the last years and on the impact that the new tendencies could have on computer graphics in the near future.

Our work begins with an analysis of the functionalities and the basic concepts of the three graphic languages that, in our opinion, can be considered as representing the main stages in the standardization process:

(a) CORE [2]: this language was formalized between 1975 and 1979 by the Graphics Standards Planning Committee (GSPC) of the Special Interest Group on Computer Graphics of ACM (ACM-SIGGRAPH). CORE represents the first real effort towards the standardization of a graphic language;

(b) GKS-3D [3]: This is the natural extension to the three-dimensional functionalities of the only graphic language that today is an International Standard (the Graphical Kernel System). At the present, this language is being studied by the ISO TC97/SC21/WG5-2 Committee and it is expected that it will become a standard before the end of 1986;

(c) PHIGS [4]: At the present, this language is at a level of Draft Proposal under investigation by ANSI X3H3 Committee. PHIGS was born as a GKS extension even if, in many aspects, it is not compatible with the GKS language.

Even if there is no strict chronological order (the first formalization of PHIGS was infact previous to GKS-3D), as far as the functionalities and the basic concepts are concerned a well defined evolution in terms of system flexibility can be noted, passing from CORE to PHIGS through GKS-3D.

In the next section, the different characteristics of the three languages are analyzed. So that our study would be on a homogeneous set, languages which offer full bi-dimensional and three-dimensional functionalities have been chosen.

The last section presents some considerations on the possible future consequences of the present rapid transformations in computer graphics.

2. A COMPARATIVE ANALYSIS OF CORE, GKS-3D AND PHIGS

In this section, the main characteristics of CORE, GKS-3D and PHIGS are compared. Our analysis refers to the main aspects of the languages: the workstation concept, segments or structures, output primitives and their attributes, geometric transformations and input.

2.1 WORKSTATION

A dominant feature in the evolution of graphic languages is represented by increasingly distinct separation between device-dependent and device-independent parts of the graphic system.

This distinction, which is made more urgent by the need for application software portability and by the strong differences between existing graphic hardware, had already been felt by the authors of CORE. Indeed, CORE presents the concept of "multiple view surfaces" to which the static attributes of the output primitives are bound. The display of the output primitives depends on the values of the attributes. In fact, the attributes partially mask the physical characteristics of the output devices.

On the contrary in GKS-3D and PHIGS the workstation concept formalizes the distinction between device-dependence and device-independence. Each user-activated workstation is not limited to playing the role of the output interface, but it represents the aggregation of a logical output device and one or more logical input devices. In addition to many attributes, a number of geometric transformations of the output primitives are dependent on the single workstations.

The user can dynamically modify the workstation characteristics using commands to modify its state.

2.2 SEGMENTS

All three languages present the concept of graphic segment understood as a set of logically correlated graphic elements (the segment wheel in the design of a car, the segment desk in an office furnishing application). The segment names are user-defined.

The type and nature of the information which the user can store in the segments and the transformations which can be applied to the graphic elements before and after storage differ widely from language to language.

CORE adopts the synthetic camera model to represent three-dimensional scenes. Thus, the CORE segments store a snapshot of each scene. All of the output primitives are passed through the viewing pipeline. At segment creation time, the current values of the attributes are bound to the output primitives and this defines their display features.

The CORE user can dynamically "move" (by rotation, scaling and translation transformations) its snapshot in a three-dimensional space. The system will then orthographically project the results onto the viewing surface.

In GKS-3D, a simple normalization transformation is applied to the output primitives at the creation time. The output primitives of a segment go through the viewing pipeline at traversal time; i.e. when the segment is executed.

As in CORE, the current values of the attributes are bound to the output primitives during the creation of the segment. Moreover, a special mechanism allows the user to have a type of dynamic attribute selection.

The PHIGS segment stores graphic primitives which have not undergone any geometric transformations. The current values of the attributes are not bound to the primitives at creation time, but each segment stores the attributes selection commands entered by user. These commands will be executed at traversal time and their execution will modify the current values of the attributes.

Unlike CORE and GKS-3D which only allow a segment to be copied into another one or to be deleted, PHIGS provides the user with a set of editing functions. Using these functions, the user can delete, insert or replace graphic commands into an already defined segment.

A further and more evoluted feature of PHIGS in comparison to the other languages is represented by the hierarchical structures of the graphic objects. In this language, segments are known as structures and a hierarchical structure is obtained by referring from within a segment to other segments using EXECUTE commands. From an operational point of view, the system state determined by a "father" segment (attribute selection, geometric transformation of the viewing pipeline, etc.) is inherited by the "son" segment. This son segment can modify the state for itself and for its ownsons, but the system state will be returned unchanged to the father.

2.3 OUTPUT PRIMITIVES

Operationally the evolution of graphic languages has not implied substantial differences in the output primitives. However, two particular aspects must be underlined.

Unlike GKS-3D and PHIGS, the CORE language contains the concept of "current position". A direct consequence of this feature is the presence of a set of "relative" output primitives together those with absolute coordinates. It should also be noted that CORE has "low level" output primitives (MOVE and LINE). These instructions are no longer present in the other languages.

The second point to be stressed refers to the raster type output primitives. Whereas CORE only provides for raster primitives at an extension level, GKS-3D and PHIGS formalize the concept of raster primitives by defining their display features on raster and vectorial output devices.

2.4 ATTRIBUTES OF THE OUTPUT PRIMITIVES

With reference to the attributes of the output primitives, it can be noted that the differences between the languages do not depend so much on the number of different attributes at the user's disposal (however, in this sense, GKS-3D and PHIGS are richer than CORE), as on the way the values of the attributes are selected and the moment when they are bound to the primitives.

CORE has only one way in which to choose the value of the attributes. This is specified either before or during the creation of a segment and is bound to the primitive(s) to which it refers to when that primitive(s) is stored in the segment. This selection mode is known as "INDIVIDUAL" selection.

In GKS-3D and PHIGS the formalization of the concept of workstation has lead to the definition of a further selection mode: the "BUNDLE" selection. Using the "BUNDLE" selection the user does not select the value for each attribute, but chooses an index to a table, which he can also modify. There is one bundle table for each active workstation and it defines the values of the set of the attributes of each output primitive. In this way, the same primitive can be displayed with different features on different workstations. It is important to stress that one or more attributes of a primitive can be selected in an individual mode even if a bundle selection for the attributes of that primitive has already been specified.

Similarly to CORE, GKS-3D binds the current values of the attributes to the output primitives, at segment creation time. However, a kind of dynamic selection is achieved through the bundle selection. The bundle table, logically resident on the workstation, can be modified by the user at any time, even after the creation of the segment.

On the contrary, in PHIGS the attribute selection commands (individual or bundle selection) are stored in the segments and executed at traversal time. This is a fixed choice in a hierarchical environment where the context of the father (hence also the current values of the attributes) must be inherited by the sons.

A further difference between PHIGS and the other two languages refers to the application of certain attributes such as visibility or highlighting. In PHIGS these attributes are referred to groups of output primitives, in CORE and GKS-3D they are referred to a segment.

2.5 GEOMETRIC TRANSFORMATIONS

The languages studied provide a complete viewing pipeline for three-dimensional scenes. Geometric transformations can be grouped into three categories: modelling transformations place the objects of the final scene in a single cartesian reference system; viewing transformations create a view of the scene to be represented; workstation transformations perform the mapping from a device-independent space to a device dependent one.

Whereas it is not possible to demonstrate substantial differences among the functionalities of the geometric transformations offered by the languages, there are considerable differences in the times at which the transformations are applied to the output primitives.

Table 1 summarizes the geometric transformations of each language, the entities to which they are applied and their application time. The table does not present the clipping transformations. In fact, each language has clipping capabilities.

2.6 INPUT

CORE, GKS-3D and PHIGS all have logical input devices. The application program can acquire external information through the logical input devices. These logical devices make the way in which the information by means of physical devices is acquired, completely transparent to the user.

The logical input devices are characterized by the "class" (the type of response) and the "mode" (request, event and sample).

Whereas the languages present similar classes, an evolution can be noted with regard to the operational mode. In CORE the mode is a characteristic of the class, i.e. each class of logical devices can operate in a fixed mode. In GKS-3D and PHIGS, the mode is independent by the class. The user can define the mode, dynamically.

In PHIGS, the functionalities of the PICK input class are both flexible and powerful. This is due to the need to "pick" the correct occurrence of a segment belonging to a hierarchical structure.

3. ASSESSMENTS

After this comparative analysis of the basic features of the three languages, we think that the main differences between them depend on the increasing flexibility which we find moving from CORE to GKS-3D and to PHIGS rather than on the possibilities that they offer.

Some operations which are quite heavy for a CORE user become more elementary in GKS_3D and far easier in PHIGS.

Let us suppose, for example, that the user wants the viewpoint in a 3-D scene to be modified and the color of some objects be changed.

The CORE user is obliged to delete all the segments of the scene and then to create new segments so that all of the primitives pass through the complete view pipeline. Moreover, new attributes must be assigned to the primitives with a change in color.

The GKS-3D user only needs to rewrite those segments which belong to objects with changed color attributes (we assumed INDIVIDUAL attribute selection) and to replace, at the workstation level, the current view transformation with the required one.

Finally, the PHIGS user can simply edit those objects that contain the attribute selection command to be modified and select a new view transformation at the workstation level.

Our opinion is that user-acquired flexibility has caused a progressive shift of the work load from segments creation time to traversal time.

This movement clearly increases the global system work load, even though this is transparent for the user, but at the same time the greater flexibility means that the user can have real time or animated applications.

Let us consider, for example, an environment in which only one object changes dynamically while the others remain unchanged in time. It could be presumed that the static objects also have to be re-executed. This would be due to the double buffer operating mode of the display device which means that complete rewriting of the frame buffer is necessary at each swapping. Otherwise it could be due to the dynamically changing object priority which is less than that of the static objects. In this two cases we would have a situation in which a simple change of an object by a PHIGS user (and in some cases also by a GKS-3D) user causes a complete retraversal of all the objects and consequently would be more costly than the rewrite of a CORE segment and also of a complete retraversal, which does not require application of the viewing pipeline for the static objects.

It is obvious that the evolution from CORE to PHIGS was not caused by the need for increasing flexibility for the user but rather by the rapid development in graphic hardware.

Ignoring the most sophisticated devices, which use firmware shading or anti-aliasing features, some graphic cards are now commercially available for personal computers and give two- and three-dimensional performances.

We thus feel that the standard graphic language proposals have evolved in response to these increased possibilities.

Neverthless, for the present, we think that the path of an extremely flexible language can not be completely smooth.

A wide use of a very flexible standard graphic language could have two immediate consequences on the graphic hardware:

a) if the hardware has high local intelligence, it will have to be programmed in accordance with the standard to be supported because, otherwise, the graphic language-hardware interface would be very heavy. In some cases, certain functions will have to be implemented in software without using the advanced hardware features in order to ensure that they are standard;

b) if the hardware has low local intelligence, an interface between the language and the system hardware must be provided so that the frequently performed traversals and the execution times are as fast as possible. One possible solution could be to pair the high level data structures in main memory (similar to those of PHIGS) with lower level structures (CORE-like, for example) obtained by a complete traversal of the high level structures. Changes can be made on these low level structures, which do not involve a complete regeneration. Probably, the execution times would be more reasonable. However, unfortunately, the same thing can not be said about memory costs.

In addition to these observations on the graphic hardware, it is worthwhile also mentioning the evolution of the graphic languages with respect to graphical data bases.

In fact, most applications require links which may be more or less strong with data bases of graphical objects.

Because of the evolution in the languages, it is now possible to work at a much higher level than that allowed by the simple archiving of static objects. PHIGS, for example, provides powerful editing functions. Many of the commonest operations performed by a DBMS on data structures are easily translated in terms of PHIGS functions.

The management and manipulation of the objects and their composition in complex images such those required by a CAD application thus becomes much simpler.

Furthermore, from an implementation viewpoint, the conceptual approach of PHIGS is very close to that used by a DBMS in the management of data structures, of attributes and of the relationships among objects.

We could conclude by saying that the present trend towards PHIGS-like languages is positive with respect to applications but leaves some uncertainties as far as the host hardware is concerned.

TABLE 1

(a) Geometric transformations in CORE

Modelling Tr.	World Coord. Tr.	one or more primitives	creation
Viewing Tr.	Viewing Tr.	segment	creation
	Projection Tr.	segment	creation
Workstation Tr.	Image Tr.	segment	traversal

(b) Geometric transformations in GKS-3D

Modelling Tr.	Normalization Tr.	one or more primitives	creation
	Segment Tr.	segment	traversal
Viewing Tr.	Viewing Tr.	one or more primitives	traversal
	Projection Tr.	one or more primitives	traversal
Workstation Tr.	Workstation Tr.	segment	traversal

(c) Geometric transformations in PHIGS

Modelling Tr.	Global Model. Tr.	one or more primitives	traversal
	Local Model. Tr.	one or more primitives	traversal
Viewing Tr.	Viewing Tr.	one or more primitives	traversal
	Projection Tr.	one or more primitives	traversal
Workstation Tr.	Workstation Tr.	structure	traversal

REFERENCES

[1] G. Enderle, K. Kansy and G. Pfaff, "Computer Graphics Programming: GKS, the Graphics Standard", Springer Verlag, New York, 1984.

[2] GSPC, "Status Report of the Graphic Standards Planning Committee", _Computer Graphics_, 13, 2, 1979.

[3] ANSC X3H3, "Graphical Kernel System", Special issue of _Computer Graphics_, February 1984.

[4] ANSC X3H3, "PHIGS Functional Description", 84/40, 1984.