stichting
mathematisch
centrum

MC

AFDELING INFORMATICA
(DEPARTMENT OF COMPUTER SCIENCE)

IW 210/82 NOVEMBER

P.J.W. TEN HAGEN

THE REVIEW OF GKS VERSION 7.0, THE FINISHING TOUCH

Preprint

kruislaan 413 1098 SJ amsterdam

Printed at the Mathematical Centre, 413 Kruislaan, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

CR Categories and Subject Descriptors: I.3.4 [Computer Graphics]: Graphics Utilities — graphics packages; I.3.6 [Computer Graphics]: Methodology and Techniques — device independence; interaction techniques

The Review of GKS version 7.0,

The Finishing Touch.

by

P.J.W. ten Hagen

ABSTRACT

The version of GKS that is to become widely available this month (version 7.2), is the result of yet another intensive review. However, this review has definitely been the final one. As a result, the ISO committee has now unanimously accepted GKS. Moreover, all national bodies actively participating in the development of GKS have decided to adopt GKS as a national standard. This means, for instance, that GKS will become a DIN-standard, an ANSI-standard, a BSI-standard, etc..

The functionality of GKS will strongly influence new graphics hardware and software design. Over the last two years a number of new functions have been introduced into GKS. The nature of the standardisation process is such that little of this innovation becomes apparent until it has been accepted by the committee. The new functions and the underlying concepts will be presented in this paper.

Keywords: Computer Graphics, Device Independence, Basic Software

General Terms: Design, Standardization

This paper is not for review, it will be published in Computer Graphics Forum Vol. 1, No. 4, 1982.

1. HISTORICAL NOTES

In the presentation given at EUROGRAPHICS 81 [1], a number of new concepts were mentioned. In doing so, it was anticipated that the meeting at Abingdon in October 1981 of ISO/TC97/SC/5/WG2 (the working group developing GKS), would accept these concepts and the associated functionality in their final form. Many of these concepts were only introduced at the Melbourne meeting of the same committee earlier that year.

For an overview of the work on GKS until 1981 the reader is referred to [2] and [3].

The ISO letter ballot on GKS version 7.0 (see [4]) made many people, especially in the USA, realize what the impact of GKS might be. The facilities for achieving device independence are so effective that hardware manufacturers see themselves forced into a new approach. They can no longer offer firmware that only runs efficiently with application programs that know about all the details of that firmware. Instead, they must be able to run efficiently under GKS. Moreover the new microprocessor technology has brought about a host of enhanced graphics terminals with all kinds of built-in functions many of which were formerly emulated in software. These functions make assumptions about the global state of the graphics system, or the terminal, which may not be valid in a device independent context. Although they may appear to correspond to GKS functions, the differences make the device functions difficult to use when implementing the GKS functions. To give some examples:

- Area fill firmware which fills a closed contour by specifying a colour and an interior point. The hardware cannot cope with self-intersecting polygons or fails to overwrite elements of the same colour.
- Hardware translation and scaling, without rotation. This would force a GKS implementation to treat all rotations separately.
- Global selection of attributes, e.g. colour. A GKS implementation can only invoke such a function at a very low level.

In addition, new, fast growing applications such as business graphics, need to be able to use GKS to drive raster devices. As a result, the GKS raster facilities could not be kept minimal whilst waiting for the raster graphics area to stabilise.

The final series of changes to GKS, and the most difficult to achieve concensus about, must be seen as an attempt to accomodate these requirements.

Not surprisingly, most of the last round criticisms were inspired by manufacturers and software houses with vested interests in alternative systems. Their criticisms illustrate the problems involved in transferring existing applications to a GKS base. The major issues were:

1. Treating all attributes as global and static as well as (in some cases) bundled and dynamic.
2. A character alignment facility.
3. A stroke input device.
4. A more general Generalized Drawing primitive.
5. Additional text attributes moved to the text bundle.
6. Changes to the minimal requirements per level.
7. Clarifications to annex C, the language conformance and binding guidelines.
8. Amplification of annex D, on implementation dependencies.

These issues were satisfactorily resolved at the most recent meeting last June at Steensel, The Netherlands. Some of these issues will be discussed below as part of the discussion of new functionality. The minimal requirements, language binding and implementation issues illustrate that there is a lack of experience with actual implementations for a variety of languages and operating systems. Fortunately the guidelines of annex C and D can be further improved and refined as they

are not part of the standard proper.

2. THE WORKSTATION

The workstation concept was already present in the earliest versions of GKS. It is of paramount importance to the whole GKS system. The majority of GKS functions directly address a workstation to control so-called workstation dependent aspects. Each global, workstation independent function is evaluated in two stages. The second stage is the realisation of the function on the individual active workstations.

The quality of a workstation depends on the **richness** of the aspect values the user can choose from, the **speed** with which the output, control and input functions are executed, and the **integration** of input and output actions. Although these quality aspects depend on the available hardware, many of them depend equally on the implementation. Together they determine the extent to which the implementor has succeeded in exploiting the hardware under device independent control.

The GKS functions can be divided into groups corresponding to the major capabilities the workstation provides.

2.1 Integration of input and output

Only one output device per workstation is permitted because the range of attribute values that can be realised depends mainly on the available output device. Two or more outputs would allow only the intersection of the various ranges to be used. This is true as much for output attributes as for input attributes.

In addition, it is preferable to have all input feedback on one and the same (output) device. This does not restrict the number of input devices per workstation as long as they can share the same screen for prompting and echoing.

The basis for the integration of input and output devices is provided by the workstation implementor, who realises the input primitive attributes, such as prompt/echo type, echo areas and input value ranges. PLP The behaviour of individual input devices can also be controlled by parameters provided via a data record. This is an extension mechanism, similar to the Generalised Drawing Primitive, and illustrates the fact that the input facilities provided by GKS may be expected to develop further. GKS already offers a number of input attributes settable by the application program. This advanced feature will attract the attention of users and implementors to a new area of interaction control.

Table 1 gives an overview of the input functions a workstation must support. The input attributes other than the echo on/off switch and mode are static attributes. They are specified in one function as a bundle, but cannot be dynamically modified. The echo on/off switch and the input mode (REQUEST, SAMPLE and EVENT) can be seen as dynamic attributes which can be changed at any time. One could imagine such a SET function selecting an input bundle index as well.

Figure 1 gives the input modes and transitions, plus the functions that are allowed in each state. To indicate the fact that REQUEST actually consists of two modes an additional mode called REQUESTED has been invented.

One might be tempted to conclude that the input facilities of workstations are anticipating future extensions rather than reflecting current practice. However, the arguments leading to these facilities all referred to state of the art input methods. The major difficulty was to develop a model placing the diversity of these methods in a common framework [5].

The next step to be taken in interaction is the integration of input and output at higher levels. This integration has two sides:

- The creation of pictures from output primitives, attributes and segments, as a direct response to input.
- The description of the visual behaviour of input devices in terms of output by predefined segments or resettings of dynamic attributes as prompt/echo types.

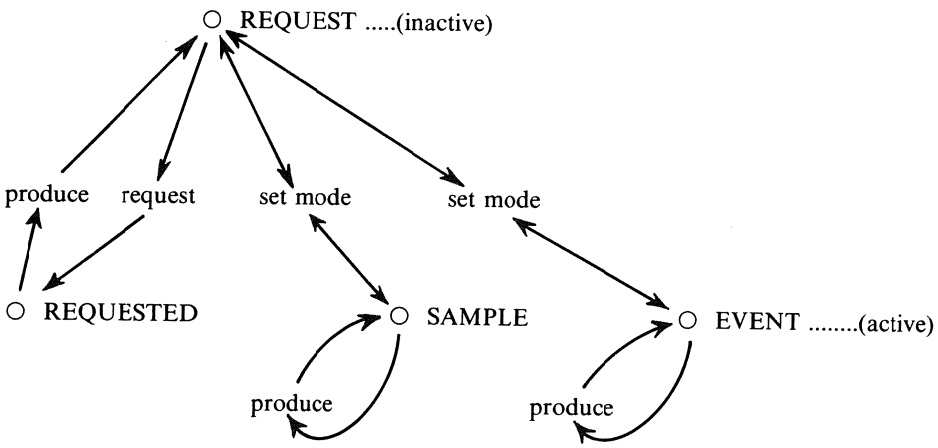

figure 1 _ input mode transitions

Table 1. INPUT FUNCTIONS	
INITIALISE XXX	← initial value for XXX ← p/e type ← echo area ← data record
SET XXX MODE	← S/R/E ← echo on/off
REQUEST XXX SAMPLE XXX GET XXX	→ value for XXX → value for XXX → value for XXX
AWAIT EVENT FLUSH XXX EVENTS	
IQU XXX STATE	

The current trend to develop separate dialogue systems on top of graphics kernel systems makes extensive use of such facilities.

The rapid development of intelligent graphics terminals with local interaction support creates a need for device independence at that level. The behaviour of these terminals will be in terms of picture changes on the GKS functional level.

2.2 Support for dynamic changes.

There are four basic ways to change a picture dynamically. The workstation implementor must provide support for each way. The workstation implementor by assigning resources to each way of change trades them off against each other.

The four ways are:

1. adding new output primitives to the display surface
2. manipulating segments
3. changing the dynamic attributes of picture primitives
4. creating new input causing feedback on the screen

For each of these manipulations the workstation may have to assign two kinds of resources: for updating the picture as fast as possible and for preparing future changes.

In the case of output primitives the transmission, followed by the conversion to display code are the two steps that matter. Output primitives containing thousands of vectors or large cell arrays for high resolution raster displays require fast transmission and conversion. For these the workstation may have to support elaborate encoding schemes and large buffer areas.

Minimising transmissions also is an argument in favor of clipping as early as possible. The decision to postpone clipping in GKS was taken because early clipping prohibits the use of clipping hardware as well as hindering device independence. An example of the latter is TEXT which can only be properly clipped when the workstation has made the selected FONT available. The application can avoid sending large amounts of data by pre-clipping (probably using application-specific indexes to speed the process).

Clipped primitives in segments can only be stored by GKS in workstation dependent segment storage. This would not solve the transmission problem if the device independent segment store is also being used, because then clipping can only take place at the workstation. In addition subsequently changing attributes may cause strange effects on that workstation, if they are not accompanied by retransmissions.

There are many interactive applications which cause relatively little change in the bulk of data on the screen. For these, it must be possible to configure a workstation so that fast output and conversion gets priority over manipulation. In the extreme case of storage type devices, changes other than additions must be buffered to avoid time consuming retransmissions.

The design of GKS has been such that these devices and applications can still be accomodated. However, not at the expense of device independence or the possibility of more dynamic use. One can tailor a workstation for a particular application field.

Picture change through segment manipulation can be implemented in two ways: using the device independent segment facility, or holding a segment at a workstation. In the latter case one may use the most efficient coding scheme for that workstation. The main* reason for letting a workstation have its own segments is fast feedback during manipulation. Possible situations where this applies are slow transmission to remote workstations or high quality interactive display hardware.

These first two means of picture change are not new. GKS permits workstations to be implemented which have different characteristics with respect to change even though the hardware is the same. Opening a different workstation type would set different characteristics.

* Space in the host may be another reason.

The workstation description table contains the initial value for the output deferral mode. This value will indicate the optimal mode. In figure 2 the values are shown in increasing order of strength. Choosing a higher value than the optimal one will usually not improve performance, but will certainly guarantee the implied effect.

Figure 2 — Deferral Modes	
ASTI	(At Some TIme)
BNIL	(Before Next Input Locally)
BNIG	(Before Next Input Globally)
ASAP	(As Soon As Possible)

Dynamic changes which require deletion or changes of appearance may require the complete picture to be regenerated. Although many hardware systems can immediately delete a segment they may well have trouble changing the colour of picture elements or the character font of some TEXT primitives. The latter are examples of dynamic attribute changes.

Fast implicit regeneration caused by changing attributes or immediate change of the displayed picture following an attribute reset, requires integration of segment and attribute bundle storage. Such integration will be able to rebind indexes to new attribute bundle values on the fly. In workstation dependent segment store, for instance, it is possible to implement a bundle index by a pointer to the bundle itself. Alternatively bundles might keep track of the segments that use them. To date no implementations exploiting these possibilities are known to me.

Workstation description tables can indicate the extent to which dynamic attributes can be given immediate effect. This is not the case for input attributes. It is more or less assumed that elaborate prompt/echo types need no deferral state or require regeneration. They must take effect As Soon As Possible (ASAP). Echoes which need to disappear, however, cause problems similar to segment deletion. There are no reasons to exclude echo types which dynamically change attributes of picture elements (e.g. turn green on PICK).

Further integration of input and output means that input and feedback data also must be integrated with the segment store. It seems that workstation implementors will find some challenging problems here.

2.3 Multiple workstations

The term multiple workstation suggests that it is a luxury for users who can afford high powered graphics equipment containing several output devices. Modern raster technology allowing multiple window terminals has opened the possibility to simulate multiple workstations on one screen. In each window on the screen a workstation with different characteristics may run. Several processes running in parallel may each use a subset of the active workstations.

This example suggests that multiple workstations will become a common feature of graphics systems running in multiprogramming environments. Implementors will aim at modest, efficiently driven workstations whose power will lie in combining them with other, similar workstations.

Multiple workstations of this kind must be subject to some kind of resource management. This means that the operating system environment assigns a workstation to a program rather than the application program taking it. True portability should allow for this and even stronger cases. A user should be able to start his application program from the graphics terminal he happens to login on.

A disadvantage of multiple workstations is the restricted attribute range they enforce upon each other. As we have seen above workstations sharing the same output device do not have that restriction.

3. RASTER FACILITIES

The raster facilities of GKS which have been kept modest, though not minimal, have not changed in this year's revision. The typical raster functions are:

FILL AREA
CELL ARRAY
SET FILL AREA INTERIOR STYLE
(PATTERN)
SET PATTERN REPRESENTATION
INQUIRE PIXEL
INQUIRE PIXEL ARRAY

The bridge between raster and non-raster output is the FILL AREA primitive. Using the bundle index for fill area representation, one can switch between say, HATCHED style for vector displays and PATTERN style for raster workstations, without further affecting the application program.

The FILL AREA boundary can be an arbitrary closed polygon. The PATTERN attribute and the CELL ARRAY primitive both initially are rectangular, but may be rotated by segment transformations. The PATTERN is repeated in all four directions until it completely overlaps the FILL AREA. In this way the interior of the area is determined by the extended pattern.

An intriguing question is how GKS would deal with the preference for rectangular shapes of many raster applications. For instance, rectangular FILL AREAs could be dealt with much faster by hardware, restricting segment transformations to translation and 90° rotations would fit bitmap displays using raster ops.

GKS provides no facilities for restricting its functionality to special cases, though they may be provided via GDP and ESCAPE. A rectangular FILL AREA makes a perfect GDP. Restricting the transformations would be more difficult. A possible solution might be an ESCAPE function which marks bitmap segments, but this would not be sufficient. Input to bitmap displays very often is echoed using rasterops on the displayed image. It will be more work to incorporate and represent these changes in the segment-like description of the screen.

Equally important for future developments is the fact that bitmap displays seem to be strictly two-dimensional. Current proposals for extending GKS to a three-dimensional system propose considering 2D as a special case of 3D. This makes no sense for strictly 2D workstations. Hence there may have to be 2D as well as 3D workstations.

4. ATTRIBUTES, BUNDLED AND SINGLE

The attribute mechanism of GKS has probably been discussed most. It is the treatment of attributes that, more than anything else, decides the extent to which GKS is device independent. The final functionality was only agreed upon at the most recent meeting. In [6] an overview is given of the evolution of the attribute bundling mechanism.

4.1 Geometric attributes

The appearance of an output primitive is determined by a number of factors:

- Its basic geometrical shape which is defined as part of the primitive itself (e.g., the vertices of a POLYLINE as given in world coordinates)
- Shape aspects which give more detailed shape information relative to the basic shape. These aspects are subjected to the same transformations as the basic shape, and are controlled by so-called geometrical attributes.
- Shape aspects which also refine the shape definition but whose values are not transformed. They are also referred to as geometric attributes. The only examples are CHARACTER PATH and TEXT ALIGNMENT. As far as GKS is concerned they could go in the bundle for text, certainly in the new version which allows all bundled attributes to be used in an unbundled fashion as well. Currently however, they are strictly global.
- Non-geometric attributes whose values are sent to each workstation without being transformed. These attributes can all be bundled.

Table 2 ATTRIBUTE TABLE				
primitive	geom1	geom2	bundle	pick
POLYLINE			LINE TYPE LINE WIDTH COLOUR	PICK
POLYMARKER			MARKER TYPE MARKER SCALE COLOUR	PICK
TEXT	CHAR UP CHAR HEIGHT	PATH ALIGN	FONT & PREC EXP. FACT COLOUR	PICK
FILL AREA	PAT. SIZE PAT. REF. PT.		INT. STYLE COLOUR PAT/HATCH	PICK
CELL ARRAY				PICK
GDP				PICK

- The PICK attribute which does not affect the appearance but is used by the input facilities as a name for the primitive.

The easiest way to describe the effect of attributes on primitives is temporarily to ignore transformations, or equivalently, to assume unit normalisation, workstation and segment transformations.

The second step is to collect all relevant individual attributes of the primitive by taking their global values and their bundled values according to the Attribute Selection Flags (ASF, see 4.2 below).

In the next step the attributes are applied in a fixed order, which is partially defined by GKS and can partly be determined by the implementor. The latter cases are those where the order has no influence on the resulting appearance of the primitive.

Finally the primitive with its shape and attributes is subjected to the three transformations and appears on the display surface.

In practice, however, this is not the way implementations have to do it. It is perfectly permissible to store transformed (normalised) primitives and geometric attributes in a segment. In that case the remaining workstation dependent attributes can be adjusted by fitting them to the already transformed geometrical attributes. The latter thus serve as some kind of carrier of the transformation information.

For TEXT the CHARACTER HEIGHT and CHARACTER UP VECTOR serve together as the carrier. For FILL AREA the PATTERN SIZE and REFERENCE POINT have the same function.

In table 2 the attributes are ordered per primitive in the four groups mentioned above.

In table 3 the attributes are partially ordered to indicate per primitive in what sequence they have to be applied. Also it is indicated where the transformation and clip must be performed.

So far, it has been assumed that all attributes are bound to the primitive at the workstation. In the next section we examine attribute binding.

4.2 Attribute binding

GKS supports two ways of binding attributes to primitives. The first is **direct, static** binding. This means that an attribute is bound to the primitive when it is created, and that this attribute value remains with the primitive as long as it exists.

The second way is **indirect, dynamic** binding. This means that primitives are selected from a bundle, via a directly bound index. This selection takes place only at the workstation and is therefore often called delayed binding. The dynamic property is due to the fact that the contents of the bundle may be redefined at the workstation, implying that the primitive will then appear with the new attributes.

Direct static binding is current practice, indirect binding results from the attempts to achieve strong device independence. The dynamic aspect is a very useful feature, which is however, difficult to suppress if not wanted.

In GKS the dynamic effects are required to take place even at the expense of a complete regeneration of the picture from segment store.

Problems can arise if an application is forced to use so many bundles that it has to reuse existing indices, and in doing so must redefine their contents. This is especially annoying if the workstation has only a limited capacity to store bundles. Note that bundles can be redefined but not thrown away.

Many current applications would be forced to do frequent regeneration if running under GKS. This is why the original design has been relaxed to allow bundleable, dynamic attributes to be used in an unbundled, static way as well.

The mechanism which makes this possible is called the ATTRIBUTE SOURCE FLAGs, one for each bundle attribute. Together these flags form a mask. For each of the attributes a global value is defined to be used when direct static binding is selected.

Table 3 — GKS Pipeline

POLYLINE

transform
linewidth
linetype
clip
colour
pick

POLYMARKER

transform
markertype
markerscale
clip
colour
pick

TEXT

font & prec
char up
char height
exp. fact
path
alignment
transform
clip
colour
pick

FILLAREA

int. style
pat. size
pat. refpt
transform
pattern
hatch
colour
clip
pick

CELL ARRAY

transform
clip
pick

GDP

.....
transform
.....
clip
.....
pick

For each primitive created a direct binding takes place for each appropriate attribute whose flag in the mask set to INDIVIDUAL. All the other attributes will be bundled.

This facility is certainly very useful for those applications which would otherwise use an unreasonable number of bundles. However, an application using it sacrifices device independence. It is therefore important that workstation implementors design attribute bundle schemes which provide sufficient bundles.

GKS defines the minimum number of bundles a workstation must provide. These figures show that the introduction of static attributes was not intended as a *substitute* for attribute bundles. This compromise will allow workstation designers gradually to develop new firmware which accomodates bundling.

On the other hand it will cause difficulties for workstation designers who had already built hardware that accommodates dynamic binding only.

5. SEGMENTS

Since the Abingdon meeting of October 1981, the function of the device independent segment storage has become much clearer, and also more useful.

The INSERT SEGMENT function has been given a narrower context. In addition two new functions have been added, each of which addresses a special capability of the Workstation Independent Segment Store (WISS). All these functions specify some kind of transfer of information between workstations.

The reuse of pictures stored on WISS in segments can be for one of the following purposes:

1. COPY SEGMENT TO WORKSTATION

   Copy a picture onto a display surface in a way similar to creating pictures outside segments. The segment administration is omitted. Only primitives and their attributes as stored in the segment are copied. This function therefore only copies the information required for making the picture visible. The segment attributes transformations are ignored.

2. ASSOCIATE SEGMENT TO WORKSTATION

   Copy a picture with its segment structure to the workstation. A segment as a closed, self contained entity cannot affect the state of the workstation with respect to segments already present or subsequent output. This function adds pictures to a workstation for subsequent manipulation.

3. INSERT SEGMENT

   Copy the primitives with their attributes into the open segment. All transformations are applied as originally defined. The clipping rectangle is, however, replaced by the current one. The primitive attributes of the inserted primitives do not change the current global primitive attribute values. This function treats the stored segments as predefined symbols.

It is interesting to see what happens to these functions in case the workstation the segments are sent to is a Metafile Output Workstation.

Case one clearly is the plotfile function.

Case 2 is capable of exchanging pictures between workstation independent segment stores. This shows that workstation independent segment storage together with a metafile permits pictures to be exchanged between any two GKS workstations for all purposes.

Case 3 is of no special interest with respect to external exchange. Note that INSERT SEGMENT has no workstation parameter.

One may compare case 2 with the case of a metafile audit trail function, i.e., the metafile was open during the whole session. In both case in principle the same picture, including structure can be stored. Reading the audit trail metafile will, however, affect the global state of the system and may therefore influence subsequent output. Metafiles used to exchange pictures should, therefore, be written using ASSOCIATE SEGMENT.

6. ACKNOWLEDGEMENT

My thanks to David Rosenthal for a critical review of this paper.

7. REFERENCES

1. P.J.W. ten Hagen, "Graphics Standards _ where are we?", EUROGRAPHICS 81, North-Holland, Amsterdam 1981.
2. P.J.W. ten Hagen, "The GKS reviewing process", in J.L. Encarnaçao and W. Strasser (eds.), Drittes Darmstädter Kolloquium, Oldenbourg München 1981.
3. P.R. Bono, J.L. Encarnaçao, F.R.A. Hopgood and P.J.W. ten Hagen, "GKS _ The first graphics standard", IEEE Computer Graphics and Applications, Vol. 2, No. 5, 1982.
4. GKS version 7.0, ISO/DP 7942, January 1982.
5. D.S.H. Rosenthal, J.C. Michener, G. Pfaff, L.R.A. Kessener and M. Sabin, "The detailed semantics of graphics input devices", Proc. SIGGRAPH 82, Computer Graphics Vol. 16, No. 3, 1982.
6. D.C. Sutcliffe, "Attribute handling in GKS", Proc. EUROGRAPHICS 82, North-Holland, Amsterdam, 1982.